



**Attribution-NonCommercial-
NoDerivatives 4.0 International
(CC BY-NC-ND 4.0)**



Este trabalho está licenciado com uma Licença [Creative Commons - Atribuição-NãoComercial-SemDerivações 4.0 Internacional](#).

Programação Orientada a Objetos - UTFPR Campus Apucarana



Programação Orientada a Objetos

BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO

PROF. LUCIO AGOSTINHO ROCHA

AULA 1: INTRODUÇÃO

2º SEMESTRE 2022

Programação Orientada a Objetos - UTFPR Campus Apucarana

Sejam Bem-Vindos!

Programação Orientada a Objetos - UTFPR Campus Apucarana

Introdução

- Apresentação do professor
- Disciplina: Programação Orientada a Objetos – POCO4A
- Turmas: COM4A - 2022/2

Introdução

5

- **Objetivos:**

- O aluno será capaz de desenvolver sistemas baseados nos conceitos formais da Orientação a Objetos, compreender os paradigmas da programação orientada a objetos e implementar estudos de caso em Linguagem de Programação Orientada a Objetos.

Introdução

6

- **Planejamento**

- Plano de Ensino e Ementa
- Sistemas de avaliação
- Previsão de datas

- **Plataforma Classroom**

- Material e trabalhos da disciplina
- Faça a sua inscrição na disciplina
- Grupo de mensagens

Breve Histórico: Programação Orientada a Objetos

Programação Orientada a Objetos - UTFPR Campus Apucarana

Breve Histórico: Programação Orientada a Objetos

- Ao longo do anos, o ciclo de desenvolvimento de software passou por muitas mudanças.



Programação Orientada a Objetos - UTFPR Campus Apucarana

Breve Histórico: Programação Orientada a Objetos

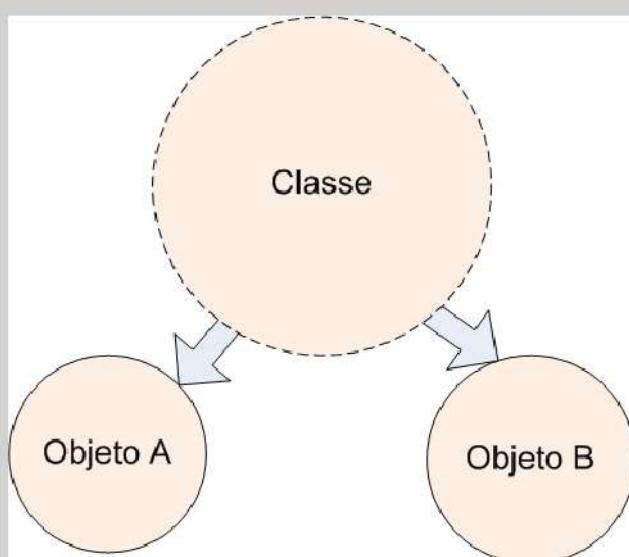
9

- A Programação Orientada a Objetos (POO) é um tipo de programação em linguagem de alto nível.
Propriedades e Comportamentos são armazenados em objetos:
 - Propriedades: são atributos do objeto.
 - Comportamentos: são as ações/operações que o objeto realiza.
 - Classe: é um modelo que descreve objetos.
 - Classe: é um modelo abstrato que define as propriedades e comportamentos comuns aos objetos da classe.
 - Objeto: é uma instância de uma classe.
 - Objetos: devem ser capazes de interagir uns com os outros através de seus métodos.

Programação Orientada a Objetos - UTFPR Campus Apucarana

Breve Histórico: Programação Orientada a Objetos

10



- Figura - Representação Simplificada de Classes e Objetos.

Programação Orientada a Objetos - UTFPR Campus Apucarana

Breve Histórico: Programação Orientada a Objetos

11

- A Programação Orientada a Objetos (POO) é uma metodologia de desenvolvimento de software que busca:
 - Reduzir a Complexidade para descrever o software: formar abstrações da aplicação em termos de “objetos” e “classes”;
 - Simplificar a criação de interfaces amigáveis “WYSIWYG” (What You See Is What You Get): interfaces gráficas, janelas, menus e ícones são vistos como “objetos”;
 - Reusabilidade:
 - ☒ Reutilizar componentes de software facilita o rápido desenvolvimento.
 - ☒ Utilizar outros componentes ao invés de criar novos.
 - ☒ Herdar bons componentes ao invés de “reinventar a roda”

Programação Orientada a Objetos - UTFPR Campus Apucarana

Breve Histórico: Programação Orientada a Objetos

12

- A ideia de “objeto” surgiu de várias abordagens diferentes, ainda na década de 70:
 - Simula [5]: linguagem de simulação de eventos discretos. Primeira linguagem a utilizar “classes” e “objetos” para modelar entidades na simulação de aplicações reais.
 - Hoare [6]: recursos protegidos em sistemas operacionais.
 - CLU [7]: abstração de dados em linguagem de programação: instâncias de tipos de dados abstratos (“objetos”) com operações encapsuladas e exclusivas em regiões protegidas.
 - Minsky [8]: estado (propriedades) e comportamentos (ações) armazenados em “frames” (“objetos”). Por exemplo: objeto estudante tem as propriedades “nome” e “data de nascimento”, e ações de “recuperar” o nome e “calcular” a idade de acordo com a data de nascimento.

Programação Orientada a Objetos - UTFPR Campus Apucarana

Breve Histórico: Programação Orientada a Objetos

13



- Figura: Paradigma Orientado a Objetos (Fonte: CAPRETZ, 2003)

Programação Orientada a Objetos - UTFPR Campus Apucarana

Breve Histórico: Programação Orientada a Objetos

14

- O termo “orientado a objetos” surgiu da linguagem Smalltalk (Xerox Palo Alto, década de 70). Smalltalk teve forte influência das linguagens Simula e LISP.
- Diferentes definições de POO:
 - 1) Capretz[1]: POO é aquela que utiliza objetos. Um objeto é o nome de um tipo de dado abstrato instanciado a partir de uma classe. Um objeto possui variáveis privadas e procedimentos locais.
 - 2) Capretz[1]: POO é aquela que utiliza objetos de um determinado tipo. O objeto pode estar relacionado a outros objetos através de relações de subtipo e supertipo definidas por suas classes.

Programação Orientada a Objetos - UTFPR Campus Apucarana

Breve Histórico: Programação Orientada a Objetos

15

- ...Diferentes definições de POO:

- 3) Rentsch[6]: POO é aquela que utiliza objetos cujos atributos (propriedades) não são visíveis fora do objeto. Toda comunicação entre objetos é feita com passagem de mensagens. Todo processamento é realizado dentro dos objetos. O compartilhamento de propriedades entre os objetos é feito por herança.
- 4) Wegner[8]: POO é aquela que utiliza objetos como entidades autônomas que possuem um estado e respondem a mensagens. Classes agrupam objetos que têm atributos e operações em comum. Herança é utilizada para organizar as classes de acordo com características comuns.

Breve Histórico: Programação Orientada a Objetos

16

- Todas essas definições citam o termo “objeto” :

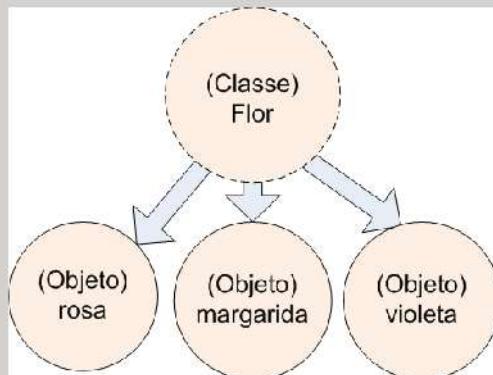
- Objeto possui variáveis privadas (atributos)
- Objeto possui procedimentos locais (métodos)
- Classe é um modelo que define os atributos e métodos (comportamentos/ações) comuns aos objetos da classe.
- Objeto é instanciado a partir de uma classe.

Breve Histórico: Programação Orientada a Objetos

17

- Exemplos:

- “rosa” (objeto) é uma instância de “Flor” (Classe)
- “margarida” (objeto) é uma instância de “Flor” (Classe)
- “violeta” (objeto) é uma instância de “Flor” (Classe)



- Figura - Representação Simplificada de Classes e Objetos.

Programação Orientada a Objetos - UTFPR Campus Apucarana

Breve Histórico: Programação Orientada a Objetos

18

Flor
- cor : string
- nome : string
+ getNome() : string
+ getCor() : string
+ setNome(nome : string)
+ setCor(cor : string)

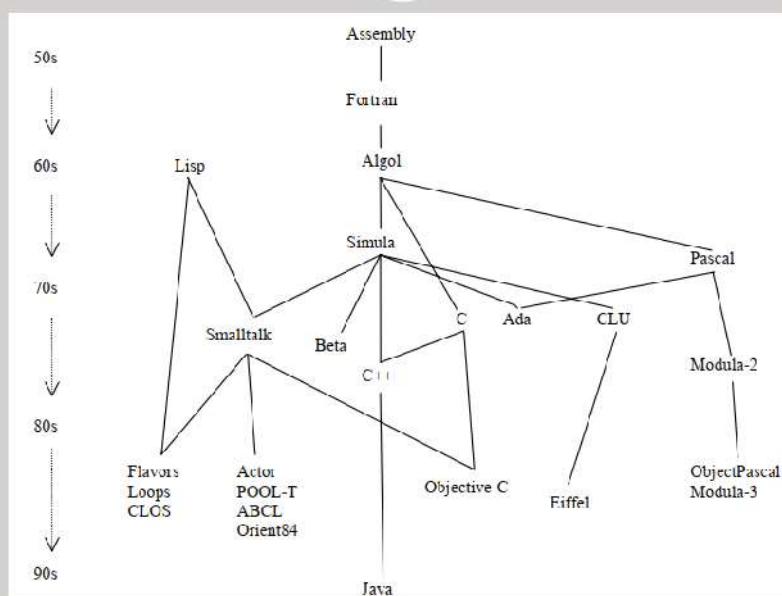
Figura - Diagrama UML de Classes.

- Atributos: cor, nome
- Métodos acessores: get
- Métodos modificadores: set

Programação Orientada a Objetos - UTFPR Campus Apucarana

Breve Histórico: Programação Orientada a Objetos

19



- Figura - Evolução das Linguagens de Programação (Fonte: CAPRETZ, 2003)

Programação Orientada a Objetos - UTFPR Campus Apucarana

20

Motivação e Uso da Tecnologia

Programação Orientada a Objetos - UTFPR Campus Apucarana

Motivação e Uso da Tecnologia

21

- Características da POO:
 - Encapsulamento: “objeto” protege os seus atributos (propriedades) e oculta a implementação de seus métodos. Os atributos e métodos são acessíveis através de:
 - ✖ Modificadores de acesso
 - ✖ Métodos acessores
 - ✖ Métodos mutadores.
 - Ocultamento: atributos e métodos são visíveis apenas dentro do contexto do objeto.
 - Fraco acoplamento: objeto possui apenas as informações específicas do seu contexto.
 - Modularidade: a aplicação é dividida em partes menores (“objetos”) que contêm informações específicas (estado) para determinado propósito.
 - Comunicação via Interfaces: objetos se comunicam por troca de mensagens através de suas interfaces.

Programação Orientada a Objetos - UTFPR Campus Apucarana

Motivação e Uso da Tecnologia

22

- Programas Orientados a Objetos são mais fáceis de entender, corrigir e modificar.
- Objetos: componentes de software reutilizáveis.
- Objetos: possuem estado (propriedades: atributos) e comportamentos (ações: métodos)
- Herança: permite o reaproveitamento de código.
- Classes: modelos genéricos que definem atributos e comportamentos comuns para instanciar objetos.

Programação Orientada a Objetos - UTFPR Campus Apucarana

Linguagem de Programação Java

Programação Orientada a Objetos - UTFPR Campus Apucarana

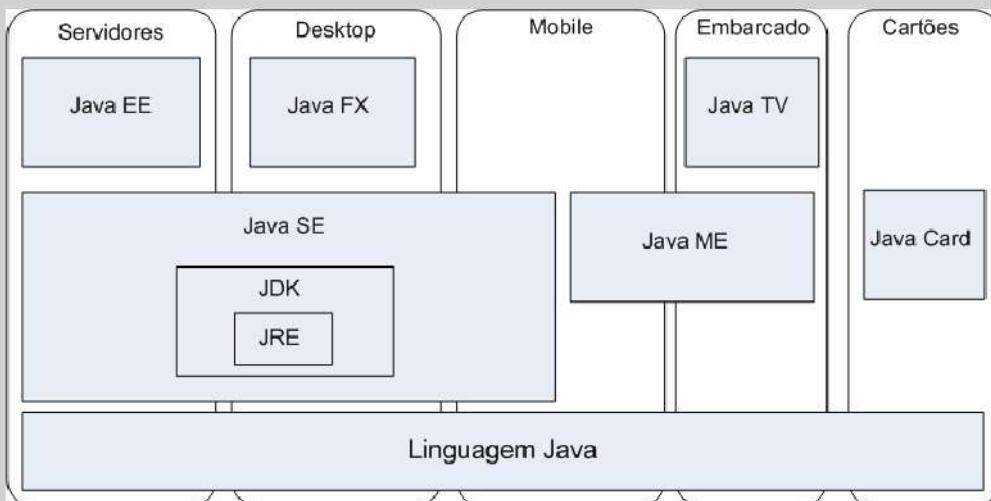
Linguagem de Programação Java

- Java é uma linguagem de programação orientada a objetos
- Java é baseada nas linguagens C e C++
- James Gosling (Sun Microsystems, 1991). A Sun Microsystems foi adquirida pela Oracle em 2010.
- Conteúdo dinâmico para a Web.
- Atualmente, mais de 8 bilhões de dispositivos (de software embarcado a sistemas de computação de alto desempenho)
- Ecossistema:
 - Java Runtime Edition (JRE): apenas para a execução de programa Java.
 - Java Development Kit (JDK): kit de desenvolvimento de programa Java. Inclui o JRE
 - Java Standard Edition (Java SE): JRE + JDK
 - Muitas outras versões: Java Card, Java ME Micro-Edition, Java Micro-Profile, Java EE Enterprise-Edition

Programação Orientada a Objetos - UTFPR Campus Apucarana

Linguagem de Programação Java

25



- Figura – Plataformas de Desenvolvimento em Linguagem Java.

Programação Orientada a Objetos - UTFPR Campus Apucarana

Linguagem de Programação Java

26

- Java é multiplataforma: escreva uma vez, execute em qualquer lugar.
- Nenhuma alteração específica de hardware precisa ser incorporada ao código-fonte.
- Código-fonte é compilado em bytecodes. Bytecodes podem ser executados em qualquer computador que possua uma JVM instalada.
- O código compilado (bytecodes) é um arquivo .class
- Programas Java compilados em bytecodes executam em uma Java Virtual Machine (JVM)
- JVM emula o hardware e a CPU do computador.

