

Objetos Distribuídos

JDBC e Coleções

O que é

▶ JDBC

- ▶ Java Database Connectivity
- ▶ Tecnologia Java para padronização e acesso ao banco de dados
- ▶ Utiliza a linguagem SQL para interação com o SGBD
- ▶ JDBC é uma API, ou seja, apenas uma especificação
- ▶ Implementado através de um driver, que é fornecido pelo fabricante.
- ▶ Pacote base: `java.sql`



JDBC

- ▶ **Driver**
 - ▶ Fornecido pelo fabricante
- ▶ **DriverManager**
 - ▶ Responsável pela criação da conexão
- ▶ **Connection**
 - ▶ Representa uma conexão física com o SGBD
- ▶ **Statement/PreparedStatement**
 - ▶ Comando executado no SGBD (insert,delete,update)
- ▶ **ResultSet**
 - ▶ Resultado da consulta do banco de dados



Revisão SQL

- ▶ **Revisão dos comandos SQL**

- ▶ <http://www.w3schools.com/sql/default.asp>

- ▶ **Inclusão de registros:**

- ▶ `INSERT INTO table_name (column1, column2, column3,...) VALUES (value1, value2, value3,...);`

- ▶ **Seleção de registros:**

- ▶ `SELECT column_name(s) FROM table_name WHERE <column_name> <operator> <value>`

- ▶ `SELECT * FROM <table_name>;`

- ▶ **Alteração de registros:**

- ▶ `UPDATE table_name SET column1=value, column2=value2 WHERE some_column=some_value`

