53 45 52 45 49 20 46 49 45 4c 20
41 4f 53 20 50 52 45 43 45 49 54
4f 53 20 44 41 20 48 4f 4e 52 41
20 45 20 44 41 20 43 49 c3 8a 4e
43 49 41 2c 20 50 52 4f 4d 4f 56
45 4e 44 4f 20 4f 20 55 53 4f 20
45 20 4f 20 44 45 53 45 4e 56 4f
4c 56 49 4d 45 4e 54 4f 20 44 41
20 49 4e 46 4f 52 4d c3 81 54 49
43 41 20 45 4d 20 42 45 4e 45 46
c3 8d 43 49 4f 20 44 4f 20 43 49
44 41 44 c3 83 4f 20 45 20 44 41
20 53 4f 43 49 45 44 41 44 45 2e

RESIDÊNCIA DE

SOFTWARE

CAPACITAR
TREINAR
EMPREGAR

TRANSFORMAR

Firjan SENAI

serratec
Parque Tecnológico Região Serrana

Empacotar aplicações serializable

29-04-2022

Serialização

interface serializable

```
public class Proprietario implements Serializable {

    private static final long serialVersionUID = 1L;
```

Serialização em Java é um processo no qual a instância de um objeto é transformada em uma sequência de bytes e utilizada quando precisamos enviar objetos pela rede ou salvar objetos. Isso porque o estado atual do objeto é congelado e do outro lado nós podemos descongelar o estado deste objeto.

Para usar a serialização devemos explicitamente implementar a interface Serializable. Esta interface é apenas de marcação, pois não tem nenhum método a ser implementado, serve apenas para que a JVM saiba que a classe é serializada.

O serialVersionUID é um identificador de versão universal para uma classe Serializable. Na deserialização, esse número é utilizado para garantir que uma classe carregada corresponde exatamente a um objeto serializado. Se nenhuma correspondência do objeto for encontrada, então é lançada uma exceção.

Serialização

Quando dizemos que um objeto é serializado, estamos afirmando que este objeto será transformado em bytes, e poderá ser armazenado em disco ou transmitido por um stream.

Vamos utilizar dois tipos de stream, o FileOutputStream e o FileInputStream para manipular objetos serializados.

O FileOutputStream é um fluxo de arquivo que permite a gravação em disco e o FileInputStream é justamente o contrário, permitindo a leitura de um arquivo em disco.

As classes chamadas ObjectInputStream e ObjectOutputStream são responsáveis por inserir e recuperar objetos serializados do stream.

Serialização

```java
public class Fornecedor {
    private String cnpj;
    private String nome;

    public Fornecedor(String cnpj, String nome) {
        super();
        this.cnpj = cnpj;
        this.nome = nome;
    }

    @Override
    public String toString() {
        return "Fornecedor [cnpj=" + cnpj + ", nome=" + nome + "]";
    }

    public String getCnpj() {
        return cnpj;
    }

    public String getNome() {
        return nome;
    }

}
```

Serialização

No exemplo ao lado foi gerado o arquivo fornecedor.dat usando o **FileOutputStream** após foi feita a gravação do objeto com **ObjectOutputStream**, depois carregamos o arquivo com o **FileInputStream** e finalmente utilizamos o **ObjetcInputStream** para exibir o objeto.

```
public class ExemploSerializable {

    public static void main(String[] args) {
        Fornecedor f1 = new Fornecedor("12343434", "Extra LTDA");
        try {
            FileOutputStream arquivoGravar = new FileOutputStream("/exemplos/fornecedor.dat");
            ObjectOutputStream gravarObjeto = new ObjectOutputStream(arquivoGravar);
            gravarObjeto.writeObject(f1);
            gravarObjeto.flush();
            gravarObjeto.close();
            arquivoGravar.flush();
            arquivoGravar.close();
            System.out.println("Objeto gravado !");

        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

        System.out.println("Recuperando objeto");
        try {
            FileInputStream arquivoLeitura = new FileInputStream("/exemplos/fornecedor.dat");
            ObjectInputStream lerObjeto = new ObjectInputStream(arquivoLeitura);
            System.out.println(lerObjeto.readObject());
            lerObjeto.close();
            arquivoLeitura.close();
        } catch (Exception e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

    }
}
```

