TREINAMENTO JAVA – BRQ/SP

Quarta-feira, 11 de Maio de 2016

Java Orientado a Objetos (Classes, Atributos e Métodos. Padrão JavaBean, Sobrecarga e Sobrescrita, Herança e Associação. Desenvolvimento em camadas

Aula 01

Sergio Mendes
Email: sergio.coti@gmail.com
Telefone: 21 96957-5900

Alunos:

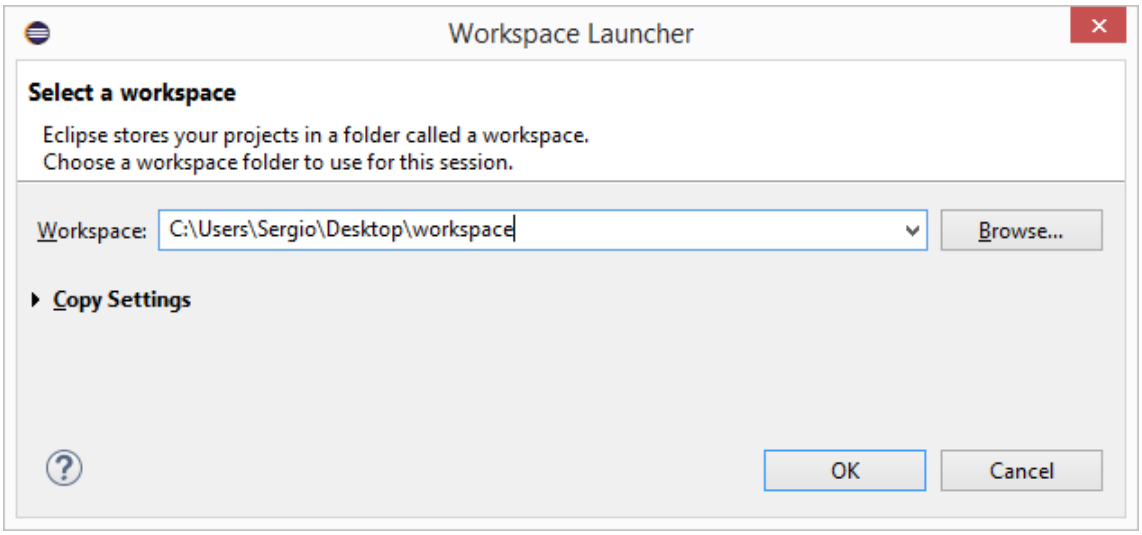
Anderson Badari	anderson.badari@gmail.com
Artur Navarro	arturnxz@gmail.com
Bruna Amancio	bruna_ams@outlook.pt
Cosmo Santos	cosmosantos13@gmail.com
Diego Fernandes	diegofernandes88@hotmail.com
Felipe Osorio	felipe.g.osorio@gmail.com
Gabriel Pereira	gabriel0ps@hotmail.com
Rafael Hiroshi	hiroshi_nuts@hotmail.com
Ingrid Stofalete	istofalete@brq.com
Jessica Rodrigues	jesanrodrigues@outlook.com
Janaina Piovani	jpiovani@outlook.com
Lernardo Ferri	leonardoferri.lnf@gmail.com
Lucas Almeida	lucascoldx@gmail.com
Marcos Takeo Hirata	marcostakeohirata@gmail.com
Rodrigo Moura	roddrigomoura@gmail.com
Rodrigo Brito	rodrigobrito.profissional@gmail.com
Vitor Soares	volive56@gmail.com
Yuri Lenzi	y.bartochevis@gmail.com
Yanka Esperança	yankaleal0@gmail.com

COTI Informatica

www.cotiinformatica.com.br

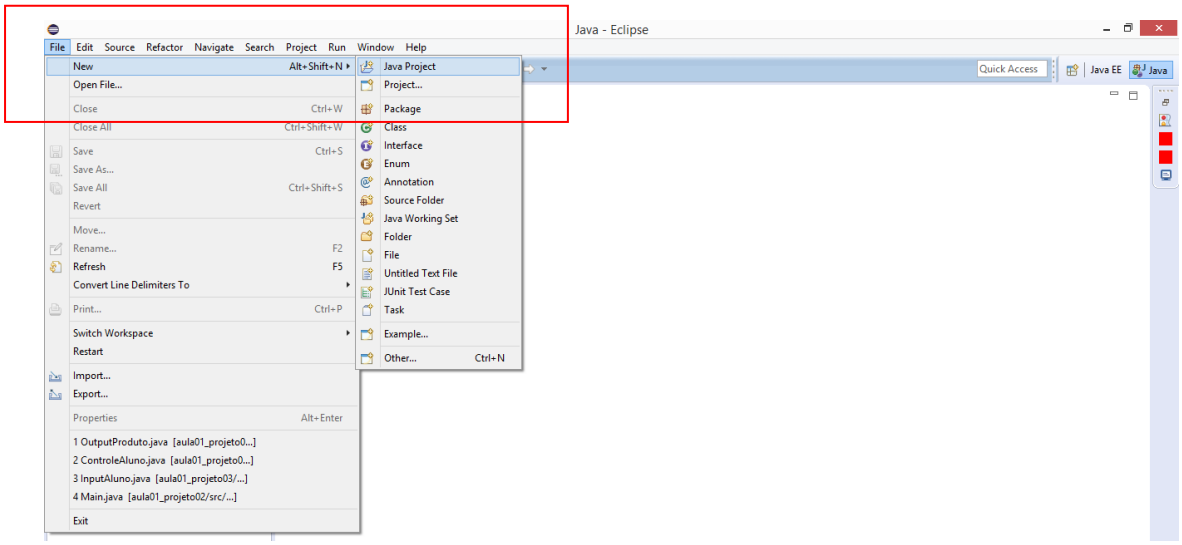
Iniciando a IDE Eclipse:

