

Java Avançado

Tópicos Avançados de I/O

SoftBlue

www.softblue.com.br

Todos os direitos de cia reservados. No  permitida a distribuio fsica ou eletrnica deste material sem a permisso expressa e por escrito do autor.

Tpicos Abordados

- Argumentos de linha de comando
- A classe *java.util.Properties*
 - Lendo e escrevendo em arquivos de propriedades
- Serializao de objetos
 - *ObjectInputStream* e *ObjectOutputStream*
 - Controlando a serializao
 - Controle de verso
- Acesso aleatrio a arquivos

Argumentos de Linha de Comando

- O Java permite que argumentos de linha de comando sejam fornecidos para a aplicao
- Estes argumentos podem ser lidos pelo programa

```
java aplicacao.ClassePrincipal 123 abc
```

JVM

Classe

Argumentos

Lendo os Argumentos

- Os argumentos passados em linha de comando são passados como parâmetro para o método **main()**

```
public static void main(String[] args) {  
    String arg1 = args[0];  
    String arg2 = args[1];  
}
```

args.length retorna o número de argumentos

"123"

"abc"

A Classe java.util.Properties

- Representa uma coleção similar a um Map
- Mapeia uma chave a um valor
- Ainda bastante usada para ler arquivos de configuração escritos na forma "*chave = valor*"

```
# Arquivo 'config.txt'  
# Configurações da aplicação  
  
nomeAplicacao = Minha Aplicação  
versao = 1.0  
debug = true
```

Lendo dados do Arquivo

- O carregamento dos dados do arquivo pode ser feito automaticamente para dentro de um objeto da classe Properties

```
Properties props = new Properties();  
  
FileInputStream fis = new FileInputStream("config.txt");  
props.load(fis);  
fis.close();  
  
String nomeAplicacao = props.getProperty("nomeAplicacao");  
String versao = props.getProperty("versao");  
String debug = props.getProperty("debug");
```

Escrevendo Dados no Arquivo

- Com o Properties é possível também salvar os dados num arquivo de configuração

```
Properties props = new Properties();  
props.setProperty("nomeAplicacao", "Minha Aplicação");  
props.setProperty("versao", "1.0");  
props.setProperty("debug", "true");  
  
FileOutputStream fos = new FileOutputStream("config_save.txt");  
props.store(fos, "Gerado pela Aplicação");  
fos.close();
```

Escrevendo Dados no Arquivo

- Arquivo de saída gerado (*config_save.txt*)

```
#Gerado pela Aplicação  
versao=1.0  
debug=true  
nomeAplicacao=Minha Aplicação
```

- Para mais informações sobre a classe Properties, consulte o Javadoc

Serialização de Objetos

- Serializar significa transformar em bytes
- Objetos são serializados para que possam ser
 - Transferidos pela rede
 - Armazenados no disco rígido

A Interface Serializable

- Deve ser implementada pelas classes cujos objetos serão serializados

```
public class Mensagem implements Serializable {  
}
```

- Esta interface não possui qualquer método que deva ser implementado pela classe

ObjectOutputStream

- Classe usada para criar uma stream de saída onde podemos escrever objetos

```
Message m = new Message("texto");  
  
ObjectOutputStream oos =  
    new ObjectOutputStream(new FileOutputStream("out.bin"));  
oos.writeObject(m);  
oos.close();
```

O construtor recebe
uma OutputStream

ObjectInputStream

- Classe usada para criar uma stream de entrada de onde podemos ler objetos

```
ObjectInputStream ois =  
    new ObjectInputStream(new FileInputStream("out.bin"));  
Message m = (Message) ois.readObject();  
ois.close();
```

O construtor recebe
uma InputStream

Serialização dos Atributos

- Quando um objeto serializado possui referências a outros objetos, estes objetos também são serializados
 - Todos os objetos referenciados pela classe devem implementar a interface **Serializable**
- É possível descartar atributos da serialização usando o modificador **transient**

```
public class Mensagem {  
    private transient String tempMsg;  
}
```

Controlando a Serialização

- Existem duas formas de ter total controle sobre a serialização
 - Implementando os métodos
 - `writeObject(ObjectOutputStream out)`
 - `readObject(ObjectInputStream in)`
 - Implementando a interface `Externalizable`
 - `writeExternal(ObjectOutput out)`
 - `readExternal(ObjectInput in)`
- A JVM chama os métodos caso eles existam

Controle de Versão

- Quando um objeto é serializado, a JVM cria um código numérico para a classe
 - Este código não é randômico
 - É baseado nos nomes dos atributos e assinaturas dos métodos da classe
- Este número identifica a classe, de forma que se algo na classe mudar, este número também muda
- Ao tentarmos ler um objeto serializado depois que a classe for modificada, uma **InvalidClassException** será lançada

Controle de Versão

- Nem sempre este comportamento é desejado
- Para evitar que isso aconteça, devemos fornecer este número de versionamento manualmente

```
private static final long serialVersionUID = 1415906L;
```

- Isto garante que o processo de leitura de um objeto serializado será feito de forma correta

Acesso Aleatório a Arquivos

- Possibilita ler e escrever em arquivos
- A posição onde haverá a leitura ou escrita é definida através de um ponteiro
- Classe **RandomAccessFile**

```
RandomAccessFile raf = new RandomAccessFile("out.bin", "rw");  
raf.writeBoolean(true);  
raf.writeInt(10);  
raf.writeDouble(4.5);  
raf.close();
```

Modo de abertura
do arquivo

Acesso Aleatório a Arquivos

```
RandomAccessFile raf = new RandomAccessFile("out.bin", "r");  
boolean b = raf.readBoolean();  
int i = raf.readInt();  
double d = raf.readDouble();
```

- O ponteiro de escrita/leitura é controlado através do método **seek()**

```
RandomAccessFile raf = new RandomAccessFile("out.bin", "r");  
raf.seek(5);  
double d = raf.readDouble();
```

Deslocamento em
bytes a partir do
início do arquivo

Colocando em Prática...



Agora que você já aprendeu a teoria, acesse as vídeo-aulas práticas e pratique os assuntos abordados neste módulo!

[Clique aqui para acessar as vídeo-aulas práticas](#)
