Входни/изходни потоци

Поток е подредена последователност от байтове. Файлът съхранява данните. Пакетът java.io **съдържа класове, чрез които се дефинират потоци с определени характеристики.**

Видове потоци:

1. Входни и изходни потоци

а) входен поток – за четене на информация;

б) изходен поток – за запис на информация.

2. Символни и двоични потоци

а) символен поток (Reader**,** Writer**) – последователност от 16-битови Unicode символи;**

б) двоичен поток (InputStream**,** OutputStream**) съдържа 8-битови байтове с двоични данни (звуци и изображения).**

3. Поток с данни и поток за обработка на данни

а) поток с данни – напр. файл върху диск, низ в паметта;

б) поток за обработка (филтриране) на данни.

Операциите с потоци предизвикват изключението IOException**.**

Йерархия на входно/изходните класове в Java

Object
- InputStream
  - ObjectInputStream
  - FileInputStream
- OutputStream
  - FilterOutputStream
    - PrintStream
  - ObjectOutputStream
  - FileOutputStream
- Reader
  - BufferedReader
  - InputStreamReader
    - FileReader
- Writer
  - BufferedWriter
  - PrintWriter
  - OutputStreamWriter
    - FileWriter

Стандартен вход/изход

1. System.in **– стандартен входен поток (клавиатура).**
2. System.out **– стандартен изходен поток (монитор).**
3. System.err **– стандартен поток на грешките (монитор).**

Обектите in**,** out **и** err **от класа** System **са декларирани** public **и** static**; те са достъпни директно чрез класа** System**. Трите стандартни потока са създават и отварят автоматично.**

System.out **и** System.err **са от тип** PrintStream **(двоичен поток), който конвертира обектите и примитивните типове данни в текст. Използват се методите** print **и** println**.**

System.in **е от тип** InputStream **(двоичен поток), който обикновено се организира като поток с по-полезни характеристики чрез декларациите:**

```
BufferedReader stdin;
stdin = new BufferedReader (new InputStreamReader (System.in));
```

Създава се обект от тип InputStreamReader**, който конвертира двоичния входен поток в символен входен поток. Този обект се трансформира в обект от тип** BufferedReader**, който притежава метода** readLine **– получаваме цял ред от символи с една операция. Ако входният ред съдържа няколко стойности, те трябва да се разделят, като се използва класът** StringTokenizer**.**

Текстови файлове

Текстов файл – всеки байт се разглежда като един символ.

Запис в текстови файлове

Клас FileWriter **(**java.io**) – запис в текстови файлове.**

public FileWriter(String fileName) throws IOException

Установява връзка между програмата и файла.

Клас BufferedWriter **(**java.io**) – записва текст в изходен текстов файл като буферира символите.**

public BufferedWriter(Writer out)

Създава буфериран изходен символен поток с изходен буфер с подразбиращ се размер.

Клас PrintWriter **(**java.io**) – отпечатва форматирано представяне на обектите в изходен текстов поток; реализира всичките методи за печат от** PrintStream**.**

public PrintWriter(OutputStream out)

Създава нов PrintWriter **от съществуващ** OutputStream **без автоматично изчистване на редовете.**

public void print(String s)

Отпечатва низа s**.**

public void println(String s)

Отпечатва низа s **и тогава прекъсва реда.**

public void close()

Затваря потока. Прехвърля IOException **при В/И грешка.**

Запис в текстов файл

```
FileWriter fw = new FileWriter (filename);
BufferedWriter bw = new BufferedWriter (fw);
PrintWriter outFile = new PrintWriter (bw);
```

Класът FileWriter **изисква името на файла и полученият обект конструира обект от тип** BufferedWriter**, който осигурява буферни свойства на потока. Полученият обект конструира обект от тип** PrintWriter**, който притежава методите** print **и** println**.**

