

## ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, ТЕРМІНІВ І ПОЗНАЧЕНЬ .....	5
ВСТУП.....	7
1 МОВА JAVA 2 ТА ЇЇ СЕРЕДОВИЩЕ РОЗРОБКИ.....	8
2 ОСНОВНІ СТАНДАРТНІ ЗАСОБИ СЕРЕДОВИЩА РОЗРОБКИ JAVA ФІРМИ ORACLE (SUN MICROSYSTEMS) .....	12
3 ЗАГАЛЬНІ ПРИНЦИПИ ПОБУДОВИ ПЛАТФОРМИ .....	59
3.1 Знайомство з Eclipse Foundation.....	60
3.2 Проекти співтовариства і архітектура платформи .....	62
3.2.1 Завдання й проекти Eclipse Foundation.....	62
3.2.2 Архітектура платформи .....	64
4 ІНТЕГРОВАНЕ СЕРЕДОВИЩЕ РОЗРОБКИ .....	68
4.1 Установлення середовища розробки .....	68
4.2 Перший запуск, перше знайомство з основними поняттями й системами платформи .....	69
4.2.1 Поняття робочого простору.....	70
4.2.2 Вікно вітання й знайомство з підсистемою допомоги.....	72
4.3 Поняття робочого місця й інструментальні засоби користувальницького інтерфейсу .....	76
4.3.1 Призначення й властивості бібліотеки для розробки графічних інтерфейсів користувача.....	77
4.3.2 Структура й призначення набору Java-класів для побудови графічного інтерфейсу користувача .....	79
4.3.3 Робоче місце: інструменти, компонування, редактори й представлення .....	80
4.3.4 Інтеграція інструментів, що підключаються, з інтерфейсом користувача .....	84
5 РОБОТА В ІНТЕГРОВАНОМУ СЕРЕДОВИЩІ РОЗРОБКИ .....	86
5.1 Інструмент Java Development Tool, як основа розробки коду Java.....	86
5.2 Створення файлу вихідного коду.....	89
5.3 Компіляція програми в середовищі .....	100
5.4 Редагування вихідного Java коду в редакторі.....	102

5.4.1	Відображення помилок у редакторі й інших представленнях .....	102
5.4.2	Виправлення помилок Java-коду в редакторі .....	107
	ДОДАТОК А Ключі командного рядку системи.....	113
	ДОДАТОК Б Стислий опис «гарячих клавішів», що налаштовані у платформі.....	115
	ДОДАТОК В Іконки та кнопки панелей інструментів середовища .....	120
	ПЕРЕЛІК ПОСИЛАНЬ .....	124

## ПЕРЕЛІК СКОРОЧЕНЬ, ТЕРМІНІВ І ПОЗНАЧЕНЬ

- Ant – Another Neat Tool, java-утиліта для автоматизації процесу зборки програмного продукту.
- API – Application Programming Interface, інтерфейс прикладного програмування (іноді інтерфейс програмування додатків), набір готових класів, процедур, функцій, структур і констант, що надаються для використання в зовнішніх програмних продуктах.
- AWT – Abstract Window Toolkit, стандартний API для реалізації графічного інтерфейсу в Java-програми.
- CRC картки – Class, Responsibilities, Collaboration, Клас, Обов'язки, Взаємодія, картки для командної розробки систем.
- EJB – Enterprise JavaBeans, специфікація технології написання й підтримки серверних компонентів, що містять бізнес-логіку.
- GUI – Graphical User Interface, графічний інтерфейс користувача.
- IDE – Integrated Development Environment, інтегроване середовище розробки
- IDL – Interface Description Language або Interface Definition Language, мова опису інтерфейсів. Слугує для опису інтерфейсів і забезпечення зв'язку програмних компонентів, створених на різних мовах програмування, наприклад, компонентів на C++ і компонентів на Java.
- JAAS – Java Authentication and Authorization Service, засоби для автентифікації й авторизації користувачів.
- JDK – Java Development Kit, безкоштовно розповсюджуваний Oracle Corporation (раніше Sun Microsystems) комплект розроблювача додатків мовою Java.
- JDT – Java Development Tools, інструменти розробки Java.
- JFace – Додатковий програмний шар над SWT (див.), набір Java-класів, що реалізує найбільш загальні завдання побудови GUI (див.).
- JRE – Java Runtime Environment, середовище виконання Java.
- JSP – Java Server Pages, технологія, що дозволяє створювати код, що містить як статичні, так і динамічні компоненти.

MVC	– Model-View-Controller, модель-уявлення-поведінка, модель-уявлення-контролер.
PDE	– Plug-in Development Environment, середовище розробки плагинів.
PKCS#7, #10	– Public-Key Cryptography Standards №7 і №10, стандарти сімейства стандартів, розроблених і опублікованих RSA Data Security Inc. Використовуються для підписання, шифрування повідомлень і для розсилань сертифікатів, а, також, визначають формат запиту про автентичність відкритого ключа відповідно.
PMC	– Project Management Committee, комітет з управління проектом.
PRE	– Platform Runtime Environment, середовище виконання платформи Eclipse.
RMI	– Remote Method Invocation, програмний інтерфейс виклику віддалених методів у мові Java.
SDK	– Software Development Kit, комплект засобів розробки, що дозволяє створювати програмне забезпечення
SHA	– Secure Hash Algorithm, алгоритм криптографічного хешування для вхідного повідомлення довільної довжини до 2-х ексібایتів (див.).
SWT	– Standard Widget Toolkit, бібліотека з відкритим вихідним кодом для розробки графічних інтерфейсів користувача мовою Java.
Tomcat	– Програма-контейнер сервлетів і така, що реалізує специфікацію сервлетів і специфікацію JSP (див.).
UI	– User Interface, інтерфейс користувача.
URL	– Universal Resource Locator, адреса сторінки в Internet.
VM	– Virtual Machine, віртуальна машина (Java).
X.509	– Стандарт обумовлює формати даних і процедури розподілу загальних ключів за допомогою сертифікатів із цифровими підписами, які надаються сертифікаційними органами.
Ексіббайт	– $2^{60}$ байтів.

## ВСТУП

Методичні вказівки призначені для студентів другого курсу факультету комп'ютерних наук, котрі вивчають курс об'єктно-орієнтованого програмування і виконують відповідні лабораторні роботи.

Виконання лабораторних робіт мовою Java за допомогою SDK – середовища розробки програм, що поставляється фірмою Oracle – розробником мови, неможливе без знання і володіння різними програмними інструментальними засобами.

Підручники мови Java містять лише відомості безпосередньо лише з семантики, синтаксису мови та бібліотеки класів Java, що містить універсальні структури даних і алгоритми, здатні задовольнити потреби більшості стандартних додатків, і зовсім не торкаються питань інструментальних засобів середовища розробки. Але ці питання досить складні і, відповідно, вимагають вивчення за допомогою додаткової літератури. Складність при цьому полягає у практично повній відсутності таких посібників українською мовою, а також у малій кількості видань навіть російською. Переважна більшість літератури з цих питань є англійською (див. перелік посилань).

Тому метою даних методичних вказівок є надання студентам досить повної інформації з різноманітних засобів розробки програм мовою Java. Ця інформація повинна допомогти студентам при створенні не тільки лабораторних робіт з об'єктно-орієнтованого програмування, але і при виконанні досить складних розробок у курсових та дипломних проектах.

Завданням методичних вказівок є навчити студентів вибирати зі всіх можливих засобів розробки додатків ті, котрі є найбільш підходящими і ефективними у кожному конкретному випадку їх застосування.

Оскільки методичні вказівки призначені в основному для самостійної роботи студентів дома перед виконанням лабораторних робіт, тому до них не включено розділ присвячений техніці безпеки.

## 1 МОВА JAVA 2 ТА ЇЇ СЕРЕДОВИЩЕ РОЗРОБКИ

У цей час найбільше поширення в програмуванні знайшли об'єктно-орієнтовані мови такі, як C++ і, особливо, Java. Основною перевагою об'єктно-орієнтованих мов програмування, є високий ступінь повторного використання коду в добре спроектованих системах. Це означає, що для розробки кожного наступного додатка потрібно набагато менше нового коду; отже, меншу кількість коду потрібно супроводжувати й підтримувати.

Повторне використання може приймати різні форми. Це й запозичення окремих рядків коду, і використання окремих класів або логічно зв'язаних між собою груп класів. Найбільш простим і таким, що не потребує великих навичок способів є, звичайно ж, повтор рядків. Практично всі програмісти хоч раз, але використовували в редакторі тексту сполучення клавіш <Ctrl-C> і <Ctrl-V> для копіювання й вставки реалізації того або іншого алгоритму з однієї програми до іншої. Але цей спосіб є й найменш вигідним, оскільки той самий фрагмент коду просто дублюється в різних додатках. Набагато ефективніше, використовуючи об'єктно-орієнтовані мови програмування, звертатися до існуючих класів, модифікуючи їх або успадковуючи від них. Але ще більших успіхів можна досягти, використовуючи набори класів, організовані в інструментальні бібліотеки, – середовища розробки. Під *середовищем розробки* (SDK – Software Development Kit) за звичай розуміють сукупність класів, що надають набір послуг у певній області. Таким чином, середовище розробки експортує ряд *окремих класів* і *механізмів*, які можуть використовуватися безпосередньо або адаптуватися до завдань користувача.

Середовища розробки можуть бути досить універсальні й бути застосовні до широкого кола додатків. До цієї категорії належать загальні фундаментальні класи, математичні бібліотеки й бібліотеки графічного інтерфейсу користувача. Середовища розробки можуть зустрічатися й у досить вузьких предметних областях. Там, де існує сімейство програм, котрі вирішують подібні завдання, з'являється привід створити й використовувати прикладне середовище розробки.

Із самого початку розроблювачі мови Java поставили перед собою завдання забезпечити для свого середовища розробки досить значне число окремих класів і пакетів класів. На рис. 1.1 з [1] приведений графік росту числа класів у середовищі розробки Java у міру переходу від однієї версії середовища до іншої.

## A very brief history of Java

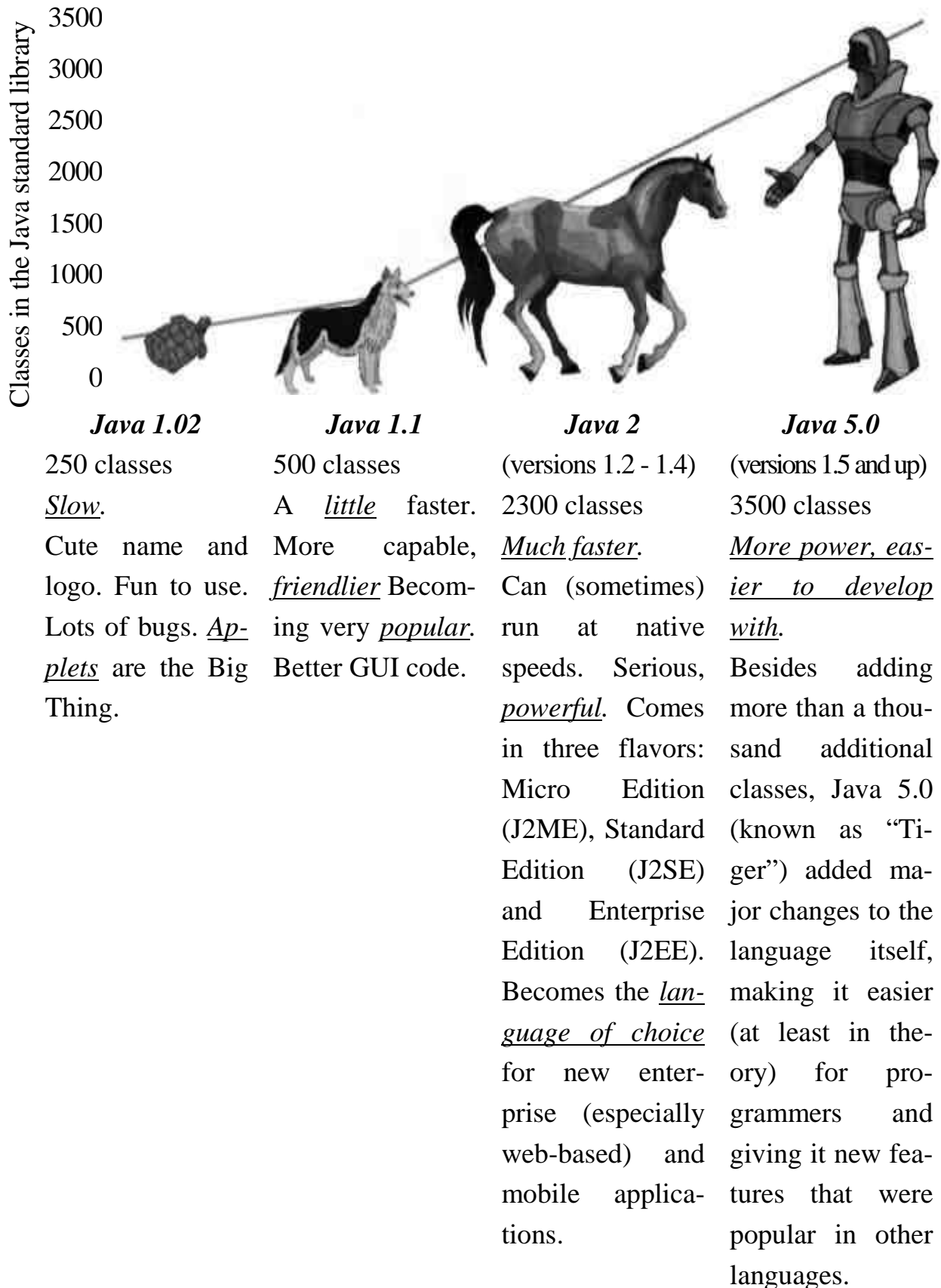


Рисунок 1.1 – Образне подання росту числа класів бібліотеки Java

Бібліотека класів Java містить універсальні структури даних і алгоритми, здатні задовольнити потреби більшості стандартних додатків. Крім того, бібліотека класів задовольняє всім вимогам, котрі пред'являються до подібних бібліотек, тобто вона є:

- повною містить сімейства класів, об'єднаних узгодженим зовнішнім інтерфейсом, але з різними уявленнями, так щоб розроблювачі могли вибрати те, семантика якого найбільш точно відповідає додаткові;
- адаптуємою всі фрагменти коду, що залежать від платформи, виділені й ізольовані в окремі класи для забезпечення можливості локальних змін у них, розроблювачі мають контроль над механізмами зберігання даних і синхронізації процесів;
- ефективною підключення різних фрагментів бібліотеки до додатка є простим (ефективність при компіляції), непродуктивні витрати оперативної пам'яті й процесорного часу на обслуговування й підключення зведені до мінімуму (ефективність при виконанні), бібліотека забезпечує більш надійну роботу, чим механізми, розроблені користувачем вручну (ефективність при розробці);
- безпечною всі абстракції безпечні з погляду типів, таким чином, що припущення про поведінку класу завжди забезпечуються компілятором, виявлення помилок динамічної семантики класів здійснюється механізмом виключень, порушення виключення не змінює стан об'єкта, що викликав виключення;
- простою бібліотека має прозору структуру, що дає можливість легко знаходити й підключати до додатка її фрагменти;
- розширюваною для користувачів реалізована можливість включення в бібліотеку нових класів, при цьому архітектурна цілісність середовища розробки не порушується.

Як механізми розробки, у поставку мови Java фірма Oracle включає дуже велику кількість інструментів для програмістів. На рис. 1.2 наводиться вміст каталогу із програмним забезпеченням, що входить у поставку пакета Java 2 SDK або як його ще називають JDK (Java Development Kit, інструменти розробки Java). Цілком очевидно, що основні інструменти – це інтерпретатор (власне машина Java), компілятор і налагоджувач Java, але є й інші.



Том в устройствe D имеет метку ProgramDisk

Серийный номер тома: 00D2-9336

Содержимое папки D:\Program Files\Java\jdk\bin\

[.]	javadoc.exe	jstack.exe	rmic.exe
[..]	javah.exe	jstat.exe	rmid.exe
javaws.exe	javap.exe	jstatd.exe	rmiregistry.exe
javaw.exe	extcheck.exe	keytool.exe	schemagen.exe
java.exe	apt.exe	kinit.exe	serialver.exe
unpack200.exe	jarsigner.exe	klist.exe	servertool.exe
packager.exe	jdb.exe	ktab.exe	idlj.exe
HtmlConverter.exe	jhat.exe	native2ascii.exe	jps.exe
jconsole.exe	jar.exe	orbd.exe	wsgen.exe
tnameserv.exe	jmap.exe	pack200.exe	xjc.exe
wsimport.exe	appletviewer.exe	jinfo.exe	java-rmi.exe
javac.exe	jrunscript.exe	policytool.exe	
	45 файлов	1650176 байт	
	2 папок	257089536 байт	свободно

Рисунок 1.2 – Вміст каталогу із програмами, що поставляються фірмою Oracle (команда системи `dir /d /-c /og-s`)

Ці інструменти входять у пакет розробки від Oracle, і їх варто розглядати як особливості реалізації, оскільки вони не описані в специфікації Java. Більшість додатків пакета (включаючи компілятор, наладжувач, програму документування й т.і.) є програмами з інтерфейсом командного рядка. Роботу в командному рядку можна до певної міри вважати протилежністю *гнучкої розробки* (agile development). Командний рядок, хоча й є трохи складним у вивченні, але дозволяє досить ефективно й гнучко управляти процесом розробки програм. І, незважаючи на складність, безліч професійних розроблювачів віддають перевагу саме командному рядку, причому працюють у ньому досить ефективно. Тому для ознайомлення із цим підходом і з'ясування фундаментальних принципів, або якщо так можна сказати, вивчення того, що відбувається за лаштунками, у розділі 2 приведений опис практично всіх основних програм пакета. Пропущені тільки розгляд засобів роботи з RMI (Remote Method Invocation) і IDL (Interface Description/Definition Language), котрі призначені для корпоративного програмування. Їхній опис можна знайти, наприклад, у книгах [2, 3].

## 2 ОСНОВНІ СТАНДАРТНІ ЗАСОБИ СЕРЕДОВИЩА РОЗРОБКИ JAVA ФІРМИ ORACLE (SUN MICROSYSTEMS)

### Команда Java

Реалізація: JDK 1.0 і вище

Призначення: інтерпретатор мови Java

Синтаксис командного рядка:

```
java [ключі_інтерпретатора] ім'я_класу [параметри_програми]
```

```
java [ключі_інтерпретатора] -jar jar-файл [параметри_програми]
```

### Опис

*java* – це інтерпретатор байта-коду. Він запускає програми, що написані мовою Java.

*ім'я\_класу* – ім'я класу програми, яку потрібно запустити. Воно повинне бути повним, тобто містити ще й ім'я пакета класу, але без розширення *.class*. Наприклад:

```
java david.games.Checkers
```

```
java Test
```

У класі обов'язково повинен бути визначений метод *main()*, що має наступну сигнатуру:

```
public static void main(String[] args)
```

Цей метод є точкою входу в програму, і інтерпретатор починає виконання байт-коду саме з нього.

В Java 1.2 і наступних версіях програму можна впакувати у виконуваний архівний файл *.jar*. Щоб запустити таку програму, використовується ключ *-jar* для зазначення файлу JAR. Маніфест файлу *.jar*, що виконується, повинен містити атрибут *Main-Class*, що визначає, у якому з файлів усередині архіву перебуває клас із методом *main()*.

Будь-які параметри командного рядка, що передують імені файлу класу або файлу JAR, – це ключі інтерпретатора Java. Всі параметри, які йдуть після імені класу або архіву JAR, являють собою параметри самої програми, вони ігноруються інтерпретатором Java і передаються у вигляді масиву рядків методу *main()*.

Інтерпретатор Java працює до завершення методу *main()*. Всі потоки програми (крім потоків, позначених як *daemon*) теж завершуються.

## Версії інтерпретатора

Залежно від використовуваної версії SDK Java можлива наявність у цих пакетах інших версій інтерпретатора Java. Кожна з них схожа на *java*, але має спеціальні можливості. Існують наступні різновиди інтерпретатора:

### *java*

Це базова версія інтерпретатора Java; звичайно потрібно використовувати саме її. В Java 1.2 набір підтримуваних ключів істотно змінився в порівнянні з Java 1.1, змінилася й дія ключів. В інших випусках були проведені незначні зміни.

### *oldjava*

Ця версія інтерпретатора включена в пакет Java 1.2 і Java 1.3 для сумісності з Java 1.1. Вона завантажує класи за схемою, що застосовувана в Java 1.1. Програм, що використовують цю версію інтерпретатора, дуже мало, тому вона не увійшла до складу більш пізніх пакетів, починаючи з Java 1.4.

### *javaw*

Ця версія інтерпретатора включена тільки в платформи для Windows. *javaw* використовується в тому випадку, коли потрібно виключити появу вікна консолі при запуску програми Java (наприклад, зі сценарію). В Java 1.2 і Java 1.3 включена програма *oldjavaw*, що сполучає в собі можливості *oldjava* і *javaw*.

### *java\_g*

В Java 1.0 і Java 1.1 програма *java\_g* – це отладочная версія інтерпретатора Java. У ній доступно кілька спеціальних ключів командного рядка, але ця версія застосовується рідко. У платформах для Windows є програма *javaw\_g*. Інструмент *java\_g* не входить в Java 1.2 і наступні версії.

### *Серверна й клієнтська VM*

Віртуальна машина «HotSpot» фірми Oracle має два різновиди: перший різновид для роботи з клієнтськими додатками, що мають малий час життя, а другий призначений для серверних програм, що мають великий час життя. В Java 1.4 і вище за допомогою ключа *-server* можна запустити серверну віртуальну машину. Клієнтську машину можна вибрати за допомогою ключа *-client*, що є ключем за замовчуванням.

### *Класична VM*

Починаючи з Java 1.3 можна задати ключ *-classic*, щоб вибрати «класичну VM» (по суті, це та ж саме, що й віртуальна машина Java 1.2).

### *Динамічний компілятор*

В Java 1.2 і Java 1.3 при зазначенні ключа *-classic* інтерпретатор Java використовує динамічний компілятор (just-in-time compiler, JIT), якщо він доступний на використовуваній платформі. JIT перетворює байт-код Java у машинні інструкції на етапі виконання. Це значно збільшує швидкість роботи більшості Java-програм. Динамічний компілятор JIT можна відключити, привласнивши змінній середовища JAVA\_COMPILER значення «NONE» або встановивши для системної властивості *java\_compiler* це ж значення за допомогою ключа *-D*:

```
set JAVA_COMPILER=NONE // установка змінної оточення  
java -Djava_compiler=NONE MyProgram
```

Якщо необхідно використовувати іншу реалізацію компілятора JIT, то змінній середовища або системній властивості привласнюється ім'я необхідного компілятора. Ця змінна не використовується в Java 1.4 з віртуальною машиною HotSpot, що застосовує більш ефективну технологію JIT.

### *Потокові системи*

В ОС Solaris і інших Unix платформах можна вибрати тип потоків для інтерпретатора Java 1.2 або класичної ВМ Java 1.3. Щоб використати тип потоків, застосовуваний даної ОС, у командному рядку задається ключ *-native*. Для використання віртуальних, або «зелених», потоків (за замовчуванням) вказується ключ *-green*. В Java 1.3 клієнтська машина за замовчуванням застосовує рідні потоки ОС. При зазначенні ключів *-green* або *-native* в Java 1.3 автоматично додається ключ *-classic*. В Java 1.4 ці ключі більше не підтримуються через непотрібність.

### **Ключі командного рядка**

**-classic**

В Java 1.3 цей ключ запускає «класичну ВМ» замість високопродуктивної клієнтської машини, використовуваної за замовчуванням.

**-classpath *шлях***

Визначає каталоги, файли JAR або ZIP, які переглядає програма *java* при спробі завантажити клас. В Java 1.0 і Java 1.1, а також при використанні *oldjava*, цей ключ визначає розташування системних класів, класів розширень і класів додатків. В Java 1.2 і наступних версіях цей ключ визначає тільки розташування класів додатків.

**-client**

Оптимізувати інкрементну компіляцію VM HotSpot для типових клієнтських додатків. Доступний в Java 1.4 і наступних версіях (див. також ключ *-server*).

**-cp**

Синонім *-classpath*. Доступний в Java 1.2 і наступних версіях.

**-cs, -checksource**

Із цими ключами *java* перевіряє час зміни зазначеного файлу класу й відповідного файлу з вихідним кодом. Якщо файл класу не знайдений або застарів, вихідний код автоматично перекомпілюється. Ключ, доступний тільки в Java 1.0 і Java 1.1, а в Java 1.2 і наступних версіях він не підтримується.

**-Димя\_властивості=значення**

Привласнити *значення* зазначеній системній *властивості*. Інтерпретатор *java* може здійснювати пошук цього значення за ім'ям властивості. Можна вказати будь-яку кількість ключів *-D*, наприклад:

```
java -Dawt.button.color=gray -Dmy.class.pointsize=14 my.class  
-d32
```

Запуск в 32-розрядному режимі. Ключ є допустимим в Java 1.4 і наступних версіях, але в цей час реалізований тільки для ОС Solaris.

**-d64**

Запуск в 64-розрядному режимі. Ключ є допустимим в Java 1.4 і наступних версіях, але в цей час реалізований тільки для ОС Solaris.

**-da[:where]**

Відмінити оператори контролю (assertions). Доступний в Java 1.4 і наступних версіях (див. також *-disableassertions*).

**-debug**

Запустити програму *java* таким чином, щоб наладжувач *jdb* міг підключитися до сесії інтерпретатора. В Java 1.2 і наступних версіях цей ключ замінений ключем *-Xdebug*.

**-disableassertions[:where]**

Відмінити оператори контролю (assertions). Ключ уперше з'явився в Java 1.4. Допускається скорочений варіант *-da*. Якщо указан тільки сам ключ (за замовчуванням), то відмінюються всі оператори контролю (крім тих, які перебувають у системних класах). Щоб відмінити оператори контролю тільки з одного класу, потрібно відразу за ключем поставити двокрапку й вказати повне ім'я класу. Щоб відмінити оператори контролю у всіх пакеті (і всіх па-

кетах, що входять до нього), після ключа потрібно поставити двокрапку, ім'я пакета й три крапки. (див. також *-enableassertions* і *-disablesystemassertions*).

#### **-disablesystemassertions**

Відмінити оператори контролю у всіх системних класах (за замовчуванням). Ключ уперше з'явився в Java 1,4. Допускається скорочений запис *-dsa*. Цей ключ не приймає ніяких параметрів.

#### **-enableassertions[:where]**

Використовувати оператори контролю. Ключ уперше з'явився в Java 1.4. Допускається скорочена форма *-ea[:where]*. Якщо ключ зазначений без параметрів, будуть використовуватися всі оператори контролю (крім тих, які перебувають у системних класах). Щоб дозволити оператори контролю тільки з одного класу, потрібно поставити після ключа двокрапку й повне ім'я класу. Щоб використовувати оператори контролю із усього пакета (і всіх пакетів, що входять до нього) після ключа ставиться двокрапка, ім'я пакета й три крапки (див. також *-disableassertions* і *-enablesystemassertions*).

#### **-enablesystemassertions**

Використовувати оператори контролю з усіх системних класів. Допускається скорочений варіант *-esa*. Ключ доступний в Java 1.4 і наступних версіях.

#### **-green**

В операційних системах, що підтримують кілька типів потоків (наприклад, Linux і Solaris), цим ключем вибирається віртуальний, або «зелений», тип потоків. Даний ключ застосовується за замовчуванням в Java 1.2. В Java 1.3 при виборі цього ключа автоматично вибирається ключ *-classic*. (див. також *-native*). Ключ доступний тільки в Java 1.2 і Java 1.3.

#### **-help** або **-?**

Виводиться довідкова інформація, і програма завершується (див. також *-X*).

#### **-jar jar-файл**

Запустити *jar-файл*, що може виконуватися. Маніфест цього файлу повинен містити атрибут *Main-Class*. Цей атрибут вказує, у якому класі перебуває *main()* - метод, з якого починається виконання програми. Ключ доступний в Java 1.2 і наступних версіях.

#### **-native**

В операційних системах, що підтримують різні типи потоків (наприклад, Solaris і Linux), цим ключем вибирається тип потоків, що використовується у даній ОС. За замовчуванням застосовується інший, «зелений» тип потоків. У деяких випадках, наприклад при запуску програми на багатопроцесорному комп'ютері, краще використовувати основний тип потоків. В Java 1.3 віртуальна машина HotSpot застосовує основні потоки. Ключ доступний тільки в Java 1.2 і 1.3.

**-showversion**

Цей ключ виконує ті ж дії, що й *-version*. Відмінність полягає в тому, що із цим ключем інтерпретатор продовжує роботу після відображення інформації про версію. Доступний в Java 1.3 і наступних версіях.

**-verbose, -verbose:class**

Виводити повідомлення при кожному завантаженні класу. В Java 1.2 і наступних версіях ключ *-verbose:class* можна використовувати як синонім.

**-verbose:gc**

Виводити повідомлення при кожному збиранні сміття. Доступний в Java 1.2 і наступних версіях. У версіях, що передують 1.2, використовується ключ *-verbosegc*.

**-verbose:jni**

Виводити повідомлення при кожному виклику методів Java, котрі залежать від платформи (native методів). Ключ є доступним в Java 1.2 і наступних версіях.

**-version**

Програма *java* виводить версію інтерпретатора й завершує роботу.

**-X**

Виводиться довідкова інформація для нестандартних ключів командного рядка інтерпретатора (тих, які починаються з *-X*), і програма завершує роботу (див. також *-help*). Ключ доступний в Java 1.2 і наступних версіях.

**-Xbatch**

Приписання віртуальній машині HotSpot виконати динамічну компіляцію (just-in-time compilation) з високим пріоритетом, незалежно від часу, необхідного для компіляції. Без цього ключа віртуальна машина виконує низькопріоритетну компіляцію методів під час їхньої інтерпретації з високим пріоритетом. Доступний в Java 1.3 і наступних версіях.

**-Xbootclasspath:шлях**

Визначає шлях пошуку системних класів, що складається з імен каталогів, ZIP і JAR архівіруваних файлів. Елементи списку шляхів розділяються символом «;» (крапка з комою). Цей ключ використовується дуже рідко. Доступний в Java 1.2 і наступних версіях.

**-Xbootclasspath/a:шлях**

Додає зазначений *шлях* у кінець списку шляхів до системних класів. Роздільником списку шляхів служить «;». Доступний в Java 1.3 і наступних версіях.

**-Xbootclasspath/p:шлях**

Додає зазначений *шлях* у початок списку шляхів до системних класів. Роздільником списку шляхів служить «;». Доступний в Java 1.3 і наступних версіях.

**-Xcheck:jni**

Виконується додаткова перевірка при кожному використанні функцій з Java Native Interface. Ключ доступний в Java 1.2 і наступних версіях.

**-Xdebug**

Запустити інтерпретатор так, щоб до нього міг підключитися наладжувач. Ключ доступний в Java 1.2 і наступних версіях. У версіях, що передують 1.2, необхідно використовувати ключ *-debug*.

**-Xfuture**

Виконати строго перевірку формату всіх завантажених файлів класів. Без цього ключа інтерпретатор *java* робить таку ж перевірку, як в Java 1.1. Ключ доступний в Java 1.2 і наступних версіях.

**-Xincgc**

Використовувати інкрементне збирання сміття. У цьому режимі збирач сміття постійно працює у фоновому режимі, а виконувана програма дуже рідко припиняється для збирання сміття. Проте, використання цього ключа знижує загальну продуктивність приблизно на 10%. Ключ доступний в Java 1.3 і наступних версіях.

**-Xint**

Із цим ключем VM HotSpot працює тільки в режимі інтерпретатора, без виконання динамічної компіляції. Доступний в Java 1.3 і наступних версіях.

**-Xloggc:ім'я\_файлу**

Приписання VM вести, у зазначеному файлі, журнал реєстрації подій, зв'язаних зі збиранням сміття.



### **-Xmixed**

Із цим ключем VM HotSpot виконує динамічну компіляцію найбільш часто використовуваних («гарячих») методів, а інші методи виконує в режимі інтерпретатора. Такий режим роботи встановлений за замовчуванням (див. також ключі *-Xbatch* і *-Xint*). Ключ доступний в Java 1.3 і наступних версіях.

### **-Xms *обсяг\_купи* [к | м]**

Визначає, скільки пам'яті буде виділено під «купу» при старті інтерпретатора. За замовчуванням обсяг пам'яті (*обсяг\_купи*) вказується в байтах. Можна вказати розмір у кілобайтах або мегабайтах, поставивши *k* або *m* відповідно.

За замовчуванням розмір «купи» дорівнює 1 Мбайт. Продуктивність великих програм або програм, що інтенсивно використовують пам'ять (таких як компілятор Java), можна збільшити, виділивши більше пам'яті при запуску інтерпретатора. Мінімальний початковий розмір «купи» – 1000 байт. Ключ доступний в Java 1.2 і наступних версіях. У версіях, що передують 1.2, застосовується ключ *-ms*.

### **-Xmx *обсяг\_купи\_тах* [к | м]**

Визначає максимальний розмір «купи», що інтерпретатор використовує для динамічно створюваних об'єктів і масивів. За замовчуванням розмір вказується в байтах, але його можна вказати в кілобайтах або мегабайтах, поставивши *k* або *m* відповідно. За замовчуванням максимальний розмір «купи» дорівнює 16 Мбайт; і він не може бути менше 1000 байт. Ключ доступний в Java 1.2 і наступних версіях. У версіях, що передують 1.2, застосовується ключ *-mx*.

### **-Xnoclassgc**

Не збирати сміття класів. Ключ доступний в Java 1.2 і наступних версіях. В Java 1.1 застосовується ключ *-noclassgc*.

### **-Xprof**

Виводить у стандартний вихідний потік інформацію про виконання коду. Ключ доступний в Java 1.3 і наступних версіях. При використанні ключа *-classic*, а також в Java 1.2 застосовується ключ *-Xrunhprof* (див. нижче). У версіях Java, що передують 1.2, задається ключ *-prof*.

### **-Xrs**

Використовувати менше сигналів операційної системи. При цьому на деяких системах може покращитися продуктивність. Ключ доступний в Java 1.2 і наступних версіях.

### **-Xrunhprof:*suboptions***

Включити профілювання процесора, «купи», монітора. *suboptions* – це список елементів виду *ім'я=значення*, розділених комами. Для одержання списку підтримуваних параметрів і значень необхідно використовувати ключ -*Xrunhprof: help*. Ключ підтримується в Java 1.2 і наступних версіях. У версіях, що передують 1.2, профілювання підтримується частково. Воно включається за допомогою ключа *-prof*. В Java 1.3 даний ключ працює тільки в класичному режимі (ключ *-classic*) і не підтримується новою VM HotSpot (див. *-Xprof*).

**-Xss стек[к | м]**

Визначає розмір стеку *java* потоку в байтах, кілобайтах або мегабайтах. Ключ доступний в Java 1.3 і наступних версіях.

Необхідно врахувати, що *-X* ключі не є стандартними і їхню зміну може бути зроблено фірмою Oracle без попередження. Так, наприклад, в Java 1.6 уведений ключ *-Xshare:off | on | auto*, що забороняє (*off*), наказує (*on*) або дозволяє по можливості (*auto*) використання поділюваних даних класу, Тому перед застосуванням цих ключів найкраще скористатися командою:

**java -X**

### **Завантаження класів**

Інтерпретаторові Java відомо, де потрібно шукати системні класи, що входять до складу платформи Java. В Java 1.2 і наступних версіях йому також відомо, де шукати файли класів розширень, що встановлені у каталозі системних розширень. Але інтерпретаторові потрібно вказати, де шукати несистемні класи додатка, що запускається.