Serialização

Quando executamos o programa recebemos a mensagem de erro

```
java.io.NotSerializableException: br.com.senai.model.Fornecedor
Recuperando objeto
        at java.base/java.io.ObjectOutputStream.writeObject0(ObjectOutputStream.java:1185)
        at java.base/java.io.ObjectOutputStream.writeObject(ObjectOutputStream.java:349)
        at br.com.senai.exemplos.ExemploSerializable.main(ExemploSerializable.java:18)
java.io.WriteAbortedException: writing aborted; java.io.NotSerializableException: br.com.senai.model.Fornecedor
```

Temos que implementar serializable

```
public class Fornecedor implements Serializable {
    /**
     *
     */
    private static final long serialVersionUID = 1L;
```

Existem diversas aplicações que utilizam o recurso de serialização como por exemplo o Hibernate, aplicações que trabalham com fluxo de I/O. Qualquer tipo de trabalho que envolver persistência ou tramitação de dados, a serialização é necessária.

O processo de serialização realiza ainda o controle de versionamento de determinado objeto, para determinar assim a compatibilidade entre objeto e classe, e assim evitar inconsistências na recuperação do estado anteriormente salvo do objeto em questão. O controle de versionamento é feito através de um atributo chamado **serialVersionUID**, ele identifica se a versão do objeto é compatível com a versão da classe que serializou este.

Métodos Equals e HashCode

O código abaixo terá como resultado Não são o mesmo proprietário.
Isso porque a classe Proprietario não sobrescreveu o método equals() da classe Object retornando assim o valor de memória dos objetos.

```
        public static void main(String[] args) {
            Proprietario proprietario1 = new Proprietario(1, "Mariana");
            Proprietario proprietario2 = new Proprietario(1, "Mariana");

            if(proprietario1.equals(proprietario2)) {
                System.out.println("São o mesmo proprietário");
            } else {
                System.out.println("Não são o mesmo proprietário");
            }

        }
```

Vamos adicionar o método equals e hashCode somente para o código, agora o resultado terá como resposta "São o mesmo proprietário"

```
@Override
public int hashCode() {
    final int prime = 31;
    int result = 1;
    result = prime * result + ((codigo == null) ? 0 : codigo.hashCode());
    return result;
}

@Override
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    Proprietario other = (Proprietario) obj;
    if (codigo == null) {
        if (other.codigo != null)
            return false;
    } else if (!codigo.equals(other.codigo))
        return false;
    return true;
}
```

O método hashCode() é utilizado para organizar os elementos de uma coleção em um mesmo compartimento. Como por exemplo um armário de fichas de clientes podem ser separadas em várias pastas em uma gaveta ficariam todos os clientes cujo nome comece com a letra A, na segunda gaveta todos os clientes que comecem com a letra B, e assim sucessivamente, desta forma a busca fica mais rápida por um cliente.

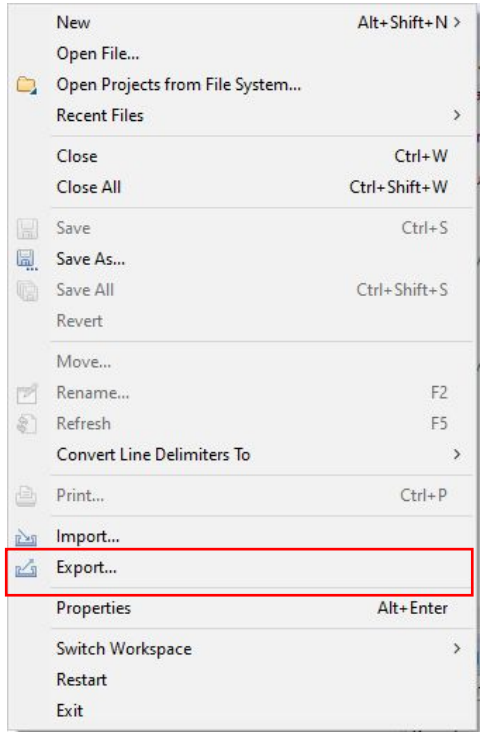
Para gerar o código hash de um objeto é necessário sobrescrever o método hascode() da classe Object,
É importante notar que esse método retorna um inteiro que representa o "gaveta" onde se encontra o cliente.

