



TREINAMENTO JAVA – BRQ/SP

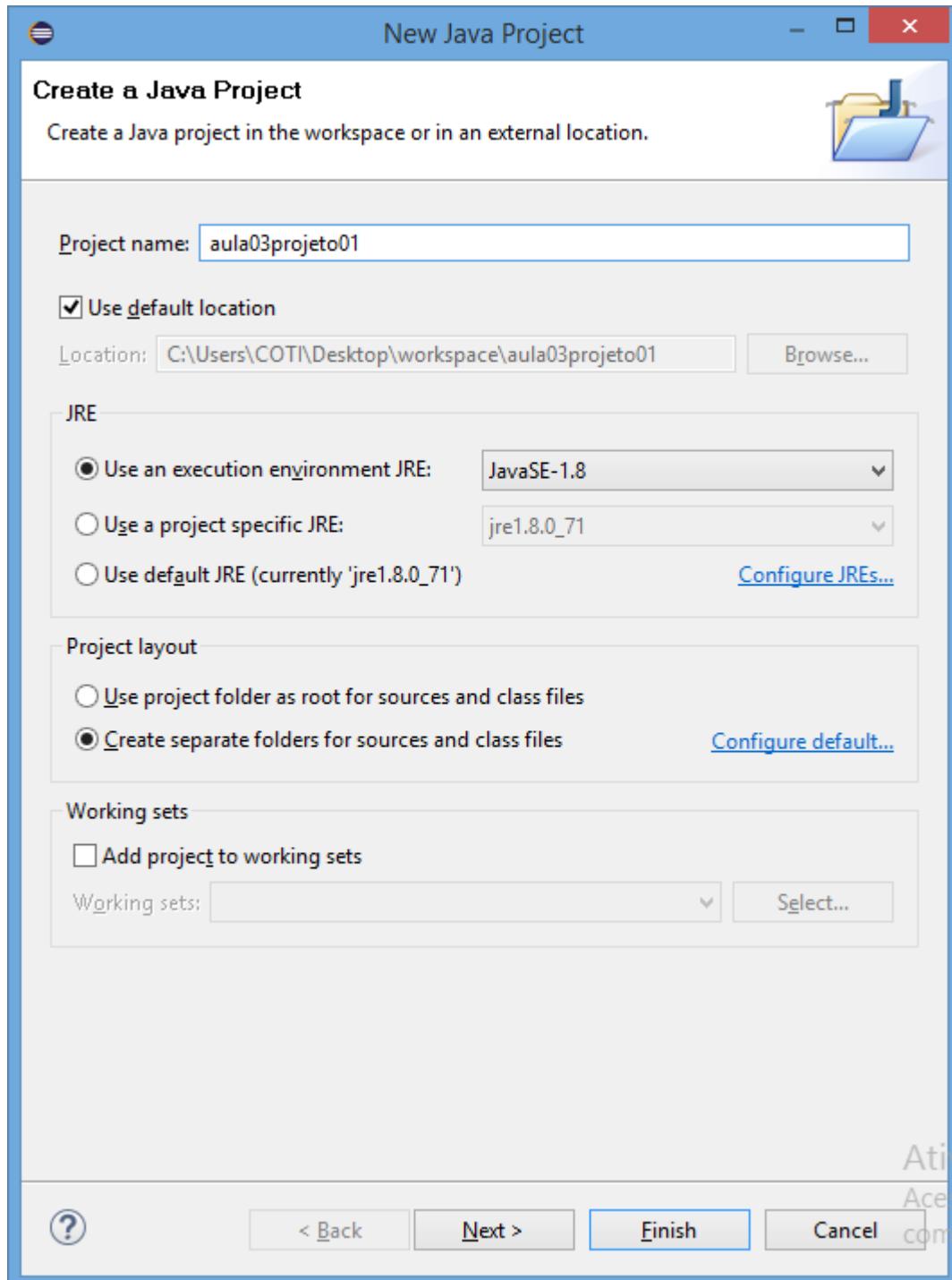
Sexta-feira, 13 de Maio de 2016

Collections. Listas, Sets e Queues. Relacionamentos de Associação.
Manipulação de coleções de dados.

Aula
03

Criando um novo projeto:

- File > New > Project





Criando Enums

```
package entities.types;

public enum EstadoCivil {

    Solteiro,
    Casado,
    Divorciado,
    Viuvo
}

package entities.types;

public enum Sexo {

    Masculino,
    Feminino
}
```

Criando uma entidade Pessoa:

/entities/Pessoa.java

```
package entities;

import entities.types.EstoadoCivil;
import entities.types.Sexo;

//classe javabean esta implementando a interface Comparable (sort, ordenação)
//quando implementamos Comparable, devemos programar o método compareTo
public class Pessoa implements Comparable<Pessoa>{

    private Integer idPessoa;
    private String nome;
    private Sexo sexo;
    private EstoadoCivil estoadoCivil;

    public Pessoa() {
        // TODO Auto-generated constructor stub
    }

    public Pessoa(Integer idPessoa, String nome, Sexo sexo, EstoadoCivil estoadoCivil) {
        this.idPessoa = idPessoa;
        this.nome = nome;
        this.sexo = sexo;
        this.estoadCivil = estoadoCivil;
    }
}
```



TREINAMENTO JAVA – BRQ/SP

Sexta-feira, 13 de Maio de 2016

Collections. Listas, Sets e Queues. Relacionamentos de Associação.
Manipulação de coleções de dados.

Aula
03

```
public Integer getIdPessoa() {
    return idPessoa;
}

public void setIdPessoa(Integer idPessoa) {
    this.idPessoa = idPessoa;
}

public String getNome() {
    return nome;
}

public void setNome(String nome) {
    this.nome = nome;
}

public Sexo getSexo() {
    return sexo;
}

public void setSexo(Sexo sexo) {
    this.sexo = sexo;
}

public EstadoCivil getEstadoCivil() {
    return estadoCivil;
}

public void setEstadoCivil(EstadoCivil estadoCivil) {
    this.estadoCivil = estadoCivil;
}

//Sobrescrita dos métodos da classe Object...
//equals -> define regra de comparação para objetos da classe
@Override //sobrescrita de método
public boolean equals(Object obj) {

    if(obj instanceof Pessoa){

        Pessoa p = (Pessoa) obj;

        if(p.getIdPessoa() != null){
            return p.getIdPessoa().equals(idPessoa);
        }
    }
    return false;
}
```



TREINAMENTO JAVA – BRQ/SP

Sexta-feira, 13 de Maio de 2016

Collections. Listas, Sets e Queues. Relacionamentos de Associação.
Manipulação de coleções de dados.

Aula
03

```
//hashCode -> define regra para ordenação e classificação
//geralmente a regra do hashCode é a mesma do equals
@Override //sobrescrita de metodo
public int hashCode() {
    return idPessoa.hashCode(); //ordenação pelo id..
}

//toString -> retorna uma string com o valor do objeto
@Override //sobrescrita de metodo
public String toString() {
    return idPessoa + ", " + nome + ", " + sexo + ", " + estadoCivil;
}

//método da interface Comparable
//utilizado para definir regras de ordenação em collections
@Override
public int compareTo(Pessoa p) {

    //regra de comparação pelo id..
    //if(p.getIdPessoa() != null){
        //ordem decrescente...
        //return p.getIdPessoa().compareTo(idPessoa);
    //}

    //ordem crescente...
    return nome.compareTo(p.getNome());
}
}
```

Comparable<T>

Existe uma interface especial em Java que é responsável por fazer comparações de objetos, que é a classe *Comparable*. Ela usada como padrão de comparação.

