

```
String sql = "SELECT NOME, CIDADE, ESTADO, SALARIO FROM FUNCIONARIO";
ResultSet rs = stmt.executeQuery(sql);
```

ResultSet cursor →

(true) rs.next() →	Igor	São Paulo	SP	7000.00
(true) rs.next() →	Cândido	Santos	SP	12000.00
(true) rs.next() →	Victor	Uberaba	MG	8500.00
(true) rs.next() →	Maria Antônia	Brasília	DF	6000.00
(true) rs.next() →	Carla	Monte Carmelo	MG	9000.00
(false) rs.next() →	null			

→ Funcionario

```
private Funcionario ormFuncionario(ResultSet rs) throws SQLException {
    return new Funcionario.Builder()
        .codigo(rs.getInt("CODIGO"))
        .nome(rs.getString("NOME"))
        .sexo(rs.getString("SEXO").charAt(0))
        .idade(rs.getInt("IDADE"))
        .cidade(rs.getString("CIDADE"))
        .estado(rs.getString("ESTADO"))
        .salario(rs.getDouble("SALARIO")).build();
}
```

ORM: Mapeamento Objeto Relacional

Agora, precisamos mostrar ao Java como fechar essa conexão quando ela não for mais necessária. Fazendo com que a classe implemente a interface **AutoCloseable**, o método `close` deverá ser sobrescrito informando os recursos que devem ser fechados:

```
27: @Override
28: public void close() throws DAOException {
29:     try {
30:         con.close();
31:     } catch (SQLException ex) {
32:         throw new DAOException("Erro ao encerrar conexão com o BD!" + ex.getMessage());
33:     }
34: }
```

Atenção: DAO

DAO é um padrão de projeto onde as classes responsáveis por executar ações no banco de dados ficam separadas das classes que fazem parte da regra de negócio. A sigla vem de **Data Access Object**.

```

1: { public FuncionarioDAO() throws DAOException {
2:     String url = "jdbc:mysql://localhost:3306/HR?useTimezone=true&serverTimezone=UTC";
3:     String username = "root";
4:     String password = "";
5:
6:     //abrir conexão com o BD
7:     try {
8:         con = DriverManager.getConnection(url, username, password);
9:     } catch (SQLException e) {
10:         throw new DAOException("Impossível obter conexão com o BD: " + e.getMessage());
11:     }
12:
13: }

```

```

1: #Arquivo: settings.properties
2: #Contem as configuração do projeto MySQLFuncionario
3: url = jdbc:mysql://localhost:3306/HR?useTimezone=true&serverTimezone=UTC
4: usuário = root
5: senha =

```

```

6: public String getValor(String chave) throws FileNotFoundException, IOException {
7:     try {
8:         FileInputStream fis = new FileInputStream("settings.properties");
9:         BufferedInputStream bis = new BufferedInputStream(fis)
10:     } {
11:         Properties properties = new Properties();
12:         properties.load(bis);
13:         return properties.getProperty(chave);
14:     }
15: }

```

Internacionalização

```

Locale de = new Locale("de", "DE");
Locale en = new Locale("en", "US");
Locale fr = new Locale("fr", "FR");
Locale pt = new Locale("pt", "BR");

```

- Formato da data
- Símbolo da moeda
- Idioma das mensagens de erro e dos menus de opção
- Nomes de meses e dias da semana

```

1: public class TesteLocale {
2:     public static void main(String[] args) {
3:         Stream<Double> valores = Stream.of(234.00, 546.50, 778.99);
4:         Stream<LocalDate> datas = Stream.of(LocalDate.now(),
5:                                             LocalDate.of(2020, 7, 9),
6:                                             LocalDate.of(2021, 12, 24));
7:
8:         Locale locale = new Locale("de", "DE");
9:         DateTimeFormatter dtf = DateTimeFormatter
10:             .ofLocalizedDate(FormatStyle.FULL).withLocale(locale);
11:
12:         valores
13:             .forEach(v -> System.out.println(getCurrencyInstance(locale).format(v)));
14:
15:         datas
16:             .forEach(d -> System.out.println(dtf.format(d)));
17:     }
18: }
19: /*
20: Output:
21: 234,00 €
22: 546,50 €
23: 778,99 €
24: Freitag, 29. Oktober 2021
25: Donnerstag, 9. Juli 2020
26: Freitag, 24. Dezember 2021
27: */

```

```

Locale.CANADA;
Locale.CHINA;
Locale.ENGLISH
Locale.GERMAN
etc....

```

```
#arquivo traducao_en_US.properties  
menu1 = add  
menu2 = update  
menu3 = delete  
menu4 = search  
menu5 = exit
```

```
#arquivo traducao_fr_FR.properties  
menu1 = ajouter  
menu2 = mettre à jour  
menu3 = supprimer  
menu4 = recherche  
menu5 = quitter
```

```
ResourceBundle traducao = ResourceBundle("traducao", Locale.CHINA); //traducao_zh_CN.properties
```

```
ResourceBundle traducao = ResourceBundle("traducao", Locale.FRANCE);  
System.out.println(traducao.getString("menu3")); //supprimer
```

Thread

Atenção: run() vs start()

Para que o processo seja executado de forma concorrente a Thread deve ser invocada pelo método `start()`. Executar o método `run()` irá fazer com que o método seja executado de forma serial.