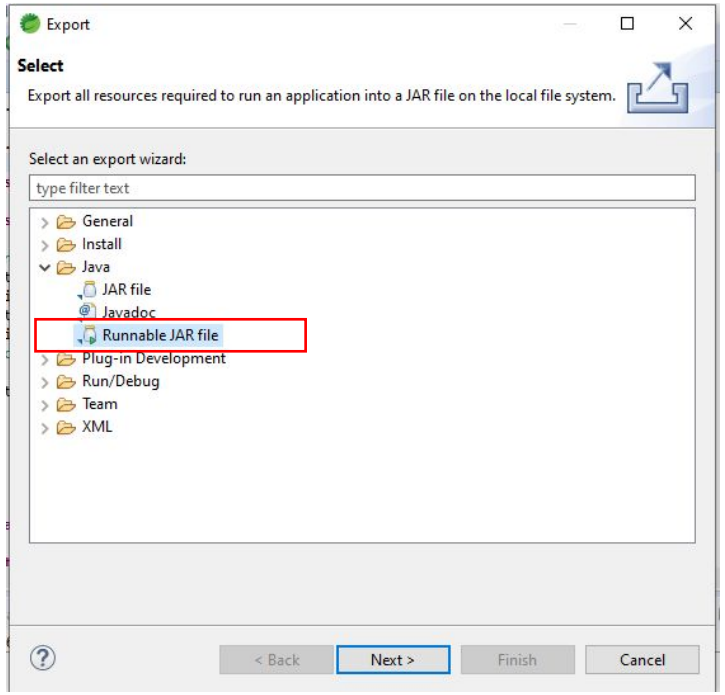
ARQUIVO JAR

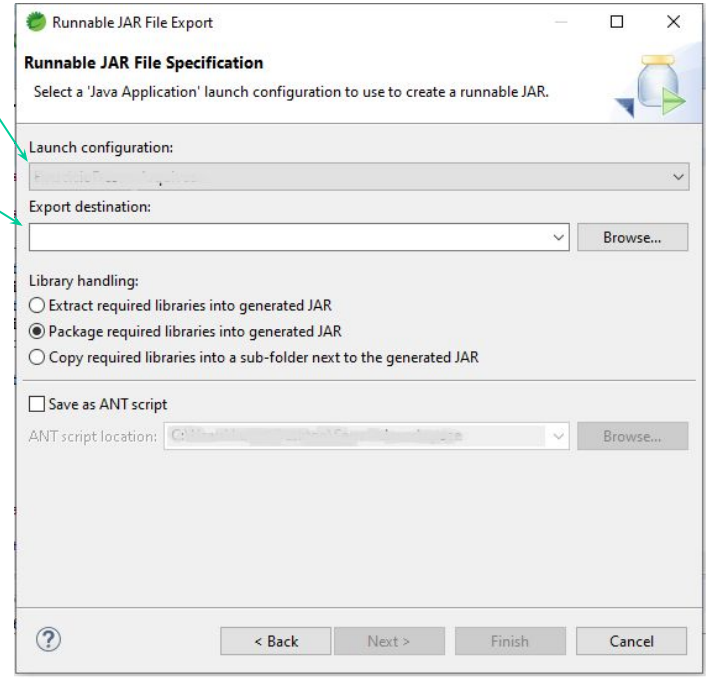
- Java é uma linguagem portável, ou seja, é multi-plataforma
- O compilador Java não cria nenhum arquivo executável (.exe)
- Para os sistemas operacionais os arquivos .java são vistos da mesma forma
- Para a JVM, os arquivos .class são visto da mesma fora
- Um arquivo .jar (**J**ava **AR**chive), tem o mesmo funcionamento de um arquivo executável