Programação Orientada a Objetos - UTFPR Campus Apucarana

Linguagem de Programação Java

27

- Ambiente de Programação Java:
 - Sistema Operacional
 - Linguagem
 - APIs
 - Bibliotecas de Classe

Linguagem de Programação Java

28

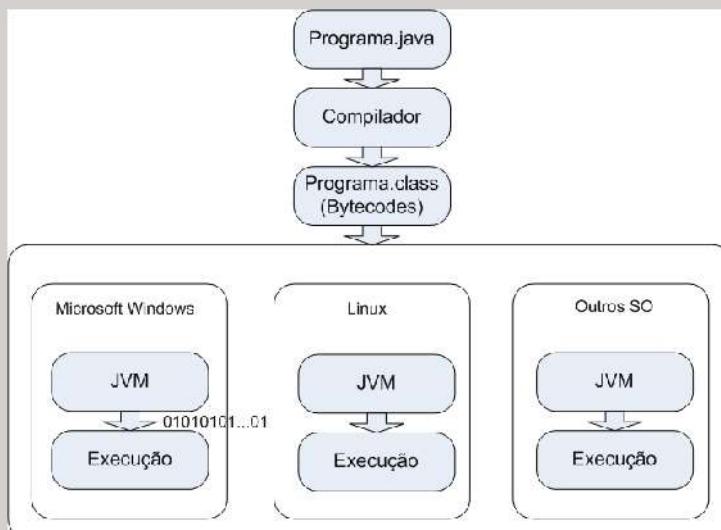


Figura: Fases de Execução de um Programa Java.

Linguagem de Programação Java

29

- **Fases de Execução de um Programa Java:**

- Edição: programador escreve o programa e salva em disco.
- Compilação: compilador cria bytecodes
- Carregamento: armazenagem de bytecodes na memória
- Verificação: verificação de requisitos de segurança
- Execução: interpretador traduz bytecodes em código de máquina.

30

Programação em Java

Programação em Java

31

- Estrutura de um programa Java:
 - Classe: é a unidade básica do código que descreve a lógica do programa.
 - Classe: é um modelo que descreve as propriedades gerais de um objeto.
 - Objeto: é uma instância da classe.
 - Pacote: é um conjunto de classes. Semelhante às bibliotecas da linguagem C.

Programação Orientada a Objetos - UTFPR Campus Apucarana

32



Revisão

Programação Orientada a Objetos - UTFPR Campus Apucarana

Revisão

33

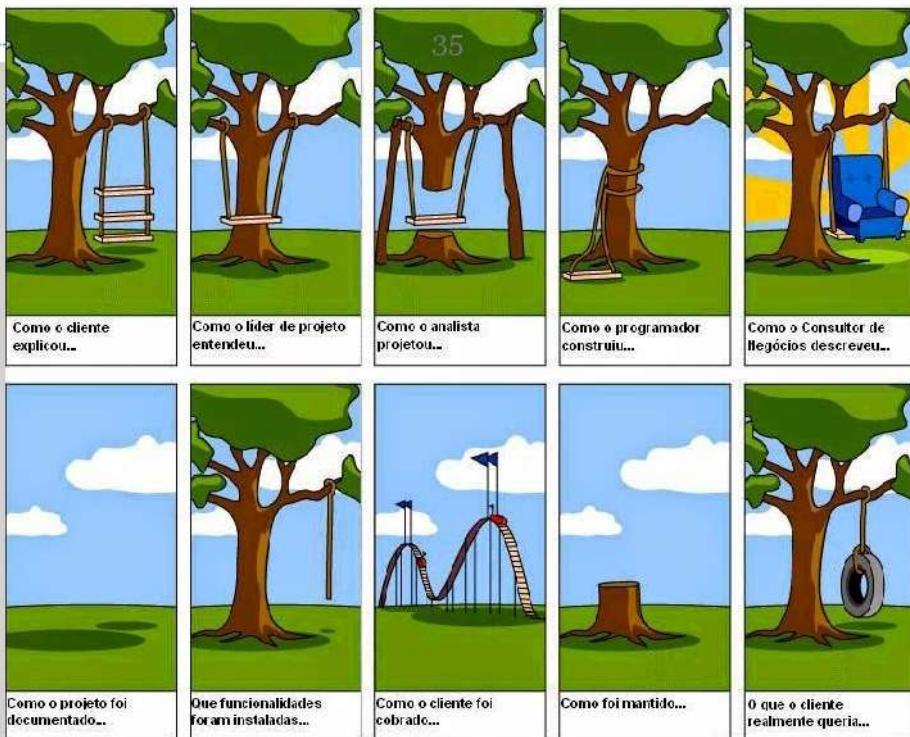
- A Programação Orientada a Objetos (POO) é um paradigma de programação no qual propriedades e comportamentos são armazenados em objetos:
 - Propriedades: são atributos do objeto.
 - Comportamentos: são as ações/operações que o objeto realiza.
 - Classe: é um modelo que descreve objetos.
 - Classe: é um modelo que define as propriedades e comportamentos comuns a todos os objetos.
 - Objeto: é uma instância de uma classe.
 - Objetos: devem ser capazes de interagir uns com os outros através de seus métodos.

Revisão

34

- POO surgiu na década de 70 como uma necessidade do mercado e do ciclo de desenvolvimento do software: redução de complexidade, contribuição de outras áreas (sistemas operacionais, IA, técnicas orientadas a abstração de dados).
- Wegner[8]: POO é um paradigma de programação que utiliza objetos como entidades autônomas que possuem um estado e respondem a mensagens. Classes agrupam objetos que têm atributos e operações em comum. Herança é utilizada para organizar as classes de acordo com características comuns.
- Características da POO:
 - Encapsulamento
 - Ocultamento
 - Fraco acoplamento
 - Modularidade
 - Comunicação via Interfaces
- Java é uma linguagem de Programação Orientada a Objetos.

Para pensar...



Fonte: <http://calvinberschacher.blogspot.com.br/2014/06/projeto-balanco-no-agile.html>. Acesso em 09/02/2015.

Programação Orientada a Objetos - UTFPR Campus Apucarana

Exercícios

36

<Ver conteúdo na Plataforma de Ensino>



Programação Orientada a Objetos - UTFPR Campus Apucarana

Para lembrar...

37

- Precedência de Operadores

Operador	Significado
()	Parênteses
* , / , %	Multiplicação, Divisão, Módulo
+ , -	Adição, Subtração
=	Atribuição

- Avaliação da esquerda para a direita. A cada avaliação, reinicia da esquerda para a direita novamente.
- Expressões delimitadas por Parênteses são avaliadas primeiro. Se há várias expressões, as expressões nos Parênteses mais internos são avaliadas primeiro.
- Multiplicação, divisão e resto têm o mesmo nível de precedência.
- Adição e Subtração têm o mesmo nível de precedência.

Programação Orientada a Objetos - UTFPR Campus Apucarana

Para lembrar...

38

- Precedência de Operadores Relacionais

Operador	Significado
>	X é maior que Y
<	X é menor que Y
>=	X é maior ou igual a Y
<=	X é menor ou igual a Y
==	X é igual a Y
!=	X é diferente de Y

Programação Orientada a Objetos - UTFPR Campus Apucarana

Referências

39

- Referências bibliográficas da disciplina.
- CAPRETZ, L. F. "A Brief History of the Object-Oriented Approach". In: ACM SIGSOFT Software Engineering Notes, vol.28, no.2, p.6, 2003.
- The Java Tutorials. Disponível em: <https://docs.oracle.com/javase/tutorial>. Acesso em Maio de 2021.



**Attribution-NonCommercial-NoDerivatives 4.0 International
(CC BY-NC-ND 4.0)**



Este trabalho está licenciado com uma Licença [Creative Commons - Atribuição-NãoComercial-SemDerivações 4.0 Internacional](#).

Programação Orientada a Objetos - UTFPR Campus Apucarana



Programação Orientada a Objetos

BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO

PROF. LUCIO AGOSTINHO ROCHA

AULA 2: AMBIENTE DE DESENVOLVIMENTO

2º SEMESTRE 2022

Programação Orientada a Objetos - UTFPR Campus Apucarana

Ambiente de Desenvolvimento

Ambiente de Desenvolvimento

- Eclipse IDE for Java Developers:
 - Instalador faz a instalação padrão do JavaSE
- Verificações:
 - PATH
 - JAVA_HOME
- Versão da JVM:

```
C:\> java -version
```

Ambiente de Desenvolvimento

5

- Diversas IDEs:

- Apache NetBeans
- IntelliJ IDEA
- BlueJ
- MyEclipse
- Xcode
- Tabnine
- Notepad++
- Editores online, ...

Ambiente de Desenvolvimento

6

- Linha de Comando:

- É necessário ter o caminho do compilador Java na variável de ambiente PATH do usuário.
- Classpath: caminho da pesquisa das classes e arquivos zip/jar

- Compilação:

```
C:\> javac -cp . Arquivo.java
```

- Execução:

```
C:\> java -cp . Arquivo
```



Revisão

Revisão

- O ambiente de desenvolvimento (IDE) simplifica a organização do projeto orientado a objetos.
- A compilação de programas Java não precisa de um IDE, porém o seu uso simplifica a criação de novos programas.
- Java é multiplataforma, mas é necessário uma configuração diferente do ambiente de desenvolvimento em cada plataforma alvo (ex.: MacOS, Microsoft Windows, Linux, etc.)

Exercícios

9

<Ver conteúdo na Plataforma de Ensino>



Programação Orientada a Objetos - UTFPR Campus Apucarana

Referências

10

- Referências bibliográficas da disciplina.

Programação Orientada a Objetos - UTFPR Campus Apucarana

Programação Orientada a Objetos

1

BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO

PROF. LUCIO AGOSTINHO ROCHA

AULA 3: INTRODUÇÃO À PROGRAMAÇÃO ORIENTADA A
OBJETOS

2º SEMESTRE 2022

Programação Orientada a Objetos - UTFPR Campus Apucarana

2

Introdução à Programação Orientada a Objetos

Programação Orientada a Objetos - UTFPR Campus Apucarana

Introdução à Programação Orientada a Objetos

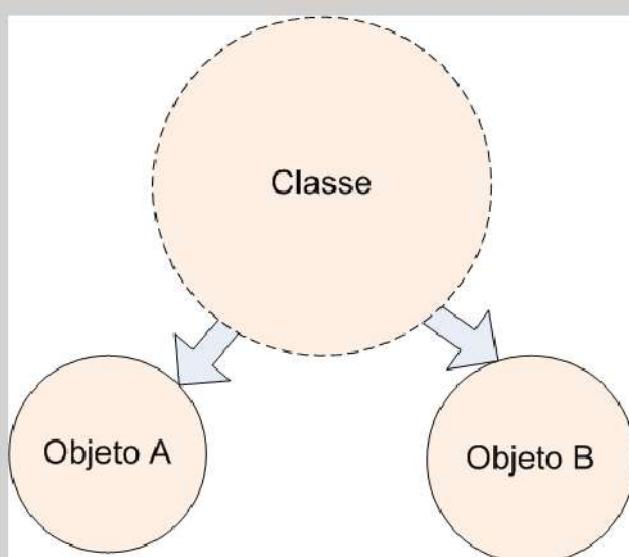
3

- A Programação Orientada a Objetos (POO) é um paradigma de programação no qual propriedades e comportamentos são armazenados em objetos:
 - Propriedades: são atributos do objeto.
 - Comportamentos: são as ações/operações que o objeto realiza.
 - Classe: é um modelo que descreve objetos.
 - Classe: é um modelo que define as propriedades e comportamentos comuns a todos os objetos.
 - Objeto: é uma instância de uma classe.
 - Objetos: devem ser capazes de interagir uns com os outros através de seus métodos.

Programação Orientada a Objetos - UTFPR Campus Apucarana

Introdução à Programação Orientada a Objetos

4



- Figura - Representação Simplificada de Classes e Objetos.

Programação Orientada a Objetos - UTFPR Campus Apucarana

Introdução à Programação Orientada a Objetos

5

- **Programação Orientada a Objetos:**

- É uma forma de modelar o mundo real
- Atributos: são as propriedades do objeto
 - ✖ Ex.: tamanho, cor, peso, nome, etc.
- Comportamentos: são as ações que o objeto realiza:
 - ✖ Ex.:
 - Um Computador calcula números;
 - Um Smartphone vibra por X>0 segundos;
 - Uma Floricultura vende flores, etc.

Introdução à Programação Orientada a Objetos

6

- **Classe:** é a unidade de programação em Java

- Java foca nos nomes (classes)
 - ✖ Ex.: Computador (nome) calcula (verbo/ação) números.
 - ✖ Ex.: Supermercado (nome) vende (verbo/ação) produtos.
- Classe é um modelo de objetos
 - ✖ Objetos são criados a partir das classes
- Classe contém métodos
 - ✖ Método implementa um comportamento (ação)
- Classe contém dados
 - ✖ Define atributos
- Classe é reutilizável
 - ✖ Formato padronizado para uso.

Introdução à Programação Orientada a Objetos

7

- **Tipos de dados primitivos**
 - byte: é um tipo de dado inteiro complemento de 2 de 8 bits com sinal.
 - short: é um tipo de dado inteiro complemento de 2 de 16 bits com sinal.
 - int: é um tipo de dado inteiro complemento de 2 de 32 bits com sinal.
 - long: é um tipo de dado inteiro complemento de 2 de 64 bits com sinal.
 - float: é um tipo de dado real de precisão única de 32 bits
 - double: é um tipo de dado real de dupla precisão de 64 bits
 - boolean: é um tipo de dado que possui apenas os valores 'true' e 'false'.
 - char: é um caracter único Unicode de 16-bits.

Programação Orientada a Objetos - UTFPR Campus Apucarana

Introdução à Programação Orientada a Objetos

8

- **Variáveis:**
 - Variáveis de instância: são atributos não-estáticos
 - Variáveis de classe: são atributos estáticos
 - Variáveis locais: existem dentro dos métodos
 - Parâmetros: são variáveis de entrada para os métodos
- A declaração de uma variável é feita como segue:

tipo variavel = valor;

- O **tipo** da variável determina o valor que ela pode conter e as operações que a variável pode realizar sobre o valor.
- Um objeto é uma variável cujo tipo é a Classe (Ricarte,2008).

Programação Orientada a Objetos - UTFPR Campus Apucarana

Introdução à Programação Orientada a Objetos

9

- Métodos:
 - São similares às funções da linguagem C.
 - Variáveis local: definidas no escopo do método.
 - Parâmetros: variáveis de entrada para o método.
- Sobrecarga de Métodos:
 - Na Classe, os métodos podem ter o mesmo nome, porém:
 - ✖ devem ter quantidades diferentes de argumentos

Programação Orientada a Objetos - UTFPR Campus Apucarana

Introdução à Programação Orientada a Objetos

10

- Sobrecarga de Métodos:
 - Na Classe, os métodos podem ter o mesmo nome, porém:
 - ✖ Parâmetros devem ser de tipos diferentes.
 - ✖ Quantidade diferente de parâmetros.
- ✖ Exemplo:

```
public void metodo1( int parâmetro );
public void metodo1( double parâmetro );
public void metodo1( int p1, double p2 );
```

Programação Orientada a Objetos - UTFPR Campus Apucarana

Introdução à Programação Orientada a Objetos

11

- **Passagem por Valor:**

- Conteúdo da variável de origem é copiado para a variável de destino.
- Tipos de dados primitivos

- **Passagem por Referência:**

- Objetos
- Vetores
- Matrizes

Introdução à Programação Orientada a Objetos

12

- **Programação Orientada a Objetos (POO):**

- Encapsula dados (atributos) e métodos (comportamentos)
- Objetos se comunicam através de interfaces.
- Classes são as unidades de programação.
- Classes encapsulam atributos e métodos.



