Programação I

Persistência: Serialização/Deserialização de objetos

Jorge Roberto Trento
Bacharel em Ciências da Computação - UNOESC
Especialização em Ciências da Computação - UFSC
Formação Pedagógica - Formadores de Educação Profissional – UNISUL
Especialização em Ensino Superior – FIE
Especialização em Gestão de Tecnologia da Informação – FIE

Introdução

- Persistência é a capacidade de um programa guardar dados ou objetos por um tempo maior do que o tempo em que o mesmo permanece em execução.
- Até o momento, todos os objetos que criamos permanecem na memória RAM e são perdidos quando a aplicação é encerrada.
- A persistência consiste em salvar dados por tempo indeterminado, de modo que estes possam, em outro momento, ser recuperados.

Exemplos de objetos que poderiam ser persistidos:

- Clientes, produtos, fornecedores e funcionários, em um sistema de loja;
- O estado de um jogo;
- As preferências do usuário em um aplicativo.

Guardando o estado de um objeto

- Para guardar o estado de um objeto, podemos utilizar as seguintes técnicas:
  - **serialização/desserialização:** consiste em “congelar” o estado dos objetos e depois reconstituí-los;
  - **arquivos:** podemos gravar os valores de cada atributo em um arquivo de texto simples (ou de bytes);
  - **banco de dados:** podemos utilizar bancos de dados relacionais ou orientados a objetos, mas não abordaremos este assunto aqui.

Serialização e Desserialização

- Serializar um objeto significa capturar seu estado e transformá-lo em uma cadeia de bytes,os quais podem ser armazenados em um arquivo ou transmitidos em uma rede;

- O processo inverso, chamado de desserialização, consiste em recuperar o conjunto de bytes recompor o objeto original.