Файли класів зберігаються у каталогах з іменами, що відповідають іменам їхніх пакетів. Наприклад, клас *com.davidflanagan.utils.Util* зберігається у файлі *com\davidflanagan\utils\Util.class*. За замовчуванням як кореневий каталог інтерпретатор використовує поточний робочий каталог і шукає в ньому й у його підкаталогах всі класи.

Інтерпретатор може здійснювати пошук класів усередині архівів ZIP і JAR. Для того, щоб повідомити інтерпретатор, у яких пакетах (каталогах), архівованих ZIP і JAR файлах потрібно шукати несистемні класи, можна створити змінну середовища CLASSPATH, аналогічну системної змінної PATH.

Для цього можна скористатися установками в системному віконці ОС Windows, показаному на рис. 2.1

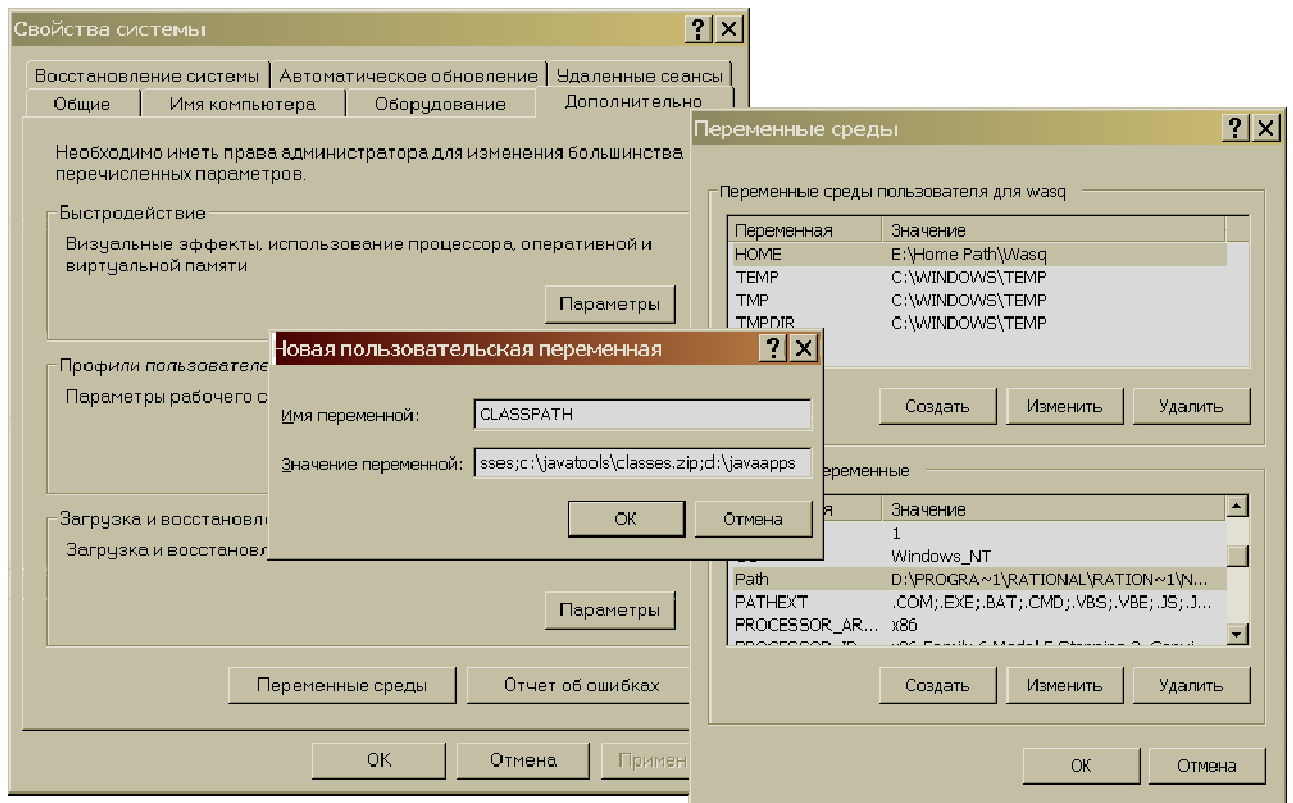


Рисунок 2.1 – Завдання змінної оточення CLASSPATH в ОС Windows XP

Більш простим м універсальним способом визначити шлях до класів – задати змінну середовища в командній оболонці Unix або в командному інтерпретаторі Windows. В Windows для завдання цієї змінної оточення необхідно використовувати команду *set*, наприклад, у такий спосіб:

```
set CLASSPATH=\.;c:\myclasses;c:\javatools\classes.zip;d:\javaapps
```

При пошуку класу інтерпретатор переглядає всі шляхи в тім порядку, у якому вони зазначені в змінній оточення.

Крім цього шлях до класу можна визначити за допомогою ключів командного рядка *-classpath* або *-cp* інтерпретатора Java. Шлях, зазначений в одному із цих ключів, заміщає будь-який шлях, визначений у змінній середовища CLASSPATH. В Java 1.2 і наступних версіях ключ *-classpath* визначає тільки шлях пошуку класів додатка й користувальницьких класів. У версіях Java, що передують 1.2, або при використанні інтерпретатора *oldjava* цей ключ визначає шлях пошуку для всіх класів, включаючи системні класи й класи розширень.

**Див. також**

*javac, jdb*

## Команда *javac*

Реалізація: JDK 1.0 і вище

Призначення: компілятор мови Java

Синтаксис командного рядка:

**javac** [ключі] **файли**

**oldjavac** [ключі] **файли**

### Опис

Програма *javac* - це компілятор Java. Він компілює вихідний код Java, що перебуває у файлах *.java*, у байт-код (файли *.class*). Сам компілятор написаний на Java. В Java 1.3 компілятор повністю переписаний, а його продуктивність істотно підвищилася. Хоча новий *javac* в основному є сумісним з попередніми версіями, старий компілятор також включений у поставку під ім'ям *oldjavac* (тільки в Java 1.3).

Програмі *javac* можна передати будь-яку кількість вихідних файлів Java, імена яких повинні мати розширення *.java*. Програма *javac* створює окремий файл із розширенням *.class* для кожного класу, визначеного у вихідних файлах. У кожному вихідному файлі може втримуватися будь-яка кількість класів, але тільки один із класів верхнього рівня може мати модифікатор *public*. Назва вихідного файлу (без розширення *.java*) повинне відповідати імені класу *public*, що міститься в цьому файлі.

Якщо в Java 1.2 і наступних версіях ім'я файлу, що указане в командному рядку, починається з @, то цей файл інтерпретується не як файл із вихідним кодом, а як список ключів компілятора й вихідних файлів. Таким чином, якщо записати імена вихідних файлів з одного проекту у файл із ім'ям *project.list*, можна скомпілювати відразу всі файли одною командою:

**javac @project.list**

Щоб компілятор *javac* зміг скомпілювати вихідний файл, для нього повинні бути доступні оголошення всіх класів з вихідного файлу. Компілятор шукає оголошення, як у вихідних файлах, так і у файлах класів. При цьому він автоматично компілює вихідні файли, що не мають відповідних файлів класів, а також файли, які були змінені з моменту останньої компіляції.

### Ключі командного рядка

**-bootclasspath** *шлях*

Визначає *шлях*, що використовується компілятором для пошуку системних класів. Цей ключ зручно застосовувати, коли *javac* використовується

ся як крос-компілятор для компіляції класів, написаних під різні версії Java API. Наприклад, компілятор Java 1.3 можна застосовувати для компіляції класів у середовищі Java 1.2. Цей ключ не може визначати системні класи, які використовуються для запуску самого компілятора; можна задати тільки системні класи, які доступні для читання компілятором (див. також *-extdirs* і *-target*). Ключ доступний в Java 1.2 і наступних версіях.

#### **-classpath *шлях***

Визначає *шлях*, що використовується компілятором для пошуку класів, на які посилається вихідний код. Цей ключ заміщає будь-який шлях зі змінного середовища CLASSPATH. Як і для інтерпретатора, *шлях* – це впорядкований список каталогів, архівів ZIP і JAR, розділених двокрапками в Unix і крапками з комами в Windows. Якщо не зазначений ключ *-sourcepath*, цей ключ також установлює шлях пошуку вихідних файлів.

У версіях Java, що передують 1.2, цей ключ визначає шлях до системних класів і класів розширень, а також до класів додатків і користувальницьких класів, тому він вимагає акуратного використання. В Java 1.2 і наступних версіях цей ключ установлює шлях пошуку тільки для класів додатків. Див. розділ «Завантаження класів» в описі команди *java*.

#### **-d *каталог***

Встановлює каталог, у якому (або в підкаталогах якого) потрібно зберегти файли класів. За замовчуванням *javac* зберігає отримані файли *.class* у тій же каталозі, де перебувають файли *.java* з оголошеннями цих класів. Якщо задано ключ *-d*, *каталог* розглядається як кореневий в ієрархії класів, а файли *.class* містяться в ньому або в його підкаталогах залежно від назви пакета класу. Таким чином, команда

```
javac -d /java/classes Checkers.java
```

поміщає файл *Checkers.class* у каталог */java/classes*, якщо у файлі *Checkers.java* немає оператора *package*. Якщо ж у вихідному файлі позначена приналежність до пакета, наприклад:

```
package com.davidflanagan.games;
```

то файл *.class* зберігається в каталозі *\Java\classes\com\davidflanagan\games*. Коли застосовується ключ *-d*, компілятор автоматично створює по відповідному шляху всі каталоги, у які будуть поміщені файли класів.

#### **-depend**

Включити рекурсивний пошук застарілих файлів класів, які необхідно перекомпілювати. Із цим ключем провадиться повна компіляція, що може сильно сповільнити процес. В Java 1.2 і наступних версіях цей ключ перейменований в *-Xdepend*.

#### **-deprecation**

Із цим ключем при кожному використанні АРІ, котре не рекомендується (deprecated), програма *javac* виводить попередження. За замовчуванням *javac* виробляє одне попередження про невірне використання АРІ на кожний вихідний файл. Ключ доступний в Java 1.1 і наступних версіях.

#### **-encoding *кодування***

Якщо кодування символів вихідного файлу відрізняється від системного, то воне повинне бути зазначеним в цьому ключі.

#### **-extdirs *шлях***

Визначає список каталогів, у яких буде вироблятися пошук JAR-Файлів розширень. Цей ключ використовується разом з *-bootclasspath* при крос-компілюванні для різних версій середовища виконання Java. Доступний в Java 1.2 і наступних версіях.

#### **-g**

Додати у вихідний файл класу номера рядків, вихідний код і інформацію про локальні змінні, котрі необхідні налогоджувачам. За замовчуванням *javac* додає тільки номера рядків.

#### **-g:none**

Не включати налогоджувальну інформацію у вихідні файли. Доступний в Java 1.2 і наступних версіях.

#### **-g:список\_ключових\_слів**

Визначає вид налогоджувальної інформації. Типи налогоджувальної інформації вказуються у вигляді *списку\_ключових\_слів* і розділяються комами. Припустимі ключові слова: *source* (інформація про вихідний код), *lines* (номера рядків), *vars* (інформація про локальні змінні). Ключ доступний в Java 1.2 і наступних версіях.

#### **-help**

Вивести список ключів.

#### **-Jjavaoption**

Передати аргумент *javaoption* безпосередньо інтерпретаторові Java. Наприклад: *-J-Xmx32m*. У параметрі *javaoption* не повинне бути пробілів; якщо по-

трібно передати інтерпретаторові кілька аргументів, необхідно задавати кілька ключів *-J*. Ключ доступний в Java 1.1 і наступних версіях.

***-nowarn***

Не показувати попереджень. Повідомлення про помилки, при використанні цього ключа, будуть виводяться однаково.

***-nowrite***

Не створювати файли класів. Вихідний код аналізується компілятором, але запис вихідних файлів не провадиться. Ключ застосовується для того, щоб перевірити, як буде компілюватися файл, не роблячи при цьому компіляції. Ключ доступний тільки в Java 1.0 і Java 1.1. В Java 1.2 і наступних версіях він відсутній.

***-o***

Дозволити оптимізацію файлу класу для збільшення швидкості його виконання. При використанні цього ключа файли класів мають більший розмір і довше компілюються. Крім того, їх складно налагоджувати. Ключ включений тільки у версії Java, що передують версії 1.2. Цей ключ не є сумісним з *-g*; при виборі *-o* автоматично вимикається *-g* і включається *-depend*.

***-source номер\_версії***

Визначає версію Java, у якій написаний код. Наприклад, для компіляції коду, у якому є присутнім оператор *assert*, необхідно використовувати ключ *-source 1.4*. Цей ключ також встановлює опцію *-target*. Ключ доступний в Java 1.4 і наступних версіях.

***-sourcepath шлях***

Визначає список каталогів, архівів ZIP і JAR, у яких провадиться пошук вихідних файлів. Кожний знайдений файл компілюється, якщо для нього немає відповідного файлу класу або вихідний файл є новішим за файлу класу. За замовчуванням шлях пошуку вихідних файлів збігається зі шляхом пошуку файлів класу. Ключ доступний в Java 1.2 і наступних версіях.

***-target номер\_версії***

Визначає формат файлу класу, що буде використовуватися при створенні вихідних файлів, за замовчуванням *номер\_версії* – 1.1, а створені файли класів можуть бути прочитані й виконані віртуальною машиною Java 1.0 і наступних версіях VM. Якщо вказати версію 1.2, *javac* збільшує номер версії файлу класу, і такий файл не може виконуватися інтерпретаторами Java 1.0 і Java 1.1. Формат

ти файлів класів різних версій практично не відрізняються; вказівка версії у файлі – це зручний спосіб не допустити запуск класу, що використовує нові можливості Java 1.2, у старих версіях інтерпретатора.

#### **-verbose**

Показувати повідомлення про кожну дію. Зокрема, при цьому виводяться імена всіх вихідних файлів, що компілюються, у тому числі й тих, які не були зазначені в командному рядку.

#### **-X**

Вивести довідкову інформацію про нестандартні ключі (вони починаються з -X). Цей ключ доступний в Java 1.2, Java 1.4 і наступних версіях.

#### **-Xdepend**

Включити рекурсивний пошук файлів, які необхідно перекомпілювати. Із цим ключем провадиться повна компіляція, що може сильно сповільнити процес. Ключ доступний тільки в Java 1.2; в Java 1.3 його немає.

#### **-Xstdout**

При використанні цього ключа повідомлення про попередження й про помилки виводяться в стандартний вихідний потік, а не в потік повідомлень про помилки. Ключ доступний тільки в Java 1.2.

#### **-Xstdout ім'я\_файлу**

Записувати повідомлення про попередження й про помилки в указаний файл, а не виводити їх у вікно консолі. Ключ доступний в Java 1.4 і наступних версіях.

#### **-Xswitchcheck**

Виводити попередження про секції case операторів switch, що не мають завершального оператора break.

#### **-Xverbosepath**

Виводити інформацію про розташування знайдених файлів класів і вихідних файлів. Ключ доступний тільки в Java 1.2.

### **Змінні оточення**

#### **CLASSPATH**

Визначає впорядкований список (з роздільником «:» в Unix або «;» в Windows) каталогів, файлів ZIP і JAR, у яких буде вироблятися пошук класів користувача й вихідних файлів. Якщо зазначено ключ *-classpath*, то він заміщає цю змінну.

**Див. також**

*java, jdb*



## Команда *jar*

Реалізація: JDK 1.1 і вище

Призначення: Архіватор Java

Синтаксис командного рядка:

```
jar c|t|u|x[f][e][m][M][O][v] [jar-файл] [маніфест]
      [-C каталог] [вхідні_файли]
jar -i [ jar-файл]
```

### Опис

Програма *jar* – це інструмент для створення й керування архівами Java ARchive (JAR). Файл JAR – це, по суті, ZIP архів, що містить файли класів Java, різні файли ресурсів, що використовуються цими класами, й (необов'язково) метадані. Остання містить у собі файл маніфесту (manifest), у якому приводиться зміст архіву й додаткова інформація про кожний файл.

За допомогою команди *jar* можна створювати архіви JAR, одержувати список файлів, що втримуються в архіві, і витягати з нього файли. В Java 1.2 і наступних версіях за допомогою цієї команди можна додавати файли в існуючий архів або оновлювати файл маніфесту архіву. В Java 1.3 і наступних версіях програма *jar* також може додавати у файл JAR елемент індексу.

Синтаксис команди *jar* нагадує синтаксис команди *tar* (tape archive) у системі Unix. Більшість ключів *jar* не є окремими параметрами, а входять у блок зв'язаних символів, переданих як один аргумент. Перша буква першого аргументу визначає, яку дію потрібно виконати. Інші букви необов'язкові. Залежно від того, які букви зазначені, застосовуються різні аргументи – імена файлів.

### Командні ключі

Перша буква першого ключа визначає основну дію, що повинна виконати програма *jar*. Можливі чотири значення:

**-c**

Створити новий JAR архів. Останнім параметром командного рядка повинен бути список вхідних файлів і каталогів. У створеному архіві першим елементом є файл маніфесту META-INF\MANIFEST.MF. У цьому інформаційному файлі, що створюється автоматично, наводиться зміст архіву JAR. Тут же перебувають профілі повідомлень для кожного файлу.

**-t**

Одержати список вмісту архіву JAR.

**-u**

Обновити вміст архіву JAR. Всі файли, що перераховані в командному рядку, додаються в архів. При використанні разом із ключем *-m* у файл коментарю додається інформація, що зазначена в командному рядку. Ключ доступний в Java 1.2 і наступних версіях.

**-x**

Витягти вміст архіву. Файли й каталоги, зазначені в командному рядку, витягаються у поточний робочий каталог. Якщо імена файлів і каталогів не зазначені, витягається весь вміст архіву JAR.

### **Ключі-модифікатори**

За кожним із чотирьох символів командних ключів може іти додатковий символ, що дає більш докладну інформацію про дію, яку потрібно виконати:

**-f**

Указує, що програма буде працювати з файлом JAR, ім'я якого міститься в командному рядку. Якщо цей ключ відсутній, програма читає файл JAR зі стандартного вхідного потоку або записує файл архіву в стандартний вихідний потік. Якщо ключ *-f* є присутнім, то командний рядок обов'язково повинний містити ім'я файлу JAR, з яким буде виконана дія.

**-m**

Коли програма *jar* створює або обновляє файл архіву, вона автоматично створює або обновляє файл маніфесту, що міститься в архіві META-INF\MANIFEST.MF. За замовчуванням маніфест містить тільки список файлів і каталогів архіву JAR. Але найчастіше буває необхідно, щоб у маніфесті файлу JAR утримувалася додаткова інформація. У цьому випадку й застосовується ключ *-m*, що вказує архіватору на те, що в командному рядку є заготівка маніфесту. Програма *jar* читає цей файл заготівки й записує інформацію з нього в створюваний файл META-INF\MANIFEST.MF. Цей ключ варто застосовувати тільки з командами *-c* або й *-u* не можна застосовувати з командами *-t* або *-x*.

**-M**

Застосовується з командами *-c* і *-u*. Якщо зазначено цей ключ, *jar* не створює файл маніфесту.

**-v**

Приписання про вивід докладної інформації про процес обробки архіву.

**-0**

Застосовується з командами *-c* і *-u*. Із цим ключем *jar* зберігає файли в архіві, не стискаючи їх. Тут обов'язково необхідно врахувати, що в ключі використовується цифра «нуль», а не буква «О».

### **Файли**

Перший ключ програми *jar* складається з першої букви команди, якої передує символ «-» (мінус) і стоячих за нею додаткових символів командних ключів без знака «-» (тобто, командний ключ являє собою рядок символів з лідируючим символом мінуса). За командним ключем перераховуються імена файлів *jar*.

#### ***jar-файл***

Якщо в першому ключі зазначена команда *-f*, то за ним повинне йти ім'я файлу, над яким потрібно виконати дію.

#### ***маніфест***

Якщо в першому ключі зазначена команда *-m*, то за ним повинне йти ім'я файлу, що містить маніфест. Якщо в першому ключі одночасно присутні букви *-f* і *-m*, то імена файлу JAR і файлу маніфесту повинні йти в тім же порядку, у якому йдуть букви *f* і *m*. Інакше кажучи, якщо *f* стоїть перед *m*, те ім'я файлу JAR повинне стояти перед ім'ям файлу маніфесту. Інакше, якщо *m* стоїть перед *f*, то ім'я файлу маніфесту повинне передувати ім'ю файлу JAR.

#### ***вхідні\_файли***

Перерахування файлів і каталогів, які потрібно включити в архів або витягти з нього.

### **Додаткові ключі**

Крім перерахованих вище, допускаються наступні ключі.

#### ***-C каталог***

Застосовується зі списком файлів, над якими необхідно виконати дію. Визначає каталог для наступних файлів і каталогів. Наступні файли й каталоги інтерпретуються як елементи зазначеного *каталогу* й включаються в архів JAR без префікса *каталогу*. Можна використовувати будь-яку кількість ключів *-C*; область дії кожного – до наступного ключа. Шлях, зазначений у ключі *-C* сприймається щодо поточного робочого каталогу й не залежить від попереднього ключа *-C*. Ключ *-C* доступний в Java 1.2 і наступних версіях.

#### ***-i jar-файл***

Ключ *-i* застосовується замість команд *-c*, *-t*, *-u* і *-x*. Якщо указаний цей ключ, то архіватор створює індекс всіх файлів JAR, перерахованих в *jar-файлі*.

Індекс поміщається у файл META-INF\INDEX.LIST архіву. Інтерпретатор Java або програма перегляду аплетів можуть використовувати цей індекс для оптимізації алгоритму пошуку класів і ресурсів і запобігання завантаження непотрібних файлів JAR. Ключ доступний в Java 1.3 і в наступних версіях.

### Приклади

Набір ключів команди *jar* може здатися занадто заплутаним, але користуватися цією командою досить легко. Щоб створити простий архів JAR, що містить всі файли класів з поточного каталогу й всі файли з підкаталогу *images*, досить увести наступну команду:

```
jar -cf my.jar *.class.images
```

Для одержання докладного списку вмісту архіву – команду:

```
jar -tvf your.jar
```

Витягти файл маніфесту з архіву JAR для перегляду або редагування:

```
jar -xf the.jar META-INF\MANIFEST.MF
```

Обновити маніфест файлу JAR:

```
jar -ufm my.jar manifest.template
```

Див. також

*jar signer*

### Команда *javadoc*

**Реалізація:** JDK 1.0 і вище

**Призначення:** Програма для створення документації Java

**Синтаксис командного рядка:**

```
javadoc [ключі] @list пакет... вихідні_файли...
```

### Опис

Програма *javadoc* генерує документацію API у форматі HTML (за замовчуванням) для зазначених пакетів і класів. У командному рядку може бути зазначена будь-яка кількість імен пакетів і вихідних файлів Java. Для зручності при роботі з більшою, кількістю ключів командного рядка або імен пакетів і класів можна помістити їх у довільний файл і вказати в командному рядку символ «@» (At комерційне), а відразу за ним – ім'я цього файлу.

Програма *javadoc* використовує компілятор *javac* при роботі із зазначеними вихідними файлами й файлами із зазначених пакетів. На основі отриманої від компілятора інформації вона створює докладну документацію API. При цьому використовуються всі коментарі з вихідних файлів.

Крім ім'я вхідного файлу, необхідно вказати повний шлях до нього. Програму *javadoc* можна застосовувати для створення документації до всього пакета. Для того щоб програма *javadoc* могла правильно знайти вихідний код пакета, шлях до якого не входить у список шляхів до класів, потрібно вказати ключ *-sourcepath*.

За замовчуванням програма *javadoc* створює документи у форматі HTML, але можна визначити клас *doclet*, що генерує документацію в потрібному форматі. Ви можете написати власний клас *doclet* за допомогою *doclet API* з пакета *com.sun.javadoc*. Цей пакет описаний у стандартній документації до Java 1.2 і наступних версій.

В Java 1.2 програма *javadoc* повністю оновлена. Тут наведений опис цієї програми з Java 1.2 і наступних версіях, а відмінності від попередніх версій не розглянуті.

### **Ключі командного рядка**

Команда *javadoc* допускає безліч різних ключів. Частина з них являє собою стандартні ключі, завжди розпізнавані програмою. Інші ключі визначені в класі *doclet*, що створює документацію. Нижче наведені ключі, припустимі при використанні стандартного *doclet* у форматі HTML.

#### **-1.1**

Використовувати стиль вихідного файлу й структуру каталогів, які прийняті в *javadoc* з Java 1.1. Цей ключ доступний тільки в Java 1.2 і Java 1.3 і є вилученим у Java 1.4.

#### **-author**

Включити у вихідний документ інформацію про авторство, що визначається параметром *@author*. Ключ доступний тільки в *doclet*, установленому за замовчуванням.

#### **-bootclasspath**

Визначає розташування додаткових системних класів. Цей ключ можна використовувати при крос-компіляції. Докладний опис представлений вище при розгляді команди *javac*.

#### **-bottom текст**

Визначає *текст*, що поміщається в кінець кожної сторінки створюваного документа, *текст* може містити теги HTML (див. також *-footer*). Ключ доступний тільки в *doclet*, установленому за замовчуванням.

#### **-breakiterator**

Використовувати алгоритм `java.text.BreakIterator` для визначення кінця зведеного речення (summary sentence) у коментарях. Ключ доступний тільки в `doclet`, установленому за замовчуванням.

**-charset кодування**

Установити набір символів вихідного файлу. Кодування вихідного файлу залежить від кодування коментарів з вихідного файлу. Значення параметра *кодування* використовується у тегу `<META>` вихідного файлу HTML. Ключ доступний тільки в `doclet`, що установлений за замовчуванням.

**-classpath шлях**

Визначає шлях, котрий використовується програмою *javadoc* при пошуку файлів класів. Якщо зазначено ключ *-sourcepath*, шлях також застосовується при пошуку файлів з вихідним кодом. Оскільки програма *javadoc* використовує компілятор *javac*, їй потрібен шлях до файлів всіх класів, на які посилається пакет, що документується. Більш докладно ключ *-sourcepath* і змінна середовища `CLASSPATH` описані вище при розгляді в команд *java* і *javac*.

**-d каталог**

Установити каталог, у який будуть поміщені вихідні документи HTML. Якщо цей ключ не зазначений, використовується поточний каталог. Ключ доступний тільки в `doclet`, установленому за замовчуванням.

**-docencoding кодування**

Визначає кодування вихідних документів HTML. Ім'я кодування може не відповідати імені набору символів, зазначеного в ключі *-charset*. Ключ доступний тільки в `doclet`, установленому за замовчуванням.

**-docfilessubdirs**

Із цим ключем програма проводить рекурсивне копіювання всіх підкаталогів каталогу *doc-files*, а не просте копіювання всіх файлів, що втримуються в каталозі *doc-files*. Ключ доступний, тільки в `doclet`, установленому за замовчуванням.

**-doclet ім'я\_класу**

Визначає ім'я класу `doclet`, що буде використовуватися при створенні документів. Якщо цей ключ не зазначений, програма *javadoc* генерує документи у форматі HTML за допомогою `doclet`, установленого за замовчуванням.

**-docletpath шлях**

Цей ключ встановлює шлях, за яким перебуває клас, зазначений у ключі *-doclet*, якщо цей шлях не зазначений у списку шляхів до класів.

### **-doctitle *текст***

Ключ визначає заголовок оглядового файлу документації. Цей файл звичайно бачать читачі під час перегляду документації. Заголовок може містити теги HTML. Ключ доступний тільки в doclet, що встановлений за замовчуванням.

### **-encoding *назва\_кодування***

Указує кодування символів у коментарях із вхідних файлів. Воно може відрізнятися від кодування вихідних файлів, котре встановлено ключем *-docencoding*. За замовчуванням використовується кодування, що є основним у даної ОС.

### **-exclude *пакети***

Виключити зазначені *пакети* з набору пакетів, що є визначеним ключем *-subpackages*. Параметр *пакети* являє собою список імен пакетів. Роздільником списку служить двокрапка. Ключ доступний тільки в doclet, установленому за замовчуванням.

### **-excludedocfilessubdir *каталоги***

Ця опція виключає зазначені підкаталоги каталогу *doc-files*, якщо задано ключ *-docfilessubdirs*. Наприклад, його можна застосовувати для виключення каталогів управління версіями. Параметр *каталоги* являє собою список імен каталогів, розділених двокрапками і такими, що є підкаталогами каталогу *doc-files*. Ключ доступний тільки в doclet, установленому за замовчуванням.

### **-extdirs *список\_каталогів***

Установити список каталогів стандартних розширень. Цей ключ буває необхідний тільки при крос-компіляції разом із ключем *-bootclasspath*, Докладніше див. опис команди *javac*.

### **-footer *текст***

Цей ключ визначає текст, відображуваний наприкінці кожної сторінки файлу документа, праворуч від навігаційної панелі. Параметр *текст* може містити теги HTML. Див. також *-bottom* і *-header*. Ключ доступний тільки в doclet, установленому за замовчуванням.

### **-group *заголовок список\_пакетів***

Із цим ключем програма *javadoc* створює оглядову сторінку верхнього рівня, у якій перераховані всі пакети, що описані в створюваній документації. За замовчуванням створюється окрема таблиця, за звичай складена з імен цих пакетів за абеткою. Але за допомогою цього ключа можна розбити список пакетів

на групи зв'язаних між собою пакетів. Параметр *заголовок* визначає ім'я групи пакетів, наприклад «Базові пакети». Параметр *список\_пакетів* – це список імен пакетів, розділених двокрапкою. Імена пакетів можуть закінчуватися груповим символом «\*» (зірочка). Командний рядок програми *javadoc* може містити будь-яку кількість ключів *-group*, наприклад:

```
javadoc -group-"AWT Packages" java.awt.* -group  
"Swing Packages" javax.accessibility:javax.swing.*  
-header текст
```

Визначає заголовки, відображувані у верхній частині кожної сторінки HTML файлу, праворуч від верхньої навігаційної панелі, *текст* може містити теги HTML (див. також *-footer*, *-doctitle* і *-windowtitle*). Ключ доступний тільки в *doclet*, установленому за замовчуванням.

**-help**

Вивести довідкову інформацію про програму *javadoc*.

**-helpfile файл**

Визначити файл HTML, що містить довідку по використанню документації. Програма *javadoc* поміщає посилання на нього в усі створені файли. Якщо цей ключ не зазначений, створюється файл довідки за замовчуванням. Ключ доступний тільки в *doclet*, установленому за замовчуванням.

**-Jjavaoption**

Пропонує програмі передати *javaoption* безпосередньо інтерпретаторові *java*. При обробці великої кількості пакетів для збільшення обсягу виділюваної для програми *javadoc* пам'яті може знадобитися використання цього ключа. Наприклад:

```
javadoc -J-Xmx64m
```

У зв'язку з тим, що ключі *-J* передаються безпосередньо інтерпретаторові *java* до старту програми *javadoc*, їх не можна поміщати в зовнішній файл, зазначений у командному рядку у вигляді *@list*.

**-link url**

Цей ключ визначає абсолютне або відносне посилання URL на каталог верхнього рівня іншого документа, створеного за допомогою *javadoc*. Зазначене посилання є базовим для всіх посилань поточного документа на пакети, класи, методи й поля, які не описані в цьому документі. Наприклад, при створенні документації до пакетів власної розробки можна за допомогою ключа зв'язати її з



документацією базових Java API. Ключ доступний тільки в doclet, установленому за замовчуванням.

Каталог, зазначений у параметрі *url*, повинен містити файл з ім'ям *package-list*, що повинен бути доступний для читання під час виконання *javadoc*. Цей файл попередньо створюється також командою *javadoc*. Він містить список пакетів, описаних у документах, на яких посилається *url*.

Можна використовувати відразу кілька ключів *-link*, але в ранніх випусках Java 1.2 це не буде працювати. Якщо ключ *-link* не зазначений, гіперпосилання на класи, не описані в поточній документації, не створюються.

***-linkoffline url package-list***

Робить ті ж дії, що й ключ *-link*, за винятком того, що *package-list* вказується в командному рядку. Ключ *-linkoffline* потрібно використовувати, якщо каталог, на який посилається *url*, не містить файлу *package-list* або якщо цей файл недоступний під час виконання програми *javadoc*. Ключ доступний тільки в doclet, установленому за замовчуванням.

***-linksource***

При використанні цього ключа створюється версія HTML файлу для кожного оброблюваного вихідного файлу, а в усі сторінки додаються посилання на цю версію. Ключ доступний тільки в doclet, установленому за замовчуванням.

***-locale language\_country\_variant***

Визначає регіональні параметри (locale), використовувані в документації. Параметр ключа *language\_country\_variant* використовується при пошуку файлу ресурсів, що містить переклад повідомлень і тексту для вихідних файлів.

***-nocomment***

Із цим ключем ігноруються всі коментарі у вихідних кодах, а створювана документація містить у собі тільки API без супровідного тексту. Ключ доступний тільки в doclet, установленому за замовчуванням.

***-nodeprecated***

Пропускати некоректні (deprecated) місця коду. При зазначенні цього ключа автоматично використовується *-nodeprecatedlist*. Ключ доступний тільки в doclet, установленому за замовчуванням.

***-nodeprecatedlist***

При зазначенні цього ключа *javadoc* не створює файл *deprecated-list.html* і не включає посилання на нього в навігаційну панель. Ключ доступний тільки в doclet, установленому за замовчуванням.

### **-nohelp**

При використанні цього ключа *javadoc* не створює файл довідки й не включає посилання на нього в навігаційну панель. Ключ доступний тільки в doclet, установленому за замовчуванням.

### **-noindex**

Вказівка програмі не створювати файлів з індексом. Ключ доступний тільки в doclet, установленому за замовчуванням.

### **-nonavbar**

Не створювати навігаційні панелі на початку й наприкінці файлів. При цьому текст, зазначений у ключах *-header* і *-footer*, також буде ігноруватися. Даний ключ корисний при створенні документації, призначеної для печатки. Ключ доступний тільки в doclet, установленому за замовчуванням.

### **-noqualifier пакети | all**

Якщо зазначено цей ключ, програма *javadoc* пропускає імена пакетів у назвах класів, що входять у поточний оброблюваний пакет. Крім того, пропускаються імена всіх пакетів, зазначених у параметрі. Якщо як параметр задане ключове слово *all*, то виключаються імена всіх пакетів. Сам параметр *пакети* – це список імен пакетів, розділений комами. У списку може бути присутнім груповий символ \* (зірочка), що вказує на підпакети. Наприклад, ключ команди:

### **-noqualifier java.io, java.nio.\***

виключає імена пакетів з назв всіх класів пакетів *java.io*, *java.nio* і його підпакетів. Ключ доступний тільки в doclet, установленому за замовчуванням.

### **-nosince**

Ігнорувати всі теги *@since* у коментарях. Ключ доступний тільки в doclet, установленому за замовчуванням.

### **-notree**

Із цим ключем *javadoc* не створює діаграму ієрархії класів *tree.html* і не поміщає посилання на неї в навігаційну панель. Ключ доступний тільки в doclet, установленому за замовчуванням.

### **-overview ім'я\_файлу**

Із цим ключем *javadoc* читає оглядовий коментар із зазначеного файлу й використовує його при створенні оглядової сторінки. Цей файл не містить вихідного Java-коду, тому коментарі у ньому можна не виділяти символами */\*\** і *\*/*.

### **-package**

При створенні вихідних файлів крім членів класів, оголошених як `public` або `protected`, ураховуються члени класів, видимі тільки усередині пакета.

**-private**

При створенні вихідних файлів ураховуються всі члени класів, у тому числі видимі тільки усередині пакета або оголошені як `private`.

**-protected**

Вказівка враховувати тільки члени класів, оголошені як `public` і `protected`. Цей ключ використовується за замовчуванням.