Essa comparação é feita através do método *compareTo(Object o)*, que recebe um objeto (se lembre que todos os objetos são derivados da classe *Object*, logo, todo e qualquer objeto que é possível criar em Java é derivado de *Object*).

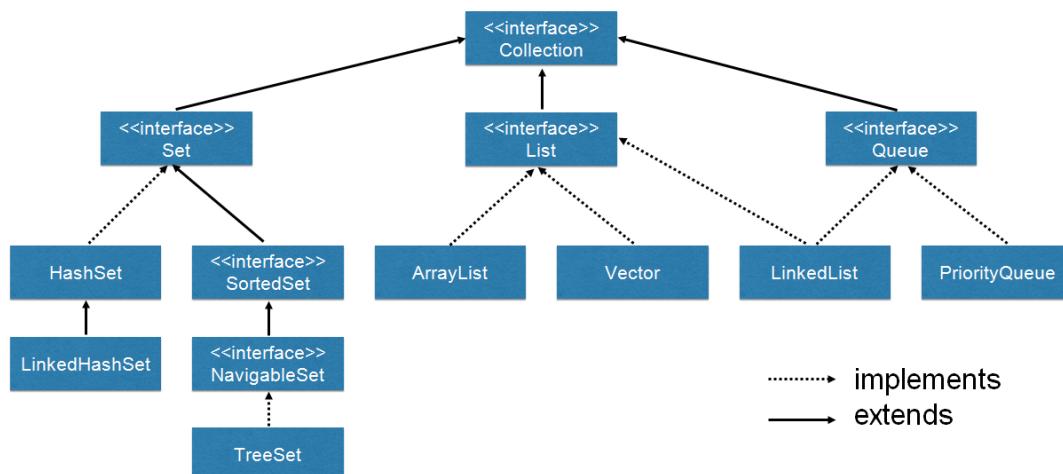
Collections

Coleções de dados

Uma Collection É um objeto onde podemos agrupar vários elementos. No dia-a-dia nos deparamos com várias situações onde as coleções estão presentes: uma **fila** de banco, uma **lista** de compras, uma **pilha** de livros, um **conjunto** de elementos, etc.

- **Collection:** O framework não possui implementação direta desta interface, porém, ela está no topo da hierarquia definindo operações que são comuns a todas as coleções;
- **Set:** Está diretamente relacionada com a idéia de conjuntos. Assim como um conjunto, as classes que implementam esta interface não podem conter elementos repetidos. Usaremos implementações de *SortedSet* para situações onde desejarmos ordenar os elementos;
- **List:** Também chamada de seqüência. É uma coleção ordenada, que ao contrário da interface *Set*, pode conter valores duplicados. Além disso, temos controle total sobre a posição onde se encontra cada elemento de nossa coleção, podendo acessar cada um deles pelo índice.
- **Queue:** Normalmente utilizamos esta interface quando queremos uma coleção do tipo FIFO (First-In-First-Out), também conhecida como fila.

Collection Interface





Manipulando listas:

```
package principal;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Iterator;
import java.util.List;

import entities.Pessoa;
import entities.types.EstadoCivil;
import entities.types.Sexo;

public class Main {

    public static void main(String[] args) {

        //criando objetos..
        Pessoa p1 = new Pessoa(1, "Ana", Sexo.Feminino, EstadoCivil.Solteiro);
        Pessoa p2 = new Pessoa(2, "Rui", Sexo.Masculino, EstadoCivil.Casado);
        Pessoa p3 = new Pessoa(3, "Leo", Sexo.Masculino, EstadoCivil.Casado);
        Pessoa p4 = new Pessoa(4, "Bia", Sexo.Feminino, EstadoCivil.Divorciado);
        Pessoa p5 = new Pessoa(5, "Max", Sexo.Masculino, EstadoCivil.Solteiro);

        //declarando uma lista de pessoas..
        List<Pessoa> lista = new ArrayList<Pessoa>();

        //adicionando objetos na lista..
        lista.add(p1); //adicionando..
        lista.add(p2); //adicionando..
        lista.add(p3); //adicionando..
        lista.add(p4); //adicionando..
        lista.add(p5); //adicionando..

        //imprimindo a quantidade de pessoas na lista..
        System.out.println("Quantidade de pessoas: " + lista.size());

        //recurso de ordenação para listas..
        //em collections, temos um método de ordenação chamado de sort
        //para que a lista possa ser ordenada pelo metodo sort, é necessário
        //que a classe que define o tipo da lista implemente uma
```



TREINAMENTO JAVA – BRQ/SP

Sexta-feira, 13 de Maio de 2016

Collections. Listas, Sets e Queues. Relacionamentos de Associação.
Manipulação de coleções de dados.

Aula
03

```
//interface chamada de Comparable
Collections.sort(lista);

//percorrendo a lista..
System.out.println("\nimprimindo com for..");
for(int i = 0; i < lista.size(); i++){
    //recuperar uma pessoa dentro da lista..
    Pessoa p = lista.get(i);
    //imprimindo..
    System.out.println("Pessoa: " + p); //toString()
}

System.out.println("\nimprimindo com foreach");
for(Pessoa p : lista){
    //imprimindo..
    System.out.println("Pessoa: " + p); //toString()
}

//podemos transformar a lista em um iterator (cursor)...
//e imprimi-la de uma forma diferente (mais antigo no java)

//colocar a lista em ordem inversa..
Collections.reverse(lista);

//convertendo a lista em um interator..
Iterator<Pessoa> iterator = lista.iterator();

//um iterator não é percorrido com for ou foreach, mas sim com while..
System.out.println("\nimprimindo com iterator");
while(iterator.hasNext()){

    //obter o elemento do iterator..
    Pessoa p = iterator.next();
    //imprimindo..
    System.out.println("Pessoa: " + p); //toString()
}
}
```



TREINAMENTO JAVA – BRQ/SP

Sexta-feira, 13 de Maio de 2016

Collections. Listas, Sets e Queues. Relacionamentos de Associação.
Manipulação de coleções de dados.

