

```
private transient Double total;
```

## Serialização

```
FileOutputStream fos = new FileOutputStream("c:/caminho/arquivo.dat");
```

```
ObjectOutputStream oos = new ObjectOutputStream(fos);
```

```

1: public class TesteCarrinho01 {
2:     public static void main(String[] args) {
3:         Carrinho carrinho = new Carrinho();
4:         carrinho.adiciona(BDProduto.getProdutos());
5:         carrinho.checkout(32, "Sônia Blanco");
6:
7:         String arquivoCarrinho = "c:/temp/compra_" + carrinho.getCodigo() + ".dat";
8:
9:         try {
10:             FileOutputStream fos = new FileOutputStream(arquivoCarrinho);
11:             ObjectOutputStream oos = new ObjectOutputStream(fos);
12:         } {
13:             oos.writeObject(carrinho);
14:             System.out.println("Arquivo gravado no disco!");
15:         } catch (IOException ex) {
16:             System.out.println("Erro ao gravar arquivo! " + "\n" + ex.getMessage());
17:         }
18:     }
19: }

```

```

private void writeObject(ObjectOutputStream oos) throws IOException {
    oos.defaultWriteObject();
    oos.writeObject(LocalDate.now());
}

```


### Atenção: Reescrevendo o método writeObject

Para que o sistema reconheça a reescrita do método `writeObject`, ele precisa ser **privado**. Se for usando qualquer outro modificador, o método não será encontrado pela JVM.

```
FileInputStream fis = new FileInputStream("c:/caminho/arquivo.dat")
```

```
ObjectInputStream in = new ObjectInputStream(fis)
```

```
1: public class TesteCarrinho02 {
2:     public static void main(String[] args) {
3:         Scanner sc = new Scanner(System.in);
4:         System.out.print("Informe o código do carrinho: ");
5:         int codigo = sc.nextInt();
6:
7:         String arquivoCarrinho = "c:/temp/compra_" + codigo + ".dat";
8:
9:         Carrinho carrinho = null;
10:        try (FileInputStream fis = new FileInputStream(arquivoCarrinho);
11:            ObjectInputStream in = new ObjectInputStream(fis)) {
12:            
13:                
14:                carrinho = (Carrinho) in.readObject();
15:                System.out.println("\n\nConteúdo arquivo: \n");
16:
17:                System.out.printf("Total Compra: %s \n", moeda(carrinho.getTotal()));
18:                System.out.printf("Caixa: %d - Atendente: %s \n", carrinho.getCheckout().getCaixa(),
19:                    carrinho.getCheckout().getAtendente());
20:                System.out.println("Lista de produtos: ");
21:                carrinho.getProdutos().forEach(p -> System.out.printf("\t%s: %s - %s \n",
22:                    p.getTipo(), p.getDescricao(), moeda(p.getValor())));
23:        } catch (ClassNotFoundException | IOException ex) {
24:            System.out.println("Erro de leitura do arquivo!\n" + ex.getMessage());
25:        }
26:    }
27: }
```

```
private void readObject(ObjectInputStream ois) throws IOException, ClassNotFoundException {
    ois.defaultReadObject();
    
    LocalDateTime dataHoraArquivo = (LocalDateTime) ois.readObject();
    System.out.println("Data e hora da geração do arquivo: " + data(dataHoraArquivo));
}
```

### Atenção: Reescrevendo o método readObject

Para que o sistema reconheça a reescrita do método `readObject`, ele precisa ser **privado**. Se for usando qualquer outro modificador, o método não será encontrado pela JVM.

Para lidar com caminhos relativos e absolutos de arquivos, usamos os comandos: **normalize**, **relativize** e **resolve**. Vejamos a seguir a definição e aplicação de cada um deles:

- **normalize** – Normaliza o path até o arquivo (../../file.txt).
- **relativize** – Faz um path de um arquivo em relação a outro.
- **resolve** – Junta dois paths

```
1: public class TesteNI003 {
2:     public static void main(String[] args) {
3:         Path p1 = Paths.get("C:/oracle/home/product/java/97875/curso.pdf");
4:         System.out.println(p1.subpath(0, 3));
5:     }
6: }
7: //Output: oracle\home\product
```

O método `subpath` funciona como o `substring` da classe `String`, porém, cada "/" representa uma posição:

```
0: oracle
1: home
2: product
3: java
4: 97875
5: curso.pdf
```

#### Atenção: "/" ou "\"

Para facilitar a definição de um caminho de arquivo independentemente do sistema operacional, o Java permite utilizar a barra "/".

```
1: public class TesteNI006 {
2:     public static void main(String[] args) {
3:         Path dir = Paths.get("C:/temp/relatorios");
4:         Path arquivo = dir.resolve("arquivo.txt");
5:         if (Files.notExists(dir)) {
6:             try {
7:                 Files.createDirectories(dir);
8:                 if (Files.notExists(arquivo)) {
9:                     Files.createFile(arquivo);
10:                }
11:                System.out.println("Pasta criada com sucesso!");
12:            } catch (IOException e) {
13:                System.out.println("Erro ao criar pasta: " + e.getMessage());
14:            }
15:        } else {
16:            System.out.println("A pasta já existe.");
17:        }
18:    }
19: }
```

