

# Classes e Objetos (Revisão)

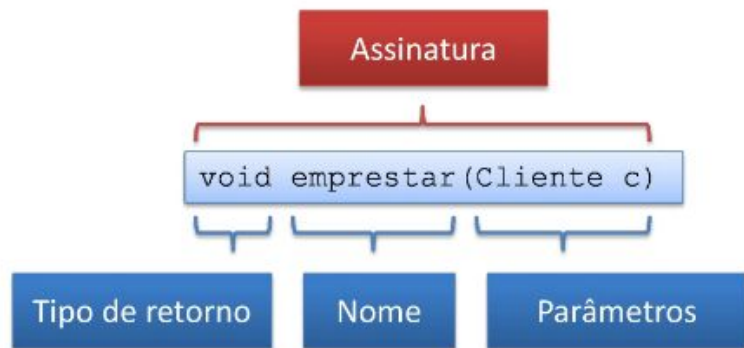
Instituto Metrópole Digital

Disciplina: IMD0040 - Linguagem de Programação II

Docente: Emerson Alencar



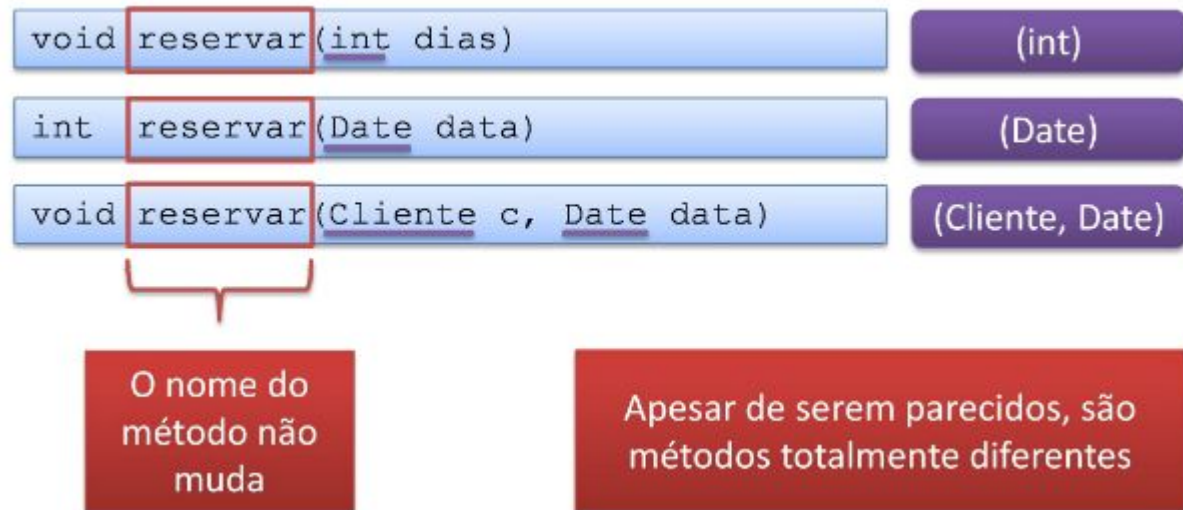
# Assinatura de um método



- Se o método não retornar valores, é utilizado o void;
- Um método pode ter zero ou mais parâmetros, e todo parâmetro deve ter um tipo definido.

# Sobrecarga de Métodos

- Sobrecarregar um método significa criar outros métodos com mesmo nome, mas com assinatura diferente



# Sobrecarga de Métodos

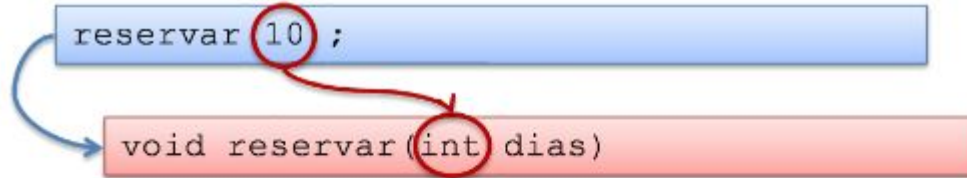


Diagram illustrating method call and definition for an integer parameter:

- Call: `reservar 10 ;` (The value `10` is circled in red)
- Definition: `void reservar (int dias)` (The parameter `int` is circled in red)
- A blue arrow points from the call to the definition.
- A red arrow points from the circled `10` to the circled `int`.