- ▶ **Remoção de registros:**

- ▶ `DELETE FROM table_name WHERE some_column=some_value`

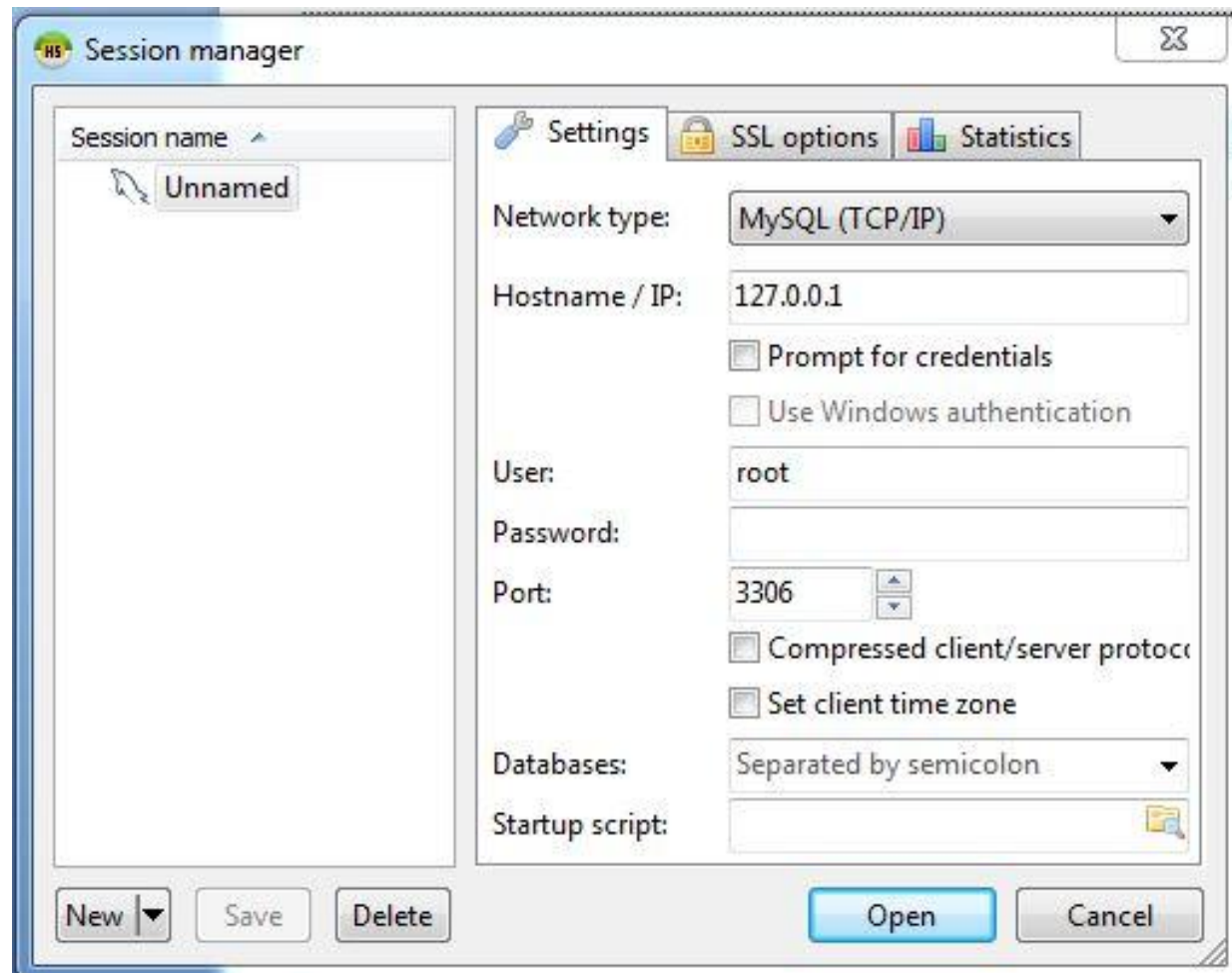


Instalar o Mysql

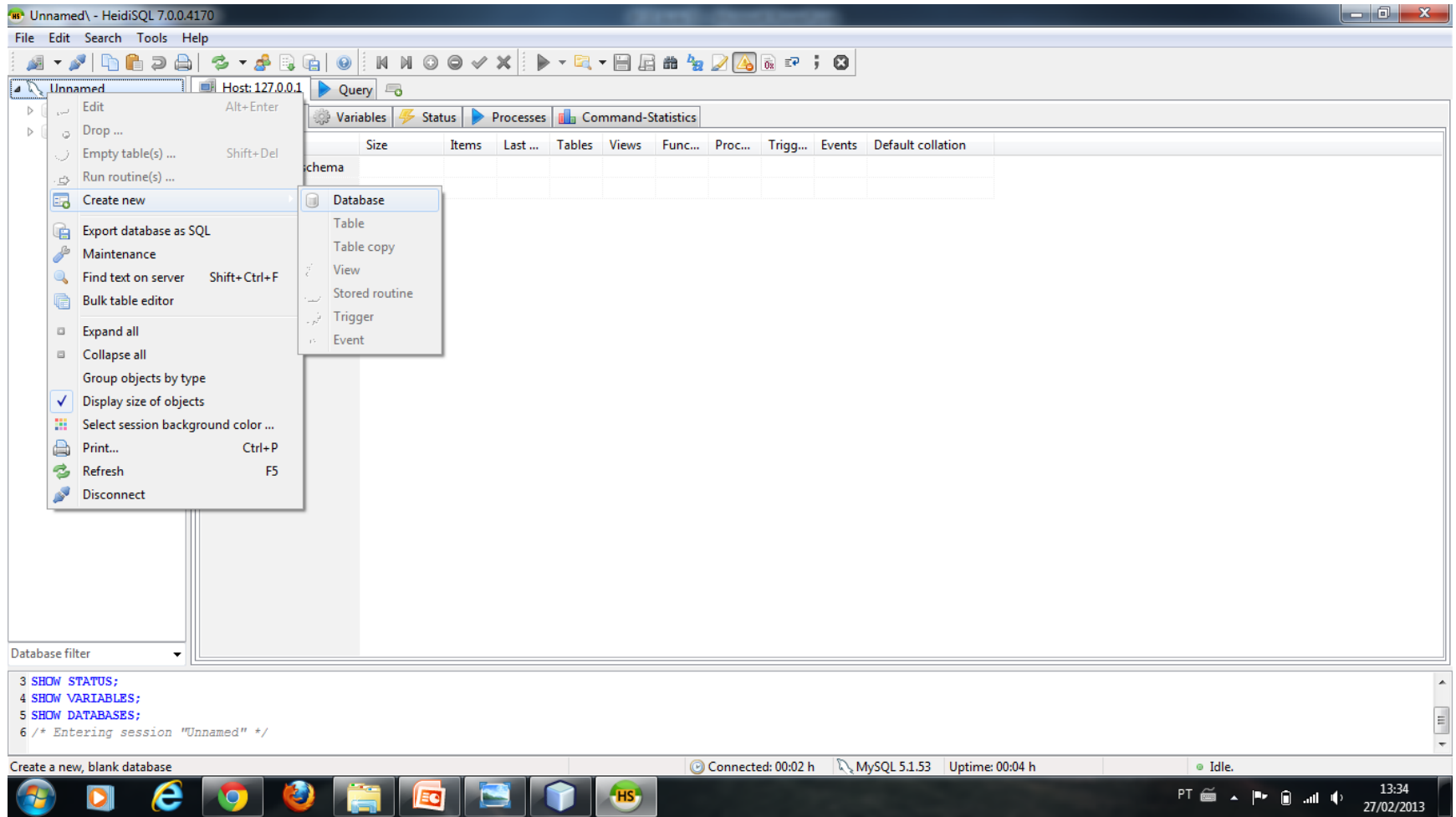
- ▶ Instalar na máquina o banco de dados Mysql
- ▶ Instalar o HeidiSQL para manipulação do banco
- ▶ Criar o banco de dados pod2013