Aula
03

Executando:

Saída do programa...

```
Java - aula03_projeto01/src/prinipal/Main.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Problems Javadoc Declaration Console History
<terminated> Main (8) [Java Application] C:\Program Files (x86)\Java\jre1.8.0_71\bin\javaw.exe (14/05/2016 08:20:09)
Quantidade de pessoas: 5

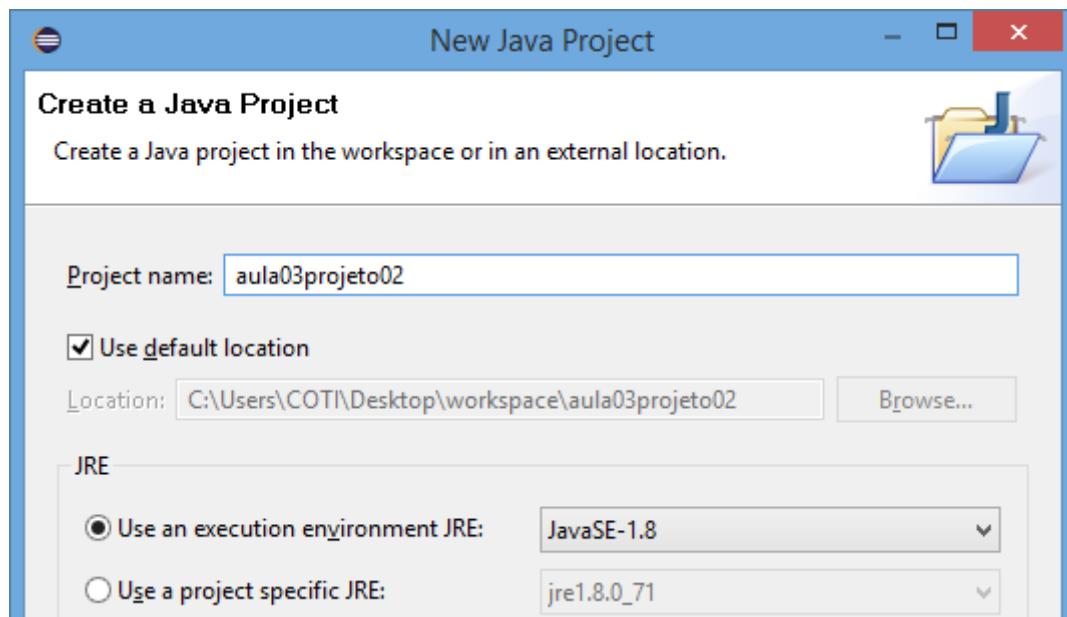
imprimindo com for..
Pessoa: 1, Ana, Feminino, Solteiro
Pessoa: 4, Bia, Feminino, Divorciado
Pessoa: 3, Leo, Masculino, Casado
Pessoa: 5, Max, Masculino, Solteiro
Pessoa: 2, Rui, Masculino, Casado

imprimindo com foreach
Pessoa: 1, Ana, Feminino, Solteiro
Pessoa: 4, Bia, Feminino, Divorciado
Pessoa: 3, Leo, Masculino, Casado
Pessoa: 5, Max, Masculino, Solteiro
Pessoa: 2, Rui, Masculino, Casado

imprimindo com iterator
Pessoa: 2, Rui, Masculino, Casado
Pessoa: 5, Max, Masculino, Solteiro
Pessoa: 3, Leo, Masculino, Casado
Pessoa: 4, Bia, Feminino, Divorciado
Pessoa: 1, Ana, Feminino, Solteiro

Ativar o Windows
Acesse as configurações do computador para ativar o Windows.
```

Novo projeto:





Criando uma entidade Produto:

```
package entities;

import entities.types.Status;

public class Produto implements Comparable<Produto>{

    private Integer idProduto;
    private String nome;
    private Double preco;
    private Integer quantidade;
    private Status status;

    public Produto() {
        // TODO Auto-generated constructor stub
    }

    public Produto(Integer idProduto, String nome,
                   Double preco, Integer quantidade, Status status) {
        this.idProduto = idProduto;
        this.nome = nome;
        this.preco = preco;
        this.quantidade = quantidade;
        this.status = status;
    }

    public Integer getIdProduto() {
        return idProduto;
    }

    public void setIdProduto(Integer idProduto) {
        this.idProduto = idProduto;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public Double getPreco() {
        return preco;
    }

    public void setPreco(Double preco) {
        this.preco = preco;
    }
}
```



TREINAMENTO JAVA – BRQ/SP

Sexta-feira, 13 de Maio de 2016

Collections. Listas, Sets e Queues. Relacionamentos de Associação.
Manipulação de coleções de dados.

Aula
03

```
public Integer getQuantidade() {
    return quantidade;
}

public void setQuantidade(Integer quantidade) {
    this.quantidade = quantidade;
}

public Status getStatus() {
    return status;
}

public void setStatus(Status status) {
    this.status = status;
}

@Override
public boolean equals(Object obj) {

    if(obj instanceof Produto){

        Produto p = (Produto) obj;

        if(p.getIdProduto() != null){
            return p.getIdProduto().equals(idProduto);
        }
    }
    return false;
}

@Override
public int hashCode() {
    return idProduto.hashCode();
}

@Override
public String toString() {
    return idProduto + ", " + nome + ", " + preco
           + ", " + quantidade + ", " + status;
}

@Override
public int compareTo(Produto p) {
    return nome.compareTo(p.getNome());
}

}
```



```
package entities.types;

public enum Status {

    Disponivel,
    Esgotado

}
```

Utilizando a interface Set<T>

```
package principal;

import java.util.LinkedHashSet;
import java.util.Set;

import entities.Produto;
import entities.types.Status;

public class Main {

    public static void main(String[] args) {

        //criando objetos...
        Produto p1 = new Produto(4, "Mouse", 30.0, 5, Status.Disponivel);
        Produto p2 = new Produto(3, "Celular", 250.0, 0, Status.Esgotado);
        Produto p3 = new Produto(1, "PenDrive", 20.0, 10, Status.Disponivel);
        Produto p4 = new Produto(2, "Mochila", 100.0, 0, Status.Esgotado);

        //Set -> tipo de collection que não permite objetos duplicados..
        //utiliza o equals para verificar se 2 objetos são iguais..

        //HashSet -> organiza os objetos pelo criterio do hashCode
        //TreeSet -> organiza os objetos pelo criterio do compareTo (Comparable)
        //LinkedHashSet -> mantem a ordem de entrada dos objetos no Set

        Set<Produto> lista = new LinkedHashSet<Produto>();

        lista.add(p1); //adicionando..
        lista.add(p2); //adicionando..
        lista.add(p3); //adicionando..
        lista.add(p4); //adicionando..
```



TREINAMENTO JAVA – BRQ/SP

Sexta-feira, 13 de Maio de 2016

Collections. Listas, Sets e Queues. Relacionamentos de Associação.
Manipulação de coleções de dados.

Aula
03

```
//imprimindo..
for(Produto p : lista){ //foreach
    System.out.println("Produto -> " + p); //toString()
}
}
```

Executando:

```
Java - aula03_projeto02/src/principal/Main.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
File Edit Source Refactor Navigate Search Project Run Window Help
Problems Javadoc Declaration Console History
<terminated> Main (9) [Java Application] C:\Program Files (x86)\Java\jre1.8.0_71\bin\javaw.exe (14/05/2016 13:04:52)
Produto -> 4, Mouse, 30.0, 5, Disponivel
Produto -> 3, Celular, 250.0, 0, Esgotado
Produto -> 1, PenDrive, 20.0, 10, Disponivel
Produto -> 2, Mochila, 100.0, 0, Esgotado
```

Java - aula03_projeto02/src/principal/Main.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

Problems Javadoc Declaration Console History

<terminated> Main (9) [Java Application] C:\Program Files (x86)\Java\jre1.8.0_71\bin\javaw.exe (14/05/2016 13:04:52)

Produto -> 4, Mouse, 30.0, 5, Disponivel
Produto -> 3, Celular, 250.0, 0, Esgotado
Produto -> 1, PenDrive, 20.0, 10, Disponivel
Produto -> 2, Mochila, 100.0, 0, Esgotado

entities

entities.types

principal

Main.java

JRE System Library [Java]

aula03_projeto03

aula03_projeto04

aula03_projeto05

aula03_projeto06

RemoteSystemsTempFiles

17 Produto p4 = new Produto(2, "Mochila", 100.0, 0, Status.Esgotado);
18
19 //Set -> tipo de collection que não permite objetos duplicados..
20 //utiliza o equals para verificar se 2 objetos são iguais..
21
22 //HashSet -> organiza os objetos pelo criterio do hashCode
23 //TreeSet -> organiza os objetos pelo criterio do compareTo (Comparable)
24 //LinkedHashSet -> mantem a ordem de entrada dos objetos no Set
25
26 Set<Produto> lista = new LinkedHashSet<Produto>();
27
28 lista.add(p1); //adicionando..
29 lista.add(p2); //adicionando..
30 lista.add(p3); //adicionando..
31 lista.add(p4); //adicionando..
32
33 //imprimindo..
34 for(Produto p : lista){ //foreach
35 System.out.println("Produto -> " + p); //toString()
36 }

Ativar o Windows
Acesse as configurações do computador para ativar o Windows.