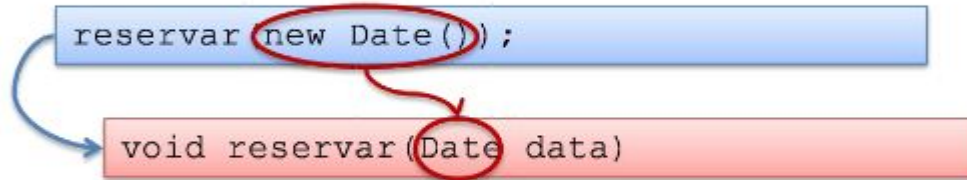


Diagram illustrating method call and definition for a `Date` object parameter:

- Call: `reservar new Date() ;` (The expression `new Date()` is circled in red)
- Definition: `void reservar (Date data)` (The parameter `Date` is circled in red)
- A blue arrow points from the call to the definition.
- A red arrow points from the circled `new Date()` to the circled `Date`.



Diagram illustrating method call and definition for multiple parameters:

- Call: `reservar new Cliente(), new Date() ;` (Both `new Cliente()` and `new Date()` are circled in red)
- Definition: `void reservar (Cliente c, Date data)` (Both `Cliente` and `Date` are circled in red)
- A blue arrow points from the call to the definition.
- Red arrows point from the circled `new Cliente()` to the circled `Cliente`, and from the circled `new Date()` to the circled `Date`.

# Criando e Manipulando Objetos

- Um objeto é sempre instância de uma classe
- para instanciar objetos, é utilizando o new

```
Livro livro1 = new Livro();  
Cliente cliente1 = new Cliente();
```

- O objeto possui acesso ao que foi definido na sua estrutura (classe) através do “.”

```
livro1.titulo = "Aprendendo Java";  
livro1.emprestar(cliente1);
```

# Criando e Manipulando Objetos

- Cada objeto criado com o new é único
- Os atributos de objetos diferentes pertencem apenas ao objeto

```
Livro livro1 = new Livro();  
livro1.isbn = "1234";
```

```
Livro livro2 = new Livro();  
livro2.isbn = "4321";
```

```
Livro livro3 = new Livro();  
livro3.isbn = "1212";
```

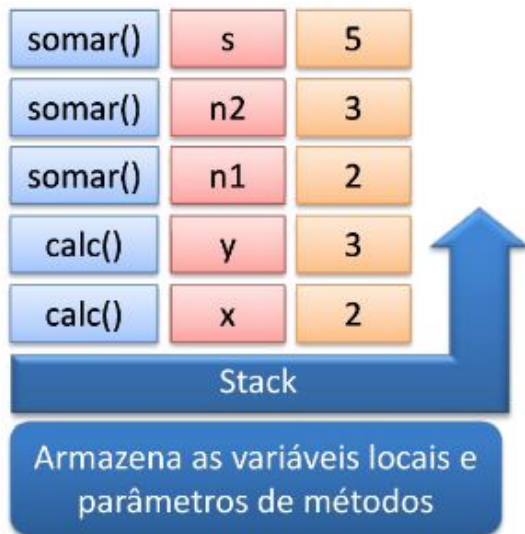
Cada livro possui o  
seu próprio ISBN

# Objetos e Referência

- Uma variável cujo tipo é uma classe não guarda o objeto diretamente
- A variável guarda uma referência ao objeto
- O new aloca uma área de memória e retorna a referência da área de memória alocada
- As variáveis declaradas em métodos são criadas numa área de memória chamada **stack**
- Os Objetos são criados numa área de memória chamada **heap**

# Como Funciona a Stack

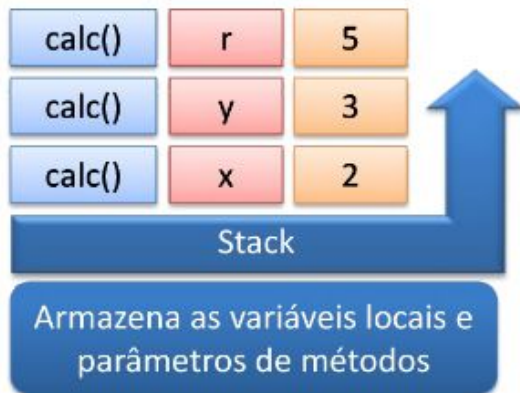
```
void calc() {  
    int x = 2;  
    int y = 3;  
    int r = somar(x, y);  
}  
  
int somar(int n1, int n2) {  
    int s = n1 + n2;  
    return s;  
}
```