Selecione o workspace (Pasta de trabalho)

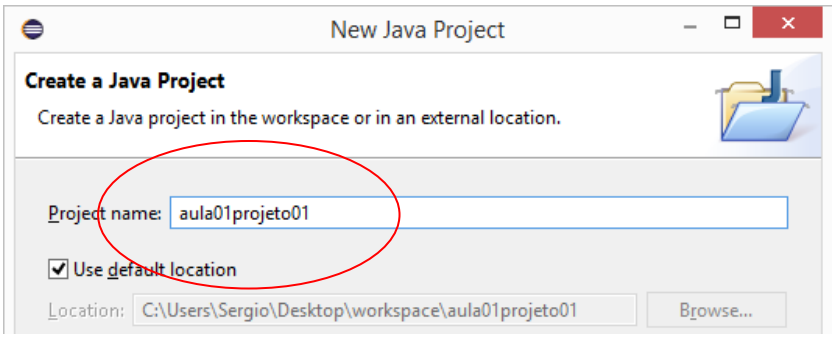


Criando o primeiro projeto Java:

File > New > Java Project



Nome do projeto:



JavaBeans

JavaBeans são componentes de *software* escritos na linguagem de programação Java. Segundo a especificação da Sun Microsystems os JavaBeans são "componentes reutilizáveis de *software* que podem ser manipulados visualmente com a ajuda de uma ferramenta de desenvolvimento".

Um bean também pode ser definido como uma classe Java que expõe propriedades, seguindo uma convenção de nomeclatura simples para os métodos getter e setter.

Praticamente são classes escritas de acordo com uma convenção em particular. São usados para encapsular muitos objetos em um único objeto (o bean), assim eles podem ser transmitidos como um único objeto em vez de vários objetos individuais.

Fonte: https://pt.wikipedia.org/wiki/JavaBeans

São características de uma Classe JavaBean:

- Atributos privados
- Construtores
  - Vazio (sem argumentos)
  - Com entrada de argumetos (Sobrecarga)
- Métodos set e get (encapsulamento)
- Sobrescrita dos métodos da Classe Object
  - toString
  - equals
  - hashCode

Criando o JavaBean: **Cliente**

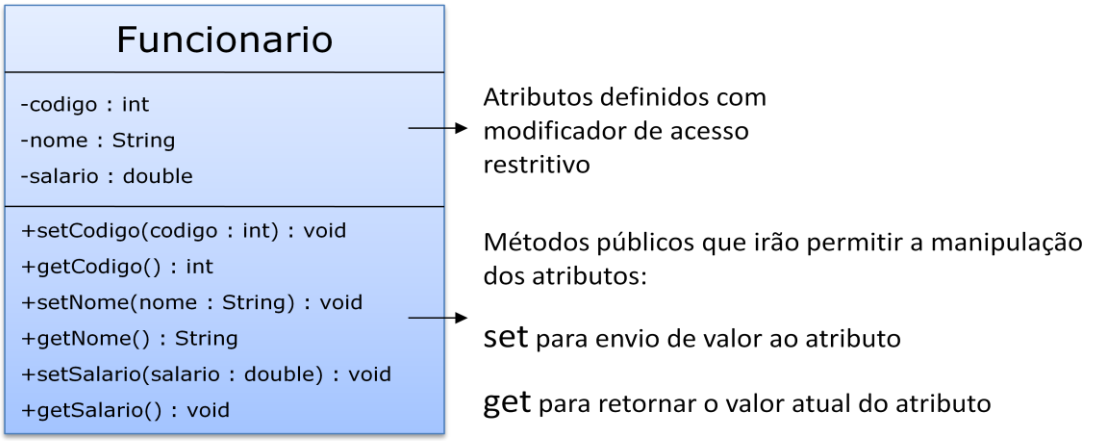
```
package entities;

//JavaBean (POJO - Plain Old Java Object)
//Caracteristicas:
      //- Atributos privados..
      //- Encapsulamento (set/get)
      //- Construtores
      //- Herança de Object
      //    - Sobrescrever os metodos equals, hashCode, toString
public class Cliente {

      //Atributos..
      //visibilidades (modificadores de acesso) -> atributos ou
        metodos
      //private   -> acesso somente dentro da propria classe
```

```
//public    -> acesso total
//protected -> acesso dentro do mesmo pacote ou por herança
//default (friendly) -> acesso dentro do mesmo pacote
//tipos primitivos: byte, boolean, char, int, float, double
//Wrappers: Byte, Boolean, Character, Integer, Float, Double
private Integer idCliente;
private String nome;
private String email;

//construtores..
public Cliente() {
	// vazio (default..)
}

//construtor..
//sobrecarga de métodos (overloading)
//criar metodos com o mesmo nome porem com entradas de
  argumentos diferentes
public Cliente(Integer idCliente, String nome, String email){
	this.idCliente = idCliente;
	this.nome = nome;
	this.email = email;
}

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

public Integer getIdCliente(){
	return idCliente;
}

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

public String getNome(){
	return nome;
}

public void setEmail(String email){
	this.email = email;
}

public String getEmail(){
	return email;
}
//Metodo para retorna a classe como texto (string)
@Override //annotation
```

```
	public String toString() {
		return idCliente + ", " + nome + ", " + email;
	}

	//Metodo booleano que indica se dois 2 objetos
	  de Cliente sao iguais
	@Override
	public boolean equals(Object obj) {
		//verificar se o objeto recebido pelo equals é um cliente
		if(obj instanceof Cliente){
			//converter obj em Cliente (casting)
			Cliente c = (Cliente) obj;
			//regra de comparação..
			return c.getIdCliente().equals(idCliente);
		}
		return false;
	}

	@Override
	public int hashCode() {
		//regra de agrupamento de objetos (organização)..
		return idCliente.hashCode();
	}
}
```

Encapsulamento

Em orientação a objetos, o Encapsulamento é o mecanismo a partir do qual os atributos de uma Classe são protegidos do acesso externo.
Esta proteção baseia-se no uso de modificadores de acesso restritivos para os atributos e na criação de métodos que irão realizar o acesso indireto a esses atributos.



