UPskill – JAVA + .NET

Programação Orientada a Objetos – Ficheiros Binários

Adaptado de Donald W. Smith (TechNeTrain)

Objetivos

- Conhecer os formatos de ficheiros de texto e binário
- Ler e escrever objetos usando serialização

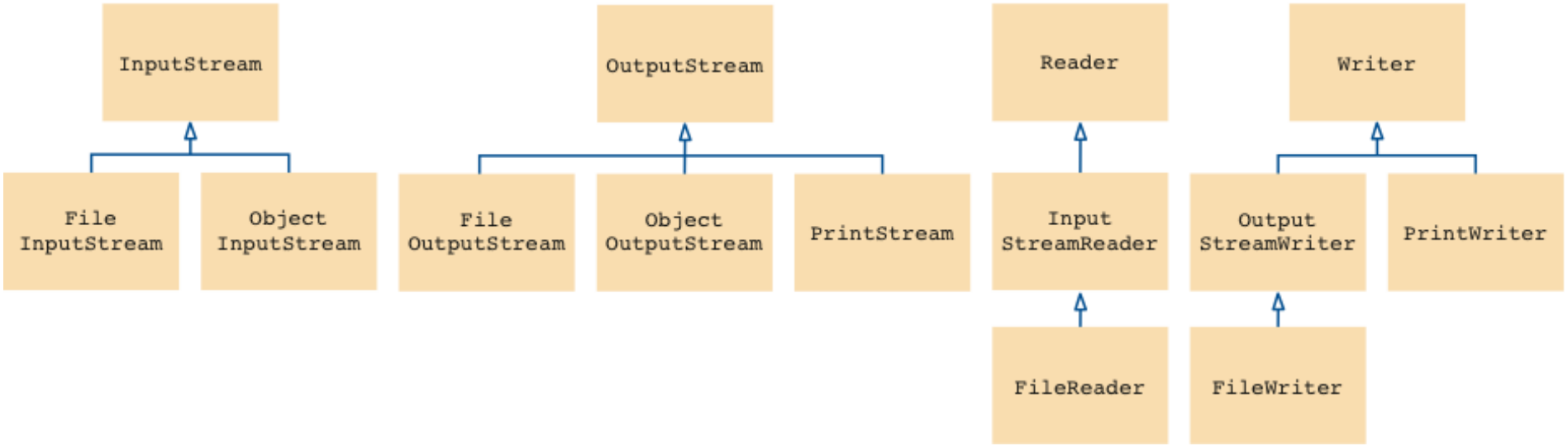
Conteúdos

- *Readers, Writers* e *Streams*
- Entrada e Saída Texto/Binárias
- *Object Streams*

Readers, Writers e Streams

- Duas formas de armazenamento:
  - *Formato texto:* forma legível, sob a forma de uma sequência de caracteres
    - Ex. integer 12345 armazenado como '1' '2' '3' '4' '5'
    - Mais conveniente para humanos: facilita as operações de entrada e saída
    - *Readers* e *Writers* lidam com dados sob a forma de texto
  - *Formato binário:* dados são representados em *bytes*
    - Mais compacto e mais eficiente
    - *Streams* lidam com dados binários

Readers, Writers e Streams (cont.)



Readers, Writers e Streams (cont.)

- `Reader`, `Writer` e suas subclasses destinam-se ao processamento de texto (entrada e saída)
- A classe `Scanner` pode ser mais indicada que a classe `Reader`
- A classe `ObjectOutputStream` pode gravar objetos para um ficheiro binário
- A classe `ObjectInputStream` pode ler objetos de um ficheiro binário

Serialização

- Processo no qual a instância de um objeto é transformada numa sequência de bytes
- Permite implementar a persistência dos objetos
- Pode ser usado para enviar objetos através de uma rede ou gravá-los em ficheiro
- Para que possa ser aplicada aos objetos de uma classe, essa classe deve implementar a interface Serializable
  - Trata-se de uma interface de marcação, pois não define qualquer método, servindo apenas para que a JVM saiba que a classe pode ser serializada

Object Streams – Serialização

- Todas as variáveis de instância são gravadas:

```
//class BankAccount or superclass implements Serializable
BankAccount b = ...;

ObjectOutputStream out = new ObjectOutputStream(
   new FileOutputStream("bank.dat"));

out.writeObject(b);

out.close();
```

Serialização – Exemplo

```
import java.io.Serializable;

public class Employee implements Serializable{
    private int employeeId;
    private String employeeName;
    private String department;

    public int getEmployeeId() { return employeeId;}
    public void setEmployeeId(int employeeId) {
        this.employeeId = employeeId;
    }

    public String getEmployeeName() { return employeeName;}
    public void setEmployeeName(String employeeName) {
        this.employeeName = employeeName;
    }

    public String getDepartment() { return department;}
    public void setDepartment(String department) {
        this.department = department;
    }
}
```

Serialização – Exemplo (cont.)

```
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectOutputStream;

public class SerializeMain {
   public static void main(String[] args) {

      Employee emp = new Employee();
      emp.setEmployeeId(101);
      emp.setEmployeeName("Arpit");
      emp.setDepartment("CS");

      try {
         FileOutputStream fileOut = new FileOutputStream("employee.ser");

         ObjectOutputStream outStream = new ObjectOutputStream(fileOut);
         outStream.writeObject(emp);

         //use finally ?
         outStream.close();
         fileOut.close();
      } catch(IOException i) {
         i.printStackTrace();
      }
   }
}
```

