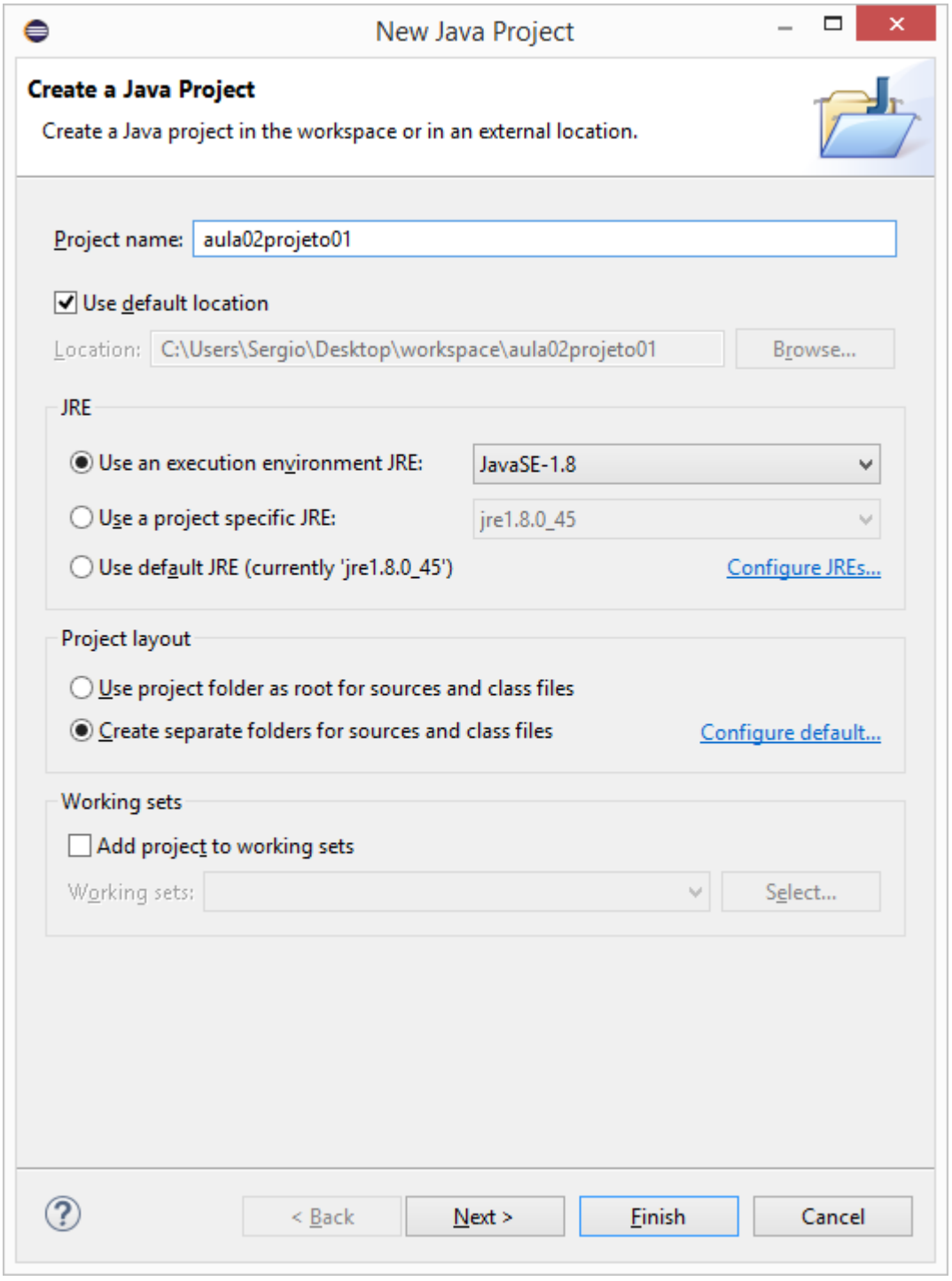
Criando um novo projeto:

- File > New > Java Project



Criando um JavaBean Funcionario:

```
package entities;

//JavaBean
public class Funcionario {

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

	// Construtores
	// 1) Construtor default (vazio) -> sem entrada de argumentos
	public Funcionario() {
		// vazio...
	}

	// 2) Sobrecarga de construtores (overloading)
	public Funcionario(Integer idFuncionario, String nome,
			Double salario) {
		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;
	}
```

```
	// Sobrescrever o método toString() da classe Object..
	@Override
	public String toString() {
		return "Funcionario [idFuncionario=" + idFuncionario
			+ ", nome=" + nome + ", salario=" + salario + "]";
	}
}
```

Interfaces:

A interface é o tipo de programação mais "puro" do Java, pois não programamos o conteúdo dos métodos de uma interface, apenas sua declaração (assinatura). Toda interface Java obedece às seguintes regras:

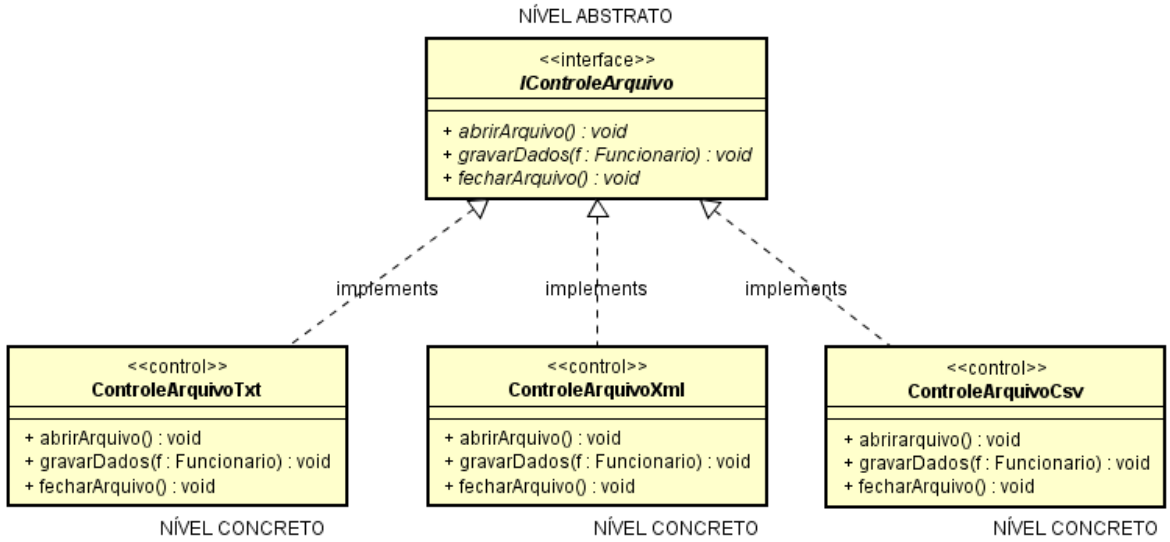
- Todos os métodos de uma interface são implicitamente públicos e abstratos
- Todos os métodos de uma interface não possuem corpo, apenas assinatura
- Os Atributos de uma interface são, por definição, constantes, ou seja, possuem valor final
- Quando uma Classe implementa uma interface, a Classe deverá forecer corpo para todos os métodos da interface, exceto se a Classe for abstrata
- Uma interface pode herdar de outras interfaces
- Uma classe pode implementar várias interfaces.

