Serializar:
Transformar um objeto numa sequência de bytes adequada à transmissão num canal de rede ou à escrita numa Stream.
O *marshalling* em RMI faz-se com esta técnica.

Em Java, os tipos primitivos são serializáveis, bem como instâncias de classes que implementem a interface `java.io.Serializable`.

STREAMS

Para Ler/Escrever recorremos às Streams, no pacote java.io :

- genéricas (abstratas): InputStream e OutputStream
- ler e escrever tipos primitivos: DataInputStream e DataOutputStream
- ler e escrever em FICHEIROS: FileInputStream e FileOutputStream
- ler e escrever OBJETOS SERIALIZÁVEIS: ObjectInputStream e ObjectOutputStream
  - simplificam o envio de objetos através de uma stream (por exemplo para comunicação na rede)

---

1. exercício (veículos 02)

1.a) Ainda sobre o exercício na atividade anterior (serviço de informação sobre veículos),
considere que o registo de informação de veículo passa a incluir também o **ano** de matrícula, sendo representado através das classe Registo, incluída no **código fonte inicial** para esta aula.

Desta vez, a intenção é usar objetos representativos do pedido, em vez de uma solução de protocolo diretamente programado, com códigos para representar cada operação.
Assim a semântica do pedido está no próprio **objeto** enviado.

Obtenha aqui o código inicial (projeto em formatos: netbeans, eclipse).

1.b) Implemente as seguintes classes em Java, copiando o código apresentado:

- Pedido (**abstrata**)
- PedidoDeRegisto
- PedidoDeConsulta

```
// # ficheiro Pedido.java:

package sd;

public abstract class Pedido {
}
```

```
// # ficheiro PedidoDeConsulta.java:
package sd;
public class PedidoDeConsulta extends Pedido {

    private String matricula;

    public PedidoDeConsulta(String matricula) {
        this.matricula= matricula;
    }

    public String getMatricula() {
        return matricula;
    }
}
```

```
// # ficheiro PedidoDeRegisto.java:
package sd;
public class PedidoDeRegisto extends Pedido {

    public Registo reg;

    public PedidoDeRegisto(Registo r) {
        this.reg= r;
    }

    public Registo getRegisto() {
        return reg;
    }
}
```

2- Ao compilar diretamente na linha de comandos, precisará de definir com precisão onde estão as dependências de cada classe.
No comando para compilar, deve indicar na **classpath** qual é a pasta com os recursos necessários à compilação com um comando semelhante a:

```
NetBeans:
$ javac -d build/classes -classpath build/classes src/sd/VeiculosClient.java
Eclipse:
$ javac -d bin -classpath bin src/sd/VeiculosClient.java


(O IDE trata de tudo, mas deve saber como fazer.)

Executar:
$ java -classpath PASTA NOME-DA-CLASSE Arg1 Arg2...
```

Poderá ter de executar mais que uma classe no mesmo projeto, e passar diferentes argumentos em cada caso...

3.a- Atualize a aplicação **cliente** (`sd.VeiculosClient`),
que desta vez forma uma instância de uma das classes (Pedido) e é apenas esse objeto que escreve no socket, com uma **Stream** apropriada.

Complete o código para fazer o envio do Pedido pelo socket, com uma stream de objetos (ver acima).

3.b) Note que ao tentar escrever um `Pedido`, ele tem de ser **serializável**...
Será necessário algum **ajuste**? (a classe `Registo e os Pedidos devem implementar java.io.Serializable`)

4. No **servidor**, depois de aceitar uma ligação deve ler do socket de dados um

```
Pedido objPedido
```

e:

```
            if (objPedido==null)
                System.err.println("PEDIDO NULL: esqueceu-se de alguma coisa");
            else if (objPedido instanceof PedidoDeConsulta) {
                // ler o objeto PedidoDeConsulta
                PedidoDeConsulta pc= (PedidoDeConsulta) objPedido; // cast para facilitar acesso a métodos de pc

                //  ... atendimento especifico...
                // procurar aquela matricula numa lista

            }
            else if (objPedido instanceof PedidoDeRegisto) {
                // ler o objeto PedidoDeRegisto
                // ... idem...


                // inserir aquele objeto Registo numa estrutura de dados local...



            }
            else
                System.err.println("PROBLEMA");
```

Na versão mais simples, o servidor mantém um repositório (para já apenas em memória, numa coleção) de instâncias da classe `Registo` como base de dados do serviço.

Teste ambas as aplicações, cliente e servidor, com diferentes pedidos sobre várias matrículas.

5- Altere o servidor para guardar os registos de veículos de modo **persistente**, em ficheiros (no futuro usaremos uma BD) .
Junto ao servidor, deve ter uma pasta "registos", dentro da qual, para cada pedido de registo deve ser escrito um **ficheiro** (com o nome <matrícula>.object).
 # onde está <matrícula> fixa algo como 12-AB-34

Para atender um pedido de consulta, deve procurar a existência do ficheiro para a matrícula procurada, dentro da mesma pasta. Se existir, deve ler e devolver à aplicação cliente o objeto (uma instância) do tipo Registo.
Use a *stream* mais apropriada ao caso.

Apontamentos diversos:

- no final, compare o seu código com este:

- até ao ex. 4 (sem som)
- ex. 5

Última alteração: quinta, 29 de setembro de 2022 às 13:09

Contactar suporte do site

Nome de utilizador: Rodrigo Alves (Sair)
Resumo da retenção de dados
Obter a Aplicação móvel

Fornecido por Moodle