É importante notar que TreeSet, HashSet e LinkedHashSet implementam a interface “Set”, ou seja, temos os mesmos métodos para as 3 estruturas, o que difere cada uma é a forma com que é implementado o algoritmo, por exemplo, na HashSet já falamos que ela usa HashTable em sua implementação, que por sinal é muito rápido mas não garante a ordenação dos seus elementos

O TreeSet implementa um algoritmo conhecido por red-black tree ou árvore rubro-negra. Sua principal característica é que ele é o único Set que implementa a interface SortedSet em vez de Set diretamente, mas de qualquer forma SortedSet implementa Set, assim continuamos tendo os mesmo métodos no TreeSet.



TREINAMENTO JAVA – BRQ/SP

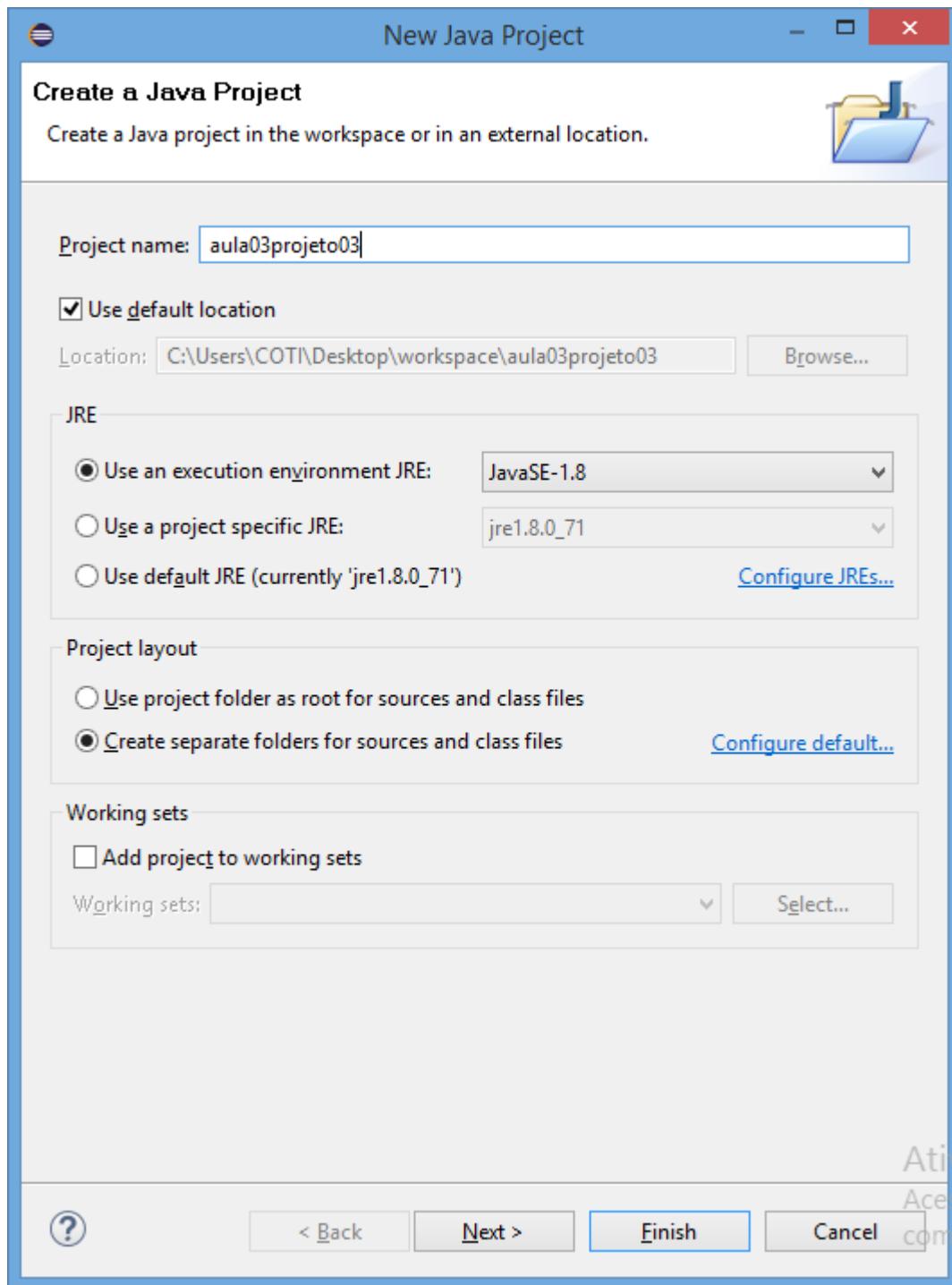
Sexta-feira, 13 de Maio de 2016

Collections. Listas, Sets e Queues. Relacionamentos de Associação.
Manipulação de coleções de dados.

Aula
03

Novo projeto:

- File / New / Project





Criando uma entidade Produto:

```
package entities;

public class Produto implements Comparable<Produto>{

    private Integer idProduto;
    private String nome;
    private Double preco;

    public Produto() {
        // TODO Auto-generated constructor stub
    }

    public Produto(Integer idProduto, String nome, Double preco) {
        this.idProduto = idProduto;
        this.nome = nome;
        this.preco = preco;
    }

    public Integer getIdProduto() {
        return idProduto;
    }

    public void setIdProduto(Integer idProduto) {
        this.idProduto = idProduto;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public Double getPreco() {
        return preco;
    }

    public void setPreco(Double preco) {
        this.preco = preco;
    }

    @Override
    public String toString() {
        return "Produto [idProduto=" + idProduto
            + ", nome=" + nome + ", preco=" + preco + "]";
    }
}
```



```
@Override
public int hashCode() {
    return idProduto.hashCode();
}

@Override
public boolean equals(Object obj) {

    //assertiva -> condição ? entao : senao
    return obj instanceof Produto
        ? ((Produto)obj).getIdProduto() != null
        ? ((Produto)obj).getIdProduto()
            .equals(idProduto)
        : false : false;
}

@Override
public int compareTo(Produto p) {
    return idProduto.compareTo(p.getIdProduto());
}

}
```

Criando uma classe de controle para manipular uma lista de Produtos:

```
package control;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

import entities.Produto;

public class ControleProduto {

    //atributo -> lista de produtos..
    private List<Produto> listagemProdutos; //null

    //construtor..
    public ControleProduto() {
        //inicializar o atributo da classe..
        listagemProdutos = new ArrayList<Produto>();
    }
}
```



TREINAMENTO JAVA – BRQ/SP

Sexta-feira, 13 de Maio de 2016

Collections. Listas, Sets e Queues. Relacionamentos de Associação.
Manipulação de coleções de dados.

