


Laboratório de Engenharia de Software

# INF1636 – PROGRAMAÇÃO ORIENTADA A OBJETOS

Departamento de Informática – PUC-Rio

Ivan Mathias Filho  
[ivan@inf.puc-rio.br](mailto:ivan@inf.puc-rio.br)



Laboratório de Engenharia de Software

## Programa – Capítulo 3

- Ponteiros e Variáveis de Objeto
- Construtores e Blocos de Inicialização
- Envio de Mensagens em Java
- Variáveis e Métodos Estáticos

© LES/PUC-Rio 2

## Programa – Capítulo 3

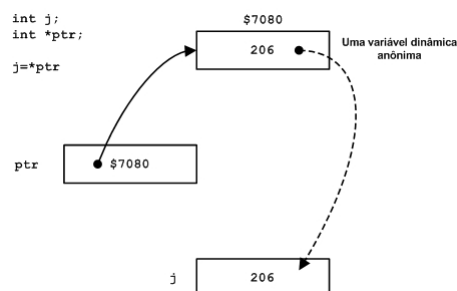


- **Ponteiros e Variáveis de Objeto**
- Construtores e Blocos de Inicialização
- Envio de Mensagens em Java
- Variáveis e Métodos Estáticos

## Ponteiros



- Um ponteiro é um tipo de dado cujo domínio é composto por endereços de memória e o valor **null**;
- O valor **null** não é um endereço de memória válido;
- Ele indica que, em dado momento, o ponteiro não pode ser usado para referenciar célula de memória alguma.



## Ponteiros em Java (1)



- A linguagem **Java** não possui ponteiros para tipos primitivos;
- Para manipular objetos em Java é preciso declarar **variáveis de objetos**;
- Uma **variável de objeto** é uma referência para um objeto;



© LES/PUC-Rio

5

## Ponteiros em Java (2)



- Enquanto um ponteiro em **C++** referencia uma área de memória, uma referência em **Java** referencia instâncias de classes;
- Como consequência, não se pode realizar operações aritméticas sobre referências em **Java**.

© LES/PUC-Rio

6

## Criação de objetos



- A simples declaração de uma variável de objeto não é suficiente para a criação de um objeto.

```
// v não referencia objeto algum
// o valor inicial de v é null
```

```
Vetor v=null;
```

- A criação de um objeto deve ser feita explicitamente, por meio do operador **new** (palavra reservada de **Java**).

```
// criação de um objeto
```

```
v=new Vetor();
```

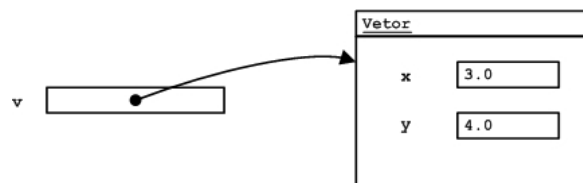
© LES/PUC-Rio

7

## Alocação de objetos



- O operador **new** aloca o objeto na área dinâmica de memória (**heap**) e retorna uma referência desse objeto;
- A referência é então atribuída à variável de objeto **v**.



© LES/PUC-Rio

8

## Atribuição de variáveis de objeto

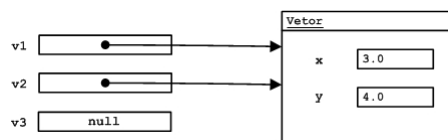


- Como as variáveis de objeto são referências, as operações de atribuição entre elas não criam novos objetos.

```
Vetor v1,v2,v3;

v1=new Vetor();
v2=v1;
```

- Após a execução da atribuição **v2=v1**, ambas as variáveis irão referenciar o mesmo objeto.



© LES/PUC-Rio

9

## Inicialização de variáveis de instância




- É uma boa prática de programação inicializar as variáveis de instância de uma classe;
- Caso isso não seja feito, o compilador Java irá atribuir valores default às variáveis não inicializadas explicitamente. Isso será feito da seguinte maneira:
  - Variáveis numéricas (**byte**, **int**, **double** e etc.) serão inicializadas com o valor **zero**;
  - Variáveis do tipo **char** serão inicializadas com **\u0000**;
  - Variáveis do tipo **boolean** serão inicializadas com **false**;
  - Variáveis de objeto serão inicializadas com **null**.