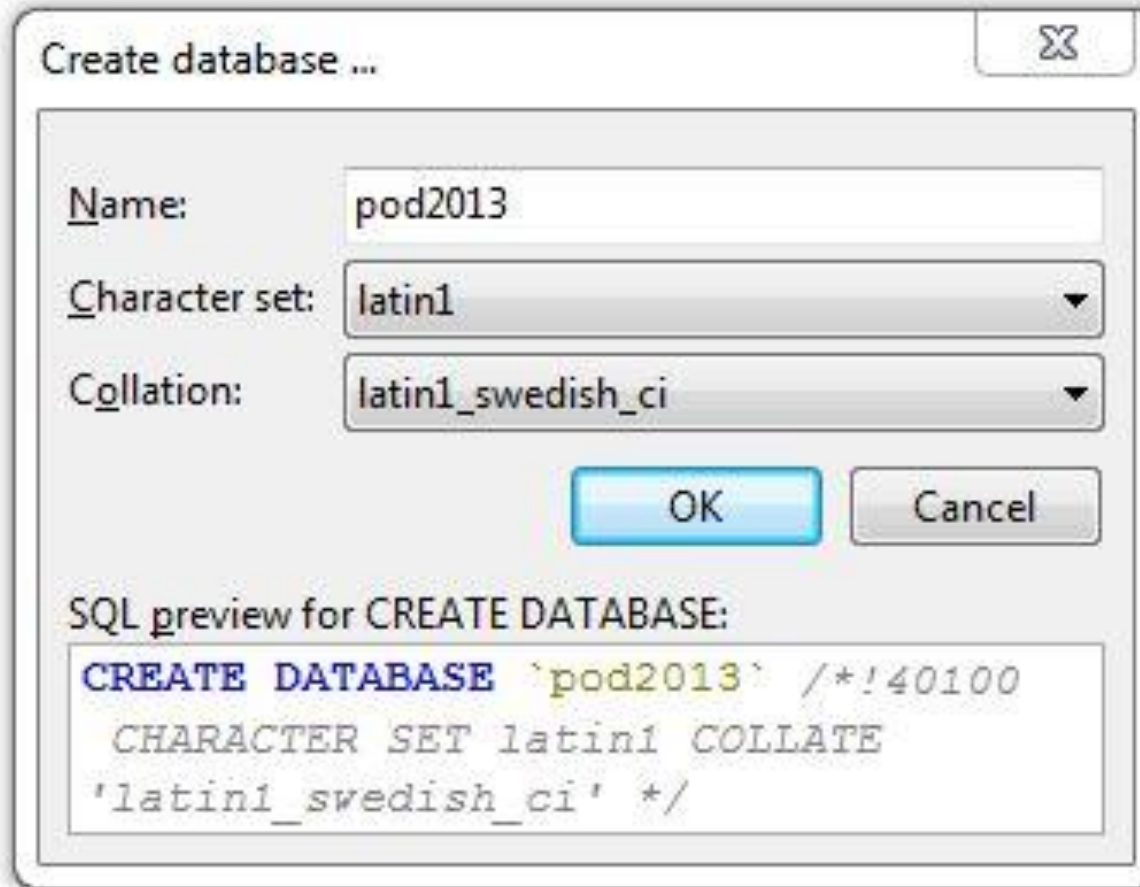
Instalando o MySql



Criar o banco de dados



Criar o banco de dados



Create database ...

Name: pod2013

Character set: latin1

Collation: latin1_swedish_ci

OK Cancel

SQL preview for CREATE DATABASE:

```
CREATE DATABASE `pod2013` /*!40100  
  CHARACTER SET latin1 COLLATE  
  'latin1_swedish_ci' */
```



Criando as tabelas - Rodoviária

▶ Onibus

- ▶ Placa: varchar PK
- ▶ Nr_lugares: int
- ▶ Ano_fabricacao: date
- ▶ Tipo : varchar

▶ Viagem

- ▶ Codigo : int PK
- ▶ Onibus_placa : varchar FK
- ▶ Data_viagem: datetime
- ▶ Cidade_destino: varchar
- ▶ Qtd_paradas: int

