Classes e Objetos

SCC0604 - Programação Orientada a Objetos

Prof. Fernando V. Paulovich
http://www.icmc.usp.br/~paulovic
paulovic@icmc.usp.br

Instituto de Ciências Matemáticas e de Computação (ICMC)
Universidade de São Paulo (USP)

9 de agosto de 2010

Introdução a POO em Java

- Na POO você só se preocupa com o que o objeto expõe, não como o mesmo é implementado

Introdução a POO em Java

- Na POO você só se preocupa com o que o objeto expõe, não como o mesmo é implementado
- Um objeto nunca deve manipular diretamente os dados internos de outro objeto

Introdução a POO em Java

- Na POO você só se preocupa com o que o objeto expõe, não como o mesmo é implementado
- Um objeto nunca deve manipular diretamente os dados internos de outro objeto
  - Esse manipulação deve ser feita somente via métodos

Introdução a POO em Java

- Na POO você só se preocupa com o que o objeto expõe, não como o mesmo é implementado
- Um objeto nunca deve manipular diretamente os dados internos de outro objeto
  - Esse manipulação deve ser feita somente via métodos
  - Isso garante o princípio de encapsulamento

Introdução a POO em Java

- O mecanismo que torna e reutilização de código efetiva é a herança (junto com polimorfismo)

Introdução a POO em Java

- O mecanismo que torna e reutilização de código efetiva é a herança (junto com polimorfismo)
- Em Java, diz-se que uma classe estende (**estends**) a outra

Introdução a POO em Java

- O mecanismo que torna e reutilização de código efetiva é a herança (junto com polimorfismo)
- Em Java, diz-se que uma classe estende (**estends**) a outra
- Toda classe em Java já estende automaticamente uma "classe base cósmica" chamada de **Object**.

Uso de Classes Existentes

- Em uma aplicação Java, criam-se objetos, especifica-se o estado inicial de cada um e depois se trabalha com eles

```
1 Classe obj; //cria-se a variável
2 obj = new Classe(); //cria-se o objeto
3 obj.metodo(); //usa-se o objeto
```

Referenciando o mesmo Objeto

```
1 Classe obj1, obj2; //cria-se a variável
2 obj1 = new Classe(); //cria-se o objeto
3 obj2 = obj1; //obj2 referencia obj1
4 obj1.metodo(); //usa-se obj1 (obj2)
5 obj2.metodo(); //usa-se obj1 (obj2)
```

Referenciando **null**

- Pode-se explicitamente fazer uma variável referenciar o objeto **null**, indicando que a mesma não referencia nenhum objeto

Referenciando **null**

- Pode-se explicitamente fazer uma variável referenciar o objeto **null**, indicando que a mesma não referencia nenhum objeto
- Chamar um método através de uma variável que referencia **null** irá causar um erro de execução (**NullPointerException**)

Referenciando **null**

- Pode-se explicitamente fazer uma variável referenciar o objeto **null**, indicando que a mesma não referencia nenhum objeto
- Chamar um método através de uma variável que referencia **null** irá causar um erro de execução (**NullPointerException**)
- As variáveis locais **NÃO** são automaticamente inicializadas com **null**

Métodos Acessadores e Modificadores

- De forma a manter a integridade do paradigma orientado a objetos, **todos os atributos de uma classe devem ser declarados como privados ou protegidos**

Métodos Acessadores e Modificadores

- De forma a manter a integridade do paradigma orientado a objetos, **todos os atributos de uma classe devem ser declarados como privados ou protegidos**
- Caso haja a necessidade de alterar/consultar o valor desses atributos, métodos devem ser providos para isso

Métodos Acessadores e Modificadores

- De forma a manter a integridade do paradigma orientado a objetos, **todos os atributos de uma classe devem ser declarados como privados ou protegidos**
- Caso haja a necessidade de alterar/consultar o valor desses atributos, métodos devem ser providos para isso
- Normalmente, os métodos que retornam o valor armazenado em um atributo (acessadores) são nomeados utilizando um prefixo **get** seguido pelo nome do atributo; e os métodos que modificam os valores dos atributos (modificadores) são nomeados utilizando um prefixo **set** seguido pelo nome do atributo