**-public**

Ураховувати тільки члени класів, оголошені як `public`. Всі інші члени класів ігноруються.

**-quiet**

Заборонити вивід будь-якої інформації, крім попереджень і повідомлень про помилки.

**-serialwarn**

Видавати попередження, якщо коментарі до серіалізуємого класу не містять формат серіалізації в тегах `@serial`. Ключ доступний тільки в `doclet`, установленому за замовчуванням.

**-source 1.4**

Цей ключ використовується при обробці коду на Java 1.4, у якому застосовується оператор `assert` нового типу.

**-sourcepath *шлях***

Установлює шлях пошуку вихідних файлів (як правило, це один каталог верхнього рівня). Програма *javadoc* використовує цей шлях при пошуку файлів з вихідним кодом зазначеного пакета.

**-splitindex**

Створити кілька індексних файлів, кожний з яких відповідає одній букві алфавіту. Цей ключ варто застосовувати при створенні документації до великої кількості коду, інакше програма *javadoc* створить тільки один індексний файл, незручний у використанні через великий розмір. Ключ доступний тільки в `doclet`, установленому за замовчуванням.

**-stylesheetfile *файл***

Визначає файл таблиці стилів CSS для створюваних HTML документів, при цьому *javadoc* вставляє в документи посилання на даний файл. Ключ доступний тільки в `doclet`, установленому за замовчуванням.

### **-subpackages пакети**

Обробляти зазначені пакети разом з усіма підпакетами. Параметр *пакети* являє собою список імен пакетів або префіксів імен пакетів, розділених комами. Часто зручніше вказати цей ключ, ніж повністю перераховувати імена всіх необхідних пакетів. Наприклад:

### **-subpackages java:javah**

Див. також *-exclude*. Ключ доступний тільки в doclet, установленому за замовчуванням.

### **-tag ім'я\_тега:where:текст\_заголовка**

При використанні цього ключа програма *javadoc* для тегу, зазначеного в параметрі *ім'я\_тегу*, вставляє *текст\_заголовка* поруч з будь-яким текстом, що йде за тегом. Це дає можливість використовувати в коментарях прості користувальницькі теги (з тим же синтаксисом, як у тегах *@return* і *@author*). Параметр *where* – це рядок із символів, що визначають тип коментарів, у яких дозволений користувальницький тег. У рядок можуть бути включені наступні символи: *a* (тег дозволений скрізь), *p* (у пакетах), *t* (у типах – класах і інтерфейсах), *c* (у конструкторах), *m* (у методах) і *f* (у полях).

Друге застосування ключа *-tag* – визначення порядку, у якому обробляються теги й з'являються вихідні дані. Після ключа *-tag* можна вказати імена стандартних тегів, щоб задати потрібний порядок. У даний список можна включати користувальницькі теги й класи виду «taglet». Ключ доступний тільки в doclet, установленому за замовчуванням.

### **-taglet ім'я\_класу**

Визначає ім'я класу типу «taglet», що буде обробляти користувальницький тег. Техніка написання таких класів не розглядається. Теги, зазначені в ключі *-taglet*, можуть стояти між тегами, зазначеними в ключі *-tag*, визначаючи порядок, у якому вони будуть оброблятися. Ключ доступний тільки в doclet, установленому за замовчуванням.

### **-tagletpath шлях**

У цьому ключі через кому вказується список файлів JAR або каталогів, у яких буде вироблятися пошук класів типу «taglet». Ключ доступний тільки в doclet, установленому за замовчуванням.

### **-use**

Створює для кожного класу й пакета додатковий файл, у який поміщаються посилання на кожний випадок використання цього класу або пакета.

### **-verbose**

Відображати додаткові інформаційні повідомлення при обробці вихідних файлів.

### **-version**

Цей ключ, на відміну від однойменного ключа інших команд (див., наприклад, *javac*), використовується не для виводу версії програми *avadoc* на екран, а для того, щоб включити дані із тегів `@version` у створювані файли. Ключ доступний тільки в `doclet`, установленому за замовчуванням.

### **-windowtitle *текст***

Помістити *текст* у тег `<TITLE>` кожного створюваного файлу. За звичай текст із цього тегу поміщається в рядок заголовка, а також у список закладок і в журнал веб-браузера. Цей *текст* не повинен містити HTML тегів. Див. також *doctitle* і *-header*. Ключ доступний тільки в `doclet`, установленому за замовчуванням.

## **Змінні оточення**

### **CLASSPATH**

Ця змінна визначає шлях, що *avadoc* використовує за замовчуванням при пошуку файлів класів і вихідних файлів. Ключі *-classpath* або *-sourcepath* заміщають значення цієї змінної. Докладніше вона розглянута при описі програм *java* і *javac*.

**Див. також**

*java*, *javac*

## **Команда *jdb***

**Реалізація:** JDK 1.0 і вище

**Призначення:** Налогоджувач Java

**Синтаксис командного рядка:**

***jdb* [ключі] клас [ключі\_програми]**

***jdb* ключі\_з'єднання**

### **Опис**

Програма *jdb* – це налогоджувач класів Java. Вона працює в текстовому режимі, управляється з командного рядка. Команди *jdb* нагадують команди налогоджувачів *dbx* і *gdb* (ОС Unix) для програм на C і C++.

Програма *jdb* написана на Java, тому вона сама виконується інтерпретатором Java. Якщо *jdb* запускається із зазначенням ім'я класу, вона запускає другу

копію інтерпретатора *Java* з усіма параметрами, зазначеними в її командному рядку. Нова копія інтерпретатора запускається зі спеціальними ключами, що дозволяють спілкування з *jdb*. Ця копія інтерпретатора запускає зазначений файл класу, а потім зупиняється до початку виконання байт-коду чекаючи команд налагоджування.

За допомогою *jdb* можна також налагоджувати програми, що виконуються вже іншою копією інтерпретатора *Java*. Для цього потрібно використовувати спеціальні опції інтерпретатора й налагоджувача. В *Java 1.3* архітектура налагодження перетерпіла серйозні зміни. Зокрема, з'явилася можливість підключення *jdb* до інтерпретатора, що уже виконується.

### **Синтаксис виражень *jdb***

Такі команди налагоджувача, як *print*, *dump* і *suspend* дозволяють звертатися до класів, об'єктів, методів, полів і потоків налагоджуваної програми. Можна звертатися до класів за ім'ям, при цьому зазначення ім'я пакета не є обов'язковим. До статичних членів класу також можна звертатися за ім'ям. До об'єктів можна звертатися за ідентифікатором (восьмизначне шістнадцяткове ціле число). Якщо налагоджуваний клас містить інформацію про локальні змінні, то при звертанні до об'єктів можна використовувати їхні імена. Для звертання до полів об'єкта й до елементів масиву можна застосовувати звичний синтаксис мови *Java*. Цей синтаксис можна використовувати й для написання досить складних виразів. В *Java 1.3* дозволяється застосовувати цей синтаксис навіть для виклику методів.

Для багатьох команд *jdb* необхідно вказати потік. Кожному потокові привласнюється цілочисельний ідентифікатор. Для звертання до потоку використовується синтаксис *t@n*, де *n* – ідентифікатор потоку.

### **Ключі**

При виклику *jdb* із зазначенням ім'я файлу класу можна використовувати будь-які ключі інтерпретатора *Java*. Крім того, *jdb* підтримує ряд власних ключів, описаних нижче.

#### ***-attach [хост:]порт***

Підключитися до клієнтської ВМ, що виконується на зазначеному хості (за замовчуванням на локальному хості) і котра очікує з'єднання на указаному порту. Ключ доступний в *Java 1.3* і наступних версіях.

Щоб налагоджувач *jdb* міг з'єднатися із ВМ під час виконання, віртуальна машина повинна бути запущена командою наступного виду:

**java -Xdebug -Xrunjdp:transport=dt\_socket,address=8000,server=y,suspend**

Оскільки процес підключення налагоджувача до віртуальної машини досить складний, то архітектурою *jdb* в Java 1.3 передбачений великий набір опцій для керування з'єднанням налагоджувача з інтерпретатором. Докладний опис цих опцій виходить за рамки даного додатка й тому не приводиться.

**-help**

Видати інформацію про доступні ключі.

**-host ім'я\_хосту**

В Java 1.2 і попередніх версіях цей ключ використовується для з'єднання з інтерпретатором, що уже виконується. Параметр ключа визначає ім'я хосту, на якому виконується сесія інтерпретатора (за замовчуванням це локальний хост). Даний ключ необхідно застосовувати разом із ключем *-password*. В Java 1.3 цей ключ замінений ключем *-attach*.

**-launch**

Запустити зазначений додаток при старті *jdb*. Це дозволяє уникнути явного використання команди *run*. Ключ доступний в Java 1.3 і наступних версіях.

**-password пароль**

В Java 1.2 і наступних версіях цей ключ визначає пароль до віртуальної машини Java на зазначеному хості. При використанні разом з *-hostname* даний ключ дозволяє *jdb* з'єднатися із працюючим інтерпретатором, що повинен бути запущений із ключем *-debug* або *-Xdebug*, що включає відображення пароля. В Java 1.3 цей ключ замінений на *-attach*.

**-sourcepath шлях**

Установити шлях пошуку вихідних файлів, які будуть використовуватися при налагодженні відповідних файлів класів. Якщо цей ключ не зазначений, *jdb* вибирає шлях пошуку, визначений за замовчуванням. Ключ доступний в Java 1.3 і наступних версіях.

**-tclassic**

Використовувати «класичну» ВМ замість клієнтської (HotSpot), застосовуваної за замовчуванням в Java 1.3. Ключ доступний в Java 1.3 і наступних версіях.

**-version**

Із цим ключем *jdb* показує номер версії й завершує свою роботу.

## Команди

Після запуску й підключення до віртуальної машини, під час сеансу налагодження можна здійснювати керування програмою налагоджувача за допомогою досить великого набору інтерактивних команд програми. Ці команди дозволяють ставити контрольну точку в місцях, де виникає той або інший указаний виняток. Якщо виняток не зазначений, виводиться список перехоплених на даний момент винятків (команда *catch [виняток]*). Скасовувати контрольні точки по цих винятках (команда *ignore*). Виводити список всіх завантажених класів (*classes*). Вивести значення всіх полів зазначеного об'єкта або об'єктів (команда *dump*). Ця ж команда, якщо вказати ім'я класу, виводить значення всіх статичних методів і змінних класу, а також батьківський клас і список реалізованих інтерфейсів. Вивести рядок вихідного коду із зазначеним номером, а також кілька попередніх і наступних рядків; якщо номер рядка не зазначений, застосовується номер рядка з поточного фрейму стека поточного потоку. Виводяться рядки вихідного файлу поточного фрейму стека, що належить поточному потоку. І багато, багато чого іншого. Нарешті, вивести список всіх команд із коротким описом кожної (команда *?* або її синонім *help*). Повний опис прийомів роботи із програмою налагоджувача *jdb* вимагає окремого посібника з її використання, що не входить до завдання даного додатка. Більш докладно із програмою можна ознайомитися по документації на сайті фірми Oracle [15], яка завершила придбання Sun Microsystems і офіційно володіє правами на всі розробки Java або в книзі [3].

### Змінні оточення

#### **CLASSPATH**

Визначає впорядкований список каталогів, файлів ZIP і JAR (розділених двокрапками в Unix або крапками з комами в Windows), у яких *jdb* буде шукати визначення класів. Якщо ця змінна визначена, *jdb* автоматично поміщає в кінець списку шлях до системних класів. У протилежному випадку шляхом пошуку за замовчуванням вважається поточний каталог і каталог системних класів. Ця змінна середовища заміщається ключем *-classpath*.

### Див. також

*java*



## Команда *appletviewer*

Реалізація: JDK 1.0 і вище

Призначення: програма перегляду аплетів Java

Синтаксис командного рядка:

```
appletviewer [ключі] url | ім'я_файлу ...
```

### Опис

Програма *appletviewer* читає або завантажує HTML документи, зазначені в командному рядку через ім'я файлу або URL. Далі вона завантажує всі аплети, на які посилаються ці файли, і запускає кожний аплет в окремому вікні. Якщо зазначені документи не містять аплетів, то *appletviewer* не робить ніяких дій. *appletviewer* розпізнає аплети, указані в тегах <APPLET>, а в Java 1.2 і наступних версіях – у тегах <OBJECT> і <EMBED>.

### Ключі командного рядка

Програма *appletviewer* має наступні ключі командного рядка:

**-debug**

Із цим ключем *appletviewer* запускається в налагоджуваче Java (*jdb*), і можна налагоджувати аплети, на які посилається документ.

**-encoding *enc***

За допомогою параметра *enc* задається кодування, у якому *appletviewer* буде читати вміст файлу або URL. Ключ застосовується для перекодування значень параметрів аплету в Unicode. Ключ доступний в Java 1.1 і наступних версіях.

**-J *javaoption***

Цей ключ передає значення *javaoption* інтерпретаторові Java як параметр командного рядка. В *javaoption* не повинне бути пробілів. Для передачі параметра з декількох слів потрібно задати кілька ключів *-J*. Ключ доступний в Java версії 1.1 і наступних версіях.

Також допускаються ключі *-classic*, *-native* і *-green* (докладніше див. команду *java*).

### Команди

Програма запускається у віконному режимі. Кожне вікно програми *appletviewer* має окреме меню Applet (Аплет). Призначення пунктів цього меню й виконувані з їхньою допомогою команди цілком очевидні з їхніх назв, і не вимагають окремих пояснень. Більш докладно з ними можна ознайомитися, на-

приклад, в [3]. Там же досить повно роз'яснені такі параметри, як файли властивостей, властивості безпеки й властивості прокси-серверів.

### **Змінні оточення**

#### **CLASSPATH**

В Java 1.0 і Java 1.1 *appletviewer* використовує змінну середовища CLASSPATH точно так само, як інтерпретатор (див. подробиці в описі команди *java*). У той час, як в Java 1.2 і наступних версіях для більше точної імітації веб-браузера *appletviewer* ігнорує цю змінну.

#### **Див. також**

*java, javac, jdb*

### **Команда *jarsigner***

**Реалізація:** Java 2 SDK 1.2 і вище

**Призначення:** засіб підпису й верифікації файлів JAR

**Синтаксис командного рядка:**

***jarsigner* [ключі] *jar-файл* *signer***

***jarsigner* -verify [ключі] *jar-файл***

#### **Опис**

Команда системи *jarsigner* додає цифровий підпис у файл, указаний у параметрі *jar-файл*. Якщо задано ключ *-verify*, *jarsigner* перевіряє цифровий підпис (або підписи) файлу JAR. Параметр *signer* – це псевдонім власника цифрового підпису. Псевдонім не чутливий до регістра. Ім'я, зазначене в параметрі *signer*, використовується для знаходження секретного ключа, що генерує підпис.

Підписування файлу, є поручительством за вміст архіву. Тим самим гарантується, що файл JAR містить тільки нешкідливий код, що він не порушує закони про авторське право й т.і. При перевірці цифрового підпису можна встановити особу, котра підписалась і визначити, чи змінювався вміст файлу, чи був він ушкоджений з моменту включення в нього цифрового підпису. Не варто плутати поняття перевірки цифрового підпису й довіри особі, що поставила підпис.

Програми *jarsigner* і *keytool* заміняють застарілу програму *javakey* з Java 1.1.

## Ключі

В *jarsigner* є кілька ключів; багато які з них визначають спосіб пошуку секретного ключа для псевдоніма *signer*. Більшість ключів не використовуються разом із ключем *-verify*, що застосовується для перевірки файлів JAR:

### **-certs**

Якщо цей ключ застосовується разом з *-verify* або *-verbose*, програма виводить інформацію про сертифікати відкритого ключа, пов'язаних з підписаним файлом JAR.

### **-Jjavaoption**

Передає зазначений параметр *javaoption* безпосередньо інтерпретаторові.

### **-keypass *пароль***

Пароль для відкритого ключа власника підпису *signer*. Якщо цей ключ не використовується, *jarsigner* пропонує ввести пароль.

### **-keystore *url***

Сховище ключів *keystore* – це файл, що містить ключі й сертифікати. Даною командою визначається ім'я або URL файлу ключів, у якому провадиться пошук сертифікатів відкритих і секретних ключів особи, що поставила підпис (*signer*). За замовчуванням, це файл *.keystore*, що перебуває в домашньому каталозі користувача, що обумовлюється значенням системної змінної `HOME`. Цей шлях також указує на місцезнаходження файлу ключів, використовуваних програмою *keytool*.

### **-sigfile *базове\_ім'я***

Визначає базові імена файлів *.SF* і *.DSA*, що додаються в каталог `META-INF` файлу JAR. Якщо не вказувати цей параметр, базові імена файлів будуть створені на основі ім'я *signer*.

### **-signedjar *вихідний\_файл***

Визначає ім'я підписаного файлу, що створюється програмою *jarsigner*. Якщо цей параметр не зазначений, програма замінює *jar-файл*, зазначений у командному рядку.

### **-storepass *пароль***

Визначає пароль, що підтверджує цілісність файлу ключів, але не секретний ключ, яким виконується шифрування. Якщо цей параметр опущений, *jarsigner* пропонує ввести пароль. Параметр використовується якщо пароль відмінний від пароля, що задається для відкритого ключа (див. *keypass*).

### **-storetype *тип***

Визначає тип файлу ключів, що вказаний в параметрі *-keystore*. За замовчуванням використовується системний тип; у більшості систем це Java Keystore, або «JKS». Якщо в системі встановлене розширення Java Cryptography Extension, можна замість «JKS» використовувати «JCEKS».

**-verbose**

Показувати додаткову інформацію про процес підпису й верифікації.

**-verify**

Із цим ключем *jarsigner* перевіряє підпис указанного файлу JAR.

**Див. також**

*jar, keytool, javakey*

**Команда *extcheck***

**Реалізація:** Java 2 SDK 1.2 і вище

**Призначення:** Утиліта для визначення конфлікту версій JAR

**Синтаксис командного рядка:**

***extcheck* [-verbose] *jar-файл***

**Опис**

Програма *extcheck* перевіряє, чи встановлено в додатку розширення (extension), що міститься в зазначеному *jar-файлі* або більш пізня версія цього розширення. Перевірка здійснюється по атрибутах Specification-Title і Specification-Version файлу маніфесту (manifest) архіву JAR і всіх файлів JAR, що містяться в системному каталозі розширення.

Програма призначена для використання в сценаріях автоматичної установки. Якщо не зазначений ключ *-verbose*, результати перевірки не виводяться. У цьому випадку програма повертає код завершення 0, якщо зазначене розширення може бути успішно встановлене без конфліктів із уже встановленими розширеннями. Код завершення приймає ненульове значення, якщо розширення з таким же ім'ям уже встановлено, а його номер версії більше або дорівнює номеру версії зазначеного файлу.

**Ключі командного рядка**

**-verbose**

Скласти список встановлених розширень і вивести результати перевірки.

**Див. також**

*jar*

## Команда *javah*

**Реалізація:** JDK 1.0 і вище

**Призначення:** Генератор заглушок мовою C для методів, залежних від платформи

**Синтаксис командного рядка:**

**javah [ключі] імена\_класів**

### Опис

Програма *javah* створює заголовні й вихідні файли на C (файли .h і .c), які використовуються при реалізації рідних (native) методів Java мовою C. Інтерфейс таких методів в Java 1.1 перетерпів зміни. В Java 1.1 і попередніх версіях програма *javah* створює файли методів старого типу. В Java 1.1 програма *javah* із ключем *-jni* створює файли нового типу. В Java 1.2 і наступних версіях цей ключ застосовується за замовчуванням.

Представлена інформація ставиться тільки до самої програми *javah*. Повний опис способів реалізації власних методів Java мовою C виходить за рамки цього додатка (про це можна довідатися, наприклад, в [Ошибка! Источник ссылки не найден.]).

### Ключі

#### **-bootclasspath**

Визначає шлях пошуку системних класів (див. команду *javac*). Ключ доступний в Java 1.2 і наступних версіях.

#### **-classpath *шлях***

Визначає шлях пошуку класів, зазначених у командному рядку. Цей ключ заміщає значення змінної оточення CLASSPATH. У версіях Java, що передують 1.2, у цьому ключі можна вказати розташування системних класів і розширень. В Java 1.2 і наступних версіях можна вказати тільки шлях до класів додатків (див. *-bootclasspath*). Шлях пошуку класів докладно розглянутий при описі команди *java*.

#### **-d *каталог***

Визначає каталог, у який будуть записані файли, що створені програмою *javah*. За замовчуванням вона зберігає файли в поточному каталозі. Цей ключ не можна використовувати одночасно із ключем *-o*.

#### **-force**

Завжди записувати вихідні файли, навіть якщо вони не містять нічого корисного.



### **-help**

Із цим ключем програма *javah* виводить коротку довідку по роботі із програмою, після чого завершується.

### **-jni**

Замість файлів інтерфейсу Java Native Interface (JNI) старого типу, що використовувався в JDK 1.0, створювати заголовні файли, сумісні з новим інтерфейсом. В Java 1.2 і наступних версіях цей ключ використовується за замовчуванням (див. також ключ *-old*). Ключ доступний в Java 1.1 і наступних версіях.

### **-o *вихідний\_файл***

Зібрати всі вихідні дані в один *вихідний\_файл*, а не створювати окремі файли для кожного класу.

### **-old**

Створювати вихідні файли для методів старого типу, залежних від платформи (Java 1.0). У версіях Java, що передували 1.2, такий тип файлів використовувався за замовчуванням (див. також *-jni*). Ключ доступний в Java 1.2 і наступних версіях.

### **-stubs**

Створювати для класів замість заголовних файлів мови C файли-заглушки .c (stub files). Цей ключ призначений для використання тільки з інтерфейсом методів Java 1.0, залежних від платформи (див. також *-old*).

### **-td *каталог***

Визначає каталог, у який програма *javah* буде поміщати тимчасові файли. В ОС Unix за замовчуванням використовується каталог */tmp*, а в Windows – каталог заданий у змінній оточення TEMP.

### **-trace**

Включити у файли-заглушки команди для трасування. В Java 1.2 і наступних версіях цей ключ виключений через непотрібність. Замість цього можна застосовувати інтерпретатор Java із ключем *-verbose:jni*.

### **-v, -verbose**

Включити режим відображення інформаційних повідомлень. У цьому режимі *javah* виводить повідомлення про кожну дію. Ключ *-v* доступний в Java 1.2 і наступних версіях. Допускається повна форма: *-verbose*.

### **-version**

Вивести номер версії програми *javah*.

### **Змінні оточення**

#### **CLASSPATH**

Визначає шлях пошуку класів, що використовується за замовчуванням. Змінна докладно описана в розділі опису команди *java*.

**Див. також**

*java, javac*

### **Команда *javap***

**Реалізація:** JDK 1.0 і вище

**Призначення:** Дісасемблер класів Java

**Синтаксис командного рядка:**

***javap* [ключі] імена\_класів**

### **Опис**

Програма *javap* читає файли класів, зазначені в параметрі *класи* командного рядка, і відображає текст API цього класу в зрозумілої користувачеві формі. Ця програма може також відображати методи у вигляді байт-коду VM.

### **Ключі**

**-b**

Ключ призначений для зворотної сумісності із *javap* з Java 1.1. Використовується в програмах, що залежать від точного формату вихідних даних дісасемблера *javap*. Ключ доступний в Java 1.2 і наступних версіях.

**-bootclasspath *шлях***

Установлює шлях пошуку системних класів. Цей ключ, використовується досить рідко й докладно вже розглянутий у розділі опису команди *javac*. Доступний в Java 1.2 і наступних версіях.

**-c**

Показувати код (на жаль, не вихідний, а байт-код VM) всіх методів, незалежно від їхнього рівня видимості, для кожного указанного класу.

**-classpath *шлях***

Визначає шлях пошуку класів, зазначених у командному рядку. Цей ключ заміщає змінну оточення CLASSPATH. У версіях Java, що передують 1.2, аргумент *шлях* визначає шлях до всіх системних класів, класів розширень і додатків. В Java 1.2 і наступних версіях він визначає шлях тільки до класів додатків. Див. також ключі *-bootclasspath* і *-extdirs*. Докладніше шлях до класів розглянутий у розділах команд *java* і *javac*.

**-extdirs *каталоги***



Цей ключ визначає один або кілька каталогів, у яких буде вироблятися пошук класів розширень. Застосовується досить рідко. Докладно розглянутий раніше в команді *javac*. Доступний у всіх версіях Java, починаючи з 1.2.

**-1**

Показувати таблиці номерів рядків і локальна змінних, якщо відповідна інформація міститься у файлах класів. Як правило, цей ключ використовується одночасно із ключем *-c*. За замовчуванням компілятор *javac* не включає інформацію про локальні змінні в створений файл класу. Див. ключ *-g* команди *javac*.

**-help**

Програма *javap* виводить довідкове повідомлення й завершує свою роботу.

**-Jjvaoption**

Параметр *jvaoption* передається безпосередньо інтерпретаторові Java.

**-package**

Із цим ключем відображаються методи, видимі тільки усередині пакета, і методи, оголошені як *protected* і *public*; методи зі специфікатором доступу *private* не виводяться. Цей ключ використовується за замовчуванням.

**-private**

Відображаються всі члени класу, у тому числі такі, що оголошені як *private*.

**-protected**

Відображаються тільки члени класу, оголошені як *protected* і *public*.

**-public**

Відображаються тільки члени класу зі специфікатором доступу *public*.

**-s**

При завданні цього ключа оголошення членів класу відображаються у форматі, котрий використовується для опису сигнатур типів і методів усередині віртуальної машини, а не у вигляді більш звичного вихідного коду Java.

**-verbose**

Включити відображення додаткової інформації. У цьому режимі виводиться додаткова інформація про кожний член зазначених класів у вигляді коментарів Java.

**-verify**

Із цим ключем програма *javap* виконує часткову верифікацію указаних класів і виводить результати на екран. Ця опція доступна тільки в Java 1.0 і Java 1.1 і вилучена в наступних версіях через незадовільну якість верифікації.

**-version**

Показати номер версії *javap*.

## **Змінні оточення**

### **CLASSPATH**

Визначає шлях пошуку класів додатків. Ключ *-classpath* заміщає значення цієї змінної. Шлях до класів докладно розглянутий у розділі *java*.

**Див. також**

*java, javac*

## **Команда *keytool***

**Реалізація:** Java 2 SDK 1.2 і вище

**Призначення:** Програма керування ключами й сертифікатами

**Синтаксис командного рядка:**

***keytool* команда ключі**

### **Опис**

Програма *keytool* управляє сховищем відкритих і секретних ключів і сертифікатів відкритих ключів (*keystore*). У ній визначена безліч команд для створення ключів, імпорту, експорту й відображення даних зі сховища ключів. У сховищі для ключів і сертифікатів визначені псевдоніми, які не чутливі до регістра, *keytool* звертається до ключа або сертифіката по псевдоніму.

Як перший параметр командного рядка повинна йти основна команда. Наступні параметри уточнюють основну команду. Потрібно вказувати тільки одну команду. Якщо їй потрібні параметри, для яких не визначені значення за замовчуванням, програма *keytool* пропонує ввести ці значення.

### **Команди**

#### ***-certreq***

Створити запит на підпис сертифіката із зазначеним псевдонімом у форматі PKCS#10 (Public-Key Cryptography Standards №10 – один із сімейства стандартів, розроблених і опублікованих RSA Data Security Inc. Визначає формат запиту про дійсність відкритого ключа). Запит записується в зазначений файл або в стандартний вихідний потік, а потім відправляється в центр сертифікації (certificate authority), що проводить автентифікацію запитуючої сторони й відсилає назад підписаний сертифікат, що підтверджує дійсність відкритого ключа. Далі підписаний сертифікат можна імпортувати в сховище ключів командою *-import* З даною командою використовуються ключі *-alias*, *-file*, *-keypass*, *-keystore*, *-sigalg*, *-storepass*, *-storetype* і *-v*.

#### ***-delete***

Видалити псевдонім із зазначеного сховища ключів. Із цією командою застосовуються наступні ключі: *-alias*, *-file*, *-keystore*, *-rfc*, *-storepass*, *-storetype* і *-v*.

#### **-export**

Записати сертифікат, пов'язаний із зазначеним псевдонімом, у файл або стандартний вихідний потік. Із цією командою використовуються наступні опції: *-alias*, *-file*, *-keystore*, *-rfc*, *-storepass*, *-storetype* і *-v*.

#### **-genkey**

Створити відкритий і секретний ключ і підписаний сертифікат X.509 (стандарт, що визначає формати даних і процедури розподілу загальних ключів за допомогою сертифікатів із цифровими підписами) для відкритого ключа. Часто ця команда використовується разом з *-certreq*, оскільки підписані сертифікати самі по собі майже не застосовуються. Із цією командою використовуються наступні ключі: *-alias*, *-dname*, *-keyalg*, *-keypass*, *-keysize*, *-keystore*, *-sigalg*, *-storepass*, *-storetype*, *-v* і *-validity*.

#### **-help**

Вивести список всіх команд *keytool* і їхніх ключів. Ця команда виконується без ключів. Тут має сенс сказати, що команда *keytool* може служити зразком того, як не потрібно робити консольні програми. Вивід короткої довідки про програму за допомогою цього ключа здійснюється не в стандартний потік виводу, а чомусь на пристрій помилок. Тому на консолі операційної системи миттєво проскакує кілька екранів інформації й користувачеві залишаються доступними тільки останні команди програми з їхніми ключами, і стандартними засобами системи (наприклад, фільтром *more*) одержати всю необхідну інформацію досить складно, іноді, навіть неможливо.

#### **-identitydb**

Записати ключі й сертифікати з бази даних ключів програми *javakey* (Java 1.1) у сховище ключів. База даних зчитується з файлу, якщо він зазначений, або зі стандартного вхідного потоку в протилежному випадку. Ключі й сертифікати записуються в зазначений файл сховища ключів. Якщо такий файл не існує, він створюється автоматично. Із цією командою застосовуються наступні ключі: *-file*, *-keystore*, *-storepass*, *-storetype* і *-v*.

#### **-import**

При зазначенні цієї команди програма читає сертифікат або ланцюжок сертифікатів PKCS#7 із указанного файлу або стандартного вхідного файлу й зберігає сертифікати в сховище ключів як надійні сертифікати із зазначеними

псевдонімами. Із цією командою застосовуються наступні ключі: *-alias*, *-file*, *-keypass*, *-keystore*, *-noprompt*, *-storepass*, *-storetype*, *-trustcacerts* і *-v*.

#### **-keyclone**

Створити копію зазначеного елемента сховища ключів і зберегти її під іншим псевдонімом. Із цією командою застосовуються ключі *-alias*, *-dest*, *-keypass*, *-keystore*, *-new*, *-storepass*, *-storetype* і *-v*.

#### **-keypasswd**

Поміняти пароль для шифрування секретного ключа, пов'язаного із зазначеним псевдонімом. Із цією командою застосовуються наступні ключі: *-alias*, *-keypass*, *-new*, *-storetype* і *-v*.

#### **-list**

Вивести в стандартний вихідний потік відбиток (fingerprint) сертифіката із зазначеним псевдонімом. Якщо зазначений ключ *-v*, то виводиться інформація про сертифікат. Із ключем *-rfc* виводиться вміст сертифіката в машинному представленні. Із цією командою застосовуються такі ключі: *-alias*, *-keystore*, *-rfc*, *-storepass*, *-storetype* і *-v*.

#### **-printcert**

Вивести вміст сертифіката із зазначеного файлу або зі стандартного вхідного потоку. На відміну від більшості команд, ця команда не використовує сховище ключів. З нею застосовуються ключі *-file* і *-v*.

#### **-selfcert**

При завданні цієї команди програма створює підписаний сертифікат для відкритого ключа із зазначеним псевдонімом. Даний сертифікат заміняє всі сертифікати або ланцюжки сертифікатів, пов'язані із цим псевдонімом. З командою застосовуються наступні ключі: *-alias*, *-dname*, *-keypass*, *-keystore*, *-sigalg*, *-storepass*, *-storetype*, *-v* і *-validity*.

#### **-storepasswd**

Поміняти пароль, що забезпечує цілісність усього сховища ключів. Мінімальна довжина пароля – 6 символів. Із цією командою застосовуються наступні ключі: *-keystore*, *-new*, *-storepass*, *-storetype* і *-v*.

### **Ключі**

Команди програми *keytool* можуть виконуватися з різними ключами, що уточнюють їхню дію. Багато ключів мають значення за замовчуванням. Якщо такі значення не визначені й не зазначені в командному рядку, програма *keytool* пропонує користувачеві ввести відповідні значення. Більш докладну інформа-

цію про ключі, їхню кількість, синтаксис, про параметри за замовчуванням можна одержати з [3].

**Див. також**

*jarsigner, javakey, policytool*

### **Команда *native2ascii***

**Реалізація:** JDK 1.1 і вище

**Призначення:** Програма перетворення коду Java в ASCII

**Синтаксис командного рядка:**

**`native2ascii [ключі] [вхідний_файл [вихідний_файл]]`**

**Опис**

Програма *javac* може обробляти файли тільки в 8-бітному кодуванні Latin-1, а будь-які інші символи в них повинні бути представлені у вигляді `\uxxxx` (формат Unicode). *native2ascii* – це нескладна програма, що перетворить вихідні файли Java, що використовують локальне кодування, у формат «Latin-1-plus-ASCII-encoded-Unicode», необхідний *javac*.

Указувати параметри *вхідний\_файл* і *вихідний\_файл* необов'язково. Якщо вони не зазначені, використовуються стандартні потоки, тому програму *native2ascii* зручно застосовувати з каналами (pipes).

### **Ключі командного рядка**

**`-encoding кодування`**

Визначає кодування вихідних файлів. Якщо цей ключ не зазначений, використовується значення системної властивості `file.encoding`.

**`-reverse`**

Пропонує програмі *native2ascii* провести конвертацію у зворотному напрямку – символи вигляду `\uxxxx` представити в локальному кодуванні.

**Див. також**

*java.io.InputStreamReader, java.io.OutputStreamWriter*

### **Команда *policytool***

**Реалізація:** Java 2 SDK 1.2 і вище

**Призначення:** Менеджер файлів політик

**Синтаксис командного рядка:**

**`policytool`**

## Опис

Програма *policytool* застосовує користувальницький інтерфейс Swing, що дає можливість легко редагувати конфігураційні файли політик. Архітектура системи безпеки Java заснована на файлах політик, у яких визначений набір дозволів для програм з різних джерел. За замовчуванням система безпеки Java визначається файлом системних політик *jre\lib\security\java.policy* і файлом користувальницьких політик *java.policy* з домашнього каталогу користувача. Редагування й зміна цих файлів системними адміністраторами й користувачами можна виконати в будь-якому текстовому редакторі, але синтаксис файлів досить складний, тому зручніше використовувати програму *policytool*. Програма має віконний інтерфейс, і тому досить проста у використанні.

### Вибір файлу політик для редагування

При старті програма *policytool* за замовчуванням відкриває файл *java.policy*, що перебуває в домашньому каталозі користувача. Щоб створити новий файл, відкрити існуючий файл або зберегти зміни, застосовуються відповідно команди New (Новий), Open (Відкрити) і Save (Зберегти) з меню File (Файл).

### Редагування файлу політик

В основному вікні програми *policytool* вміст файлу політик відображається у вигляді списку елементів. Кожний елемент цього списку визначає джерело програм і права, які даються програмам із цього джерела. Вікно містить кнопки, за допомогою яких можна додати новий елемент, редагувати або видалити існуючий елемент файлу політик.

Коли в ядро платформи Java 1.4 був уведений JAAS API (Java Authentication and Authorization Service Application Programming Interface, засоби для автентифікації й авторизації користувачів), у програму *policytool* були внесені зміни, що дозволяють визначати Principal (привілейованого користувача), якому надаються права.