Modificadores de Visibilidade:

private
Acesso somente dentro da própria Classe
(tipo mais restritivo de visibilidade)

default
Acesso dentro da própria Classe ou por Classes
que estão no mesmo pacote.

protected
Acesso por Classes do mesmo pacote ou em pacotes
diferentes por meio de herança

public
Acesso total.

Sobrecarga de Métodos (Overloading)

A sobrecarga de métodos ocorre quando em uma classe, declaramos métodos com o mesmo nome, porem com entrada de argumentos diferentes. Exemplo: Construtores da classe Cliente.

```
//construtores..
public Cliente() {
	// vazio (default..)
}

//construtor..
//sobrecarga de métodos (overloading)
//criar metodos com o mesmo nome porem com entradas de argumentos
diferentes
public Cliente(Integer idCliente, String nome, String email){
	this.idCliente = idCliente;
	this.nome = nome;
	this.email = email;
}
```

Sobrescrita de métodos (Override)

A sobrescrita de métodos ocorre quando uma subclasse sobrepõe métodos da sua superclasse, modificando o comportamento de tais métodos, reprogramando-os na subclasse.

Exemplo: Sobrescrita dos métodos equals, hashCode e toString da classe Object:

```
//Metodo para retorna a classe como texto (string)
@Override //annotation
public String toString() {
	return idCliente + ", " + nome + ", " + email;
}

//Metodo booleano que indica se dois 2 objetos de Cliente sao iguais
@Override
public boolean equals(Object obj) {

	//verificar se o objeto recebido pelo equals é um cliente
	if(obj instanceof Cliente){

		//converter obj em Cliente (casting)
		Cliente c = (Cliente) obj;

		//regra de comparação..
		return c.getIdCliente().equals(idCliente);
	}

	return false;
}

@Override
public int hashCode() {
	//regra de agrupamento de objetos (organização)..
	return idCliente.hashCode();
}
```

Objeto:

Uma instância de uma Classe
Armazenamento de estados através de seus atributos e reação a mensagens enviadas por outros objetos.

```
Cliente c1 = new Cliente();
```

[Classe] [Objeto] [Construtor → Instância]

Executando a classe Cliente no método Main():

```
package principal;

import entities.Cliente;
import entities.PessoaFisica;
import entities.PessoaJuridica;
```

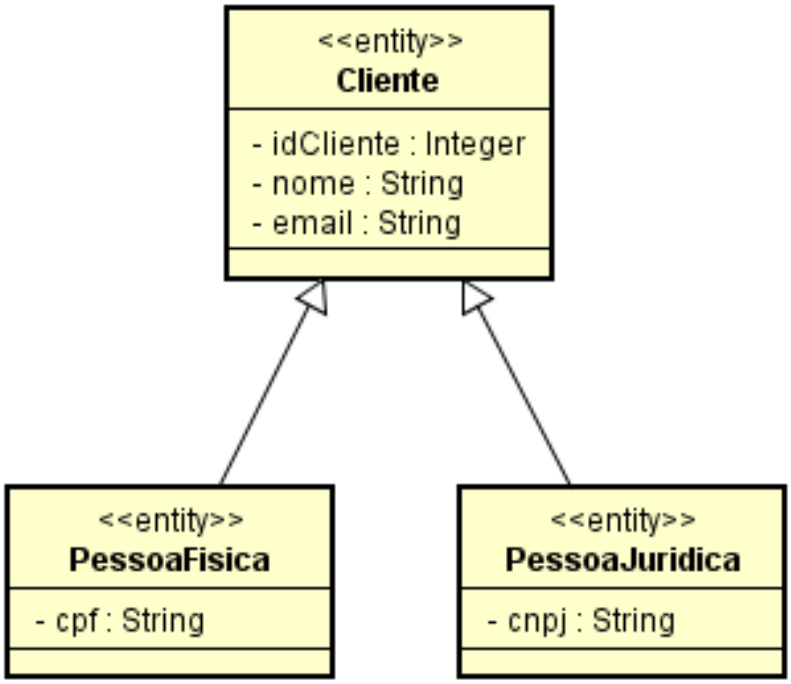
```
public class Main {

	// ctrl + shift + O -> imports
	// ctrl + shift + F -> tabulação

	public static void main(String[] args) {

		//Classe -> Objeto
		Cliente c1 = new Cliente(1, "Sergio Mendes", "sergio.coti@gmail.com");
		Cliente c2 = new Cliente(1, "Yuri", "yuri@gmail.com");

		//c.setIdCliente(1); //entrada
		//c.setNome("Sergio Mendes"); //entrada
		//c.setEmail("sergio.coti@gmail.com"); //entrada

		//System.out.println("IdCliente..: " + c.getIdCliente());
		//System.out.println("Nome.....: " + c.getNome());
		//System.out.println("Email.....: " + c.getEmail());

		if(c1.equals(c2)){
			System.out.println("Clientes iguais");
		}
		else{
			System.out.println("Clientes diferentes");
		}

		System.out.println("Cliente: " + c1); //toString()
		System.out.println("Cliente: " + c2); //tostring()
	}
}
```

Herança

A Herança está diretamente relacionada ao reuso de código. É praticamente impossível, em termos de modelagem, projetar uma solução orientada a objetos sem uso de herança.

Sendo assim, Classes mais genéricas e menos especializadas possuem características que podem ser herdadas por classes menos genéricas, porém mais especializadas.

