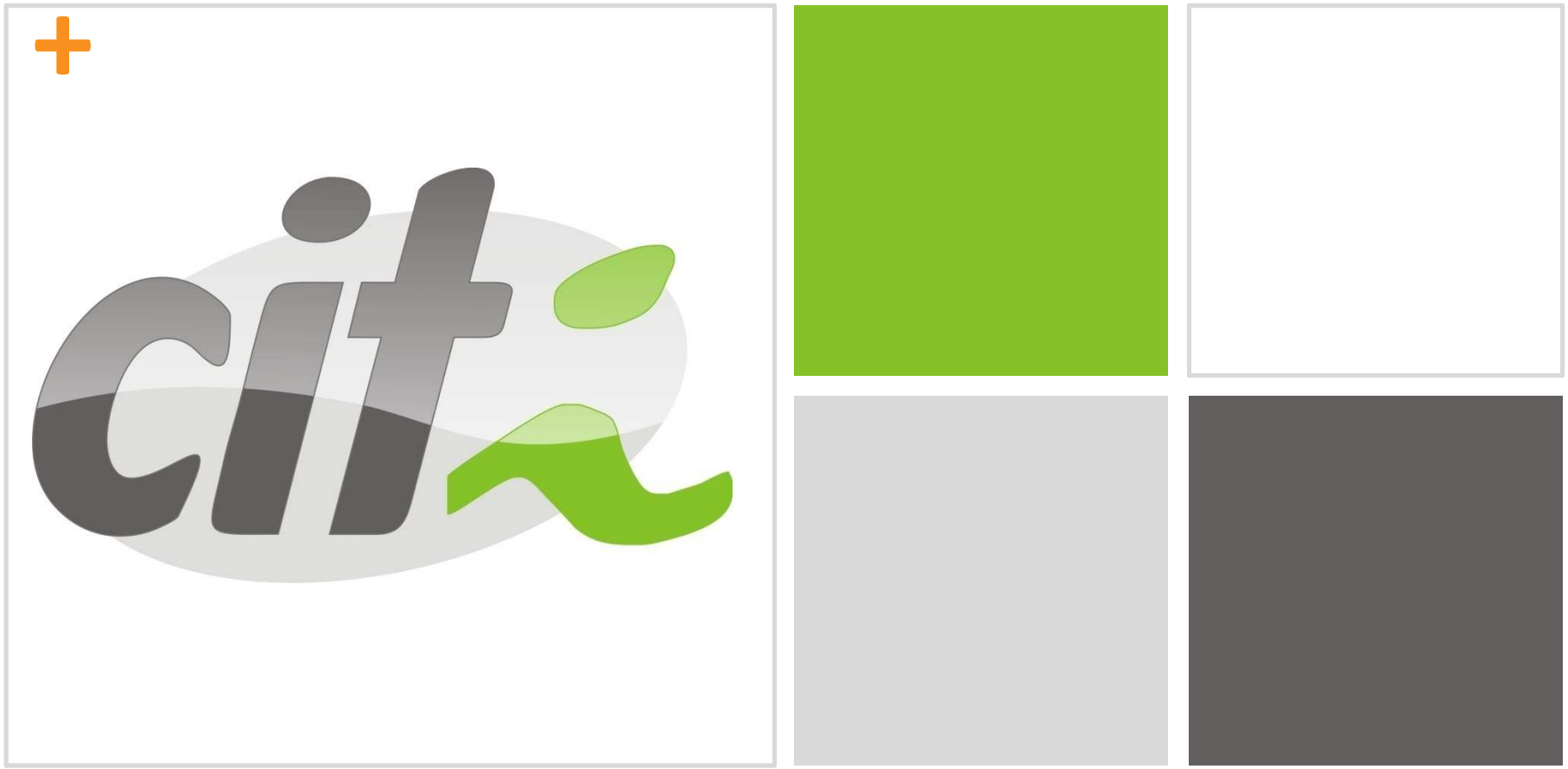


Java Básico

Igor Ebrahim (ies@cin.ufpe.br)

Módulo 5

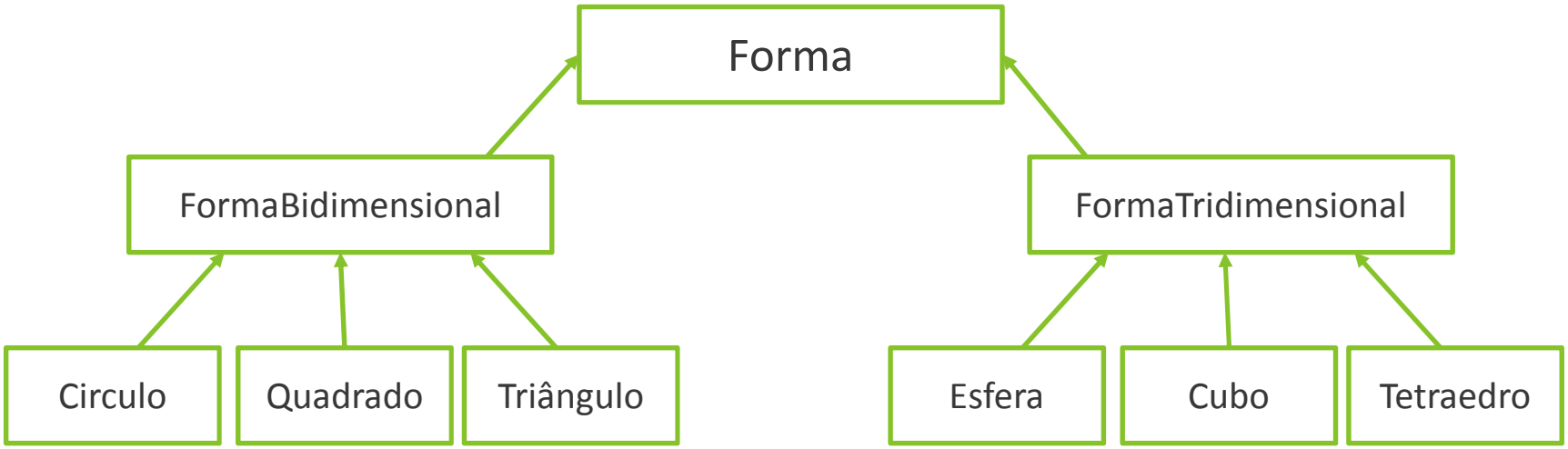
Herança, polimorfismo e ligação dinâmica