З кожним файлом політик зв'язане сховище ключів, з якого витягають сертифікати, необхідні при перевірці цифрових підписів Java-програм. Як правило, використовується сховище ключів, визначене за замовчуванням, але при необхідності можна зв'язати з файлом політик інше сховище. Для цього служить команда Change Keystore (Перемінити Сховище Ключів) у меню Edit (Редагувати) головного вікна програми.

## Додавання й зміна пунктів політик

У вікні редактора політик відображається джерело програм даного пункту й список прав пов'язаних із цим джерелом. Крім того, у вікні містяться кнопки для додавання нового права, а також видалення або зміни існуючих.

Першим кроком при створенні нової політики є визначення джерела програм. Він обумовлюється URL і переліком цифрових підписів, які повинні міститися в цих програмах. Уводиться URL джерела й (необов'язково) через кому перераховуються псевдоніми сертифікатів зі сховища ключів, пов'язаного з файлом, що редагується, політик.

Після того як джерело коду визначене, вказуються права для програм із цього джерела. При додаванні або редагуванні прав застосовуються відповідно кнопки Add Permission (Додати Дозвіл) і Edit Permission (Редагувати Дозвіл). Після натискання цих кнопок з'являється вікно редактора прав.

## Визначення прав

Щоб установити право, у вікні редактора прав спочатку вибирається тип права зі списку, що випадає, Permission (дозволи). Потім вибирається відповідний об'єкт зі списку Target Name (ім'я, що призначається). Цей список автоматично змінюється залежно від обраного типу права. Для деяких типів прав, наприклад FilePermission, список об'єктів не складається, тому значення потрібно вводити вручну. Наприклад, якщо об'єктом є каталог `\tmp`, уводиться `<\tmp>`, а якщо всі файли із цього каталогу і його підкаталоги, то – `<\tmp\*>` або `<\tmp\->`. Підтримувані типи об'єктів описані в документації до відповідного класу Permission.

У залежності від типу права може знадобитися вибрати тип дії з меню Actions (Дії). Коли указані право, об'єкт і дія, натискається клавіша ОК.

**Див. також**

*jarsigner, keytool*

## Команда *serialver*

**Реалізація:** JDK 1.1 і вище

**Призначення:** Генератор версії класу

**Синтаксис командного рядка:**

**`serialver [-classpath шлях] [-show] ім'я_класу...`**

## Опис

Програма *serialver* показує номер версії класу (або класів). Номер версії використовується при серіалізації: він міняється щораз, коли змінюється формат серіалізації класу.

Якщо в класі оголошена константа `longserialVersionUID`, то виводиться її значення. Інакше програма обчислює унікальний номер версії, застосовуючи алгоритм SHA (Secure Hash Algorithm, алгоритм криптографічного хешування для вхідного повідомлення довільної довжини до 2-х ексіббайтів) до API цього класу. Основне призначення програми – обчислити унікальний номер версії класу, що потім можна помістити в клас у вигляді константи. На виході програма створює код Java, якої можна вставити в оголошення класу.

### Ключі командного рядка

#### **-classpath *шлях***

Визначає каталоги, файли JAR або ZIP, які переглядає програма (див. *java*).

#### **-show**

Якщо зазначений цей ключ, програма *serialver* відображає вікно, що містить поле для введення ім'я класу (можна ввести тільки одне ім'я) і таке, що відображає серійний ідентифікатор UID. При використанні ключа *-show* у командному рядку не можна вказувати ім'я класу.

### Змінні оточення

#### **CLASSPATH**

Оскільки програма *serialver* написана на Java, вона використовує цю змінну так само, як інтерпретатор *java*. Пошук зазначених класів виробляється з урахуванням цієї змінної.

#### **Див. також**

*java.io.ObjectStreamClass*

На завершення опису стандартних засобів розробки, хотілося б ще раз підкреслити, що з появою кожної нової версії SDK можливе виключення, зміна або додавання тих або інших ключів у наборі команд. Тому як рекомендацію до використання команд можна дати наступну пораду з обов'язкового використання ключа *-help* при роботі з програмами.



### 3 ЗАГАЛЬНІ ПРИНЦИПИ ПОБУДОВИ ПЛАТФОРМИ

Як було сказано вище (стор. 11), інструменти SDK Java являють собою набір програм, котрі в основному працюють у режимі командного рядка. Але крім цього, із самого початку, з появою мови Java, постійно робилися спроби створення інтегрованих середовищ розробки що мають GUI (Graphical User Interface – графічний інтерфейс користувача). Це такі програмні продукти як IntelliJ від JetBrains, NetBeans від Sun Microsystems, BlueJ – розробка фахівців університету Кента (Великобританія), нарешті, JBuilder від Borland і ряд інших. Останнім часом серед усього цього різноманіття IDE (Integrated Development Environment) інтегрованих середовищ розробки різко виділилася SDK Eclipse – досить корисна й високоефективна IDE для програмування мовою Java, і не тільки нею. Комплект SDK Eclipse досить надійний сам по собі, але що насправді робить його дуже потужним засобом, так це те, що він є також платформою з однозначними стандартами для розробки доповнень і невеликих додатків, які запускаються усередині Eclipse. В [12] так описується процес розробки Java-додатків за допомогою інструментів командного рядка й пояснюється необхідність використання інтегрованого середовища розробки.

«...Якщо ви читаєте цю книгу, то, швидше за все, ви програмуєте мовою Java, і досить добре знаєте, якою вередливою часом може бути Java. Пропущені оператори імпорту, не оголошені змінні, забуті крапки з комою, невірний синтаксис і неправильне приведення типів – всі ці проблеми змушують компілятор командного рядка `javac`, відкрито знущатися з вас (в оригіналі – *to cough in your face*) і виводити цілі сторінки незрозумілих повідомлень про помилки. Повідомлення про помилки свідчать про те, що `javac` знає, які помилки допущені, так чому ж він не виправляє їх і не дає вам можливості продовжити роботу?

Тому що `javac` не здатний вирішити цю проблему, він не редактор. Це робить довгі списки помилок, що прокручуються на екрані, загальним головним болем для всіх Java-розроблювачів, і викликає в них почуття, що Java занадто розбірлива в ситуаціях, які можуть розвиватися не так, як очікувалося. Щоб змінити цю практику, вам необхідно використовувати інтегроване середовище розробки (IDE), що не буде просто накопичувати помилки, до того моменту як ви спробуєте відкомпілювати програму, а буде намагатися пропонувати варіанти їхнього виправлення. Мова Java має велику потребу в гарних IDE, і, у цей час, є достатнє число таких середовищ, але IDE номер один для Java є Eclipse,

якому й присвячена дана книга...»

Але перш ніж переходити до вивчення властивостей IDE і прийомів роботи в ній, має сенс коротко ознайомитися з історією створення середовища та з її розроблювачами.

### 3.1 Знайомство з Eclipse Foundation

Сайт розроблювача [6] наводить такі дані про Eclipse: «...Eclipse is an open source community, whose projects are focused on building an open development platform comprised of extensible frameworks, tools and runtimes for building, deploying and managing software across the life cycle... The Eclipse Foundation is a not-for-profit, member supported corporation that hosts the Eclipse projects and helps cultivate both an open source community and an ecosystem of complementary products and services...».

Тобто Eclipse – це співтовариство відкритого вихідного коду, чий проект спрямований на створення відкритої платформи розробки, що складається з розширюваних структур, інструментів і бібліотек часу виконання для створення, розвитку й управління програмним забезпеченням протягом його життєвого циклу. Eclipse Foundation – некомерційне об'єднання учасників, що підтримують проекти Eclipse і допомагають розвивати як співтовариство відкритого вихідного коду, так і систему додаткових продуктів і послуг.

Проект Eclipse був спочатку створений IBM у листопаді 2001 і підтриманий консорціумом виробників програмного забезпечення. Eclipse Foundation був створений у січні 2004 як незалежна некомерційна корпорація, щоб виступати в ролі провідника співтовариства Eclipse. Незалежна некомерційна корпорація була створена для того, щоб об'єднати розроблювачів у нейтральне, відкрите й прозоре співтовариство навколо Eclipse. Сьогодні, співтовариство Eclipse складається з людей і організацій, що працюють у різних напрямках індустрії програмного забезпечення.

Далі там же на сайті говориться, що в листопаді 2001 року, консорціум Board of Stewards (eclipse.org) був сформований такими індустріальними лідерами, як Borland, IBM, MERANT, QNX Software Systems, Rational Software, Red Hat, SuSE, TogetherSoft і Webgain. До кінця 2003 року цей консорціум нараховував понад 80 учасників.

Перераховувати всіх учасників проекту в цей час немає ніякої необхідно-

сті. Але на рис. 3.1 наведена сторінка сайту, на якій представлені спонсори проекту, і один тільки список фірм перерахованих на сторінці дає уявлення про серйозність і якість робіт із проекту Eclipse.

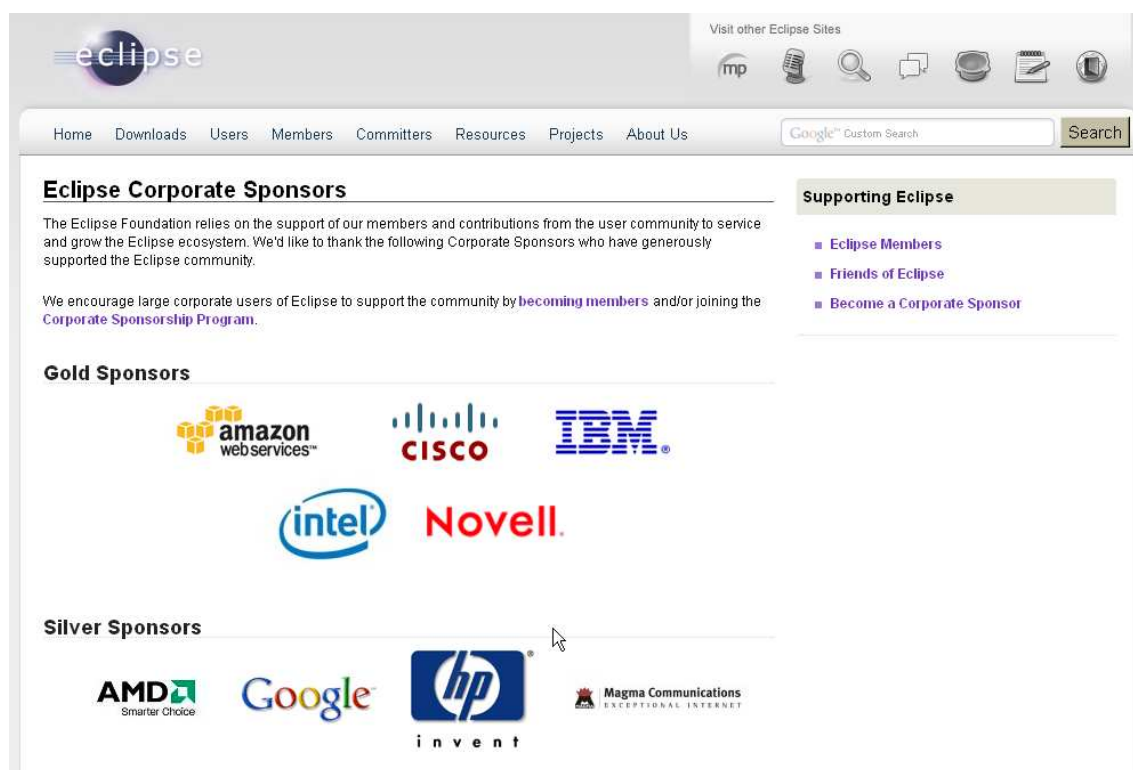


Рисунок 3.1 – Сторінка сайту Eclipse з перерахуванням спонсорів проекту

Ще з історії проекту можна вказати на те, що первісно система Eclipse була розроблена компанією Object Technology International (OTI), що згодом була придбана корпорацією IBM. Через деякий час, корпорація IBM пожертвувала технологію Eclipse (ціною приблизно 40 мільйонів доларів) співтовариству реалізації з відкритим вихідним кодом, а також мобілізувала згадані вище корпорації на спільну розробку високо інтегрованих продуктів для цієї платформи (у вигляді доповнень).

Організація Eclipse Foundation подібна організації Apache Foundation, що також надає інструментальні засоби з відкритим вихідним кодом. Але між ними є одне істотне розходження, інструменти Eclipse мають здебільш графічний характер, на відміну від інструментів Apache, які, як правило, орієнтовані на текст, наприклад сервери, такі API (Application Programming Interface, інтерфейс прикладного програмування) і інструменти як Ant (Another Neat Tool – java-утиліта для автоматизації процесу зборки програмного продукту), або

Tomcat (програма-контейнер сервлетів і така, що реалізує специфікацію сервлетів і специфікацію JSP – Java Server Pages).

### 3.2 Проекти співтовариства і архітектура платформи

Насправді, зовсім не набір спонсорів, не велика кількість учасників проекту й не славетні імена розроблювачів (Бард Канінгем – засновник Віки, співавтор карток CRC (Class, Responsibilities, Collaboration), залишив групу Patterns and Practices Team у корпорації Microsoft і приєднався до організації Eclipse Foundation, Еріх Гама – учасник перевірки середовища JUnit, а зараз один із ключових фахівців проекту JDT Project) зробили Eclipse таким популярним у середовищі програмістів на Java. Популярність Eclipse у першу чергу обумовлена архітектурою самої платформи.

#### 3.2.1 Завдання й проекти Eclipse Foundation

При створенні співтовариства організацією Eclipse Foundation перед його учасниками ставилося досить широке коло завдань, які повинні були бути вирішені в платформі Eclipse. Знання й розуміння цих завдань може допомогти зрозуміти особливості архітектури платформи та необхідність проектів, які розробляються в рамках співтовариства, а, також, багатство функціональних можливостей, котрі надаються Eclipse. Нижче наведений далеко не повний перелік цих завдань:

- підтримка конструювання різноманітних інструментів для розробки додатків;
- підтримка необмеженого ряду постачальників інструментарію, включаючи незалежних постачальників програмного забезпечення (ISV);
- підтримка інструментів у маніпулюванні довільним типом вмісту (тобто, HTML, Java, JSP, C/C++, JSP, EJB, XML, GIF, документи Word і т.і.);
- забезпечення «безшовної» інтеграції інструментів з різними типами вмісту й різних постачальників інструментарію;
- підтримка середовищ розробки додатків як із графічним інтерфейсом користувача (GUI), так і без нього;
- виконання на широкому спектрі операційних систем, включаючи

Windows і Linux;

- використання популярності мови програмування Java для написання інструментарію.

Різноманіття й кількість поставлених завдань повністю визначили й число учасників проекту, і кількість часткових проектів, які розробляються у рамках платформи. У табл. 3.1 наведений перелік невеликої кількості таких проектів [8]. Таблиця наочно демонструє стан проектів Eclipse, що перебувають у процесі реалізації або вже завершених. Тут варто звернути увагу на те, що за кожним проектом спостерігає комітет з керування проектом (Project Management Committee – PMC). Кожний проект може бути розділений на підлеглі проекти зі своїм лідером на чолі. Крім того, кожний підлеглий проект, у свою чергу, може складатися з одного або декількох компонентів.

Таблиця 3.1 – Проекти співтовариства Eclipse

Тема	Проект (проекти)
Розробка до- датків	Середовища керування життєвим циклом додатка; Інфраструктура розробки, керована моделлю; Середовище зв'язку Eclipse; Проект Buckminster Component Assembly; Платформа SDK Eclipse; Відмовостійкі інструменти розробки API (DaliORM) EJB 3.0/Java; Орієнтований на задачі UI (проект Mvlar); Інструменти паралельної розробки; Проект Business Intelligence and Reporting Tools (BIRT); Проект Voice Tools; Проект Eclipse Web Tools Platform;
Продуктив- ність і переві- рка	Проект Eclipse Test and Performance Tools Platform
Редактори	Середовище графічних редакторів (Graphical Editor Framework – GEF); Візуальний редактор (Visual Editor – VE); Проект Eclipse Web Tools Platform (HTML, JSP).

Тема	Проект (проекти)
Моделювання	Перетворювач, що породжує, моделі; Середовище графічного моделювання; Середовище моделювання EMF UML2 – реалізація UML 2.0 на базі EMF
Мови програмування	COBOL Проект AspectJ Development Tools C/C++ IDE Проект Photran (Fortran) Інструменти розроблювача Java (Java Development Tools – JDT) Проект Eclipse Web Tools Platform (Java/JSP)

### 3.2.2 Архітектура платформи

У документації до платформи, Eclipse значиться як SDK. По видимому, це пов'язане з тим, що під SDK на увазі, за звичай, мають API, а Eclipse містить API, як один з основних компонентів платформи. Але в літературі [6, 8, 10, 11] частіше зустрічається назва IDE Eclipse, тобто система програмування та налагодження тому, що це другій не менш важливий компонент системи.

Eclipse – це міжплатформна реалізація з відкритим вихідним кодом і розширюваним убудованим IDE, котра використовує мову Java. По суті, платформа Eclipse – це середовище, що надає набір служб, якими інші доповнення можуть користуватися. Кожне доповнення є розробленим для тієї ж платформи, що поєднує їх у набір високоінтегрованих інструментів. Для Eclipse доступні сотні доповнень, наприклад, [7] указує на 1077 різних доповнень і на 414627 установок доповнень безпосередньо з Eclipse!

Загалом кажучи, парадигма доповнень переводить реалізацію систем з відкритим вихідним кодом на новий рівень, насамперед, тому, що такі організації, як Eclipse Foundation, можуть не витратити своїх зусиль на рішення проблем платформи. Але співтовариство в цілому поліпшує й розширює платформу з використанням доповнень.

Таким чином, роль платформи Eclipse зводиться до забезпечення розроблювачів інструментів механізмами й правилами, використання і підтримка яких

приводить до органічної інтеграції інструментів у середовище розробки. Ці механізми представляються через чітко визначені інтерфейси, класи й методи в API середовища. Платформа також забезпечує корисні вбудовані блоки й шаблони, які полегшують розробку нових інструментів.

Схематично архітектура платформи може бути представлена так, як це показано на рис. 3.2

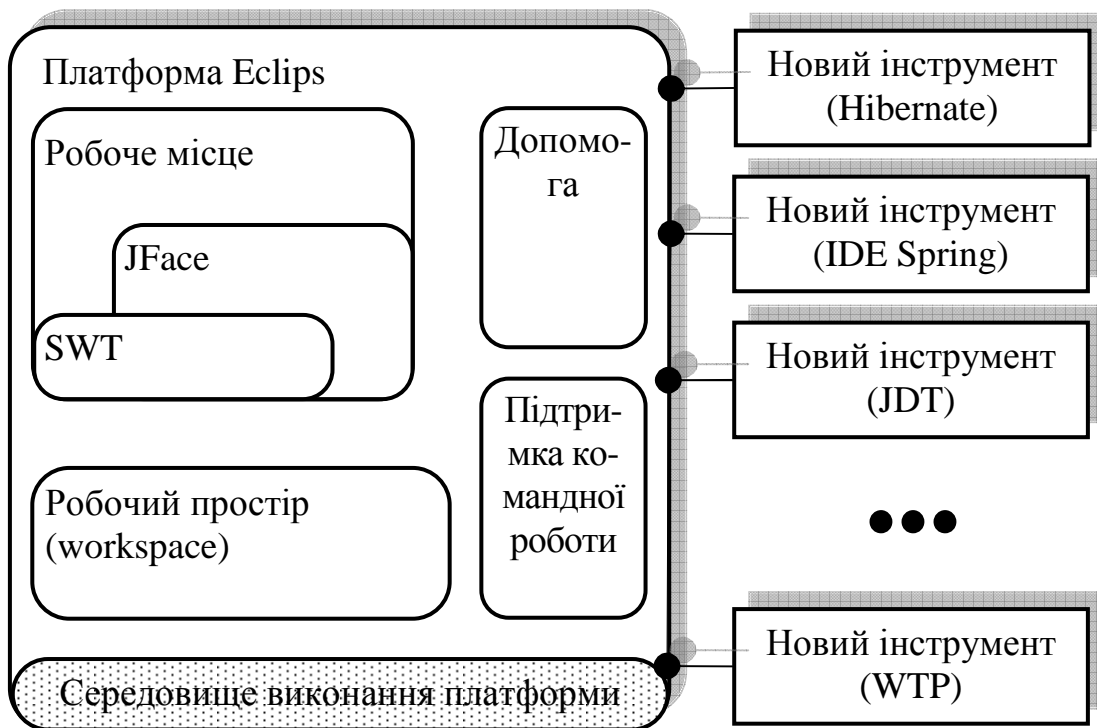


Рисунок 3.2 – Архітектура платформи Eclipse з підключеними доповненнями

За винятком невеликого ядра, що зветься середовищем виконання платформи (Platform Runtime Environment – PRE), вся функціональність Eclipse перебуває в модулях, котрі підключаються до платформи (plug-in – плагінах). Плагин – найменша одиниця функціональності Eclipse, що може бути розроблена й поставлена окремо. За звичай невеликий інструмент пишеться як один плагин, а складні інструменти розподіляються на декілька. Сама ж PRE базується й працює на віртуальній машині Java (VM), тому плагини кодуються мовою Java і підключаються через бібліотеки JAR (Java ARchive – архів Java) це запобігає складанню плагинів з декількох різних фрагментів, кожний з яких перебуває у власному каталозі. Деякі плагини взагалі не містять коду, наприклад, плагини які надають онлайнкову допомогу у формі сторінок HTML або мовні

пакели для локалізованих плагинів.

При старті Eclipse, його PDE визначає набір доступних плагинів і будує в пам'яті реєстр підключень. При цьому перевіряється правильність і можливість підключення розширень і будь-які проблеми, наприклад, підключення розширення до неіснуючої точки входу, виявляються й протоколюються. Отриманий у такий спосіб результуючий реєстр підключень плагинів доступний через API платформи. Після запуску плагини додаватися вже не можуть.

Складання реєстру ще не означає активізацію й включення плагинів у роботу. Вони активізуються тільки тоді, коли дійсно потрібно виконувати їхній код. Активізувавшись, плагини залишаються активними до закінчення роботи платформи. Причому інформація в реєстрі про той або інший плагин доступна й без активізації останнього або завантаження якого-небудь його коду. Ця властивість дозволяє підтримувати велику базу інстальованих плагинів, з яких тільки мала частка необхідна в кожному даному сеансі користувача. Поки код підключення не завантажений, він займає незначну пам'ять і не впливає на час запуску. Кожний плагин має власний підкаталог, у якому він зберігає специфічні для себе дані, цей механізм дозволяє плагинам зберігати інформацію про їхній стан між виконаннями.

Механізм плагинів застосовується й для розбивки на секції самої платформи Eclipse. І дійсно, робоче місце, робочий простір і тому подібні елементи забезпечуються окремими плагинами. Навіть PDE Eclipse має свої специфічні плагини. Такий підхід дозволяє, наприклад, у конфігураціях котрі не використовують GUI (графічний інтерфейс користувача) платформи, просто обходити плагини робочого місця та інші, робота яких залежить від інтерфейсу користувача.

Використання механізму плагинів полегшує також і самостійне написання самих додаткових інструментів, які підтримують створення нових плагинів. Прикладом такого розширення платформи може слугувати інструмент PDE (Plug-in Development Environment) тобто середовище розробки плагинів, що є включеним в SDK Eclipse.

Платформа Eclipse, з великою кількістю підключених високоінтегрованих доповнень, по суті, служить об'єднаним програмним середовищем розробки. Наприклад, існують доповнення для схем UML, програмування, налагодження, керування базами даних, перевірки модулів, керування сервером додатків, документування і багато, багато чого іншого.



Середовище Eclipse поставляється в комплекті з JDT (Java Development Tools) – інструментами розробки Java, які, безумовно, дозволяють проводити розробку додатків будь-якої складності мовою Java, як із застосуванням GUI, так і без нього. Однак Eclipse не зупиняється на Java; доступні доповнення для різних мов, включаючи HTML, C/C++, COBOL і Eiffel. Крім того, існують доповнення сторонніх виробників для інших мов, таких як PERL, PHP, Ruby on Rails...

Підбиваючи підсумок, можна сказати, що платформа Eclipse будується на механізмі виявлення, інтеграції й виконання модулів, що зветься плагинами. Постачальник інструмента пише інструмент як окремий плагин, що оперує з файлами в робочому просторі, і виставляє елементи користувацького інтерфейсу свого інструмента на робоче місце. Коли платформа завантажується, користувачеві представляється IDE, складена з ряду доступних плагинів.

## 4 ІНТЕГРОВАНЕ СЕРЕДОВИЩЕ РОЗРОБКИ

Цілком очевидно, що безпосереднє знайомство з IDE Eclipse необхідно почати з опису процесу установки системи і її першого запуску.

### 4.1 Установлення середовища розробки

Для установлення системи необхідно виконати досить «складну» послідовність дій:

- 1) зі сторінки завантаження сайту eclipse.org скачати на свій комп'ютер архів із системою, що задовольняє вимогам розроблювача;
- 2) розпакувати архів, наприклад, у каталог «Program Files» на комп'ютері або в будь-який інший, на розсуд користувача;
- 3) при бажанні, створити на робочому столі й у головному меню ОС символічні посилання (ярлики) на файл eclipse.exe...

Все! Система Eclipse повністю готова до роботи, ніяких додаткових дій з її «злому», більше того, навіть з реєстрації робити не потрібно.

Але, як уже згадувалося вище (див. стор. 65), саме середовище виконання платформи базується й працює на віртуальній машині Java. Тому, хоча сама платформа Eclipse уже готова до роботи, зовсім не обов'язково, що вона буде працювати на комп'ютері користувача. Але це пов'язане не з платформою, а з відсутністю у користувача віртуальної машини Java, або встановлена VM, версія якої не відповідає версії системи Eclipse. Щоб переконатися в тому, що машина Java встановлена і її версія відповідає версії IDE, котра застосовується, досить у командному рядку системи набрати команду **java -version**, і, якщо при цьому, не буде отримана відповідь на зразок такої:

```
java version "1.6.0_03"
```

```
Java(TM) SE Runtime Environment (build 1.6.0_03-b05)
```

```
Java HotSpot(TM) Client VM (build 1.6.0_03-b05, mixed mode),
```

то це буде означати, що середовище виконання Java (JRE – Java Runtime Environment) на комп'ютері не є встановленим. Якщо ж відповідь отримана, то обов'язково треба переконатися в тім, що встановлена версія VM має номер не нижче ніж 1.3 (для Eclipse потрібна версія 1.3 середовища Java 2 Standard Edition JRE або вища). І тільки переконавшись, що все в порядку можна запус-

тити Eclipse на виконання. У протилежному випадку доведеться скачати й установити останню доступну версію JRE з сайту компанії Oracle.

#### 4.2 Перший запуск, перше знайомство з основними поняттями й системами платформи

Запуск IDE Eclipse робиться стандартними засобами ОС. Тут необхідно врахувати те, що платформа Eclipse, як серйозна системна UNIX-сумісна програма, має цілий ряд ключів (див. додаток А), які уточнюють умови завантаження й попередньої конфігурації системи. Після запуску системи, на екрані монітора з'являється вікно завантаження (якщо його поява не скасована завданням відповідного ключа в командному рядку<sup>1</sup>), у якому відображається весь процес завантаження з підключенням відповідних плагинів (рис. 4.1).



Рисунок 4.1 – Вікно завантаження робочого простору Eclipse

Ще до повного завантаження платформи, з'являється діалогове вікно із запитом на вибір робочого простору «Select a workspace». Зовнішній вигляд вікна представлений на рис. 4.2. Тут можна або погодитися з каталогом робочого простору, котрий призначається за замовчуванням, або вказати свій власний каталог, шляхом вписування його імені в рядок вводу або за допомогою вибору по дереву каталогів після натискання кнопки «Browse...». Можна також включити прапорець «Use this as the default and do not ask again». Його встановлення запобіжить появі цього діалогового вікна при наступних запусках системи. Але, у всякому разі, для продовження роботи системи необхідно закрити вікно нати-

---

<sup>1</sup> Докладніше з ключами програми eclipse.exe можна ознайомитися у додатку А

сканням кнопки «ОК». Пізніше, якщо знадобиться, каталог, обраний для робочого простору, можна змінити вже безпосередньо під час роботи в IDE (див. стор. 88).

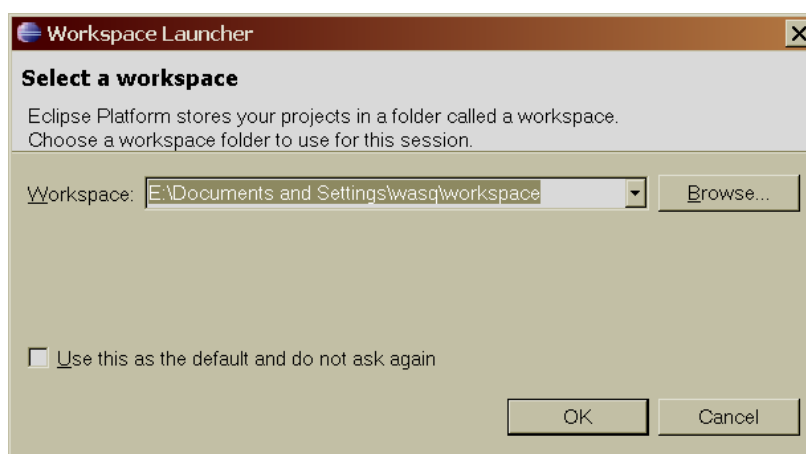


Рисунок 4.2 – Скрін-шот діалогового вікна призначення робочого простору системи Eclipse

#### 4.2.1 Поняття робочого простору

Перш ніж переходити до подальшого знайомства із системою, необхідно зрозуміти, що ж то за робочий простір було встановлено за результатом діалогу.

Коли вперше зіштовхуєшся із середовищем Eclipse, то воно, через нерозуміння концепцій, які визначають структуру й зовнішній вигляд IDE, здається трохи складним. Але, у принципі, досить усвідомити три з п'яти фундаментальних концепцій Eclipse, щоб почати працювати в середовищі й опанувати цим потужним інструментом. Одним з основних понять платформи, саме і є концепція робочого простору (див. рис. 3.2, 4.2). Але насправді, тут немає нічого складного, оскільки, у першому наближенні робочий простір (workspace) – це просто каталог для зберігання файлів проектів, котрі розробляються (наприклад, файлів Java). Якщо говорити більш точно, то це кореневий (батьківський) каталог, у якому перебувають всі файли, пов'язані із проектом (project), як вихідні коди, файли ресурсів проекту, бінарні відкомпільовані коди, так і файли SDK Eclipse, необхідні для розробки проекту.

Фактично всі інструменти, що підключаються до платформи Eclipse (див. рис. 3.2), працюють зі звичайними файлами в цьому самому робочому просторі користувача. Робочий простір на верхньому рівні складається з одного або

більш проектів, де кожний проект відображається у відповідний каталог користувача у файловій системі. Існує можливість відображати різні проекти в робочі простори в різних каталогах файлової системи й на різних дисках, хоча за замовчуванням всі проекти відображаються в «родинні» підкаталоги одного каталогу робочого простору.

Убудований до платформи механізм типу проекту (project nature) дозволяє інструментам позначати проект таким чином, щоб додати йому певну персоніфікацію або тип. *Проект* (project) – це набір файлів користувача, якими він маніпулює, наприклад файли .java, .xml і т.і. Звичайно проект позначається типом, що відповідає типу збережених файлів. Наприклад, типом Web-сайту позначається проект, що містить статичний зміст для Web-сайту, а типом Java позначається проект, що містить вихідний код для Java-програми. Крім того, Eclipse забезпечує можливість працювати з декількома проектами різних типів (наприклад, простий код Java і Web-додаток) в одному робочому просторі, причому, з кожним у конфігурації, яка є специфічною для проекту, а не в стандартній інструментальній конфігурації. Механізм типу проекту є відкритим. Плагіни можуть оголошувати нові типи проектів і забезпечувати коди для здійснення конфігурації проектів цього типу. Один проект може мати стільки типів, скільки потрібно. Це надає для інструментів спосіб спільного використання проектів, нічого не знаючи друг про друга.

Кожний проект має файли, які створює і якими маніпулює користувач. Всі файли в робочому просторі безпосередньо доступні для стандартних програм і інструментів операційної системи. Інструменти, інтегровані у платформу, забезпечуються API для роботи з ресурсами робочого простору.

Для мінімізації ризику випадкової втрати файлів, низькорівневий механізм історії робочого простору зберігає попередній зміст будь-яких файлів, які були змінені або вилучені інтегрованими інструментами. Eclipse надає можливість управляти роботою механізму історії, шляхом налаштування параметрів простору.

Крім цього, робочий простір забезпечує механізм маркерів, котрий слугує для анотування ресурсів. Маркери використовуються для запису різних зауважень, таких, як повідомлення про помилки компіляції, елементів списку того, що робиться в системі, закладок, результатів пошуку й точок зупинки при налагодженні. Інструменти можуть повідомляти про нові підтипи маркерів і управляти їхнім збереженням між виконаннями.

Такі інструменти, як компілятори або редактори зв'язків повинні виконувати координований аналіз і трансформацію тисяч окремих файлів. Платформа забезпечує каркас інкрементного будівника проекту (incremental project builder); на вхід якого подається дерево приростів ресурсів, котре відбиває розходження в ресурсі з моменту останньої побудови. Побудова дерева приростів ресурсів (resource delta), які описують ефект усього пакета операцій у термінах створення, видалення й зміни ресурсів, забезпечується загальним механізмом платформи, що дозволяє інструментам відслідковувати зміни в ресурсах робочого простору. Це робиться для того, щоб у кожному новому сеансі, коли той або інший плагин знову активізується й приєднується до процесу збереження-відновлення ресурсу, йому передається приріст ресурсів, що описує їхні відмінності від останнього збереження, у якому брав участь плагин. Це дозволяє інструментам змінити свій збережений стан для коректування й приведення у відповідність зі змінами ресурсів, зробленими за той час, поки плагин був неактивним.

На рис. 4.3 наведено дерево каталогів робочого простору з ім'ям workspace, котрій вміщує кілька проектів. Усе, що перебуває в обраному каталозі workspace, наприклад, каталог Fraction\ і каталоги всередині нього, вважається просто частиною того ж самого робочого простору.

#### 4.2.2 Вікно вітання й знайомство з підсистемою допомоги

Після завершення завантаження й проведення діалогу про робочий простір, на екрані з'являється вікно вітання, можливий вигляд якого представлений на рис. 4.4

На екрані вітання розташований ряд іконок. При наведенні курсору миші на них з'являється підказка (рис. 4.5), котра пояснює як система буде реагувати на натискання тієї або іншої іконки. Таким чином, перш ніж приступити до роботи в системі, користувачеві пропонується здійснити екскурс по ній, ознайомитися з основними поняттями, знайти потрібні для роботи шаблони і т.і. Кажучи іншими словами, екран вітання системи Eclipse – це не просте віконечко, за допомогою якого розроблювачі системи привітають користувача, а, певною мірою, є частиною загальної підсистеми допомоги, що вбудованої до платформи (див. рис. 3.2).