Exemplo de Classe

```
1  public class Data {
2     private int dia;
3     private int mes;
4     private int ano;
5
6     public int getDia() {
7         return this.dia;
8     }
9
10    public void setData(int dia, int mes, int ano) {
11        this.dia = dia;
12        this.mes = mes;
13        this.ano = ano;
14    }
15 }
```

Objetos como Argumentos de Métodos

- Em Java todos os métodos trabalham com passagem por valor, e não passagem por referência

- O seguinte método funciona?

```
1 public static void trocaDatas(Data a, Data b) {
2   Data temp = a;
3   a = b;
4   b = temp;
5 }
```

O Seguinte Método Funciona?

```
1 public static void mudarData(Data d, int anosAtrasos) {
2   int dia = d.getDia();
3   int mes = d.getMes();
4   int ano = d.getAno();
5   d = new Data(dia, mes, ano+anosAtrasos);
6 }
```

- Qual a data de meta após a execução do seguinte trecho de código?

```
1 Data meta = new Data(1, 1, 2004);
2 Data.mudarData(meta, 2);
```

O Seguinte Método Funciona?

```
1 public static void mudarData(Data d, int anosAtrasos) {
2   int dia = d.getDia();
3   int mes = d.getMes();
4   int ano = d.getAno();
5   d.setData(dia, mes, ano+anosAtrasos);
6 }
```

- Qual a data de meta após a execução do seguinte trecho de código?

```
1 Data meta = new Data(1, 1, 2004);
2 Data.mudarData(meta, 2);
```

Como Começar a Construir suas Próprias Classes

- A sintaxe mais simples de uma classe Java é:

```
1 modificadorDeAcesso class nomeDaClasse {
2 }
```

Tipos de Modificadores de Acesso

- friendly (amiga) : somente as classes do próprio pacote que a mesma está inserida é que podem acessa-la
- **public** (pública) : qualquer classe do sistema pode ter acesso a mesma

Como Começar a Construir suas Próprias Classes

- As classes Java devem ser implementadas dentro de arquivos (texto) com extensão java

Como Começar a Construir suas Próprias Classes

- As classes Java devem ser implementadas dentro de arquivos (texto) com extensão java
- Em cada arquivo podem ser declaradas quantas classes forem necessário, mas somente uma classe com acesso público (**public**) por arquivo pode existir

Como Começar a Construir suas Próprias Classes

- As classes Java devem ser implementadas dentro de arquivos (texto) com extensão java
- Em cada arquivo podem ser declaradas quantas classes forem necessário, mas somente uma classe com acesso público (**public**) por arquivo pode existir
  - O nome do arquivo deve ser o mesmo da única classe pública presente, inclusive observando letras maiúsculas e minúsculas

Uma classe Data

```
1  public class Data {
2    private int dia;
3    private int mes;
4    private int ano;
5
6    public int getDia() {
7      return this.dia;
8    }
9
10   public void setData(int dia, int mes, int ano) {
11     this.dia = dia;
12     this.mes = mes;
13     this.ano = ano;
14   }
15 }
```

Dissecando a Classe Data

- Todos os métodos também são acompanhados de modificadores de acesso, que refletem o nível de acesso aos mesmos

Dissecando a Classe Data

- Todos os métodos também são acompanhados de modificadores de acesso, que refletem o nível de acesso aos mesmos
- Os atributos também devem ter seus modificadores de acesso especificados, e obrigatoriamente devem usar modificadores privados (**private**)

Problemas com Métodos Acessadores e Modificadores

```
1  public class Funcionario {
2    private Data dataContratacao;
3
4    public Data getDataContratacao() {
5      return dataContratacao;
6    }
7  }
8
9  ...
10
11 Funcionario paulo = new Funcionario("Fernando", 1,1,2004);
12 System.out.println(paulo.toString());
13
14 Data d = paulo.getDataContratacao();
15 d.setData(1,1,1970);
16
17 System.out.println(paulo.toString());
```

O método **toString()** usado acima é um método padrão existente em Java que retorna uma String com o valor dos atributos da classe. Na verdade esse método deve ser provido explicitamente pelo programador e tem a assinatura: **public String toString()**