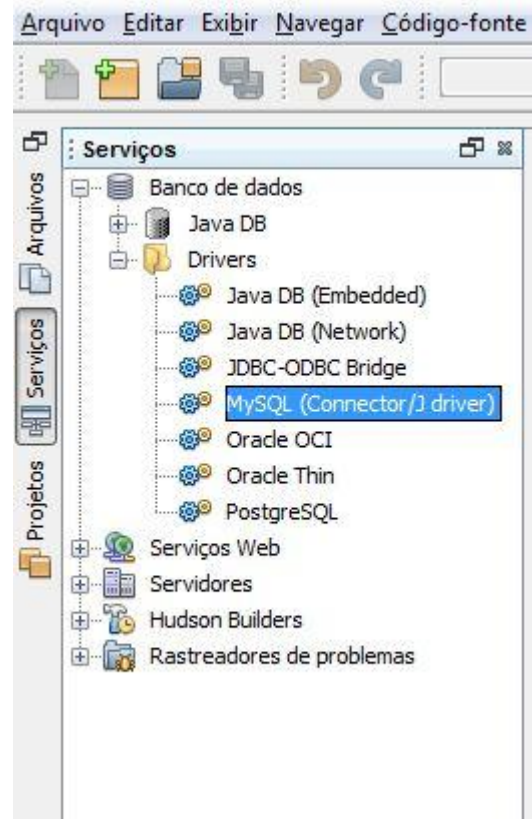
▶ Passagem

- ▶ Codigo: int PK
- ▶ Viagem_codigo: int FK
- ▶ Nr_poltrona: int



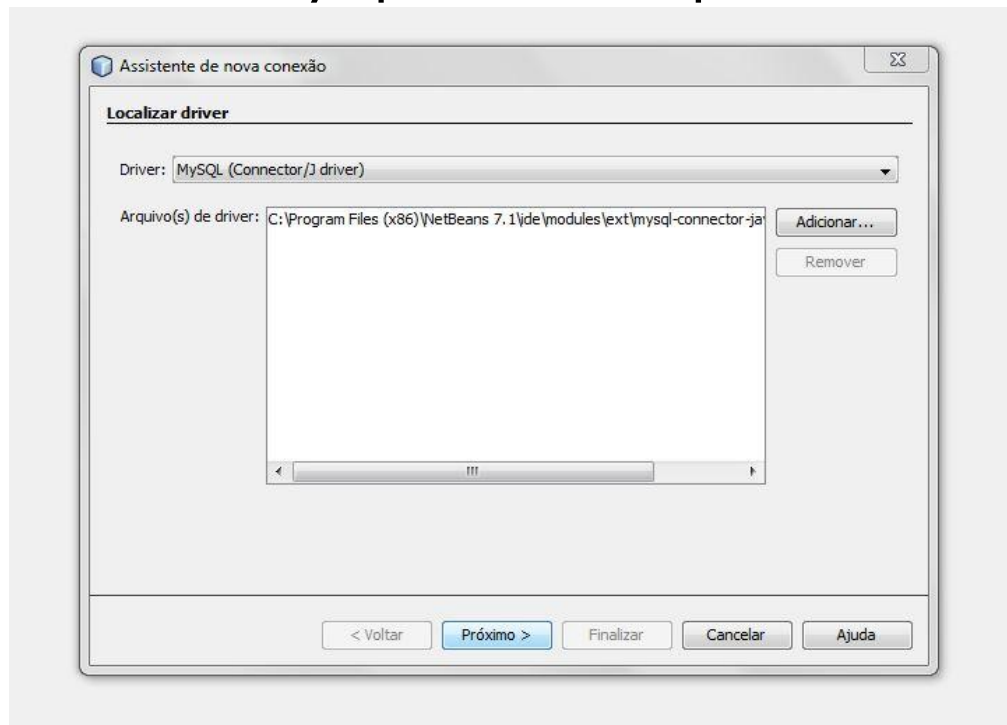
Criando o banco de dados

- ▶ Verificar se o driver do MySQL já está registrado



Conexão com o banco

- ▶ Clicar em serviços
- ▶ Botão direito em banco de dados
- ▶ Nova conexão
- ▶ Selecionar o Driver Mysql e clicar em proximo



Nova Conexão

Assistente de Nova Conexão

Personalizar Conexão

Nome do Driver: MySQL (Connector/J driver)

Host: localhost Porta: 3306

Banco de dados: pod2013

Nome do Usuário: root

Senha:

☒ Lembrar senha

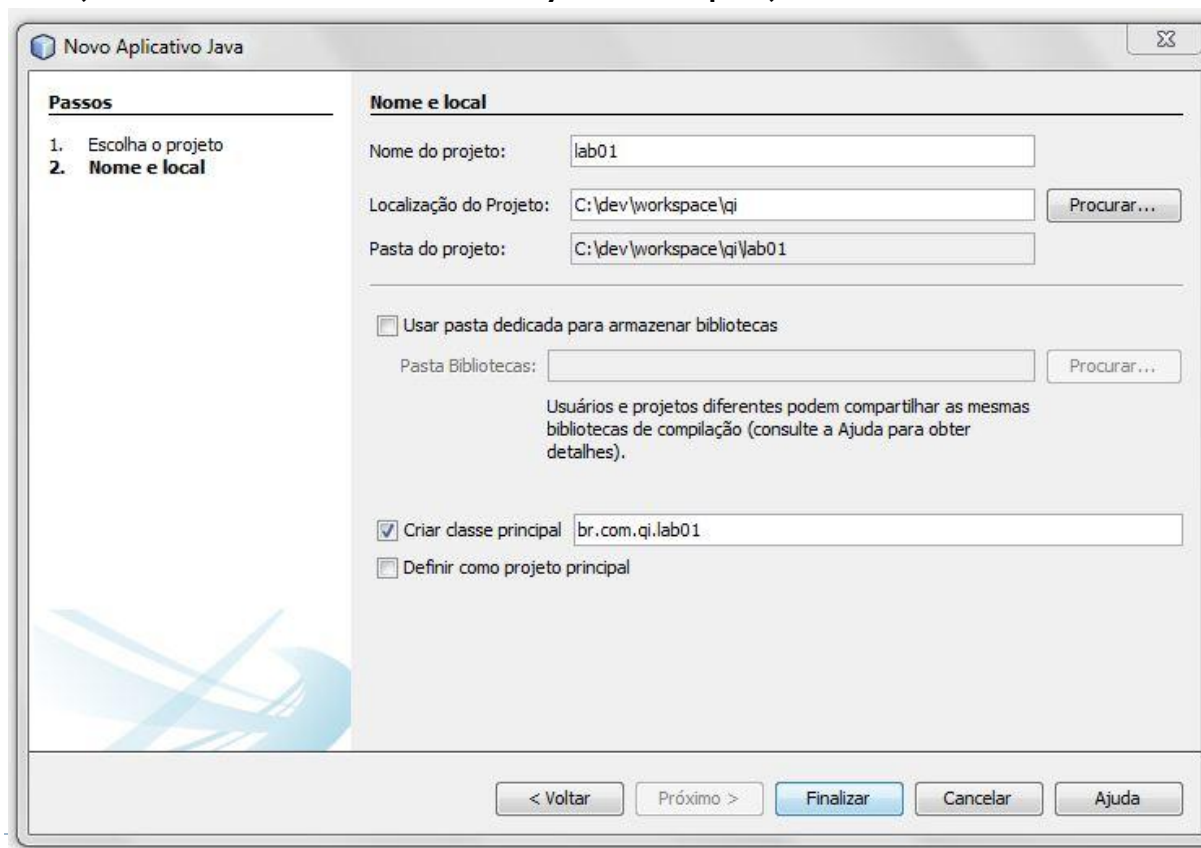
Testar Conexão

JDBC URL: jdbc:mysql://localhost:3306/pod2013

< Voltar Próximo > Finalizar Cancelar Ajuda

Driver JDBC

- ▶ Cada banco de dados tem sua implementação para API JDBC. Chamamos essa implementação de Driver
- ▶ Criar um novo projeto Java
- ▶ Adicionar o .jar da classe de conexão MySQL ao projeto



Driver Manager

- ▶ Cada classe de conexão possui um tipo de implementação
- ▶ O Driver Manager interage com o Driver para criar e retornar uma conexão

```
Connection conexao = DriverManager.getConnection(  
    "jdbc:<fornecedor>://end_servidor/nome_banco,\"root\",\"senha\");
```



Statement

- ▶ Oferece meios para passar comandos ao SGBD.
 - ▶ `Statement stmt = conexao.createStatement();`
- ▶ Com a variavel `stmt` podemos usar os métodos:
 - ▶ `execute()`, `executeQuery()`, `executeBatch()` e `executeUpdate()` para enviar instruções ao SGBD.
- ▶ Subinterfaces:
 - ▶ `PreparedStatement` (instruções SQL pré-compiladas)
 - ▶ `CallableStatement` (StoredProcedures)



Statement

▶ Exemplo:

- ▶ `stmt.execute("CREATE TABLE Pessoa "+ " (id MEDIUMINT NOT NULL AUTO_INCREMENT "+ ", nome varchar(50), endereco varchar(50) "+ ", telefone varchar(20), PRIMARY KEY(id)) ");`
- ▶ `int quantidadeDeRegistros = stmt.executeUpdate("INSERT INTO "+ " PESSOA (nome, endereco, telefone, cpf) "+ " VALUES ('José', 'Rua da Alegria', '33445566')");`



ResultSet

- ▶ O ResultSet encapsula o resultado de uma consulta a uma tabela na forma de um cursor.
- ▶ Métodos de navegação:
 - ▶ `next()`, `previous()`, `first()` e `last()`.
- ▶ Métodos para obter dados de uma coluna:
 - ▶ `getInt()`, `getString()`, `getDate()`, `getObject()`, etc.



ResultSet

► Exemplo:

```
ResultSet rs = stmt.executeQuery("SELECT * FROM  
    Pessoa");  
while (rs.next()) {  
    int idPessoa = rs.getInt("id");  
    String nomeDaPessoa = rs.getString("nome");  
    String enderecoDaPessoa =  
        rs.getString("endereco")  
}
```



PreparedStatement

- ▶ É um Statement pré-compilado que é mais eficiente quando temos o mesmo comando repetido várias vezes, mudando apenas os parâmetros.

- ▶ **Exemplo:**

```
String sql = "INSERT INTO Pessoa (nome,  
    endereco, telefone) values (?, ?, ?)";  
PreparedStatement stmt =  
    con.prepareStatement(sql);  
stmt.setString(1, "Alexandro");  
stmt.setString(2, "Rua da Alegria");  
stmt.setString(3, "33445566");  
stmt.executeUpdate();
```



Você deve saber...