CRIANDO ARQUIVO JAR

- A criação do arquivo .jar pode ser feita através do terminal utilizando o comando “jar”
- As IDEs dão suporte para a criação
- Um jar pode ser uma aplicação, uma biblioteca, uma instalação...







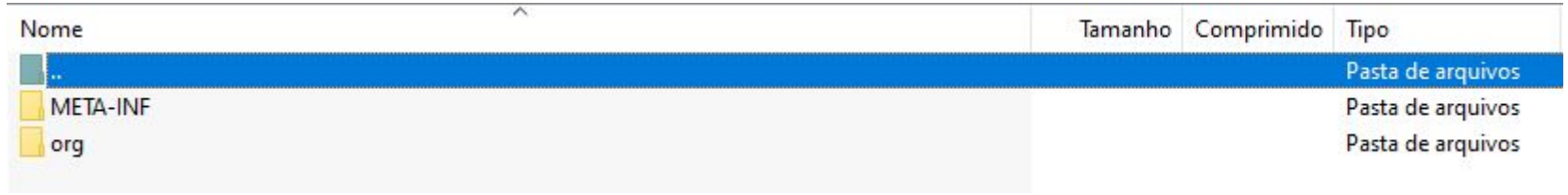
CRIANDO ARQUIVO JAR

- Um arquivo jar possui um arquivo manifesto localizado META-INF/MANIFEST.MF. As entradas do arquivo manifesto determinam como o arquivo jar será usado. Arquivos jar para serem executáveis (como os “.exe” do Windows) terão uma de suas classes especificadas como a classe "principal“, a classe que contém a “main”

- Comando para executar o jar: “java –jar <nome_do_arquivo.jar>”