Aula
03

```
//método para adicionar um produto na lista..
public void adicionarProduto(Produto p){
    listagemProdutos.add(p);
}

//método para buscar um produto dentro da lista pelo id..
public Produto buscarPorId(Integer idProduto){
    //varrendo a lista.
    for(Produto p : listagemProdutos){
        //verificando se o produto da lista tem o mesmo id do parametro..
        if(p.getIdProduto().equals(idProduto)){
            return p; //retornando o produto..
        }
    }
    return null; //vazio..
}

//método para retornar produtos pelo nome (contendo).. 
public List<Produto> buscarPorNome(String nome){

    //criando uma lista vazia..
    List<Produto> lista = new ArrayList<Produto>();

    //varrer a listagem de produtos da classe..
    for(Produto p : listagemProdutos){
        //verificar se o produto contem o nome passado no metodo..
        if(p.getNome().toLowerCase().contains(nome.toLowerCase())){
            lista.add(p); //adicionar na lista de resposta..
        }
    }

    //retornar a lista..
    return lista;
}

//método para limpar a lista..
public void limparListagemProdutos(){
    listagemProdutos.clear();
}
```



TREINAMENTO JAVA – BRQ/SP

Sexta-feira, 13 de Maio de 2016

Collections. Listas, Sets e Queues. Relacionamentos de Associação.
Manipulação de coleções de dados.

Aula
03

```
//método para ordenar a lista..
public void ordenarListagemProdutos(){
    Collections.sort(listagemProdutos);
}

//método para retornar a listagem de produtos..
public List<Produto> getListagemProdutos() {
    return listagemProdutos;
}
```

Executando:
Método Main

```
package principal;

import control.ControleProduto;
import entities.Produto;

public class Main {

    public static void main(String[] args) {

        ControleProduto c = new ControleProduto(); //classe de controle..

        Produto p1 = new Produto(1, "Livro de Java", 80.0);
        Produto p2 = new Produto(3, "Livro de Oracle", 100.0);
        Produto p3 = new Produto(2, "Notebook", 2000.0);

        //adicionar os produtos no controle..
        c.adicionarProduto(p1);
        c.adicionarProduto(p2);
        c.adicionarProduto(p3);

        //ordenar a lista..
        c.ordenarListagemProdutos();

        //imprimir a listagem de produtos..
        for(Produto p : c.getListagemProdutos()){
            System.out.println(p); //toString()..
        }
    }
}
```



TREINAMENTO JAVA – BRQ/SP

Sexta-feira, 13 de Maio de 2016

Collections. Listas, Sets e Queues. Relacionamentos de Associação.
Manipulação de coleções de dados.

Aula
03

```
//buscar 1 produto pelo id..
Produto prod = c.buscarPorId(1);
if(prod != null){ //se produto foi encontrado..
    System.out.println("\n" + prod);
}
else{
    System.out.println("\nProduto não encontrado.");
}

System.out.println("\nProdutos com nome: Livro");
//buscar produtos pelo nome..
for(Produto p : c.buscarPorNome("Livro")){
    System.out.println(p); //imprimindo..
}
}

-----
-----
```

Saída do programa:

The screenshot shows the Eclipse IDE interface with the title bar "Java - aula03_projeto03/src/principal/Main.java - Eclipse". The menu bar includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The toolbar has various icons for file operations. The left sidebar shows the project structure with packages like control, entities, principal, and JRE System Library [Java]. Below the sidebar is the code editor with Main.java open. The code implements a controller to manage products, including methods to add products to a list, sort them, and print them. It also includes logic to search for a product by ID and print its details if found. The right side of the interface is the "Console" view, which displays the execution results. The output shows three products added to the list: "Produto [idProduto=1, nome=Livro de Java, preco=80.0]", "Produto [idProduto=2, nome=Notebook, preco=2000.0]", and "Produto [idProduto=3, nome=Livro de Oracle, preco=100.0]". When the program searches for the first product again, it prints its details: "Produto [idProduto=1, nome=Livro de Java, preco=80.0]". A status bar at the bottom right indicates "Ativar o Windows" and "Acesse as configurações do computador para ativar o Windows".

```
Java - aula03_projeto03/src/principal/Main.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
File Edit Source Refactor Navigate Search Project Run Window Help
Problems Javadoc Declaration Console History
<terminated> Main (10) [Java Application] C:\Program Files (x86)\Java\jre1.8.0_71\bin\javaw.exe (14/05/2016 13:10:10)
Produto [idProduto=1, nome=Livro de Java, preco=80.0]
Produto [idProduto=2, nome=Notebook, preco=2000.0]
Produto [idProduto=3, nome=Livro de Oracle, preco=100.0]

Produto [idProduto=1, nome=Livro de Java, preco=80.0]

Produtos com nome: Livro
Produto [idProduto=1, nome=Livro de Java, preco=80.0]
Produto [idProduto=3, nome=Livro de Oracle, preco=100.0]

control
  ControleProduto
entities
  Produto.java
principal
  Main.java
JRE System Library [Java]
  aula03_projeto04
  aula03_projeto05
  aula03_projeto06
  RemoteSystemsTempFiles

15  //adicionar os produtos no controle..
16  c.adicionarProduto(p1);
17  c.adicionarProduto(p2);
18  c.adicionarProduto(p3);
19
20
21  //ordenar a lista..
22  c.ordenarListagemProdutos();
23
24  //imprimir a listagem de produtos..
25  for(Produto p : c.getListagemProdutos()){
26      System.out.println(p); //toString()..
27  }
28
29
30  //buscar 1 produto pelo id..
31  Produto prod = c.buscarPorId(1);
32  if(prod != null){ //se produto foi encontrado..
33      System.out.println("\n" + prod);
34  }

Ativar o Windows
Acesse as configurações do
computador para ativar o Windows.
```



Novo projeto:

```
package entities;

public class Funcionario implements Comparable<Funcionario>{

    private Integer idFuncionario;
    private String nome;
    private Double salario;

    public Funcionario() {
        // TODO Auto-generated constructor stub
    }

    public Funcionario(Integer idFuncionario,
                       String nome, Double salario) {
        super();
        this.idFuncionario = idFuncionario;
        this.nome = nome;
        this.salario = salario;
    }

    public Integer getIdFuncionario() {
        return idFuncionario;
    }

    public void setIdFuncionario(Integer idFuncionario) {
        this.idFuncionario = idFuncionario;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public Double getSalario() {
        return salario;
    }

    public void setSalario(Double salario) {
        this.salario = salario;
    }

    @Override
    public int hashCode() {
        return idFuncionario.hashCode();
    }
}
```



```
@Override
public boolean equals(Object obj) {

    if(obj instanceof Funcionario){
        Funcionario f = (Funcionario) obj;
        if(f.getIdFuncionario() != null){
            return f.getIdFuncionario()
                .equals(idFuncionario);
        }
    }

    return false;
}

@Override
public String toString() {
    return "Funcionario [idFuncionario=" + idFuncionario
           + ", nome=" + nome + ", salario=" + salario + "]";
}

@Override
public int compareTo(Funcionario f) {
    return idFuncionario.compareTo(f.getIdFuncionario());
}
}
```