© LES/PUC-Rio

10

Programa – Capítulo 3




Laboratório de Engenharia de Software

- Ponteiros e Variáveis de Objeto
- **Construtores e Blocos de Inicialização**
- Envio de Mensagens em Java
- Variáveis e Métodos Estáticos

© LES/PUC-Rio

11

Construtores



Laboratório de Engenharia de Software

- Um construtor é um método especial que é executado todas as vezes que um objeto é criado;
- Ele é normalmente usado para fins de inicialização;
- Características de um construtor:
  - O nome de um construtor tem de ser igual ao nome da classe a qual ele pertence;
  - Um construtor possui zero ou mais parâmetros;
  - A chamada de um construtor é feita exclusivamente por meio do operador **new**;
  - Um construtor não retorna valor algum, embora seja incorreto usar o tipo **void** para indicar este fato.

© LES/PUC-Rio

12

## Exemplo de construtor



```
public class Contribuinte {
    private String nome;
    private String endereco;
    private String dtNasc;
    private long cpf;

    public Contribuinte() { // construtor
        nome="João da Silva";
        dtNasc="01/01/1900";
    }

    public String getCpf() {
        // o código não é relevante
    }
}
```

© LES/PUC-Rio

13

## Sobrecarga de método



- Sobrecarregar (*overloading*) um método significa definir vários métodos com o mesmo nome, mas com diferentes parâmetros.

```
public class Teste {
    public void umMetodo() {
        // código irrelevante
    }
    public int umMetodo(int x) {
        return 2;
    }
    public void umMetodo(String y,char z) {
        // código irrelevante
    }
}
```

© LES/PUC-Rio

14

## Sobrecarga de construtor



- Como qualquer outro método, o construtor também pode ser sobrecarregado;
- Caso nenhum construtor seja definido, o compilador **Java** fornece um construtor default para a classe;
- O construtor default não possui parâmetros;
- Embora um construtor default possa ser inserido implicitamente pelo compilador **Java**, pode-se defini-lo explicitamente.

## Construtor *default* - Exemplo



```
public class Contribuinte {  
    private String nome;  
    private String endereco;  
    private String dtNasc;  
    private long cpf;  
  
    public Contribuinte() { // construtor default  
        nome="João da Silva";  
        dtNasc="01/01/1900";  
    }  
  
    public String getCpf() {  
        // o código não é relevante  
    }  
}
```



## Omissão do construtor *default*



- Caso um construtor diferente do default seja definido, o compilador Java **NÃO** irá fornecer implicitamente um construtor default;
- Nesse caso, a invocação do construtor default, por meio do operador **new**, irá causar um erro de compilação.

```
public class Contribuinte {
    public Contribuinte(String n) {
        nome=n;
    }
}

...
{
    Contribuinte c=new Contribuinte(); // erro
}
```

© LES/PUC-Rio

17

## Blocos de inicialização



- Um bloco inicialização é um bloco de código não-nomeado definido no escopo de uma classe;
- Uma classe pode ter vários blocos de inicialização;
- Os blocos de inicialização são executados sempre que um objeto da classe na qual eles foram definidos é criado.

```
public class Contribuinte {
    // variáveis


    public Contribuinte() {
        nome="João da Silva";
        dtNasc="01/01/1900";
    }
    {
        cpf=0;
    }
}
```

© LES/PUC-Rio

18

Laboratório de Engenharia de Software

## Sequência de eventos na criação de objetos




- Pode-se observar a seguinte sequência de eventos após a criação de um objeto:
  - Todas as variáveis de instância são inicializadas, na ordem em que foram declaradas;
  - Todos os blocos de inicialização são executados, na ordem em que foram definidos;
  - O construtor invocado pelo operador **new** é executado.