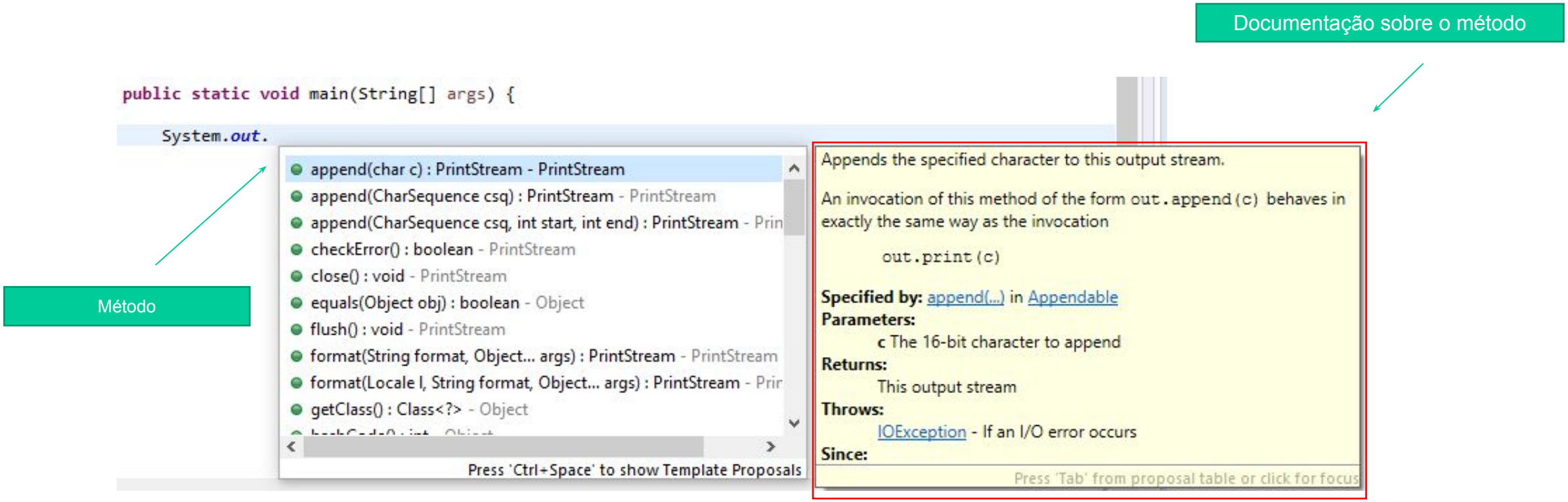
Manifesto

Jar pode ser aberto com programas de compactação



Javadoc

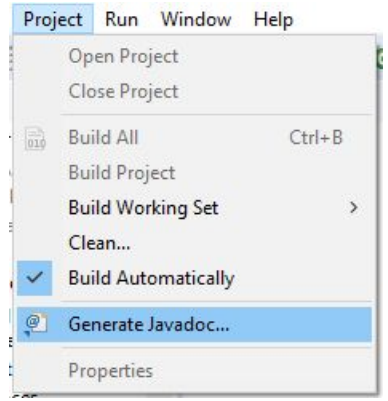
- Permite escrever comentários no código que serão usados para gerar documentação textual que pode ser entregue ao usuário



Javadoc

- Permite escrever comentários no código que serão usados para gerar documentação textual que pode ser entregue ao usuário
- Observar que tag html podem ser utilizadas
- Utilizamos anotações próprias para descrever

parâmetros retornos, autor, exceções entre outros



Pasta com documentação

```
/**
 * Appends the specified character sequence to this output stream.
 *
 * <p> An invocation of this method of the form <tt>out.append(csq)</tt>
 * behaves in exactly the same way as the invocation
 *
 * <pre>
 *     out.print(csq.toString()) </pre>
 *
 * <p> Depending on the specification of <tt>toString</tt> for the
 * character sequence <tt>csq</tt>, the entire sequence may not be
 * appended.  For instance, invoking then <tt>toString</tt> method of a
 * character buffer will return a subsequence whose content depends upon
 * the buffer's position and limit.
 *
 * @param  csq
 *         The character sequence to append.  If <tt>csq</tt> is
 *         <tt>null</tt>, then the four characters <tt>"null"</tt> are
 *         appended to this output stream.
 *
 * @return  This output stream
 *
 * @since  1.5
 */
public PrintStream append(CharSequence csq) {
```

Javadoc

- Os trechos de javadoc são blocos com as seguintes delimitações /** */
- As tags são inseridas dentro do bloco de comentários, antecedidas pelo caracter @ e após o nome da tag, segue o conteúdo que deseja descrever

Tag	Significado
`@author`	Especifica o autor da classe ou do método em questão.
`@deprecated`	Identifica classes ou métodos obsoletos. É interessante informar nessa tag, quais métodos ou classes podem ser usadas como alternativa ao método obsoleto.
`@link`	Possibilita a definição de um link para um outro documento local ou remoto através de um URL.
`@param`	Mostra um parâmetro que será passado a um método.
`@return`	Mostra qual o tipo de retorno de um método.
`@see`	Possibilita a definição referências de classes ou métodos, que podem ser consultadas para melhor compreender idéia daquilo que está sendo comentada.
`@since`	Indica desde quando uma classe ou métodos foi adicionado na aplicação.
`@throws`	Indica os tipos de exceções que podem ser lançadas por um método.
`@version`	Informa a versão da classe.