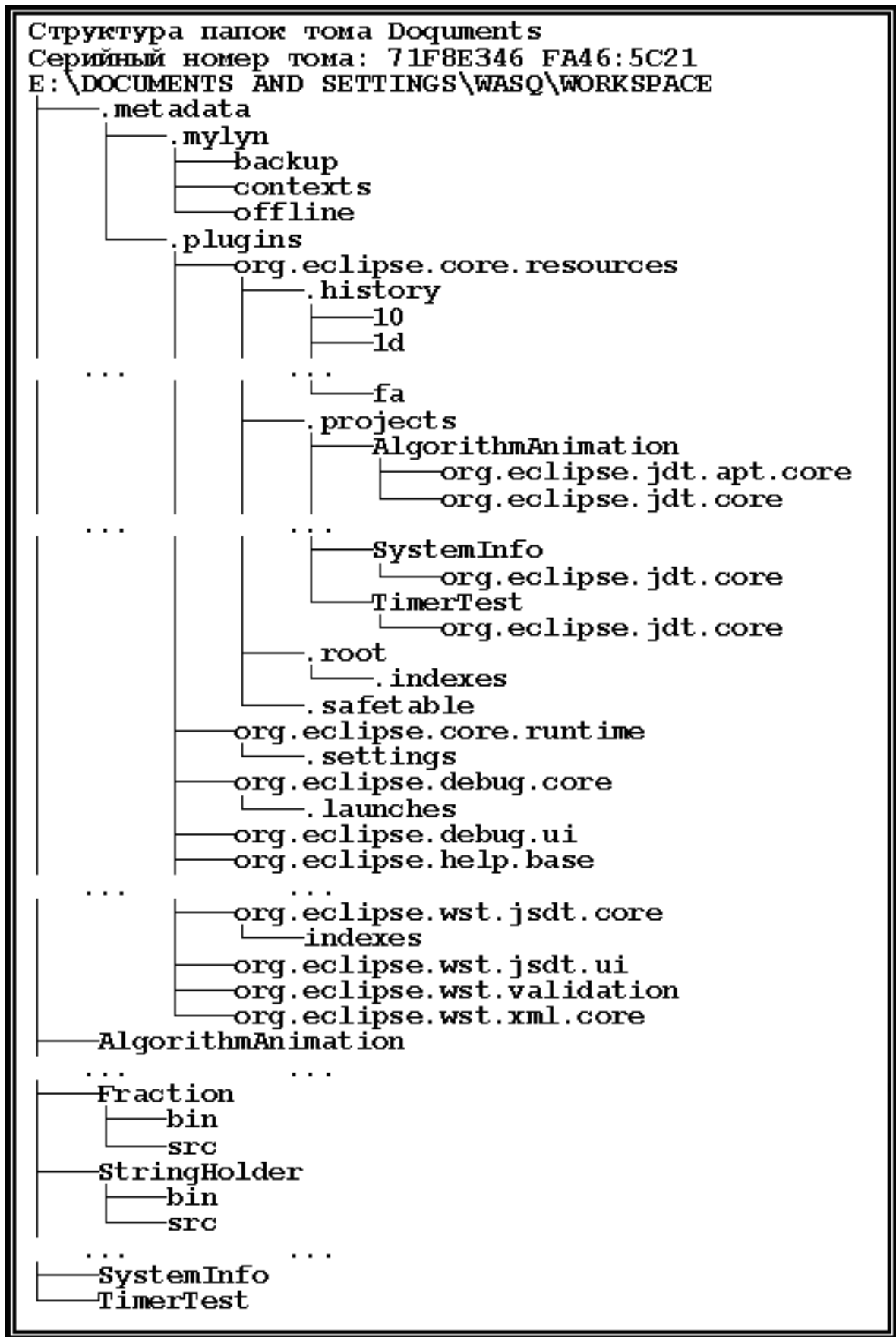


Рисунок 4.3 – Деякі гілки дерева каталогів робочого простору Eclipse

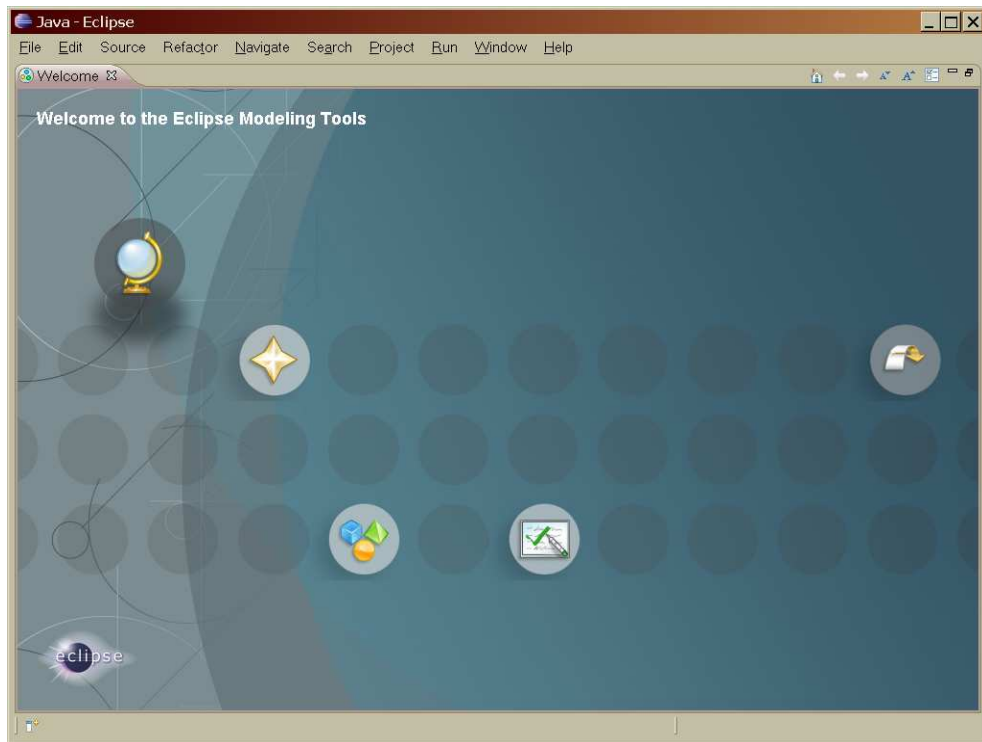


Рисунок 4.4 – Зовнішній вигляд екрана вітання Eclipse

На рис. 4.6 показаний зовнішній вигляд вікна допомоги, яке відкривається, наприклад, при натисканні на іконку огляду властивостей і концепцій SDK – «Overview Get overview of the features» (рис. 4.5а). Тут користувач може ознайомитися із цілим рядом розділів підсистеми допомоги, які дають можливість зрозуміти основні принципи функціонування Eclipse.

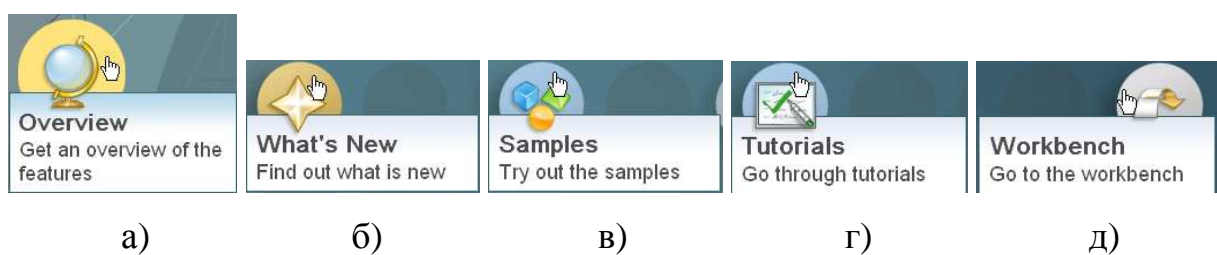


Рисунок 4.5 – Зовнішній вигляд іконок вікна вітання з підказками  
 а) – огляд властивостей і концепцій SDK, б) – знайомство зі змінами й доповненнями в версії системи що встановлена, в) – установлення додаткових шаблонів і знайомство з ними, г) – навчання програмуванню в системі, д) – перехід до робочого місця



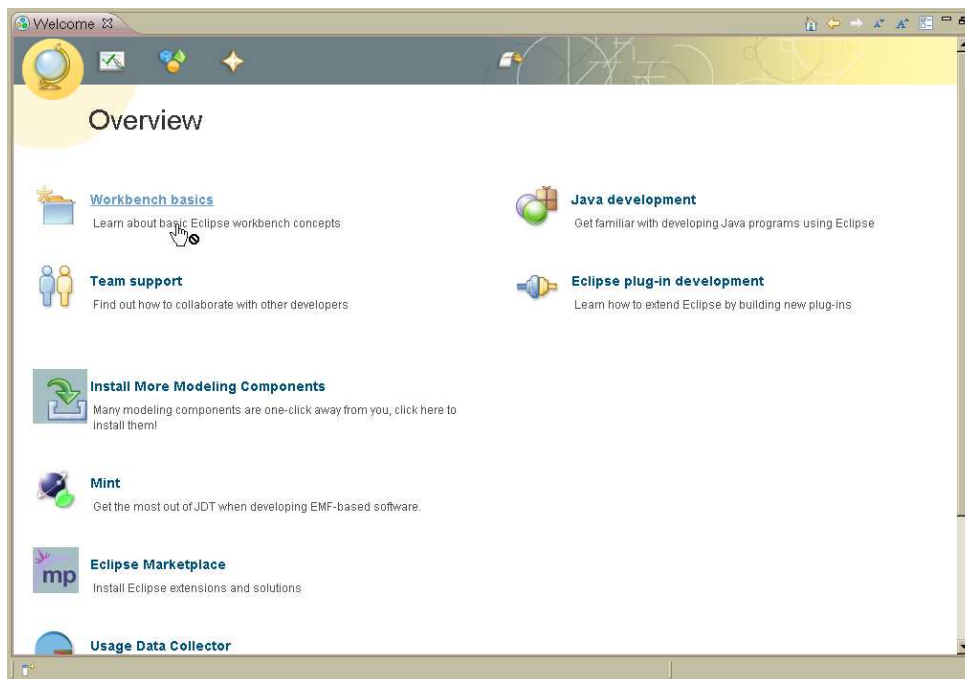








Рисунок 4.6 – Загальний вид вікна ознайомлення з системою «Overview»

Крім цього користувач може за допомогою іконок, розташованих у верхній частині сторінки допомоги й аналогічних іконкам на рис. 4.5, перейти до інших розділів знайомства з системою за допомогою кнопок   на панелі інструментів, у правому верхньому куті вікна, переходити до попередніх  або наступних  сторінок допомоги. Зовнішній вигляд екрана допомоги, характерний для вікна вітання, взагалі кажучи, є виключенням для підсистеми допомоги. Стандартом для подання розділів допомоги є документ HTML, котрий відображається у спеціальному, вбудованому до системи Eclipse переглядачі таких документів. Одне з вікон, з таким документом HTML, що переглядається, є показаним на рис. 4.7

У цьому вікні видно, що у вбудованому переглядачі відображається HTML документ. У цьому випадку потрібний документ був знайдений на локальному вузлі (IP-адреса 127.0.0.1, на рис. 4.7 ліворуч унизу), але для деяких розділів системи допомоги може знадобитися наявність підключення до Internet.

Вище вже говорилося, що на рис. 4.4 представлений тільки можливий вигляд вікна вітання. Це зауваження було зроблено тому, що за допомогою кнопки  на панелі інструментів у правому верхньому куті вікна можна викликати

діалогове вікно настроювання вікна вітання. Тут можна змінити фоновий малюнок, підключити або відключити кнопки навігації по системі допомоги і т.і.

Покинути вікно вітання можна або кликнувши по іконці «Workbench Go to the workbench» (див. рис. 4.5д), або закривши вікно, натиснувши на відповідний елемент керування – .

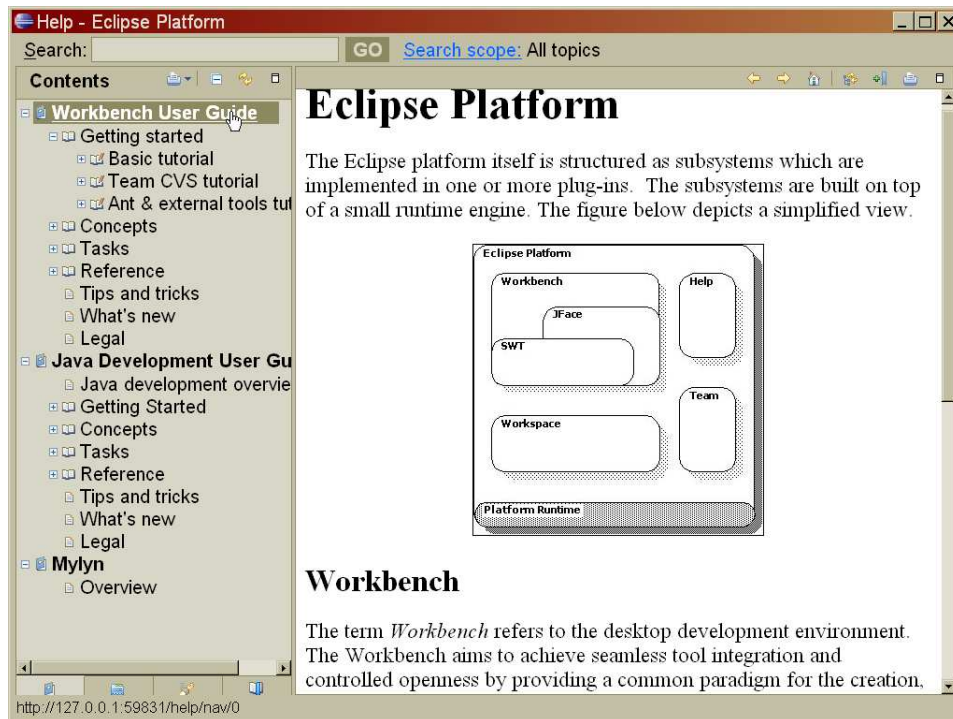


Рисунок 4.7 – HTML документ розділу допомоги в переглядачі Eclipse

### 4.3 Поняття робочого місця й інструментальні засоби користувальницького інтерфейсу

Закривши вікно вітання, користувач переходить у підсистему робочого місця (workbench) платформи Eclipse (рис. 3.2). Поняття «робоче місце» у платформі базується на моделі MVC (Model-View-Controller, «модель – представлення – поведінка» або «модель – представлення – контролер»), тобто на принципах програмного забезпечення з такою архітектурою коли модель даних додатку, користувальницький інтерфейс і керуюча логіка розділені на три окремі компоненти таким чином, що модифікація одного з компонентів впливає на інші.

Шаблон MVC [9] дозволяє розділити дані, представлення й обробку дій користувача на три окремих компоненти:

- модель (Model) – надає дані (звичайно для View), а також реагує на запи-

ти (за звичай від контролера), змінюючи свій стан;

- представлення (View) – відповідає за відображення інформації (користувальницький інтерфейс);
- поведінка (Controller) – інтерпретує дані, що були введені користувачем, і інформує модель і представлення про необхідність відповідної реакції.

В MVC принципово те, що як представлення, так і поведінка залежать від моделі. Однак модель не залежить ні від представлення, ні від поведінки. Це одне із ключових достоїнств подібного розподілу. Це дозволяє будувати модель незалежно від візуального подання, а також створювати кілька різних подань для однієї моделі.

Уперше даний шаблон проектування був запропонований для мови Smalltalk.

В Eclipse така модель забезпечує загальну структуру й подання розширюваного інтерфейсу (UI) для користувача API. Саме робоче місце і його реалізація будується на базі двох інструментальних засобів:

- SWT (Standard Widget Toolkit) – набір елементів і графічна бібліотека, інтегровані з віконною системою базової платформи, але незалежні від API операційної системи;
- JFace – інструментальний засіб UI, реалізований за допомогою SWT, що спрощує загальні завдання програмування UI.

#### 4.3.1 Призначення й властивості бібліотеки для розробки графічних інтерфейсів користувача

SWT – бібліотека з відкритим вихідним кодом для розробки графічних інтерфейсів користувача забезпечує загальний, незалежний від ОС API для елементів і графіки. Цей API реалізується таким чином, що допускає тісну інтеграцію з віконною системою базової платформи. Весь UI платформи Eclipse і інструменти, які підключені до нього, використовують SWT для подання інформації користувачеві.

Як відомо, основною проблемою в проектуванні інструментальних засобів елементів GUI є протиріччя між підходящими інструментальними засобами й інтеграцією з віконною системою базової платформи. Стандартна незалежна від платформи віконна бібліотека графічного інтерфейсу мови Java (AWT) забезпечує низькорівневі елементи, такі, як списки, текстові поля й кнопки, але

без високорівневих елементів, таких, як дерева або складний текст. Елементи AWT реалізуються безпосередньо за допомогою елементів базової платформи всіх основних операційних систем. Побудова UI з використанням тільки AWT означає цілком кросплатформне програмування для віконних систем всіх ОС. Інструментальні засоби Java Swing вирішують проблему інтеграції емуляцією таких елементів, як дерева, таблиці й складний текст. Бібліотека Swing також забезпечує емуляцію зовнішнього вигляду намагаючись зробити його таким, що б він був як у базовій віконній системі. Однак елементи, що емулюються, неминуче програють вигляду елементів базової віконної системи, і взаємодія користувача з ними, за звичаєм, досить помітно відрізняється від стандартної. Це робить побудову додатків, які були б повністю порівнянні з додатками, розробленими спеціально для віконної системи базової платформи, досить важкою.

В SWT проблема вирішена визначенням API, що є доступним для великого числа підтримуваних віконних систем. Для кожної платформної віконної системи реалізація SWT використовує платформні елементи, де це можливо; там же, де немає придатних платформних елементів, реалізація SWT забезпечує відповідну емуляцію. Загальні низькорівневі елементи, такі як списки, текстові поля й кнопки, скрізь реалізуються на базі платформи. Але ряд використовуваних високорівневих елементів можуть зажадати емуляції в деяких віконних системах. Наприклад, елемент лінійки інструментів SWT є реалізованим як платформна лінійка інструментів в Windows і як елемент, котрий емулюється для бібліотеки Motif під X Window System. Ця стратегія дозволяє SWT підтримувати цілісну програмну модель у всіх середовищах.

SWT також взаємодіє з властивостями платформного робочого столу, такими, як «drag & drop», і може використовувати такі компоненти, як елементи управління Windows ActiveX, які розроблені за допомогою компонентної моделі ОС.

Внутрішня організація Java коду SWT (на відміну від Java AWT) забезпечує роздільні й різні реалізації для кожної платформної віконної системи. Платформні бібліотеки Java для різних ОС повністю різні, і кожна зроблена з API, що є специфічним для базової віконної системи. Проте, код Java виглядає знайомим для розроблювача на платформі базової ОС. Будь-який програміст на C для Windows безумовно знайде Java-реалізацію знайомою, оскільки вона складається з викликів Windows API, які добре йому знайомі. Ця стратегія значно

спрощує втілення, налагодження й супроводження SWT, оскільки вона дозволяє виконати всю розробку, що є необхідною, повністю на Java.

#### 4.3.2 Структура й призначення набору Java-класів для побудови графічного інтерфейсу користувача

Набір Java-класів, що реалізує найбільш загальні завдання побудови GUI JFace – це інструментальний засіб UI із класами для обробки багатьох завдань програмування UI. JFace не залежить ні від API, ні від реалізації віконної системи й розроблений для роботи з SWT без приховання його.

JFace містить у собі звичайні компоненти UI – реєстри зображень і шрифтів, каркаси діалогів, настроювань і майстрів і індикатори ходу виконання для довгих операцій. Дві з його найцікавіших властивостей – дії (action) і представлення (viewer).

Дія являє собою команду, що може бути запущена користувачем через кнопку, пункт меню, або пункт панелі інструментів. Кожна дія знає свої власні ключові властивості UI (мітка, іконка, підказка, що спливає і т.і.), які використовуються для конструювання відповідних елементів, котрі й представляють дію. Цей поділ дозволяє тієї самої дії бути використаною у декількох місцях в UI. Такий підхід означає, що місце, де дія представляється в UI, можна легко змінити без необхідності зміни коду самої дії. Таким чином механізм дій дозволяє командам користувача бути визначеними незалежно від їхнього точного місцезнаходження в UI.

Представлення є сполучною ланкою для певних елементів SWT на базі моделей. Представлення здійснюють спільну поведінку елементів SWT. Стандартні представлення для списків, дерев і таблиць підтримують використання елементів із клієнтського прикладного домену й збереження їх разом зі змінами в цьому домені. Такі представлення конфігуруються за допомогою постачальників контенту й міток. Постачальник контенту знає, як відобразити вхідні елементи на майбутній уміст представлення і як використовувати зміни домену для зміни цього вмісту. Постачальник міток знає, як створити специфічну строкову мітку й іконку для відображення будь-якого елемента із прикладного домену в елемент UI. При необхідності, представлення можуть поєднуватися з фільтрами елементів і сортувальниками. Наприклад, стандартне представлення для тексту підтримує такі загальні операції, як реакція на подвійне клацання

миші, скасування проведених змін, зміна кольору тексту, навігація по індексах символів або номерах рядків і т.і. Воно забезпечує для клієнта модель документа з перетворенням його в інформацію, необхідну для елемента стилізованого тексту SWT. Багато представлень можуть бути відкриті на одній і тій же моделі документа, причому всі вони автоматично змінюються, коли модель або документ змінюється в будь-якому з них.

#### 4.3.3 Робоче місце: інструменти, компонування, редактори й представлення

На відміну від SWT і JFace, які є інструментальними засобами UI загального призначення, робоче місце забезпечує UI саме для платформи Eclipse і вводить структури, у яких інструменти взаємодіють із користувачем. Завдяки цій центральній і визначальній ролі, робоче місце є синонімом UI платформи в цілому і її головного вікна (див. рис. 4.8), яке користувач бачить після закриття вікна вітання. API робочого місця залежить від API SWT і, у меншому ступені, розширює API JFace, причому в реалізації робочого місця не використовуються стандартні бібліотеки мови Java – AWT і Swing.

Парадигма інтерфейсу користувача платформи Eclipse базується на трьох основних поняттях: редакторах, представленнях і перспективах (perspective) або компонуваннях. З погляду користувача, вікно робочого місця являє собою деякий інструментарій (workbench) і візуально складається з набору відповідних редакторів і представлень. Цей специфічний для конкретного завдання набір, називається перспективою. Компонування проявляють себе у виборі й у порядку редакторів і представлень, котрі видні на екрані. Інакше кажучи, це, по суті, платформа Eclipse, плюс деякі прості функціональні можливості – такі, як управління проектом. Фактична робота, наприклад, перегляд і редагування, здійснюються доповненнями, що підключаються до платформи, наприклад, JDT, як це було описано вище (див. п. 2.2.2) і більш докладно буде розглянуто нижче (див. розд. 4). За звичай припускається, що доповнення – це невеликі прості програми, однак в Eclipse це не так, наприклад, той же JDT або набір доповнень Eclipse Web Tools Platform (WTP).

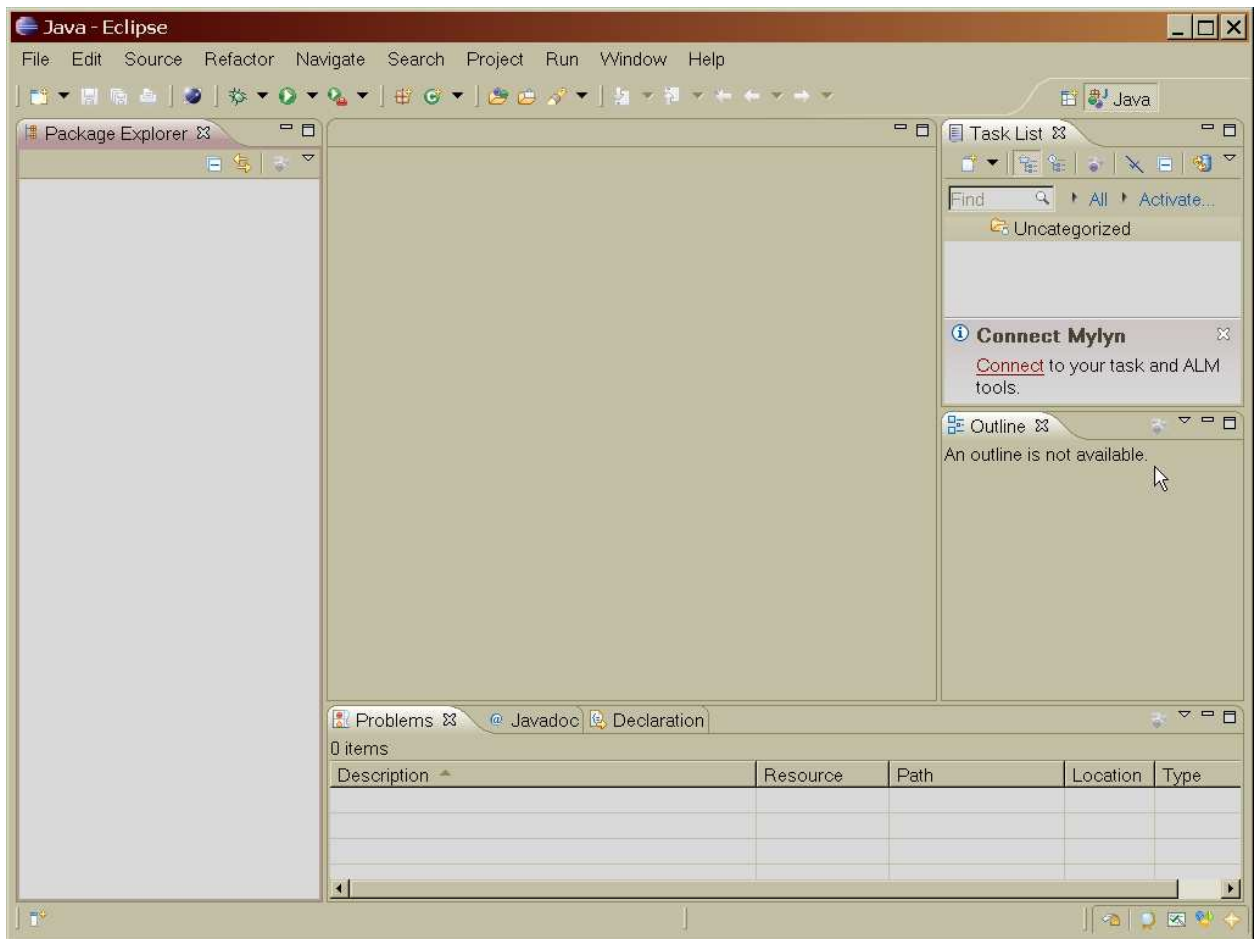


Рисунок 4.8 – Приклад зовнішнього вигляду головного вікна робочого місця

Редактори (editors) дозволяють користувачеві відкривати, редагувати й зберігати об'єкти. Їхній життєвий цикл – «відкрити-зберегти-закрити», багато в чому подібний до життєвого циклу інструментів на основі файлової системи, але вони більш тісно інтегровані в робоче місце. Коли редактор активний, він може додати функції в меню робочого місця й у лінійку інструментів. Платформа забезпечує стандартний редактор для текстових ресурсів; більш специфічні редактори поставляються в інших підключеннях. Наприклад, існують редактори для звичайного тексту, файлів .java, .jsp, .xml і ін.

Представлення (views) надають інформацію про деякий об'єкт, з яким користувач працює на робочому місці. Представлення доповнюють і допомагають редакторам у подачі інформації про документ, що редагується, надаючи доступну тільки для читання інформацію. Наприклад, представлення «Outline» (див. рис. 4.8) надає список методів файлу Java, котрий редагується у теперішній час. Однак для файлу XML тут відображаються різні елементи XML і атрибути файлу, що редагується. Представлення може доповнювати інші представлення

шляхом забезпечення інформації про обраний у цей момент об'єкт. Наприклад, стандартне представлення властивостей надає інформацію про властивості об'єкта, який може бути обраним навіть в іншому представленні. Представлення мають більш простий життєвий цикл, ніж редактори. Модифікації, що зроблені в представленні, за звичаєм зберігаються негайно, і також негайно відбиваються в інших зв'язаних частинах UI.

Платформа Eclipse, у комбінації з JDT, має пакети з різними представленнями, такими як Ant, Console, Breakpoints, Package Explorer і т.п. Інші пакети містять компонування Bookmarks, Properties, Tasks, Problems, Progress, Call Hierarchy і багато, багато інших. Кілька редакторів і представлень можуть бути згруповані разом за допомогою вкладок. Так на рис. 4.8 подібним чином організовані представлення Problems, Javadoc та Declarations із вкладками, що групують їх у правій нижній частині екрана.

Перспектива або компонування – це набір представлень і редакторів, розташованих у тім порядку, який подобається користувачеві й щонайкраще відповідає завданню, що ним розв'язується. Вікно робочого місця має кілька окремих і різних перспектив, але тільки одна з них може бути видимою в кожний конкретний момент часу. Кожне компонування має власні редактори й представлення, які можуть вишиковуватися «черепицею», каскадом або окремо, причому деякі в цей момент можуть бути сховані. У компонуванні одночасно можуть бути відкриті кілька різних типів редакторів і представлень. Перспектива управляє початковою видимістю представлення, розміщенням і видимістю дій. Користувач може швидко перемкнути перспективу на роботу з іншим завданням і може легко упорядкувати і настроїти перспективу для кращої відповідності конкретному завданню. Платформа забезпечує стандартні перспективи (Java, Java Browsing, Debug і ін.) для загальної навігації по ресурсах, онлайнної підказки й завдань підтримки командної роботи. Крім того, можна створювати й зберігати власні компонування. Додаткові перспективи поставляються в інших підключеннях.

Рис. 4.9 демонструє досить просте, але цілком придатне для вирішення найпростіших завдань компонування Eclipse, а на рис. 4.10 – є наданим стандартне компонування Debug.



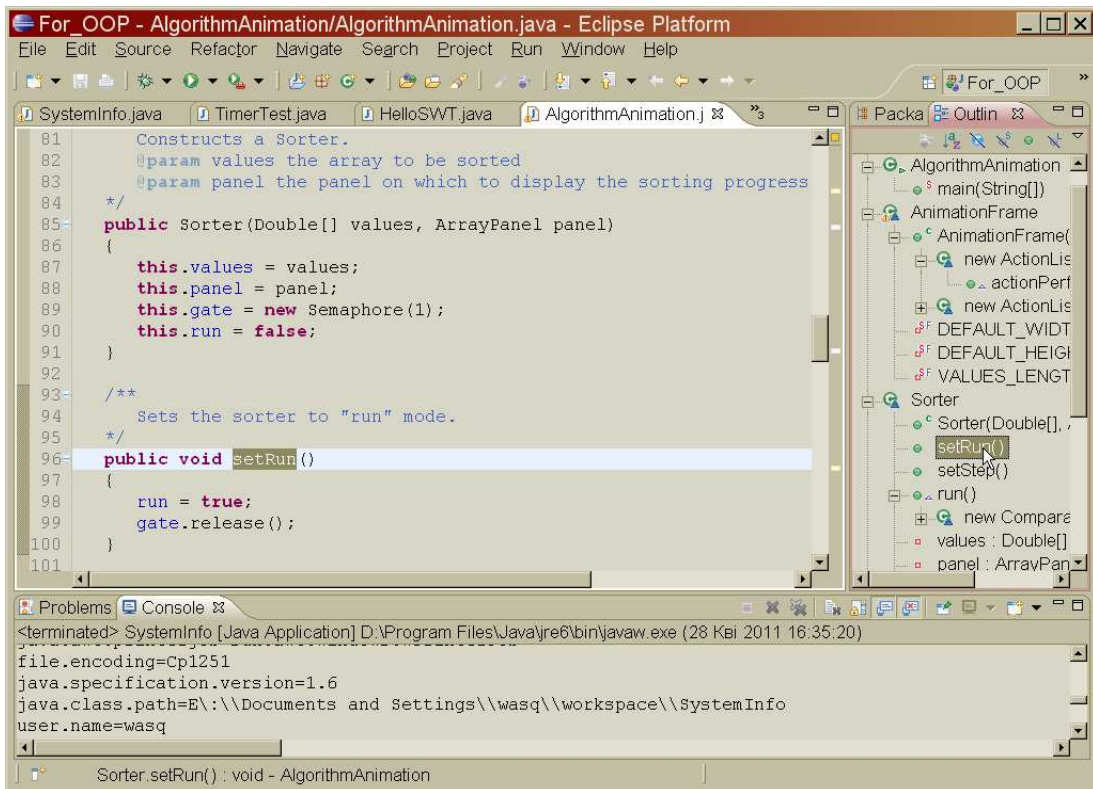


Рисунок 4.9 – Перспектива For\_OOP, котра збережена для найпростіших проектів мовою Java

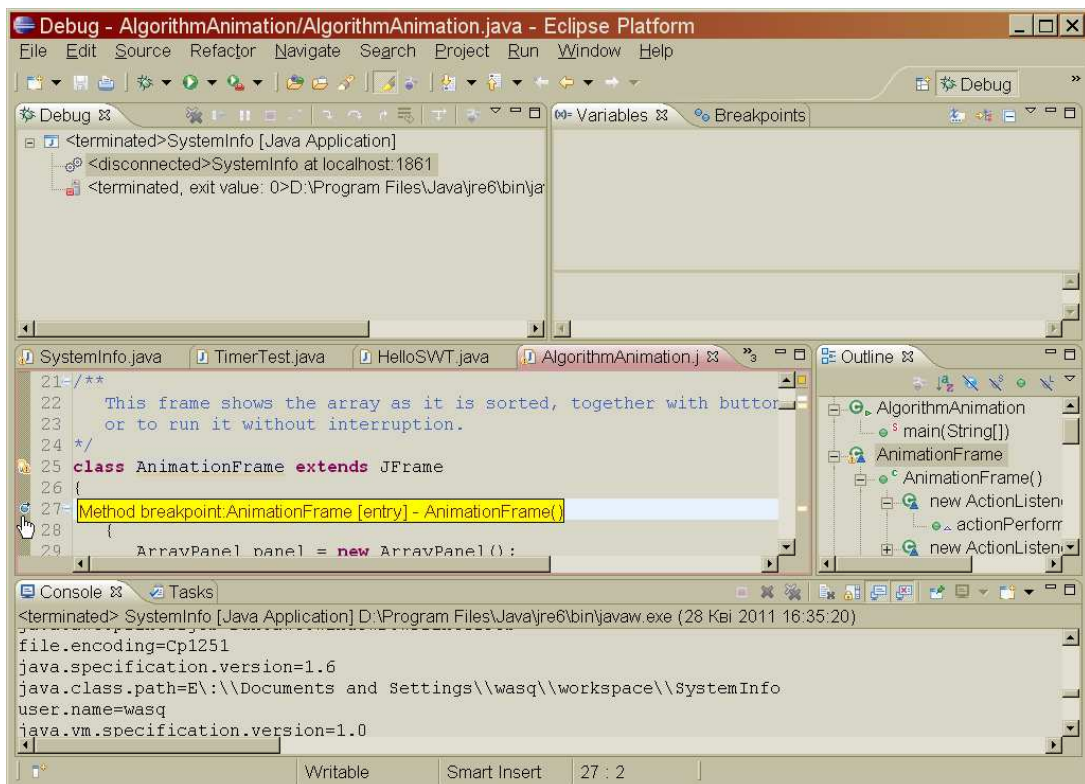


Рисунок 4.10 – Стандартне убудоване компонування Debug

Інструменти розробки, представляються доповненнями-плагином (див. п. 2.2.2) інтегруються в парадигму UI «редактори – представлення – перспективи» чітко визначеними способами. Головна точка розширення дозволяє інструментам нарощувати робоче місце:

- додавати нові типи редакторів;
- додавати нові типи представлень;
- додавати нові перспективи, які вишиковують старі й нові представлення для відповідності новим завданням користувача.

Всі стандартні редактори й представлення платформи підключені з використанням цих механізмів.

Інструменти можуть також нарощувати існуючі редактори, представлення й перспективи:

- додавати нові дії в локальне меню й панель інструментів існуючого представлення;
- додавати нові дії в меню й панель інструментів робочого місця, коли існуючий редактор стає активним;
- додавати нові дії в контекстне меню існуючого представлення або редактора;
- додавати нові представлення, набори дій і гарячі клавіші в існуючій перспективі.