Revisão

Revisão

- **Programação Orientada a Objetos:**
 - É uma forma de modelar o mundo real
 - Atributos: são as propriedades do objeto
 - Comportamentos: são as ações que o objeto realiza
 - Classe: é a unidade básica de programação em Java
 - ✖ Implementa métodos (ações) que o objeto realiza.
 - ✖ Define atributos: atributos que o objeto deve possuir.

Exercícios

15

<Ver conteúdo na Plataforma de Ensino>



Programação Orientada a Objetos - UTFPR Campus Apucarana

Referências

16

- Referências bibliográficas da disciplina.
- RICARTE, Ivan. 1.3.1 – Leitura do arquivo de origem. Introdução à Compilação. Elsevier Editora, 2008.

Programação Orientada a Objetos - UTFPR Campus Apucarana

Programação Orientada a Objetos

1

BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO

PROF. LUCIO AGOSTINHO ROCHA

AULA 5: PROGRAMAÇÃO ORIENTADA A OBJETOS

2º SEMESTRE 2022

Programação Orientada a Objetos - UTFPR Campus Apucarana

2

Programação Orientada a Objetos

Programação Orientada a Objetos - UTFPR Campus Apucarana

Introdução à Programação Orientada a Objetos

3

- **Programação Orientada a Objetos (POO):**

- Encapsula dados (atributos) e métodos (comportamentos)
- Objetos se comunicam através de interfaces.
- Classes são as unidades de programação.
- Classes encapsulam atributos e métodos.

Introdução à Programação Orientada a Objetos

4

- **Em Java, toda Classe herda de outra Classe:**

- Se não explicitado, implicitamente a Classe herda da Classe `java.lang.Object`

- **Construtor**

- É semelhante a um método, porém tem o mesmo nome da Classe
- Construtor não possui retorno.
- É utilizado para inicializar as variáveis de instância
- Classe pode ter vários construtores (sobrecarga), mas com parâmetros diferentes

Introdução à Programação Orientada a Objetos

5

- Sobrecarga de Métodos:
 - Por padrão, todo objeto possui um método `java.lang.Object.toString` que identifica unicamente o objeto instanciado.
 - O método ‘`toString`’ pode ser invocado:
 - Implicitamente : `System.out.println(objeto);`
 - Explicitamente: `System.out.println(objeto.toString());`

Introdução à Programação Orientada a Objetos

6

- Escopo da Classe:
 - Variáveis de instância e métodos (chamados de Membros)
 - Membros são acessíveis para todos os métodos da Classe.
- Escopo dos Métodos:
 - Variáveis locais existem apenas dentro dos métodos
 - Variáveis locais não são acessíveis fora do escopo do método.

Programação Orientada a Objetos

7

- **Modificadores de Acesso:**

- Muitas vezes, o programador não deseja que todos os atributos e métodos (Membros) de uma classe estejam disponíveis para outros usuários.
- Java fornece Ocultamento de Informação através de Modificadores de Acesso.
- Um Modificador de Acesso indica se outras Classes podem ou não utilizar um Membro da Classe.
- Nota: Modificador de Acesso é para visibilidade externa dos membros. Todos os membros de Classe são visíveis dentro da Classe, independente do modificador de acesso.

Programação Orientada a Objetos

8

- **Modificadores de Acesso:**

- Nível Superior (Classe):
 - ✖ public, ou private-package (sem modificador)
- Nível Membro (Variáveis, Métodos ou Classes Internas):
 - ✖ public, private, protected, ou private-package (sem modificador)

Programação Orientada a Objetos

9

- **Modificadores de Acesso: Nível Superior**

- public: classe visível para todas as outras classes de qualquer lugar
- Sem modificador (package-private): classe visível apenas dentro do seu package.
- Apenas public, abstract e final são permitidos para a Classe.

Programação Orientada a Objetos

10

- **Modificadores de Acesso: Nível Membro**

- public: membro é visível para todas as outras classes de qualquer lugar
- Sem modificador (package-private (privado de pacote)): membro é visível apenas dentro do seu package.
- protected: membro é visível apenas por classes dentro do seu package (mesmo que package-private) + visível por classes que herdam dessa classe (subclasses) em outros pacotes.
- private: membro é acessível por classes apenas dentro do seu pacote.
 - Métodos Acessores: get
 - Métodos Mutadores: set

Programação Orientada a Objetos

11

- Visibilidade dos membros da Classe:

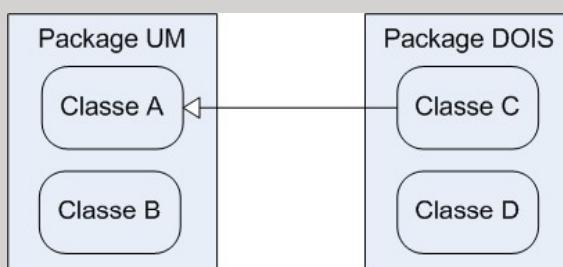
Modificador da Classe	Classe	Package	Subclasse	Fora do Pacote
public	●	●	●	●
protected	●	●	●	●
Sem modificador	●	●	●	●
private	●	●	●	●

Programação Orientada a Objetos - UTFPR Campus Apucarana

Programação Orientada a Objetos

12

- Visibilidade dos membros da Classe A:



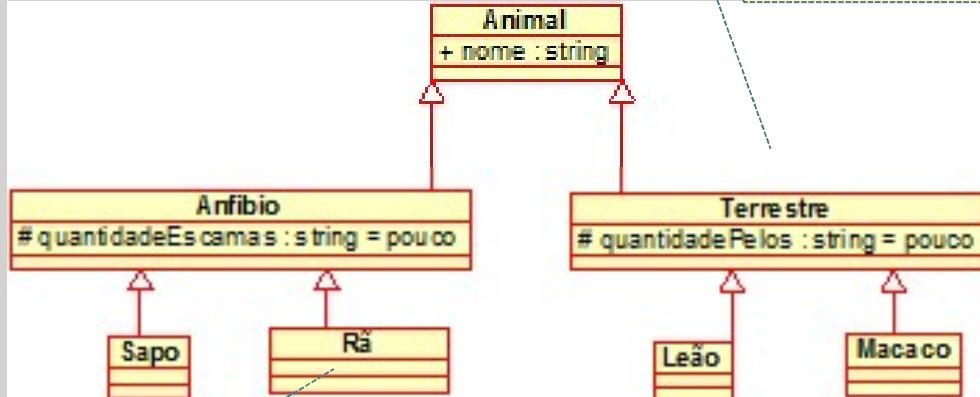
Modificador da Classe A	Classe	Package	Subclasse	Fora do Pacote
public	●	●	●	●
protected	●	●	●	●
Sem modificador	●	●	●	●
private	●	●	●	●

Programação Orientada a Objetos - UTFPR Campus Apucarana

Programação Orientada a Objetos

13

- Protected:



Protected: visível dentro do package + subclasses

Protected:
'quantidadePelos' não
é visível aqui.

Protected:
'quantidadeEscamas'
não é visível aqui.

Programação Orientada a Objetos - UTFPR Campus Apucarana

Programação Orientada a Objetos

14

- Protected:

```
package abc;
public class ABC {
    protected int abc;
}
```

```
package abc;
public class GHI {
    //Protected: acessível dentro do package
    new ABC().abc = 10;
    //Protected: ou acessível com método 'get'
}
```

```
package abc;
public class DEF extends ABC{
    System.out.println(abc);
}
```

```
package def;
import abc;
public class DEF extends ABC{
    System.out.println(abc);
}
```

Programação Orientada a Objetos - UTFPR Campus Apucarana

Programação Orientada a Objetos

15

- **Referência ‘this’:**

- É uma palavra reservada que referencia o próprio objeto.
- ‘this’ também é utilizada para acessar os membros da classe (variáveis de instância e métodos)

Programação Orientada a Objetos - UTFPR Campus Apucarana

Programação Orientada a Objetos

16

- **Encadeamento:**

- Utiliza a referência ‘this’ para acessar vários métodos em um única chamada.

```
public class ABC {  
    ...  
    public ABC metodo1( int a){  
        return this;  
    }  
    public ABC metodo2( int b){  
        return this;  
    }  
}
```

```
public class DEF {  
    ...  
    ABC abc = new ABC( );  
    abc.metodo1( 123 ).metodo2( 456 );  
}  
return abc
```

abc.metodo2(456)

Programação Orientada a Objetos - UTFPR Campus Apucarana

Modificador de Acesso Static

17

- **Static**

- Variável de Classe
- Método de Classe
- ✖ Existem independente da criação de objetos.

18

Garbage Collection

Garbage Collection

19

- Garbage Collection

- Retornar memória para o sistema
- Java realiza automaticamente
 - Objetos são marcados para garbage collection se não há referência para eles

- Método Finalizador

- Retorna recursos para o sistema
- Método: java.lang.Object.finalize
- Método não recebe parâmetros
- Método não retorna valor (void)

Programação Orientada a Objetos - UTFPR Campus Apucarana

20



Revisão

Programação Orientada a Objetos - UTFPR Campus Apucarana

Revisão

21

- **Programação Orientada a Objetos:**

- É uma forma de modelar o mundo real
- Atributos: são as propriedades do objeto
- Comportamentos: são as ações que o objeto realiza
- Classe: é a unidade básica de programação em Java
 - Implementa métodos (ações) que o objeto realiza.
 - Define atributos: atributos que o objeto deve possuir.

Exercícios

22

<Ver conteúdo na plataforma de ensino>



Referências

23

- Referências bibliográficas da disciplina.

Programação Orientada a Objetos

1

BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO

PROF. LUCIO AGOSTINHO ROCHA

AULA 7: HERANÇA

2º SEMESTRE 2022

Programação Orientada a Objetos - UTFPR Campus Apucarana

2

Herança

Programação Orientada a Objetos - UTFPR Campus Apucarana

Herança

3

- Herança:

- Reusabilidade de Software
- Novas classes de objetos aproveitam características de classes já existentes.
 - Aproveitamento de Atributos e Métodos
 - Adição de novas funcionalidades
 - Exemplo: Classe Sapo herda da Classe Animal

Herança

4

- Herança:

- Generalização x Especialização
- Subclasse herda da Superclasse
 - Subclasse: adiciona novas variáveis de instância e métodos
- Java não suporta herança múltipla, porém admite múltiplas Interfaces
- Herança: define um relacionamento do tipo “é um”:
 - Ex.: Sapo é um Animal
- Composição: define um relacionamento do tipo “tem um”:
 - Ex.: Empregado tem uma Data de Contratação.

Herança

5

- Herança: relacionamento “é um”:

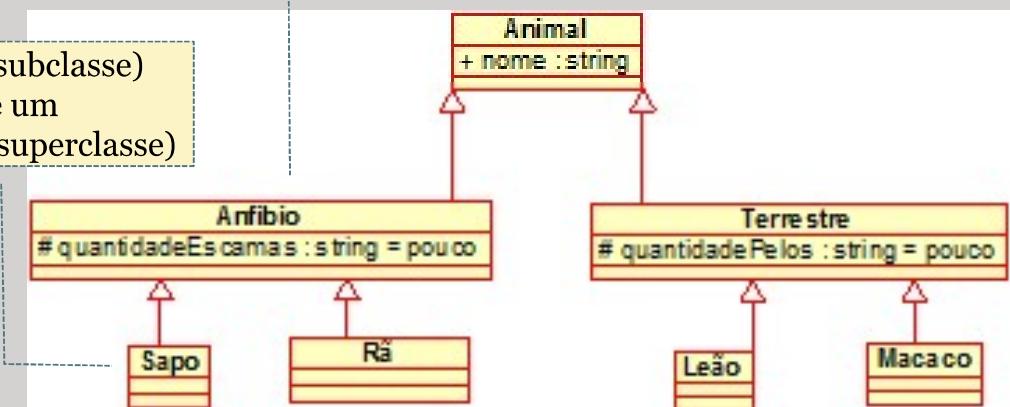
- Objeto “é um” objeto de outra classe
- Herança define uma hierarquia de árvore.
 - ☒ Nota: Herança múltipla é fonte de problemas.
 - ☒ Nota: Hierarquia em árvore remove o problema de loops sintáticos e semânticos.

Herança

6

Anfibio (subclasse)
é um
Animal (superclasse)

Sapo (subclasse)
é um
Animal (superclasse)

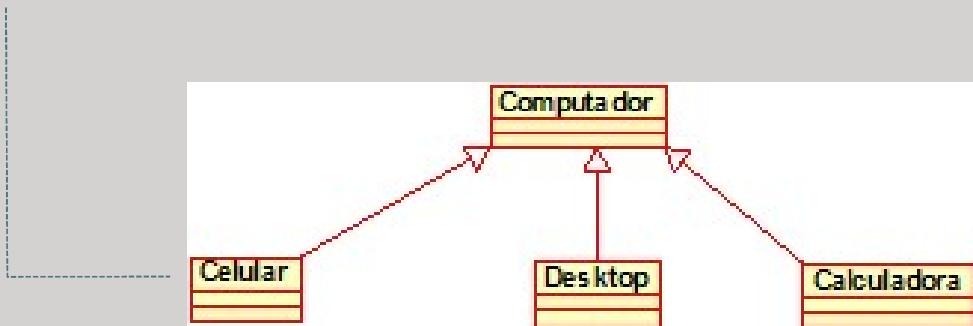


- Figura - Exemplo de Relacionamento de Herança.

Herança

7

Celular (subclasse)
é um
Computador (superclasse)



- Figura - Exemplo de Relacionamento de Herança.

Herança

8

- **Herança:**

- Ocultamento de informação:
 - ✗ Objetos não sabem como outros objetos são implementados.
 - Ex.: é possível usar um computador sem saber como ele calcula os números.
- Abstração:
 - ✗ Pensar em termos de propriedades comuns a vários objetos.
 - Celular (Classe) é um tipo de Computador (Classe)
 - Desktop (Classe) é um tipo de Computador (Classe)
 - Calculadora (Classe) é um tipo de Computador (Classe)

Herança

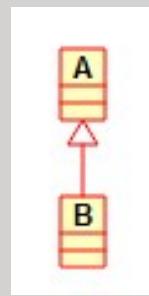
9

- Objeto da Subclasse (mais especializado) pode ser tratado como um objeto da Superclasse (menos especializado)
 - Subclasse contém todos os Membros da Superclasse
 - Ex.: Sapo é um Anfíbio E Sapo é um Animal
- Objeto da Superclasse (menos especializado) não podem ser diretamente tratados como um objeto da Subclasse (mais especializado).
 - Há perda de informação.
 - Ex.:
 - Animal é um Anfíbio X
 - Animal é um Sapo X

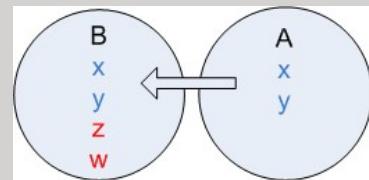
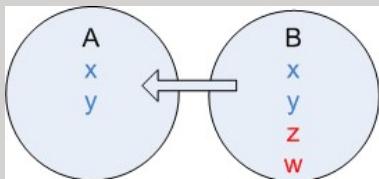
Programação Orientada a Objetos - UTFPR Campus Apucarana

Herança

10



- B é um A



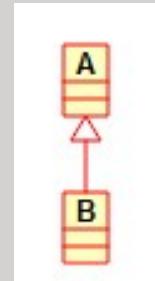
- Faltam campos

Programação Orientada a Objetos - UTFPR Campus Apucarana

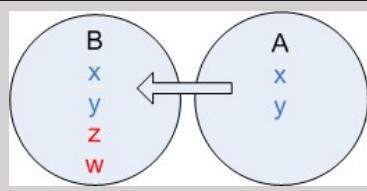
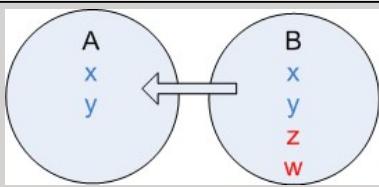
Herança

11

```
public class A {  
    public A() {}  
}  
  
public class B extends A {  
    public B() {}  
    public void metodoB(){  
        return "B";  
    }  
}
```



```
public class C {  
    A a = new A();  
    B b = new B();  
    a = b; // 'a' continua sendo 'A'  
    ((B) a).metodoB();  
  
    b = (B) a; // downcast  
    // (necessário a=b)  
}
```



- Lado direito deve, pelo menos, possuir todos os campos do lado esquerdo.