EXERCÍCIO

Criar uma classe **Fornecedor** com método aumentarSalario(double aumento) e adicionar os javadoc para a classe e método

```java
public class Fornecedor implements Serializable {
    /** Classe para criação de objetos do tipo fornecedor
     * @author Admin
     * @since criada em 2020
     */
    private static final long serialVersionUID = 1L;
    /**
     * Identifacado único da empresa no cadastro de pessas jurídicas
     */
    private String cnpj;
    private String nome;
    // private String razaoSocial;

    /*
     * Construtor cheio
     */
    public Fornecedor(String cnpj, String nome) {
        super();
        this.cnpj = cnpj;
        this.nome = nome;
    }


    /**
     * Método para pagamento dos funcionários para calculo da venda será acrescida a comissão de 1%
     * @author Admin
     * @param valor double - valor do pagamento do fornecedor
     * @return retorna o valor do pagamento do fornecedor
     * @throws IllegalArgumentException se valor for nulo
     */
    public Double calcularPagamento(double valor) throws IllegalArgumentException {
        return 0.0;
    }
}
```

EXERCÍCIO

Criar uma classe com o nome Veiculo com os atributos marca e modelo. Criar uma classe com o main e instanciar três objetos do tipo Veiculo com marcas diferentes e quarto objeto com a marca igual. Qual será o resultado da listagem do Set

```java
public class Veiculo {
    private String marca;
    private String modelo;

    public Veiculo(String marca) {
        super();
        this.marca = marca;
    }

    public Veiculo(String marca, String modelo) {
        super();
        this.marca = marca;
        this.modelo = modelo;
    }

    @Override
    public String toString() {
        return "Veiculo [marca=" + marca + ", modelo=" + modelo + "]";
    }

}
```

EXERCÍCIO

```
public class TesteVeiculo {

    public static void main(String[] args) {
        Veiculo veiculo1 = new Veiculo("VW");
        Veiculo veiculo2 = new Veiculo("Honda");
        Veiculo veiculo3 = new Veiculo("Renault");
        Veiculo veiculo4 = new Veiculo("Renault");

        Set<Veiculo> veiculos = new HashSet<Veiculo>();

        veiculos.add(veiculo1);
        veiculos.add(veiculo2);
        veiculos.add(veiculo3);
        veiculos.add(veiculo4);

        for (Veiculo veiculo : veiculos) {
            System.out.println(veiculo.toString());
        }
    }

}
```

EXERCÍCIO

O que devemos fazer para não repetir as marcas?

```java
@Override
public int hashCode() {
    final int prime = 31;
    int result = 1;
    result = prime * result + ((marca == null) ? 0 : marca.hashCode());
    return result;
}

@Override
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    Veiculo other = (Veiculo) obj;
    if (marca == null) {
        if (other.marca != null)
            return false;
    } else if (!marca.equals(other.marca))
        return false;
    return true;
}
```

Implementar o método equals e hashcode

EXERCÍCIO

E se quisermos que a marca e modelo não se repitam

```java
@Override
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    Veiculo other = (Veiculo) obj;
    if (marca == null) {
        if (other.marca != null)
            return false;
    } else if (!marca.equals(other.marca))
        return false;
    if (modelo == null) {
        if (other.modelo != null)
            return false;
    } else if (!modelo.equals(other.modelo))
        return false;
    return true;
}
```

Implementar o método equals e hashcode para marca e modelo

E se quisermos que a marca e modelo não se repitam

```
@Override
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    Veiculo other = (Veiculo) obj;
    if (marca == null) {
        if (other.marca != null)
            return false;
    } else if (!marca.equals(other.marca))
        return false;
    if (modelo == null) {
        if (other.modelo != null)
            return false;
    } else if (!modelo.equals(other.modelo))
        return false;
    return true;
}
```

Implementar o método equals e hashcode para marca e modelo

```
public static void main(String[] args) {
    Veiculo veiculo1 = new Veiculo("VW","Gol");
    Veiculo veiculo2 = new Veiculo("Honda","Fit");
    Veiculo veiculo3 = new Veiculo("Renault","Sandero");
    Veiculo veiculo4 = new Veiculo("Renault","Kwid");
```