#### 4.3.4 Інтеграція інструментів, що підключаються, з інтерфейсом користувача

Цілком очевидно, що інструменти, які написані на Java з використанням API платформи, сягають найвищого рівня інтеграції із платформою. Якщо ж зовнішні інструменти, що запускаються в платформі, використовують інші засоби, то в цьому крайньому випадку вони повинні відкривати власні окремі вікна, щоб взаємодіяти з користувачем, і повинні звертатися до даних користувача через файлову систему. Їхня інтеграція, отже, дуже слабка, особливо на рівні UI. У деяких середовищах, наприклад в Windows, платформа Eclipse підтримує також проміжні рівні інтеграції між цими крайніми випадками:

- робоче місце має убудовану підтримку для вбудовування будь-якого документа OLE як редактора. Ця опція забезпечує тісну інтеграцію UI;

- інструмент, що підключається, може реалізувати контейнер, котрий є сполучним елементом між API платформи Eclipse і керуючим елементом ActiveX, так що він може бути використаний у редакторі, представленні, діалозі або у майстрові. SWT забезпечує необхідну низькорівневу підтримку. Ця опція забезпечує тісну інтеграцію UI;
- інструмент, що підключається, може використовувати AWT або Swing для відкриття окремих вікон. Ця опція забезпечує слабку інтеграцію UI, але допускає тісну інтеграцію нижнього рівня UI. (Зрозуміло, що AWT і Swing повинні бути представлені в конфігурації базового середовища виконання Java.)

## 5 РОБОТА В ІНТЕГРОВАНОМУ СЕРЕДОВИЩІ РОЗРОБКИ

У комплект поставки середовища Eclipse, як його невід’ємна частина, входить інструмент JDT (Java Development Tool). Саме доповнення JDT побудоване на платформі Eclipse, для додання їй функціональних можливостей, пов’язаних з Java. JDT – це базова частина SDK Eclipse. Платформа Eclipse, будучи об’єднана з доповненням JDT, надає так багато можливостей, що досить повний і докладний його опис, навіть для «чайників», вимагає створення багатосторінкових книг, наприклад [10], не кажучи вже про книгу [14], яка містить 896 сторінок. Тому у подальшому будуть розглянуті тільки деякі із цих можливостей, в основному, питання, пов’язані зі створенням Java-проекту, написанням і редагуванням вихідного коду в редакторі й налагодженням програми засобами платформи Eclipse. Більш докладніша інформація про роботу засобів JDT міститься в інтерактивній довідковій системі Eclipse, або може бути почерпнута в тій же книзі [14].

### 5.1 Інструмент Java Development Tool, як основа розробки коду Java

При розгляді JDT можна виділити такі його основні властивості й можливості.

- Eclipse разом з JDT забезпечує керування проектами Java і такими файлами, як .java, .class, .jar, а також файлами javadoc.
- У редакторі Eclipse легко виконується редагування вихідного коду Java, з виділенням кольором ключових слів і синтаксису, інтелектуальне редагування коду, швидке виправлення помилок, імпорт організації й багато чого іншого.
- JDT Eclipse надає просто величезну кількість параметрів форматування коду, в [8] говориться про кількість більше за сотню.
- Середовище забезпечує різноманіття представлень Java. Наприклад, перспектива «Java Browsing» (див. рис. 5.1) дозволяє переглядати вміст пакетів Java, типи даних (класи, інтерфейси), їх змінні й члени. Це також досить зручний засіб перегляду файлів .jar. Так, наприклад, на зазначеному рисунку представлена структура класу JFrame з пакета javax.swing стандартного архівного бібліотечного файлу JRE – rt.jar. Різні компонування Java надають велику кількість представлень, таких як «JUnit», «Ant», «Package Explorer» і інші.

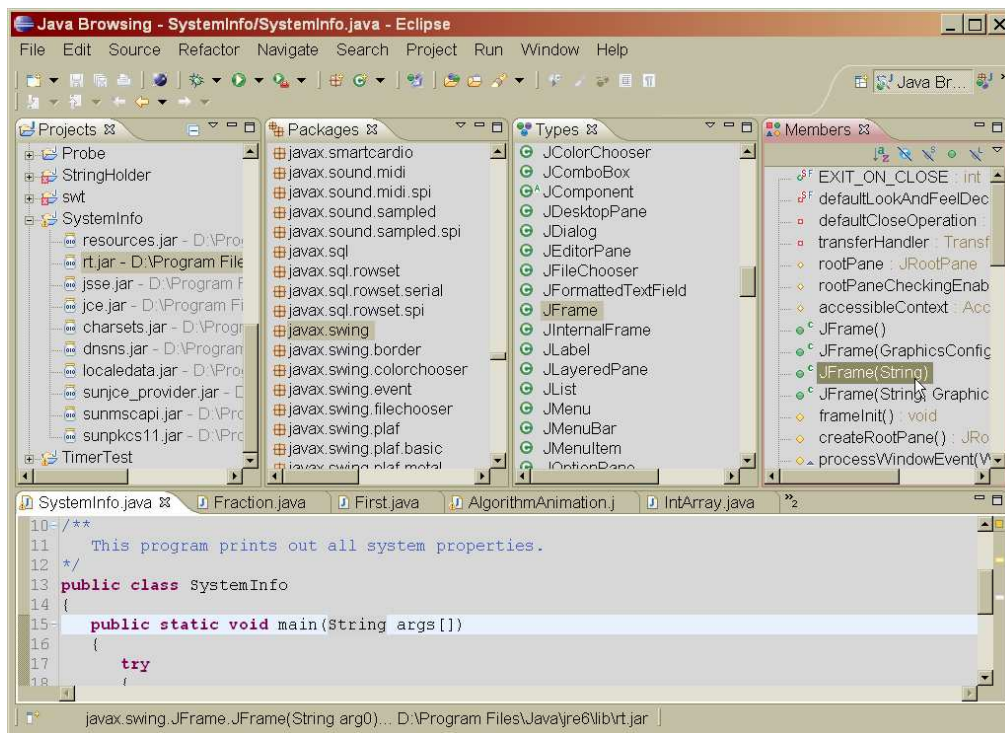


Рисунок 5.1 – Вигляд основного вікна Eclipse у компонуванні «Java Browsing»

- У середовищі коли в редакторі виконується збереження файлу .java, також провадиться і компіляція програми у файл .class (див. п. 5.3 на стор. 100).
- Eclipse підтримує такі потужні засоби роботи з вихідним кодом, як швидке «інтелектуальне» виправлення помилок і помічник умісту. При цьому помічник виконує різні дії на підставі положення курсору, тобто поводить себе інтелектуально, пропонуючи на вибір список адекватних можливостей. Використання цих засобів описано нижче (див. стор. 107).
- Є можливість створення коду з використанням упакованих або спеціальних шаблонів (для фрагментів коду), укладання коду в блок try/catch, автоматичний імпорт і багато чого іншого. Створення методів установки й одержання значень полів класу (сетерів/гетерів). Все це описано нижче.
- У системі забезпечується миттєва реакція компілятора на помилки при написанні коду, з багаторазовим наочним поданням повідомлень про попередження та помилки.
- Доповнення JDT надає всі засоби налагодження, які тільки може побажати Java-програміст. Наприклад, контрольні точки, перегляд значень змінних, відстеження виразів і багато чого іншого. Eclipse підтримує також унікальну

для налагоджувачів можливість швидкого обміну (hotswap), котра, якщо цей механізм забезпечується ще й установленим на комп'ютері JRE, дозволяє змінювати код безпосередньо у налагоджувачі й негайно перезавантажувати вірний код, без необхідності проведення нового сеансу налагодження. Більш докладно налагодження програм буде розглянута пізніше.

- Інструмент JDT забезпечує безконфліктне перейменування методів і класів, а також посилань на них у всьому проекті. При перейменуванні класу, Eclipse намагається змінити всі посилання на нього в кодї Java і файлах XML.

- Дуже тісно з можливістю перейменування в Eclipse зв'язане здійснення наймогутнішого пошуку на підставі сигнатур методів Java, що використовуються та інші можливості, котрі підтримуються JDT. Рис. 5.2 демонструє зовнішній вигляд діалогового вікна «Search». З малюнка видно, що пошук може здійснюватися у файлі, в усім кодї Java-проекту, включаючи пакети й бібліотеки класів, що є підключеними, у завданнях і навіть у плагінах. На кожній з вкладок, існують свої особливі параметри пошуку й повторного пошуку.

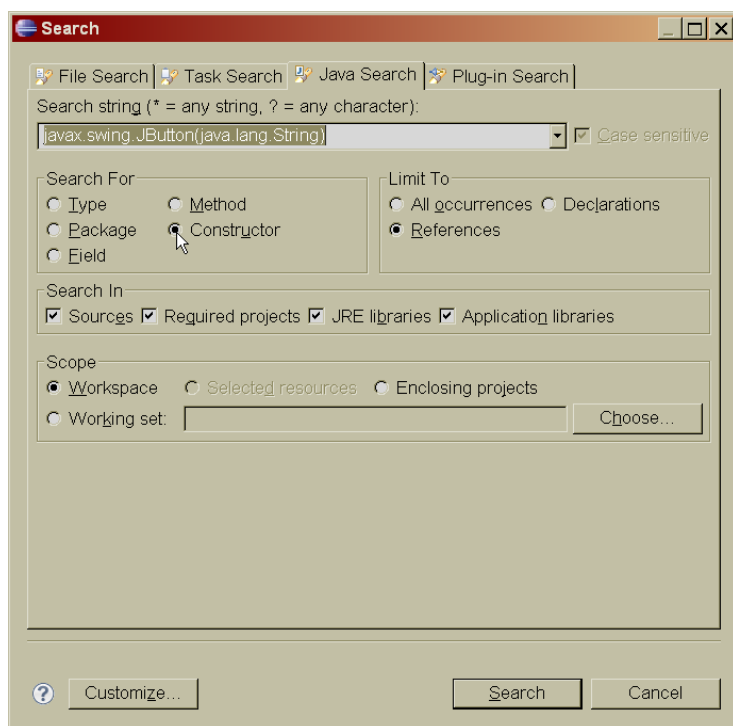


Рисунок 5.2 – Зовнішній вигляд діалогового вікна «Search» із вкладками

- У доповнення JDT є одна чудова здатність – вставляти коментарі зі словом «TODO» (можливий вибір і інших слів). Це не просто коментарі, а вказівка системі зробити так, щоб зміст коментарів з'являвся в представленні



«Tasks». Рис. 5.3 демонструє представлення «Tasks» зі списком елементів «TODO», котрі включені у різні файли вихідного коду проекту. Цілком очевидно, що такий механізм дозволяє краще організувати роботу як окремого програміста, так і цілої групи. Завдання в представленні «Tasks» можуть бути позначені як першочергові або такі, що вже є виконаними. Є можливість сортування завдань по цих ознаках і видалення завдань, котрі є вже виконаними. Це можна зробити за допомогою локального меню представлення «Tasks», вигляд якого представлений на рис. 5.9.

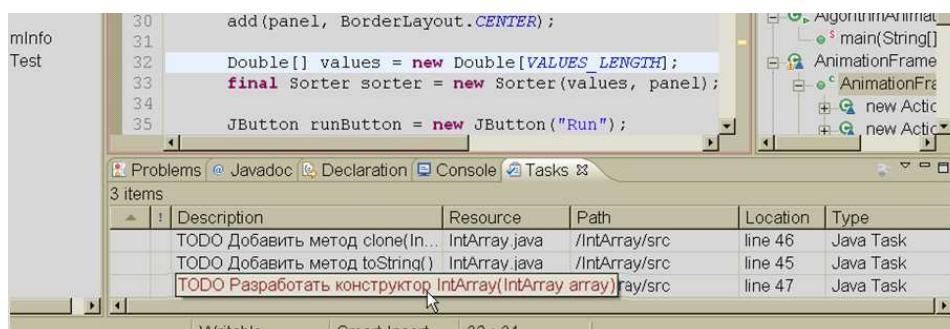


Рисунок 5.3 – Вигляд представлення «Tasks» зі списком завдань «TODO»

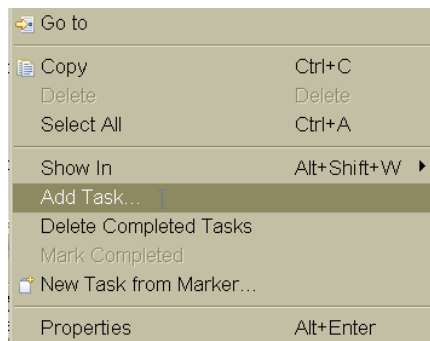


Рисунок 5.4 – Пункти локального меню представлення «Tasks»

## 5.2 Створення файлу вихідного коду

Як уже вказувалося вище (див. стор. 23 70) безпосередньо перед створенням нового проекту, якщо є така необхідність, можна змінити каталог робочого простору. Для цього в меню «File» необхідно вибрати пункт «Switch Workspace/Other...» (див. рис. 5.5). Після його вибору відкривається вікно подібне до вікна, що було показано на рис. 4.2 (див. рис. 5.6).



Рисунок 5.5 – Пункти меню «File» для зміни робочого простору

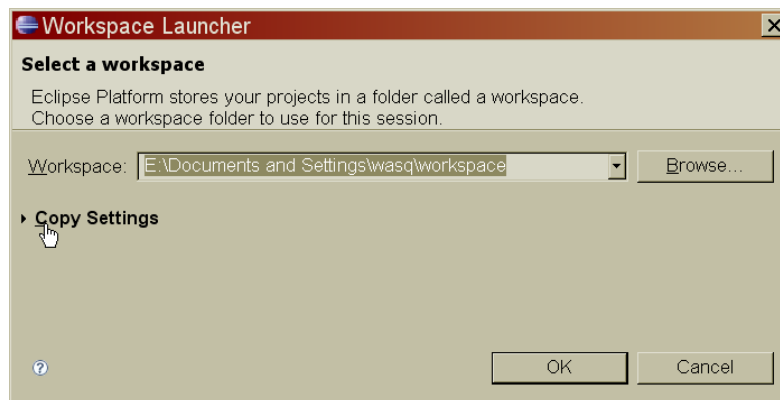


Рисунок 5.6 – Зовнішній вигляд вікна «Workspace Launcher» при зміні робочого простору

На відміну від попереднього вікна, тут передбачена можливість копіювання параметрів стандартного робочого простору у той, що створюється знову. Вибір параметрів, котрі будуть скопійовані, робиться після натискання на кнопку «Copy Settings» (див. рис. 5.7)

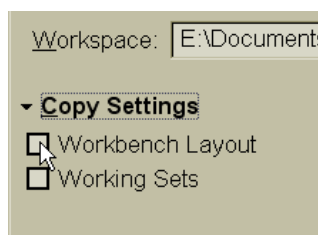


Рисунок 5.7 – Кнопки вибору параметрів робочого простору для копіювання у вікні «Workspace Launcher»

Як це є видимим з рисунка, тут можна скопіювати конфігурацію робочого місця («Workbench Layout») і набори робочих інструментів («Working Sets»).

При зміні робочого простору може статися перезапуск платформи Eclipse.

Безпосередньо сама робота над новим додатком мовою Java в IDE Eclipse починається зі створення проекту. Пізніше, коли проект уже створений, він буде автоматично відкриватися в IDE у тій перспективі, котра була останньою



при роботі з проектом. Для створення проекту необхідно в меню «File» вибрати пункт «New» і потім «Java Project» (рис. 5.8)



Рисунок 5.8 – Створення нового Java-проекту

При цьому відкривається вікно майстра створення Java-проекту «New Java Project» (див. рис 5.9а), у рядку вводу («Project name») якого необхідно вказати ім'я проекту, котрій буде створюватися. Після цього роботу з майстром можна припинити, нажавши на клавішу «Finish». Якщо ж проект вимагає завдання яких-небудь параметрів, що відрізняються від встановлених за умовчанням, роботу з майстром можна продовжити, зробивши необхідні зміни параметрів, як у цьому вікні, так і у вкладках наступного діалогового вікна, що відкривається після натискання кнопки «Next» (рис. 5.9б).

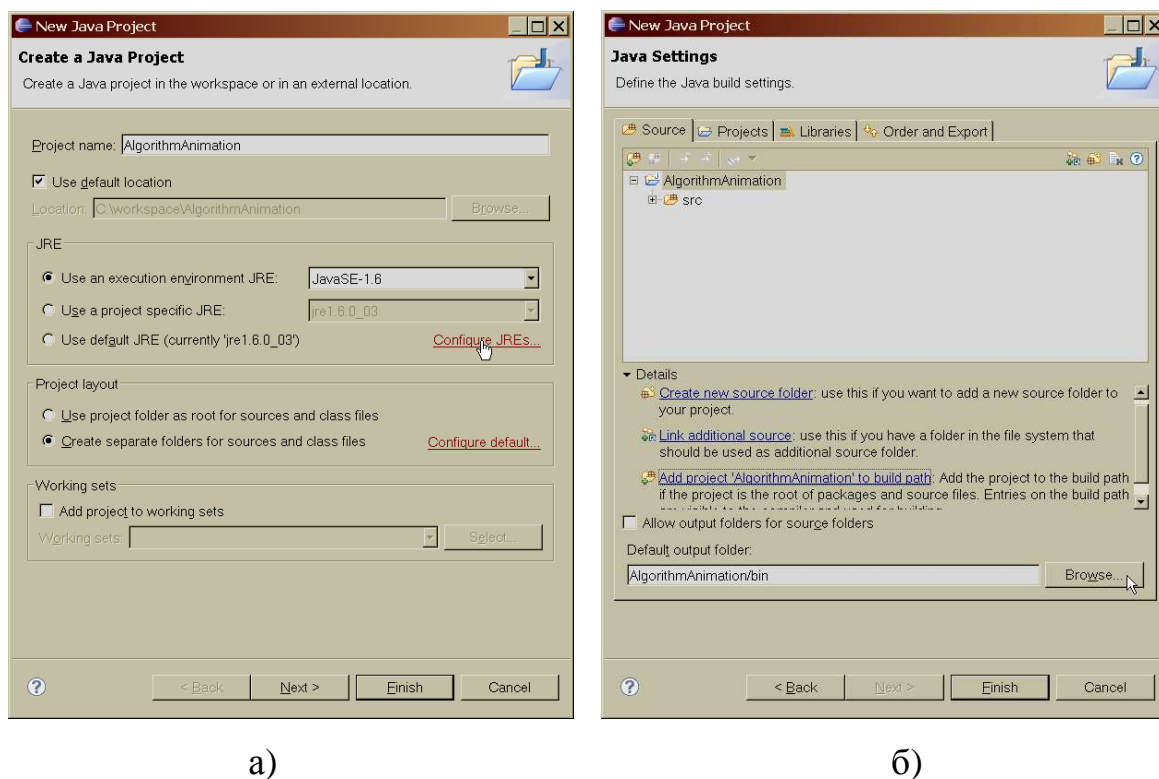


Рисунок 5.9 – Діалогові вікна майстра створення Java-проекту  
а) – початкове вікно, б) – вікно завдання параметрів середовища

При завершенні діалогу по створенню проекту, якщо все уведено вірно, у робочому просторі створюється каталог з ім'ям проекту, і це відразу ж відображається у вікні представлення «Package Explorer», що добре видно на рис. 5.10.

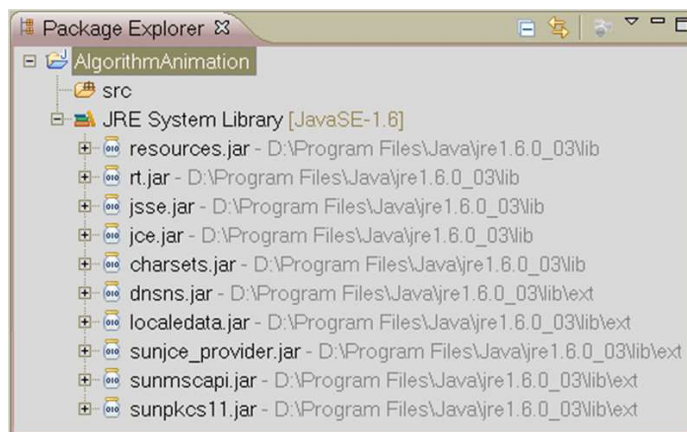


Рисунок 5.10 – Каталог з ім'ям створеного проекту у робочому просторі, відображений у представленні «Package Explorer»

На цьому рисунку видно, що крім каталогу з ім'ям проекту «AlgorithmAnimation», створений ще й підкаталог «src», у цьому каталозі повинні зберігатися файли вихідних кодів проекту, але поки що каталог є порожнім. Крім цього, поки ще порожнього, каталогу на рис. 5.10 видно ще один каталог «JRE System Library» з безліччю підкаталогів, але насправді, це не каталог, а умовна позначка середовища виконання Java з бібліотеками часу виконання.

Для створення файлу вихідного коду мовою Java необхідно скористатися новим майстром, що допоможе у виконанні цього завдання. Тут необхідно сказати, що платформа Eclipse має дуже велику кількість майстрів (так в [8] вказується на наявність приблизно 100 майстрів), які допомагають практично при будь-яких діях в Eclipse, надаючи відповідні діалогові вікна, котрі дозволяють настроїти дуже велику кількість різних параметрів.

Рис. 5.11 демонструє лише частину списку майстрів, яку можна побачити, коли вибирається пункт меню «File/New», а потім пункти «Project» або «Other», або коли використовується комбінація клавішів <Ctrl+N>, що досить добре відома по роботі практично в усіх редакторах ОС Windows. Крім створення проекту й відкриття нового файлу Java-проекту, майстри використовуються, наприклад, при конфігуруванні Web-служб, створенні EJB (Enterprise JavaBeans –

специфікація технології написання й підтримки серверних компонентів, що мають бізнес-логіку) і в багатьох, багатьох інших випадках.

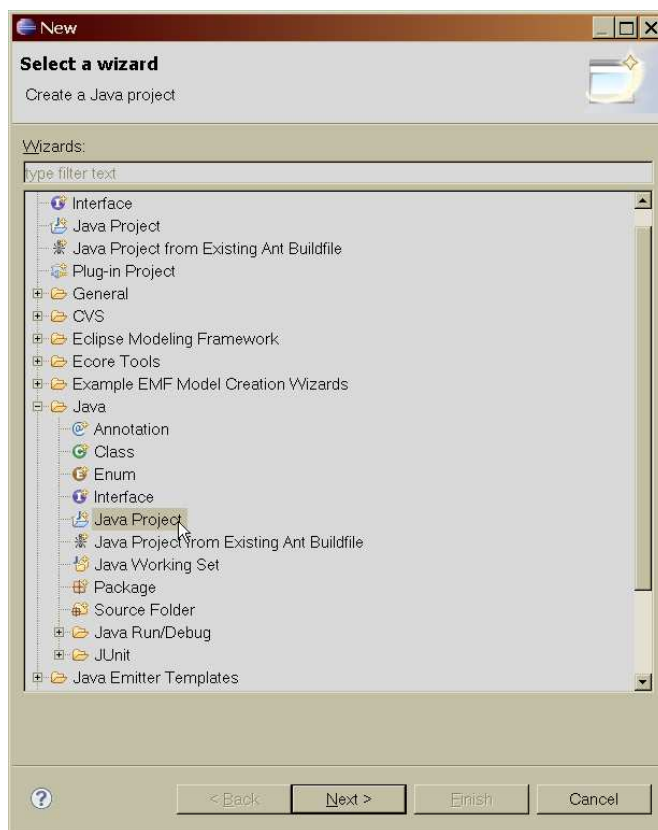


Рисунок 5.11 – Вікно вибору майстра Eclipse

Платформа Eclipse підтримує чотири види вихідного коду, які відповідають чотирьом типам у мові Java – загальним типам class і interface і спеціалізованим Enum і Annotation. Створення файлу кожного із цих типів виконується за допомогою відповідного майстра, вибір якого робиться за допомогою меню «File/New» (див. рис. 5.12).

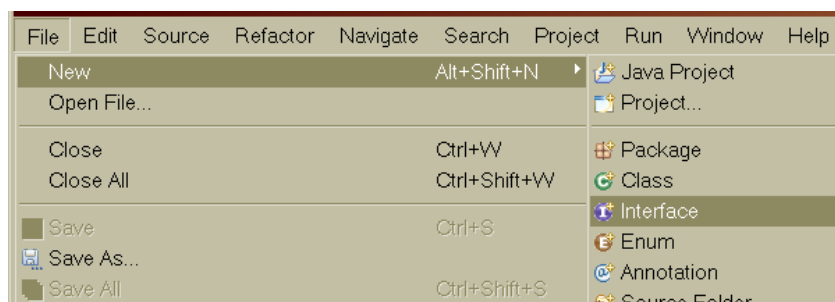
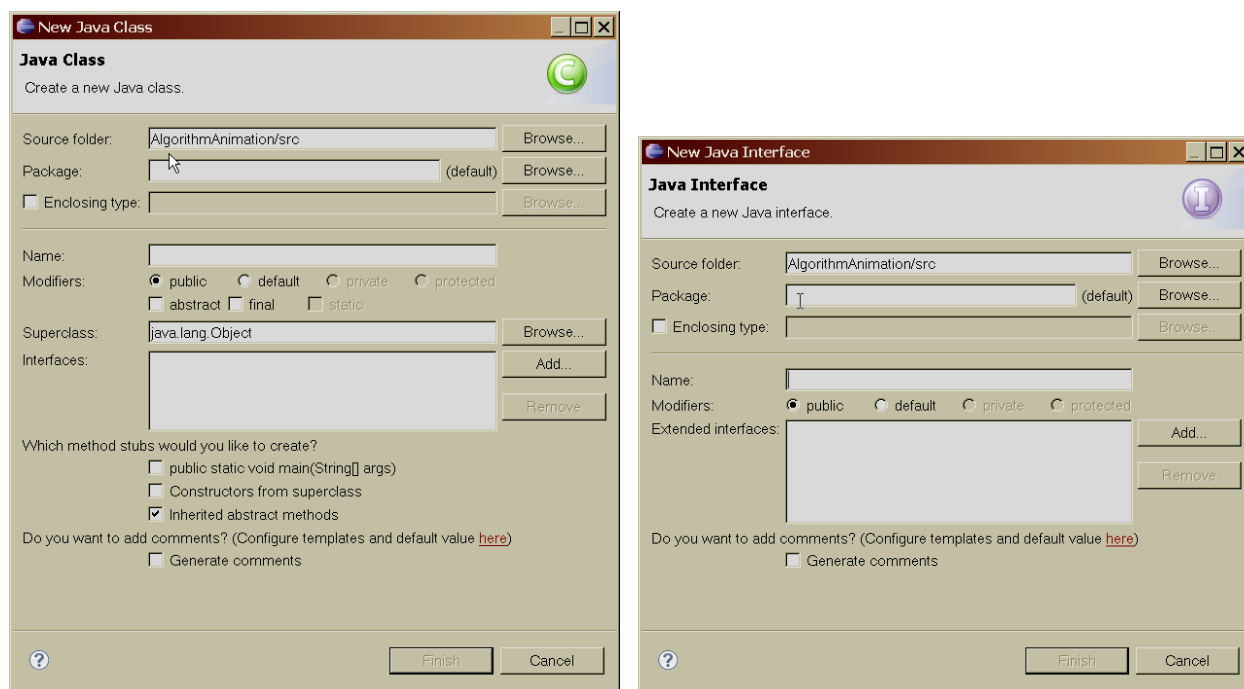


Рисунок 5.12 – Вибір майстра файлу вихідного коду в створюваному проекті

Оскільки найбільше часто доводиться зіштовхуватися з файлами, що містять вихідний код класів або інтерфейсів, то має сенс розглянути два відповідні майстри, а з майстрами файлів типів, що є перелічуваними і файлів анотацій можна ознайомитися, наприклад, в [12]. Зовнішній вигляд діалогових вікон майстрів класів «New Java Class» і інтерфейсів «New Java Interface» представлений на рис. 5.13 а) і б) відповідно.



а)


б)

Рисунок 5.13 – Зовнішній вигляд діалогових вікон майстрів створення класів – а) і інтерфейсів – б)

На цьому малюнку є видимим, що обидва майстри для початку пропонують ввести ім'я класу або інтерфейсу (активним є рядок вводу «Name»). Крім того в обох майстрах можна вказати каталог пакета в робочому просторі платформи та каталог для розміщення в ньому файлу вихідного коду (рядки вводу «Package» і «Source folder», відповідно). Обидва рядки вводу постачені кнопками «Browse» для виклику переглядача дерева каталогів робочого простору. Ще один подібний рядок вводу – «Enclosing type» за замовчуванням не активний і не є доступним для вводу доти, поки не буде встановлений відповідний прапорець. У цьому рядку можна вказати або вибрати ім'я типу, який є зовнішнім відповідно до даного. Обидва майстри надають можливість, за допомогою ра-

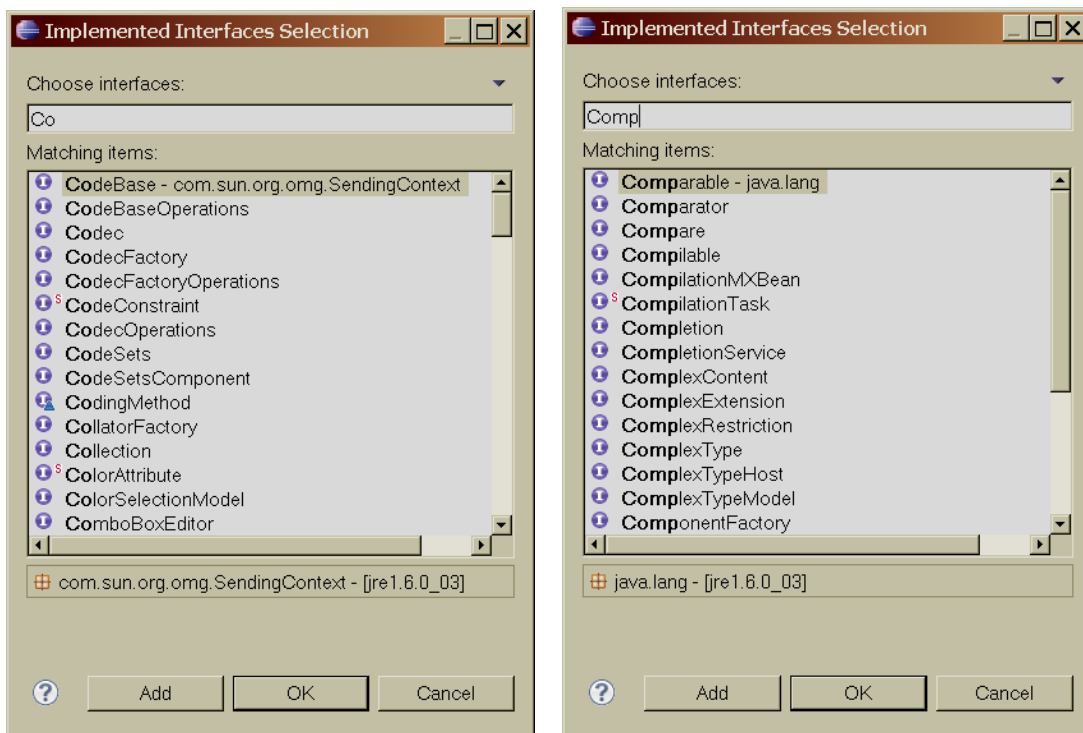
діо-кнопок у полі «Modifier», додати в оголошення класу й інтерфейсу модифікатори доступу: `public`, `default` (за замовчуванням), `private` або `protected`. Причому два останніх стають доступними тільки для внутрішніх і вкладених класів і інтерфейсів, тобто тоді, коли встановлений прапорець «Enclosing type». Крім того для класів можна задати ще й модифікатори `abstract`, `final` і `static` (останній тільки для внутрішніх класів).

Далі для класів можна вказати ім'я суперкласу (поле «Superclass» і відповідна кнопка «Browse»). Цілком очевидно, що за замовчуванням, у якості батьківського класу, пропонується клас `java.lang.Object`. У полі списку «Interfaces» у вікні майстра класів Java із кнопками «Add» і «Remove» (є доступною тільки тоді, коли обраний і доданий для реалізації хоча б один інтерфейс) можна підключити для реалізації у класі, що створюється, будь-яку необхідну кількість інтерфейсів. Для додавання нового інтерфейсу необхідно натиснути на кнопку «Add», при цьому відкриється нове діалогове вікно «Implemented Interfaces Selection».

У полі вводу «Choose interfaces» цього вікна необхідно ввести ім'я інтерфейсу, котрий буде реалізовуватися у класі. У процесі набору ім'я, майстер робить пошук інтерфейсів з підходящим ім'ям у всіх бібліотеках мови Java і виводить список тих, що знайдені, у полі «Matching items» (цей процес показаний на рис. 5.14). Т.ч., користувачеві немає необхідності повністю вводити ім'я необхідного інтерфейсу, на певному етапі його можна вибрати зі списку. Більше того, у цьому випадку можна правильно вказати потрібний інтерфейс навіть якщо два або більше інтерфейсів мають однакові імена, але перебувають у різних пакетах. Пакет інтерфейсу вказується в рядку статусу вікна, який позначається значком .

Процес вибору завершується натисканням клавіші «Add» або «OK» якщо обрані всі потрібні інтерфейси. Кнопка «OK» закриває вікно «Implemented Interfaces Selection», а натискання кнопки «Add» дозволяє продовжити вибір інтерфейсів для реалізації, причому зроблений вибір одразу ж відображається в полі «Interfaces» вікна «New Java Class» (див. рис. 5.15).

Аналогічно здійснюється введення імен інтерфейсів, що розширюються, у полі «Extended interfaces» діалогового вікна майстра створення інтерфейсів «New Java Interface».



а)

б)

Рисунок 5.14 – Різні етапи вводу ім'я інтерфейсу, що реалізується  
а) початок пошуку, б) введення може бути завершено вибором зі списку

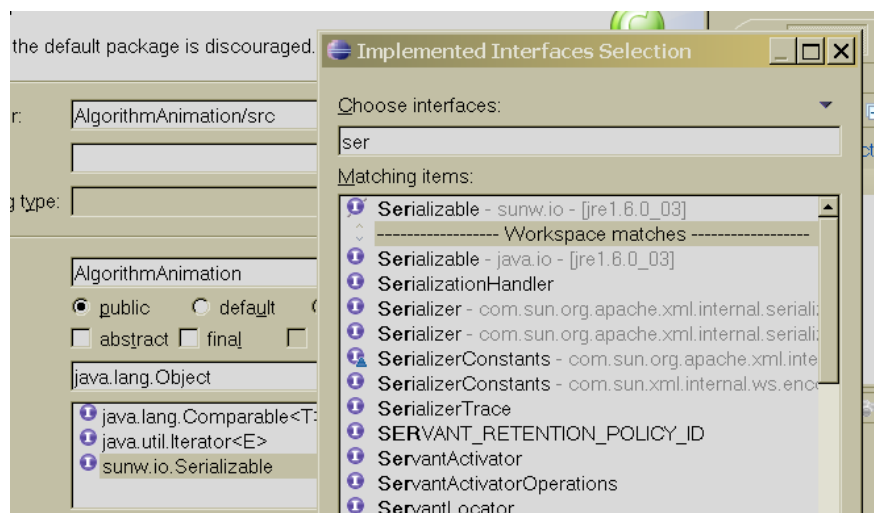




Рисунок 5.15 – Ілюстрація передачі даних з вікна «Implemented Interfaces Selection» у вікно «New Java Class»


Крім того, майстер класів дає можливість додати до вихідного коду класу, що створюється, ще й метод main (прапорець «public static void main(String[] args)»), створити заголовки конструкторів на основі конструкторів батьківсько-


го класу (прапорець «Constructors from superclass») і, нарешті, підключити до вихідного коду заголовки переобумовлених абстрактних методів (прапорець «Inherited abstract methods»).