- Faltam campos
- Lado direito deve, pelo menos, possuir todos os campos do lado esquerdo.

Herança

12

- Construtor:
 - Construtores nunca são herdados.
 - É a primeira instrução invocada pela classe.
 - Há chamadas sucessivas na hierarquia para inicializar as variáveis de instância de cada classe

Herança

13

- Garbage Collector:
 - Método finalize da subclasse deve invocar o método finalize da superclasse com sobrecarga.
 - Caso contrário, apenas a subclasse será indicada para Garbage Collector.
 - Portanto, é uma boa prática de programação sempre incluir um método finalize na superclasse.

Programação Orientada a Objetos - UTFPR Campus Apucarana

14



Revisão

Programação Orientada a Objetos - UTFPR Campus Apucarana

Revisão

15

- **Herança:**

- É um mecanismo nativo de linguagens de Programação Orientadas a Objetos que permite o reaproveitamento de código.
- Herança melhora a legibilidade do código e auxilia na organização da estrutura do projeto.

Exercícios

16

<Ver conteúdo na Plataforma de Ensino>



Referências

17

- Referências bibliográficas da disciplina.

1



**Attribution-NonCommercial-
NoDerivatives 4.0 International
(CC BY-NC-ND 4.0)**



Este trabalho está licenciado com uma Licença [Creative Commons - Atribuição-NãoComercial-SemDerivações 4.0 Internacional](#).

Programação Orientada a Objetos - UTFPR Campus Apucarana



Programação Orientada a Objetos

2

BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO

PROF. LUCIO AGOSTINHO ROCHA

**AULA 10:
INTERFACES E CLASSES ABSTRATAS**

2º SEMESTRE 2022

Programação Orientada a Objetos - UTFPR Campus Apucarana

Interfaces

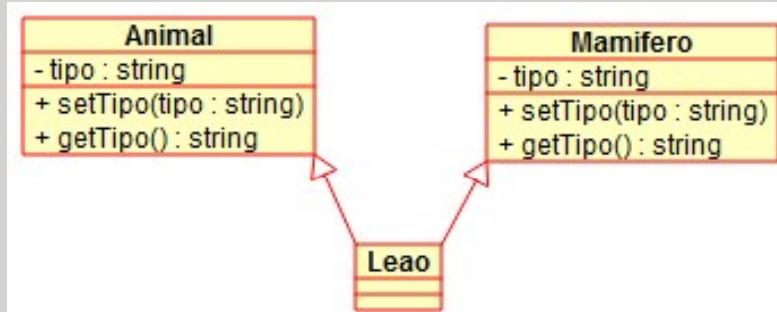
Interfaces

- **Interfaces:**

- Java não suporta herança múltipla, mas admite múltiplas Interfaces.
- Classes implementam Interfaces
 - ☒ A Interface garante que as classes implementem os métodos.
 - ☒ Métodos na interface devem ser declarados ‘public abstract’
- Interfaces permitem que métodos sejam implementados em Interfaces diferentes, e não todos em uma única classe.
- Ao implementar uma interface a classe explicitamente deve definir qual método será implementado.

Interfaces

5



- Figura: Diagrama de Classes com Herança Múltipla.

Interfaces

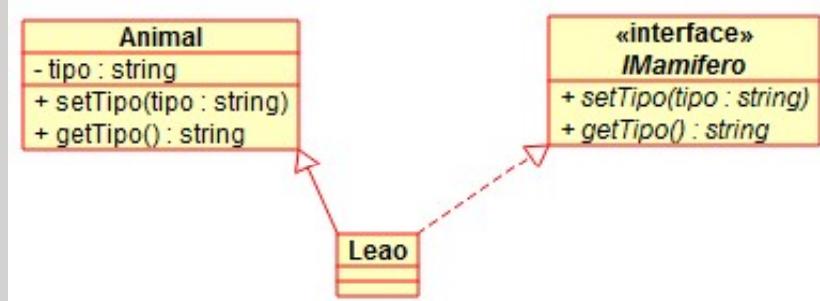
6

```
1 #include <iostream>
2 #include <string>
3
4 #include "Leao.h"
5
6 using namespace std;
7
8 int main() {
9
10    Leao leao;
11    cout << leao.Animal::getTipo() << endl;
12    cout << leao.Mamifero::getTipo() << endl;
13
14
15    return 0;
16 }
17
```

- C++ explicitamente informa a superclasse na chamada do método.

Interfaces

7



- Figura: Diagrama de Classes com Interface.

Programação Orientada a Objetos - UTFPR Campus Apucarana

Interfaces

8

```
1  public interface IMamifero {  
2  
3      public final String tipo="mamifero";  
4  
5      public void setTipo(String tipo);  
6  
7      public String getTipo();  
8  
9  }  
10
```

- Java: 1) Declaração dos métodos da interface.

Programação Orientada a Objetos - UTFPR Campus Apucarana

Interfaces

9

```
1 public class Leao extends Animal implements IMamifero{  
2  
3  
4     public Leao(){  
5  
6     }  
7  
8     public String toString(){  
9         return this.getTipo();  
10    }  
11  
12 }  
13
```

- Java: 2) métodos da interface devem ser implementados ou sobreescritos.

Programação Orientada a Objetos - UTFPR Campus Apucarana

Interfaces

10

```
1 public class Principal {  
2  
3     public static void main(String[] args) {  
4  
5         Leao leao = new Leao();  
6  
7         System.out.println(leao);      //Animal  
8         System.out.println(leao.tipo); //Mamifero  
9  
10    }  
11  
12 }  
13
```

- Java: 3) Declaração e Instanciação do objeto.

Programação Orientada a Objetos - UTFPR Campus Apucarana

Classes Abstratas

Classes Abstratas

- **Classe Abstrata:**

- Permitir que todas as classes herdem umas das outras é um risco de segurança.
- Classe abstrata: fornece uma superclasse para a qual outras classes podem herdar.
- Não instanciam objetos.
- Subclasses devem implementar todos os métodos abstratos. Se não, a subclasse se torna abstrata.

- **Classe Concreta:**

- Classes que permitem instanciar objetos.
- Fornece modelo para instanciar objetos específicos.
 - Ex.: Circulo, Quadrado, Triangulo, Rosa, Margarida, Samambaia.

Classes Abstratas

13

Estudo de Caso com Classe Abstrata:

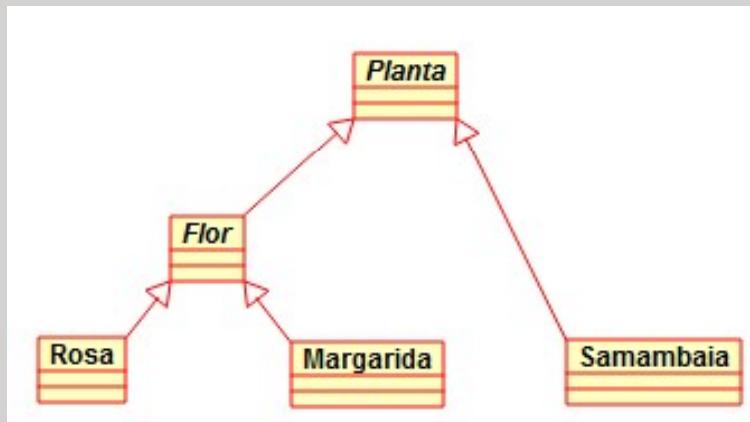


Figura: Classe Planta e Classe Flor são classes Abstract.
Subclasses folha não deveriam ser herdadas (final).

Programação Orientada a Objetos - UTFPR Campus Apucarana

Classes Abstratas

14

Estudo de Caso com Classe Abstrata:



Figura: Classe Empregado é classe Abstract.
Subclasses folha não deveriam ser herdadas (final).

Programação Orientada a Objetos - UTFPR Campus Apucarana

Classes Abstratas

15

- **Modificador de acesso ‘final’:**

- Classes ‘final’ não podem ser herdadas.
- Métodos ‘final’ não podem ser sobre carregados.
- Variáveis de instância ‘final’ são herdadas, mas não podem ser modificadas.

16



Revisão

Revisão

17

- Interfaces
- Classes Abstratas

Exercícios

18

<Ver conteúdo na plataforma de ensino>



Referências

19

- Referências bibliográficas da disciplina.



**Attribution-NonCommercial-NoDerivatives 4.0 International
(CC BY-NC-ND 4.0)**



Este trabalho está licenciado com uma Licença Creative Commons - Atribuição-NãoComercial-SemDerivações 4.0 Internacional.

Programação Orientada a Objetos - UTFPR Campus Apucarana



Programação Orientada a Objetos

BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO

PROF. LUCIO AGOSTINHO ROCHA

AULA 13: CLASSE INTERNA E CLASSE ANÔNIMA

2º SEMESTRE 2022

Programação Orientada a Objetos - UTFPR Campus Apucarana

Classe Interna

Classe Interna

- **Classe Interna:**

- É uma definição de Classe dentro de uma Classe.
- Propósito:
 - Agregar várias Classes, sem necessidade de criar novos arquivos.
 - Proteger a visibilidade de uma Classe de outras Classes.
- Comporta-se como uma parte da Classe.
- Possui todos os membros da Classe convencional:
 - Variáveis de instância e métodos.
- Classes Internas possuem acesso aos membros privados da Classe Externa.
- Nota: Classe Interna comum não pode definir membros 'static'. Para isso, a Classe Interna deve ser 'static'.

Classe Interna

5

- **Classe Interna:**

- Deve ser instanciada com uma referência ao objeto da ClasseExterna.

```
public class ClasseExterna { //1)
    private int var;

    public void executar(){
        ClasseInterna classeInterna = new ClasseInterna();
    }

    private class ClasseInterna {
        //TEM acesso aos membros da ClasseExterna
        var = 111;
    }
}//fim classe Externa
...
ClasseExterna classeExterna = new ClasseExterna();
classeExterna.executar();
```

Classe Interna

6

- **Classe Interna:**

- Deve ser instanciada com uma referência ao objeto da ClasseExterna.

```
public class ClasseExterna { //2)
    private int var;

    ...
    private class ClasseInterna {
        //TEM acesso aos membros da ClasseExterna
        var = 111;
    }
    ...
    ClasseExterna.ClasseInterna classeInterna =
        new ClasseExterna().new ClasseInterna();

}
```

Classe Interna

7

- **Classe Interna ‘static’:**

- Motivação: Classe Interna com uso exclusivo dentro da Classe.
- Comporta-se como uma Classe isolada dentro da Classe Externa.
- É uma Classe interna, mas sem uma referência para a Classe Externa.
- Não possui acesso imediato aos membros da Classe Externa.
- Outras Classes não têm acesso à Classe Interna static (deveria ser declarada ‘private’).

```
public class ClasseExterna{  
    private int var;  
    private static class ClasseInterna {  
        //NÃO TEM acesso aos  
        // membros da ClasseExterna  
        //var = 111;  
    }  
    ...  
    ClasseExterna.ClasseInterna classeInterna =  
        new ClasseExterna().new ClasseInterna();  
}
```

Resumo

8

- **Classe Interna:**

- Gera um arquivo .class separado
- Modificadores de acesso permitidos:
 - public, protected, private, ou acesso de package
- Classe externa é responsável por criar objetos da Classe Interna
- Classe interna também pode ser ‘static’

Classe Anônima

Classe Anônima

- **Classe Anônima:**
 - Uma classe anônima é uma subclasse sem nome de uma superclasse OU
 - É uma classe que implementa uma interface.
 - Em ambos os casos, É uma Classe que não tem um nome.
 - Não há a palavra reservada ‘class’, mas não é só isso: a classe é criada, mas não se tem a referência para criar uma instância da classe anônima.
 - Não tem Construtor:
 - Utiliza o construtor da superclasse
 - Classe interna anônima pode acessar os membros da sua Classe de primeiro nível.

```
public class GUI3 {  
    private int var;  
  
    botao.addActionListener(  
        new ActionListener(){ //chamada da  
            //classe interna anônima para  
            //implementar a interface ActionListener  
  
            //Classe interna anônima TEM acesso aos  
            //membros da classe superior de primeiro nível  
            public void actionPerformed(ActionEvent e){  
                var=1;  
                System.out.println(var);  
            }  
        });//fim classe interna  
  
}//fim classe
```



Revisão

Revisão

13

- Classe Interna
- Classe Anônima

Exercícios

14

<Ver conteúdo na plataforma de ensino>



Referências

15

- Referências bibliográficas da disciplina.



**Attribution-NonCommercial-NoDerivatives 4.0 International
(CC BY-NC-ND 4.0)**



Este trabalho está licenciado com uma Licença [Creative Commons - Atribuição-NãoComercial-SemDerivações 4.0 Internacional](#).

Programação Orientada a Objetos - UTFPR Campus Apucarana



Programação Orientada a Objetos

BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO

PROF. LUCIO AGOSTINHO ROCHA

AULA 15: POLIMORFISMO

2º SEMESTRE 2022

Programação Orientada a Objetos - UTFPR Campus Apucarana

Polimorfismo

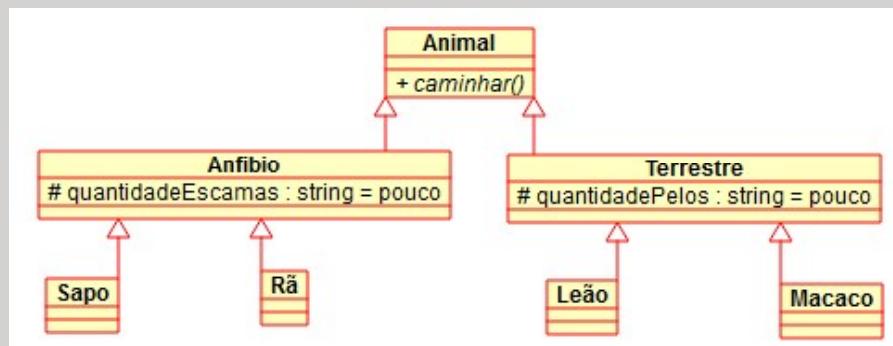
Polimorfismo

- Polimorfismo é um recurso de programação onde objetos que compartilham a mesma superclasse são tratados como objetos dessa superclasse.
- Polimorfismo: é um recurso que permite programar “no geral”, ao invés de programar “no específico”.
- Exemplo:
 - Suponha um programa que implemente a ação de caminhar dos seguintes animais: sapo, rã, leão e macaco.
 - Todos os animais caminham.
 - Todos os animais herdam da Classe Animal

Polimorfismo

5

- Polimorfismo:



- O que estas subclasses têm em comum?

