UNIRIO

Classes e métodos abstratos / Interfaces

Aula 11

BSI — 2018.2

Jefferson Elbert Simões

CCET/DIA

25 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
- Polimorfismo (+ sobrecarga, sobreposição)

Polimorfismo via sobreposição

- Na aula passada, vimos uma maneira de utilizar sobreposição para realizar polimorfismo
  - Método "genérico" — esqueleto da implementação
  - Métodos "específicos" sobrepostos em classes derivadas — preenchem os detalhes

Polimorfismo via sobreposição

- Na aula passada, vimos uma maneira de utilizar sobreposição para realizar polimorfismo
  - Método "genérico" — esqueleto da implementação
    - `imprimeDeclaracao()`: classe `Funcionario`
  - Métodos "específicos" sobrepostos em classes derivadas — preenchem os detalhes
    - `calculaSalarioAnual()`: implementado de formas diferentes para as classes `Funcionario`, `Gerente` e `Terceirizado`

Polimorfismo via sobreposição

- No nosso exemplo, o método "genérico" era uma método da classe Base, mas ele pode ser um método de qualquer classe

  Exemplo:

  Método "genérico": `associaMonitor()` na classe `Turma`
  Método "específico": `podeSerMonitor()`, implementado nas classes `Aluno` e `Professor`

- Sempre que um método execute sua função baseado apenas em métodos da classe pai, sem saber que as classes filhas sobrepõem estes métodos, isto é polimorfismo!

Polimorfismo via sobreposição

- Problema: sempre faz sentido o método chamado ser implementado na classe Base?
  - Exemplo: em uma empresa, pode não existir funcionário "padrão" com salário "padrão"
    - Todo funcionário pertence a alguma categoria (subclasse), com um cálculo de salário próprio

Métodos abstratos

- a.k.a. métodos puramente virtuais
- Métodos que não possuem implementação
  - Devem ser implementados por classes derivadas
  - Declaração:
    - `abstract double calculaSalarioAnual();`

Métodos abstratos

- Tem algo estranho no ar...

```
class Funcionario {
    double salario;
    abstract double getSalarioAnual();
}

Funcionario fantasma = new Funcionario();
System.out.println( fantasma.getSalarioAnual() );
```

O que acontece?

Métodos abstratos

- Tem algo estranho no ar...

```
class Funcionario {
    double salario;
    abstract double getSalarioAnual();
}

Funcionario fantasma = new Funcionario();
System.out.println( fantasma.getSalarioAnual() );
```

O que deveria acontecer?

Classes abstratas

- Se uma classe possui um método sem implementação, não faz sentido instanciá-la
  - Objeto ficaria "incompleto":
  - Como chamar um método que um objeto possui (porque foi declarado) mas não sabe o que faz (porque não foi definido)?
- Para evitar este caso, podemos declarar uma classe como abstrata
  - Classe que não pode ser instanciada
- Toda classe que possui um método abstrato deve ser ela própria abstrata
  - Caso contrário, erro de compilação

Classes abstratas

- Consertando a classe anterior...

```
abstract class Funcionario {
    double salario;
    abstract double getSalarioAnual();
}
```

~~`Funcionario fantasma = new Funcionario();`~~
~~`System.out.println( fantasma.getSalarioAnual() );`~~

Instanciação de objetos proibida explicitamente

Classes abstratas

- Instanciação de objetos explicitamente proibida
  - Mas ainda podemos instanciar objetos de classes derivadas
- Toda classe derivada deve implementar todos os métodos abstratas da classe Base
  - (ou então, ser também uma classe abstrata)
- É possível ter uma classe abstrata sem métodos abstratos
  - Basta declará-la como abstrata!