В обох майстрах за допомогою прапорця «Generate comments» можна до коду класу додати коментарі, які надалі стануть основою для формування файлу документації до розроблювального класу. Тут необхідно відзначити, що Eclipse дозволяє настроїти коментарі, що вносяться до вихідного коду. Для цього варто звернутися до гіперпосилання «Configure templates and default value [here](#)».

При заповненні обох діалогових форм, Eclipse стежить за правильністю дій користувача й, у випадку помилкових дій сповіщає про це в заголовку вікна. На рис. 5.16 є представленими кілька помилкових ситуацій з відповідними попередженнями й повідомленнями про помилки вводу. Тут значок  сигналізує про наявність грубої помилки, а значок  говорить про те, що є помилка рівня попередження і, цілком можливо, вона ні яким чином не вплине на процес створення й на роботу класу. Наприклад, попередження на рис. 5.16в) говорить про те, що написання імені класу з маленької літери не відповідає угодам мови Java, у якої прийняте писати ім'я класу із заголовної літери. Повідомлення ж про помилку на рис. 5.16г) говорить про ситуацію присвоювання класу одночасно двох модифікаторів – abstract і final, котрі взаємно виключають один одного, а це зробити принципово неможливо.

Поки у вводі є помилки, майстри не активізують кнопку завершення «Finish» і, тим самим, не дають можливості ввести помилкові дані. Наявність попереджень не перешкоджає завершенню діалогів.

Після того, як усе є введеним вірно й натиснута кнопка «Finish», майстер починає створення файлу класу або інтерфейсу. Хід процесу створення відображається відповідним індикатором у діалоговому вікні майстра. Створення файлу можна перервати нажавши на кнопку , розташовану праворуч від індикатора (див. рис. 5.17).

Ліворуч унизу вікна майстрів знаходиться кнопка , натисканням на яку можна звернутися до системи допомоги платформи (див. рис. 5.18).



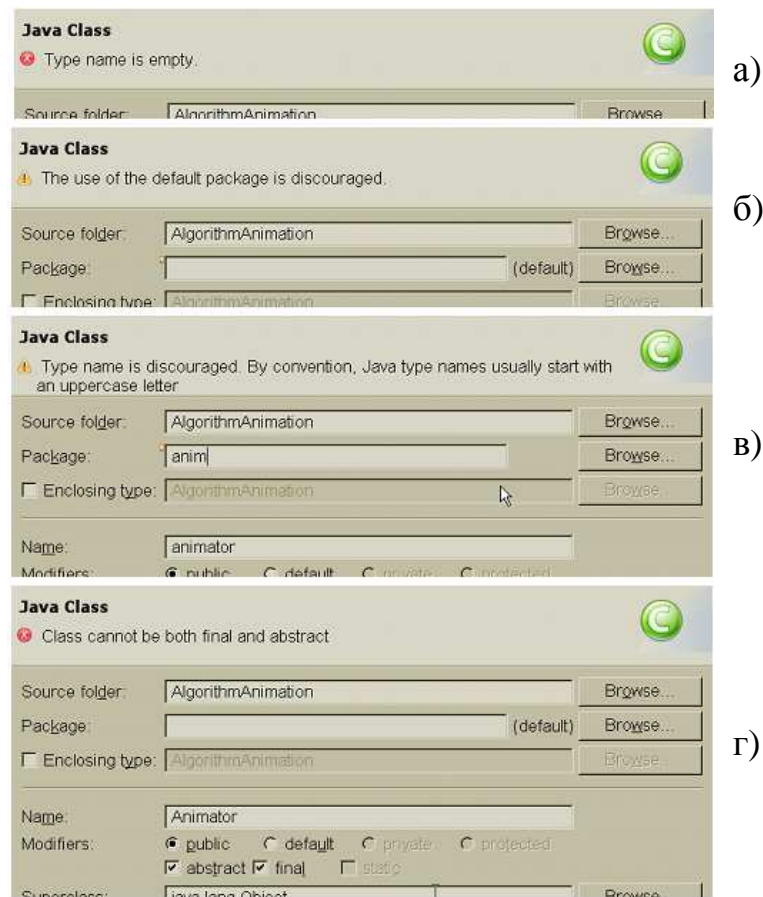


Рисунок 5.16 – Приклади помилкового вводу в майстру створення класів  
 а) не уведене ім'я класу, б) небажане використання пакета за замовчуванням, в) ім'я класу починається з малої літери, г) використані несумісні модифікатори класу

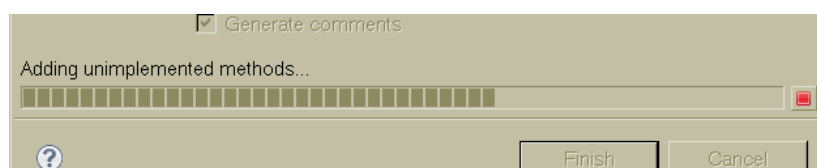




Рисунок 5.17 – Індикатор процесу створення класу

Вікно допомоги, що з'являється при цьому, є досить малим (рис. 5.18), і користуватися ним не дуже зручно. Тому в панелі інструментів віконця допомоги –  передбачена кнопка ( див. рис. 5.19) показу розділів системи допомоги в окремому вікні звичного стандартного розміру. Таке окреме розкрите вікно допомоги показане на рис. 5.20.



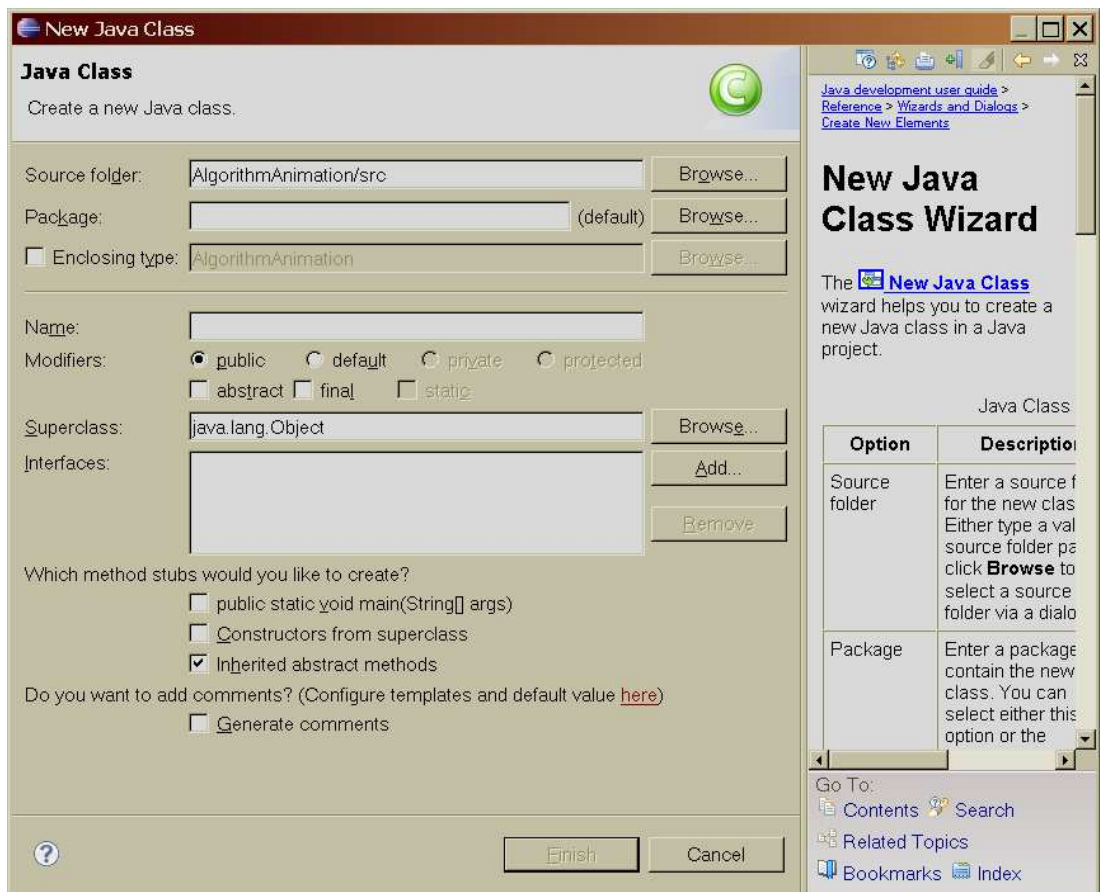


Рисунок 5.18 – Діалогове вікно майстра з відкритим вікном розділа системи допомоги

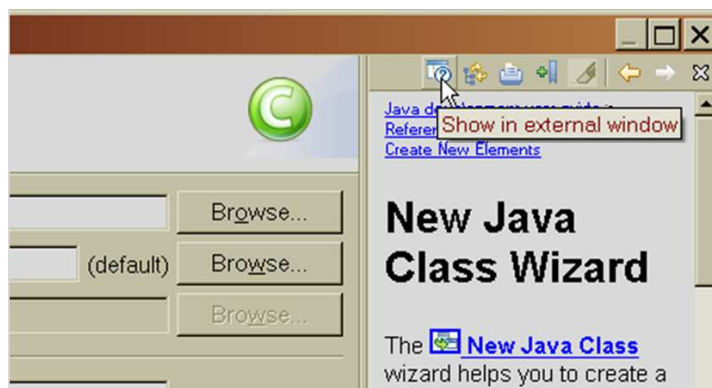


Рисунок 5.19 – Кнопка розкриття вікна допомоги

Властиво після натискання кнопки «Finish», робота зі створення файлу вихідного коду завершується й можна приступати до процесу редагування й наповнення коду бізнес-логікою роботи.

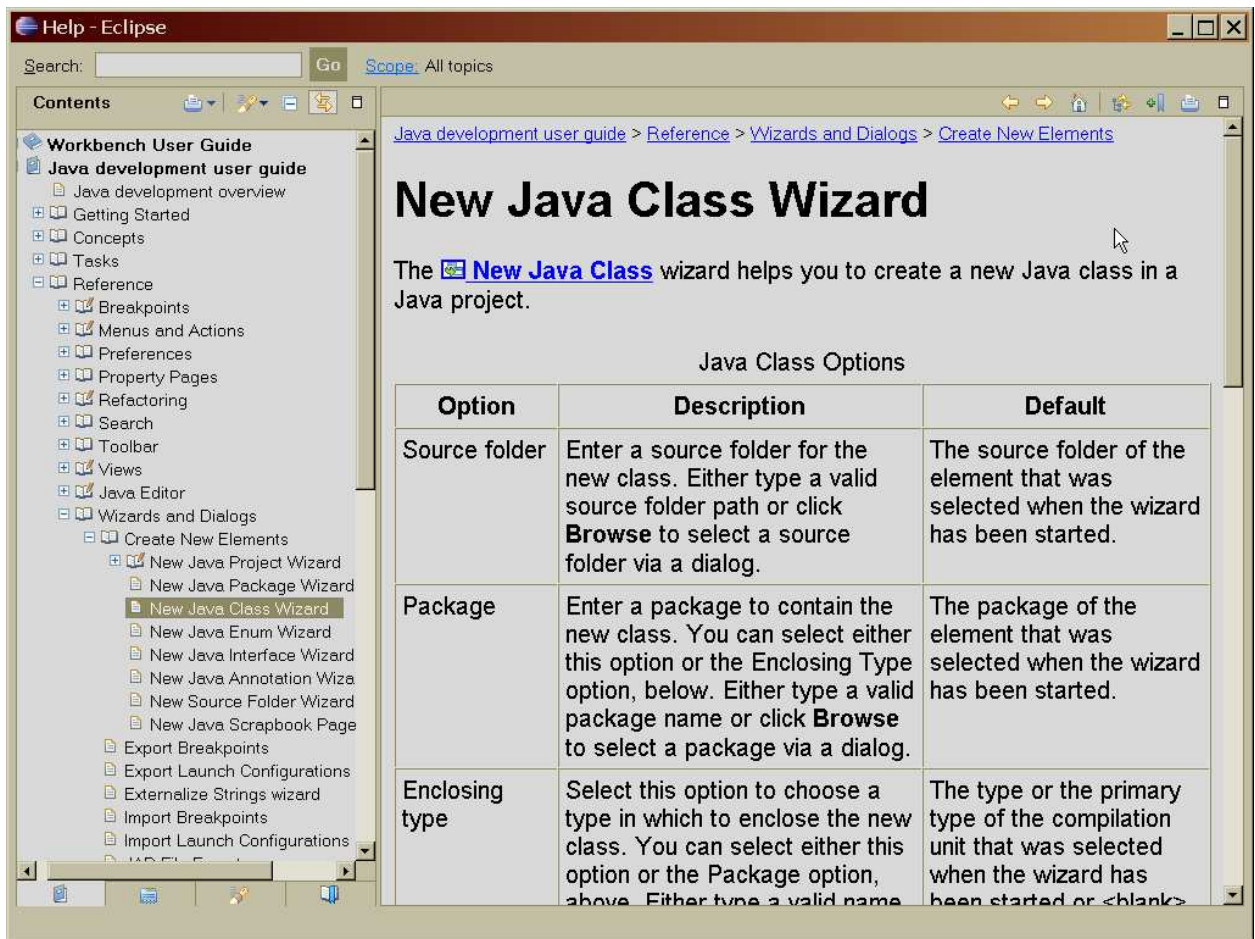


Рисунок 5.20 – Відокремлене, повністю розкрите вікно системи допомоги Eclipse

### 5.3 Компіляція програми в середовищі

Як видно зі сказаного вище (п. 5.2) різні майстри являють собою дуже потужні й зручні засоби інструмента JDT системи Eclipse. Але основні достоїнства цього інструмента проявляються в іншому. А саме у процесі компіляції програми й у можливостях вбудованого редактора коду Java-програми.

На рис. 5.21 у компонуванні Java представлений тільки що створений клас AlgorithmAnimation. Тут дуже добре видно, що, незважаючи на коротке «життя» програми, Eclipse уже знайшов у ній велику кількість (8 штук) помилок, синтаксичного та семантичного характеру. Це пов'язане з тим, що Eclipse поставляється з вбудованим компілятором, котрий призначений для компіляції файлів .java у файли .class, при кожному збереженні вихідних файлів. Це, можливо, одне з найбільш важливих, хоча й найменш рекламованих переваг Eclipse.

Такий підхід повністю рятує користувача від необхідності використовувати звичну методику проб і помилок, при якій:

- пишеться деякий вихідний код;
- код компілюється;
- виправляються помилки компіляції;
- компіляція повторюється;
- усуваються наступні помилки компіляції;
- повторюється компіляція;
- результат запускається на виконання;
- знаходяться помилки часу виконання програми;
- усе повторюється спочатку.

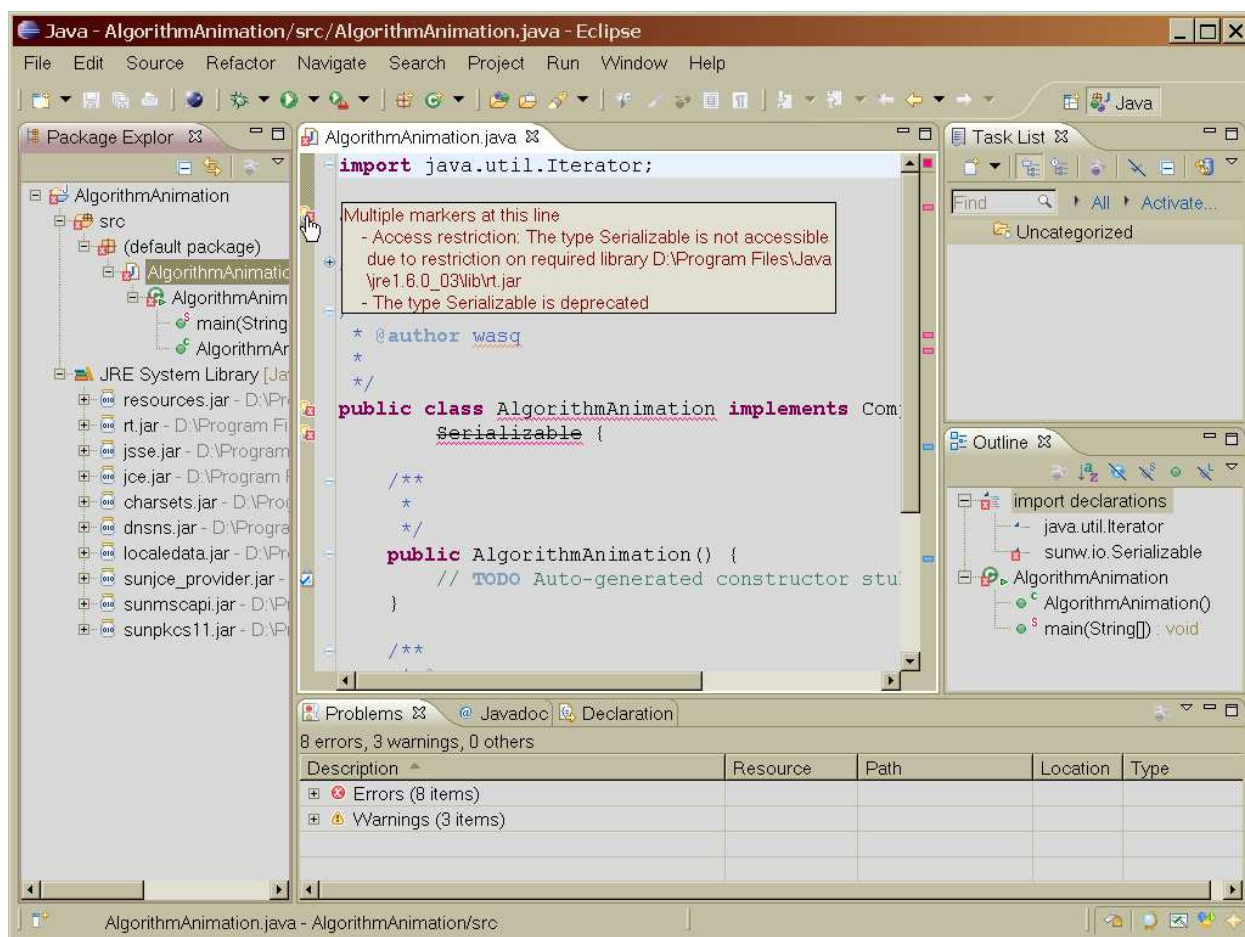



Рисунок 5.21 – Тільки що створений клас `AlgorithmAnimation` у редакторі Java










При роботі ж в Eclipse, необхідно відзначити той факт, що у користувача створюється враження практично миттєвого виконання компіляції. Але це про-

сто зовнішнє враження. Компілятор мови Java, убудований до Eclipse, виконується окремим потоком у фоновому режимі й постійно відслідковує будь-які зміни вихідного коду. Тому до закінчення роботи над файлом уже практично готова його відкомпільована версія (якщо, звичайно немає синтаксичних помилок, попередження допускаються), і навіть немає необхідності окремо здійснювати компіляцію програми. При збереженні відредагованого файлу .java (вибір пункту меню «File/Save», натискання <Ctrl-S> або кнопки ) одночасно зберігається й файл .class.

## 5.4 Редагування вихідного Java коду в редакторі

Незважаючи на велику кількість помилок і попереджень, які обумовлюються поки що лише відсутністю бізнес-коду й не дуже вірним вибором інтерфейсів (застарілі типи), файл вже є створеним, і з ним можна працювати далі. Але спочатку потрібно розглянути як компілятор Eclipse, за допомогою редактора й інших представлень компонування Java, відображає різну інформацію про помилки.

### 5.4.1 Відображення помилок у редакторі й інших представленнях

На рис. 5.21 видно, що в системі Eclipse, з метою полегшення створення програмного забезпечення, особлива увага приділена питанням нагадування розроблювачу про помилки вихідного коду. По-перше, повідомлення про всі помилки й попередження заносяться в спеціальне представлення «Problems» (рис. 5.22). Помилки тут позначаються символом , а попередження – . Тут же дається короткий опис помилки (попередження), ім'я ресурсу, у якому вона виникла, вказується шлях до файлу, що містить помилку, конкретний номер рядка з помилкою, тип помилки. По-друге, весь проект у представленнях «Project Explorer» і «Package Explorer» (рис. 5.23а й б, відповідно) позначається як такий, що містить помилки. Причому, як це видно з рисунка, безпосередньо такими, що містять помилки, позначаються: проект – символом , вихідний каталог проекту – , пакет, до якого входить файл, – , файл Java-програми – , вказується клас, у якому допущена помилка –  і, нарешті, метод (методи) з помилкою – . У всіх цих піктограмах у лівому нижньому куті присутній значок , що вказує на наявність саме помилок у вихідному коді. Якщо поми-



лок у кодї немає, а є тільки попередження, то в цьому випадку піктограма має ознаку 🏠, тобто значки для елементів з попередженнями мають вигляд: 🏠, 🏠, 🏠, 🏠, 🏠 і 🏠. Піктограми помилок мають пріоритет щодо значків попереджень, тобто якщо в проєкті є як помилки, так і попередження, то проєкт позначається тільки значками помилок.

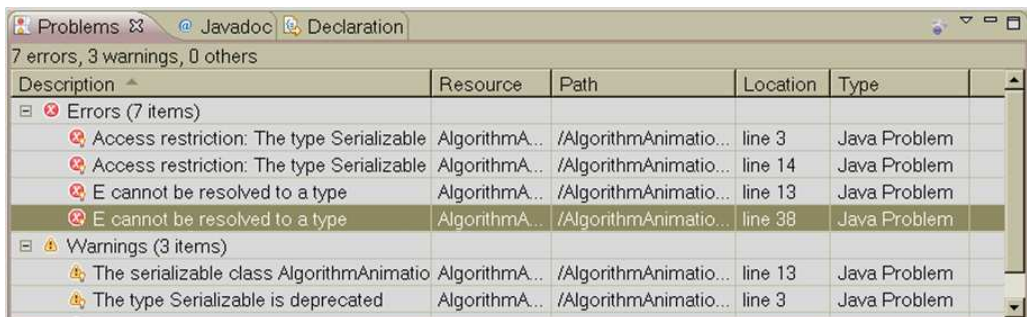


Рисунок 5.22 – Відображення повідомлень про помилки й попередження в представленні «Problems»

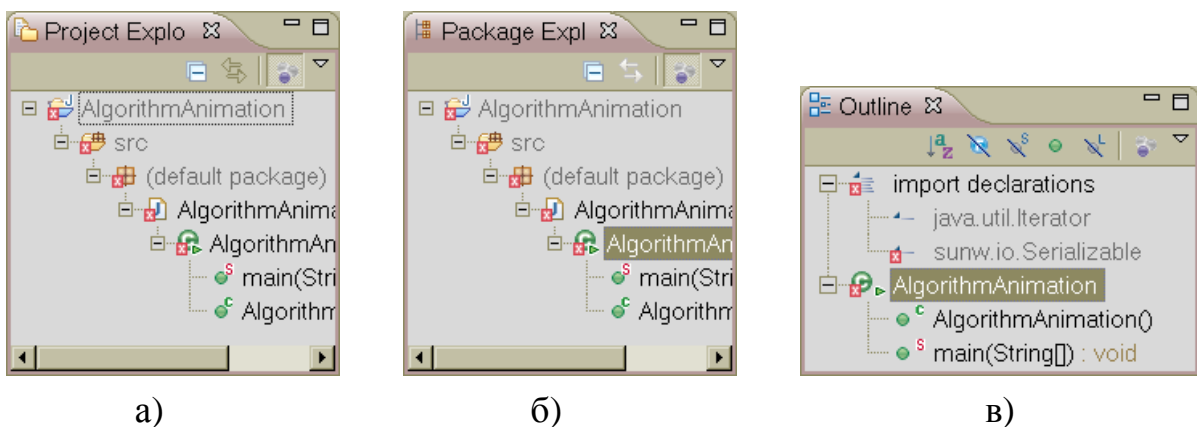







Рисунок 5.23 – Відображення помилок у представленнях перспективи «Java»  
а) – «Project Explorer», б) – «Package Explorer», в) – «Outline»

Крім цього, повідомлення про помилки (попередження) відображаються в представленні «Outline» (рис. 5.23в). Як видно з малюнка, тут помилковим може бути позначене навіть окреме речення імпорту з бібліотек класів.

І, по-третє, основна інформація про попередження компілятора й повідомлення про помилки в програмі відображається безпосередньо в тексті програми, а також у лівій і правій частині вікна редактора (рис. 5.24). У тексті програми помилки й попередження підкреслюються хвилястою лінією

(11, strHol2("str), (ort java.math.\*), помилки виділяються червоним кольором, а попередження – жовтим. Маркер із хрестиком на червоному фоні –  або  (для грубих синтаксичних помилок) у лівому полі вікна вказує на позицію помилки усередині відображуваної сторінки, відповідно маркером з жовтим трикутником –  позначається позиція попередження. Індикатор  червоного кольору в правому полі – це позиція помилки відносно до файлу, а жовтим прямокутником  вказується позиція попередження у файлі.

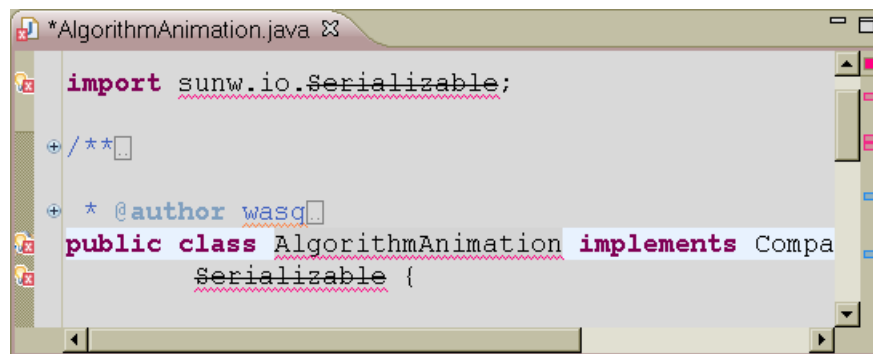


Рисунок 5.24 – Відображення помилок у вікні редактора Java



Квадратик  жовтого або  червоного кольору у правому верхньому куті вікна редактора слугує індикатором, при наведенні вказівника миші на який, з'являється спливаюче віконце із зазначенням загальної кількості помилок і попереджень у файлі. Ця ситуація показана на рис. 5.25.



Рисунок 5.25 – Індикація загальної кількості помилок і попереджень у файлі

Наведення вказівника миші на індикатори в правому і лівому полі вікна редактора також викликає появу спливаючого вікна, але вже з повідомленням про допущену помилку або попередження компілятора (див. рис. 5.26). Причому, якщо в рядку, на якій вказує індикатор, є кілька помилок і попереджень, то в

спливаючому вікні перераховуються вони всі. На рис. 5.26, саме й є представленими два подібні випадки.

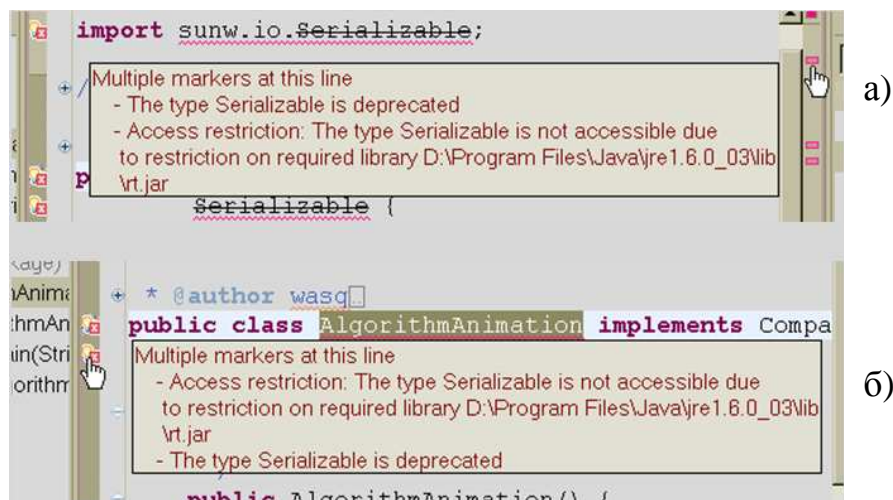


Рисунок 5.26 – Приклад спливаючих попереджень про помилки для індикаторів у правому – а) і лівому – б) полі вікна редактора

Аналогічно поводить редактор і тоді, коли вказівник миші наводиться на підкреслену частину рядка вихідного коду (див. рис. 5.27).

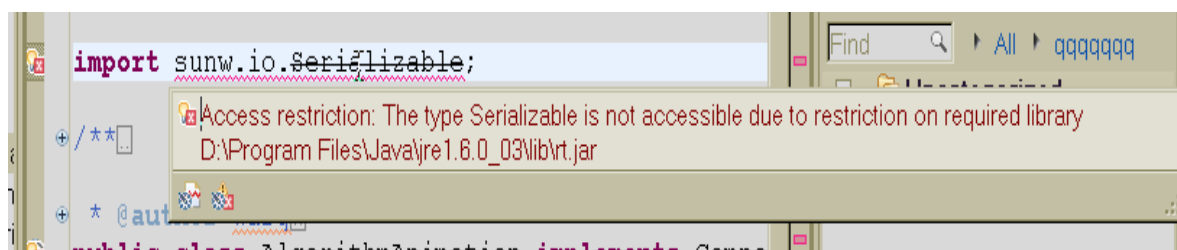


Рисунок 5.27 – Приклад спливаючого вікна з повідомленням про помилку для підкресленого вихідного коду в редакторі

Крім описаних методів індикації помилок, у редакторі перспективи Java є передбаченим й ще один спосіб індикації. У цьому разі звертання до деяких методів, котрі використовуються у програмі, відображаються шрифтом із закреслюванням, наприклад, `frame.show();` або `suspend();` (див. рис. 5.28). На цьому ж малюнку видно, що в такий спосіб позначаються попередження компілятора про звертання до застарілих методів бібліотечних класів, застосування яких у цей час не є рекомендованим.

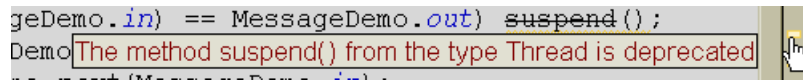


Рисунок 5.28 – Приклад індикації звертання до застарілого методу

Як видно з приведених прикладів Eclipse досить ефективно й наполегливо повідомляє програміста про допущені помилки. Причому процес повідомлення про помилки здійснюється безпосередньо при вводі вихідного коду, а не після компіляції, як це має місце в інших IDE. Такий підхід значно полегшує й інтенсифікує процес створення додатка мовою Java.

На закінчення цього розділу необхідно зупинитися на одній з оригінальних властивостей редактора Java по обробці помилок, котра практично не підтримується в редакторах інших інтегрованих середовищ розробки для різних мов програмування, у тому числі й Java. Ця властивість – спроможність редактора виявляти не тільки синтаксичні помилки у вихідному коді мови Java, але й граматичні помилки в коментарях (див. рис. 5.29). Це дійсно унікальна властивість. Але розроблювачі JDT для Eclipse включили такий сервіс у зв'язку з тим, що Eclipse, разом із програмою javadoc з пакета поставки JDK від Oracle, на основі коментарів до вихідного коду здатний створювати стандартний пакет документації для додатка, що розроблюється.

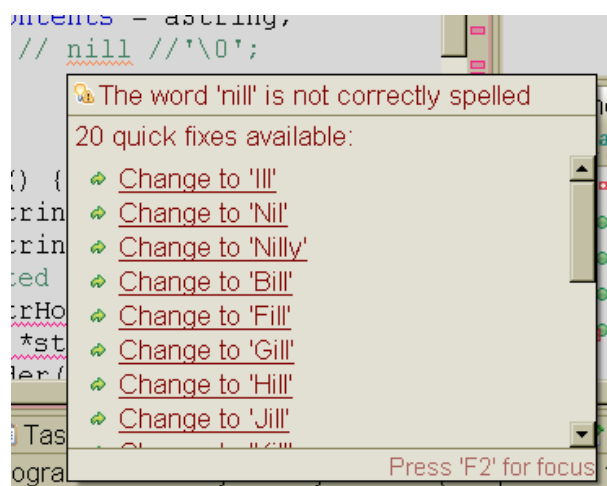


Рисунок 5.29 – Список пропонованих виправлень слова в коментарі

На рис. 5.29 видно, як граматичні помилки в словах коментарю підкреслюються жовтогарячою хвилястою лінією й, як при наведенні на них вказівника миші, з'являється вікно з попередженням про помилкове написання підкресле-

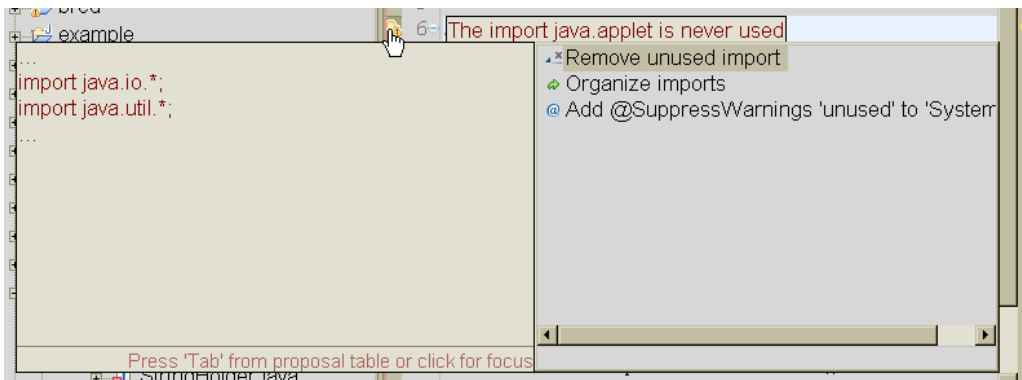


ного слова. Більше того, у цьому ж вікні пропонується кілька варіантів виправлення помилки (у цьому випадку 20). Необхідне виправлення можна просто вибрати із запропонованого списку, при умові, якщо воно там є. Самим останнім варіантом виправлення пропонується занесення невідомого слова в словник для подальшого використання.

#### 5.4.2 Виправлення помилок Java-коду в редакторі

Є цілком очевидним, що якщо Eclipse справляється з перевіркою й виправленням граматичних помилок у коментарях, то система повинна ще більш ефективно вирішувати питання, пов'язані з помилками безпосередньо в Java-кодi. І дійсно, як уже вказувалося вище (дiв. стор. 87), JDT здійснює не тільки всiляку iндикацiю помилок, але й, майже в кожному випадку, намагається iдказати можливі шляхи їхнього виправлення.

Так на рис. 5.30 зiдображена ситуацiя, коли Eclipse, при наведеннi курсору мишi на значок у полi лiворуч вiкна редактора, повiдомляє про наявнiсть попередження про те, що класи з бiблiотеки `java.applet`, котрi пiдключається до програми, у вихiдному кодi не використовуються. Ситуацiя абсолютно безпечна, але, проте, при клацаннi лiвою кнопкою мишi, коли курсор знаходиться на значку, виводиться вiконце (рис. 5.30б) зi списком з трьох доступних у данiй ситуацiї способiв автоматичного усунення можливостi появи цього попередження надалi. Це вiконце супроводжується ще одним вiкном (рис. 5.30а), у якому зiдображається вихiдний код, що буде одержаним, пiсля вибору того або iншого методу автоматичного виправлення помилки (попередження). Таким чином, перемiщаючи мишкою або клавішами зi стрiлками  $\uparrow$  та  $\downarrow$  курсор у першому вiконцi й, спостерiгаючи за вiдповiдним другим, можна вибрати найбільш придатний спiсiб виправлення помилки. Наступне натискання на клавішу `<Enter>` або подвiйне кликання мишкою по обраному пункту виправлень, автоматично приводить до вiдповiдних змiн у вихiдному кодi. Рис. 5.30 демонструє можливий результат змiни вихiдного коду при виборi пункту «Remove unused import», а на рис. 5.31 показаний прогнозований результат автоматичних змiн коду при виборi пункту «Add @SuppressWarnings 'unused'». На рис. 5.32 наведений остаточний результат виправлень пiсля вибору пункту «Organize imports».