Sapo Rã Leão Macaco

Programação Orientada a Objetos - UTFPR Campus Apucarana

Polimorfismo

6

- Cada classe derivada herda o método ‘caminhar()’ da superclasse Animal.
- Na Classe Principal é mantida uma lista de objetos das subclasses.
- O mesmo método “genérico” é enviado para cada objeto da classe derivada.
- Esse método é sobreescarregado por cada objeto, que o define da sua própria maneira.

Programação Orientada a Objetos - UTFPR Campus Apucarana

Polimorfismo

7

- **Polimorfismo:**

- Auxilia a construir programas extensíveis.
- Programas processam genericamente objetos como superclasses
 - ✖ Novas classes podem ser facilmente inseridas no sistema
 - Novas classes devem fazer parte da hierarquia.
- Exemplo:
 - ✖ Classe Coelho como subclasse da Classe Terrestre.
 - ✖ Classe Coelho herda o método genérico caminhar() e o implementa da sua própria maneira.

Polimorfismo

8

- **Sem Polimorfismo:**

```
public class Principal {  
...  
public static void main(String [ ] args){  
    Sapo sapo = new Sapo();  
    Ra ra = new Ra();  
    Leao leao = new Leao();  
    Macaco macaco = new Macaco();  
  
    sapo.caminhar();  
    ra.caminhar();  
    leao.caminhar();  
    macaco.caminhar();  
}
```

Polimorfismo

9

○ Com Polimorfismo:

```
public class Principal {  
    ...  
    public static void main(String [ ] args){  
        ArrayList<Animal> lista = new ArrayList<>();  
        lista.add( new Sapo() );  
        lista.add( new Ra() );  
        lista.add( new Leao() );  
        lista.add( new Macaco() );  
  
        for( Animal animal : lista )  
            animal.caminhar();  
    }  
}
```

Programação Orientada a Objetos - UTFPR Campus Apucarana

Polimorfismo

10

○ Polimorfismo:

- Um objeto da classe base (superclasse) sempre pode receber um objeto da classe derivada (subclasse).
- Ex.: Sapo é um Animal.

```
Animal animal = new Sapo();  
animal.caminhar();           //Método da classe derivada
```

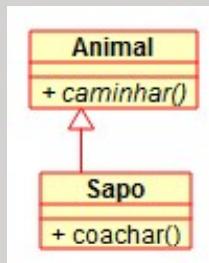
- O inverso não é permitido.
- O objeto ‘animal’ continua sendo da Classe Animal, porém, adquiriu mais membros da classe derivada. Logo, caso se queira acessar membros da subclasse específica, é necessário explicitar a subclasse:

```
((Sapo) animal).coachar(); //Método específico da subclasse
```

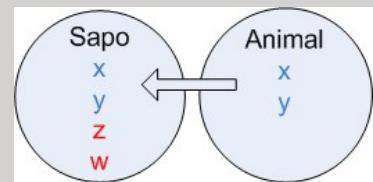
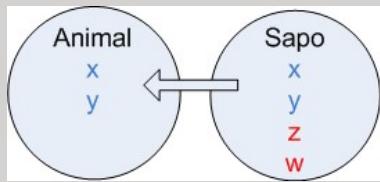
Programação Orientada a Objetos - UTFPR Campus Apucarana

Polimorfismo

11



- Sapo é um Animal



- Faltam campos

Polimorfismo

12

```
public class Principal {
    public static void main(String [] args){
        Animal animal = new Sapo();
        animal.caminhar(); //metodo generico

        animal = new Leao(); //mudou de forma
        animal.caminhar(); //metodo generico
    }
}
```

Polimorfismo

13

- Outro exemplo de Polimorfismo:

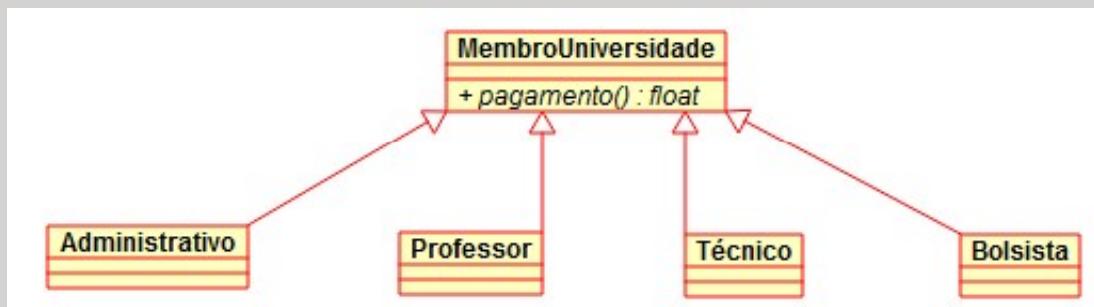
- Um programa de planilha de pagamentos dos membros de uma universidade.
- Os membros da universidade são:
 - ✖ Administrativo, Professor, Técnicos, Bolsistas.
- Todos os membros herdam da Classe MembroUniversidade.
- Todos os membros recebem um pagamento.

Programação Orientada a Objetos - UTFPR Campus Apucarana

Polimorfismo

14

- Polimorfismo:



- O que estas subclasses têm em comum?

Administrativo

Professor

Técnico

Bolsista

Programação Orientada a Objetos - UTFPR Campus Apucarana

Polimorfismo

15

○ Com Polimorfismo:

```
public class Principal {  
...  
    public static void main(String [ ] args){  
        ArrayList<MembroUniversidade> lista = new ArrayList<>();  
        lista.add( new Administrativo() );  
        lista.add( new Professor() );  
        lista.add( new Tecnico() );  
        lista.add( new Bolsista() );  
  
        for( MembroUniversidade membro : lista )  
            membro.pagamento();  
    }  
}
```

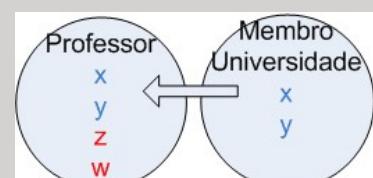
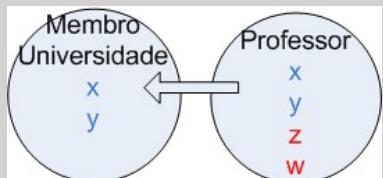
Programação Orientada a Objetos - UTFPR Campus Apucarana

Polimorfismo

16



- Professor é um MembroUniversidade



- Faltam campos

Programação Orientada a Objetos - UTFPR Campus Apucarana

Polimorfismo

17

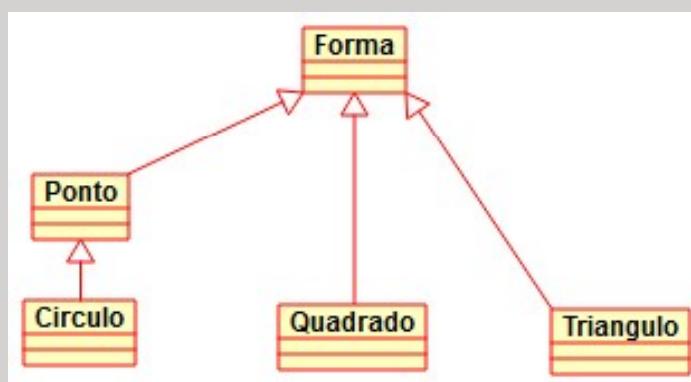
```
public class Principal {  
  
    public static void main(String [] args){  
        MembroUniversidade membro = new Professor();  
        membro.pagamento(); //metodo generico  
  
        membro = new Bolsista(); //mudou de forma  
        membro.pagamento(); //metodo generico  
    }  
}
```

Programação Orientada a Objetos - UTFPR Campus Apucarana

Polimorfismo

18

- Outro exemplo de Polimorfismo:



Programação Orientada a Objetos - UTFPR Campus Apucarana

Polimorfismo

19

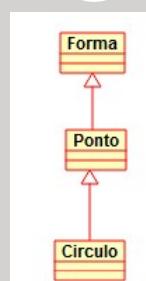
○ Com Polimorfismo:

```
public class Principal {  
...  
    public static void main(String [ ] args){  
        ArrayList<Forma> lista = new ArrayList<>();  
        lista.add( new Circulo() );  
        lista.add( new Triangulo() );  
        lista.add( new Quadrado() );  
        lista.add( new Ponto() );  
  
        for( Forma forma : lista )  
            forma.imprimir();  
    }  
}
```

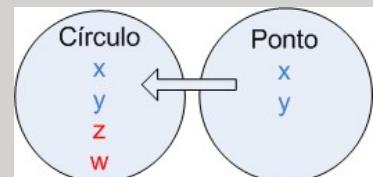
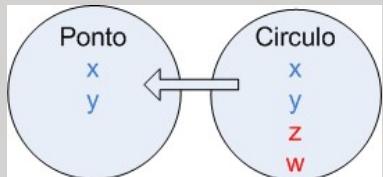
Programação Orientada a Objetos - UTFPR Campus Apucarana

Polimorfismo

20



- Círculo é um Ponto.
- Círculo é uma Forma.



- Faltam campos

Programação Orientada a Objetos - UTFPR Campus Apucarana

Polimorfismo

21

```
public class Principal {  
  
    public static void main(String [] args){  
        Forma forma = new Circulo();  
        forma.imprimir();           //metodo generico  
  
        forma = new Triangulo();    //mudou de forma  
        forma.imprimir();          //metodo generico  
    }  
  
}
```

Programação Orientada a Objetos - UTFPR Campus Apucarana

22



Revisão

Programação Orientada a Objetos - UTFPR Campus Apucarana

Revisão

23

- Polimorfismo
- Interface
- Classe Abstrata

Revisão

24

- Ligação dinâmica:
 - Implementa processamento polimórfico de objetos.
 - Utiliza a referência da Superclasse para o objeto da Subclasse.
 - Os métodos são os mesmos, mas cada objeto de uma Subclasse implementa o seu (sobrecarga).

Revisão

25

- Alternativas ao polimorfismo: switch

- Tratar cada novo objeto em uma estrutura switch
- Problemas:
 - ✖ Deixar a critério exclusivo do programador a validação dos Membros da Classe
 - ✖ Adicionar e remover novas classes.

Exercícios

26

<Ver conteúdo na plataforma de ensino>



Referências

27

- Referências bibliográficas da disciplina.

Programação Orientada a Objetos

1

BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO

PROF. LUCIO AGOSTINHO ROCHA

TRATAMENTO DE EXCEÇÕES

2º. SEMESTRE 2022

Programação Orientada a Objetos - UTFPR Campus Apucarana

2

Tratamento de Exceções

Programação Orientada a Objetos - UTFPR Campus Apucarana

Tratamento de Exceções

3

- **Exceção:**

- É a ocorrência de um erro infrequente durante a execução do programa:
 - Ex.: conversão de tipos, erros aritméticos (divisão por zero), tentativa de acesso fora dos limites de um vetor, acesso a classes inexistentes, e outras exceções.
- Tratamento de exceções:
 - Evita que o programa seja interrompido abruptamente em um estado inválido.
 - O programa trata a exceção e continua a executar como se nada tivesse acontecido, ou encerra o programa.

Tratamento de Exceções

4

- **Quando utilizar o Tratamento de Exceções:**

- Em regiões do código onde a exceção poderá existir com maior frequência;
- Em muitos casos, a própria linguagem Java exige a inserção do tratamento de exceção (Ex.: arquivos).
- Padronização

Tratamento de Exceções

5

- O método que encontra um erro durante a execução dispara (throw) uma exceção.
 - O tratador da exceção processa especificamente esse erro.
- try:
 - Bloco que inclui o trecho de código que potencialmente gerará um erro.
- catch:
 - Bloco que captura o erro e faz o tratamento da exceção.
- finally:
 - Bloco que é sempre executado, com ou sem o erro.

Tratamento de Exceções

6

- Estrutura do bloco try-catch:

```
try {  
    ...  
}  
catch ( TipoExceção1 exceção1 ) {  
    ...  
}  
catch ( TipoExceção2 exceção2 ) {  
    ...  
}  
finally {  
    ...  
}
```

Tratamento de Exceções

7

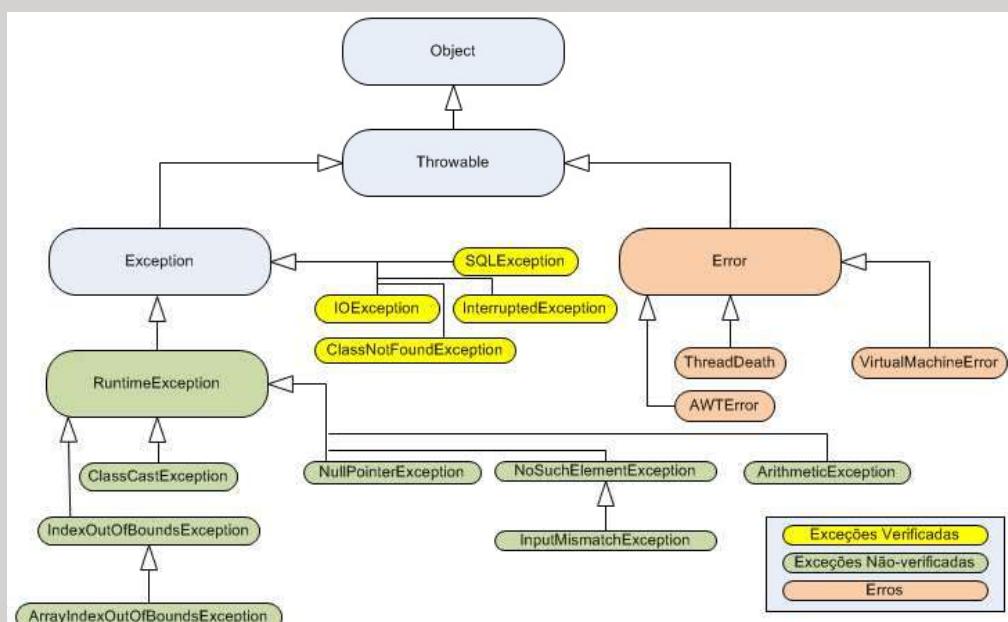
- **throws:**
 - É uma declaração utilizada na declaração do método para indicar que a execução do método poderá disparar uma exceção.
 - Atua em qualquer classe derivada de 'Throwable'
 - ✖ **Throwable possui as subclasses:**
 - Exception: erros que podem ser capturados e tratados.
 - Error: erros que não deveriam ser capturados.
- Programa termina se há um erro durante a execução e não houver um tratamento da exceção.
- **throw:**
 - Indica que o método dispara uma exceção.
- Um único bloco 'catch' pode capturar múltiplas exceções.
- Apenas um catch é utilizado caso uma exceção seja capturada.

Programação Orientada a Objetos - UTFPR Campus Apucarana

Tratamento de Exceções

8

- Hierarquia da Classe Throwable (algumas das principais Classes):



Programação Orientada a Objetos - UTFPR Campus Apucarana

Tratamento de Exceções

9

- **Exceções Verificadas:**
 - Todas as subclasses derivadas diretamente da Classe Exception são verificadas.
 - Tratáveis na compilação.
 - É exigido o tratamento na escrita do código pelo compilador.
 - Exceção deve ser capturada (catch) ou disparada (throw)
 - Exemplos: exceções personalizadas, abertura de arquivos, threads, acesso a base de dados, classes não encontradas, sockets, etc.
- **Exceções Não-verificadas:**
 - Subclasses derivadas da Classe RuntimeException.
 - Tratáveis em tempo de execução.
 - Não é exigido o tratamento na escrita do código pelo compilador.

