ОСНОВИ ВВОДУ / ВИВОДУ

Лекція 6.2

доц. кафедри Інформатики
Сінельнікова Т.Ф.

ХНУРЕ, Кафедра Інформатики
informatika@kture.kharkov.ua

- Основи вводу / виводу
- Потоки
- Байтові і символьні потоки
- Класи байтових потоків
- Попередньо визначені потоки
- Зчитування консольного вводу
- Зчитування символів
- Зчитування рядків
- Запис консольного виводу
- Клас PrintWriter
- Зчитування і запис файлів
- Аплети
- Модифікатори *transient* і *volatile*
- Використання instanceof

Основи вводу / виводу

- Реальні програми Java не засновані на консольних текстових програмах.
- Вони є графічними апплетами, які для взаємодії з користувачем покладаються на систему класів AWT (Abstract Window Toolkit, інструментарій абстрактного вікна) мови Java.
- Підтримка Java для консольного вводу / виводу обмежена і не дуже зручна у використанні - навіть в простих прикладах програм.
- Текстовий консольний вводу / виводу в дійсності не дуже важливий для Java-програмування.
- Незважаючи на це, Java забезпечує сильну, гнучку підтримку по суті текстового вводу / виводу для файлів і мереж. Система вводу / виводу Java связна і несуперечлива.

Потоки

- Java-програми виконують введення / виведення через потоки. Потік є абстракцією, яка або виробляє, або споживає інформацію.
- Потік пов'язується з фізичним пристроєм за допомогою системи введення / виводу Java (Java I / O system).
- Всі потоки поводяться однаковим чином, хоча фактичні фізичні пристрої, з якими вони пов'язані, можуть сильно відрізнятися. Таким чином, одні й ті ж класи і методи вводу / виводу можна застосовувати до пристроїв будь-якого типу.
- Потік введення може витягувати багато різних видів вхідних даних: з дискового файлу, з клавіатури або мережевого роз'єму. Потік виведення може звернутися до консолі, дискового файлу або мережному з'єднанню (сокету).
- Завдяки потокам ваша програма виконує введення / виведення, не розуміючи різниці між клавіатурою і мережею.
- Java реалізує потоки за допомогою ієрархії класів, визначених в пакеті java.io.

Байтові і символьні потоки

- Java 2 визначає два типи потоків (точніше - потокових класів і об'єктів): байтовий і символьний.
- Байтові потоки надають зручні засоби для обробки введення і виведення байтів. Байтові потоки використовуються, наприклад, при читанні або запису даних в двійковому коді.
- Символьні потоки надають зручні засоби для обробки введення і виведення символів. Вони використовують Unicode і тому можуть бути інтернаціоналізовані. Крім того, в деяких випадках символьні потоки більш ефективні, ніж байтові.
- На самому низькому рівні введення / виведення все ще байтове. Символьно-орієнтовані потоки забезпечують зручні та ефективні засоби для обробки символів.

Класи байтових потоків

- Байтові потоки визначаються у двох ієрархіях класів.
- Наверху цієї ієрархії - два абстрактних класу: Inputstream і Outputstream.
- Кожен з цих абстрактних класів має кілька конкретних підкласів, які обробляють відмінності між різними пристроями, такими як дискові файли, мережеві з'єднання і навіть буфери пам'яті.
- Абстрактні класи Inputstream і Outputstream визначають декілька ключових методів, які реалізуються іншими поточними класами. Два найбільш важливих-read () і write (), які, відповідно, читають і записують байти даних.
- Обидва методи оголошені як абстрактні всередині класів inputstream і outputstream і перевизначаються похідними потоковими класами.

