Programação Orientada a Objetos

Aula 3 – Introdução a Java
Classes e Construtores em Java

Ana Patrícia F. Magalhães Mascarenhas

anapatriciamagalhaes@gmail.com

- Objetivo
  - Descrever as vantagens e conceitos do encapsulamento
  - Distinguir os modificadores de acesso e sua aplicação
  - Desenvolver classes que utilizem encapsulamento
  - Apresentar o conceito de sobrecarga
  - Desenvolver classes que utilizem sobrecarga
- Bibliografia básica

  - BARNES, J. David, KÖLLING, Michael. **Programação Orientada a Objetos com Java.** Pearson, 2004.
  - HORSTMANN, Gay S., CORNELL, Gary. **Core Java 2. Volume I – Fundamentos.** Makron Books, 2000.
  - Santos, Rafael. **Introdução a Programação Orientada a Objetos com Java.** Ed. Campos, 2. ed., 2013

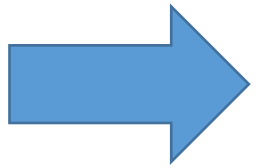
- O que acontece quando você está com dor de cabeça e toma um comprimido?
- Não sabemos, mas também não precisamos saber! O médico precisa.
- Sabemos que o comprimido tem um serviço de analgesia e nos consumimos esse serviço!

O que acontece quando compramos algo pela internet e pagamos com cartão de crédito?

- Os sites geralmente estão filiados a um serviço de pagamento, ex. paypal
- O site não tem relacionamento com bancos ou administradoras de cartão, esses serviços é que tem
- Nos geralmente conhecemos esse serviço e sabemos que é seguro

Vamos imaginar que a comunicação funciona assim:

- O site informa o valor
- Você
  - Informa o número do cartão
  - Validade
  - Código de verificação
  - Senha



Sistema de pagamento

Checa seus dados

Valida sua senha

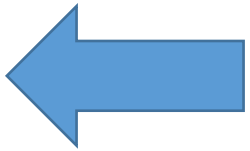
Verifica se você tem saldo disponível

Se tiver saldo

- Atualiza seu saldo
- Gera o número de autorização
- Devolve a autorização

Se não tiver saldo

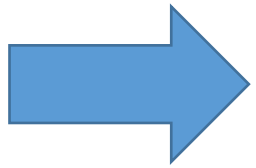
- Devolve código de erro



- O site
  - Recebe o retorno da solicitação de pagamento
  - (Sim/Não)

E se a forma como o sistema de pagamento checa os dados mudar?

- O site informa o valor
- Você
  - Informa o número do cartão
  - Validade
  - Código de verificação
  - Senha



Sistema de pagamento

Checa seus dados

Valida sua senha

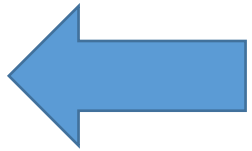
Verifica se você tem saldo disponível

Se tiver saldo ou (o valor for até 1000,00)

- Atualiza seu saldo
- Gera o número de autorização
- Devolve a autorização

Se não tiver saldo

- Devolve código de erro



- O site
  - Recebe o retorno da solicitação de pagamento
  - (Sim/Não)

O que mudou no site?