Пример: Създаване на текстов файл, който представлява опис на склад. Всеки ред от файла съдържа име на артикул, количество налични стоки и единична цена.

```
// InventoryItem.java
// Класът InventoryItem съдържа информацията:
// име на артикул, количество налични стоки и единична цена.
import java.io.*;
import java.util.StringTokenizer;
public class InventoryItem {
   private String name;                    // име на артикул
   private int units;                      // количество
   private float price;                    // цена
```

```
  // Конструктор без параметри
  public InventoryItem () {
     InputStreamReader isr = new InputStreamReader(System.in);
     BufferedReader stdin = new BufferedReader (isr);
     try {
        System.out.print ("Име на артикул: ");
        name = stdin.readLine ();
        System.out.print ("Количество: ");
        units = Integer.parseInt(stdin.readLine ());
        System.out.print ("Цена: ");
        price = Float.parseFloat (stdin.readLine ());
     } catch (IOException exception) {
        System.out.println(exception);
     }
  }
```

```
   // Конструктор с един параметър низ.
   public InventoryItem (String line) {
      StringTokenizer tokenizer;
      tokenizer = new StringTokenizer (line);
      name = tokenizer.nextToken();
      try {
         units = Integer.parseInt (tokenizer.nextToken());
         price = Float.parseFloat(tokenizer.nextToken());
      } catch (NumberFormatException exception) {
         System.out.println ("Редът се игнорира: ");
         System.out.println (line);
      }
   }
   // Символно представяне на InventoryItem
   public String toString () {
     return name+"\t"+units+"\t"+price;
   }
}
```

```
// CreateInventory.java
// Създава текстов файл с пет реда данни за артикулите в склада.
// Проверката за изключения е елиминирана.
import java.io.*;
import InventoryItem;
class CreateInventory {
   public static void main(String[] args) throws IOException {
      final int MAX = 5;
      InventoryItem item;
      String file = "inventory.dat";
      FileWriter fw = new FileWriter (file);
      BufferedWriter bw = new BufferedWriter (fw);
      PrintWriter outFile = new PrintWriter (bw);
      for(int line=1; line<=MAX; line++) {
         item = new InventoryItem ();
         outFile.println (item);
      }
      outFile.close();
      System.out.println ("Изходен файл: " + file);
   }
}
```

Четене от текстови файлове

Class FileReader (java.io) **– четене от текстови файлове**

public FileReader(String fileName) throws FileNotFoundException

Установява връзка между програмата и файла.

Клас BufferedReader (java.io) **– чете текст от входен символен поток като буферира символите.**

public BufferedReader(Reader in)

Създава буфериран входен символен поток с подразбиращ се размер на входния буфер.

public String readLine() throws IOException

Чете един текстов ред.

void close() throws IOException

Затваря потока.

Клас InputStreamReader (java.io) **– мост между байтови и символни потоци: чете байтове и ги преобразува в символи.**

public InputStreamReader(InputStream in)

Създава InputStreamReader **с подразбиращото се декодиране на символите.**

public int read(char[] cbuf,int off,int len) throws IOException

Чете символи в масив cbuf **с отместване** off **и максимален брой** len **на символи за четене. Връща броя на прочетените символи или -1 при достигане край на файл.**

Четене от текстов файл

```
FileReader fr = new FileReader (filename);
BufferedReader inFile = new BufferedReader (fr);
```

Класът FileReader **изисква името на файла и полученият обект конструира обект от тип** BufferedReader**, който притежава метода** readLine **за четене на цял символен ред в една операция.**

Пример: Четене от текстов файл, който представлява опис на склад. Всеки ред от файла съдържа име на артикул, количество налични стоки и единична цена.

```
// Inventory.java
// Чете данните за артикулите в склад от текстов файл и ги отпечатва.
import InventoryItem;
import java.io.*;
public class Inventory {
  public static void main (String[] args) {
    InventoryItem item;
    String line, file = "inventory.dat";
    int count = 0;
```

```
    try {
      FileReader fr = new FileReader (file);
      BufferedReader inFile = new BufferedReader(fr);
      line = inFile.readLine();
      while (line != null) {
        count++;
        item = new InventoryItem (line);
        System.out.println (item);
        line = inFile.readLine();
      }
      inFile.close();
      System.out.println ("Брой артикули: " + count);
    }
    catch (FileNotFoundException exception) {
      System.out.println ("Файлът " + file + " не е намерен.");
    }
    catch (IOException exception) {
      System.out.println (exception);
    }
  }
}
```

Двоични файлове

Двоичен файл – съдържа двоичното представяне на данните (последователност от 0 и 1).

