Programação Orientada a Objetos
SANTOS, Rafael

Capítulo 8 – Reutilização de classes

Atividade de revisão

- Neste capítulo e nos seguintes serão usados intensamente os conceitos básicos de OO e da linguagem estudados até agora, com ênfase para os de número 1, 2 e 3. Releia-os (e os slides referentes a eles) e resolva novamente os exercícios de maior complexidade, aqueles marcados com três ou mais asteriscos.

Modalidades

- Delegação (associações do tipo “tem-um”, “tem-vários”)
- Herança (associações do tipo “é-um”)

Delegação

- Um campo (variável) declarado em uma classe faz referência a um objeto de outra classe
- Representada por associações entre classes: simples, agregação e composição
- Ver classes:
  - DataHora (Cap. 8)
  - RegistroAcademicoDeGraduacao (Cap. 8)
  - DemoRegistroAcademicoDeGraduacao (Cap. 8)

Exercício

- Copie a classe Data do capítulo 2, Criando Classes, para a pasta onde estão os fontes do capítulo 8, Reutilização, e promova as alterações necessárias para eliminação dos erros (Dica: falta construtor em Data)
- Crie uma classe Hora de modo que a classe DataHora compile e funcione corretamente
- Crie uma classe TestaDataHora para instanciar e testar objetos da classe DataHora

Exercício (continuação)

- Troque o campo (variável de instância) do tipo Data definido na classe RegistroAcademicoDeGraduacao, por um do tipo DataHora e promova todas as alterações para que as classes compilem e funcionem, aproveitando os recursos adicionais oferecidos pela classe DataHora

Delegação e Modificadores de Acesso

- As **restrições** impostas por modificadores de acesso a um determinado membro são **somadas**, isto é, são **consideradas em conjunto**.
- Ver classes:
  - PessoaO (Cap. 8)
  - FuncionarioO (Cap. 8)
  - Demo FuncionarioO (Cap. 8)

Delegação e Construtores

- Os objetos das classes contidas podem ser **criados** em qualquer ponto da classe que as contém, ou serem **passados como argumento** do construtor ou de um método
- A chamada aos construtores das classes contidas é feita por meio da **palavra-chave *new***, sempre **dando origem** a um novo objeto, como nos demais casos.

Herança

- Trata-se de uma **extensão**, onde uma classe base, mais **genérica**, é usada para criação de uma classe filha, mais **específica**, que define **características adicionais**.
- Ver classes:
  - Pessoa (Cap. 8)
  - Funcionario (Cap.8)
  - ChefeDeDepartamento (Cap. 8)

Modificador de acesso protected

- Permite acesso direto pelas classes do **mesmo pacote** (igual ao acesso padrão, sem modificadores) e pelas **subclasses**, mesmo que estas estejam em pacotes diferentes
- Para uma **subclasse de fora do pacote**, o **membro protected** só pode ser acessado por meio de **herança**
- Para os demais níveis e modificadores de acesso, ver slides do capítulo 2.

protected: acesso do mesmo pacote

```
package testeDeProtected.outro;
public class Mae {
    protected int x;
}

package testeDeProtected.outro;
public class FilhaNoMesmoPacote extends Mae {
    void imprime() {
        System.out.println(x);

        // Eh valido, mas nao esta usando o acesso
        // protected e sim o acesso de pacote, que
        // todos os membros protected possuem
        System.out.println(new Mae().x);
    }
}
```

protected: acesso de outro pacote

```
package testeDeProtected.outro;
public class Mae {
    protected int x;
}

package testeDeProtected;
import testeDeProtected.outro.Mae;
public class FilhaEmPacoteDiferente extends Mae {
    void imprime() {
        System.out.println(x);

        // Nao eh valido, pois nao se pode
        // fazer acesso direto a um membro protected
        // Erro de compilacao
        System.out.println(new Mae().x);
    }
}
```

protected: acesso a partir de neta em outro pacote

```
package testeDeProtected;

public class NetaEmPacoteDiferente
        extends FilhaEmPacoteDiferente {

    void qqCoisa() {
        x = 12;
        System.out.println(x);

        // Erro de compilacao, x tem acesso protegido
        new FilhaEmPacoteDiferente().x = 12;
    }
}
```

Palavra-chave super

- Dá **acesso** aos **membros da superclasse**, inclusive ao **construtor** da superclasse, permitindo reutilização de código, respeitadas as restrições impostas
- Ver classes:
  - Automovel (Cap. 8)
  - AutomovelBasico (Cap.8)
  - AutomovelDeLuxo (Cap. 8)
  - DemoAutomoveis (Cap. 8)

Herança e construtores

- O construtor da superclasse é chamado pelo construtor da subclasse, na **primeira** linha, mesmo que implicitamente – **super()**
- Quando especificada pelo programador, a chamada ao construtor deve ser feita também na **primeira** linha
- Não é possível invocar o construtor da superclasse a partir um método
- Construtores **não** são **herdados**

Palavra-chave instanceof

- Serve para verificar se uma variável **referencia** um objeto de uma determinada classe
- Quando usada com uma classe **ancestral**, direta ou indireta, da referência, retorna *true*, se houver instância, e *false*, se não houver instância referenciada (= *null*)
- **Não** pode ser usada com classes que não estão na **hierarquia** do tipo da referência

Sobreposição e ocultação

- A subclasse pode **ocultar campos** (em tempo de compilação) ou **sobrepor** métodos (em tempo de execução) da superclasse, tornado suas características mais específicas
- Os modificadores de acesso dos métodos que sobrepõem **não podem ser mais restritivos** que os declarados nos métodos sobrepostos