© LES/PUC-Rio

19

Laboratório de Engenharia de Software

## Programa – Capítulo 3



- Ponteiros e Variáveis de Objeto
- Construtores e Blocos de Inicialização
- **Envio de Mensagens em Java**
- Variáveis e Métodos Estáticos

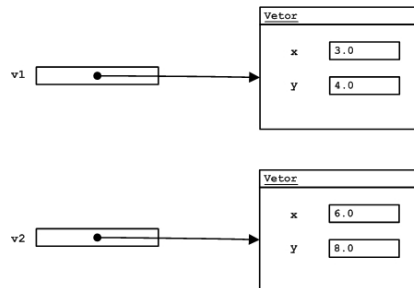
© LES/PUC-Rio

20

## Envio de mensagens em Java



- Objetos distintos ocupam áreas de memória distintas.

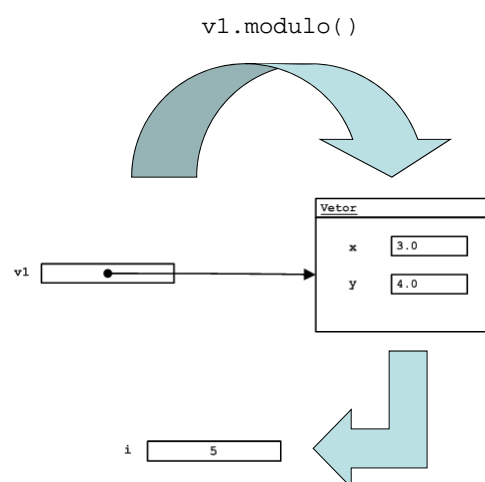


- Logo, quando se envia uma mensagem para um objeto deve-se fazê-lo por meio de uma referência para o mesmo;
- Em Java, isso é feito por uma **variável de objeto**.

© LES/PUC-Rio


21


## Execução do Comando `i=v1.modulo()`



© LES/PUC-Rio

22

Laboratório de Engenharia de Software	<b>Consequências</b>	
	<ul style="list-style-type: none"><li>• Para enviar mensagens para um objeto é necessário ter acesso a uma referência para o mesmo;</li><li>• Isso nos obriga a trafegar referências nas trocas de mensagens (parâmetros);</li><li>• Caso não haja pelo menos uma referência para um objeto ele se torna inacessível;</li><li>• Em Java, não é responsabilidade do programador liberar as áreas de memória inacessíveis;</li><li>• Uma thread que é executada em background, chamada <b>Garbage Collector</b>, reclama as áreas de memórias não-referenciadas e as devolve para a área de memória livre do heap.</li></ul>	
	© LES/PUC-Rio	23

Laboratório de Engenharia de Software	<b>Programa – Capítulo 3</b>	
	<ul style="list-style-type: none"><li>• Ponteiros e Variáveis de Objeto</li><li>• Construtores e Blocos de Inicialização</li><li>• Envio de Mensagens em Java</li><li>• <b>Variáveis e Métodos Estáticos</b></li></ul>	
	© LES/PUC-Rio	24

## Variáveis estáticas



- Em algumas situações é conveniente que um grupo de objetos de uma mesma classe tenha acesso a uma área de memória compartilhada;
- Como exemplo, pode-se citar a situação em que os objetos de uma classe têm de estar a par do número de objetos instanciados a partir dessa classe;
- Nesse caso, cada objeto instanciado deve somar uma unidade a uma variável que guarda o número de objetos instanciados até o momento;
- A linguagem Java oferece tal possibilidade por meio de **variáveis estáticas**.