Criando uma interface para definir um contrato de operações com Funcionario:

```
package contracts;

import java.util.List;

import entities.Funcionario;

//nível abstrato (abstração)
public interface IControleFuncionario {

    //adicionar 1 funcionario na lista
    void adicionarFuncionario(Funcionario f);

    //limpar o conteúdo da lista
    void limparListagemFuncionarios();

    //ordenar a lista
    void ordenarListagemFuncionarios();
}
```



```
//buscar 1 Funcionario pelo id dentro da lista
Funcionario obterPorId(Integer idFuncionario);

//buscar funcionarios pelo nome dentro da lista
List<Funcionario> obterPorNome(String nome);

//buscar funcionarios pelo salario dentro da lista
List<Funcionario> obterPorSalario(Double salarioIni, Double salarioFim);

//retornar todos os funcionarios da lista
List<Funcionario> getListagemFuncionarios();
}
```

Implementando a interface:

```
package control;

import java.util.ArrayList;
import java.util.Collections;
import java.util.LinkedHashSet;
import java.util.List;
import java.util.Set;

import contracts.IControleFuncionario;
import entities.Funcionario;

public class ControleFuncionario implements IControleFuncionario {

    //atributo..
    private Set<Funcionario> lista;

    //construtor..
    public ControleFuncionario() {
        //LinkedHashSet -> mantem a ordem de entrada dos objetos..
        lista = new LinkedHashSet<Funcionario>();
    }

    @Override
    public void adicionarFuncionario(Funcionario f) {
        lista.add(f); //adicionando..
    }

    @Override
    public void limparListagemFuncionarios() {
        lista.clear();
    }
}
```



TREINAMENTO JAVA – BRQ/SP

Sexta-feira, 13 de Maio de 2016

Collections. Listas, Sets e Queues. Relacionamentos de Associação.
Manipulação de coleções de dados.

Aula
03

```
@Override
public void ordenarListagemFuncionarios() {

    List<Funcionario> auxiliar = new ArrayList<Funcionario>(lista);
    Collections.sort(auxiliar); //ordenação..

    lista = new LinkedHashSet<Funcionario>(auxiliar);
}

@Override
public Funcionario obterPorId(Integer idFuncionario) {

    for(Funcionario f : lista){
        if(f.getIdFuncionario().equals(idFuncionario)){
            return f; //retornar o funcionario..
        }
    }

    return null; //vazio..
}

@Override
public List<Funcionario> obterPorNome(String nome) {

    List<Funcionario> resultado = new ArrayList<Funcionario>();

    for(Funcionario f : lista){
        if(f.getNome().toUpperCase().contains(nome.toUpperCase())){
            resultado.add(f);
        }
    }

    return resultado;
}

@Override
public List<Funcionario> obterPorSalario(Double salarioIni, Double salarioFim) {

    List<Funcionario> resultado = new ArrayList<Funcionario>();

    for(Funcionario f : lista){
        if(f.getSalario() >= salarioIni && f.getSalario() <= salarioFim){
            resultado.add(f);
        }
    }

    return resultado;
}
```



TREINAMENTO JAVA – BRQ/SP

Sexta-feira, 13 de Maio de 2016

Collections. Listas, Sets e Queues. Relacionamentos de Associação.
Manipulação de coleções de dados.

Aula
03

```
@Override
public List<Funcionario> getListagemFuncionarios() {
    //criando um ArrayList de funcionario e retornando-o...
    return new ArrayList<Funcionario>(lista);
}

-----
-----
```

Testando no Main:

```
package principal;

import control.ControleFuncionario;
import entities.Funcionario;

public class Main {

    public static void main(String[] args) {

        Funcionario f1 = new Funcionario(1, "Joao", 2000.0);
        Funcionario f2 = new Funcionario(3, "Pedro", 3000.0);
        Funcionario f3 = new Funcionario(2, "Maria", 4000.0);

        ControleFuncionario c = new ControleFuncionario();

        c.adicionarFuncionario(f1);
        c.adicionarFuncionario(f2);
        c.adicionarFuncionario(f3);

        c.ordenarListagemFuncionarios();

        System.out.println("\nTodos os Funcionarios:");
        for(Funcionario f : c.getListagemFuncionarios()){
            System.out.println(f); //imprimindo..
        }

        System.out.println("\nFuncionarios por Nome -> 'A'");
        for(Funcionario f : c.obterPorNome("A")){
            System.out.println(f);
        }

        System.out.println("\nFuncionarios com Salario entre 2500 e 4500");
        for(Funcionario f : c.obterPorSalario(2500.0, 4500.0)){
            System.out.println(f);
        }
    }
}
```



TREINAMENTO JAVA – BRQ/SP

Sexta-feira, 13 de Maio de 2016

Collections. Listas, Sets e Queues. Relacionamentos de Associação.
Manipulação de coleções de dados.

Aula
03

```
//buscar por id..
System.out.println("\nConsulta de Funcionario por Id");
Funcionario f = c.obterPorId(2);

if(f != null){
    System.out.println(f); //toString()
}
else{
    System.out.println("Funcionario não encontrado.");
}
}
```

Saida do programa:

```
Java - aula03_projeto04/src/principal/Main.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
File Editors View Window Help Java EE Java
Problems Javadoc Declaration Console History <terminated> Main (11) [Java Application] C:\Program Files (x86)\Java\jre1.8.0_71\bin\javaw.exe (14/05/2016 13:15:27)
Todos os Funcionarios:
Funcionario [idFuncionario=1, nome=Joao, salario=2000.0]
Funcionario [idFuncionario=2, nome=Maria, salario=4000.0]
Funcionario [idFuncionario=3, nome=Pedro, salario=3000.0]

Funcionarios por Nome -> 'A'
Funcionario [idFuncionario=1, nome=Joao, salario=2000.0]
Funcionario [idFuncionario=2, nome=Maria, salario=4000.0]

Funcionarios com Salario entre 2500 e 4500
Funcionario [idFuncionario=2, nome=Maria, salario=4000.0]
Funcionario [idFuncionario=3, nome=Pedro, salario=3000.0]

Consulta de Funcionario por Id
Funcionario [idFuncionario=2, nome=Maria, salario=4000.0]
```

Todos os Funcionarios:

```
Funcionario [idFuncionario=1, nome=Joao, salario=2000.0]
Funcionario [idFuncionario=2, nome=Maria, salario=4000.0]
Funcionario [idFuncionario=3, nome=Pedro, salario=3000.0]
```

Funcionarios por Nome -> 'A'

```
Funcionario [idFuncionario=1, nome=Joao, salario=2000.0]
Funcionario [idFuncionario=2, nome=Maria, salario=4000.0]
```

Funcionarios com Salario entre 2500 e 4500

```
Funcionario [idFuncionario=2, nome=Maria, salario=4000.0]
Funcionario [idFuncionario=3, nome=Pedro, salario=3000.0]
```