Класи байтових потоків

Поточний клас	Значення
BufferedInputstream	Буферизують потік вводу
BufferedOutputStream	Буферизують потік виводу
ByteArrayInputstream	Потік введення, який читає з байт-масиву
ByteArrayOutputStream	Потік виводу, який записує в байт-масив
DataInputstream	Потік введення, який містить методи для читання даних стандартних типів Java
DataOutputStream	Потік виводу, який містить методи для запису даних стандартних типів Java
FileInputstream	Потік введення, який читає з файлу
FileOutputStream	Потік виводу, який записує в файл
FilterInputStream	Реалізує Inputstream
FilterOutputStream	Реалізує OutputStream
Inputstream	Абстрактний клас, який описує поточний введення
OutputStream	Абстрактний клас, який описує поточний висновок
PipedInputStream	Канал вводу
PipedOutputStream	Канал виводу
Printstream	Потік виводу, який підтримує print () і println ()
PushbackInputStream	Потік (введення), який підтримує однобайтових операцію "unget", повертає байт в потік введення
RandomAccessFile	Підтримує введення / виведення файлу довільного
SequenceInputStream	Потік введення, який є комбінацією двох або декількох потоків введення, які будуть читатися послідовно, один за іншим

Класи символьних потоків

- Символьні потоки визначені у двох ієрархіях класів. Наверху цієї ієрархії два абстрактних класу: Reader і Writer.
- Вони обробляють потоки символів Unicode.
- Абстрактні класи Reader і Writer визначають декілька ключових методів, які реалізуються іншими поточними класами.
- Два найважливіших методу - read () і write (), які читають і записують символи даних, відповідно. Вони перевизначаються похідними потоковими класами.

Класи символьних потоків

Поточний клас	Значення
BufferedReader	Буферизованний символьний потік вводу
BufferedWriter	Буферизованний символьний потік виводу
CharArrayReader	Потік вводу, якій читає з символьного масиву
CharArrayWrite	Вихідний потік, який записує в символьний масив
FileReader	Потік введення, який читає з файлу
FileWriter	Вихідний потік, який записує в файл
FilterReader	Відфільтрований потік вводу
FilterWriter	Відфільтрований потік виводу
InputstreamReader	Потік введення, який переводить байти в символи
LineNumberReader	Потік вводу, який рахує рядки
OutputStreamWriter	Потік вводу, який переводить байти в символи
PipedReader	Канал вводу
PipedWriter	Канал виводу
PrintWriter	Потік виводу, який підтримує print () і println ()
PushbackReader	Потік введення, який повертає символи в потік введення
Reader	Абстрактний клас, який описує символьний потік вводу
StringReader	Потік введення, який читає з рядка
StringWriter	Потік виводу, який записує в рядок
Writer	Абстрактний клас, який описує символьний потік виводу

Попередньо визначені потоки

- Всі програми Java автоматично імпортують пакет java.lang.
- Цей пакет визначає клас з ім'ям system, що інкапсулює деякі аспекти виконавчої середовища Java.
- Клас system містить також три зумовлені потокові змінні in, out і err. Ці поля оголошені в System зі специфікаторами public і static.
- Об'єкт System.out називають потоком стандартного виводу. За умовчанням з ним пов'язана консоль.
- На об'єкт System.in посилаються як на стандартне введення, який за замовчуванням пов'язаний з клавіатурою.
- До об'єкту System.err звертаються як до стандартного потоку помилок, який за замовчуванням також пов'язаний з консоллю.
- Проте ці потоки можуть бути перепризначені на будь-який сумісний пристрій вводу / виводу.

Зчитування консольного вводу

- Консольний введення в Java виконується за допомогою зчитування з об'єкта System.in.
- Щоб отримати символьний потік, який приєднаний до консолі, ви переносите ("упаковуєте") System.in в об'єкт типу BufferedReader.
- Клас BufferedReader підтримує буферізірованний вхідний потік. Зазвичай використовується наступний його конструктор:

BufferedReader (Reader inputReader)

- де inputReader - потік, який пов'язаний з створюються екземпляром класу BufferedReader.
- Reader - абстрактний клас. Один з його конкретних підкласів - це InputStreamReader, який перетворює байти в символи.
- Щоб отримати InputStreamReader-об'єкт, який пов'язаний з System.in, використовуйте наступний конструктор:

InputstreamReader (InputStream inputstream)

- Оскільки System.in посилається на об'єкт типу inputstream, його можна використовувати як параметр inputstream.
- Наступний рядок коду створює об'єкт класу BufferedReader, який пов'язаний з клавіатурою:

BufferedReader br = new BufferedReader (new InputstreamReader (System.in));

- Після того як цей оператор виконається, об'єктна мінлива br стане символьним потоком, пов'язаним з консоллю через System.in.

Зчитування символів

- Для читання символу з BufferedReader використовуйте метод read (). Версія read (), яку ми будемо застосовувати, така:

int read () throws IOException

- При кожному виклику read () читає символ з вхідного потоку і повертає його у вигляді цілочисельного значення. Коли read про стикається з кінцем потоку, то повертає -1. Як ви бачите, він може викидати виняток введення / виводу (I / O-виключення - IOException).

Зчитування символів

```
//Використовує BufferedReader для читання символів з консолі,
    import java.io. *;
    class BRRead
    {
    public static void main (String args [])
    throws IOException
    {
    char c;
    BufferedReader br = new
    BufferedReader (new InputStreamReader (System.in));
    System.out.println ("Введіть символи, 'q' - для завершення.");
    / / Зчитування символів
    do
    {
    c = (char) br.read ();
    System.out.println (c);
    }
    while (c! = 'q');
    }
    }
```

Зчитування рядків

- Для читання рядка, що вводиться з клавіатури, використовуйте версію методу readLine (), який є елементом класу BufferedReader. Його спільна форма:

String readLine () throws IOException

- Як видно, він повертає string-об'єкт.

Зчитування рядків

```
/ / Крихітний редактор.
    import java.io.. *;
    class TinyEdit {
    public static void main (String args [];
    throws IOException (
    / / Створити BufferedReader-об'єкт, використовуючи System.in
    BufferedReader br = new BufferedReader (new
    InputStreamReader (System.in));
    String str [] = new String [100];
    System.out.println ("Введіть рядки тексту.");
    System.out.println ("Введіть 'stop' для завершення.");
    for (int i = 0; i <100; i + +)
    {
    str [i] = br.readLine ();
    if (str [i]. equals ("stop"))
    break;}
    System.out.println ("\ n Ось ваш файл:");
    / / Вивести рядки на екран.
    for (int i = 0; i <100; i + +) {
    if (str [i]. equals ("stop")) break;
    System.out.println (str [i]);}
    }}
```

Запис консольного виводу

- Консольний висновок найлегше виконати за допомогою методів print () і println ().
- Ці методи визначені класом Printstream (який є типом (класом) об'єкта System.out).
- Оскільки PrintStream - вихідний потік, похідний від OutputStream, онтакже реалізує метод нижнього рівня write ().
- Найпростіша форма write (), визначена в Printstream, має вигляд:

void write (int byteval) throws IOException

- Цей метод записує в файл байт, вказаний в параметрі byteval. Хоча byteval оголошений як ціле число, записуються лише молодші вісім бітів.

Запис консольного виводу

```
/ / Демонструє System.out.write.
class WriteDemo {
public static void main (String args [])
{
int b;
b = 'A';
System.out.write (b);
System.out.write ('\ n');
}
}
```

- Ви не часто будете застосовувати write () для виконання консольного виводу (хоча це може бути корисно в деяких ситуаціях), тому що використовувати print () println () набагато легше.