Запис в двоични файлове

Клас FileOutputStream **(**java.io**) – запис на данни във файл.**

public FileOutputStream(String name) throws FileNotFoundException

Създава изходен файлов поток за запис във файл с име name**.**

Клас ObjectOutputStream **(**java.io**) – записва примитивни типове данни и графики на Java обекти в** OutputStream**. Само обекти, които поддържат интерфейса** java.io.Serializable**, могат да бъдат записани в потоци.**

```
public ObjectOutputStream(OutputStream out) throws IOException
```

Създава ObjectOutputStream**, който записва в изходния поток** out**.**

```
public final void writeObject(Object obj) throws IOException
```

Записва обекта obj **в** ObjectOutputStream**.**

```
public void close() throws IOException
```

Затваря потока.

Запис в двоичен файл

```
FileOutputStream outFile = new FileOutputStream (filename);
ObjectOutputStream outObject = new ObjectOutputStream (outFile);
```

Класът FileOutputStream **изисква името на файла и полученият обект конструира обект от тип** ObjectOutputStream**, който притежава метода** writeObject **за запис на обект в потока.**

Пример: Създаване на двоичен файл, който представлява опис на склад. Всеки ред от файла съдържа име на артикул, количество налични стоки и единична цена.

```
// InventoryItemB.java
// Класът InventoryItemB съдържа информацията:
// име на артикул, количество налични стоки и единична цена.
// Реализира интерфейса Serializable.
import java.io.*;
import java.util.StringTokenizer;
public class InventoryItemB implements Serializable {
  private String name;                  // име на артикул
  private int units;                    // количество
  private float price;                  // цена
```

```
  // Конструктор без параметри.
  public InventoryItemB () {
    InputStreamReader isr = new InputStreamReader(System.in);
    BufferedReader stdin = new BufferedReader (isr);
    try {
      System.out.print ("Име на артикул: ");
      name = stdin.readLine ();
      System.out.print ("Количество: ");
      units = Integer.parseInt(stdin.readLine ());
      System.out.print ("Цена: ");
      price = Float.parseFloat (stdin.readLine ());
    }
    catch (IOException exception) {
      System.out.println(exception);
    }
  }
```

```
  // Конструктор с три параметъра.
  public InventoryItemB (String itemName, int numUnits, float coast) {
    name = itemName;
    units = numUnits;
    price = coast;
  }
```

```
  // Конструктор с един параметър низ.
  public InventoryItemB (String line) {
    StringTokenizer tokenizer;
    tokenizer = new StringTokenizer (line);
    name = tokenizer.nextToken();
    try {
      units = Integer.parseInt (tokenizer.nextToken());
      price = Float.parseFloat (tokenizer.nextToken());
    }
    catch (NumberFormatException exception) {
      System.out.println ("Игнориран ред: ");
      System.out.println (line);
    }
  }
  // Връща символното представяне на InventoryItemB
  public String toString () {
    return name + "\t" + units + "\t" + price;
  }
}
```

```
// WriteBinary.java
// Създава двоичен файл с данни за артикули в склад.
import java.io.*;
import InventoryItemB;
class WriteBinary extends Object {
  public static void main(String[] args) {
    final int MAX = 3;
    InventoryItemB item;
    String file = "inv.dat";
```

```
    try {
      FileOutputStream outFile = new FileOutputStream (file);
      ObjectOutputStream outObject = new ObjectOutputStream (outFile);
      for(int obj=1; obj<=MAX; obj++) {
        item = new InventoryItemB ();
        try {
          outObject.writeObject (item);
        }
        catch (IOException exception) {
          System.out.println (exception);
        }
      }
      outObject.close();
      System.out.println ("Изходен двоичен файл: " + file);
    }
    catch (Exception exception) {
      System.out.println (exception);
    }
  }
}
```