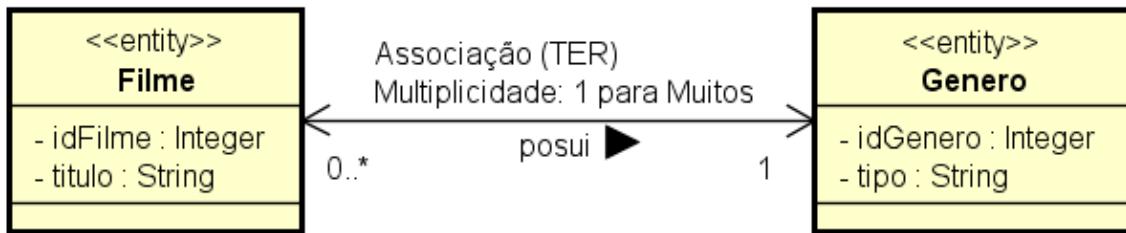
Consulta de Funcionario por Id

```
Funcionario [idFuncionario=2, nome=Maria, salario=4000.0]
```



Novo projeto:

Definindo uma modelagem de 1 para muitos



```
package entities;

public class Filme {

    private Integer idFilme;
    private String titulo;
    private Genero genero; //associação (TER-1)

    public Filme() {
        // TODO Auto-generated constructor stub
    }

    public Filme(Integer idFilme, String titulo) {
        this.idFilme = idFilme;
        this.titulo = titulo;
    }

    public Filme(Integer idFilme, String titulo, Genero genero) {
        this(idFilme, titulo);
        this.genero = genero;
    }

    public Integer getIdFilme() {
        return idFilme;
    }

    public void setIdFilme(Integer idFilme) {
        this.idFilme = idFilme;
    }

    public String getTitulo() {
        return titulo;
    }

    public void setTitulo(String titulo) {
        this.titulo = titulo;
    }
}
```



TREINAMENTO JAVA – BRQ/SP

Sexta-feira, 13 de Maio de 2016

Collections. Listas, Sets e Queues. Relacionamentos de Associação.
Manipulação de coleções de dados.

Aula
03

```
public Genero getGenero() {
    return genero;
}

public void setGenero(Genero genero) {
    this.genero = genero;
}

@Override
public String toString() {
    return "Filme [idFilme=" + idFilme + ", titulo="
           + titulo + "]";
}
}

package entities;

import java.util.List;

public class Genero {

    private Integer idGenero;
    private String tipo;
    private List<Filme> filmes; //Associação (muitos)

    public Genero() {
        // TODO Auto-generated constructor stub
    }

    public Genero(Integer idGenero, String tipo) {
        this.idGenero = idGenero;
        this.tipo = tipo;
    }

    public Genero(Integer idGenero, String tipo, List<Filme> filmes)
    {
        this(idGenero, tipo);
        this.filmes = filmes;
    }

    public Integer getIdGenero() {
        return idGenero;
    }

    public void setIdGenero(Integer idGenero) {
        this.idGenero = idGenero;
    }
}
```



```
public String getTipo() {
    return tipo;
}

public void setTipo(String tipo) {
    this.tipo = tipo;
}

public List<Filme> getFilmes() {
    return filmes;
}

public void setFilmes(List<Filme> filmes) {
    this.filmes = filmes;
}

@Override
public String toString() {
    return "Genero [idGenero=" + idGenero + ", tipo="
           + tipo + "]";
}
-----
```

Criando uma classe para impressão dos dados do Genero e Filmes

```
package output;

import entities.Filme;
import entities.Genero;

public class OutputGenero {

    //impressão no console..
    public void imprimir(Genero g){

        System.out.println("Id Genero...: " + g.getIdGenero());
        System.out.println("Tipo.....: " + g.getTipo());

        //verificando se o genero contem filmes..
        if(g.getFilmes() != null && g.getFilmes().size() > 0){

            System.out.println("\tFilmes:");
        }
    }
}
```



TREINAMENTO JAVA – BRQ/SP

Sexta-feira, 13 de Maio de 2016

Collections. Listas, Sets e Queues. Relacionamentos de Associação.
Manipulação de coleções de dados.

Aula
03

```
for(Filme f : g.getFilmes()){

    System.out.println("\tId do Filme....: " + f.getIdFilme());
    System.out.println("\tTitulo.....: " + f.getTitulo());
    System.out.println("\t...");

}

}

-----
```

Testando:

```
package principal;

import java.util.ArrayList;

import entities.Filme;
import entities.Genero;
import output.OutputGenero;

public class Main {

    public static void main(String[] args) {

        Genero g1 = new Genero(1, "Ação"); //instanciando a entidade..
        g1.setFilmes(new ArrayList<Filme>()); //instanciando a lista..

        Genero g2 = new Genero(2, "Aventura"); //instanciando a entidade..
        g2.setFilmes(new ArrayList<Filme>()); //instanciando a lista..

        //adicionar filmes aos generos..
        g1.getFilmes().add(new Filme(1, "Guerra Civil"));
        g1.getFilmes().add(new Filme(2, "Batman v Superman"));

        g2.getFilmes().add(new Filme(3, "Star Wars - Rogue One"));
        g2.getFilmes().add(new Filme(4, "X-men Apocalipse"));
        g2.getFilmes().add(new Filme(5, "Doutor Estranho"));

        //imprimindo..
        OutputGenero out = new OutputGenero();
        out.imprimir(g1); //imprimindo..
        out.imprimir(g2); //imprimindo..
    }

}
```



TREINAMENTO JAVA – BRQ/SP

Sexta-feira, 13 de Maio de 2016

Collections. Listas, Sets e Queues. Relacionamentos de Associação.
Manipulação de coleções de dados.

Aula
03

Saida do programa:

The screenshot shows the Eclipse IDE interface with the title bar "Java - aula03_projeto05/src/principal/Main.java - Eclipse". The Console view displays the program's output:

```
<terminated> Main [12] [Java Application] C:\Program Files (x86)\Java\jre1.8.0_71\bin\javaw.exe (14/05/2016 13:18:21)
Id Genero....: 1
Tipo.....: Ação
Filmes:
Id do Filme....: 1
Titulo.....: Guerra Civil
...
Id do Filme....: 2
Titulo.....: Batman v Superman
...
Id Genero....: 2
Tipo.....: Aventura
Filmes:
Id do Filme....: 3
Titulo.....: Star Wars - Rogue One
...
Id do Filme....: 4
Titulo.....: X-men Apocalipse
...
Id do Filme....: 5
Titulo.....: Doutor Estranho
...
```

The code editor below shows the Java code for the Main class:

```
32
33 }
34
35
36
```

The taskbar at the bottom shows various application icons, and the system tray indicates the date and time as 14/05/2016 13:18.

```
Id Genero....: 1
Tipo.....: Ação
Filmes:
Id do Filme....: 1
Titulo.....: Guerra Civil
...
Id do Filme....: 2
Titulo.....: Batman v Superman
...
Id Genero....: 2
Tipo.....: Aventura
Filmes:
Id do Filme....: 3
Titulo.....: Star Wars - Rogue One
...
Id do Filme....: 4
Titulo.....: X-men Apocalipse
...
Id do Filme....: 5
Titulo.....: Doutor Estranho
...
```



Novo projeto:

Definindo uma entidade Cliente:

```
package entities;

public class Cliente implements Comparable<Cliente>{

    private Integer idCliente;
    private String nome;

    public Cliente() {
        // TODO Auto-generated constructor stub
    }

    public Cliente(Integer idCliente, String nome) {
        super();
        this.idCliente = idCliente;
        this.nome = nome;
    }

    public Integer getIdCliente() {
        return idCliente;
    }

    public void setIdCliente(Integer idCliente) {
        this.idCliente = idCliente;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

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

    @Override
    public int compareTo(Cliente c) {
        return idCliente.compareTo(c.getIdCliente());
    }
}
```



Queue (Fila)

Nesta interface, o elemento é inserido na parte de trás da fila. Esta operação é chamada de **enfileiramento**. Este mesmo elemento sai a partir da frente da fila, operação chamada de **desenfileiramento**. Esse procedimento de entrada e saída recebe o nome de fila, ou **FIFO** (first-in first-out), ou seja, “primeiro a entrar, primeiro a sair”.

