Curso de Java

aula 11

Prof. Anderson Henrique

Serialização de objetos (Serializable)

Construímos uma classe para gravar e recuperar uma lista de contas bancárias, para isso descrevemos os estados das variáveis de cada conta em um arquivo texto, mas, a conta era um objeto muito simples.

Agora imagine que precisaríamos salvar o estado de objetos compostos de outros objetos, que sejam bem mais complexos do que aquela conta. Então o processo de gravação agora será bem mais complexo, e é para isso que serve a serialização de objetos.

Ela permite simplesmente que você diga para o seu programa, salve esse objeto e todas as variáveis de instância para mim.

Vamos criar uma classe principal chamada Serializa no mesmo pacote para apresentar a serialização de objetos. A nossa serialização só pode ser feita em objetos serializáveis.

E como você sabe se um objeto é ou não serializável? Ele implementa a interface Serializable, p.ex.: a classe String implementa Serializable, e tem diversas outras classes que também a implementa.

```
public static void main(String[ ] args) throws Exception{
        String[ ] nomes = {“Ana”,”Jorge”,”Maria”};
        FileOutputStream fos = new FileOutputStream(“C:/files/objeto.ser”);
        ObjectOutputStream oos = new ObjectOutputStream(fos);
        oos.writeObject(nomes);
        oos.close( );
```

```
        FileInputStream fis = new FileInputStream(“C:/files/objeto.ser” );
        ObjectInputStream ois = new ObjectInputStream(fis);
        String[ ] rnomes = (String[ ]) ois.readObject( );
        ois.close( );
        System.out.println(Arrays.toString(rnomes));
}
```

Agora vamos serializar os nossos objetos do tipo Conta.

```
        Conta conta1 =  new Conta(“Roberto”, 111_222_333_444);
```

```
FileOutputStream fos = new FileOutputStream(“C:/files/objeto.ser”);
ObjetoOutputStream oos = new ObjectOutputStream(fos);
oos.writeObject(conta1);
oos.close( );

FileInputStream fis = new FileInputStream(“C:/files/objeto.ser” );
ObjectInputStream ois = new ObjectInputStream(fis);
Conta c1 = (Conta) ois.readObject( );
ois.close( );
c1.exibeSaldo( );
```

Será que vai funcionar? Não, porque a nossa classe Conta não implementou a interface Serializable.

public class Conta implements **java.io.Serializable** {

E se tivéssemos várias contas? Chamamos várias vezes os métodos **writeObject** e **readObject**. Quando estamos trabalhando com serialização de objetos, o que é serializado são as variáveis do objeto. Variáveis de classe, ou seja, variáveis estáticas não serão serializadas.

Temos a opção de não querer serializar uma variável utilizando o modificador **transient**.

Vamos utilizar a serialização para modificar a classe que armazena Contas e recupera Contas bancárias.

```
public void armazenarContas(ArrayList<Conta> contas) throws IOException{
        try(FileOutputStream fos = new FileOutputStream("C:/files/contas.ser")){
        try(ObjetoOutputStream oos = new ObjectOutputStream(fos)){
                oos.writeObject(contas);
        }
}
```

```
        try(FileInputStream fis = new FileInputStream(“C:/files/contas.ser” )){
        try(ObjectInputStream ois = new ObjectInputStream(fis){
                return (ArrayList<Conta>) ois.readObject( );
        }
}
```

Poderá lançar diversas exceções no método main. Existem algumas perdas quando trabalhamos com objetos serializados, a leitura é incompreensível, pois o mesmo trabalha com bytes, mas, a serialização simplifica o processo de armazenar e recuperar os dados.

Prosseguiremos no próximo slide...

Professor: Anderson Henrique

Programador nas Linguagens Java e PHP