Exemplo:




```
package entities;

//JavaBean
public class PessoaFisica extends Cliente{

	//Atributos...
	private String cpf;

	//construtores..
	//Primeiro: Construtor default (vazio)
	public PessoaFisica() {
	}

	//construtor com entrada de parametros..
	public PessoaFisica(Integer idCliente, String nome,
			String email, String cpf) {
		super(idCliente, nome, email);
		//executando o construtor da superclasse..
		this.cpf = cpf;
	}

	//encapsulamento..
	public String getCpf() {
		return cpf;
	}
```

```
	public void setCpf(String cpf) {
		this.cpf = cpf;
	}

	//sobrescrever o metodo ToString()
	@Override
	public String toString() {
		return super.toString() + ", " + cpf;
	}
}
```

```
package entities;

public class PessoaJuridica extends Cliente{

	private String cnpj;

	public PessoaJuridica() {

	}

	public PessoaJuridica(Integer idCliente, String nome,
				String email, String cnpj) {
		super(idCliente, nome, email);
		this.cnpj = cnpj;
	}

	public String getCnpj() {
		return cnpj;
	}

	public void setCnpj(String cnpj) {
		this.cnpj = cnpj;
	}

	@Override
	public String toString() {
		return super.toString() + ", " + cnpj;
	}
}
```

Executando no método Main():

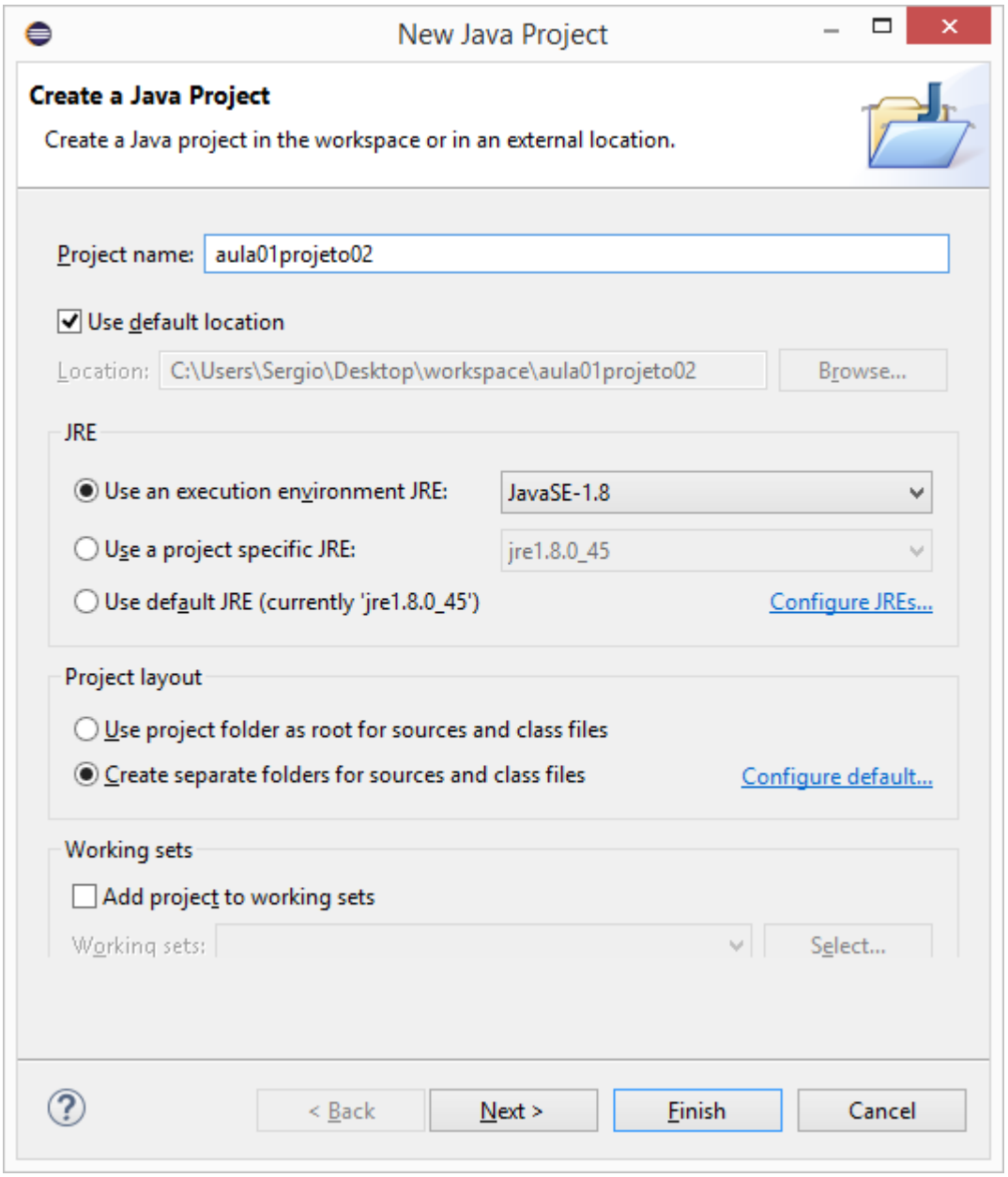
```
package principal;

import entities.Cliente;
import entities.PessoaFisica;
import entities.PessoaJuridica;
```

```
public class Main {

	// ctrl + shift + O -> imports
	// ctrl + shift + F -> tabulação

	public static void main(String[] args) {

		//Classe -> Objeto
		Cliente c1 = new Cliente(1, "Sergio Mendes", "sergio.coti@gmail.com");
		Cliente c2 = new Cliente(1, "Yuri", "yuri@gmail.com");

		//c.setIdCliente(1); //entrada
		//c.setNome("Sergio Mendes"); //entrada
		//c.setEmail("sergio.coti@gmail.com"); //entrada

		//System.out.println("IdCliente..: " + c.getIdCliente());
		//System.out.println("Nome.....: " + c.getNome());
		//System.out.println("Email.....: " + c.getEmail());

		if(c1.equals(c2)){
			System.out.println("Clientes iguais");
		}
		else{
			System.out.println("Clientes diferentes");
		}

		System.out.println("Cliente: " + c1); //toString()
		System.out.println("Cliente: " + c2); //tostring()

		PessoaFisica pf = new PessoaFisica
			(3, "Lernardo", "leo@gmail.com", "1234567890");


		PessoaJuridica pj = new PessoaJuridica
			(4, "Loja", "loja@gmailcom", "0987654321");


		System.out.println(pf); //toString()

		System.out.println(pj); //toString()

	}
}
```