Exemplo:

```
package principal;
```

```
import java.util.PriorityQueue;  
import java.util.Queue;
```

```
import entities.Cliente;
```

```
public class Main {
```

```
    public static void main(String[] args) {
```

```
        //Queue -> manipulando fila (FIFO)
```

```
        //Existem 2 tipos de fila: LinkedList e PriorityQueue
```

```
        //PriorityQueue -> precisa que o objeto implemente Comparable!
```

```
        //LinkedList -> não precisa da implementação de Comparable...  
        //mostra os elementos da fila na ordem em que entraram na fila
```

```
        //FILA - FIFO (First In First Out)
```

```
        Queue<Cliente> fila = new PriorityQueue<Cliente>();
```

```
        //adicionar elementos na fila..
```

```
        fila.add(new Cliente(1, "Ana"));  
        fila.add(new Cliente(4, "Rui"));  
        fila.add(new Cliente(3, "Bia"));  
        fila.add(new Cliente(2, "Leo"));
```

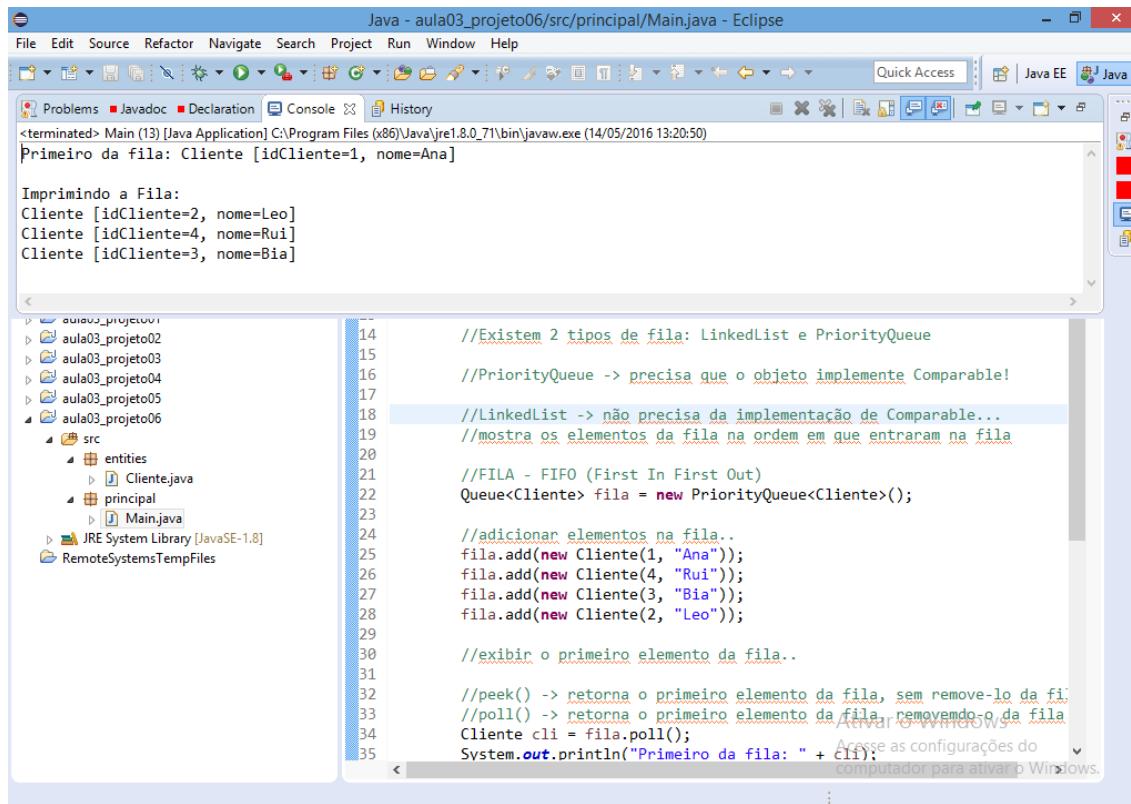
```
        //exibir o primeiro elemento da fila..
```



```
//peek() -> retorna o primeiro elemento da fila, sem remove-lo da fila..
//poll() -> retorna o primeiro elemento da fila, removendo-o da fila..
Cliente cli = fila.poll();
System.out.println("Primeiro da fila: " + cli);

//imprimir a fila..
System.out.println("\nImprimindo a Fila:");
for(Cliente c : fila){
    System.out.println(c);
}
}
```

Executando:



The screenshot shows the Eclipse IDE interface with the title bar "Java - aula03_projeto06/src/principal/Main.java - Eclipse". The console tab is active, displaying the following output:

```
Java - aula03_projeto06/src/principal/Main.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
File Edit Source Refactor Navigate Search Project Run Window Help
Problems Javadoc Declaration Console History
<terminated> Main (13) [Java Application] C:\Program Files (x86)\Java\jre1.8.0_71\bin\javaw.exe (14/05/2016 13:20:50)
Primeiro da fila: Cliente [idCliente=1, nome=Ana]

Imprimindo a Fila:
Cliente [idCliente=2, nome=Leo]
Cliente [idCliente=4, nome=Rui]
Cliente [idCliente=3, nome=Bia]
```

The left pane shows the project structure under "Project Explorer". The "src" folder contains "entities" (with "Cliente.java") and "principal" (with "Main.java"). The "JRE System Library [JavaSE-1.8]" and "RemoteSystemsTempFiles" are also listed.

The right pane shows the code editor with the following Java code:

```
14 //Existem 2 tipos de fila: LinkedList e PriorityQueue
15
16 //PriorityQueue -> precisa que o objeto implemente Comparable!
17
18 //LinkedList -> não precisa da implementação de Comparable...
19 //mostra os elementos da fila na ordem em que entraram na fila
20
21 //FILA - FIFO (First In First Out)
22 Queue<Cliente> fila = new PriorityQueue<Cliente>();
23
24 //adicionar elementos na fila..
25 fila.add(new Cliente(1, "Ana"));
26 fila.add(new Cliente(4, "Rui"));
27 fila.add(new Cliente(3, "Bia"));
28 fila.add(new Cliente(2, "Leo"));
29
30 //exibir o primeiro elemento da fila..
31
32 //peek() -> retorna o primeiro elemento da fila, sem remove-lo da fila
33 //poll() -> retorna o primeiro elemento da fila, removendo-o da fila
34 Cliente cli = fila.poll();
35 System.out.println("Primeiro da fila: " + cli);
```

Primeiro da fila: Cliente [idCliente=1, nome=Ana]

Imprimindo a Fila:
Cliente [idCliente=2, nome=Leo]
Cliente [idCliente=4, nome=Rui]
Cliente [idCliente=3, nome=Bia]