Programação Orientada a Objetos - UTFPR Campus Apucarana

Tratamento de Exceções

10

- **Exceções personalizadas:**
 - Estendem a Classe Exception ou uma Classe derivada dela.
 - Tornam o detalhamento do erro mais consistente para o usuário.
- **4 (quatro) construtores:**
 - Construtor sem argumentos: mensagem de erro padrão.
 - Construtor com uma String: mensagem enviada para a superclasse.
 - Construtor com uma String e uma Throwable (para encadear exceções)
 - Construtor com uma Throwable: enviada para a superclasse.

Programação Orientada a Objetos - UTFPR Campus Apucarana



Revisão

Revisão

- Tratamento de Exceções

Exercícios

13

<Ver conteúdo na plataforma de ensino>



Referências

14

- Referências bibliográficas da disciplina.

Programação Orientada a Objetos

1

BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO

PROF. LUCIO AGOSTINHO ROCHA

TRATAMENTO DE EXCEÇÕES

2º. SEMESTRE 2022

Programação Orientada a Objetos - UTFPR Campus Apucarana

2

Tratamento de Exceções

Programação Orientada a Objetos - UTFPR Campus Apucarana

Tratamento de Exceções

3

- **Exceção:**

- É a ocorrência de um erro infrequente durante a execução do programa:
 - Ex.: conversão de tipos, erros aritméticos (divisão por zero), tentativa de acesso fora dos limites de um vetor, acesso a classes inexistentes, e outras exceções.
- Tratamento de exceções:
 - Evita que o programa seja interrompido abruptamente em um estado inválido.
 - O programa trata a exceção e continua a executar como se nada tivesse acontecido, ou encerra o programa.

Tratamento de Exceções

4

- **Quando utilizar o Tratamento de Exceções:**

- Em regiões do código onde a exceção poderá existir com maior frequência;
- Em muitos casos, a própria linguagem Java exige a inserção do tratamento de exceção (Ex.: arquivos).
- Padronização

Tratamento de Exceções

5

- O método que encontra um erro durante a execução dispara (throw) uma exceção.
 - O tratador da exceção processa especificamente esse erro.
- try:
 - Bloco que inclui o trecho de código que potencialmente gerará um erro.
- catch:
 - Bloco que captura o erro e faz o tratamento da exceção.
- finally:
 - Bloco que é sempre executado, com ou sem o erro.

Tratamento de Exceções

6

- Estrutura do bloco try-catch:

```
try {  
    ...  
}  
catch ( TipoExceção1 exceção1 ) {  
    ...  
}  
catch ( TipoExceção2 exceção2 ) {  
    ...  
}  
finally {  
    ...  
}
```

Tratamento de Exceções

7

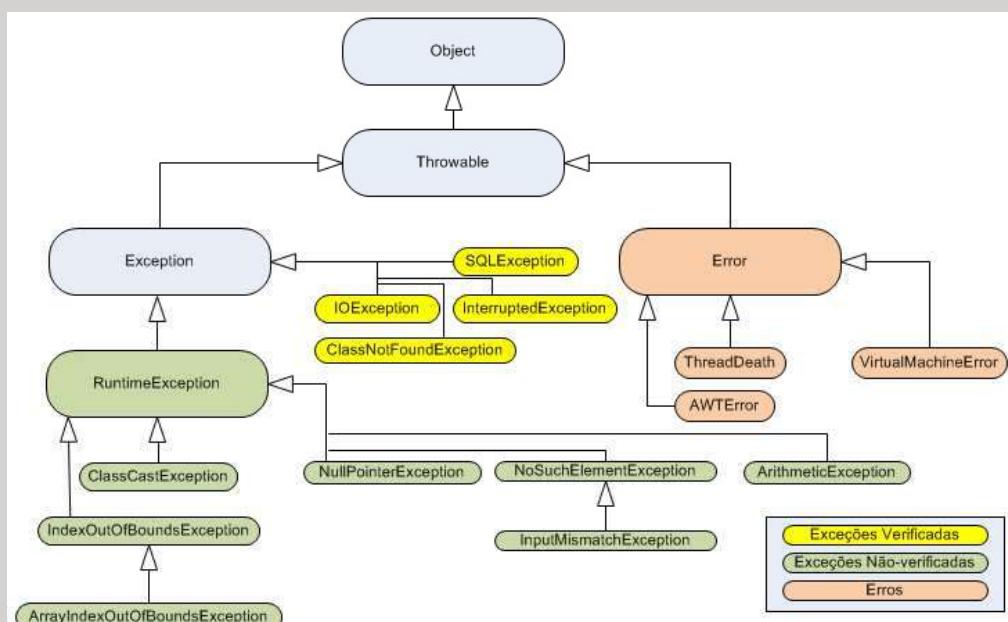
- **throws:**
 - É uma declaração utilizada na declaração do método para indicar que a execução do método poderá disparar uma exceção.
 - Atua em qualquer classe derivada de 'Throwable'
 - ✖ **Throwable possui as subclasses:**
 - Exception: erros que podem ser capturados e tratados.
 - Error: erros que não deveriam ser capturados.
- Programa termina se há um erro durante a execução e não houver um tratamento da exceção.
- **throw:**
 - Indica que o método dispara uma exceção.
- Um único bloco 'catch' pode capturar múltiplas exceções.
- Apenas um catch é utilizado caso uma exceção seja capturada.

Programação Orientada a Objetos - UTFPR Campus Apucarana

Tratamento de Exceções

8

- Hierarquia da Classe Throwable (algumas das principais Classes):



Programação Orientada a Objetos - UTFPR Campus Apucarana

Tratamento de Exceções

9

- **Exceções Verificadas:**
 - Todas as subclasses derivadas diretamente da Classe Exception são verificadas.
 - Tratáveis na compilação.
 - É exigido o tratamento na escrita do código pelo compilador.
 - Exceção deve ser capturada (catch) ou disparada (throw)
 - Exemplos: exceções personalizadas, abertura de arquivos, threads, acesso a base de dados, classes não encontradas, sockets, etc.
- **Exceções Não-verificadas:**
 - Subclasses derivadas da Classe RuntimeException.
 - Tratáveis em tempo de execução.
 - Não é exigido o tratamento na escrita do código pelo compilador.

Programação Orientada a Objetos - UTFPR Campus Apucarana

Tratamento de Exceções

10

- **Exceções personalizadas:**
 - Estendem a Classe Exception ou uma Classe derivada dela.
 - Tornam o detalhamento do erro mais consistente para o usuário.
- **4 (quatro) construtores:**
 - Construtor sem argumentos: mensagem de erro padrão.
 - Construtor com uma String: mensagem enviada para a superclasse.
 - Construtor com uma String e uma Throwable (para encadear exceções)
 - Construtor com uma Throwable: enviada para a superclasse.

Programação Orientada a Objetos - UTFPR Campus Apucarana



Revisão

Revisão

- Tratamento de Exceções

Exercícios

13

<Ver conteúdo na plataforma de ensino>



Referências

14

- Referências bibliográficas da disciplina.

1



**Attribution-NonCommercial-
NoDerivatives 4.0 International
(CC BY-NC-ND 4.0)**



Este trabalho está licenciado com uma Licença [Creative Commons - Atribuição-NãoComercial-SemDerivações 4.0 Internacional](#).

Programação Orientada a Objetos - UTFPR Campus Apucarana



Programação Orientada a Objetos

2

BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO

PROF. LUCIO AGOSTINHO ROCHA

AULA 21: PROJETO DE SOFTWARE

2º SEMESTRE 2022

Programação Orientada a Objetos - UTFPR Campus Apucarana

Unified Modeling Language

Programação Orientada a Objetos - UTFPR Campus Apucarana

Unified Modeling Language

- **UML (Unified Modeling Language):**
 - É uma linguagem de modelagem padronizada e formal para descrever sistemas orientados a objetos (Priestley 2000, Larman 2002).
 - UML fornece modelos e notações formais para a documentação e apresentação dos relacionamentos entre as partes do sistema.
 - Diagramas UML são inter-relacionados e identificam etapas do desenvolvimento do projeto com detalhes que são relevantes para a etapa atual do projeto.
 - Projetos bem elaborados mantêm diagramas padronizados das principais partes do sistema para documentação, consulta, modificação, entendimento e reuso do projeto.

Programação Orientada a Objetos - UTFPR Campus Apucarana

Diagramas UML

Diagramas UML

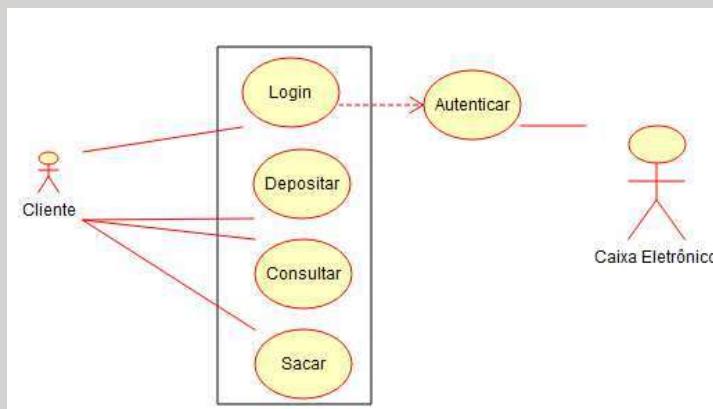
- **Diagramas UML (Unified Modeling Language):**
 - São representações gráficas formais para documentar, representar e apresentar o relacionamento entre as partes do sistema sem o uso de linguagens de programação.
 - Diagramas UML são inter-relacionados e identificam etapas do desenvolvimento do projeto com detalhes que são relevantes para a etapa atual do desenvolvimento.
 - Projeto bem elaborados mantêm diagramas padronizados das principais partes do sistema para consulta, modificação e entendimento do projeto.

Diagramas UML

7

- **Diagrama UML de Casos de Uso**

- Mostra uma visão-geral do sistema, com foco nas funcionalidades.
- Mostra os atores (usuários/entidades do sistema)
- Mostra os casos de uso (funcionalidades)
- Mostra os relacionamentos entre os casos de uso e os atores.



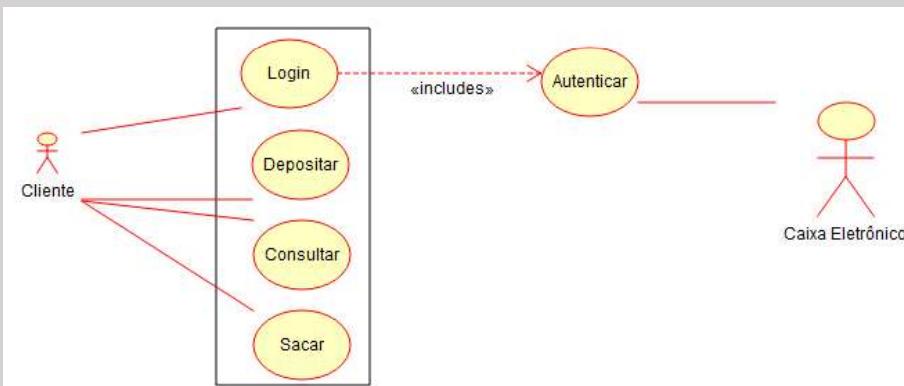
Programação Orientada a Objetos - UTFPR Campus Apucarana

Diagramas UML

8

- **Diagrama UML de Casos de Uso**

- **Include:** É uma associação que aproveita a modelagem do caso de uso anterior. Todas as vezes que Login for realizado, a ação Autenticar também será realizada.



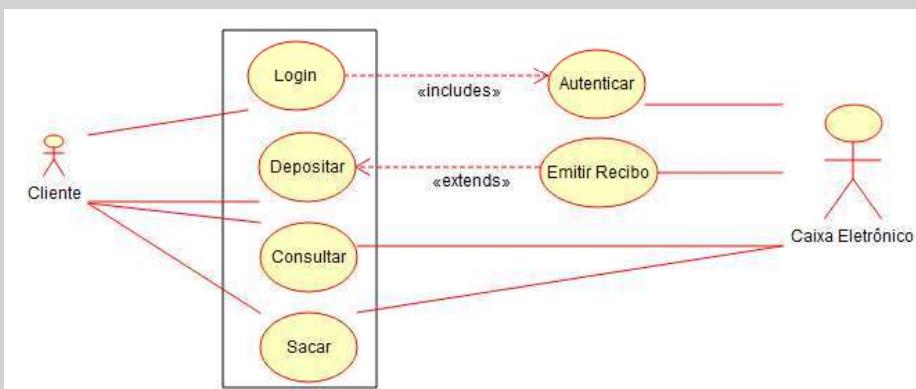
Programação Orientada a Objetos - UTFPR Campus Apucarana

Diagramas UML

9

- Diagrama UML de Casos de Uso

- Extends: Toda extensão está associada a uma condição. A ação Emitir Recibo só é realizada SE o Cliente solicitar, após a ação Depositar.



Programação Orientada a Objetos - UTFPR Campus Apucarana

10

Diagrama de Classes

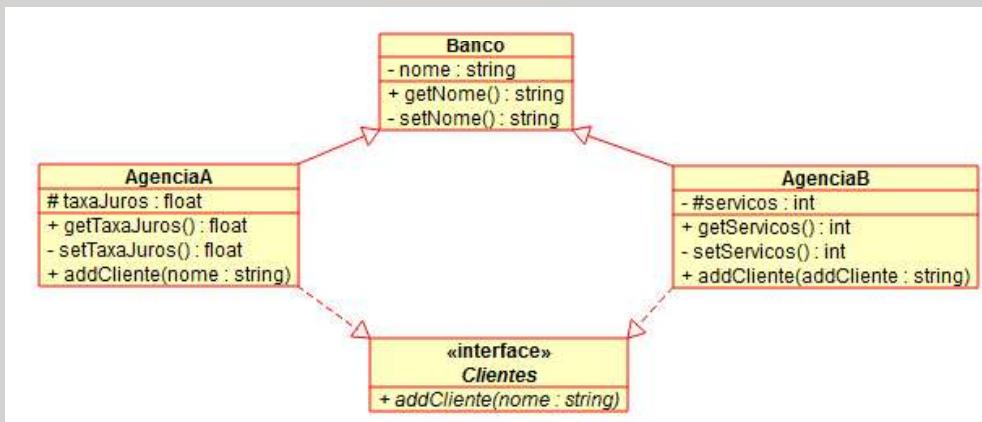
Programação Orientada a Objetos - UTFPR Campus Apucarana

Diagramas UML

11

• Diagrama UML de Classes

- Mostra as classes e os relacionamentos entre as classes.
- Cada classe possui variáveis privadas (-), públicas (+) e protected(#)
- Cada classe possui métodos privados (-), públicos (+) e protected(#)



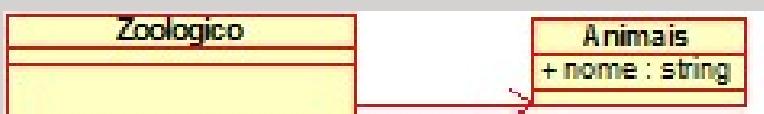
Programação Orientada a Objetos - UTFPR Campus Apucarana

Notações em UML

12

• Associação

- Relação entre duas Classes



```
public class Zoologico {  
    ...  
    Animais animal = new Animais();  
    ...  
}
```