Uma Thread só pode ser iniciada se estiver no estado **NEW**. Para verificar o estado de uma Thread podemos usar o método `getState()`. O Enum `Thread.State` pode apresentar os seguintes estados:

- **NEW**: A thread ainda não foi iniciada.
- **RUNNABLE**: A thread está em execução pela JVM.
- **BLOCKED**: A thread foi bloqueada por um monitor
- **WAITING**: A thread está aguardando o fim da execução de outra thread.
- **TIMED_WAITING**: A thread está aguardando o fim da execução de outra thread por um tempo especificado.
- **TERMINATED**: A thread foi finalizada.

```
Callable<Long> c = () -> {  
    Long r = IntStream.rangeClosed(1, 1000).sum();  
    return r;  
};  
  
ExecutorService es = Executors.newFixedThreadPool(2);  
Future<Long> f1 = es.submit(c);  
Future<Long> f2 = es.submit(c);  
  
es.shutdown();
```

```

6: public class ContadorSynchronized {
7:
8:     private static int c;
9:
10:    public synchronized void incremento() {
11:        c++;
12:    }
13:
14:    public synchronized void decremento() {
15:        c--;
16:    }
17:
18:    public synchronized int getC() {
19:        return c;
20:    }
21: }

```

Cada objeto no Java está associado a um monitor que uma Thread pode bloquear ou desbloquear:

- Métodos `synchronized` usam o monitor para o objeto `this`
- Métodos `static synchronized` usam o monitor da classe
- Bloco `synchronized` necessita que seja especificado o objeto para o qual o monitor deverá ser bloqueado ou desbloqueado.

```

synchronized (this) { }
synchronized (new Object()) { }

```

Atenção: Bloco `synchronized`

Use com cuidado bloco `synchronized` pois eles podem gerar gargalo de execução na aplicação. Demos utilizar esse recurso apenas em casos indispensáveis.

```

1: public class ThreadSoma extends Thread {
2:     private long total;
3:     @Override
4:     public void run() {
5:         synchronized (this) {
6:             IntStream.rangeClosed(1, 500).forEach(n -> {
7:                 dormir(10);
8:                 total += n;
9:             });
10:            notify(); //fim da soma
11:        }
12:    }
13:    public long getTotal() {
14:        return total;
15:    }
16:    static private void dormir(long milis) {
17:        try {
18:            sleep(milis);
19:        } catch (InterruptedException e) { e.printStackTrace(); }
20:    }
21: }

```



```

1: public class ThreadSoma extends Thread {
2:     private long total;
3:     @Override
4:     public void run() {
5:         synchronized (this) {
6:             IntStream.rangeClosed(1, 500).forEach(n -> {
7:                 dormir(10);
8:                 total += n;
9:             });
10:            notify(); //fim da soma
11:        }
12:    }
13:    public long getTotal() {
14:        return total;
15:    }
16:    static private void dormir(long milis) {
17:        try {
18:            sleep(milis);
19:        } catch (InterruptedException e) { e.printStackTrace(); }
20:    }
21: }

```

```

22: public class TesteWaitNotify {
23:     public static void main(String[] args) throws InterruptedException {
24:         ThreadSoma ts = new ThreadSoma();
25:         ts.start(); //inicio da soma
26:         synchronized (ts) {
27:             System.out.println("Aguardando a soma terminar....");
28:             ts.wait(); //aguardando o notify() de ts
29:             System.out.println("O resultado da soma é: " + ts.getTotal());
30:         };
31:     }
32: }

```

Exemplo de deadlock:

```

synchronized (objA) {
    synchronized(objB) {
    }
}

synchronized (objB) {
    synchronized (objA) {
    }
}

```

Constantes de prioridade (usamos o método `setPriority(int)` para altear a prioridade):

```

public static final int MIN_PRIORITY = 1;
public static final int NORM_PRIORITY = 5;
public static final int MAX_PRIORITY = 10;

```

- **Livelock:** É quando threads tentam se acionar de forma simultânea. É semelhante a uma ligação telefônica onde duas pessoas ligam uma para outra ao mesmo tempo, o telefone dará ocupado para as duas.

```
1: public class TesteParallelStream {
2:     public static void main(String[] args) {
3:         List<Funcionario> funcionarios = Funcionario.getFuncionarios();
4:
5:         Optional<Funcionario> localizado = funcionarios.parallelStream()
6:             .filter(f -> f.getNome().startsWith("A"))
7:             .findAny();
8:
9:         if (localizado.isPresent()) {
10:             System.out.println(localizado.get());
11:         }
12:     }
13: }
```

e-mail: igorth.genesis@gmail.com

blog: <http://igordev.com.br>

instagram: @igorth.genesis

linkedin: igorribeirodev