Saida do programa:

```
Clientes iguais
Cliente: 1, Sergio Mendes, sergio.coti@gmail.com
Cliente: 1, Yuri, yuri@gmail.com
3, Lernardo, leo@gmail.com, 1234567890
4, Loja, loja@gmailcom, 0987654321
```

Novo projeto:



Associação (TER)

Utilizado para relacionamentos de objetos de classes distintas. A navegabilidade é representada através de uma seta nas extremidades, pois representa o sentido em que as informações são disparadas.

Exemplo:

Criando a Classe Endereço no padrão JavaBean:

```
package entities;

public class Endereco {

	// Atributos
	private Integer idEndereco;
	private String logradouro;
	private String cidade;
	private String estado;

	// Construtor default..
	public Endereco() {
		// TODO Auto-generated constructor stub
	}

	// sobrecarga de construtores..
	public Endereco(Integer idEndereco, String logradouro,
			String cidade, String estado) {
		this.idEndereco = idEndereco;
		this.logradouro = logradouro;
		this.cidade = cidade;
		this.estado = estado;
	}

	public Integer getIdEndereco() {
		return idEndereco;
	}

	public void setIdEndereco(Integer idEndereco) {
		this.idEndereco = idEndereco;
	}
```

```
	public String getLogradouro() {
		return logradouro;
	}

	public void setLogradouro(String logradouro) {
		this.logradouro = logradouro;
	}

	public String getCidade() {
		return cidade;
	}

	public void setCidade(String cidade) {
		this.cidade = cidade;
	}

	public String getEstado() {
		return estado;
	}

	public void setEstado(String estado) {
		this.estado = estado;
	}

	@Override
	public String toString() {
		return "Endereco [idEndereco=" + idEndereco + ",
			logradouro=" + logradouro + ", cidade=" + cidade
			+ ", estado=" + estado + "]";
	}
}
```

Criando a Classe Funcionario e relacionando com Endereco:

```
package entities;

//JavaBean (POJO)
public class Funcionario{

	// atributos..
	private Integer idFuncionario;
	private String nome;
	private Double salario;
	private Endereco endereco; //Associação (TER-1)

	// construtor default..
	public Funcionario() {
		// vazio..
	}
	// sobrecarga de construtores
```

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

//sobrecarga de construtores..
public Funcionario(Integer idFuncionario, String nome,
                   Double salario, Endereco endereco){
      this(idFuncionario, nome, salario);
      this.endereco = endereco;
}

// encapsulamento..
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;
}

public Endereco getEndereco() {
      return endereco;
}

public void setEndereco(Endereco endereco) {
      this.endereco = endereco;
}



@Override
```

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

Executando no método Main...

Podemos afirmar que **Funcionario** "possui" **Endereco**, ou seja, a Classe Endereço faz parte da Classe Funcionario.
Note que podemos definir a **multiplicidade** deste tipo de relacionamento, são eles:

0..1	No mínimo zero e no máximo 1
1	1 e somente 1
0..*	No mínimo zero e no máximo muitos
1..*	No mínimo 1 e no máximo muitos
*	Muitos

Executando no método Main():

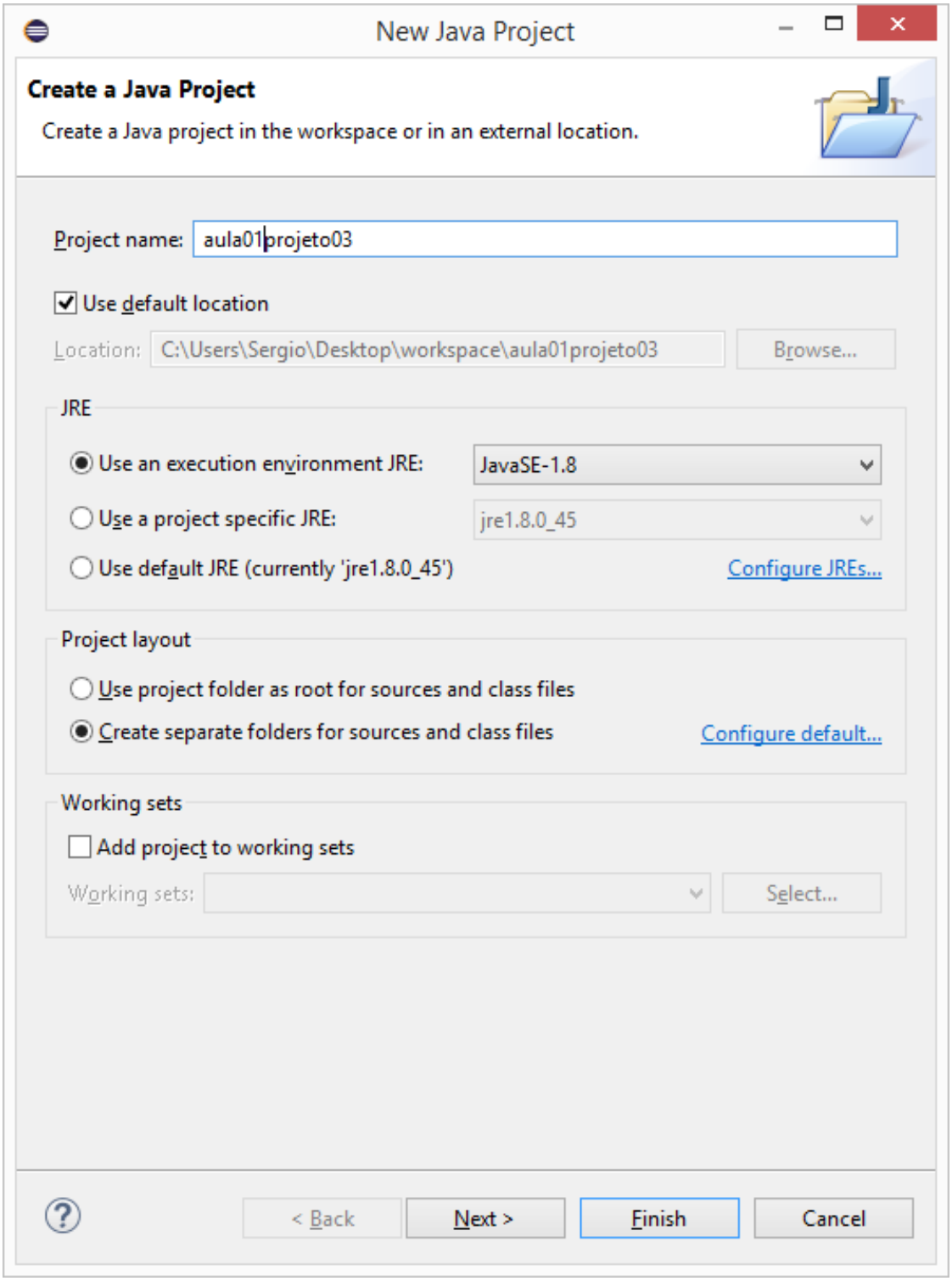
```
package principal;