Programação Orientada a Objetos - UTFPR Campus Apucarana

Notações em UML

13

• Agregação

- É uma Associação com mais informações.
- A Classe filha existe independente da Classe pai.
Ex.: Animais existem independente do Zoológico.



```
public class Zoologico {  
    ...  
    Animais animal = new Animais();  
    ...  
}
```

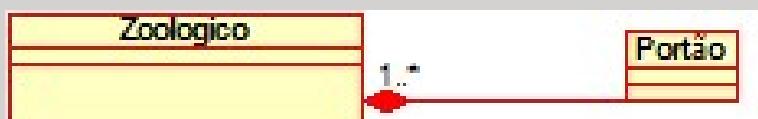
Programação Orientada a Objetos - UTFPR Campus Apucarana

Notações em UML

14

• Composição

- É uma Associação com mais informações.
- Classe filha não existe sem a Classe pai.
Ex.: Portão não existe sem o Zoológico.



```
public class Zoologico {  
    ...  
    Portão portaoPrincipal = new Portão();  
    ...  
}
```

Programação Orientada a Objetos - UTFPR Campus Apucarana

Notações em UML

15

• Dependência

- É uma relação mais fraca que a Associação.
- Indica que uma Classe apenas interage com a outra através de métodos. Classes com pouca relação semântica entre si.



```
public class Zoologico {  
    ...  
    Ingresso.consultaNumero();  
    public consulta (Ingresso p){  
        ...  
    }  
    ...  
}
```

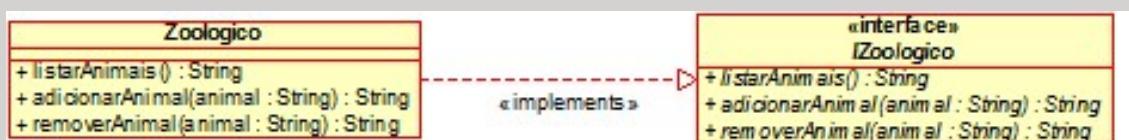
Programação Orientada a Objetos - UTFPR Campus Apucarana

Notações em UML

16

• Implementação

- Indica a implementação de uma interface.



```
public class Zoologico implements IZoologico {  
    ...  
    public String listarAnimais( ){...}  
    public String adicionarAnimal(String animal){...}  
    public String removerAnimal(String animal){...}  
    ...  
}
```

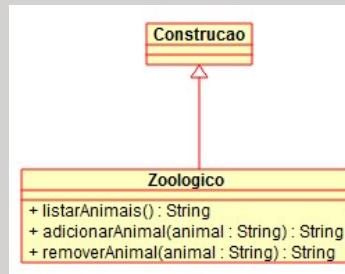
Programação Orientada a Objetos - UTFPR Campus Apucarana

Notações em UML

17

• Generalização

- Utiliza herança para indicar que uma classe (subclasse) deriva de outra (superclasse).



```
public class Zoologico extends Construcao {  
    ...  
}
```

Programação Orientada a Objetos - UTFPR Campus Apucarana

Exercícios

18

<Ver conteúdo na plataforma de ensino>



Programação Orientada a Objetos - UTFPR Campus Apucarana

Referências

19

- Referências bibliográficas da disciplina.
- JACOBSEN, I; BOOCHE, G.; RUMBAUGH, J. *The Unified Software Development Process*. Addison-Wesley, 1999
- LARMAN, C. *Applying UML and Patterns*. Prentice Hall, 2002
- RUMIANCEV, P. UML Class Diagram Arrows Guide. Disponível em: <https://medium.com/the-innovation/uml-class-diagram-arrows-guide-37e4b1bb11e>. Acessado em Maio de 2021.
- (Gráficos de referência para projetos) <https://stackoverflow.com/questions/1874049/explanation-of-the-uml-arrows/23256583>
- UML Class Diagrams Reference. Disponível em <https://docs.microsoft.com/en-us/previous-versions/visualstudio/visual-studio-2015/modeling/uml-class-diagrams-reference?view=vs-2015>. Acessado em Maio de 2021.

Programação Orientada a Objetos

1

BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO

PROF. LUCIO AGOSTINHO ROCHA

AULA 23: PROJETO DE SOFTWARE:
DIAGRAMA DE ESTADO E
DIAGRAMA DE ATIVIDADES

2º SEMESTRE 2022

Programação Orientada a Objetos - UTFPR Campus Apucarana

2

Unified Modeling Language

Programação Orientada a Objetos - UTFPR Campus Apucarana

Unified Modeling Language

3

- **UML (Unified Modeling Language):**

- É uma linguagem de modelagem padronizada e formal para descrever sistemas orientados a objetos (Priestley 2000, Larman 2002).
- UML fornece modelos e notações formais para a documentação e apresentação dos relacionamentos entre as partes do sistema.
- Diagramas UML são inter-relacionados e identificam etapas do desenvolvimento do projeto com detalhes que são relevantes para a etapa atual do projeto.
- Projetos bem elaborados mantêm diagramas padronizados das principais partes do sistema para documentação, consulta, modificação, entendimento e reuso do projeto.

Programação Orientada a Objetos - UTFPR Campus Apucarana

4

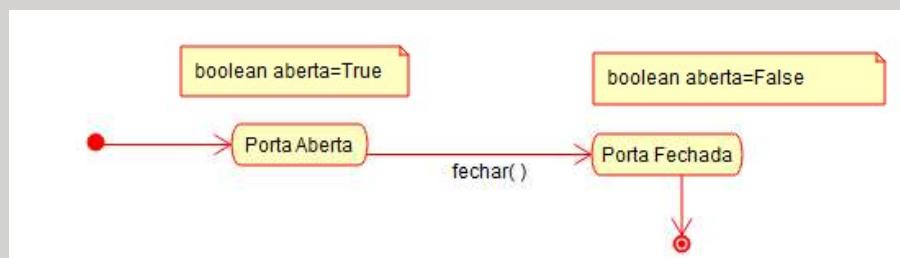
Diagrama de Estado

Programação Orientada a Objetos - UTFPR Campus Apucarana

Diagramas UML

5

- Diagrama de Estados



- Figura: Diagrama de Estados: Objeto Porta.

Programação Orientada a Objetos - UTFPR Campus Apucarana

Diagramas UML

6

- Diagrama de Estado

- Estado: é a condição (atributos) do objeto em um dado momento.
- Diagrama mostra como o estado do objeto muda em resposta a eventos.
- Diagrama mostra as condições nas quais o estado do objeto pode mudar.
- Notações:
 - Estados são representados por retângulos com bordas arredondadas.
 - Exemplo: “Conta com Saldo” e “Conta sem Saldo”
 - Círculo preenchido indica o estado inicial.
 - Areias orientadas indicam a transição de estados, i.e., mudança de estado.
 - Objetos mudam de estado em resposta a mensagens (eventos):
 - Exemplo: “deposito” e “saque”

Programação Orientada a Objetos - UTFPR Campus Apucarana

Diagramas UML

7



Figura: Diagrama UML de Estado (objeto ‘Conta’).

Programação Orientada a Objetos - UTFPR Campus Apucarana

8

Diagrama de Atividades

Programação Orientada a Objetos - UTFPR Campus Apucarana

Diagramas UML

9

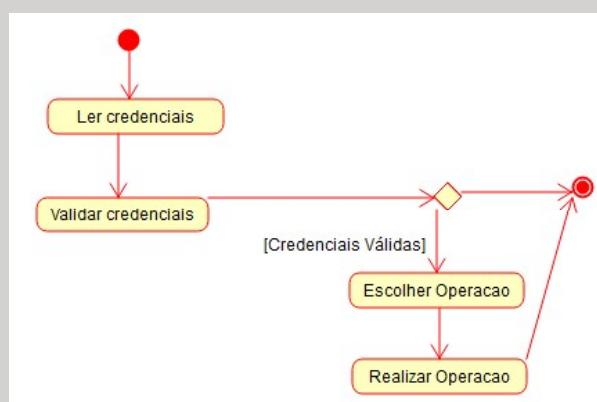
• Diagrama de Atividades

- Fluxo de execução (workflow) comportamentos (métodos) do objeto durante a execução do programa.
- Modela as ações (comportamentos/métodos) que o objeto realizará.
- Notações:
 - Atividades são representadas por retângulos com bordas arredondadas.
 - Círculo preenchido indica a ação inicial.
 - Arestas orientadas indicam a transição de ação, i.e., mudança de ação.
 - Losangos indicam uma condição:
 - Condição deve possuir uma descrição booleana.

Programação Orientada a Objetos - UTFPR Campus Apucarana

Diagramas UML

10



- Figura: Diagrama UML de Atividades (objeto 'Conta').

Programação Orientada a Objetos - UTFPR Campus Apucarana



Revisão

Revisão

- UML é uma linguagem de modelagem padronizada e formal para descrever sistemas orientados a objetos.
- Diagramas UML são representações gráficas formais para a representação, documentação e o entendimento do relacionamento entre as partes do código.
- Leitura adicional recomendada: <ver referências>

Exercícios

13

<Ver conteúdo na plataforma de ensino>



Referências

14

- Referências bibliográficas da disciplina.
- JACOBSEN, I.; BOOCHE, G.; RUMBAUGH, J. *The Unified Software Development Process*. Addison-Wesley, 1999
- LARMAN, C. *Applying UML and Patterns*. Prentice Hall, 2002
- RUMIANCEV, P. UML Class Diagram Arrows Guide. Disponível em: <https://medium.com/the-innovation/uml-class-diagram-arrows-guide-37e4b1bb11e>. Acessado em Maio de 2021.
- (Gráficos de referência para projetos) <https://stackoverflow.com/questions/1874049/explanation-of-the-uml-arrows/23256583>
- UML Class Diagrams Reference. Disponível em <https://docs.microsoft.com/en-us/previous-versions/visualstudio/visual-studio-2015/modeling/uml-class-diagrams-reference?view=vs-2015>. Acessado em Maio de 2021.



**Attribution-NonCommercial-NoDerivatives 4.0 International
(CC BY-NC-ND 4.0)**



Este trabalho está licenciado com uma Licença [Creative Commons - Atribuição-NãoComercial-SemDerivações 4.0 Internacional](#).

Programação Orientada a Objetos - UTFPR Campus Apucarana



Programação Orientada a Objetos

BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO

PROF. LUCIO AGOSTINHO ROCHA

AULA 24: PROJETO DE SOFTWARE:
DIAGRAMA DE SEQUÊNCIA E
DIAGRAMA DE COLABORAÇÃO

2º SEMESTRE 2022

Programação Orientada a Objetos - UTFPR Campus Apucarana

Unified Modeling Language

Programação Orientada a Objetos - UTFPR Campus Apucarana

Unified Modeling Language

- **UML (Unified Modeling Language):**
 - É uma linguagem de modelagem padronizada e formal para descrever sistemas orientados a objetos (Priestley 2000, Larman 2002).
 - UML fornece modelos e notações formais para a documentação e apresentação dos relacionamentos entre as partes do sistema.
 - Diagramas UML são inter-relacionados e identificam etapas do desenvolvimento do projeto com detalhes que são relevantes para a etapa atual do projeto.
 - Projetos bem elaborados mantêm diagramas padronizados das principais partes do sistema para documentação, consulta, modificação, entendimento e reuso do projeto.

Programação Orientada a Objetos - UTFPR Campus Apucarana

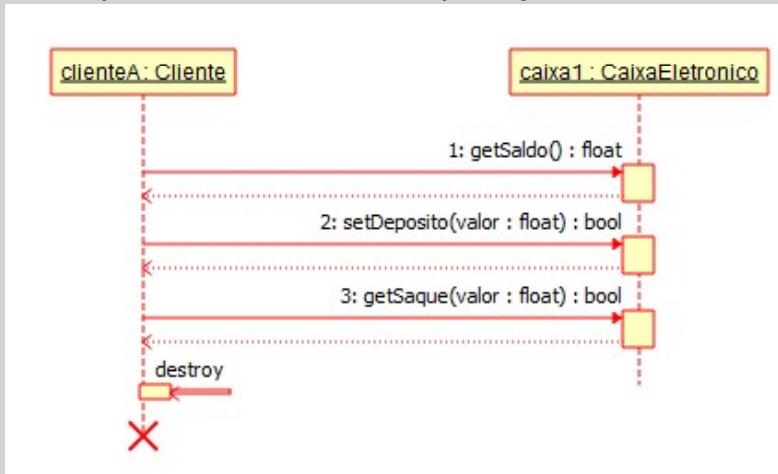
Diagrama de Sequência

Programação Orientada a Objetos - UTFPR Campus Apucarana

Diagramas UML

- **Diagrama UML de Sequência**

- Mostra o fluxo temporal de mensagens entre os objetos.
- Mostra a sequência de mensagens síncronas e assíncronas entre os objetos, e o tempo de validade das operações realizadas.



Programação Orientada a Objetos - UTFPR Campus Apucarana

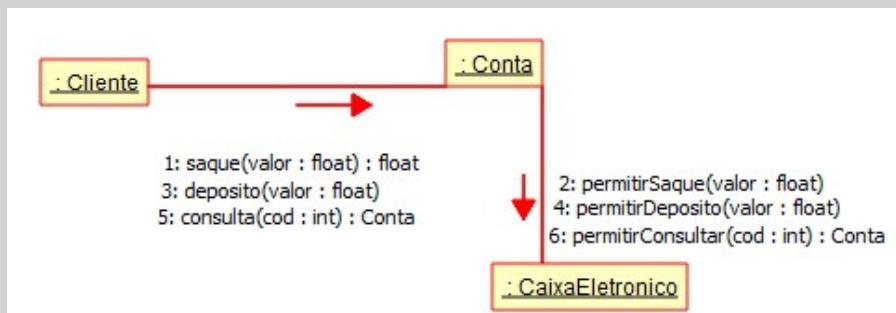
Diagrama de Colaboração

Programação Orientada a Objetos - UTFPR Campus Apucarana

Diagramas UML

- **Diagrama UML de Colaboração**

- Mostra os objetos e seus relacionamentos, foco na troca de mensagens entre objetos através de seus métodos.



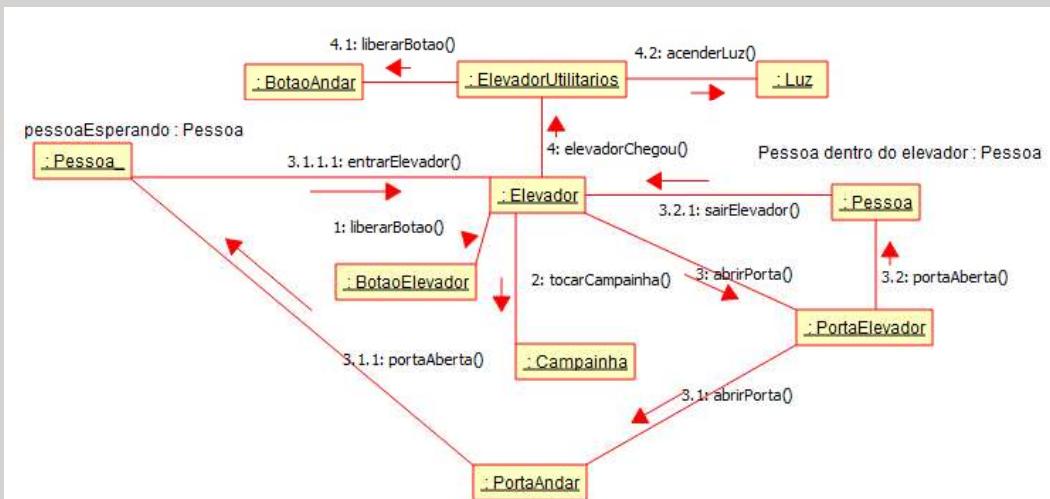
Programação Orientada a Objetos - UTFPR Campus Apucarana

Diagramas UML

9

- Diagrama UML de Colaboração

- A numeração indica a ordem de execução dos métodos.
- Objetos interagem por meio de seus métodos.



