Universidade Presbiteriana Mackenzie
Faculdade de Computação e Informática

Entrada e Saída de dados em Java (2)

NOTAS DE AULA

Linguagem de Programação II
2º semestre de 2015
Prof. Tomaz Mikio Sasaki
Versão 1.1

Objetivos

- Aprender o mecanismo de serialização de objetos em Java.
- Saber utilizar a serialização para salvar e recuperar objetos.

Referência

A referência para esta aula é o capítulo 1 de:

> HORSTMANN, C.S.; CORNELL, G. **Core Java, Volume II - Advanced Features** Prentice Hall, 2013.

Observação: As notas de aula são material de apoio para estudo e não têm o objetivo de apresentar o assunto de maneira exaustiva. Não deixe de ler o material de referência da disciplina.

Para reduzir o número de linhas de código, os exemplos apresentados omitem propositalmente a importação das classes. Para esta aula, a maior parte das classes pertencem ao pacote **java.io**. As principais IDEs Java possuem recursos para auxiliar a inclusão das importações das classes.

Aula passada

```
InputStream (implements Closeable)
|------ ByteArrayInputStream
|------ FileInputStream
|------ FilterInputStream
|         |------ BufferedInputStream
|         |------ DataInputStream (implements DataInput)
|         |------ LineNumberInputStream
|         `------ PushbackInputStream
|
|------ ObjectInputStream (implements ObjectInput, ObjectStreamConstants)
|------ PipedInputStream
|------ SequenceInputStream
`------ StringBufferInputStream
```

Aula passada (cont.)

```
OutputStream (implements Closeable, Flushable)
|------ ByteArrayOutputStream
|------ FileOutputStream
|------ FilterOutputStream
|         |------ BufferedOutputStream
|         |------ DataOutputStream (implements DataOutput)
|         `------ PrintStream (implements java.lang.Appendable, Closeable)
|
|------ ObjectOutputStream (implements ObjectOutput, ObjectStreamConstants)
`------ PipedOutputStream
```

Aula passada (cont.)

```
Reader (implements Closeable, java.lang.Readable)
|
|------ BufferedReader
|       |
|       `------ LineNumberReader
|
|------ CharArrayReader
|------ FilterReader
|       |
|       `------ PushbackReader
|
|------ InputStreamReader
|       |
|       `------ FileReader
|
|------ PipedReader
`------ StringReader
```

Aula passada (cont.)

```
Writer (implements java.lang.Appendable, Closeable, Flushable)
|------ BufferedWriter
|------ CharArrayWriter
|------ FilterWriter
|------ OutputStreamWriter
|       |
|       `------ FileWriter
|
|------ PipedWriter
|------ PrintWriter
`------ StringWriter
```

Aula passada (cont.)

- **InputStream** e **OutputStream** são orientados à leitura e escrita de **bytes**.
- **Reader** e **Writer** são orientados à leitura e escrita de **caracteres**.
- **BufferedReader** e **PrintWriter** conseguem ler e escrever **linhas inteiras** compostas por **strings**.

DataInputStream e DataOutputStream

Leitura e escrita de **tipos primitivos** representados em bytes de dados.

Comparação entre uso de DataOuput e Writer

```
public class AppDataOutput {

    public static void main(String[] args)
                throws FileNotFoundException, IOException {
        FileOutputStream fos = new FileOutputStream("dataoutput.txt");
        DataOutputStream dos = new DataOutputStream(fos);
        dos.writeShort(65);
        dos.writeShort(66);
        dos.writeShort(67);
        dos.writeShort(68);
        dos.close();

        FileWriter fw = new FileWriter("datawriter.txt");
        fw.append("65");
        fw.append("66");
        fw.append("67");
        fw.append("68");
        fw.close();
    }
}
```

Introdução

Como Java é uma linguagem orientada a objetos, é útil podermos persistir os próprios objetos em arquivo ou transmiti-los.

Classe Carro

Suponha que a nossa intenção seja a gravação e leitura de instâncias da classe **Carro** abaixo.

```
public class Carro  {
    private String marca;
    private String modelo;
    private int ano;
    public Carro(String marca, String modelo, int ano) {
        super();
        this.marca = marca;
        this.modelo = modelo;
        this.ano = ano;
    }
    public String toString() {
        return marca + ":" + modelo + ":" + ano;
    }
}
```

Gravação em arquivo texto

Exemplo de como salvar os dados das instâncias de **Carro** em um arquivo texto:

```
public class SalvarCarrosTxt {
    public static void main(String[] args) throws IOException {
        Carro c1 = new Carro("VW", "Up!", 2014);
        Carro c2 = new Carro("GM", "Corsa", 2009);
        Carro c3 = new Carro("Honda", "Fit", 2007);

        FileWriter fw = new FileWriter("carros.txt");
        PrintWriter pw = new PrintWriter(fw);
        pw.println(c1.toString());
        pw.println(c2.toString());
        pw.println(c3.toString());
        pw.close();
        fw.close();
    }
}
```