Polimorfismo

Todo objeto que possa passar em mais de um teste É-UM pode ser considerado polimórfico, ou seja, referências a Classes mais genéricas terão seu comportamento definidos através de instâncias de Classes mais específicas.

Polimorfismo significa "muitas formas". Em Orientação a Objetos, o conceito do polimorfismo é aplicado quando utilizamos o vertbo SER entre pelo menos 2 ou mais subclasses, podendo ser feito utilizando-se interfaces ou Classes abstratas.

Exemplo:



Criando a interface:

```
package contracts;

import entities.Funcionario;

public interface IControleArquivo {

	//Regras sobre interfaces..
	/*
	 * Interfaces só podem ter atributos se estes forem constantes
	 * Interfaces não possuem construtores
	 * Métodos de interface não posuem corpo, apenas assinatura
	 * Todo método de interface ja é implicitamente public abstract
	 * Em interfaces são podemos ter metodos private, default ou protected
	 */

	void abrirArquivo(); //método abstrato

	void gravarDados(Funcionario f); //metodo abstrato

	void fecharArquivo(); //metodo abstrato

}
```

Implementando a interface para arquivos do tipo TXT:

```
package controls;

import java.io.File;
import java.io.FileWriter;
```

```
import java.io.IOException;
import contracts.IControleArquivo;
import entities.Funcionario;

//Regra: quando uma classe implementa uma interface, a classe é obrigada
//a programar (fornecer implementação) para todos os métodos da interface
public class ControleArquivoTxt implements IControleArquivo{

	//atributo..
	private FileWriter arquivo; //null

	@Override
	public void abrirArquivo() {
		try {
			arquivo = new FileWriter(new File("c:\\temp\\funcionario.txt"),
								true);
		} catch (IOException e) {
			e.printStackTrace();
		}
	}

	@Override
	public void gravarDados(Funcionario f){
		try {
			arquivo.write("\nDados do Funcionario:");
			arquivo.write("\nId do Funcionario.....: " + f.getIdFuncionario());
			arquivo.write("\nNome.....: " + f.getNome());
			arquivo.write("\nSalario.....: " + f.getSalario());
			arquivo.write("\n");
		} catch (IOException e) {
			e.printStackTrace();
		}
	}

	@Override
	public void fecharArquivo() {
		try {
			arquivo.close();
		} catch (IOException e) {
			e.printStackTrace();
		}
	}
}
```

Implementando a interface para aquivos XML:

```
package controls;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

import contracts.IControleArquivo;
import entities.Funcionario;

public class ControleArquivoXml implements IControleArquivo {

	private FileWriter arquivo;

	@Override
	public void abrirArquivo() {
		try {
			arquivo = new FileWriter(new File("c:\\temp\\funcionario.xml"));
		} catch (IOException e) {
			e.printStackTrace();
		}
	}

	@Override
	public void gravarDados(Funcionario f) {

		try {

			arquivo.write("<?xml version='1.0' encoding='iso-8859-1'?>");

			arquivo.write("<aula>");
			arquivo.write("<funcionario>");
			arquivo.write("<idfuncionario>" + f.getIdFuncionario()
					+ "</idfuncionario>");
			arquivo.write("<nome>" + f.getNome() + "</nome>");
			arquivo.write("<salario>" + f.getSalario() + "</salario>");
			arquivo.write("</funcionario>");
			arquivo.write("</aula>");

		} catch (IOException e) {
			e.printStackTrace();
		}
	}
	@Override
```

```
	public void fecharArquivo() {
		try {
			arquivo.close();
		} catch (IOException e) {
			e.printStackTrace();
		}
	}
}
```

Implementando a interface para arquivos do tipo CSV:

```
package controls;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import contracts.IControleArquivo;
import entities.Funcionario;

public class ControleArquivoCsv implements IControleArquivo{

	private FileWriter arquivo; //null

	@Override
	public void abrirArquivo() {
		try {
			arquivo = new FileWriter(new File("c:\\temp\\funcionario.csv"),
							true);
		} catch (IOException e) {
			e.printStackTrace();
		}
	}

	@Override
	public void gravarDados(Funcionario f){

		try {
			arquivo.write(f.getIdFuncionario() + ";" + f.getNome() + ";"
							+ f.getSalario() + "\n");

		} catch (IOException e) {
			e.printStackTrace();
		}
	}
```

```
	@Override
	public void fecharArquivo() {
		try {
			arquivo.close();
		} catch (IOException e) {
			e.printStackTrace();
		}
	}
}
```

Criando uma Enum para definir os tipos de arquivos manipulados pela aplicação:

De uma maneira simplificada, você pode ver enum como uma 'classe' especial para tratar constantes.

```
package types;

//enum é uma classe de valores constantes
//atributo multivalorado...
public enum TipoArquivo {

	TXT, //constante
	XML, //constante
	CSV  //constante
}
```

Criando uma classe "Factory" para gerar o polimorfismo da interface:

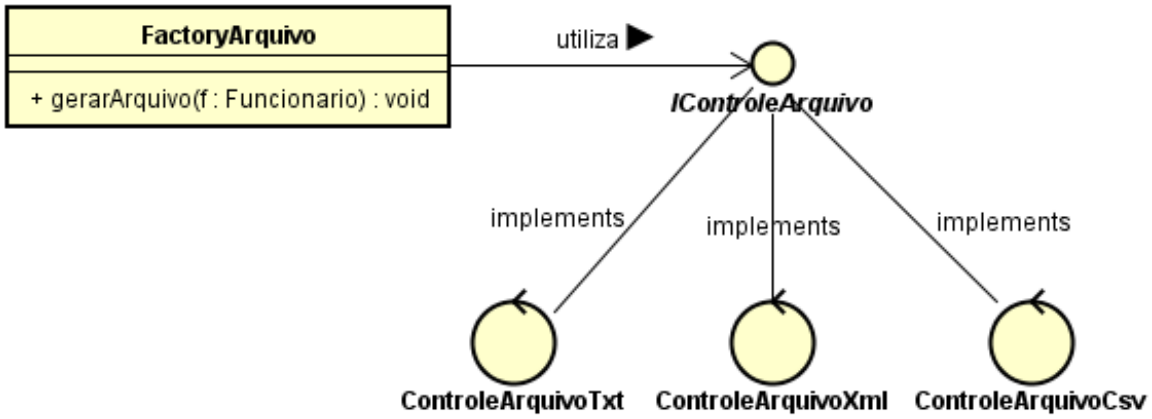
Com o padrão Factory podemos encapsular o código que cria objetos. É muito comum termos interfaces que instanciam classes concretas e essa parte do código normalmente sofre diversas modificações, portanto nesses casos usamos um Factory que encapsula esse comportamento de instanciação.