Sobreposição e ocultação

```
public class Primeira {
    int a = 1;
    void imprime() {
        System.out.println(a);
    }
    void imprimeSemOverride() {
        System.out.println(a);
    }
}

public class Segunda extends Primeira {
    int a = 2; // ocultacao
    void imprime() { // sobrescricao
        System.out.println(a);
    }
}
```

Sobreposição e ocultação

```
public class TestaOcultacao {
    public static void main(String[] args) {
        Primeira p0 = new Primeira();
        System.out.println(p0.a); // 1
        p0.imprime();             // 1
        p0.imprimeSemOverride();  // 1

        Primeira p1 = new Segunda();
        System.out.println(p1.a); // 1
        p1.imprime();             // 2
        p1.imprimeSemOverride();  // 1

        Segunda p2 = new Segunda();
        System.out.println(p2.a); // 2
        p2.imprime();             // 2
        p2.imprimeSemOverride();  // 1
    }
}
```

Polimorfismo

- Muitas formas
- Depende da herança
- Permite a manipulação de várias subclasses de uma classe ancestral de forma unificada e mais genérica
- Ver classes:
  - ConcessionariaDeAutomoveis (Cap. 8)
  - EmprestimoBancario (Cap. 8)
  - EmprestimoBancarioComCast (Cap. 8)

Idéias para exercícios extras

- Em um vídeo game com temática espacial existem vários **objetos espaciais** que precisam ser desenhados na tela, cada um com suas características
- Em um empresa existem vários tipos de **empregados**: que trabalham por hora, assalariados e que recebem por projetos. Todos eles devem ter seus salários calculados mensalmente

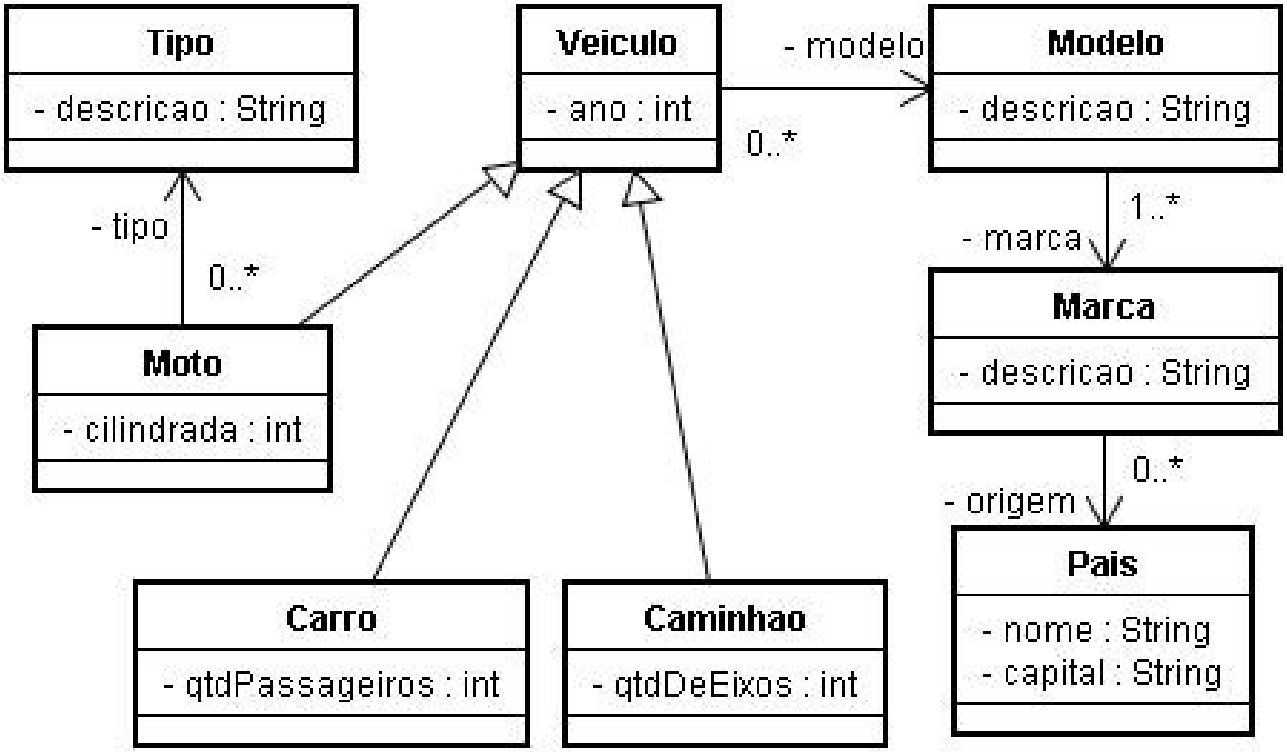
Idéias para exercícios extras

- Um **quadrilátero** pode ser um retângulo, um quadrado, um paralelogramo ou um trapezóide. Para todos eles devem ser calculados os perímetros e as áreas

Exercícios sugeridos para sala

- 8.4 a 8.9, 8.28, 8.29, 8.31, 8.32, 8.33 a 8.43
- Todos os exercícios do capítulo devem ser resolvidos e eventuais dúvidas trazidas para debate
- Quando necessário, resolver exercícios de capítulos anteriores que são solicitados como base para exercícios do capítulo corrente

Implemente



Adote:

- Encapsulamento
- Construtores Sobrecarregados
- toString()

Crie:

- Uma classe de teste