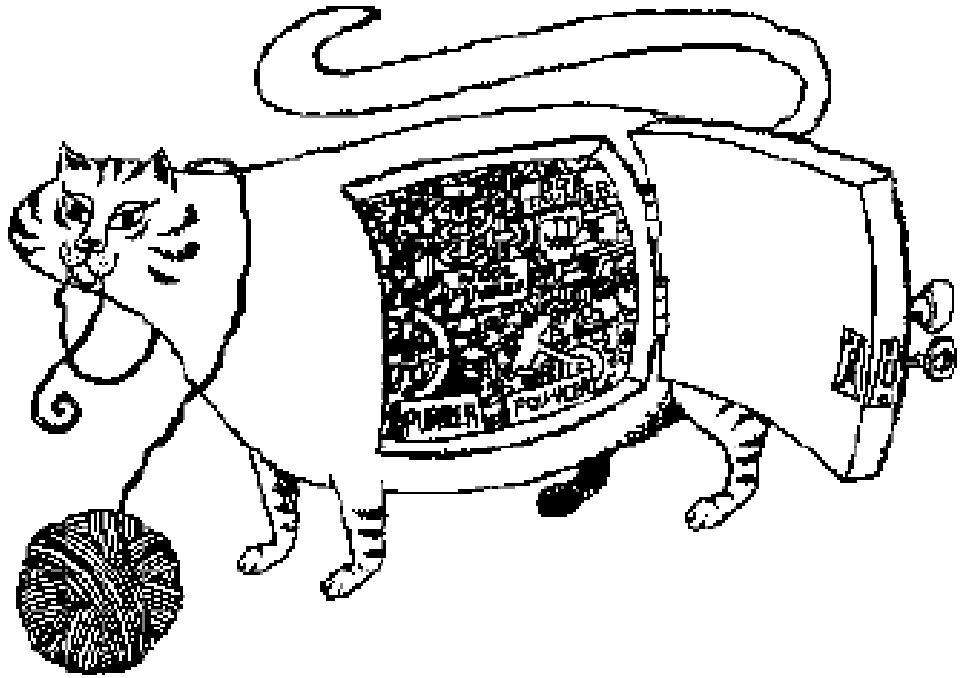
Encapsulamento

Em muitos casos, será desejável ocultar detalhes de funcionamento interno de das classes

Ex: Máquina fotográfica – o mecanismo da câmera oculta os dados e a maneira como são processados.

Encapsulamento

- Interação entre objetos sem conhecimento do funcionamento



Encapsulamento

O encapsulamento pode trazer diversas vantagens:

- Protege os dados contra acessos indevidos
- Tende a criar sistemas mais coesos
- Facilita o reuso
- Facilita a manutenção
- Facilita a evolução

Encapsulamento

A capacidade de ocultar dados dentro do modelos, permitindo que somente operações especializadas ou dedicadas manipulem os dados ocultos chama-se ***encapsulamento;***

Encapsulamento é um dos benefícios mais palpáveis da POO;

Modelos que encapsulam os dados possibilitam a criação de programas com menos erros e mais clareza.

Como implementar o encapsulamento?

Modificadores de acesso

Java permite a restrição ao acesso a campos e métodos por intermédio de *modificadores de acesso* que são declarados dentro das classes.

Existem quatro modificadores de acesso, por enquanto vamos estudar dois:

public: garante que o campo ou método da classe declarado com este modificador poderá ser acessados ou executado a partir de qualquer outra classe.

Private: só podem ser acessados, modificados ou executados por métodos da mesma classe, sendo ocultos para o programador usuário que for usar instâncias desta classe ou criar classes herdeiras ou derivadas.

Ao criar classes, o programador de classes deve implementar uma política de ocultação ou de acesso a dados e a métodos internos.

Como implementar o encapsulamento?

Modificadores de acesso

Regras básicas para implementação de políticas para classes simples:

Todo campo deve ser declarado como ***private*** .

Métodos que devem ser acessíveis devem ser declarados com o modificador ***public***. Caso classes não venham a ser agrupadas em pacotes, a omissão não gera problemas.

Métodos para controle dos campos devem ser escritos, e estes métodos devem ter o modificador ***public***.

Se for desejável, métodos podem ser declarados como ***private***.

Get e Set

- Na OO temos um padrão de métodos para atualizar e consultar dados ocultos (privados) em uma classe

  - Get: utilizados para obter (consultar) o valor do atributo oculto.

  - Set: utilizado para alterar o valor de um atributo oculto.