Usando o Factory temos o nosso código de criação em um objeto ou método, evitando assim a duplicação e além disso temos um local único para fazer manutenção. O padrão também nos dá um código flexível e extensível para o futuro.

Exemplo:

```
package factory;

import contracts.IControleArquivo;
import controls.ControleArquivoCsv;
import controls.ControleArquivoTxt;
import controls.ControleArquivoXml;
import entities.Funcionario;
import types.TipoArquivo;

//fábrica de arquivos...
public class FactoryArquivo {

	//atributo..
	private IControleArquivo controle; //null

	//construtor..
	public FactoryArquivo(TipoArquivo opcao) {

		switch(opcao){
			case TXT:
				controle = new ControleArquivoTxt(); //polimorfismo!
				break;
			case XML:
				controle = new ControleArquivoXml(); //polimorfismo!
				break;
			case CSV:
				controle = new ControleArquivoCsv(); //polimorfismo!
				break;
		}
	}

	//método para executar a gravação de um arquivo..
	public void gerarArquivo(Funcionario f){

		controle.abrirArquivo();
		controle.gravarDados(f);
		controle.fecharArquivo();
	}

}
```

Modelagem:



Criando uma classe para ler os dados do funcionário informado pelo usuário:

Utilizando métodos estáticos:

Quando definimos métodos com a palavra *static* em uma classe ela terá um comportamento especial: **ela será a mesma para todos os objetos daquela classe.**

Ou seja, não haverá um tipo dela em cada objeto. Todos os objetos, ao acessarem e modificarem esse metodo, acessarão o mesmo espaço da memória, e a mudança poderá ser vista em todos os objetos.

Variáveis e métodos estáticos pertencem à classe e não a uma instância. Isso quer dizer que eles podem ser utilizados sem ter que instanciar a classe. Além disso, quando esta classe for carregada, as variáveis estáticas serão compartilhadas por todos os objetos.

Variáveis estáticas não necessitam serem inicializadas, pois recebem o valor default, da mesma forma que variáveis de instância.

Utilizam-se variáveis estáticas para guardar valores que são específicos da classe e não de um objeto, ou seja, todos os objetos terão acesso à mesma cópia da variável estática. Isso quer dizer que quando a classe for chamada pela primeira vez pela JVM, a variável estática será inicializada; se essa variável for incrementada no construtor da classe, toda vez que essa classe for instanciada, essa variável será incrementada e esse valor será compartilhado para todas as instâncias.

Exemplo:

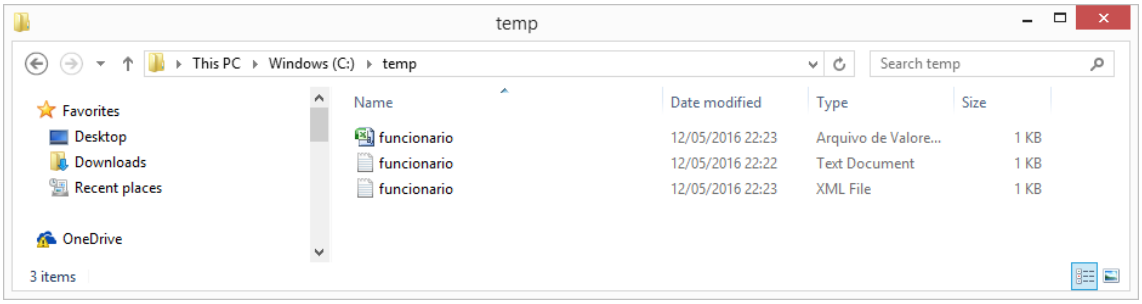
```
package input;

import javax.swing.JOptionPane;

//classe para ler os dados do funcionario..
public class InputFuncionario {

	/*
	 * Metodos estaticos são aqueles que podem
	      ser executados diretamente
	 * pela classe, sem que a classe precise de uma instancia
	 */

	public static Integer lerIdFuncionario(){
		String valor = JOptionPane.showInputDialog
			("Informe o Id do Funcionario:");
		return Integer.parseInt(valor);
	}

	public static String lerNome(){
		return JOptionPane.showInputDialog
			("Informe o Nome do Funcionario:");
	}

	public static Double lerSalario(){
		String valor = JOptionPane.showInputDialog
			("Informe o Salario do Funcionario:");
		return Double.parseDouble(valor);
	}

}
```

Executando na Classe Main()

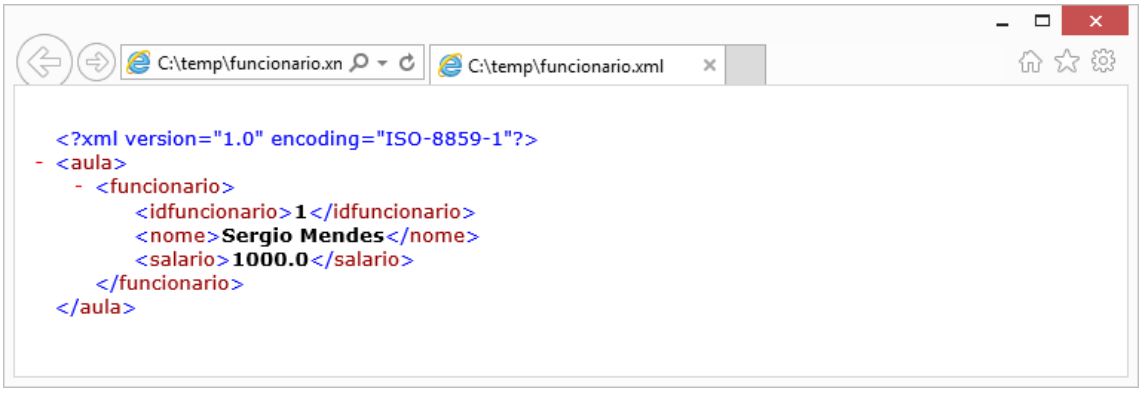
```
package principal;

import contracts.IControleArquivo;
import controls.ControleArquivoXml;
import entities.Funcionario;
import factory.FactoryArquivo;
import input.InputFuncionario;
import types.TipoArquivo;
```

```
public class Main {

	public static void main(String[] args) {

		//ler os dados do funcionario..
		Funcionario f = new Funcionario(); //entidade (javabean)

		f.setIdFuncionario(InputFuncionario.lerIdFuncionario());
		f.setNome(InputFuncionario.lerNome());
		f.setSalario(InputFuncionario.lerSalario());

		//imprimindo..
		System.out.println(f); //toString()

		FactoryArquivo factory = new FactoryArquivo(TipoArquivo.TXT);
		factory.gerarArquivo(f);

		System.out.println("Dados gravados.");
	}
}
```