Programação Orientada a Objetos - UTFPR Campus Apucarana

10



Revisão

Programação Orientada a Objetos - UTFPR Campus Apucarana

Revisão

11

- UML é uma linguagem de modelagem padronizada e formal para descrever sistemas orientados a objetos.
- Diagramas UML são representações gráficas formais para a representação, documentação e o entendimento do relacionamento entre as partes do código-fonte.
- Diagramas são inter-relacionados e apresentam visões diferentes do mesmo software orientado a objetos.
- Leitura adicional recomendada: <ver referências>

Exercícios

12

<Ver conteúdo na plataforma de ensino>



Referências

13

- Referências bibliográficas da disciplina.
- JACOBSEN, I; BOOCHE, G.; RUMBAUGH, J. *The Unified Software Development Process*. Addison-Wesley, 1999
- LARMAN, C. *Applying UML and Patterns*. Prentice Hall, 2002
- RUMIANCEV, P. UML Class Diagram Arrows Guide. Disponível em: <https://medium.com/the-innovation/uml-class-diagram-arrows-guide-37e4b1bb11e>. Acessado em Maio de 2021.
- (Gráficos de referência para projetos) <https://stackoverflow.com/questions/1874049/explanation-of-the-uml-arrows/23256583>
- UML Class Diagrams Reference. Disponível em <https://docs.microsoft.com/en-us/previous-versions/visualstudio/visual-studio-2015/modeling/uml-class-diagrams-reference?view=vs-2015>. Acessado em Maio de 2021.

Programação Orientada a Objetos

1

BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO

PROF. LUCIO AGOSTINHO ROCHA

AULA 28: INTERFACES GRÁFICAS DO USUÁRIO

2º. SEMESTRE 2022

Programação Orientada a Objetos - UTFPR Campus Apucarana

2

Interfaces Gráficas do Usuário

Programação Orientada a Objetos - UTFPR Campus Apucarana

Interfaces Gráficas do Usuário

3

Componente	Descrição
JLabel	Exibe texto e/ou ícones não editáveis
JTextField	Recepção de Entradas do usuário.
JButton	Botão interativo
JCheckBox	Opção que pode ou não ser selecionada.
JRadioButton	Opções mutuamente exclusivas.
JComboBox	Lista drop-down de itens. Apenas um item pode ser selecionado.
JList	Lista dos itens. Múltiplos itens podem ser selecionados.
JFrame	Área principal onde os componentes são inseridos e organizados de acordo com um layout
JPanel	Área dentro de um JFrame. Os componentes podem ser inseridos e organizados de acordo com um novo layout.

Programação Orientada a Objetos - UTFPR Campus Apucarana

Interfaces Gráficas do Usuário

4

-É necessário informar a forma de inicialização da GUI:

```
import javax.swing.JFrame;  
  
public class GUI extends JFrame {  
  
    ...  
    public GUI() {  
        GUI gui = new GUI();  
        gui.setSize(largura,altura);  
        gui.setVisible(true);  
    }  
    ...  
}
```

Programação Orientada a Objetos - UTFPR Campus Apucarana

Interfaces Gráficas do Usuário

5

-É necessário informar a forma de finalização da GUI:

- `System.exit(0);`
- `objetoGUI.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);`
 - `EXIT_ON_CLOSE`: encerra a aplicação.
 - `DISPOSE_ON_CLOSE`: finaliza a janela atual.
 - `DO NOTHING ON CLOSE`: não faz nada (impede o encerramento).

Interfaces Gráficas do Usuário

6

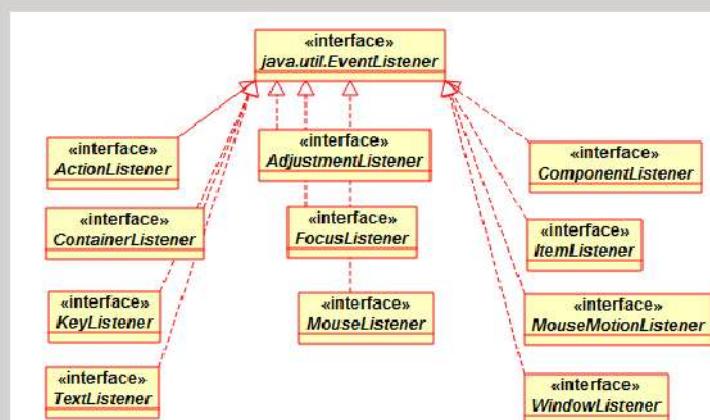


Figura: Interfaces Listeners.

Interfaces Gráficas do Usuário

7

JComponent:

- Aparência personalizável para diferentes Sistemas Operacionais.
- Mnemônicos (teclas de atalho).
- Tips sobre o componente.
- Acessibilidade.
- Internacionalização.

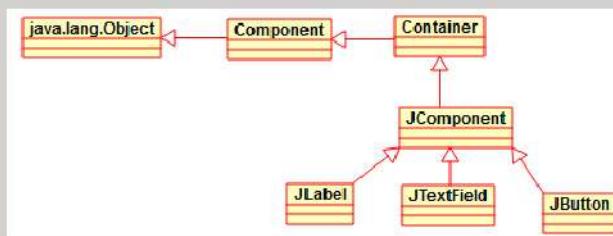


Figura: Hierarquia Parcial de Classes javax.swing.

Programação Orientada a Objetos - UTFPR Campus Apucarana

Interfaces Gráficas do Usuário

8

Gerenciador de Layout	Descrição
FlowLayout	Padrão da javax.swing.JPanel. Components da esquerda para a direita. Um “Container” recebe um “Component” e um índice inteiro da posição do componente.
BorderLayout	Padrão para JFrames. Componentes são organizados em NORTH, SOUTH, EAST, WEST e CENTER.
GridLayout	Componentes organizados em linhas e colunas.

Programação Orientada a Objetos - UTFPR Campus Apucarana



Revisão

Interfaces Gráficas do Usuário

Componente	Descrição
JLabel	Exibe texto e/ou ícones não editáveis
JTextField	Recepção de Entradas do usuário.
JButton	Botão interativo
JCheckBox	Opção que pode ou não ser selecionada.
JRadioButton	Opções mutuamente exclusivas.
JComboBox	Lista drop-down de itens. Apenas um item pode ser selecionado.
JList	Lista dos itens. Múltiplos itens podem ser selecionados.
JFrame	Área principal onde os componentes são inseridos e organizados de acordo com um layout
JPanel	Área dentro de um JFrame. Os componentes podem ser inseridos e organizados de acordo com um novo layout.

Exercícios

11

<Ver conteúdo na plataforma de ensino>



Programação Orientada a Objetos - UTFPR Campus Apucarana

Referências

12

- Referências bibliográficas da disciplina.

Programação Orientada a Objetos - UTFPR Campus Apucarana

Programação Orientada a Objetos

1

BACHARELADO EM ENGENHARIA DE COMPUTAÇÃO

PROF. LUCIO AGOSTINHO ROCHA

AULA 29: INTERFACES GRÁFICAS DO USUÁRIO

MODELO DE EVENTOS

2º SEMESTRE 2022

Programação Orientada a Objetos - UTFPR Campus Apucarana

2

Modelo de Eventos

Programação Orientada a Objetos - UTFPR Campus Apucarana

Modelo de Eventos

3

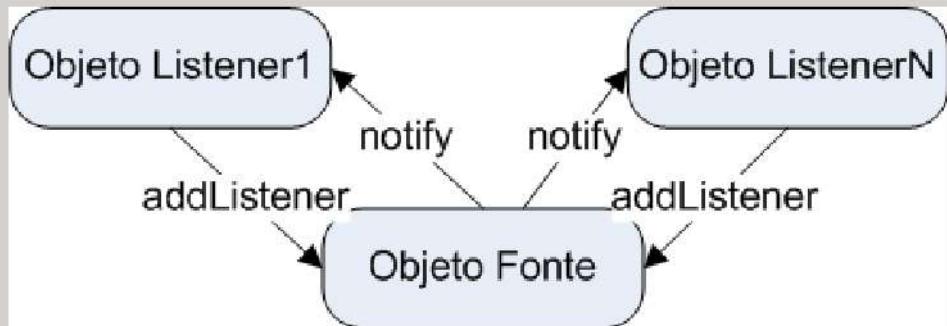


Figura: Modelo de Eventos.

Programação Orientada a Objetos - UTFPR Campus Apucarana

Modelo de Eventos

4

-**Objeto Fonte:**

-JComponent: JLabel, JButton, JTextField, etc.

-**Listener:**

-Um objeto que “escuta” interações do objeto fonte.

-**Mensagem de Notificação:**

-Uma mensagem enviada do objeto fonte para o Listener, indicando que um evento aconteceu.

-**Evento:**

-É uma ação realizada sobre o objeto fonte.

-Cada objeto fonte pode adicionar um ou mais objetos listener próprios para os seus eventos.

Programação Orientada a Objetos - UTFPR Campus Apucarana

Modelo de Eventos

5

-O objeto fonte pode estar associado a vários eventos:

-Deve Implementar múltiplas interfaces Listener.

-O objeto listener deve implementar a interface que “escuta” o evento.

-O objeto listener, que implementa o Listener, é adicionado ao objeto fonte:

objetoFonte.addListener(Listener listener)

Programação Orientada a Objetos - UTFPR Campus Apucarana

Modelo de Eventos

6

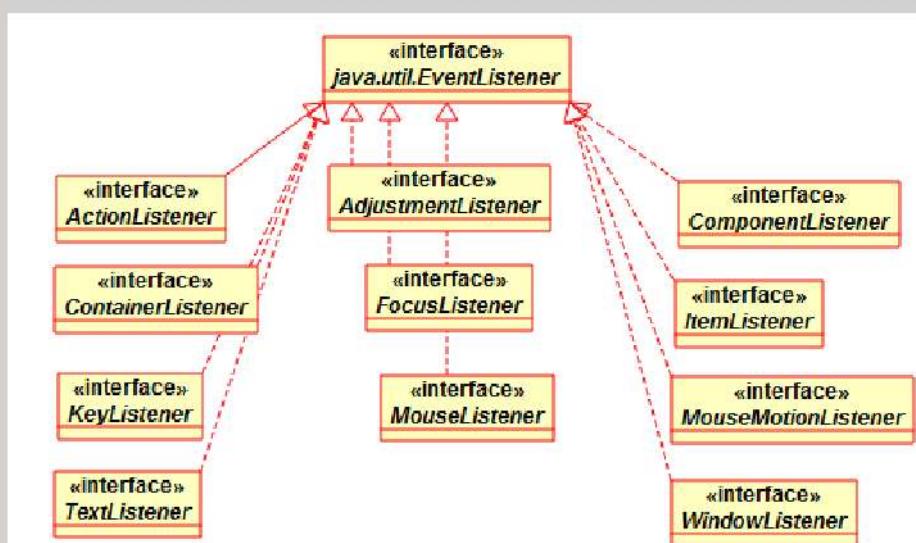


Figura: Interfaces Listeners.

Programação Orientada a Objetos - UTFPR Campus Apucarana

Modelo de Eventos

7

-O objeto listener que deseja “escutar” um JButton deve implementar uma interface ActionListener que deriva de EventListener:

```
public interface ActionListener extends EventListener {  
  
    public void actionPerformed(ActionEvent e);  
  
}
```

- O objeto ‘ActionEvent e’ possui:

- e.getSource(): um ponteiro para o objeto fonte.
- e.getWhen(): o timestamp de quando o evento aconteceu.
- e.getModifiers(): teclas pressionadas juntas (ex.: ALT, CTRL, etc.)
- e.getActionCommand(): a string do objeto que disparou a ação.

Modelo de Eventos

8

```
public class MeuListener implements ActionListener {  
  
    //JButton notifica o objeto Listener  
    public void actionPerformed(ActionEvent evento){  
  
        Toolkit.getDefaultToolkit().beep();  
  
    }  
    ...  
    JButton beep = new JButton ("Beep");  
    beep.addActionListener( new MeuListener() );
```

Modelo de Eventos

9

-Evento:

Botão é clicado.

- O objeto fonte envia para o objeto listener a notificação que o evento aconteceu.

-No caso, o objeto fonte JButton utiliza o método

actionPerformed(ActionEvent e)

para enviar a notificação para o objeto listener.

Programação Orientada a Objetos - UTFPR Campus Apucarana

Modelo de Eventos

10

-Há diferentes formas de adicionar um Listener para um JComponent.

- Método 1:
 - Diretamente na Classe.
 - Utiliza o objeto ‘this’ como Listener.

```
public class BotaoPersonalizado extends JComponent
    implements ActionListener {

    public void actionPerformed(ActionEvent e){
        Toolkit.getDefaultToolkit().beep();
    }
    public BotaoPersonalizado(){
        JButton b = new JButton();
        b.addActionListener(this);
    }
}
```

Programação Orientada a Objetos - UTFPR Campus Apucarana

Modelo de Eventos

11

- Método 2: - Criar uma Classe Interna que implementa o Listener

```
public class BotaoPersonalizado {  
    public BotaoPersonalizado(){  
        JButton b = new JButton();  
        ActionListener listener = new ClasseInterna();  
        b.addActionListener(listener);  
        //OU  
        //b.addActionListener(new ClasseInterna());  
    }  
    public class ClasseInterna implements ActionListener{  
        public void actionPerformed(ActionEvent e){  
            Toolkit.getDefaultToolkit().beep();  
        }  
    }  
}
```

Programação Orientada a Objetos - UTFPR Campus Apucarana

Modelo de Eventos

12

- Método 3:
 - Com uma Classe Interna Anônima.
 - No caso, a classe interna anônima implementa a interface ActionListener.

```
public class BotaoPersonalizado {  
    public BotaoPersonalizado(){  
        JButton b = new JButton();  
        b.addActionListener( new ActionListener(){  
            public void actionPerformed(ActionEvent e){  
                Toolkit.getDefaultToolkit().beep();  
            }  
        });  
    }  
}
```

Programação Orientada a Objetos - UTFPR Campus Apucarana



Revisão

Modelo de Eventos

-Objeto Fonte:

-JComponent: JLabel, JButton, JTextField, etc.

-Listener:

-Um objeto que “escuta” interações do objeto fonte.

-Mensagem de Notificação:

-Uma mensagem enviada do objeto fonte para o Listener, indicando que um evento aconteceu.

-Evento:

-É uma ação realizada sobre o objeto fonte.

-Cada objeto fonte pode adicionar um ou mais objetos listener próprios para os seus eventos.

Modelo de Eventos

15

-O objeto fonte pode estar associado a vários eventos:

-Deve Implementar múltiplas interfaces Listener.

-O objeto listener deve implementar a interface que “escuta” o evento.

-O objeto listener, que implementa o Listener, é adicionado ao objeto fonte:

objetoFonte.addListener(Listener listener)

Programação Orientada a Objetos - UTFPR Campus Apucarana

Exercícios

16

<Ver conteúdo na plataforma de ensino>



Programação Orientada a Objetos - UTFPR Campus Apucarana

Referências

17

- Referências bibliográficas da disciplina.