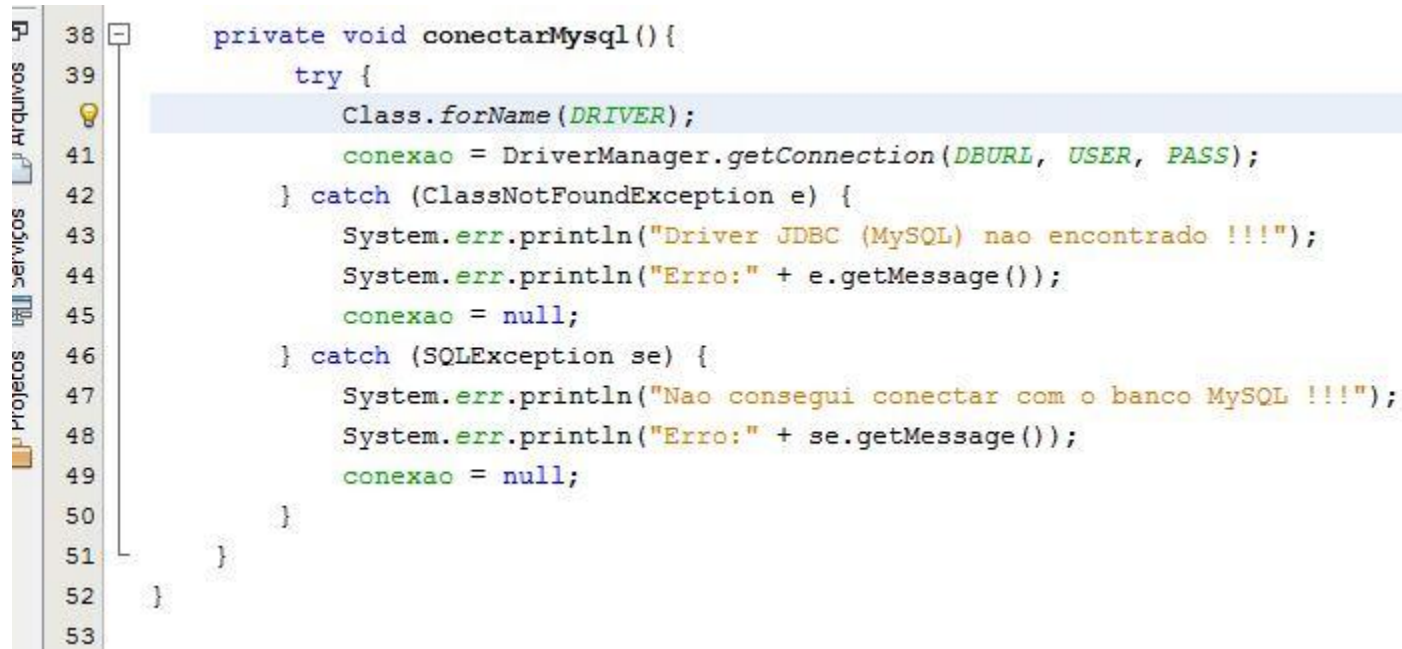
```
try {  
    //codigo  
} catch (SQLException) {  
    //codigo  
} finally {  
    con.close();  
    stmt.close();  
    rs.close();  
}
```



Exercício – Criando um Singleton 1/2

```
14  L  */
15  public class ConexaoMysql {
16
17      private static final String DRIVER = "com.mysql.jdbc.Driver";
18      private static final String DBURL = "jdbc:mysql://localhost:3306/pod2013";
19      private static final String USER = "root";
20      private static final String PASS = "";
21      private Connection conexao;
22      private static ConexaoMysql instance;
23
24      private ConexaoMysql() {
25          conectarMysql();
26      }
27
28      public static ConexaoMysql getInstance() {
29          if(instance == null) {
30              instance = new ConexaoMysql();
31          }
32          return instance;
33      }
34
35      public Connection getConexao() {
36          return conexao;
37      }
38  }
```

Exercício – Criando um Singleton 2/2



The image shows a screenshot of an IDE with a sidebar on the left containing icons for 'Arquivos', 'Serviços', and 'Projetos'. The main editor displays a Java code snippet for a method named `conectarMysql()`. The code is as follows:

```
38 private void conectarMysql() {  
39     try {  
40         Class.forName(DRIVER);  
41         conexao = DriverManager.getConnection(DBURL, USER, PASS);  
42     } catch (ClassNotFoundException e) {  
43         System.err.println("Driver JDBC (MySQL) nao encontrado !!!");  
44         System.err.println("Erro:" + e.getMessage());  
45         conexao = null;  
46     } catch (SQLException se) {  
47         System.err.println("Nao consegui conectar com o banco MySQL !!!");  
48         System.err.println("Erro:" + se.getMessage());  
49         conexao = null;  
50     }  
51 }  
52 }  
53
```