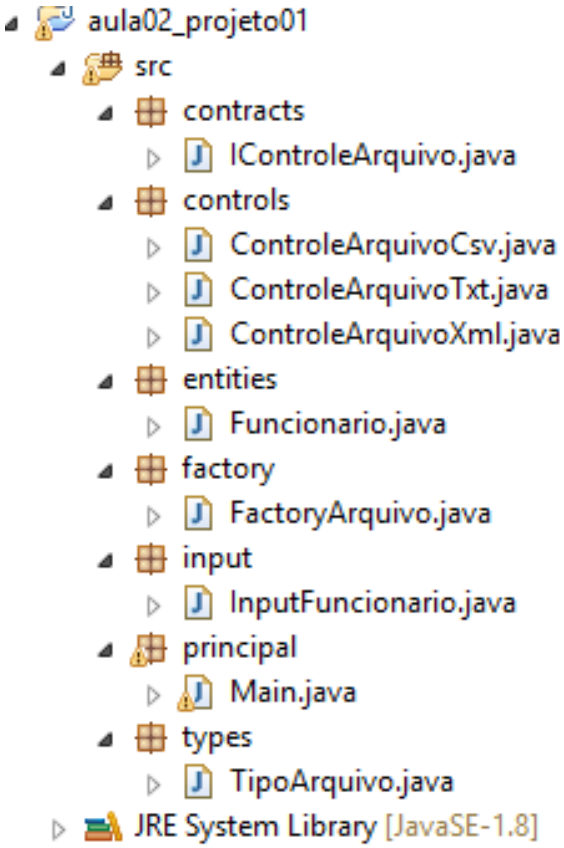
Executando:
Arquivos gerados...



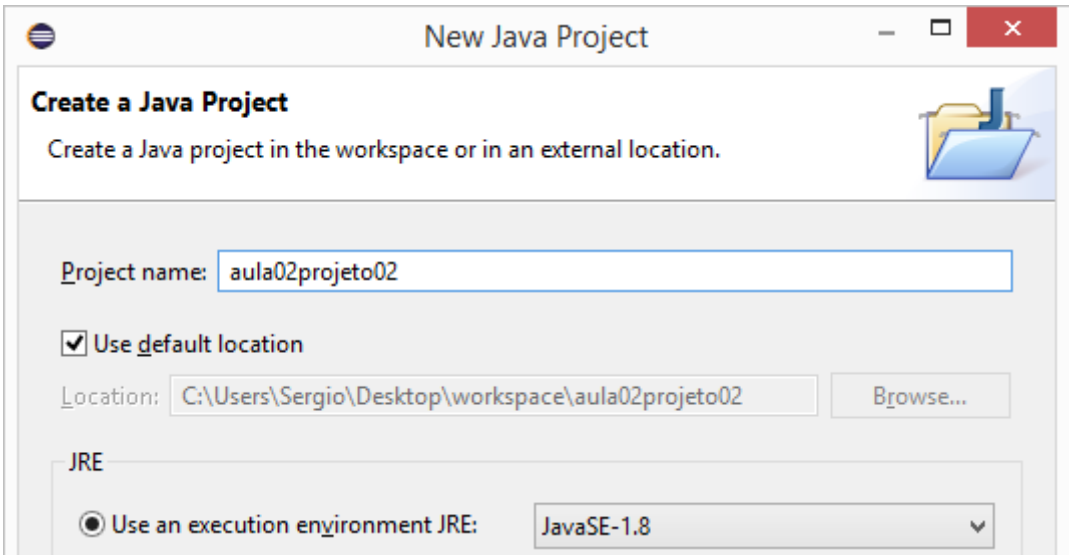
Exemplo de XML gerado:



Estrutura do projeto:



Novo projeto:



Criando uma entidade Produto:

```
package entities;

public class Produto {

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

	public Produto() {
	}

	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 + "]";
	}

}
```

Criando uma classe de validação para os dados do Produto:

```
package validators;

import java.util.regex.Matcher;
import java.util.regex.Pattern;

//classe para validação dos atributos de produto
public class ValidadorProduto {

	//método para validar o id de um produto..
	public boolean validarIdProduto(Integer idProduto){
		return idProduto > 0; //true, false
	}

	//método para validar o nome de um produto..
	public boolean validarNome(String nome){

		//Expressões Regulares (REGEX)
		//regra: nome só devera conter letras, numeros, espaços,
		//de 3 a 30 caracteres..
		Pattern p = Pattern.compile("^[A-Za-zÀ-Üà-ü0-9\\s]{3,30}$");
		Matcher m = p.matcher(nome);

		//retornar o nome passou na regra do regex..
		return m.matches(); //true, false
	}

	//método para validar o preço do produto..
	public boolean validarPreco(Double preco){
		return preco > 0 && preco <= 10000;
	}
}
```

Tratamento de Exceções:

As exceções ocorrem quando algo imprevisto acontece, elas podem ser provenientes de erros de lógica ou acesso a recursos que talvez não estejam disponíveis.

- Tentar abrir um arquivo que não existe.
- Tentar fazer consulta a um banco de dados que não está disponível.
- Tentar escrever algo em um arquivo sobre o qual não se tem permissão de escrita.

- Tentar conectar em servidor inexistente.
- Etc...

Uma maneira de tentar contornar esses imprevistos é realizar o tratamento dos locais no código que podem vir a lançar possíveis exceções.