Клас PrintWriter

- Хоча використання об'єкта System.out для запису на консоль все ще допустимо в Java, його застосування рекомендується головним чином для налагоджувальних цілей або для демонстраційних програм, типу тих, які показані в цій книзі.
- Для реальних Java-програм для запису на консоль рекомендується працювати з потоком типу PrintWriter.
- PrintWriter - це один з класів символьного введення / виведення. Використання подібного класу для консольного виводу полегшує інтернаціоналізацію вашої програми.
- PrintWriter визначає кілька конструкторів. Ми будемо використовувати наступний:

PrintWriter (OutputStream OutputStream, boolean flushOnNewline)

- Тут OutputStream - об'єкт типу OutputStream; flushOnNewline - булевський параметр, який використовується як засіб управління скиданням вихідного потоку в буфер виводу (на диск) кожен раз, коли виводиться символ newline (\ n). Якщо fiushOnNewilne - true, потік скидається автоматично, якщо - false, то не автоматично.
- Printwriter підтримує методи print () і println () для всіх типів, включаючи Object.
  Щоб записувати на консоль, використовуючи клас Printwriter, створіть об'єкт System.out для вихідного потоку, і скидайте потік після кожного символу newline.
- Наприклад, наступний рядок коду створює об'єкт типу Printwriter, який з'єднаний з консольним висновком:

PrintWriter pw = new Printwriter (System.out, true);

Клас PrintWriter

```
// Демонстрирує Printwriter.
import java.io.*;
public class PrintWriterDemo
{
public static void main(String args[j])
{
Printwriter pw = new Printwriter(System.out, true);
pw.println(«Це рядок:");
int i = -7;
pw.println(i);
double d = 4.5e-7;
pw.println(d);
}
```

- Вивод цієї програми:

```
Це рядок:
-7
4.5E-7
```

Зчитування і запис файлів

- Java забезпечує ряд класів і методів, які дозволяють читати і записувати файли.
- Для Java всі файли мають байтове структуру, а Java забезпечує методи для читання і запису байтів в файл.
- Java дозволяє упаковувати байтовий файловий потік в символьно-орієнтований об'єкт. (Про це пізніше).
- Для створення байтових потоків, пов'язаних з файлами, найчастіше за все використовуються два поточних класу - FilelnputStream і FileOutputStream.

FilelnputStream (String fileName) throws FileNotFoundException
FileOutputStream (String fileName) throws FileNotFoundException

де fileName визначає ім'я файлу, який ви хочете відкрити.

- Коли ви створюєте вхідний потік при відсутньому файлі, викидається виключення FileNotFoundException.
- Для вихідних потоків, якщо файл не може бути створений, викидається таке ж виключення (FileNotFoundException).
- Коли вихідний файл відкривається, будь-який файл, що існував раніше з тим же самим ім'ям, руйнується.

Зчитування і запис файлів

- Після завершення роботи з файлом, його потрібно закрити, викликавши метод close (). Він визначений як В FilelnputStream, так і в FileOutputStream в такій формі:

void close () throws IOException

- Для читання файлу можна використовувати версію методу read (), що визначений в FilelnputStream.

int read () throws IOException

- При кожному виклику він (метод) читає один байт з файлу і повертає його у формі цілочисельного значення. Коли read про зустрічає символ кінця файлу (eof), то повертає -1.

Зчитування і запис файлів

```
/ * Виведе на екран текстовий файл.
     При запуску програми вкажіть (в параметрі
     команди запуску) ім'я файлу, який ви хочете
     переглянути. Наприклад, щоб переглянути файл з
     ім'ям TEST.TXT, Використовуйте наступну
     командний рядок Java ShowFile TEST.TXT */

import j ava.io.*;
class ShowFile
{
public static void main(String args[])
throws IOException {
int i ;
FileInputStream fin;
try
{
fin = new FileInputStream(args[0]);
}
catch(FileNotFoundException e)
{
System.out.println("Файл не знайден");
return;
}
catch(ArrayIndexOutOfBoundsException e)
{
System.out.println(«Використовуйте: ShowFile
      ім'я_файлу");
return;
}
// зчитувати символи файлу, поки
// не зустрінеться символ EOF
Do
 {
i = fin.read();
if(I != -1) System.out.print((char) i);
}
while (i != -1) ;
 in.close ();
}
}
```