Problemas com Métodos Acessadores e Modificadores

```
1  public class Funcionario {
2    private Data dataContratacao;
3
4    public Data getDataContratacao() {
5      return (Data)dataContratacao.clone();
6    }
7  }
8
9  ...
10
11 Funcionario paulo = new Funcionario("Fernando", 1,1,2004);
12 System.out.println(paulo.toString());
13
14 Data d = paulo.getDataContratacao();
15 d.setData(1,1,1970);
16
17 System.out.println(paulo.toString());
```

O método **toString()** usado acima é um método padrão existente em Java que retorna uma String com o valor dos atributos da classe. Na verdade esse método deve ser provido explicitamente pelo programador e tem a assinatura: **public String toString()**

Regra Indispensável

Sempre que for necessário retornar um objeto que é um atributo da classe, retorne um clone desse atributo por meio do método **public Object clone()**

Acesso de Métodos a Dados Privados

- Um método pode acessar os dados privados do objeto no qual é chamado. Além disso, um método pode acessar os dados privados de todos os objetos de sua classe

```
1  public class Data {
2    private int dia;
3    private int mes;
4    private int ano;
5
6    public boolean igual(Data d) {
7      return(this.ano == d.ano && this.mes == d.mes && ↩
           this.dia == d.dia);
8    }
9  }
```

Métodos Privados

- Os métodos além de públicos também pode ser privados

```
1  public class Data {
2    private int dia;
3    private int mes;
4    private int ano;
5
6    ...
7
8    private boolean bissexto(int ano) {
9      if (year % 400 == 0 || (year%4 == 0 && year%100 != 0)) {
10       return true;
11     } else {
12       return false;
13     }
14   }
15 }
```

Primeiros Passos com Construtores

- Os construtores são usados para inicializar objetos de uma classe, dando aos atributos o estado inicial que se quer que os mesmos tenham

Primeiros Passos com Construtores

- Os construtores são usados para inicializar objetos de uma classe, dando aos atributos o estado inicial que se quer que os mesmos tenham
- Um construtor:

Primeiros Passos com Construtores

- Os construtores são usados para inicializar objetos de uma classe, dando aos atributos o estado inicial que se quer que os mesmos tenham
- Um construtor:
  - Tem o mesmo nome da classe

Primeiros Passos com Construtores

- Os construtores são usados para inicializar objetos de uma classe, dando aos atributos o estado inicial que se quer que os mesmos tenham
- Um construtor:
  - Tem o mesmo nome da classe
  - Pode ter um ou mais parâmetros

Primeiros Passos com Construtores

- Os construtores são usados para inicializar objetos de uma classe, dando aos atributos o estado inicial que se quer que os mesmos tenham
- Um construtor:
  - Tem o mesmo nome da classe
  - Pode ter um ou mais parâmetros
  - Sempre é chamado através do comando **new**

Primeiros Passos com Construtores

- Os construtores são usados para inicializar objetos de uma classe, dando aos atributos o estado inicial que se quer que os mesmos tenham
- Um construtor:
  - Tem o mesmo nome da classe
  - Pode ter um ou mais parâmetros
  - Sempre é chamado através do comando **new**
  - Não retorna valor

Primeiros Passos com Construtores

- Os construtores são usados para inicializar objetos de uma classe, dando aos atributos o estado inicial que se quer que os mesmos tenham
- Um construtor:
  - Tem o mesmo nome da classe
  - Pode ter um ou mais parâmetros
  - Sempre é chamado através do comando **new**
  - Não retorna valor
  - Não podem ser explicitamente chamados como um método da classe

Observações sobre Inicialização de Atributos

- É possível atribuir valores padrão para os atributos de uma classe

```
1 public class Data {
2    private int dia = 1;
3    private int mes = 1;
4    private int ano = 1900;
5 }
```

Referência **this**

- Em um método, a palavra-chave **this** faz referência ao objeto no qual o método opera

Referência **this**

- Em um método, a palavra-chave **this** faz referência ao objeto no qual o método opera
- **this** tem o significado de: o objeto para o qual este trecho de código está sendo executado