Exemplo:

```
package input;

import javax.swing.JOptionPane;

import validators.ValidadorProduto;

public class InputProduto {

	//atributo..

	private ValidadorProduto validador;		//null



	//construtor..

	public InputProduto() {

		//inicializando o validador..

		validador = new ValidadorProduto(); //instanciando..

	}



	public Integer lerIdProduto(){

		//tratamento de exceções..

		try{

			int idProduto = Integer.parseInt(

			JOptionPane.showInputDialog("Informe o Id do Produto:"));
```

```
                if(validador.validarIdProduto(idProduto)){
                        return idProduto;
                }
                else{
                        //lançar uma exceção...
                        throw new Exception("Id do Produto invalido.");
                }
        }
        catch(Exception e){
                //exibir mensagem de erro..
                JOptionPane.showMessageDialog(null, "Erro: "
                                + e.getMessage());
                //recursividade..
                return lerIdProduto();
        }
}

public String lerNome(){

        try{
                String nome = JOptionPane.showInputDialog("Informe
                        o Nome do Produto:");
```

```
                if(validador.validarNome(nome)){
                        return nome;
                }
                else{
                        throw new Exception("Nome do Produto invalido.");
                }
        }
        catch(Exception e){
                //exibir mensagem de erro..
                JOptionPane.showMessageDialog(null, "Erro: "
                                + e.getMessage());
                //recursividade..
                return lerNome();
        }
}

public Double lerPreco(){
        try{
                Double preco = Double.parseDouble(
                JOptionPane.showInputDialog("Informe o Preço do Produto:"));
                if(validador.validarPreco(preco)){
                        return preco;
                }
```

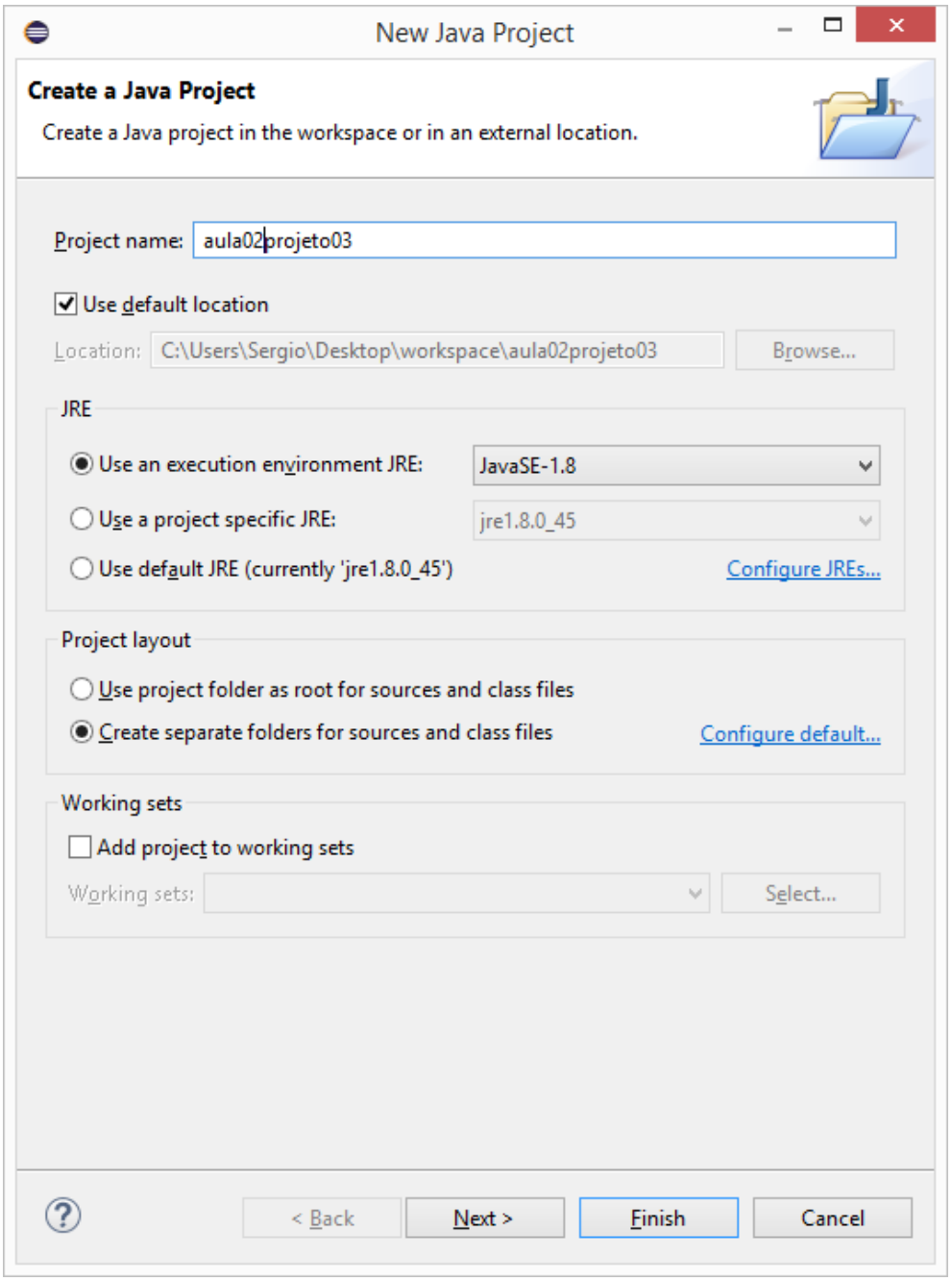
```
                else{
                        throw new Exception("Preço do Produto invalido.");
                }
            }
            catch(Exception e){
                //exibir mensagem de erro..
                JOptionPane.showMessageDialog(null, "Erro: "
                        + e.getMessage());
                //recursividade..
                return lerPreco();
            }
        }
}
```

Executando na Classe Main:

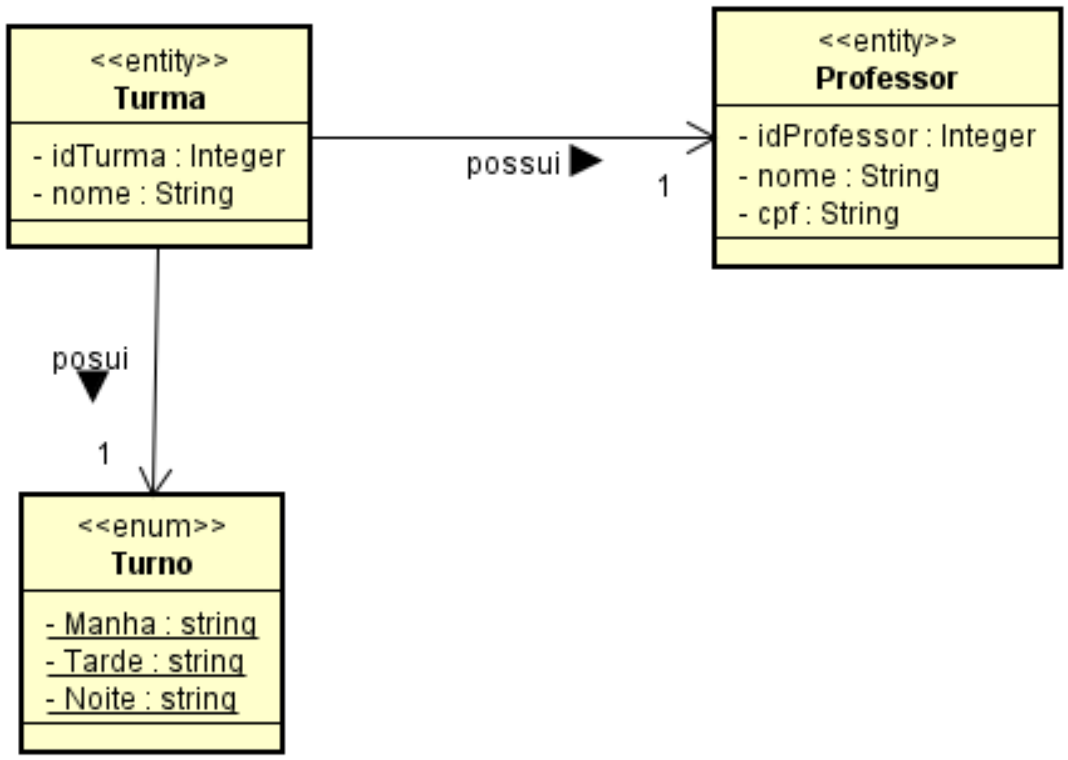
```
package principal;

import entities.Produto;
import input.InputProduto;

public class Main {

    public static void main(String[] args) {

        Produto p = new Produto(); //entidade..
        InputProduto input = new InputProduto(); //entrada de dados..

        //ler os dados do produto..
        p.setIdProduto(input.lerIdProduto());
        p.setNome(input.lerNome());
```

```
                p.setPreco(input.lerPreco());

                //imprimir..
                System.out.println(p);
        }
}
```

Novo projeto:



Modelagem de entidades:



Criando o enum:

```
package entities.types;

public enum Turno {

	Manha,
	Tarde,
	Noite
}
```

Criando as entidades:

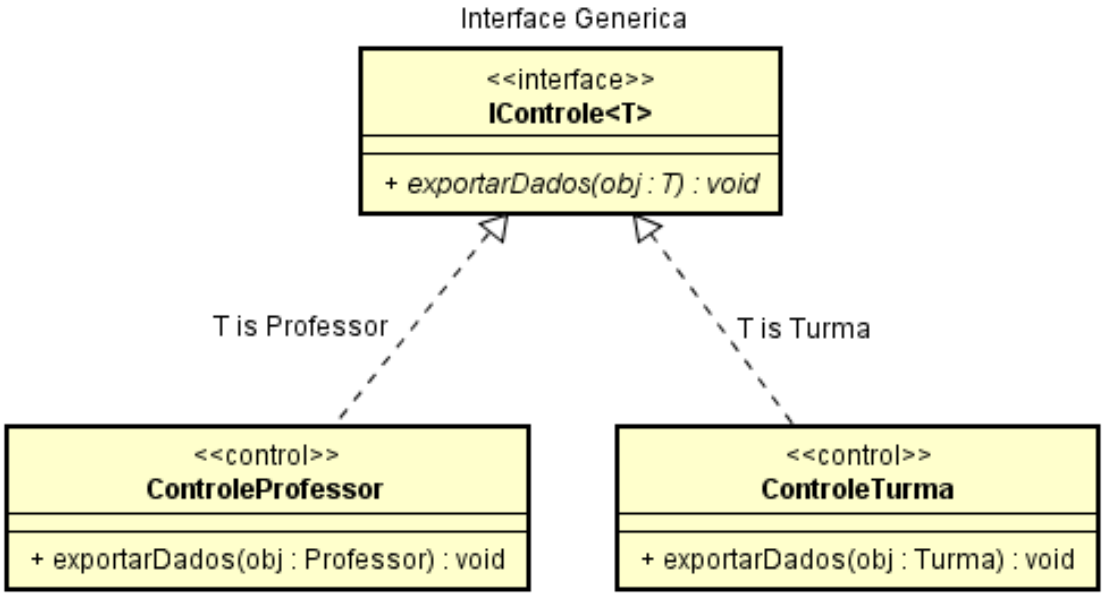
```
package entities;

//JavaBean (POJO)
public class Professor {

	private Integer idProfessor;
	private String nome;
	private String cpf;
```

```
	public Professor() {
		// construtor default..
	}

	// Sobrecarga de construtores..
	public Professor(Integer idProfessor, String nome, String cpf) {
		this.idProfessor = idProfessor;
		this.nome = nome;
		this.cpf = cpf;
	}

	public Integer getIdProfessor() {
		return idProfessor;
	}

	public void setIdProfessor(Integer idProfessor) {
		this.idProfessor = idProfessor;
	}

	public String getNome() {
		return nome;
	}

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

	public String getCpf() {
		return cpf;
	}

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

	@Override
	public String toString() {
		return "Professor [idProfessor=" + idProfessor
			+ ", nome=" + nome + ", cpf=" + cpf + "]";
	}

}


package entities;

import entities.types.Turno;
```

```
public class Turma {

	private Integer idTurma;
	private String nome;
	private Turno turno; // enum..
	private Professor professor; // entidade (Associação)

	public Turma() {
		// construtor default..
	}

	// sobrecarga de construtores..
	public Turma(Integer idTurma, String nome, Turno turno) {
		this.idTurma = idTurma;
		this.nome = nome;
		this.turno = turno;
	}

	// sobrecarga de construtores..
	public Turma(Integer idTurma, String nome, Turno turno,
				Professor professor) {
		this(idTurma, nome, turno);
		this.professor = professor;
	}

	public Integer getIdTurma() {
		return idTurma;
	}

	public void setIdTurma(Integer idTurma) {
		this.idTurma = idTurma;
	}

	public String getNome() {
		return nome;
	}

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

	public Turno getTurno() {
		return turno;
	}

	public void setTurno(Turno turno) {
		this.turno = turno;
	}

	public Professor getProfessor() {
```

```
			return professor;
		}

		public void setProfessor(Professor professor) {
			this.professor = professor;
		}

		@Override
		public String toString() {
			return "Turma [idTurma=" + idTurma + ", nome="
					+ nome + ", turno=" + turno + "]";
		}
}
```