Аплети

- Всі попередні приклади були Java-додатками.
- Проте програма - це тільки один тип Java-програм.
- Інший тип програм представлений аплетом - невеликих додатків, які доступні на Internet-сервері, транспортуються по Internet, автоматично встановлюються і виконуються як частина Web-документа.
- Після того, як аплет прибуває до клієнта, він має обмежений доступ до ресурсів системи, які використовує для створення довільного мультимедійного інтерфейсу користувача і виконання комплексних обчислень без ризику зараження вірусами або порушення цілісності даних.
- Аплети відрізняються від додатків в декількох ключових областях.

Аплети

```
import java. awt. *;
Import java.applet.*;

public class SimpleApplet
   extends Applet
{

public void paint(Graphics g)
   {
g.drawstring ("A 'Simple
   Applet",   20,   20);
}
}
```

- Аплет починається; з двох операторів import. Перший імпортує AWT (Abstract Windowing Toolkit) - абстрактний віконний інтерфейс класи.
- Наступний рядок у програмі оголошує клас SimpleApplet. Він повинен бути оголошений як public, тому що до нього необхідно забезпечити доступ з кодів, які знаходяться поза програмою.
- Усередині SimpleApplet оголошений метод paint (). Цей метод визначено в AWT і повинен бути перевизначений аплетом. Метод paint про викликається кожного разу, коли аплет повинен відновлювати зображення свого висновку.
- Метод paint () викликається також, коли аплет починає виконання.
- Метод має один параметр типу Graphics, через який отримує графічний контекст, що описує графічне середовище виконання аплету.
- Усередині paint () знаходиться звернення до методу drawstring (), який є членом класу Graphics. Цей метод виводить рядок, починаючи, з зазначених його аргументами (x, y)-координат у вікні аплету. Він має таку загальну форму:

void drawstring (String *massage,* int *x,* int y)

Аплети

- Існує два способи виконання аплету:
- Виконання аплету Java-сумісному Web-браузером, типу Netscape Navigator або Microsoft Internet Explorer.
- Використання програми перегляду аплетів, типу стандартної утиліти JDK appletviewer. Програма перегляду аплетів виконує аплет в його вікні.
- Для виконання апплета в Web-браузері потрібно записати короткий текстовий файл у форматі мови HTML, який містить спеціальний тег <applet>. HTML-файл, який виконує simpieApplet, зовсім простий:

<applet code="SimpleApplet" width=200 height=60> </ applet>

- Усередині тега <applet> його параметри width і height визначають розміри вікна аплету. Після створення файлу, ви можете запустити свій браузер і потім завантажити цей файл, що призведе до виконання SimpleApplet.

Аплети

- Кілька ключових моментів.
- Аплети не потребують методі main ().
- Аплети повинні виконуватися програмою перегляду аплетів або браузером, що підтримує Java.
- Користувацький ввід / вивід в аплетах не виконується за допомогою Java-класів поточного введення / виведення. Замість цього аплети використовують? Інтерфейс, забезпечений системою AWT.

Модифікатори *transient* і *volatile*

- Коли екземплярність мінлива оголошена як transient, то її значення не буде запомнено при збереженні об'єкта. наприклад:

```
class T {
    transient int a; / / не буде зберігатися
    int b; / / зберігатиметься
    }
```

- Модифікатор volatile повідомляє компілятору, що змінна, модифікована з його допомогою, може бути несподівано змінена іншими частинами вашої програми.