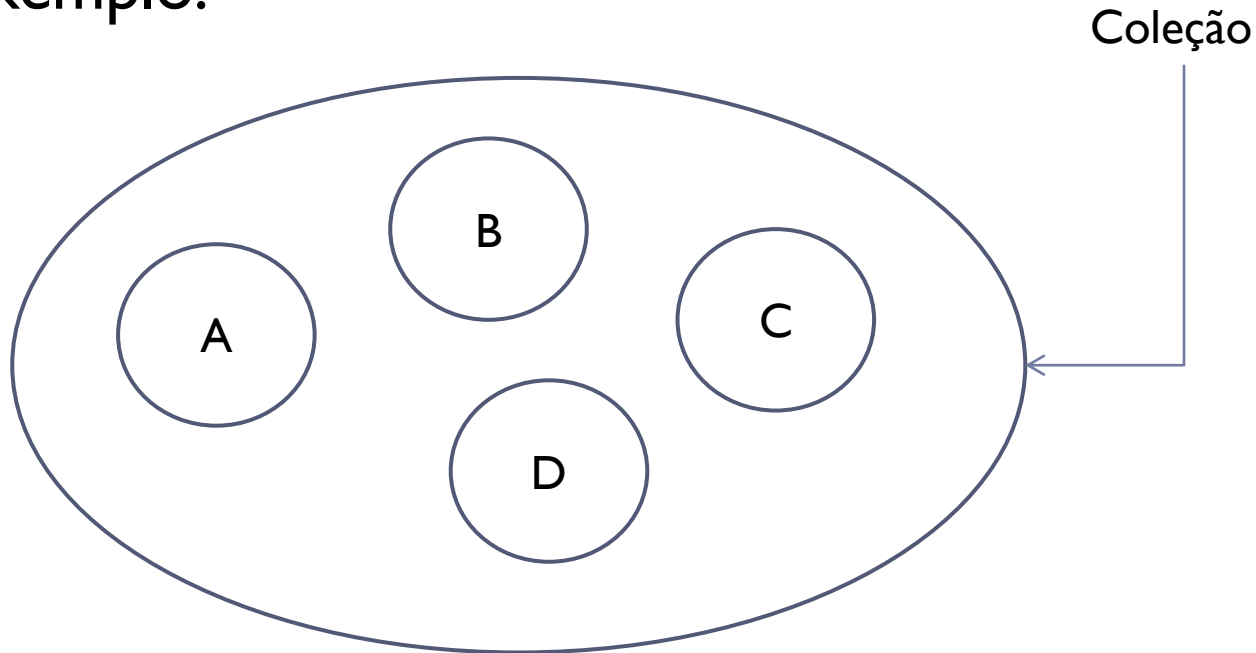
Exercício – Onibus Model

- ▶ Criar o JavaBean Onibus
- ▶ Criar a classe OnibusDao
- ▶ Implementar os métodos da classe OnibusDao
- ▶ Adicionar o Driver JDBC do Mysql no projeto



Coleções

- ▶ Uma coleção é um objeto que representa um grupo de objetos
- ▶ Exemplo:



Coleções

- ▶ Utiliza a API Java Collections para representar e manipular as coleções
- ▶ Permite a manipulação dos objetos independentemente dos detalhes de como foi implementado
- ▶ Pacote java.util
- ▶ Interface Collection
- ▶ Permite:
 - ▶ Adição, remoção, acesso, pesquisa e indagação sobre atributos



Coleções

▶ Os tipos de coleções são:

- ▶ List
- ▶ Set
- ▶ Map
- ▶ Collections



Coleções - List

- ▶ São coleções de elementos arrumados numa ordem linear, onde cada elemento tem um antecessor (exceto o primeiro) e um sucessor (exceto o último)
- ▶ Normalmente implementada como “Array” ou “Lista encadeada”
- ▶ A lista pode ser mantida ordenada ou não



Coleções - List

- ▶ Operações mais importantes sobre uma *List* são:
 - ▶ Adição de elementos (em qualquer lugar da lista, fornecendo o índice desejado)
 - ▶ Remoção de elementos (em qualquer lugar da lista, fornecendo o índice desejado)
 - ▶ Acesso aos elementos (obter o elemento de qualquer posição da lista e iterar sobre os elementos)
 - ▶ Pesquisa de elementos (descobrir se um certo elemento está na lista e em que posição está - índice)
 - ▶ Indagar sobre atributos (ex: numero de elementos da coleção)



Coleções – List

► Exemplo

```
public class ListExample {  
    public static void main(String[] args) {  
        //List Example implement with ArrayList  
        List<String> ls=new ArrayList<String>();  
        ls.add("one"); ls.add("Three");  
        ls.add("two"); ls.add("four");  
        Iterator it=ls.iterator();  
        while(it.hasNext()) {  
            String value=(String)it.next();  
            System.out.println("Value :"+value);  
        }  
    }  
}
```



Coleções - Set

- ▶ Está relacionada a ideia de conjuntos
- ▶ As classes que implementam esta interface não podem conter elementos repetidos
- ▶ A coleção set implementa *SortedSet* para ordenar os elementos



Coleções - Set

► Exemplo

```
Set cores = new HashSet(0);
cores.add("azul");
cores.add("amarelo");
cores.add("verde");
Iterator it = cores.iterator();
while (it.hasNext()) {
    String cor = (String) it.next();
    System.out.println(cor);
}
```



Coleções - Maps

- ▶ Um *Map* armazena pares (chave, valor) chamados itens
 - ▶ Chaves e valores podem ser de qualquer tipo
 - ▶ Elemento neste caso é sinônimo de Valor
- ▶ A chave é utilizada para achar um elemento rapidamente
 - ▶ Estruturas especiais são usadas para que a pesquisa seja rápida
- ▶ Diz-se portanto que um *Map* "mapeia chaves para valores"
 - ▶ O acesso à coleção sempre é feita conhecendo a chave
- ▶ O Mapa pode ser mantido ordenado ou não (com respeito às chaves)
- ▶ Normalmente implementada como "Tabela Hash" ou "Árvore"



Coleções - Maps

- ▶ As operações mais importantes de uma coleção do tipo *Maps* são:
- ▶ Adição de elementos (fornecendo chave e valor)
- ▶ Remoção de elementos (com chave dada)
- ▶ Acesso aos elementos (Iterar sobre os itens)
- ▶ Pesquisa de elementos (Descobrir se um elemento com chave dada está na coleção)
- ▶ Indagar sobre atributos (número de elementos)