  O uso de get e set garante um único ponto de acesso ao atributo

  - Permite realização de críticas, etc

Get e Set (continuação)

```
public class Aluno{//classe Aluno
  private int matricula;
  private String nome, curso;
  private float mensalidade;
  public Aluno(int matricula, String
     nome, String curso, float
     mensalidade){
     this.matricula = matrcula;
     this.nome=nome;
     this.curso=curso;
     this.mensalidade=mensalidade;
  };
  public void setMatricula(int
     matricula){
    this.matricula=matricula;
}
```

```
public int getMatricula(){
   return this.matricula;
}

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

public String getNome(){
  return this.nome;
}
```

Sobrecarga

- No mundo real temos homônimos, pessoas com mesmo nome
  - Ex: José da Silva
  - Essas pessoas em geral tem assinaturas diferentes que podem ser reconhecidas e distinguidas por um profissional especializado

Sobrecarga

- Na OO um método pode ter o mesmo nome que outro método na mesma classe porém com assinaturas diferentes.
- Assinatura do método: nome do método + conjunto de argumentos
  - Diferenças no nome dos parâmetros não são relevantes
  - Diferenças no tipo de retorno são permitidas, mas não são relevantes
  - Ex.:

```
public void calcular()
public void calcular(float percentual)
public float calcular(String tipo)
 public float calcular(String nome)
public float calcular(int tipo)
public float calcular(String tipo, float percentual)
```

- Isto é usual quando existem duas ou mais formas diferentes de se realizar a mesma tarefa
- Neste caso diz-se que o método está **sobrecarregado**

Sobrecarga

```
class Ponto{
  float x;
  float y;
  public Ponto(float pX,float pY){
   x = pX;
   y = pY;
  }
  void mover(float novoX,float novoY){
   x = novoX;
   y = novoY;
 }
 void mover(){
   x = 0;
   y = 0;
 }
}
```

Classe que representa um ponto em um plano cartesiano

Construtor

Método mover para mover o ponto para uma localização x, y informada

Método mover para mover o ponto para a origem, x=0 e y=0

Sobrecarga

- Java admite também, que se sobrecarregue os construtores de uma classe.
- As restrições são similares àquelas aplicadas aos métodos sobrecarregados
- Pode-se se referir de dentro de um construtor sobrecarregado para outro, através do uso da palavra reservada **`this`**

Sobrecarga

```
class Ponto{
  float x;
  float y;
  public Ponto(float pX,float pY){
   x = pX;
   y = pY;
  }
  public Ponto(){
   x = 0;
   y = 0;
  }
 void mover(float novoX,float novoY){
   x = novoX;
   y = novoY;
 }
...}
```

Classe que representa um ponto em um plano cartesiano

Construtor

Construtor sobrecarregado

Método mover para mover o ponto para a origem, x=0 e y=0

Sobrecarga

```
class Ponto{
  float x;
  float y;
  public Ponto(float pX,float pY){
   x = pX;
   y = pY;
  }
  public Ponto(){
   this(0,0)
  }

 void mover(float novoX,float novoY){
   x = novoX;
   y = novoY;
  }
...}
```

Classe que representa um ponto em um plano cartesiano

Construtor

Construtor sobrecarregado usando this

Método mover para mover o ponto para a origem, x=0 e y=0

A referência **`this`**

- Na chamada dos métodos de uma classe, a máquina virtual passa implicitamente como parâmetro uma referência ao objeto ao qual a mensagem foi enviada.

- Quando se deseja recuperar esta referência de dentro do corpo da classe, usa-se a palavra reservada **`this`**

A referência **`this`**

- A referência this é usada para:
  - referência ao próprio objeto
  - acesso a variáveis do objetos
  - passagem do objeto como parâmetro

```
class Circulo{//classe Círculo
  float raio = 0;
  void alterarRaio(float raio){
    this.raio = raio;
  }
}
```

Exercício

1) Crie uma classe que represente um medicamento em uma farmácia. O medicamento possui um nome, princípio ativo e preço de custo. Em seguida construa os seguintes métodos:
   - Método para criar um medicamento informando todos os seus dados;
   - Método para criar um medicamento informando somente o nome do medicamento e o preço de custo. Neste caso o princípio ativo tem o mesmo nome do medicamento.
   - Método para calcular o preço de venda de um medicamento. Para isso receber como argumento o percentual de lucro da farmácia e aplicar ao preço de custo do medicamento.
   - Método para calcular o preço de venda de um medicamento considerando um lucro de 30%.

Exercícios para casa

Baixar a Ide BlueJ

Resolver a lista de exercícios nr. 2