import entities.Endereco;
import entities.Funcionario;

public class Main {

	public static void main(String[] args) {

		Funcionario f1 = new Funcionario(); //instanciando..
		f1.setEndereco(new Endereco()); //instanciando..

		f1.setIdFuncionario(1);
		f1.setNome("Artur");
		f1.setSalario(3000.0);
		f1.getEndereco().setIdEndereco(1);
		f1.getEndereco().setLogradouro("Boa Vista, 254");
		f1.getEndereco().setCidade("São Paulo");
		f1.getEndereco().setEstado("SP");
```

```
                //imprimindo..
                System.out.println(f1); //toString()
                System.out.println(f1.getEndereco()); //toString()

                System.out.println("\n");

                //outra forma..
                Funcionario f2 = new Funcionario(2, "Leonardo", 2500.0);
                f2.setEndereco(new Endereco(2, "Boa Vista, 123", "São Paulo", "SP"));

                //imprimindo..
                System.out.println(f2); //toString()
                System.out.println(f2.getEndereco()); //toString()

                System.out.println("\n");

                //outra forma...
                Funcionario f3 = new Funcionario(3, "Bruna", 3500.0,
                                new Endereco(3, "Boa Vista, 456", "São Paulo", "SP"));

                System.out.println(f3);
                System.out.println(f3.getEndereco());
        }
}
```

Saida do programa:

```
Funcionario [idFuncionario=1, nome=Artur, salario=3000.0]
Endereco [idEndereco=1, logradouro=Boa Vista, 254, cidade=São
Paulo, estado=SP]


Funcionario [idFuncionario=2, nome=Leonardo, salario=2500.0]
Endereco [idEndereco=2, logradouro=Boa Vista, 123, cidade=São
Paulo, estado=SP]


Funcionario [idFuncionario=3, nome=Bruna, salario=3500.0]
Endereco [idEndereco=3, logradouro=Boa Vista, 456, cidade=São
Paulo, estado=SP]
```

Novo projeto:



Criando uma entidade Aluno:

```
package entities;

import java.util.Arrays;

public class Aluno {

	private Integer idAluno;
	private String nome;
	private Double notas[]; // array

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

	public Aluno(Integer idAluno, String nome, Double[] notas) {
		this.idAluno = idAluno;
		this.nome = nome;
		this.notas = notas;
	}

	public Integer getIdAluno() {
		return idAluno;
	}

	public void setIdAluno(Integer idAluno) {
		this.idAluno = idAluno;
	}

	public String getNome() {
		return nome;
	}

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

	public Double[] getNotas() {
		return notas;
	}
```

```
	public void setNotas(Double[] notas) {
		this.notas = notas;
	}

	@Override
	public String toString() {
		return "Aluno [idAluno=" + idAluno + ", nome=" + nome + ",
			notas=" + Arrays.toString(notas) + "]";
	}
}
```

Criando uma classe para ler os dados do aluno utilizando a api **Java.util.Scanner**

```
package input;

import java.util.Scanner;

//entrada de dados pelo console do Java..
public class InputAluno {

	//atributo..
	private Scanner s; //null

	public InputAluno() {
		s = new Scanner(System.in); //instanciando..
	}

	// método para ler e retornar o id de um aluno..
	public Integer lerIdAluno() {
		System.out.print("Informe o Id do Aluno.....: ");
		return Integer.parseInt(s.nextLine());
		// ler o valor informado como inteiro
	}


	// método para ler e retornar o nome de um aluno..
	public String lerNome() {
		System.out.print("Informe o Nome do Aluno...: ");
		return s.nextLine(); // ler o valor informado como string
	}
```

```
	// método para ler e retornar a nota de um aluno..
	public Double lerNota() {
		System.out.print("Informe o Nota do Aluno...: ");
		return Double.parseDouble(s.nextLine());
		// ler o valor informado como double
	}
}
```

Criando uma classe para calcular operações com a entidade Aluno:

```
package control;

import entities.Aluno;

public class ControleAluno {

	//método para calcular e retornar a media do aluno..
	//VO -> Value Object (passagem de objeto por parametro..)
	public Double obterMedia(Aluno a){

		double somatorioNotas = 0.0; //variavel local..