EXERCÍCIO

- **Criar a tabela veiculo no banco de dados conforme o script abaixo:**
- **Criar a classe para conexão e a classe VeiculoDao**
- **Criar os métodos adicionar e listar na classe VeiculoDao**
- **Criar uma classe com o Main para inserir dados e listar utilizando o Scanner.**
- **Usar um menu com três opções:**
  - **1- Adicionar**
  - **2- Listar**
  - **3- Sair**

```
use bancoaula;
create table veiculo(codigo int primary key auto_increment,
marca varchar(20),modelo varchar(30));
```

EXERCÍCIO

```java
public class ConnectionFactory {
    String urlConexao = "jdbc:mysql://localhost:3306/bancoaula?useSSL=false";
    String usuario = "root";
    String senha = "mysql";
    Connection connection;

    public Connection getConnection() {
        //System.out.println("Tentando conectar ao banco de dados !");
        try {
            Class.forName("com.mysql.jdbc.Driver");
            connection = DriverManager.getConnection(urlConexao, usuario, senha);
            if (connection != null) {
        //      System.out.println("Conectado com sucesso !!");
            } else {
                System.out.println("Não possível conectar !!");
            }
            return connection;
        } catch (ClassNotFoundException e) {
            System.out.println("Driver não encontrado ou problema no path");
            return null;
        } catch (SQLException e) {
            System.out.println("Erro de Sql Exception !!");
            return null;
        }

    }
}
```

Podemos configurar a conexão para utilizar ou não SSL

EXERCÍCIO

```java
public class VeiculoDao {
    private Connection connection;

    public VeiculoDao() {
        connection = new ConnectionFactory().getConnection();
    }

    public void adicionar(Veiculo veiculo) {
        String sql = "insert into veiculo values(null,?,?)";
        try {
            PreparedStatement stmt = connection.prepareStatement(sql);
            stmt.setString(1, veiculo.getMarca());
            stmt.setString(2, veiculo.getModelo());
            stmt.execute();
            stmt.close();
            connection.close();
        } catch (SQLException e) {
            System.out.println(e.getMessage());
        }

    }

    public List<Veiculo> listar() {
        String sql = "select * from veiculo";
        List<Veiculo> listaVeiculos = new ArrayList<Veiculo>();
        try {
            PreparedStatement stmt = connection.prepareStatement(sql);
            ResultSet rs = stmt.executeQuery();
            while (rs.next()) {
                Veiculo veiculo = new Veiculo(rs.getInt("codigo"), rs.getString("marca"), rs.getString("modelo"));
                listaVeiculos.add(veiculo);
            }
            rs.close();
            stmt.close();
            connection.close();
        } catch (SQLException e) {
            System.out.println(e.getMessage());
        }
        return listaVeiculos;

    }

}
```

EXERCÍCIO

```java
public class TesteConsoleVeiculo {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Scanner entradaString = new Scanner(System.in);

        String marca, modelo;
        int menu;
        Veiculo veiculo = new Veiculo();
        VeiculoDao veiculoDao;
        do {
            exibirMenu();
            menu = sc.nextInt();
            switch (menu) {
            case 1:
                System.out.print("Marca:");
                marca = entradaString.nextLine();
                System.out.print("Modelo:");
                modelo = entradaString.nextLine();
                veiculoDao = new VeiculoDao();
                veiculo = new Veiculo(marca, modelo);
                veiculoDao.adicionar(veiculo);
                break;
            case 2:
                System.out.println("\n========Listagem de Carros============");
                veiculoDao = new VeiculoDao();
                for(Veiculo v: veiculoDao.listar()) {
                    System.out.println(v.toString());
                }
                System.out.println("\n=====================================");
                break;
            case 3:
                System.out.println("Saindo do sistema");
                break;

            default:
                System.out.println("Opção inválida");
                break;
            }

        } while (menu != 3);

    }

    public static void exibirMenu() {
        System.out.println("============Sistemas de Veículos================");
        System.out.println("1-Adicionar");
        System.out.println("2-Listar");
        System.out.println("3-Sair");
        System.out.println("Digite a opção:");
    }

}
```