#### Atenção: `Files.exists` e `Files.noExists`

! `Files.exists` não é a mesma coisa que `Files.noExists`. Em drives de CD-ROM no Windows por exemplo, se o CD não estiver presente, ambos `exists` e `noExists` podem retornar falso.



```

1: public class TesteNI007 {
2:     public static void main(String[] args) {
3:         Path dir = Paths.get("C:/temp/relatorios");
4:         Path arquivo = dir.resolve("arquivo.txt");
5:         if (Files.exists(dir)) {
6:             try {
7:                 Files.deleteIfExists(arquivo);
8:                 Files.deleteIfExists(dir);
9:                 System.out.println("Pasta e arquivos apagados com sucesso!");
10:            } catch (IOException e) {
11:                System.out.println("Erro ao apagar pasta/arquivo: " + e.getMessage());
12:            }
13:        } else {
14:            System.out.println("A pasta não existe.");
15:        }
16:    }
17: }

```

#### Atenção: Files.delete

A instrução `Files.delete` pode lançar as exceções: `NoSuchFileException`, `DirectoryNotEmptyException` ou `IOException`. Se usar a opção `Files.deleteIfExists`, nenhuma exceção será lançada.

#### Copiar/Mover arquivos

Para copiar arquivos ou pastas, usamos o comando `Files.copy(path_origem, path_destino, opções)`.

O enum `StandardCopyOption` contém as seguintes opções de cópia:

- `REPLACE_EXISTING`: Substitui o arquivo/pasta no destino
- `COPY_ATTRIBUTES`: Copia os atributos do arquivo de origem no arquivo de destino
- `NOFOLLOW_LINKS`: Indica que links simbólicos não devem ser seguidos.

```

51: static public void copiaPasta(Path raiz, Path dest) {
52:     try (Stream<Path> stream = Files.walk(raiz)) {
53:         stream.forEach(origem -> {
54:             Path destino = dest.resolve(raiz.relativeTo(origem));
55:             System.out.printf("Copiando de: %s para %s \n", origem, destino);
56:             copia(origem, destino);
57:         });
58:     } catch (IOException e) {
59:         System.out.println("Erro ao copiar pasta: " + e.getMessage());
60:     }
61: }
62:
63: static private void copia(Path origem, Path destino) {
64:     try {
65:         Files.copy(origem, destino, StandardCopyOption.REPLACE_EXISTING);
66:     } catch (IOException e) {
67:         System.out.println("Erro ao copiar: " + e.getMessage());
68:     }
69: }

```

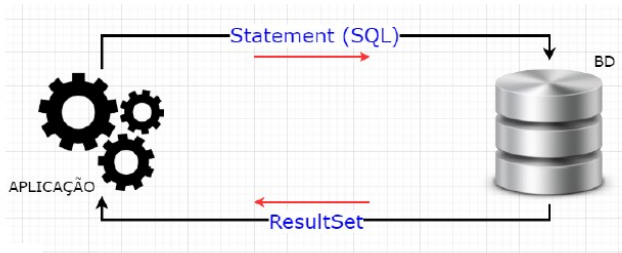
# BANCO DE DADOS

## JDBC

MySQL => Escrever uma implementação para **JDBC**  
--> driver mysql

interfaces

```
14: public Connection getConnection() {
15:     String url = "jdbc:mysql://localhost:3306/HR";
16:     String username = "admin";
17:     String password = "123456";
18:     try {
19:         con = DriverManager.getConnection(url, username, password);
20:     } catch (SQLException ex) {
21:         System.out.println("Erro ao obter conexão com o MySQL! " + ex.getMessage());
22:     }
23: }
```



- incluir dados
- alterar dados
- excluir dados
- buscar dados

CRUD

```
try (Statement statement = con.createStatement()) {
    statement.executeQuery("select * from funcionario");
} catch (SQLException e) {
    e.printStackTrace();
}
```

```
24: public void salvar(Funcionario t) throws DAOException {
25:     String sqlInsert = "INSERT INTO FUNCIONARIO (NOME, SEXO, IDADE, CIDADE, ESTADO, SALARIO)"
26:         + " VALUES (?,?,?,?,?,?)";
27:     try (PreparedStatement stmt = con.prepareStatement(sqlInsert)) {
28:         stmt.setString(1, t.getNome());
29:         stmt.setString(2, String.valueOf(t.getSexo()));
30:         stmt.setInt(3, t.getIdade());
31:         stmt.setString(4, t.getCidade());
32:         stmt.setString(5, t.getEstado());
33:         stmt.setDouble(6, t.getSalario());
34:         stmt.executeUpdate();
35:     } catch (SQLException ex) {
36:         throw new DAOException("Erro ao INSERIR Funcionario! " + ex.getMessage());
37:     }
38: }
```

Método	Retorno	Usado para
executeQuery(sqlString)	ResultSet	instrução SELECT
executeUpdate(sqlString)	int (linhas afetadas)	instruções INSERT, UPDATE, DELETE ou DDL
execute(sqlString)	boolean (true se contiver um ResultSet)	Qualquer instrução SQL