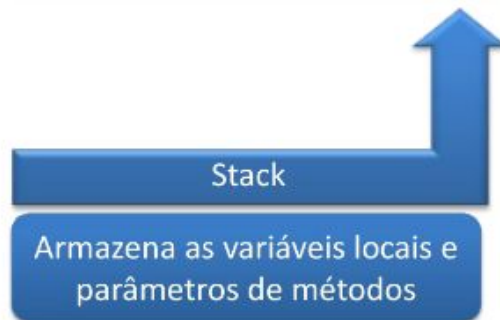
# Como Funciona a Stack

```
void calc() {  
    int x = 2;  
    int y = 3;  
    int r = somar(x, y);  
}  
  
int somar(int n1, int n2) {  
    int s = n1 + n2;  
    return s;  
}
```



# Como Funciona a Stack

```
void calc() {  
    int x = 2;  
    int y = 3;  
    int r = somar(x, y);  
}  
  
int somar(int n1, int n2) {  
    int s = n1 + n2;  
    return s;  
}
```



# Como funciona o Heap

```
double n = 10.0;  
Computador comp = new Computador();  
Telefone tel = new Telefone();
```

O operador *new()* cria um objeto no *heap*

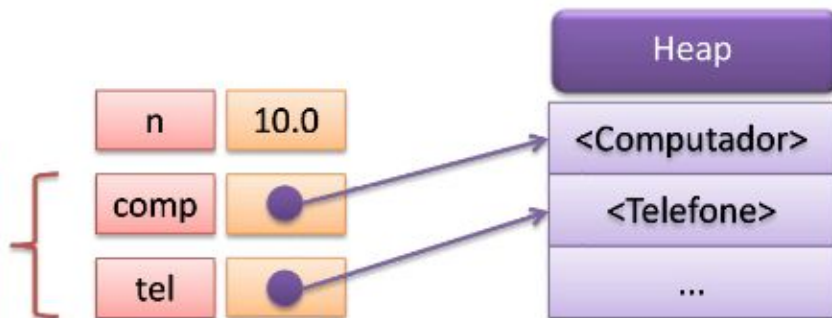
O operador "=" faz a ligação da variável com a referência do objeto



# Como funciona o Heap

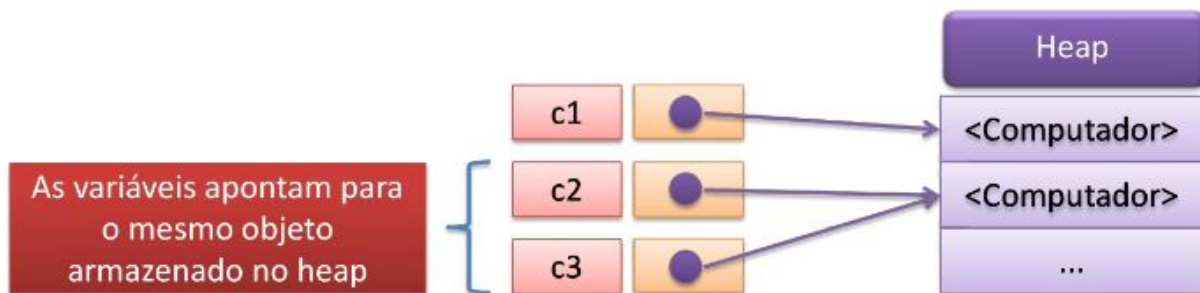
```
double n = 10.0;  
Computador comp = new Computador();  
Telefone tel = new Telefone();
```

A conteúdo da variável é  
uma referência (ponteiro)  
para o objeto no heap



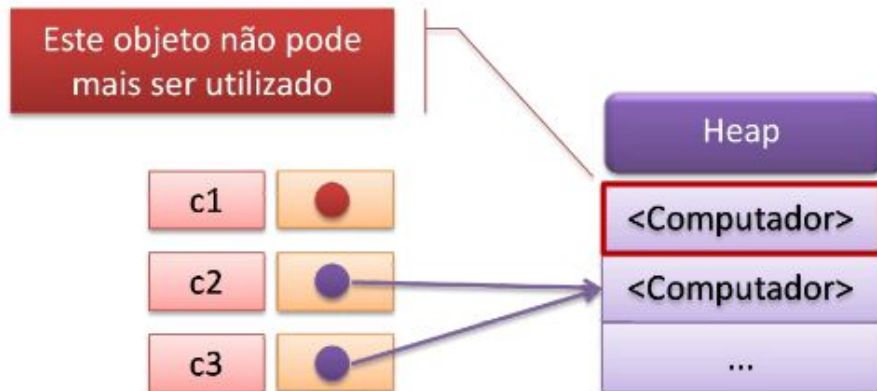
# Como funciona o Heap

```
Computador c1 = new Computador();  
Computador c2 = new Computador();  
Computador c3 = c2;  
c1 = null;
```



