Пакет java.io и работа с ресурсами

Рассматриваемые вопросы

1. Потоки данных
2. Классы пакета java.io
3. Сериализация

Потоки данных

java.io предназначен для чтения и записи данных в ресурс:

1. файл;
2. при работе с сетевым подключением;
3. System.err, System.in, System.out;
4. канал (pipe);
5. при работе с буфером;
6. другие источники (например, подключение к интернету).

Классы InputStream и OutputStream

Классы InputStream и OutputStream

Для работы с указанными источниками используются подклассы базового класса InputStream.

Класс OutputStream – это абстрактный класс, определяющий байтовый потоковый вывода. Наследники данного класса определяют куда направлять данные: в массив байтов, в файл или канал.

ByteArrayInputStream и ByteArrayOutputStream

Классы ByteArrayInputStream и ByteArrayOutputStream реализуют потоки для чтения и записи в массив байт.

ByteArrayInputStream и ByteArrayOutputStream

```
byte[] array1 = new byte[]{1, 3, 5, 7};
ByteArrayInputStream byteStream1 = new ByteArrayInputStream(array1);
int b;
while( (b=byteStream1.read())!=-1 ){
    System.out.println(b);
}

String text = "Hello world!";
byte[] array2 = text.getBytes();
ByteArrayInputStream byteStream2 = new ByteArrayInputStream(array2, 0, 5);
int c;
while((c=byteStream2.read())!=-1){
    System.out.println((char)c);
}
```

FileInputStream и FileOutputStream

Пример конструктора FileInputStream:

FileInputStream(String fileName) throws FileNotFoundException

Если файл не может быть открыт, то генерируется исключение FileNotFoundException.

FileInputStream и FileOutputStream

```
try (FileInputStream fin=new FileInputStream(“file.txt")) {
        int i=-1;
        while((i=fin.read())!=-1){
            System.out.print((char)i);
        }
} catch(IOException ex){
        System.out.println(ex.getMessage());
}
```

FileInputStream и FileOutputStream

```
String text = "Hello world!"; // строка для записи
try(FileOutputStream fos=new FileOutputStream("file.txt")) {
        // перевод строки в байты
        byte[] buffer = text.getBytes();
        fos.write(buffer, 0, buffer.length);
}  catch(IOException ex)  {
        System.out.println(ex.getMessage());
}
```

BufferedInputStream и BufferedOutputStream

Классы BufferedInputStream и BufferedOutputStream служат для буферизации потока информации.

Класс BufferedInputStream накапливает вводимые данные в специальном буфере без постоянного обращения к устройству ввода.

BufferedInputStream и BufferedOutputStream

```
String text = "Hello world!";
try(FileOutputStream out=new FileOutputStream("notes.txt");
        BufferedOutputStream bos = new BufferedOutputStream(out))  {
        byte[] buffer = text.getBytes();
        bos.write(buffer, 0, buffer.length);
}  catch(IOException ex) {
        System.out.println(ex.getMessage());
}
```

DataInputStream и DataOutputStream

Классы DataInputStream и DataOutputStream реализуют потоки для работы с остальными примитивными типами и со String.

Класс DataOutputStream представляет поток вывода и предназначен для записи данных примитивных типов, таких, как int, double и т.д. Для записи каждого из примитивных типов предназначен свой метод.

Класс DataInputStream действует противоположным образом - он считывает из потока данные примитивных типов.

ObjectInputStream и ObjectOutputStream и сериализация

ObjectInputStream и ObjectOutputStream позволяют работать с потоками для чтения/записи объектов с использованием механизма сериализации.

Сериализация представляет процесс записи состояния объекта в поток, соответственно процесс извлечения или восстановления состояния объекта из потока называется десериализацией.

ObjectInputStream и ObjectOutputStream и сериализация

```
class Person implements Serializable {
    public String name;
    public int age;
    public double height;
    public boolean married;

    Person(String n, int a, double h, boolean m){
        name=n;
        age=a;
        height=h;
        married=m;
    }
}
```

ObjectInputStream и ObjectOutputStream и сериализация

```
try(ObjectOutputStream oos =
            new ObjectOutputStream(new FileOutputStream("person.dat"))) {
        Person p = new Person("Джон", 33, 178, true);
        oos.writeObject(p);
} catch(Exception ex)  {
        System.out.println(ex.getMessage());
}
```

Сериализация

Сериализация – это процесс записи состояния объектов в поток вывода байтов.

В дальнейшем эти объекты можно восстановить в процессе десериализации.

Интерфейс Serializable

Средствами сериализации может быть сохранен и восстановлен только объект класса, реализующего интерфейс Serializable.

Переменные, объявленные как transient или static не сохраняются средствами сериализации.

При Java-сериализации используется свойство serialVersionUID, которое помогает справиться с разными версиями объектов в сценарии сериализации.

Интерфейс Serializable

```
import java.io.Serializable;
public class Person implements Serializable {
    private static final long serialVersionUID = 20100515;
    // ...
}
```

Вопросы для самоконтроля

1. Перечислите основные методы класс InputStream?
2. Что такое граф сериализации?
3. Какой паттерн проектирования использует java.io?
4. В чем назначение serialVersionUID?