UNIRIO

Polimorfismo

Aula 9

BSI — 2018.2

Jefferson Elbert Simões

CCET/DIA

18 de setembro de 2018

Previously on TP2...

- Fundamentos: classes, objetos, atributos, métodos
- Referências
- Construtores e destrutores
- Métodos e atributos estáticos
- Encapsulamento e visibilidade
- Tipos de enumeração
- Exceções
- Agregação
- Herança

Sobrecarga e sobreposição

- Até agora, trabalhamos com a noção de "um método, um nome"
  - Métodos possuem um nome exclusivo, que nenhum outro método possui
- Isto não é uma imposição do Java — ou de qualquer outra linguagem
  - É possível criar métodos que compartilhem nomes
  - Na verdade, isto pode ser utilizado como um recurso de boa programação

Sobrecarga

- (em inglês, overload)
- Definição de dois ou mais métodos na mesma classe com o mesmo nome, porém com parâmetros diferentes
- Pode ser aplicada quando:
  - uma ação pode ser feita de múltiplas formas distintas
  - ações equivalentes podem ser feitas em objetos de tipos distintos

Sobrecarga

Exemplo I:

```
class Funcionario {
    private String cargo;
    private double salario;
    public void transfere( String novoCargo ) {
        cargo = novoCargo;
    }
    public void transfere( String novoCargo,
                           double novoSalario ) {
        cargo = novoCargo;
        salario = novoSalario;
    }
}
```

Sobrecarga

Exemplo 2:

```
class Aritmetica {
    // soma dois números inteiros
    public static int soma (int n1, int n2) {
        return n1 + n2;
    }
    // soma dois números reais
    public static double soma (double n1, double n2) {
        return n1 + n2;
    }
}
```

Sobreposição

- (em inglês, override)
- Ocorre quando uma classe derivada "reimplementa" um método herdado de sua classe Base
- Pode ser utilizada quando:
  - A classe derivada possui características adicionais que podem gerar uma implementação melhor (mais eficiente, mais detalhada)...
  - ...Mas a funcionalidade do método permanece a mesma

Sobreposição

Exemplo:

```
class Pessoa {
    protected String nome;
    public String escreveBiografia() {
        return "Meu nome é " + nome + ".";
    }
}
class Aluno extends Pessoa {
    private String curso;
    public String escreveBiografia() {
        return "Meu nome é " + nome +
                " e sou aluno de " + curso + ".";
    }
}
```

Sobreposição

É possível reutilizar a implementação da classe base:

```
class Pessoa {
    private String nome;
    public String escreveBiografia() {
        return "Meu nome é " + nome + ".";
    }
}
class Aluno extends Pessoa {
    private String curso;
    public String escreveBiografia() {
        return super.escreveBiografia() +
                " Eu sou aluno de " + curso + ".";
    }
}
```

Sobrecarga x Sobreposição

- Quando se usa sobrecarga, a "versão" da função a ser chamada pode ser identificada pelo conjunto de argumentos
  - A correspondência entre a chamada e a implementação é feita em tempo de compilação
- Quando se usa sobreposição, é preciso analisar o objeto para determinar a "versão" apropriada da função
  - Esta correspondência é feita quando o objeto é instanciado, em tempo de execução
- Sobrecarga tende a gerar programas mais eficientes do que sobreposição

Polimorfismo

- Quarto pilar de OO
- De maneira abstrata, é a ideia de utilizar uma única interface para lidar com objetos de múltiplos tipos diferentes
  - A mesma função/método/descrição adquire "múltiplas formas" para servir a muitos casos diferentes
- Cada linguagem fornece diferentes "ferramentas" de suporte a polimorfismo

Polimorfismo

Algumas variedades de polimorfismo:

Ad hoc: o código é escrito de forma a suportar múltiplos tipos individuais, cada um com uma implementação específica

- Sobrecarga de métodos e funções

Polimorfismo

Algumas variedades de polimorfismo:

Ad hoc: o código é escrito de forma a suportar múltiplos tipos individuais, cada um com uma implementação específica