Herança

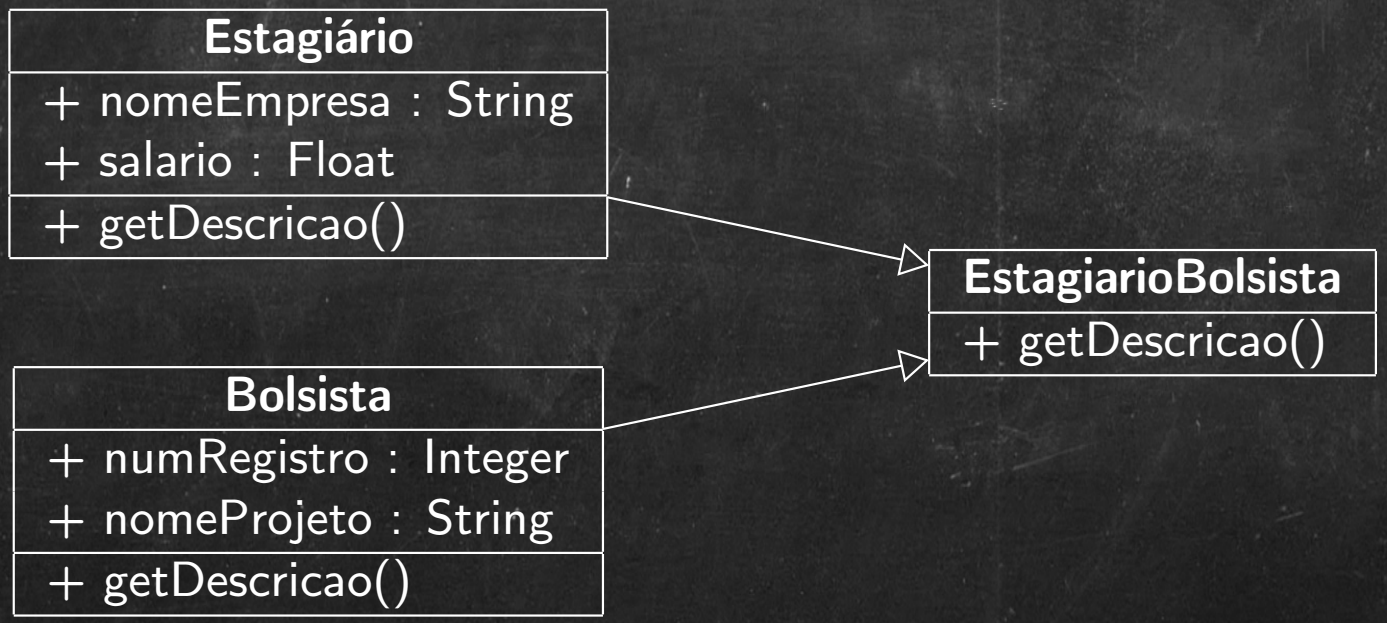
- Relacionamento entre classes
  - Classe derivada herda estruturas de classe Base e as especializa
  - Descreve objetos mais específicos
    - Todo `Funcionario` é também `Pessoa`
- Estrutura de heranças gera "hierarquias" no código
  - Evidente no diagrama de classes

Herança múltipla

- É razoável que classes tenham mais de um "pai"?

Herança múltipla

- É razoável que classes tenham mais de um "pai"?
- Em vários casos, gostaríamos de classes que descrevessem a "interseção" de duas outras



Herança múltipla

- Para permitir herança múltipla, é necessário lidar com alguns problemas técnicos
  - Ambiguidade de heranças
  - Problema do diamante
- Cada linguagem dá uma solução diferente para estas questões
  - Herança virtual (C++)
  - Herança ordenada (Perl, Python)
- Java proíbe herança múltipla
  - Versão "aproximada": interfaces

Interfaces

Uma interface é um tipo (semelhante a uma classe) que deve seguir as seguintes restrições:

- Somente possui atributos constantes (estáticos)
- Métodos não podem possuir implementação[1]
  - são automaticamente abstratos

Interfaces não podem ser instanciadas

- Apenas prescrevem funcionamento de outras classes
- Devem ser implementadas por elas
  - semelhante a uma herança

[1] ou quase: aguarde próximos slides

Interfaces

Exemplo:

```
public interface NumeroComModulo {
    double modulo();
    String formaPadrao();
}
public class Racional implements NumeroComModulo {
    bool positivo;
    int numerador, denominador;
    double modulo() {
        return (numerador + 0.0)/denominador;
    }
    String formaPadrao() {
        String expressao = "";
        if( !positivo )
            expressao += "-";
        expressao += numerador + "/" + denominador;
        return expressao;
    }
}
```

Interfaces

Exemplo:

```
public interface NumeroComModulo {
    double modulo();
    String formaPadrao();
}
public class Complexo implements NumeroComModulo {
    double parteReal, parteImag;
    double modulo() {
        return Math.sqrt( parteReal*parteReal
                        + parteImag*parteImag );
    }
    String formaPadrao() {
        String expressao = "";
        expressao += parteReal;
        if( Math.signum(parteImag) != -1 ) // 1: pos; 0: zero; -1: neg
            expressao += "+";
        expressao += parteImag + "i"
        return expressao;
    }

}
```

Interfaces

Exemplo:

```
public static void imprimeNumero( NumeroComModulo num ) {
    System.out.println( "Número:" + num.formaPadrao() );
    System.out.println( "Módulo:" + num.modulo() ); }
}
```

Interface vs. Classe abstrata

Interfaces e classes abstratas são bastante parecidas:

- Sintaxe semelhante
- Não podem ser instanciadas
- Especificam detalhes que irão compor outras classes

Como decidir qual utilizar?