Referência **this**

- Em um método, a palavra-chave **this** faz referência ao objeto no qual o método opera
- **this** tem o significado de: o objeto para o qual este trecho de código está sendo executado
- O uso do **this** é especialmente importante para tratar a ambiguidade que ocorre quando um parâmetro de um método tem o mesmo nome e tipo de um atributo da classe

Referência **this**

- No caso do método **setNome()** ao lado, a identificador nome se refere ao parâmetro nome. Assim para referenciar o atributo nome usamos a referência **this**

```
1  public class Funcionario {
2    private String nome;
3    ...
4
5    public void setNome(String ↩
         nome) {
6      this.nome = nome;
7    }
8    ...
9  }
```

Finalizadores

- Java não tem o conceito de destruidor (ou destrutor), mas oferece um método chamado **finalize** que é chamado logo antes de um objetos ser destruído. Porém, isso não é garantido, de forma que não devemos contar com esse método para fazer qualquer limpeza necessária (por exemplo, fechar um arquivo)

Finalizadores

- Java não tem o conceito de destruidor (ou destrutor), mas oferece um método chamado **finalize** que é chamado logo antes de um objetos ser destruído. Porém, isso não é garantido, de forma que não devemos contar com esse método para fazer qualquer limpeza necessária (por exemplo, fechar um arquivo)
- Para se fazer essa limpeza, devemos prover métodos para tal fim. Como padrão, esses métodos devem chamar **dispose()** - se a classe tiver algum atributo que seja um objeto de uma classe que ofereça esse método, devemos chamar explicitamente esse método na implementação do nosso **dispose()**

Métodos e Campos **static**

- Os atributos estáticos não mudam de objeto de uma classe para outro, de modo que podem ser vistos como se pertencessem a uma classe

Métodos e Campos **static**

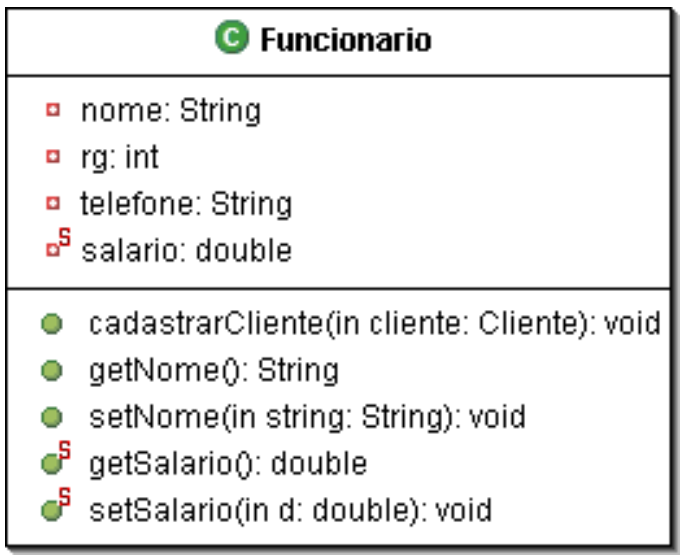
- Os atributos estáticos não mudam de objeto de uma classe para outro, de modo que podem ser vistos como se pertencessem a uma classe
- Da mesma forma, os métodos estáticos pertencem a uma classe e não operam em nenhum objeto da classe - isso significa que podemos usá-los sem criar um objeto da classe

Métodos e Campos **static**

- Os atributos estáticos não mudam de objeto de uma classe para outro, de modo que podem ser vistos como se pertencessem a uma classe
- Da mesma forma, os métodos estáticos pertencem a uma classe e não operam em nenhum objeto da classe - isso significa que podemos usá-los sem criar um objeto da classe
- Para usar métodos estáticos, usamos a seguinte sintaxe:

```
NomeClasse.MetodoEstatico(parametros);
```

Exemplo Modelagem e Código




```
1  public class Funcionario {
2    private static double salario↩
           = 1000;
3    ...
4
5    public static double ↩
         getSalario() {
6      return salario;
7    }
8
9    public static void setSalario↩
         (double d) {
10     salario = d;
11   }
12
13   ...
14 }
```

Observações

- Como os métodos estáticos não operam sobre objetos de uma classe, eles só podem acessar atributos estáticos da classe - não usam a referência **this**

Programa Inicial

- Todo programa (conjunto de classes) Java começa sua execução em um método público estático chamado **main**