# Como funciona o Heap

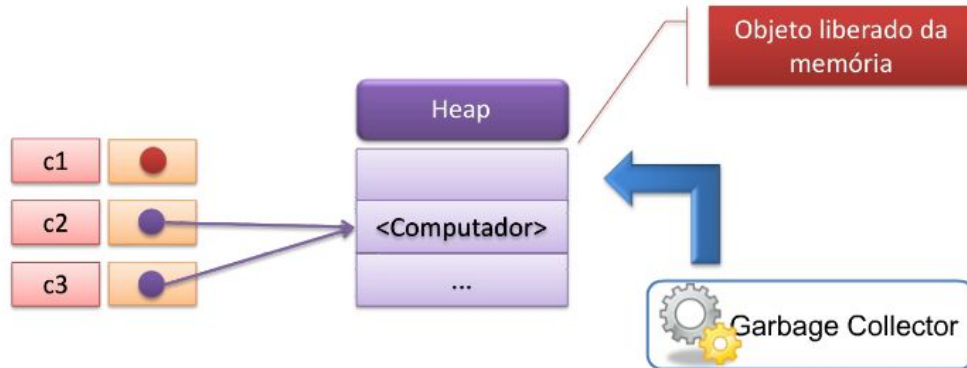
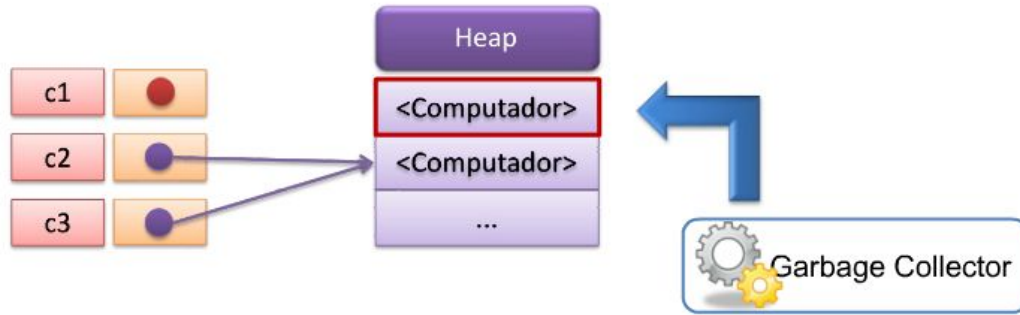
```
Computador c1 = new Computador();  
Computador c2 = new Computador();  
Computador c3 = c2;  
c1 = null;
```



# Garbage Collector

- Serviço da JVM que executa em segundo plano
- Procurar objetos no heap que não são mais utilizados pela aplicação e os remove
- Não pode ser controlado pelo desenvolvedor

# Garbage Collector





# Operador this

- Normalmente não é obrigatório
- Usado em basicamente em duas situações
  - Diferenciar um atributo do objeto de um argumento do método
  - Fornecer a referência do próprio objeto para outro método

# Operador *this*

```
public class Circulo {  
    private double raio;  
  
    public void setRaio(double raio) {  
        this.raio = raio;  
    }  
}
```

Atributo da classe

Parâmetro do  
método

# Modificadores de Acesso

- O acesso a atributos e métodos é restringido através do uso de modificadores
- Alguns modificadores
  - `private`
    - Visível apenas para a classe que o declara
  - `public`
    - Visível a todas as classes

# Atributos e Métodos

- Marcar um atributo ou método como *private* esconde o atributo de quem usa a classe
- É interessante marcar métodos como *private* quando este é um método auxiliar da classe, que não deve ser acessível externamente

# Atributos e Métodos

```
class Livro {  
    private String isbn;  
    private int numPaginas;  
  
    public void emprestar(Cliente c) {  
        ...  
    }  
  
    public void devolver() {  
        ...  
    }  
}
```

Atributos

Métodos

# Atributos e Métodos

- Apesar de não ser regra, normalmente:
  - Atributos são declarados como `private`
  - Métodos são declarados como `public`
- Esta abordagem faz sentido, já que o ideal é que os objetos colaborem através de troca de mensagens (chamadas de métodos), e não através de manipulação direta de atributos.