а)

б)

Рисунок 5.30 – Повідомлення про попередження з можливими способами автоматичного виправлення – б) і відображенням виправленого вихідного коду – а)

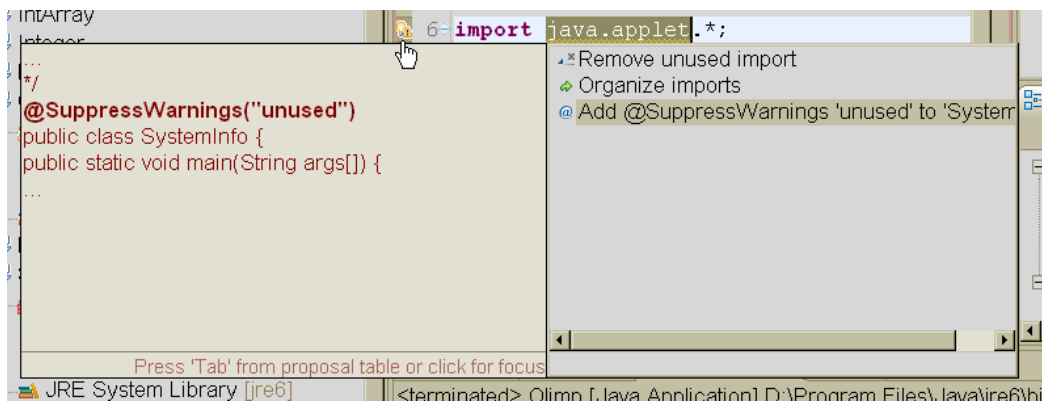


Рисунок 5.31 – Вибір пункту «Add @SuppressWarnings 'unused'» і прогнозований результат виправлення вихідного коду

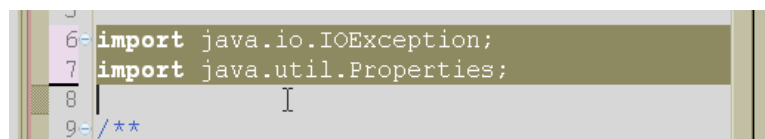


Рисунок 5.32 – Результат автоматичного виправлення вихідного коду при виборі пункту «Organize imports»


На рис. 5.32 також видно, що, за допомогою горизонтальної лінії в полі нумерації рядків – , у редакторі Java показується видалення рядка вихідного коду в цьому місці.

Рис. 5.33 ілюструє можливі виправлення допущеної помилки, котрі пропонуються системою. У першому можливому варіанті виправлення (рис. 5.33а)

пропонується створити заголовок нового методу класу з відповідним формальним параметром і включити в тіло методу коментар «TODO» для нагадування про необхідність подальшої роботи зі створення коду (див. вище, стор. 88)



Рисунок 5.33 – Можливі варіанти автоматичного виправлення помилки у вихідному коді програми: а) – шляхом автоматичної генерації оголошення нового методу, б) – перейменуванням невірною ідентифікатора

Другий можливий варіант виправлення, який підказує JDT (рис. 5.33б), полягає в автоматичному перейменуванні невірною ідентифікатора. Причому, у лівому вікні, у якому виводяться результати виправлення, вказується, що будуть перейменовані всі посилання на цей ідентифікатор у поточному файлі, але перейменування в інших файлах зроблено не буде.

У деяких випадках, наприклад, через обмеження, що накладаються на використання бібліотеки `sunw.io.*` (див. рис 5.26), інтелектуальний помічник Eclipse яких-небудь корисних виправлень автоматично зробити не може. Єдина можлива пропозиція, котру він дає в такому випадку, це включення до вихідного коду директиви придушення контролю відповідного попередження (див. рис. 5.34).

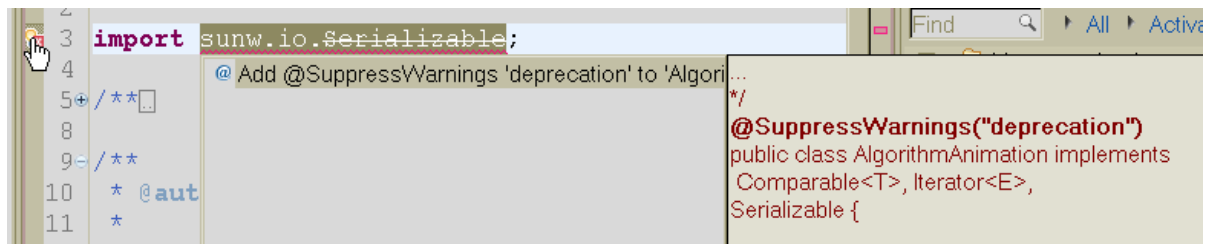


Рисунок 5.34 – Єдина можлива пропозиція по усуненню попередження про застарілий інтерфейс

Але в таких випадках може допомогти потужна система пошуку, котра вбудована в Eclipse. Приклад використання такого пошуку показується на рис. 5.35. У цьому прикладі пошук був зроблений по всіх бібліотеках SDK Java, з метою можливого знаходження оголошення інтерфейсу «Serializable» в інших бібліотеках крім «sunw.io». І, дійсно, оголошення інтерфейсу було знайдене ще й у пакеті «java.io», це добре видно на рис. 5.36. Тут показано, що знайдені оголошення одночасно відображаються відразу в духу представленнях.

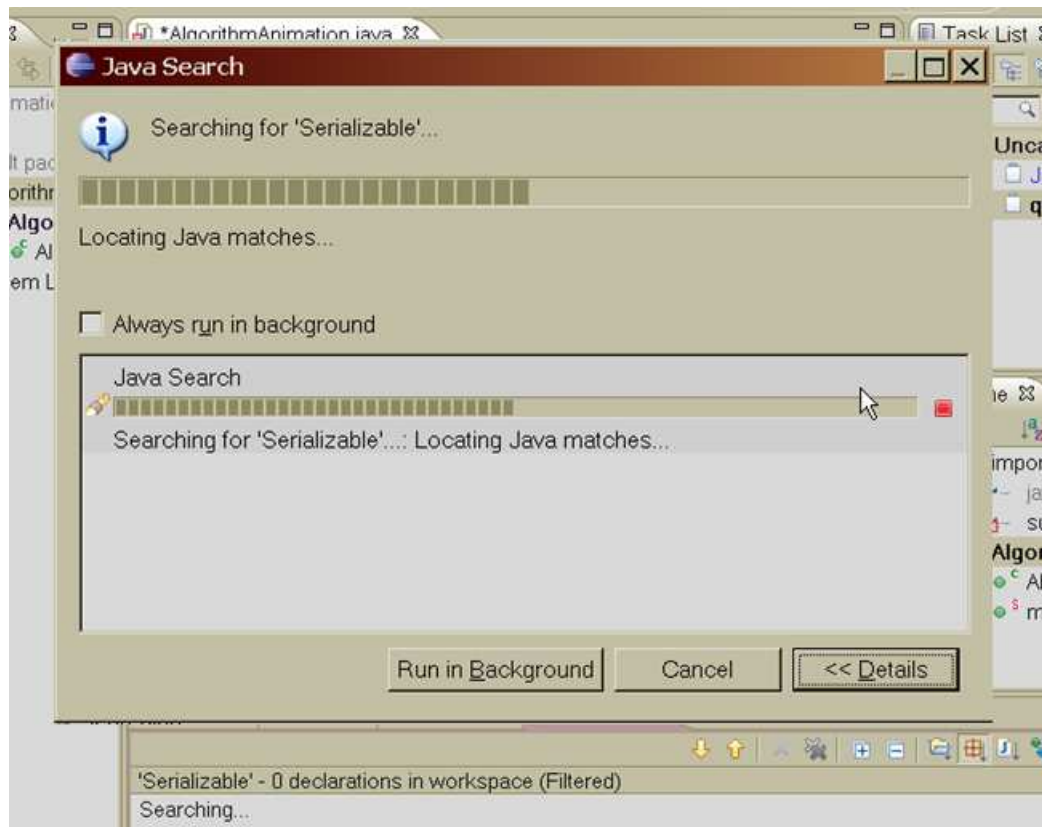


Рисунок 5.35 – Здійснення пошуку інтерфейсу «Serializable» по всіх бібліотеках системи програмування Java

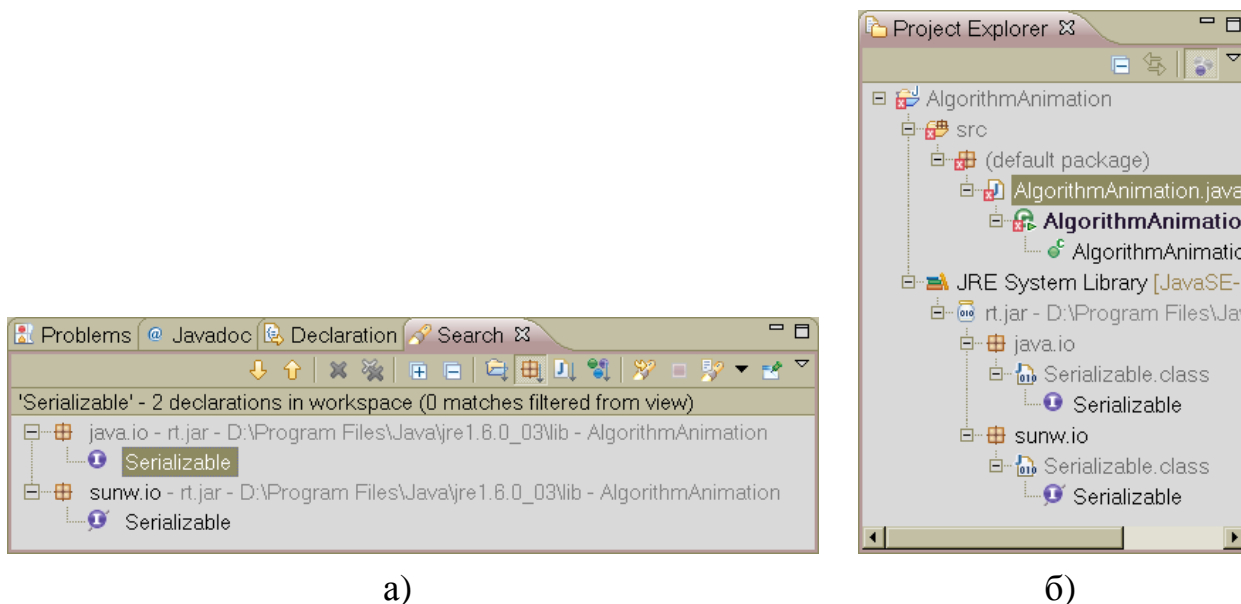





Рисунок 5.36 – Відображення результатів пошуку в представленні «Search» – а) і в представленні «Project Explorer» – б)

Факт знаходження потрібного інтерфейсу у двох різних пакетах SDK дозволяє внести виправлення у вихідний код уже в ручному режимі. Для цього досить у третьому рядку програми (див. рис. 5.37) замінити «sunw» на «java». При цьому, як видно з рис. 5.37б), редактор Eclipse відразу ж реагує на внесені зміни заміною значка наявності помилки  в полі ліворуч вікна редактора на значок виправленої помилки  (білий хрестик у сірому кружечку). Символ з'являється відразу у двох рядках коду (у третій і чотирнадцятій), крім того в цих же рядках зникає підкреслення хвилястою червоною лінією й шрифт із закреслюванням замінюється на звичайний. Червоні прямокутники у відповідних позиціях у полі праворуч вікна редактора замінюються на білі .

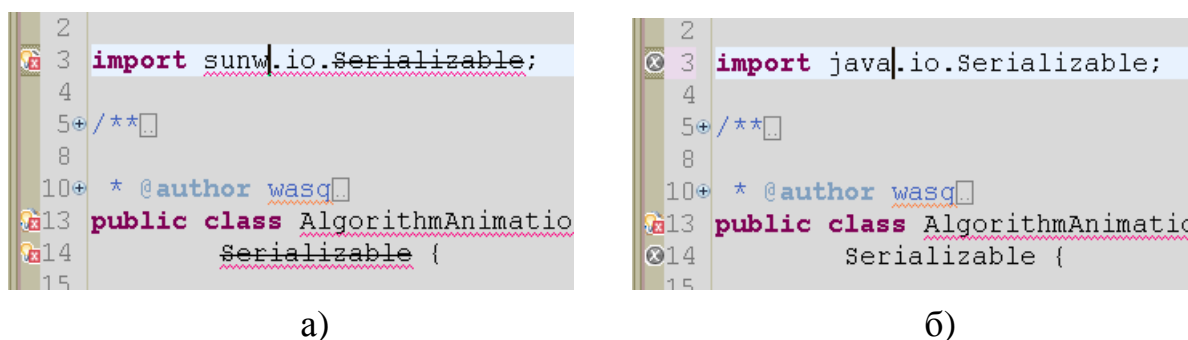


Рисунок 5.37 – Відображення вихідного коду в редакторі Java до виправлення помилки – а) і після – б)

Нарешті, редактор Java-коду надає ще одну можливість швидкого виправлення помилок, що наявні у програмі. Ця можливість реалізується за допомогою вікна, котре з'являється при наведенні курсору миші на частину коду, що підкреслена червоною хвилястою лінією. Приблизний вигляд такого вікна ілюструється рис. **Ошибка! Источник ссылки не найден..** Тут повідомляється причина помилки, вказується кількість можливих способів швидкого усунення помилки й виводиться список таких способів з можливістю вибору одного з них мишкою або клавішами зі стрілками ↑ та ↓. Для здійснення вибору необхідно попередньо натиснути клавішу <F2> (підказка «Press 'F2' for focus» у правому нижньому куті вікна) або перемістити вказівник миші на вікно.

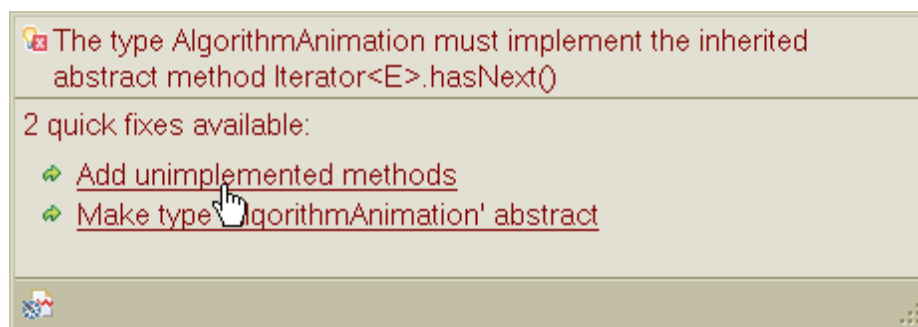


Рисунок 5.38 – Вікно зі списком способів швидкого виправлення помилок

## ДОДАТОК А

### Ключі командного рядку системи

У табл. А.1 наведені відомості про ключі командного рядка програми Eclipse, котрі слугують для зміни параметрів запуску платформи з метою налаштування її характеристик.

Таблиця ДОДАТОК А.1 Ключі платформи Eclipse

Ключ	Опис
-application [App ID]	Слугує для зазначення додатка, котрий запускається і є додатком за замовчуванням для робочого місця платформи Eclipse. Цей ключ є дуже важливим для розробників, котрі використовують та працюють з Rich Client Platform.
-boot [boot code path]	Цей ключ дозволяє змінити розташування для класів, що розташовані у файлах boot.jar та startup.jar і підвантажуються до системи при її запуску. Зазвичай розробники, що працюють з Eclipse не застосовують цей ключ.
-consolelog	Змушує систему поміщати звіт про помилки, на доповнення до вбудованого вікна помилок Eclipse, ще і до консолі. Ключ може допомогти при з'ясуванні причини помилок, виникаючих при старті системи.
-data [workspace path]	За допомогою цього ключа вказується розташування робочого простору системи. Тобто вказується місце де Eclipse зберігає свої мета-дані і є ключем, що найбільш часто застосовується розробниками на Eclipse.
-debug [options path]	Зазначення цього ключа зі шляхом до файлу налаштувань дозволяє вказати параметри налагоджувача платформи Eclipse. Файл параметрів дуже корисний при розробці плагинів для Eclipse.
-dev [classpath entries]	Ключ дозволяє додавати параметр classpath до classpath всіх плагинів.
-nosplash	Подавляє виведення вікна завантажування при старті Eclipse.

Ключ	Опис
-os [os-id]	Необхідність вказувати цей ключ виникає тільки у разі, коли Eclipse не в змозі виконати свою конфігурацію у середовищі операційної системи. За звичай таке трапляється коли нова операційна система додана у перелік віОС, що підтримуються. В більшості інсталяцій цей ключ не є потрібним.
-vm [vm-path]	Цей ключ дозволяє вказати альтернативну JVM для запуску системи Eclipse. Якщо у Windows ключ не вказаний, Eclipse використовує перший файл java.exe, що він розшукує його у системній змінній PATH.
-ws [ws-id]	Необхідність вказувати цей ключ виникає тільки у разі, коли Eclipse не в змозі виконати свою конфігурацію у середовищі операційної системи. За звичай таке трапляється коли нова віконна система додана у перелік віконних систем, що підтримуються. В більшості інсталяцій цей ключ не є потрібним.

Користувачі Eclipse часто застосовують ключ *-data* для зазначення альтернативного місця де Eclipse зберігає свої проекти. Заміна цього місцеположення дозволяє розташовувати свої проекти в каталозі, котрий може бути збережений стандартними засобами ОС. Незалежно від того, застосовується для проектів робочий простір за замовчуванням або ні, рекомендується зберігати вміст каталогу робочого простору стандартними засобами ОС. Eclipse зберігає важливі метадані у каталозі *.metadata* в робочому просторі користувача. Якщо щось трапляється з цими даними, Eclipse іноді не може стартувати, і доводиться відновлювати всі проекти заново, щоб продовжити користуватись Eclipse.




## ДОДАТОК Б

### Стислий опис «гарячих клавiшiв», що налаштованi у платформi

Даний додаток призначений для надання короткого огляду з основних «гарячих клавiшiв» iнтегрованого середовища Eclipse, котрi полегшують процес роботи у середовищi та роблять його бiльш ефективним i швидким. За допомогою «гарячих клавiшiв» деякi операцiї виконуються скорiш, нiж при застосуванні пунктів меню за допомогою мишки.

На сам перед треба вiдзначити таку комбiнацiю клавiш як <Ctrl+Shift+L>. На рис. Б.1 показане вiкно, що викликається по одночасному натисканню цих клавiшiв.



Inline	Alt+Shift+I
Inspect	Ctrl+Shift+I
Last Edit Location	Ctrl+Q
Make Landmark	Ctrl+Alt+Shift
Make Less Interesting	Ctrl+Alt+Shift
Maximize Active View or Editor	Ctrl+M
Move - Refactoring	Alt+Shift+V
New	Ctrl+N
New menu	Alt+Shift+N
Next	Ctrl+.
Next Editor	Ctrl+F6
Next Page	Alt+F7
Next Perspective	Ctrl+F8
Next Sub-Tab	Alt+PageDown
Next View	Ctrl+F7
Open Attached Javadoc	Shift+F2
Open Call Hierarchy	Ctrl+Alt+H

Press "Ctrl+Shift+L" to open the preference page.

Рисунок Б.1 – Вiкно швидкого перегляду перелiку «гарячих клавiшiв» Eclipse

Список у даному вiкнi дозволяє досить швидко переглянути перелiк всiх гарячих клавiшiв, налаштованих у Eclipse. Якщо при цьому активоване диалогове вiкно, то показується список клавiш дозволених для виконання функцiй у вiкнi. Як видно з напису у останньому рядку вiкна, повторне натискання той же комбiнацiї повинно викликати вiдкриття диалогового вiкна редагування комбiнацiй клавiшiв. Зовнiшнiй вигляд цього вiкна показаний на рис. Б.2.

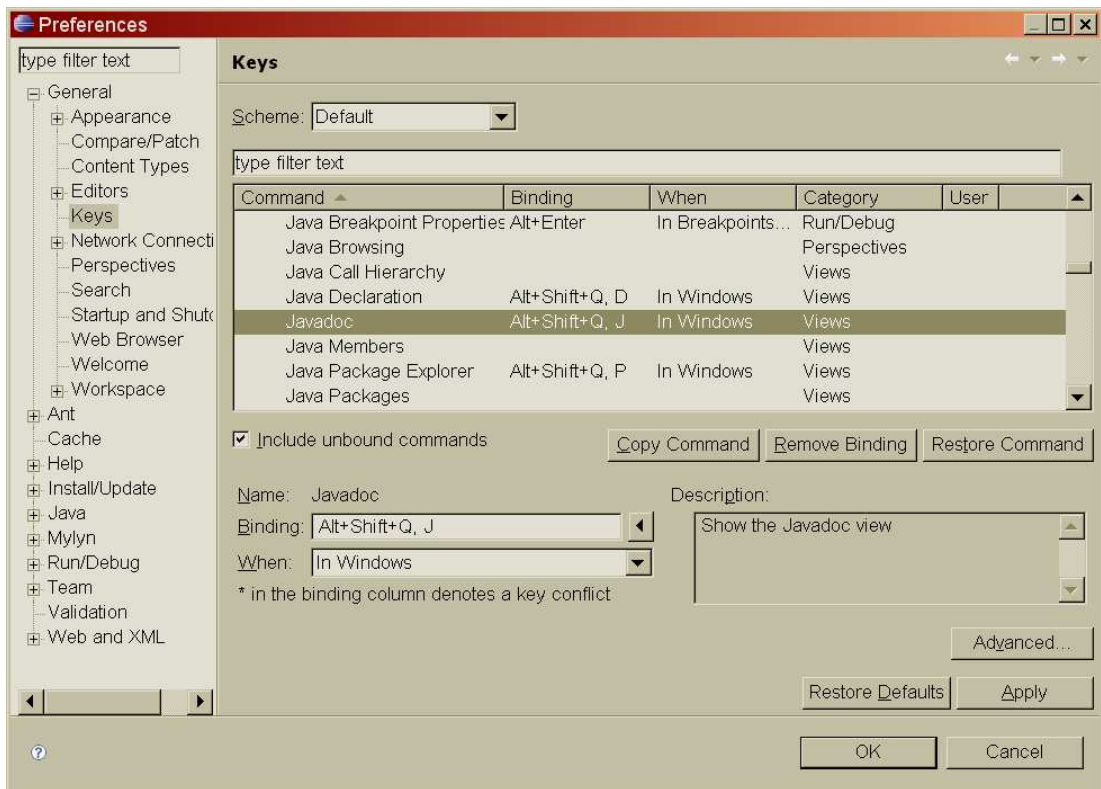


Рисунок Б.2 – Зовнішній вигляд діалогового вікна редагування комбінацій клавішів

За допомогою цього вікна можна налаштувати свої власні комбінації клавішів та змінити, найбільш зручним для себе чином, вже існуючі комбінації. Опис цього процесу не є завданням нашого розгляду, тому приводити його не має сенсу.

Найбільш часто при роботі у інтегрованому середовищі розробки програм доводиться виконувати операції з вікнами (розкривати вікно до максимально можливого розміру, переключатись між різними вікнами, тощо). Зазвичай це робиться за допомогою миші, шляхом натискування на відповідні іконки (див. рис. Б.3), або робочу поверхню іншого вікна. Однак для цього є необхідним відірватися від клавіатури, розшукати рукою мишу, знайти відповідну іконку, натиснути на неї и знов повернутися до клавіатури. Більш продуктивним буде скористуватися «гарячими клавішами», наприклад, для розгортання вікна (або його згортання) застосовується комбінація <Ctrl+M>.

В середовищі Eclipse, коли на екрані одночасно розташовується багато різних вікон, доводиться досить часто переключатись між ними. Замість миші набагато краще це робити за допомогою комбінації клавішів <Ctrl+F7>. При першому натисканні цієї комбінації на екрані з'являється віконце у якому пере-

раховані всі вікна середовища, котрі наявні в ньому у цей час (див. рис. Б.4). Якщо відразу відпустити клавіші, фокус буде змінено на попереднє активне вікно. Але якщо не відпускати клавішу <Ctrl>, а зробити нове натискування на клавішу <F7> буде сфокусоване передостаннє активне вікно і так далі. Тобто дія цієї комбінації нагадує дію комбінації <Alt+Tab> у Windows. Якщо при відкритому вікні натиснути курсорну клавішу вгору або вниз, то вікно не закриється доти, коли клавішею <Enter> або кліком мишки явно не буде вибрано потрібне вікно.



Рисунок Б.3 – Іконка для розгортання віконця редактору

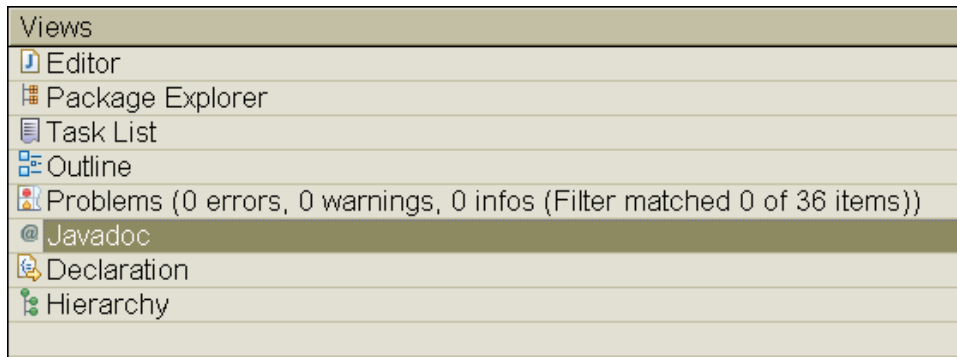


Рисунок Б.4 – Зовнішній вигляд вікна переключення вікон за допомогою комбінації <Ctrl+F7>

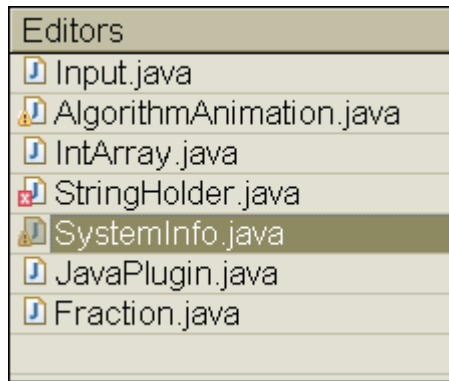


Рисунок Б.5 – Швидке переключення редакторів

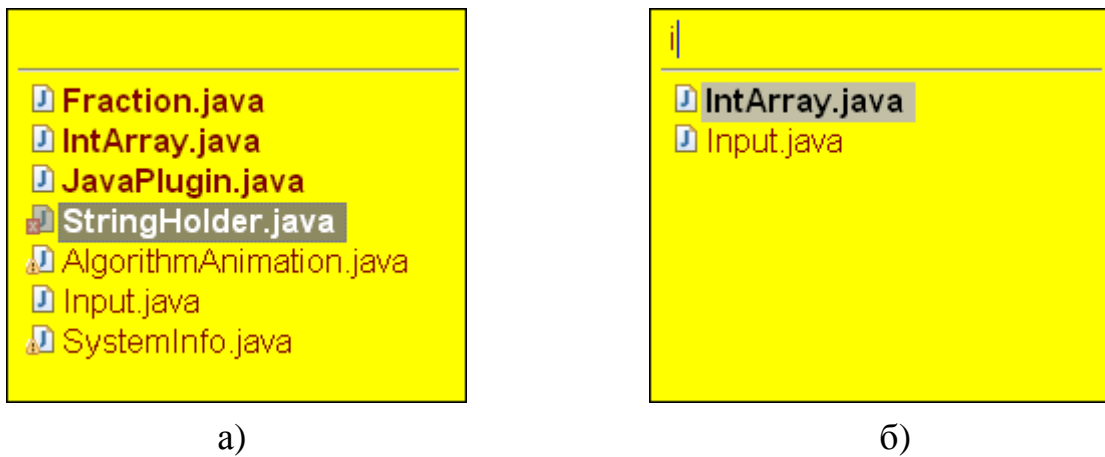


Рисунок Б.6 – Швидке переключення редакторів з пошуком

Перейти до останнього вікна (місця) де відбувалось редагування <Ctrl+Q>  
Зручно після навігації по коду для повернення назад в точку редагування.

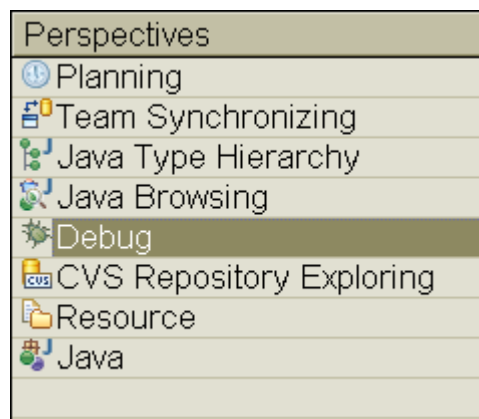


Рисунок Б.7 – Швидке переключення перспектив





Взагалі на одному з форумів в Інтернеті за адресою <http://www.javaworld.com/javaforums/showflat.php?Cat=0&Number=21505&an=0&page=0#Post21505> є таке висловлення про можливість застосування гарячих клавішів у Eclipse:

«What about Windows->Preferences->General->Keys. There are all the list for key-shortcuts and you can configure it. That's why Eclipse is the great option in IDEs.» («Ну, а що з приводу Windows->Preferences->General->Keys. Там міститься повний перелік гарячих клавішів і ви можете конфігурувати його. От чому Eclipse є найбільш функціональною IDE з усіх інших.»).








## ДОДАТОК В

### Іконки та кнопки панелей інструментів середовища




Наступні іконки можуть зустрітися у віконці навігації по проекту Project Explorer:

Іконка	Опис
	Проект (відкритий)
	Папка (відкрита)
	Проект (закритий)
	Узагальнений файл

В полі маркерів редактора (зліва від області редагування) можуть бути розташовані наступні маркери:

Іконка	Опис
	Закладка
	Точка зупинки
	Маркер завдання
	Результат пошуку
	Маркер помилки
	Маркер попередження
	Інформаційний маркер

У віконці завдань можуть зустрітися такі маркери:













Іконка	Опис
	Завдання з високим пріоритетом
	Завдання з низьким пріоритетом
	Завершене завдання

Наступні кнопки розташовуються у панелі інструментів робочого столу, у панелях інструментів для представлень і в панелі коротких викликів:










Кнопка	Опис	Кнопка	Опис
	Відкрити нову перспективу		Зберегти вміст активного редактора
	Зберегти вміст всіх редакторів		Зберегти вміст редактора під іншим ім'ям, або у іншому місці
	Відкрити діалог пошуку		Друкувати вміст редактора
	Відкрити майстер створення ресурсів		Відкрити майстер створення файлів
	Відкрити майстер створення каталогів		Відкрити майстер створення проектів
	Відкрити майстер імпорту		Відкрити майстер експорту
	Запустити покрокове будівництво		Запустити програму
	Налагодження програми		Запустити зовнішній інструмент або Ant
	Вирізати виділення у буфер		Копіювати виділення у буфер
	Вставити виділення з буфера		Відмінити останнє редагування
	Відновити останнє редагування		Перейти до наступного пункту у переліку
	Перейти до попереднього пункту у переліку		Перейти уперед
	Перейти назад		Піднятися на один рівень
	Додати закладку або завдання		Відкрити у представленні меню, що випадає
	Закрити представлення або редактор		Прикріпити редактор щоб уникнути автоматичного повторення
	Відібрати завдання або властивості		Перейти до завдання, проблеми, або закладки у редакторі
	Відновити властивості за умовчанням		Показувати пункти як дерево
	Обновити вміст представлення		Сортувати перелік за алфавітом
	Перервати тривалу операцію		Видалити обраний пункт або вміст
	Остання позиція редагування		Переключити входження, що є маркірованим
	Показувати тільки джерело елемента, що виділений		

## Іконки зовнішніх інструментів та Ant

### Об'єкти

-  Ant-файл побудови проекту
-  Цільовий об'єкт Ant має помилку
-  Неможливий будівник проекту
-  Ціль за умовчанням
-  Доступна ціль Ant (ціль з описом)
-  Внутрішня ціль Ant (ціль без опису)
-  Файл .jar
-  Властивості Ant
-  Завдання Ant
-  Тип Ant
-  Завдання імпорту Ant
-  Завдання макровизначення Ant

### Конфігурування запуску

-  Запустити зовнішній інструмент
-  Конфігурування запуску Ant
-  Конфігурування запуску програми
-  Основна таблиця
-  Оновити таблицю
-  Побудувати таблицю
-  Таблиця цілей
-  Таблиця властивостей
-  Таблиця Classpath



## Представлення Ant



Представлення Ant



Додати файл побудови проекту



Знайти і додати файли побудови проекту



Виконати обраний файл побудови або обрану ціль



Видалити обраний файл побудови проекту



Видалити всі файли побудови проекту



Властивості

## ПЕРЕЛІК ПОСИЛАНЬ

1. Kathy Sierra, Bert Bates, Head First Java 2nd Edition, Covers Java 5.0 – O'Reilly Media, Inc., 2008. – 676 p.
2. Java Enterprise in a Nutshell, Third Edition / Jim Farley, William Crawford etc. – O'Reilly Media, Inc., 1999. – 896 p.
3. David Flanagan, Java in a Nutshell. A Desktop Quick Reference, 5th edition – O'Reilly Media, Inc., 2005. – 1264 p.
4. Java SE 6 Documentation [Електронний ресурс] – Режим доступу: <http://download.oracle.com/javase/6/docs/index.html>
5. About the Eclipse Foundation [Електронний ресурс] – Режим доступу: <http://www.eclipse.org/org/>
6. Steve Holzner, Eclipse – O'Reilly Media, Inc., 2004. – 336 p.
7. Eclipse marketplace [Електронний ресурс] – Режим доступу: <http://marketplace.eclipse.org/>
8. Anil Hemrajani, Agile Java Development with Spring, Hibernate and Eclipse, 1st edition – Sams Publishing, 2006. – 360 p.
9. Бердачук С., Eclipse RCP. Файловый менеджер [Електронний ресурс] – Режим доступу [http://berdaflex.com/ru/eclipse/books/rcp\\_filemanager/index.html](http://berdaflex.com/ru/eclipse/books/rcp_filemanager/index.html)
10. Barry Burd, Eclipse For Dummies – Wiley Publishing, Inc., 2005 – 361 p.
11. Joe Pluta, Eclipse: Step-by-Step – MC Press, 2003 – 376 p.
12. Steve Holzner, Eclipse – O'Reilly Media, Inc., 2004 – 334 p.
13. Bill Dudley, Eclipse 3 Live – SourceBeat, LLC., 2004 – 288 p.
14. The Java Developer's Guide to Eclipse. By Sherry Shavor, Pat McCarthy, John Kellerman, Jim D'Anjou, Scott Fairbrother. Publisher: Addison-Wesley. Pub. Date: May 2003. ISBN: 0321159640. Pages: 896.
15. Java SE 6 Documentation [Електронний ресурс] – Режим доступу <http://download.oracle.com/javase/6/docs/index.html>
16. C++ и Java: совместное использование [Електронний ресурс] – Режим доступу <http://www.codenet.ru/webmast/java/jcc.php>