© LES/PUC-Rio

25

## Variáveis estáticas – Exemplo (1)



- Uma variável estática pertence à classe, e não aos objetos da classe;
- Existe apenas uma instância de uma variável estática, independentemente do número de objetos instanciados.

```
public class Empregado {
    private int id;
    private double salario;
    private static int numEmp=0; // variável estática

    public Empregado() {
        numEmp++; //incrementa o número de empregados criados
    }
    public int getNumEmp() {
        return numEmp;
    }
}
```

© LES/PUC-Rio

26

## Variáveis estáticas – Exemplo (2)

Laboratório de Engenharia de Software

LES

```

public class Ex07 {
    public static void main(String[] args) {
        Empregado e1,e2,e3;

        e1=new Empregado();
        System.out.println(e1.getNumEmp());

        e2=new Empregado();
        System.out.println(e2.getNumEmp());

        e3=new Empregado();
        System.out.println(e3.getNumEmp());
    }
}

```

O valor 1 será exibido

O valor 2 será exibido

O valor 3 será exibido

© LES/PUC-Rio

27

## Variáveis estáticas - Ilustração

Laboratório de Engenharia de Software

LES

Os objetos compartilham uma única cópia de uma variável estática.

Cada objeto possui a sua própria área de memória para uma variável de instância.

```

classDiagram
    class Empregado {
        static int numEmp
        int id
        double salario
        setId()
    }
    Empregado "1" -- "3" Empregado : shared static variable

```

© LES/PUC-Rio

28

## Métodos estáticos



- Um método estático, ou método de classe, é um método que pertence à classe;
- Um método estático não pode manipular variáveis de instância, apenas as variáveis estáticas;
- A especificação da linguagem Java diz que um método de classe **deve** ser invocado sem referência para um objeto particular;
- Dessa forma, um método de classe deve ser invocado por meio do nome da classe.

```
a=Math.sqrt(Math.pow(x,2.0)+Math.pow(y,2.0));
```

© LES/PUC-Rio

29

## Referência this



- Todo objeto mantém uma referência para si mesmo;
- Esta referência é feita pela palavra reservada **this**.

```
public Contribuinte() { // construtor
    this.nome="João da Silva";
    this.dtNasc="01/01/1900";
}
```

- No exemplo acima, o uso da referência **this** é desnecessário, pois sempre que se envia uma mensagem para um objeto deve-se fazê-lo por meio de uma referência para o mesmo;
- Logo, cada método de instância referencia, implicitamente, as variáveis de instância do objeto sobre o qual ele foi aplicado.

© LES/PUC-Rio

30

## Referência this – Exemplo (1)



- O uso da referência **this** é fundamental quando se deseja passar o próprio objeto como argumento em uma troca de mensagem.

```
public class Empregado {
    public void alocaGerente(Projeto p) {
        p.registraGerente(this);
    }
}

public class Projeto {
    Empregado gerente;

    public void registraGerente(Empregado e) {
        gerente=e;
    }
}
```

© LES/PUC-Rio

31

## Referência this – Exemplo (2)



```
public class Ex05 {
    public static void main(String[] args) {
        Projeto p=new Projeto();
        Empregado e=new Empregado();

        e.alocaGerente(p);
    }
}
```

Um novo projeto e um novo empregado são criados

A mensagem **alocaGerente** é enviada para o empregado (e) passando o projeto (p) como argumento

```
public class Empregado {
    public void alocaGerente(Projeto p) {
        p.registraGerente(this);
    }
}
```

O empregado envia a mensagem **registraGerente** para o projeto (p) passando ele próprio (this) como argumento

© LES/PUC-Rio

32



## Referência this – Exemplo (3)



```
public class Projeto {
    Empregado gerente;

    public void registraGerente(Empregado e) {
        gerente=e;
    }
}
```

O projeto registra o empregado recebido como parâmetro (e) como gerente do projeto

## Referência this – Exemplo (4)



- Pode-se usar a referência **this** para invocar um construtor a partir de outro construtor da mesma classe.

```
public class Vetor {
    // variaveis
    private double x=3.0;
    private double y=4.0;

    // métodos
    public Vetor(double a,double b) {
        x=a;
        y=b;
    }

    public Vetor() {
        this(0.0,0.0);
    }
}
```

O construtor default chama o construtor **Vetor(double, double)** passando o ponto (0.0,0.0) como argumento