Object Streams – Desserialização

- O método readObject devolve uma referência para Object
- É necessário conhecer os tipos dos objetos gravados e fazer a respetiva conversão (*cast*)

```
ObjectInputStream in = new ObjectInputStream(
   new FileInputStream("bank.dat"));
BankAccount b =(BankAccount) in.readObject();
```

- O método readObject pode lançar uma exceção do tipo ClassNotFoundException caso alguma classe não esteja marcada com a interface Serializable
  - *É uma checked exception* ⇒ é necessário tratar

Desserialização – Exemplo (cont.)

```
import java.io.IOException;
import java.io.ObjectInputStream;

public class DeserializeMain {
   public static void main(String[] args) {
      Employee emp = null;

      try {
         FileInputStream fileIn =new FileInputStream("employee.ser");

         ObjectInputStream in = new ObjectInputStream(fileIn);
         emp = (Employee) in.readObject();

         //use finally ?
         in.close();
         fileIn.close();
      } catch(IOException i) {
         i.printStackTrace();
         return;
      } catch(ClassNotFoundException c) {
         System.out.println("Employee class not found");
         c.printStackTrace();
         return;
      }

      System.out.println("Deserialized Employee...");
      System.out.println("Emp id: " + emp.getEmployeeId());
      System.out.println("Name: '"+ emp.getEmployeeName());
      System.out.println("Department: " + emp.getDepartment());
   }
}
```

Serialização ArrayList

- Serialização

```
ArrayList<BankAccount> a =
   new ArrayList<BankAccount>();

// Adicionar várias instâncias de BankAccount em a
ObjectOutputStream out = new ObjectOutputStream(
   new FileOutputStream(”accounts.dat"));
out.writeObject(a);
```

- Desserialização

```
ObjectInputStream in = new ObjectInputStream(
   new FileInputStream("bank.dat"));

ArrayList<BankAccount> a =
   (ArrayList<BankAccount>) in.readObject();
```

Interface Serializable

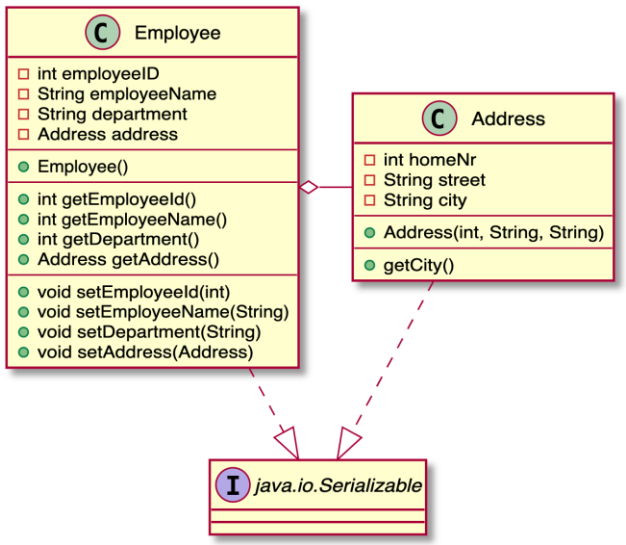
- Os objetos que são escritos num *object stream* devem pertencer a uma classe que implementa a interface `Serializable`:

```
class BankAccount implements Serializable
{
   …
}
```

- A interface `Serializable` não tem métodos

Serialização com referências

- Quando um objeto que contém referências para outros objetos é serializado, a JVM serializa todos os objetos relacionados
- *E.g.*, se um objeto `Employee` contém uma referência para um objeto do tipo `Address`, quando se serializa o objeto `Employee` o objeto `Address` também será serializado



Serialização com referências (cont.)

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;

public class SerializeDeserializeMain {
   public static void main(String[] args) {
      Employee emp = new Employee();

      emp.setEmployeeId(101);
      emp.setEmployeeName("Arpit");
      emp.setDepartment("CS");

      Address address=new Address(88,"MG road","Pune");
      emp.setAddress(address);

      try {
         FileOutputStream fileOut = new FileOutputStream("employee.ser");
         ObjectOutputStream outStream = new ObjectOutputStream(fileOut);

         outStream.writeObject(emp);

         outStream.close();
         fileOut.close();
      } catch(IOException i) {
         i.printStackTrace();
      }
```

Desserialização com referências

```
        emp = null;
        try {
           FileInputStream fileIn =new FileInputStream("employee.ser");
           ObjectInputStream in = new ObjectInputStream(fileIn);

           emp = (Employee) in.readObject();

           in.close();
           fileIn.close();
        } catch(IOException i) {
           i.printStackTrace();
           return;
        } catch(ClassNotFoundException c) {
           System.out.println("Employee class not found");
           c.printStackTrace();
           return;
        }

        System.out.println("Deserialized Employee...");
        System.out.println("Emp id: " + emp.getEmployeeId());
        System.out.println("Name: " + emp.getEmployeeName());
        System.out.println("Department: " + emp.getDepartment());

        address=emp.getAddress();
        System.out.println("City : "+address.getCity());
    }
}
```

Serialização – supressão de atributo

- Caso não se pretenda serializar um atributo específico de um determinado objeto, basta marcá-lo como `transient`
  - o objeto serializado não conterá a informação referente ao atributo `transient`
- *E.g.*, se pretendermos excluir o atributo `Address` da serialização dos objetos de `Employee`

```
private transient Address address;
```

  - Após a desserialização, se tentarmos aceder ao atributo `address` será lançada uma exceção `NullPointerException`