Един обект може да „живее“ отделно от изпълняващата се програма, която го създава. Java притежава механизъм за създаване на обекти, които съществуват продължително време – нарича се сериализиране на обекти. Обектът се представя като последователност от байтове, които могат да бъдат запазени във файл или транспортирани до друг компютър. Интерфейсът Serializable **служи като флаг, че обектите от този тип могат да бъдат предавани на части. За тази цел се извиква методът** writeObject **на** ObjectOutputStream**. За да се премахне сериализацията за един обект, извиква се методът** readObject **на** ObjectInputStream**.**

Четене от двоични файлове

Клас FileInputStream **(**java.io**) – получава входни байтове от файл.**

public FileInputStream(String name) throws FileNotFoundException

Създава FileInputStream **чрез отваряне на връзка към действителен файл с име на път name във файловата система.**

Клас ObjectInputStream **(**java.io**) – отменя сериализацията на примитивни данни и обекти, които са записани чрез използване на** ObjectOutputStream**. Само обектите, които поддържат интерфейсите** java.io.Serializable **или** java.io.Externalizable **могат да бъдат прочетени от потока.**

public ObjectInputStream(InputStream in) throws IOException,
StreamCorruptedException

Създава ObjectInputStream**, който чете от входния поток** in**.**

public final Object readObject() throws OptionalDataException,
ClassNotFoundException, IOException

Чете обект от ObjectInputStream**.**

public void close() throws IOException

Затваря входния поток.

Четене от двоичен файл

```
FileInputStream inFile = new FileInputStream (filename);
ObjectInputStream inObject = new ObjectInputStream (inFile);
```

Класът FileInputStream **изисква името на файла и полученият обект конструира** ObjectInputStream**, който притежава метода** readObject **за четене на обект от входния поток.**

Пример: Четене от двоичен файл, който представлява опис на склад. Всеки ред от файла съдържа име на артикул, количество налични стоки и единична цена.

```
// ReadBinary.java
// Чете данни за артикулите в склад от двоичен файл и ги отпечатва.
import InventoryItemB;
import java.io.*;
public class ReadBinary extends Object {
  public static void main (String[] args) {
    String file = "inv.dat";
    InventoryItemB obj;
    int count = 0;
    try {
      FileInputStream inFile = new FileInputStream (file);
      ObjectInputStream inObject = new ObjectInputStream (inFile);
      try {
        while (true) {
          obj=(InventoryItemB)inObject.readObject();
          count++;
          System.out.println (obj);
        }
      }
      catch (IOException e) {
        inObject.close();
        System.out.println ("Брой артикули: " + count);
      }
    }
    catch (Exception exception) {
      System.out.println(exception);
    }
  }
}
```

Задачи:

1. **Проектирайте и реализирайте програма, която се използва в болница за анализ на потока от пациенти в спешното отделение. Входният файл съдържа цели числа, които представят броя на пациентите, които влизат в спешното отделение на всеки час на деня за четири седмици. Прочетете информацията и я запазете в тримерен масив. Анализирайте информацията, като сравните общия брой пациенти за седмица, за ден от седмицата и за час от деня.**
2. **Проектирайте и реализирайте приложение с графичен потребителски интерфейс, което дава възможност на потребителя да пише текст в текстова област и при натискане на бутон да запише информацията в текстов файл.**
3. **Проектирайте и реализирайте програма, която изчислява броя на препинателните знаци в текстов файл. Покажете диаграма, която показва колко пъти се среща всеки препинателен знак.**