Leitura de dados em arquivo texto

Exemplo de como ler os dados das instâncias de **Carro** em um arquivo texto:

```
public class CarregarCarrosTxt {
    public static void main(String[] args) throws IOException {
        FileReader fr = new FileReader("carros.txt");
        BufferedReader br = new BufferedReader(fr);

        List<Carro> carros = new ArrayList<>();

        String linha;
        while ((linha = br.readLine()) != null) {
            String[] v = linha.split(":");
            Carro c = new Carro(v[0], v[1], Integer.parseInt(v[2]));
            carros.add(c);
        }

        for(Carro carro: carros) {
            System.out.println("Dados do carro: " + carro.toString());
        }
    }
}
```

Gravação e leitura de objetos serializados

As classes **ObjectInputStream** e **ObjectOutputStream** podem ser utilizadas para ler e escrever instâncias de objetos em arquivos.

Para isso, a classe das instâncias que serão gravadas e lidas deve implementar a interface *Serializable*.

Classe Carro serializada

Para poder salvar e recuperar objetos é necessário que a classe implemente a interface *Serializable*. Esta é uma interface somente de **marcação** (ou seja, não implica na implementação de nenhum método específico).

```
public class Carro implements Serializable {
    private String marca;
    private String modelo;
    private int ano;
    public Carro(String marca, String modelo, int ano) {
        super();
        this.marca = marca;
        this.modelo = modelo;
        this.ano = ano;
    }
    public String toString() {
        return marca + ":" + modelo + ":" + ano;
    }
}
```

Gravação de objetos serializados em arquivos

```
public class SalvarCarrosObj {
    public static void main(String[] args) throws IOException {
        Carro c1 = new Carro("VW", "Up!", 2014);
        Carro c2 = new Carro("GM", "Corsa", 2009);
        Carro c3 = new Carro("Honda", "Fit", 2007);


        FileOutputStream fos = new FileOutputStream("carros.dat");
        ObjectOutputStream oos = new ObjectOutputStream(fos);
         oos.writeObject(c1);
         oos.writeObject(c2);
         oos.writeObject(c3);

        oos.close();
        fos.close();
    }
}
```

Leitura de objetos serializados em arquivos

```
public class CarregarCarrosObj {

    public static void main(String[] args) throws ClassNotFoundException,
            IOException {
        FileInputStream fis = new FileInputStream("carros.dat");
        ObjectInputStream ois = new ObjectInputStream(fis);

        List<Carro> carros = new ArrayList<>();

        while (fis.available() != 0) {
             Carro c = (Carro) ois.readObject();
            carros.add(c);
        }
        ois.close();
        fis.close();

        for (Carro carro : carros) {
            System.out.println("Dados do carro: " + carro.toString());
        }
    }
}
```

Uso de *Serializable* em classes associadas

Se a classe possui atributos que são referências para instâncias de outras classes, estas outras classes também deverão implementar esta interface.

Exemplo:

```
public class Empresa implements Serializable {
     private Empresario empresario;
    // outros atributos da classe
    // métodos
}

public class Empresario implements Serializable {
    // atributos e métodos
}
```

Problema de objetos referenciados mais de uma vez

Quando um mesmo objeto é referenciado mais de uma vez pelos atributos dos objetos que são serializados, em teoria haveria o risco do objeto ser persistido duas vezes. No entanto, em Java há um mecanismo que marca cada instância com um número identificador e verifica se a referência é de um objeto já serializado ou desserializado anteriormente.

Versão das classes

Um problema que pode ocorrer quando salvamos os objetos serializados é não sermos mais capazes de recuperar os objetos corretamente após a aplicação sofrer uma evolução. Se as classes serializadas tiverem, por exemplo, o nome de um atributo alterado, não será mais possível recuperar corretamente as instâncias gravadas utilizando uma versão anterior da classe.

Versão das classes (cont.)

Para checar se o processo de leitura está utilizando a mesma versão utilizada na gravação, o compilador calcula um número denominado *serialVersionUID*.

Quando uma classe sofre uma alteração, o cálculo deste número resulta em um valor diferente.

Versão das classes (cont.)

Para saber qual é o valor do *serialVersionUID* de uma classe, pode-se utilizar o utilitário **serialver** que deve ser executado na linha de comando. Para executá-lo para a classe **Carro**, por exemplo, devemos digitar na linha de comando:

```
serialver Carro
```

Caso a classe sofra uma modificação e o programador saiba que não irá implicar em incompatibilidade com uma versão anterior (por exemplo, quando há somente a inclusão de um novo método), o programador pode indicar isto explicitamente incluindo um atributo com o valor de *serialVersionUID* da versão anterior da classe:

```
// nova versão da classe Carro que é compatível com a versão anterior
public class Carro implements Serializable {

    // valor de serialVersionUID da versão anterior da classe
    public static final long serialVersionUID = -1814239825517340645L;

    // atributos e métodos da classe Carro
}
```

Bom estudo!