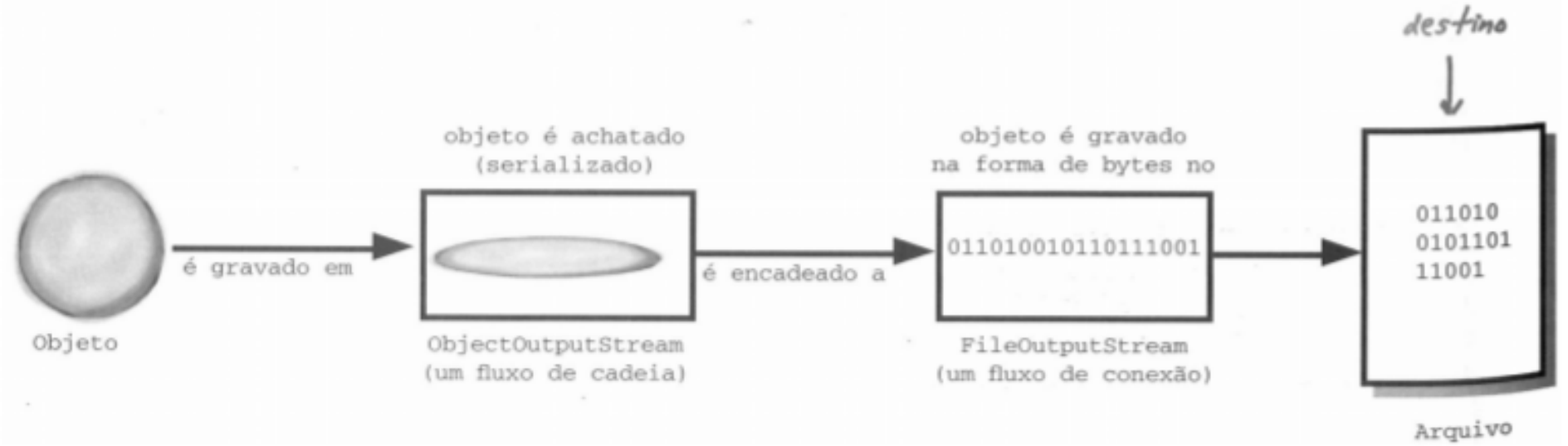
Serialização

- A serialização envolve o estabelecimento de um stream (ou fluxo) de cadeia capaz de converter objetos em dados que podem ser gravados ou transmitidos.
- Mas, para que a gravação ou transmissão se efetive, é necessário o auxílio de um stream (ou fluxo) de conexão.
- Um fluxo de conexão é um objeto de transmissão de dados que conecta uma origem e um destino (pode ser um arquivo ou soquete de rede).
- Para gravar e recuperar objetos serializados em arquivos, vamos utilizar como fluxo de conexão as classes:
  - **FileOutputStream:** stream de arquivo que permite a gravação em disco;
  - **FileInputStream:** stream que permite a leitura de um arquivo em disco.

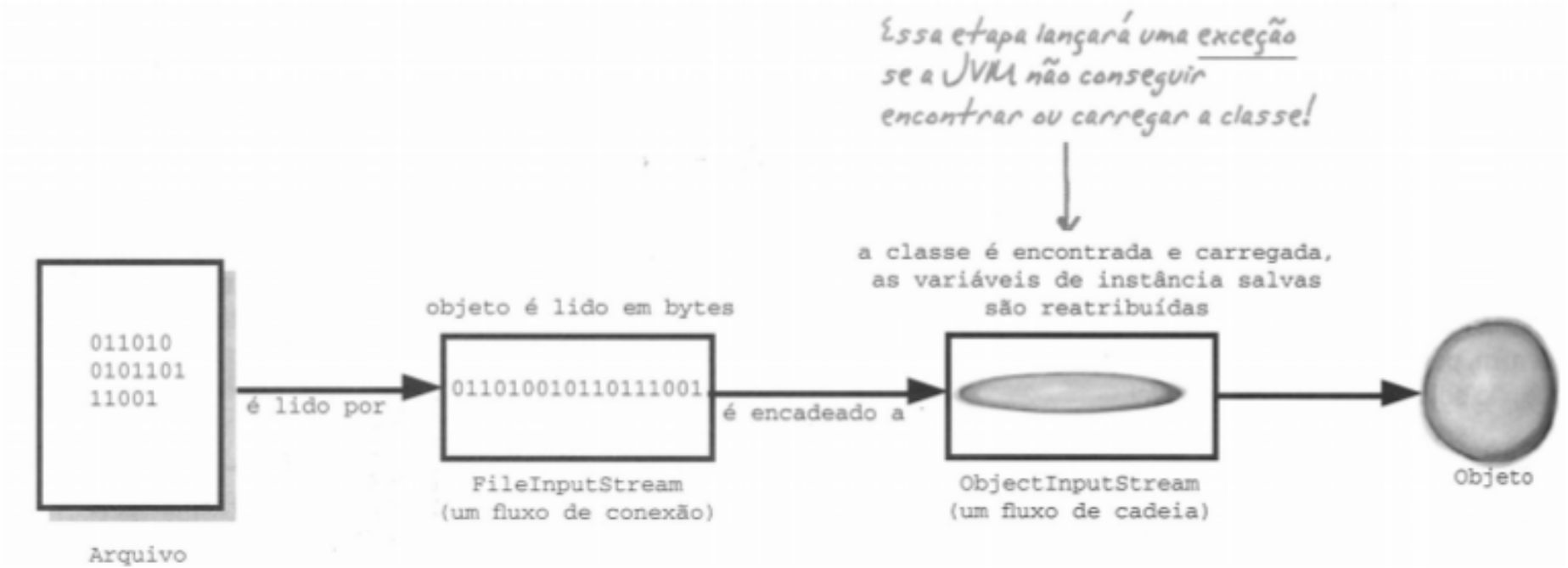
Serialização

- Para converter objetos em um fomato de dados compatível com o fluxo de conexão descrito anteriormente, vamos utilizar as seguintes classes:
- **ObjectOutputStream:** através do método writeObject, cria uma cadeia de bytes a partir de um objeto (serialização) e a insere em um fluxo de conexão;
- **ObjectInputStream:** recebe bytes de um fluxo de conexão e recupera um objeto, por meio do método readObject (desserialização).