Por subtipos: o código é escrito com base em um tipo geral, mas os objetos pertencem a subtipos

- Sobreposição de métodos
- Métodos virtuais
- Classes abstratas

Polimorfismo

Algumas variedades de polimorfismo:

Ad hoc: o código é escrito de forma a suportar múltiplos tipos individuais, cada um com uma implementação específica

Por subtipos: o código é escrito com base em um tipo geral, mas os objetos pertencem a subtipos

Paramétrico o código é escrito sem descrever nenhum tipo específico; também conhecido como programação genérica ou templates

Polimorfismo

- Polimorfismo por subtipos:
  - Código é escrito com base em um tipo geral, mas os objetos pertencem a subtipos (subclasses)

    Sobreposição de métodos: implementação de métodos específicos depende da classe do objeto em questão
  - Uso geral: implementação de uma tarefa geral utilizando métodos para realizar subtarefas
    - A tarefa geral é feita de forma genérica
    - Mas as subtarefas variam de acordo com o objeto
  - Vantagens:
    - Legibilidade (tarefa geral fica fácil de entender)
    - Flexibilidade (não precisa reimplementar tudo para novas classes)

Polimorfismo

Exemplo de polimorfismo por subtipos:

```
class Funcionario {
    protected double salario;
    protected static int NUM_SALARIOS_ANUAIS = 13;
    Funcionario( double meuSalario ) {
        salario = meuSalario; }
    public double calculaSalarioAnual() {
        return NUM_SALARIOS_ANUAIS * salario; }
    public void imprimeDeclaracao() {
        System.out.println( "Meu salário é R$"+
            calculaSalarioAnual() );
    }
}
```

Polimorfismo

Exemplo de polimorfismo por subtipos:

```
class Gerente extends Funcionario {
    private double gratificacao;
    private static int NUM_SALARIOS_ANUAIS = 14;
    Gerente( double meuSalario,
                    double minhaGratificacao ) {
        super(meuSalario);
        gratificacao = minhaGratificacao;
    }
    public double calculaSalarioAnual() {
        return NUM_SALARIOS_ANUAIS *
                (salario + gratificacao);
    }
}
```

Polimorfismo

Exemplo de polimorfismo por subtipos:

```
class Terceirizado extends Funcionario {
    private double adicional;
    private static int NUM_SALARIOS_ANUAIS = 14;
    Terceirizado( double meuSalario,
                  double meuAdicional ) {
        super(meuSalario);
        adicional = meuAdicional;
    }
    public double calculaSalarioAnual() {
        return NUM_SALARIOS_ANUAIS *
               (salario * (1.0 + adicional) );
    }
}
```

Polimorfismo

Exemplo de polimorfismo por subtipos:

```
ArrayList<Funcionario> listaFuncionarios =
    new ArrayList<>();
listaFuncionarios.add( new Funcionario( 1500 ) );
listaFuncionarios.add( new Gerente( 4500, 500 ) );
listaFuncionarios.add( new Terceirizado( 800, 0.3 ) );

for( Funcionario func :  listaFuncionarios )
    func.imprimeDeclaracao();
```

Exercícios

Implemente as seguintes funcionalidades no sistema acadêmico (ex. 5 da aula 7).

- Para uma disciplina ser oferecida, é preciso que uma turma seja criada para essa disciplina;
  - Além da disciplina, a turma também está associada a um determinado período de oferta (ex: 2018.2)
  - Toda turma precisa possuir um professor
  - Alunos podem se inscrever em uma turma, mas não se desinscrever

Exercícios

- Toda turma pode possuir um monitor, que pode ser um aluno ou um professor
- Um professor só pode se tornar monitor de uma turma se não for o responsável pela disciplina
- Um aluno só pode se tornar monitor de uma turma se o professor responsável pela disciplina for seu orientador

Métodos a implementar:

classe Turma: `associarMonitor (Pessoa)`

classe Pessoa e subclasses: `podeSerMonitor (Disciplina)`