Coleções - Maps

```
public class MapExample {  
    public static void main(String[] args) {  
        Map<Object,String> mp=new HashMap<Object, String>();  
        mp.put(new Integer(2), "Two");  
        mp.put(new Integer(1), "One");  
        mp.put(new Integer(3), "Three");  
        mp.put(new Integer(4), "Four");  
        Set s = mp.entrySet();  
        Iterator it = s.iterator();  
        while(it.hasNext()) {  
            Map.Entry m =(Map.Entry)it.next();  
            int key = (Integer) m.getKey();  
            String value = (String) m.getValue();  
            System.out.println("Key :"+key+" Value :"+value);  
        }  
    }  
}
```



Coleções - Collections

- ▶ Uma *collection* é uma coleção que não possui elementos duplicados
- ▶ O Conjunto pode ser mantido ordenado ou não
- ▶ Normalmente implementada como "Tabela Hash" ou "Árvore"



Coleções - Collections

- ▶ As operações mais importantes de uma coleção do tipo *collection* são:
 - ▶ Adição de elementos (descartando duplicações)
 - ▶ Remoção de elementos
 - ▶ Acesso aos elementos (Iterar sobre os elementos)
 - ▶ Pesquisa de elementos (Descobrir se um certo elemento está na coleção)
 - ▶ Indagar sobre atributos (Obter o número de elementos)
 - ▶ As operações são semelhantes as aplicadas para uma *List*

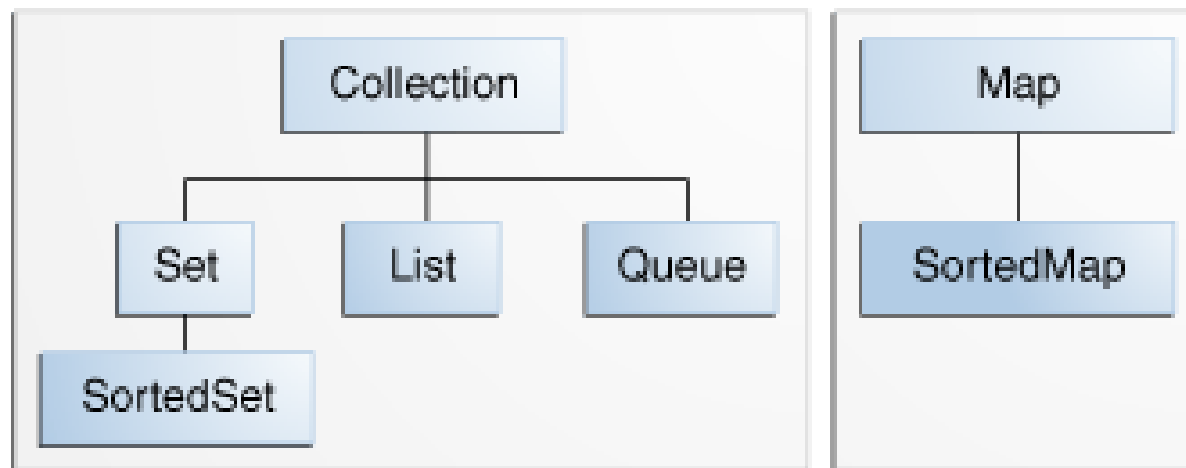


Coleções - Vantagens

- ▶ Reduz o esforço de programação fornecendo estruturas de dados úteis e algoritmos próprios do Java
- ▶ Aumenta o desempenho fornecendo implementações prontas de dados e estruturas úteis.
- ▶ Fornece interoperabilidade entre diferentes APIs fornecendo uma linguagem comum para passagem de coleções
- ▶ Estimula o reuso de software fornecendo interfaces padrão para coleções e algoritmos para manipulá-las



Collections



Exercício Collections

- ▶ Voltar ao código e imprimir na tela os clientes cadastrados no banco usando Collections

```
public List<Cliente> listar(){
    List<Cliente> clientes = new ArrayList<Cliente>();
    String sql = "select * from cliente";
    Connection conn = ConexaoMysql.getInstance().getConnection();
    PreparedStatement ps = null;
    ResultSet rs = null;
    try{
        ps = conn.prepareStatement(sql);
        rs = ps.executeQuery();
        Cliente cliente = null;
        while(rs.next()){
            cliente = new Cliente();
            cliente.setId(rs.getLong("id"));
            cliente.setNome(rs.getString("nome"));
            cliente.setEndereco(rs.getString("endereco"));
            cliente.setEmail(rs.getString("email"));
            clientes.add(cliente);
        }
    }catch(SQLException se){
        System.out.println("Erro ao consultar os clientes."+se.getMessage());
    }
    return clientes;
}
```

Exercícios

- ▶ Criar as classes Java para as outras tabelas
- ▶ Criar as classes Java de acesso aos dados <Model>
- ▶ Implementar os métodos (update, delete,)
- ▶ Utilizar as classes inserindo, listando, editando e deletando dados e mostrar na tela as operações
 - ▶ Ex: listar todos os ônibus, viagens ...
 - ▶ Ex: listar as viagens que um ônibus á realizou
 - ▶ Inserir ônibus, passagem, viagem