		//somando as notas do aluno.. (foreach)
		for(Double nota : a.getNotas()){
			somatorioNotas += nota; //acumulador..
		}

		//retornando o calculo da media..
		return somatorioNotas / a.getNotas().length;
	}

	//Método para calcular a situação do aluno..
	public String obterSituacao(Aluno a){

		//executando o metodo obterMedia()
		double media = obterMedia(a);

		if(media >= 7){
			return "Aprovado";
		}
		else if(media >= 5){
			return "Recuperação";
		}
		else{
			return "Reprovado";
		}
	}
}
```

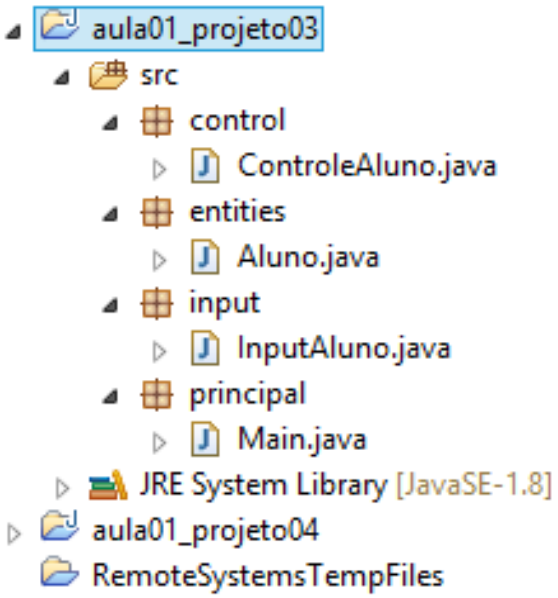
Executando no método Main():

```
package principal;

import control.ControleAluno;
import entities.Aluno;
import input.InputAluno;

public class Main {
	public static void main(String[] args) {
		Aluno a = new Aluno(); //entidade..
		InputAluno input = new InputAluno(); //entrada de dados..

		a.setIdAluno(input.lerIdAluno());
		a.setNome(input.lerNome());

		a.setNotas(new Double[4]); //vetor de 4 posições..

		for(int i = 0; i < a.getNotas().length; i++){
			//ler cada nota do aluno..
			a.getNotas()[i] = input.lerNota();
		}

		System.out.println("\nDados do Aluno:");
		System.out.println(a); //toString()

		ControleAluno control = new ControleAluno();
		System.out.println("Media do Aluno...: " + control.obterMedia(a));
		System.out.println("Situacao.....: " + control.obterSituacao(a));
	}
}
```

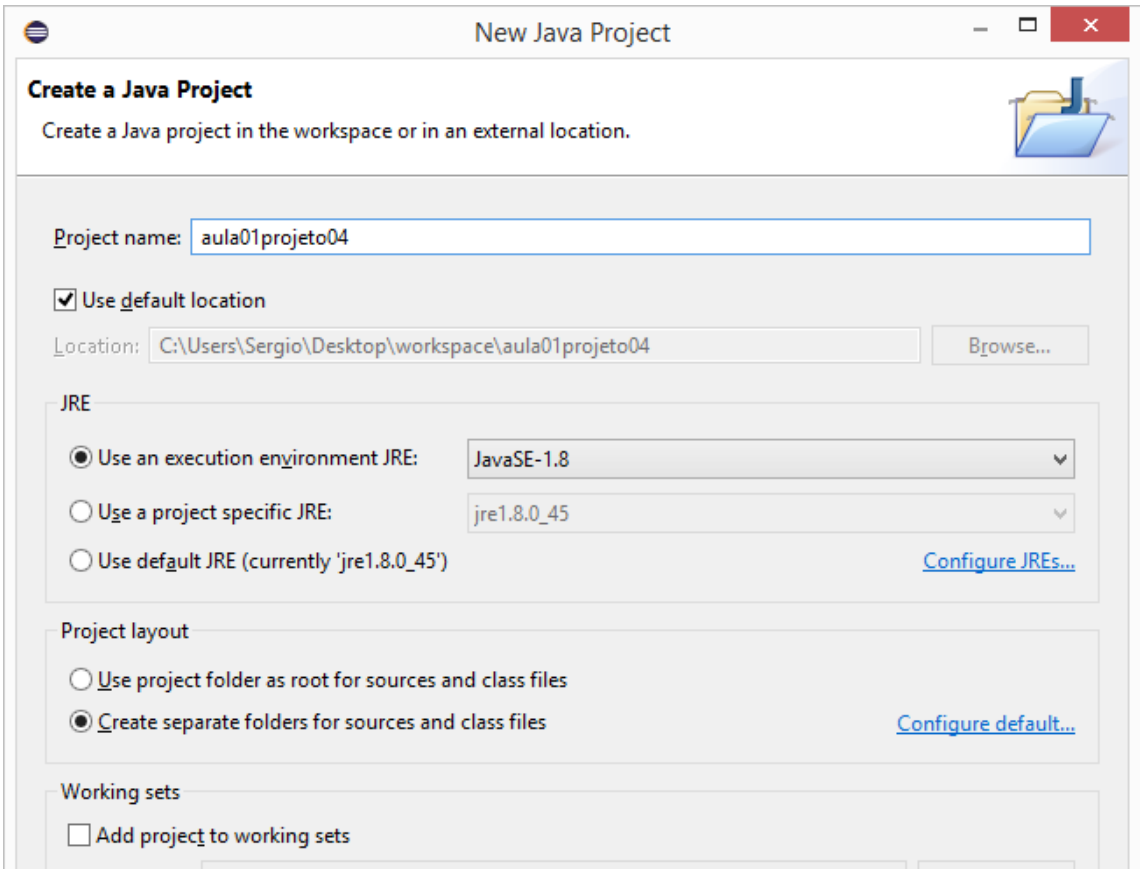
Saida do programa:

```
Informe o Id do Aluno.....: 1
Informe o Nome do Aluno...: Sergio Mendes
Informe o Nota do Aluno...: 8
Informe o Nota do Aluno...: 7
Informe o Nota do Aluno...: 6
Informe o Nota do Aluno...: 10

Dados do Aluno:
Aluno [idAluno=1, nome=Sergio Mendes, notas=[8.0, 7.0, 6.0, 10.0]]
Media do Aluno...: 7.75
Situacao.....: Aprovado
```

Organização do projeto:



Novo projeto:



Entidade: Produto

```
package entities;

//JavaBean..
public class Produto {

	//Atributos..
	private Integer idProduto;
	private String nome;
	private Double preco;
	private Integer quantidade;

	//Construtores..
	public Produto() {
		// default (vazio..)
	}

	//Sobrecarga de metodos (Overloading)
	public Produto(Integer idProduto, String nome,
			Double preco, Integer quantidade) {
		this.idProduto = idProduto;
		this.nome = nome;
		this.preco = preco;
		this.quantidade = quantidade;
	}

	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;
	}
```

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

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

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

Classe para Calculo de operações com Produto:
ControleProduto.java

```
package control;

import entities.Produto;

//serviços com a entidade Produto..
public class ControleProduto {

	//VO (Value Object..)
	public Double obterTotal(Produto p){
		return p.getPreco() * p.getQuantidade();
	}

	//VO (Value Object..)
	public Double obterTotalComDesconto(Produto p){

		//calculando o total..
		double total = obterTotal(p);

		if(p.getQuantidade() > 10){
			return total - (total * 0.10); //desconto de 10%
		}
		else{
			return total - (total * 0.04); //desconto de 4%
		}
	}
}
```

Classe para impressão dos dados dos produtos:
Método utilizando Varargs (passagem de parâmetros variante, similar a um vetor de objetos)

```
package output;

import control.ControleProduto;
import entities.Produto;

public class OutputProduto {

	//método para receber produtos e imprimir no prompt..
	//varargs -> parametro do tipo vetor...
	public void imprimirDados(Produto... vetor){

		System.out.println("*** Relatorio de Produtos ***");
		System.out.println("----------------------------");

		ControleProduto c = new ControleProduto(); //serviços..

		//pecorrer o vetor de produtos..
		//for each
		for(Produto p : vetor){
			System.out.println("Id do Produto.....: " + p.getIdProduto());
			System.out.println("Nome do Produto.....: " + p.getNome());
			System.out.println("Preco.....: " + p.getPreco());
			System.out.println("Quantidade.....: " + p.getQuantidade());
			System.out.println("Total.....: " + c.obterTotal(p));
			System.out.println("Total com Desconto..: "
					+ c.obterTotalComDesconto(p));
			System.out.println("---");
		}
	}
}
```

Testando e executando:

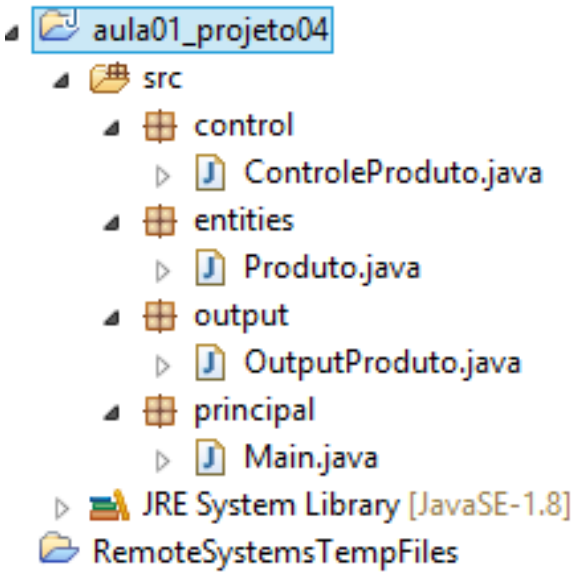
```
package principal;

import entities.Produto;
import output.OutputProduto;

public class Main {

	public static void main(String[] args) {

		//vetor de produtos..
		Produto[] vetor = new Produto[4];
```

```
            vetor[0] = new Produto(1, "Mouse", 30.0, 10);
            vetor[1] = new Produto(2, "Celular", 250.0, 5);
            vetor[2] = new Produto(3, "PenDrive", 50.0, 15);
            vetor[3] = new Produto(4, "Caderno", 20.0, 10);

            OutputProduto out = new OutputProduto();
            out.imprimirDados(vetor);
        }
}
```

Saida do programa:

```
*** Relatorio de Produtos ***
-----------------------------
Id do Produto.......: 1
Nome do Produto.....: Mouse
Preco...............: 30.0
Quantidade..........: 10
Total...............: 300.0
Total com Desconto..: 288.0
---
Id do Produto.......: 2
Nome do Produto.....: Celular
Preco...............: 250.0
Quantidade..........: 5
Total...............: 1250.0
Total com Desconto..: 1200.0
---
Id do Produto.......: 3
Nome do Produto.....: PenDrive
Preco...............: 50.0
Quantidade..........: 15
Total...............: 750.0
Total com Desconto..: 675.0
---
Id do Produto.......: 4
Nome do Produto.....: Caderno
Preco...............: 20.0
Quantidade..........: 10
Total...............: 200.0
Total com Desconto..: 192.0
---
```

Organização das classes:



A Orientação a Objetos é uma maneira alternativa de pensar os problemas de sistemas de informação utilizando modelos organizados a partir de conceitos do mundo real.

O artefato base é o "objeto" capaz de combinar estrutura e comportamento em uma única "entidade".

Tudo o que podemos ver no mundo real é considerado um objeto com atributos e comportamentos definidos.

Na qualidade de método de modelagem, é tida como a melhor estratégia para se eliminar a dificuldade recorrente no processo de modelar o mundo real do domínio do problema em um conjunto de componentes de software que seja o mais fiel na sua representação deste domínio.