Criando uma interface genérica:




```
package contracts;

//<T> tipo de dado generico..
public interface IControle<T> {

	//método para exportar dados..
	//em uma interface os metodos são implicitamente
	//publicos e abstratos (somente assinatura)

	void exportarDados(T obj) throws Exception;

}
```

Implementando a interface para a entidade Professor:

```
package controls;

import java.io.File;
import java.io.FileWriter;

import contracts.IControle;
import entities.Professor;

public class ControleProfessor implements IControle<Professor>{

	@Override
	public void exportarDados(Professor obj) throws Exception {

		//abrindo um arqivo para escrita...
		FileWriter arq = new FileWriter(
			new File("c:\\temp\\professores.txt"), true);

		arq.write(obj.toString()); //toString() do Professor..
		arq.write("\r\n"); //quebra de linha..

		arq.close(); //fechar o arquivo
	}

}
```

Implementando a interface para a entidade Turma:

```
package controls;

import java.io.File;
import java.io.FileWriter;

import contracts.IControle;
import entities.Turma;

public class ControleTurma implements IControle<Turma>{

	@Override
	public void exportarDados(Turma obj) throws Exception {


		//abrindo um arquivo para escrita..
```

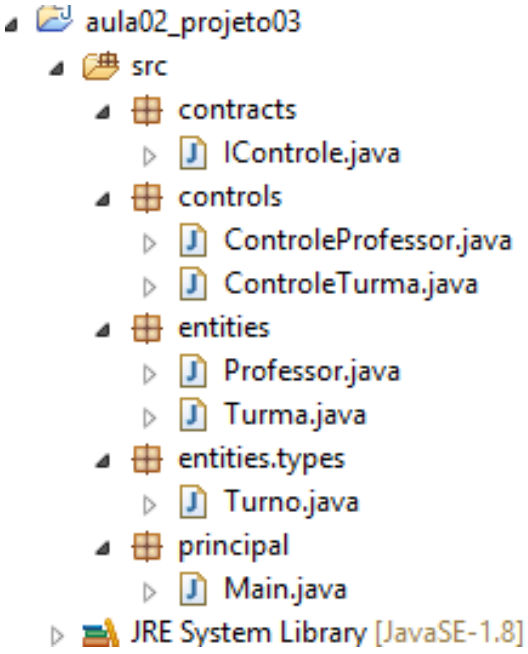
```
                FileWriter arq = new FileWriter(new File("c:\\temp\\turmas.txt"), true);

                arq.write(obj.toString()); //gravando o toString() da turma..
                arq.write("\r\n");

                //verificar se a turma tem Professor instanciado..
                if(obj.getProfessor() != null){
                        arq.write(obj.getProfessor().toString());
                        //toString() do professor..
                        arq.write("\r\n\r\n");
                }

                arq.close(); //fechar o arquivo..
        }
}
```

Testando na Classe Main:

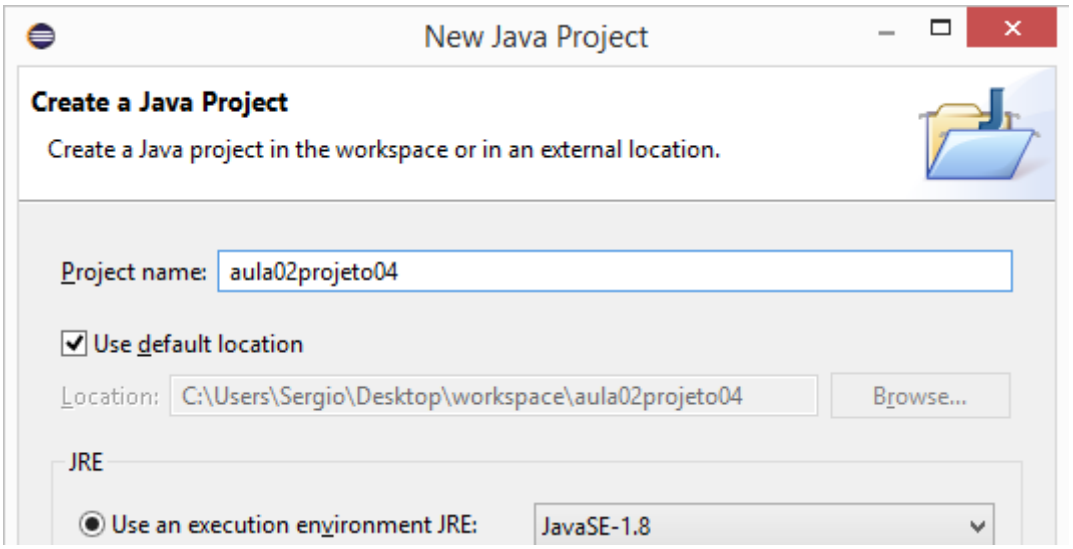
```
package principal;

import controls.ControleProfessor;
import controls.ControleTurma;
import entities.Professor;
import entities.Turma;
import entities.types.Turno;

public class Main {

        public static void main(String[] args) {

                Turma t = new Turma(1, "Java BRQ SP", Turno.Tarde);
                Professor p = new Professor(1, "Sergio Mendes", "1234567890");

                //atribuir o professor à turma
                t.setProfessor(p);

                ControleTurma controleTurma = new ControleTurma();
                ControleProfessor controleProfessor = new ControleProfessor();

                try {
                        controleProfessor.exportarDados(p);
                        controleTurma.exportarDados(t);

                        System.out.println("Dados gravados.");
```

```
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
}
```

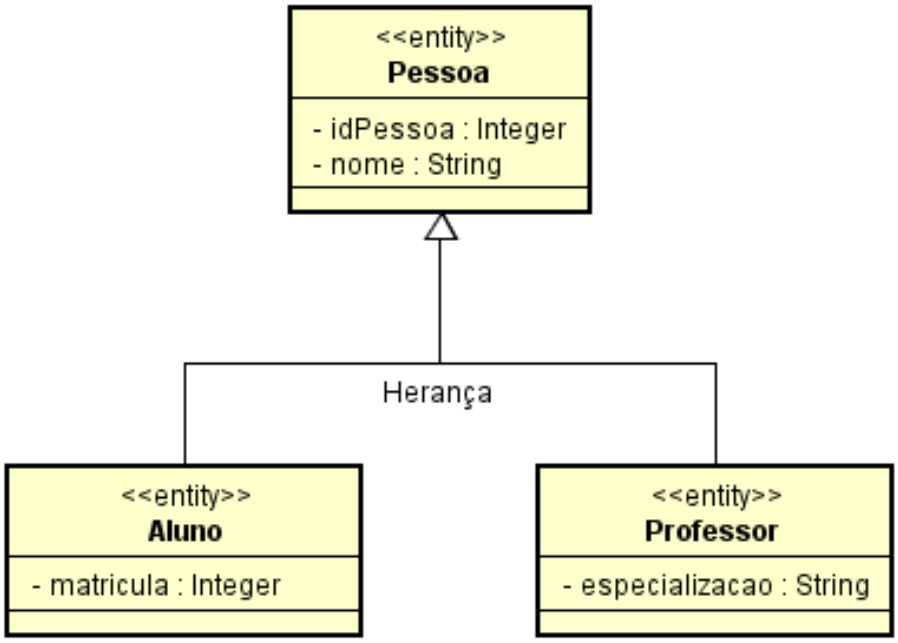
Estrutura do projeto:



Novo projeto:



Modelando uma relação de herança:




```
package entities;

public class Pessoa {

	private Integer idPessoa;
	private String nome;

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

	public Pessoa(Integer idPessoa, String nome) {
		this.idPessoa = idPessoa;
		this.nome = nome;
	}

	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;
	}

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

}


package entities;

public class Aluno extends Pessoa{

	private Integer matricula;

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

	public Aluno(Integer idPessoa, String nome, Integer matricula) {
		super(idPessoa, nome);
		this.matricula = matricula;
	}

	public Integer getMatricula() {
		return matricula;
	}

	public void setMatricula(Integer matricula) {
		this.matricula = matricula;
	}

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


package entities;

public class Professor extends Pessoa{

	private String especializacao;
```

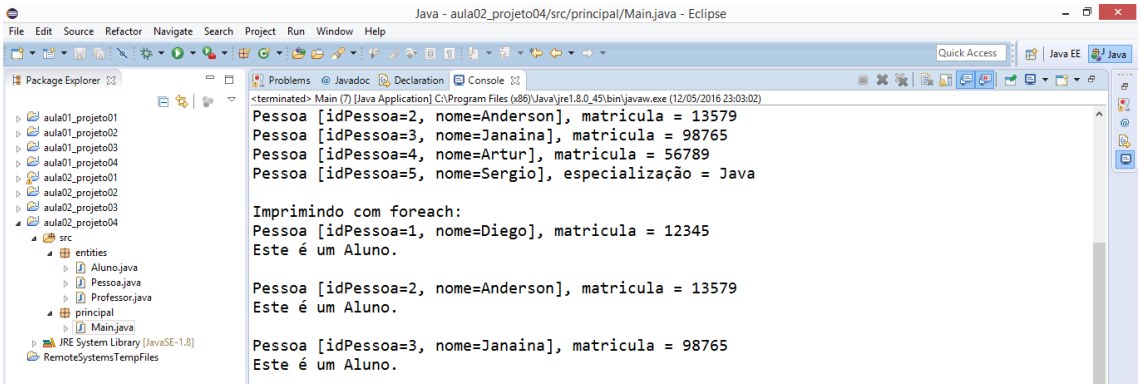
```
	public Professor() {
		// TODO Auto-generated constructor stub
	}

	public Professor(Integer idPessoa, String nome,
			String especializacao) {
		super(idPessoa, nome);
		this.especializacao = especializacao;
	}

	public String getEspecializacao() {
		return especializacao;
	}

	public void setEspecializacao(String especializacao) {
		this.especializacao = especializacao;
	}

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

Criando um vetor de objetos da classe Pessoa:

```
package principal;

import entities.Aluno;
import entities.Pessoa;
import entities.Professor;

public class Main {

	public static void main(String[] args) {

		//vetor...
		Pessoa[] vetor = new Pessoa[5];

		vetor[0] = new Aluno(1, "Diego", 12345);
		vetor[1] = new Aluno(2, "Anderson", 13579);
		vetor[2] = new Aluno(3, "Janaina", 98765);
		vetor[3] = new Aluno(4, "Artur", 56789);
		vetor[4] = new Professor(5, "Sergio", "Java");
```

```
        //varrendo o vetor..
        System.out.println("\nImprimindo com while:");
        int i = 0;
        while(i < vetor.length){
                System.out.println(vetor[i]); //toString()
                i++;
        }

        System.out.println("\nImprimindo com for:");
        for(int j = 0; j < vetor.length; j++){
                System.out.println(vetor[j]); //toString()
        }

        System.out.println("\nImprimindo com foreach:");
        for(Pessoa p : vetor){

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

                // se p (Pessoa) é um Aluno
                if(p instanceof Aluno){
                        System.out.println("Este é um Aluno.\n");
                }
                else if(p instanceof Professor){
                        System.out.println("Este é um Professor.\n");
                }
        }

    }

}
```

Saída do programa:



```
Imprimindo com while:
Pessoa [idPessoa=1, nome=Diego], matricula = 12345
Pessoa [idPessoa=2, nome=Anderson], matricula = 13579
Pessoa [idPessoa=3, nome=Janaina], matricula = 98765
Pessoa [idPessoa=4, nome=Artur], matricula = 56789
Pessoa [idPessoa=5, nome=Sergio], especialização = Java

Imprimindo com for:
Pessoa [idPessoa=1, nome=Diego], matricula = 12345
Pessoa [idPessoa=2, nome=Anderson], matricula = 13579
Pessoa [idPessoa=3, nome=Janaina], matricula = 98765
Pessoa [idPessoa=4, nome=Artur], matricula = 56789
Pessoa [idPessoa=5, nome=Sergio], especialização = Java

Imprimindo com foreach:
Pessoa [idPessoa=1, nome=Diego], matricula = 12345
Este é um Aluno.

Pessoa [idPessoa=2, nome=Anderson], matricula = 13579
Este é um Aluno.

Pessoa [idPessoa=3, nome=Janaina], matricula = 98765
Este é um Aluno.

Pessoa [idPessoa=4, nome=Artur], matricula = 56789
Este é um Aluno.

Pessoa [idPessoa=5, nome=Sergio], especialização = Java
Este é um Professor.
```

Classe	• Representação de um conjunto de objetos com características afins. Definição das funções do objeto (Métodos) e seus dados (Atributos)
Objeto	• Uma instância de uma Classe • Armazenamento de estados através de seus atributos e reação a mensagens enviadas por outros objetos.
Herança	• Mecanismo pela qual uma classe (subclasse) pode estender ou derivar de outra classe (superclasse), herdando seus comportamentos e atributos.
Polimorfismo	• Principio pelo qual as instâncias de duas ou mais Classes derivadas de uma mesma superclasse podem invocar métodos com a mesma assinatura, mas com comportamentos distintos.
Encapsulamento	• Proibição do acesso ao conteúdo de uma classe (geralmente atributos), disponibilizando apenas métodos que permitam tal acesso.