# Classes

- Quase sempre, classes também são declaradas como public
  - Apenas uma classe definida como public pode existir num arquivo Java
  - O nome do arquivo deve ser igual ao nome da classe definida como public
- Classes não declaradas como public são chamadas de inner classes

# Princípio do Encapsulamento

- Encapsular é esconder detalhes de funcionamento do programa
- É fundamental para permitir que o programa seja suscetível a mudanças



# Métodos Getters e Setters

- Quando os atributos são declarados como private, getters e setters podem ser usados
- Getters
  - Usados para expor os valores de atributos
- Setters
  - Usados para alterar os valores de atributos

# Exemplos de Getters e Setters

```
class SalaDeAula {  
    private int numAlunos;  
  
    public int getNumAlunos() {  
        return this.numAlunos;  
    }  
  
    public void setNumAlunos(int numAlunos) {  
        this.numAlunos = numAlunos;  
    }  
}
```

Getter

Setter

# Mais Sobre os Getters e Setters

- Benefícios
  - Protegem os atributos
  - Evitam a mudança de código em vários lugares
- Não utilize getters e setters quando não for necessário

# Assinatura dos Getters e Setters

- A assinatura dos getters e setters segue um padrão

Atributo	Getter	Setter
numConta	getNumConta()	setNumConta()
saldo	getSaldo()	setSaldo()
ativo	isAtivo()	setAtivo()



Para atributos booleanos, o padrão do getter é **isXXX()**, mas **getXXX()** também pode ser utilizado

# Construtores

- O construtor de uma classe é chamado toda vez que um objeto da classe é instanciado
- O construtor possui o mesmo nome da classe

```
public class SalaDeAula {  
    public SalaDeAula() {  
        ...  
    }  
}
```

Construtor

```
SalaDeAula s = new SalaDeAula();
```

Invoca o  
construtor

# Construtor Padrão

- Quando o construtor não é fornecido, o Java fornece um construtor padrão (sem parâmetros)

```
public class SalaDeAula {  
}
```



```
public class SalaDeAula {  
    public SalaDeAula() {  
    }  
}
```

Construtor  
padrão

# Construtor com Parâmetros

```
public class SalaDeAula {  
    private int numAlunos;  
  
    public SalaDeAula(int numAlunos) {  
        this.numAlunos = numAlunos;  
    }  
}
```

Recebe um int  
como parâmetro

```
SalaDeAula s = new SalaDeAula(20);
```

Invoca o  
construtor

# Sobrecarga de Construtores

```
public class SalaDeAula {  
    private int numAlunos;  
    private boolean temArmario;  
  
    1 { public SalaDeAula() {  
        this.temArmario = true;  
    }  
  
    2 { public SalaDeAula(int numAlunos) {  
        this();  
        this.numAlunos = numAlunos;  
    }  
}
```

```
SalaDeAula s1 = new SalaDeAula();  
SalaDeAula s2 = new SalaDeAula(20);
```

Invoca o construtor 1

Invoca o construtor 2



# Atributos e Métodos Estáticos

- Algumas vezes, atributos e/ou métodos podem não estar atrelados a um objeto específico, mas sim à classe
- Atributos ou métodos da classe são assim definidos através do modificador ***static***

# Declarando Elementos Estáticos

Os valores dos atributos estáticos são compartilhados entre todas as instâncias da classe

```
public class ContaBancaria {  
    private static String banco = "JavaBank";  
  
    private static String getBanco() {  
        return ContaBancaria.banco;  
    }  
}
```

Métodos estáticos só podem acessar atributos ou outros métodos que também sejam estáticos

# Invocando Elementos Estáticos

```
String banco = ContaBancaria.getBanco();
```

O acesso é feito utilizando diretamente a classe. Não é necessário criar um objeto

# Criação de Constantes

- Atributos estáticos são uma forma bastante usada para criar constantes no Java

```
public class Constantes {  
    public static final int VERSAO = 1;  
}
```

Todas as classes  
possuem acesso

Pertence à  
classe, e não ao  
objeto

Valor fixo

```
int versao = Constantes.VERSAO;
```

# O Bloco static

- Uma classe pode ter um bloco static (apenas um)
- O bloco static é executado quando a classe é referenciada pela primeira vez
  - Inicializar atributos estáticos
  - Executar um código antes que a classe seja utilizada

```
public class MinhaClasse {  
    private static int x;  
  
    static {  
        x = 10;  
        Programa.inicializar();  
    }  
}
```

O bloco só é  
executado uma vez

# Métodos Estáticos: A Classe System

- A classe **System** do Java possui diversos métodos estáticos úteis

Método	Descrição
System.in	Entrada padrão
System.out	Saída padrão
System.exit(int)	Termina a JVM
System.currentTimeMillis()	Retorna o tempo atual em ms

**Por hoje é só...**