


Fundamentos de Java

Modificadores de Acesso, Construtores e Elementos Estáticos




Tópicos Abordados



- Modificadores de acesso
 - Atributos e métodos
 - Classes
- Princípio do encapsulamento
- Métodos getters e setters
- Construtores
 - Padrão e com parâmetros
 - Sobrecarga
- Atributos e métodos estáticos
 - Criação de constantes
 - Bloco *static*
- A classe *System*

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 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 objetos colaborem através de troca de mensagens (chamadas de métodos), e não através da 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 *inner classes*

Classes



```
public class Livro {  
    ...  
}
```



Livro.java

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



- Construtores podem receber parâmetros da mesma forma que métodos

```
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;  
    }  
}
```

O **this()** invoca outro
construtor da classe

```
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




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

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

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*
- 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