Programa Inicial

- Todo programa (conjunto de classes) Java começa sua execução em um método público estático chamado **main**
- Como qualquer outro método, o **main** deve fazer parte de uma classe, mas por ser um método estático, o mesmo pode ser executado sem ser necessário a instanciação de um objeto dessa classe

Programa Inicial

- Todo programa (conjunto de classes) Java começa sua execução em um método público estático chamado **main**
- Como qualquer outro método, o **main** deve fazer parte de uma classe, mas por ser um método estático, o mesmo pode ser executado sem ser necessário a instanciação de um objeto dessa classe
- O método **main** só pode acessar campos estáticos da classe

Programa Principal

```
1 public class Principal {
2    public static void main(String[] args) {
3       System.out.println(''Teste.'');
4    }
5 }
```

Pacotes

- A linguagem Java permite agrupar classes em uma coleção chamada pacote

Pacotes

- A linguagem Java permite agrupar classes em uma coleção chamada pacote
- Ao escrever um pacote é preciso colocar o nome do pacote no topo do arquivo-fonte que declara a classe

Pacotes

- A linguagem Java permite agrupar classes em uma coleção chamada pacote
- Ao escrever um pacote é preciso colocar o nome do pacote no topo do arquivo-fonte que declara a classe
- Um pacote reflete uma estrutura de diretórios

Pacotes

- A linguagem Java permite agrupar classes em uma coleção chamada pacote
- Ao escrever um pacote é preciso colocar o nome do pacote no topo do arquivo-fonte que declara a classe
- Um pacote reflete uma estrutura de diretórios
- Um pacote é declarado usando-se a palavra-chave **package**

Exemplo de Pacotes

```
1  package calendario;
2
3  public class Data {
4    private int dia;
5    private int mes;
6    private int ano;
7
8    public int getDia() {
9      return this.dia;
10   }
11
12   public void setData(int dia, int mes, int ano) {
13     this.dia = dia;
14     this.mes= mes;
15     this.ano = ano;
16   }
17 }
```

Usando Pacotes

- Para se usar um pacote criado, usa-se:

```
import pacote.*;
```

Usando Pacotes

- Para se usar um pacote criado, usa-se:

```
import pacote.*;
```

- Também é possível se importar um classe específica:

```
import pacote.Classe;
```

Usando Pacotes

- Os pacotes ficam armazenados em subdiretórios do sistema de arquivos ou dentro de arquivos compactados zip ou jar

Usando Pacotes

- Os pacotes ficam armazenados em subdiretórios do sistema de arquivos ou dentro de arquivos compactados zip ou jar
- Observe que todos os arquivos de um pacote precisam estar em um subdiretório que coincida com o nome completo do pacote

Usando Pacotes

- Os pacotes ficam armazenados em subdiretórios do sistema de arquivos ou dentro de arquivos compactados zip ou jar
- Observe que todos os arquivos de um pacote precisam estar em um subdiretório que coincida com o nome completo do pacote
- Esses subdiretórios podem começar em qualquer diretório citado no caminho de classe (CLASSPATH)

Usando Pacotes

- Os pacotes ficam armazenados em subdiretórios do sistema de arquivos ou dentro de arquivos compactados zip ou jar
- Observe que todos os arquivos de um pacote precisam estar em um subdiretório que coincida com o nome completo do pacote
- Esses subdiretórios podem começar em qualquer diretório citado no caminho de classe (CLASSPATH)
- O CLASSPATH é uma variável de ambiente que indica onde as classes se encontram

Dicas para Codificação de Classes

- SEMPRE mantenha os atributos privados;

Dicas para Codificação de Classes

- SEMPRE mantenha os atributos privados;
- SEMPRE inicialize os dados;

Dicas para Codificação de Classes

- SEMPRE mantenha os atributos privados;
- SEMPRE inicialize os dados;
- Não use tipos básicos em demasia em uma classe;

Dicas para Codificação de Classes

- SEMPRE mantenha os atributos privados;
- SEMPRE inicialize os dados;
- Não use tipos básicos em demasia em uma classe;
- Use uma forma padrão de definição de classes, levando em consideração a visibilidade de seus membros.