- Não existe "regra geral", mas existem pistas...

Interface vs. Classe abstrata

Interface	Classe abstrata
Não pode possuir atributos	Pode possuir atributos
Classes podem implementar mais de uma interface	Classes somente podem herdar de uma superclasse
Declara um comportamento esperado (protótipo de métodos e documentação)	Impõe um comportamento padrão (atributos e métodos implementados)
Filhos são independentes em implementação	Filhos são implementados sobre estrutura anterior
Filhos naturalmente têm estruturas diferentes	Filhos naturalmente têm estrutura similar

É interessante usar classes abstratas se classes "filhas" são naturalmente semelhantes, ou possuem descrição de estado comum (atributos comuns)

É interessante usar interfaces se classes "filhas" são naturalmente diferentes, ou implementam funcionalidades de formas muito diferentes

Algumas interfaces importantes

Comparable: objetos que podem ser comparados uns com os outros pelo método `compareTo`

Runnable: objetos que podem ser "executados" em threads

Printable: objetos que podem ser renderizados na forma de páginas (como uma impressão ou geração de PDF)

Serializable: objetos que podem ser convertidos em sequências binárias e recuperados

Iterable: objetos que podem ser percorridos em um laço "for each" (por exemplo: `ArrayList`'s)

Iterator: objeto utilizado para percorrer um `Iterable`

Interfaces

- Interfaces podem estender outras interfaces
  - Especificar mais métodos a serem implementados
  - Hierarquia de interfaces mais restritivas
  - Uma única interface pode estender múltiplas outras
- É possível misturar interfaces e classes abstratas
  - Classe abstrata implementa parte dos métodos de uma interface, declara os demais como abstratos
  - Classes filhas implementam os métodos restantes
- Desde Java 8: interfaces podem ter métodos padrão (default methods)
  - Implementação padrão de métodos ($\approx$ herança)
  - Palavra-chave `default` antes do nome do método
  - Objetivo: permitir evoluções retrocompatíveis

Exercícios

1. Modifique seu sistema acadêmico (ver aulas anteriores) para representar acadêmicos: pessoas que atuam no meio acadêmico. Somente acadêmicos podem ser monitores de disciplinas.
   - Na nossa implementação anterior, a classe `Pessoa` possuía um método `podeSerMonitor` que verifica se a pessoa pode ser monitor ou não de uma disciplina. Cada subclasse sobrescreve este método segundo seus próprios critérios de monitoria.
   - Agora, nosso objetivo é que esta exigência não esteja mais na classe `Pessoa`, mas em uma interface projetada especificamente para isso. Objetos de uma classe poderão ser monitores de disciplina somente se esta classe implementar a nova interface.

Exercícios

- Crie uma classe `PosDoc` para representar pesquisadores de pós-doutorado. Inclua como atributo desta classe uma lista de tópicos de pesquisa;
- Crie uma subclasse de `Aluno`, `AlunoPos`, para representar alunos de pós-graduação. Alunos de pós-graduação devem, obrigatoriamente, possuir um orientador (para alunos em geral, isto é opcional) e um título de tese;
- Crie uma interface `Academico`, que requer a implementação de um método com o protótipo:

```
public boolean podeSerMonitor (Disciplina disciplina)
```

Exercícios

- Modifique as classes `Professor`, `PosDoc` e `AlunoPos` para que implementem esta interface, segundo as seguintes (novas) regras de monitoria:
  - Um professor pode ser monitor de disciplina se ele não for o professor responsável por ela;
  - Alunos em geral não podem ser monitores de disciplina (a classe `Aluno` não implementa a interface), mas um aluno de pós-graduação pode ser monitor de disciplina se seu orientador for o responsável;
  - Um pesquisador pós-doc pode ser monitor de qualquer disciplina;
- Modifique a classe `Turma` para que o atributo `monitor` seja do tipo `Academico`;
- Por fim, remova o método abstrato `podeSerMonitor` da classe `Pessoa`.