Processo de Serialização



Processo de Desserialização



Gravando um objeto serializado em disco

1) Criar um objeto FileOutputStream:

```
FileOutputStream outFile = new
FileOutputStream("file.ser");
```

2) Criar um objeto ObjectOutputStream:

```
ObjectOutputStream os = new
ObjectOutputStream(outFile);
```

OBS: Encadeando um fluxo a outro.

Gravando um objeto serializado em disco

3) Gravar o (s) objeto(s):

os.writeObject(obj1);

os.writeObject(obj2);

OBS: Serializa o objeto referenciado por obj1 e obj2 e grava no arquivo file.ser.

4) Fechar o ObjectOutputStream:

os.close();

Recuperando um objeto serializado

- Para recuperar um objeto serializado que se encontra em um arquivo em disco, devemos seguir estas etapas:

1) Criar um objeto FileInputStream:

FileInputStream inFile = new FileInputStream("file.ser");

OBS: Se o arquivo file.ser não existe, será lançada uma FileNotFoundException.

2) Criar um objeto ObjectOutputStream:

ObjectInputStream os = new ObjectInputStream(inFile);

OBS: Encadeando um fluxo a outro.

Recuperando um objeto serializado

3) Ler o(s) objeto(s):

```
TipoObjeto obj1 = (TipoObjeto)os.readObject();
TipoObjeto obj2 = (TipoObjeto)os.readObject();
```

OBS: Cada execução de readObject lerá o próximo objeto do fluxo. Se tentarmos ler mais objetos do que existem, teremos uma exceção.

4) Fechar o ObjectOutputStream:

```
os.close();
```

Interface Serializable

- Para que um objeto possa ser serializado em Java, sua classe deve ser declarada como serializável. Isso é feito implementando-se a **interface java.io.Serializable**.
- Esta interface não possui nenhum método, funcionando apenas como uma marcação para informar à JVM que a referida classe pode ser serializada.

```
import java.io.Serializable;

    public class Aluno implements Serializable {

        ...

    }
```

- É importante destacar que quando uma superclasse implementa a interface “Serializable”,todas as suas subclasses estarão implicitamente marcadas como serializáveis, mesmo que não declarem isso explicitamente.

Serialização e associações/composições

- Quando um objeto é serializado, todos os objetos que ele referencia em suas variáveis de instância também serão serializados.
- Por sua vez, se estes objetos referenciam outros, estes serão também serializados, e assim sucessivamente.
- Suponha um objeto Kernel que possua uma referência para uma matriz de Dog (lembre-se, matrizes são também objetos). Esta matriz contém 2 referências a objetos Dog.
- Cada objeto Dog possui um atributo nome (que é uma referência a um objeto String) e um objeto Collar, o qual possui um int. Ao serializar o objeto Kernel, a matriz, os objetos Dog, os String e os objetos Collar serão serializados automaticamente.

Atributos transient

- Para evitar que um atributo específico seja serializado, basta marcá-lo como transient .
- Assim, no momento da chamada ao método writeObject , o objeto será serializado sem aquele atributo.

```
public class Carro implements Serializable{
    private String modelo;
    private transient String cor;
    private PortaMalas pm;
    ...
```

Variáveis estáticas

- Variáveis estáticas não são serializadas, pois pertencem à classe e não às suas instâncias.
- Ao ser desserializado, o objeto terá o valor das variáveis estáticas de sua classe naquele momento.
- Portanto, objetos que podem ser serializados não devem depender de valores de variáveis estáticas que sejam alteradas dinamicamente, tendo em vista que seus valores podem não ser os mesmos quando o objeto for reconstituído.

Exemplo

- Apostila moodle página 127 – 129;

Persistência: Serialização/Deserialização de objetos

Prof. Doglas André Finco
doglas.andref@uffs.edu.br