Використання instanceof

- Іноді корисно розпізнавати тип об'єкта під час виконання.
- Наприклад, можна мати один потік виконання для генерації різних типів об'єктів, а інший - для їх обробки. У цій ситуації обробляє процесу корисно було б знати тип кожного об'єкта, що приймається на обробку.
- В Java неприпустиме приведення викликає помилку часу виконання. Багато неприпустимих привидів можна перехопити під час компіляції. Однак операції приведення, пов'язані з типами об'єктів (тобто з ієрархіями класів), можуть виявитися неприпустимими і можуть бути виявлені тільки під час виконання. Операція instanceof має таку загальну форму:

object instanceof type

- де object - екземпляр класу; type - клас (як тип). Якщо object-операнд має тип або його тип може бути приведений до типу, зазначений в type-операнді, то результат операції instanceof має значення true. Інакше, її результат - false. Таким чином, instanceof - це засіб, за допомогою якого ваша програма може отримати інформацію про тип об'єкта під час виконання.
- Операція instanceof не потрібна для більшості програм, тому що, взагалі-то, ви знаєте тип об'єкта, з яким працюєте.

Native-методи

- Іноді може виникнути бажання викликати підпрограму, яка написана на іншій мові, а не на Java.
- Як правило, така підпрограма існує як виконується код для CPU і середовища, в якій ви працюєте - тобто як "рідний" (native) код.
- Однак через те, що Java-програми компілюються в байт-код, який потім інтерпретується (або компілюється "на льоту") виконавчою системою Java, здавалося б, неможливо викликати підпрограму native-коду зсередини Java-програми.
- В Java існує ключове слово native, яке використовується для оголошення методів native-коду. Після оголошення ці методи можна викликати усередині Java-програми точно так само, як викликається будь-який інший метод Java.
- Для оголошення native-методу потрібно випередити його заголовок модифікатором native, при цьому, однак, не слід визначати ніякого тіла. Наприклад:

public native int meth ();

- Після оголошення native-методу, слід записати сам рідний метод і виконати досить складну процедуру для зв'язку його з кодом Java.
- Більшість рідних методів записуються на C. Механізм, використовуваний для інтеграції C-коду з Java-програмою, називається JNI (Java Native Interface - native-інтерфейс Java).

Native-методи

```
/ / Простий приклад, який використовує native-
    метод,
public class NativeDemo {
int i;
public static void main (String args [])
{NativeDemo ob = new NativeDemo ();
ob.i = 10;
System.out.println ("Цей ob.i перед native-
    методом:" + ob.i);
ob.testO;/ / виклик native-метода
System.out.println ("Цей ob.i після native-методу:"
    + ob.i);}
/ / Оголосити native-метод public native
void test ();
/ / Завантажити DLL, який містить static-метод
static {
System.loadLibrary ("NativeDemo");}}
```

- Зауважимо, що метод test () оголошений як native і не має тіла. Він буде реалізований на C.
- Зверніть також увагу на блок static. Як пояснювалося раніше, static-блок виконується тільки один раз, коли програма починає виконуватися (або, більш точно, коли його клас вперше завантажується).
- В даному випадку він використовується для завантаження DLL (Dynamic Link Library)-бібліотеки, яка містить native-реалізацію методу test ().
- Бібліотека завантажується методом loadLibrary (), який є частиною класу system. Ось його загальна форма:

  static void loadLibrary (String filename)

- Тут filename - рядок, який специфікує ім'я файлу, що містить бібліотеку.

Проблеми native-методів

- **Потенційний ризик безпеки.** Оскільки native-метод виконує фактичний машинний код, він може отримувати доступ до будь-якої частини хост-системи. Тобто native-код не обмежений середовищем виконання Java. Це може привести до зараження вірусом, наприклад. З цієї ж при чині native-методи не можуть використовувати аплети. Завантаження DLL-файлів може бути обмежена і підпорядкована схваленню керівника служби безпеки.
- **Втрата мобільності.** Оскільки native-код міститься в DLL-файлі, він повинен бути присутнім на машині, що виконує програму Java. Далі, так як будь-native-метод залежить від CPU і операційної системи, кожен такий DLL-файл неминуче нестерпний. Таким чином, додаток Java, яке використовує native-методи, буде здатне виконатися тільки на машині, де був встановлений відповідний DLL-файл.