Herança

- Técnica de programação orientada a objetos usada para organizar e criar classes para reuso;
- Possibilita derivar uma classe de outra já existente;
- A classe existente é chamada de super-classe ou classe base;
- A classe derivada é chamada de sub-classe;
- A sub-classe herda as características da super-classe, ou seja, a sub-classe herda os métodos e os atributos definidos pela super-classe;

Herança

- A herança cria uma relação “é um”, isto é, a sub-classe é uma versão mais específica que a super-classe;’



Herança

- O programador pode implementar uma classe derivada como for preciso, adicionando novos atributos ou métodos, ou até mesmo, modificando os herdados;
- Reuso de software é uma vantagem no uso de herança;

Criando Sub-classes

- Em Java, usa-se a palavra reservada extends para estabelecer uma relação de herança:

```
class FormaBidimensional extends Forma
{
   // conteúdo da classe
}
```

Modificadores de Acesso

- Modificadores de acesso afetam a maneira como atributos e métodos podem ser usadas nas sub-classes;
- Atributos e métodos declarados privados (private) não podem ser referenciados diretamente em sub-classes;
- Atributos e métodos declarados públicos (public) podem ser referenciados diretamente em sub-classes – mas atributos públicos ferem o princípio do encapsulamento;

Modificadores de Acesso

- O modificador protected permite à sub-classe referenciar uma variável ou métodos da classe herdada diretamente;
- Isso oferece um maior encapsulamento do que o uso do public, mas não tanto quanto o private;
- Um atributo (ou método) protected é visível a qualquer classe dentro do mesmo pacote da super-classe;

super

- Construtores não são herdados, mesmo que sejam public;
- Ainda assim, geralmente precisamos usar o contrutor da super-classe para inicializar a parte herdada;
- A referência super pode ser usada para referenciar a classe herdada ou para invocar o seu construtor.

super

- O construtor da sub-classe é responsável por chamar o construtor da super-classe;
- A primeira linha do construtor da sub-classe deve conter a referência super para chamar o construtor da super-classe;
- A referência super também pode ser usada para referenciar outros atributos ou métodos definidos na super-classe.

Herança Múltipla

- Uma classe pode herdar apenas de apenas uma outra classe;
- Colisões, como duas super-classes terem o mesmo nome de um atributo, têm que ser resolvidas;
- Java não suporta herança múltipla de classes
  - Mas permite herança múltipla de interfaces (veremos em breve)

Overriding Métodos

- Uma sub-classe pode sobrescrever (override) a definição de um método herdado;
- O novo método tem que possuir a mesma assinatura do método herdado, mas pode ter um corpo completamente diferente;
- O tipo do objeto que executa o método é que determina qual das versões é realmente invocada;
- Se um método for definido na super-classe com o modificador de acesso final, ele não pode ser sobrescrito;

Overloading vs. Overriding

- Overloading (sobrecarregar) → múltiplos métodos com o mesmo nome e na mesma classe, mas com assinaturas diferentes;
- Overriding (sobrescrever) → métodos em classes diferentes (um na sub-classe e outro na super-classe) que possuem a mesma assinatura;
- Overloading permite ao programador definir operações semelhantes, mas de diferentes formas e para parâmetros diferentes;
- Overriding permite ao programador definir operações semelhantes, mas de diferentes formas e para diferentes tipos de objetos.

A Classe Object

- A classe Object é definida no pacote java.lang da biblioteca básica de Java;
- Todas as classes são derivadas da classe Object;
- Se uma classe não é explicitamente definida como filha de uma outra classe já existente, Java assume que ela é filha direta de Object;

A Classe Object

- A classe Object possui alguns métodos que podem ser úteis, que são herdados por todas as classes;
- Por exemplo, o método toString é definido em Object;
- Toda vez que definimos o método toString, nós estamos sobrescrevendo o método herdado;
- O método toString na classe Object é definido para retornar uma String que contém o nome da classe do objeto, além de outras informações;

A Classe Object

- O método equals da classe Object retorna true se duas referências forem “aliases”;
- Podemos sobrescrever o equals em qualquer classe para definir a igualdade de uma melhor maneira;
- Para verificar a igualdade entre duas String´s, devemos usar o método equals;
- Os programadores da classe String sobrescreveram o método equals herdado de Object em favor de seu melhor uso.

Herança

- Uma relação de herança bem feita pode contribuir bastante com a elegância, a manutenção e o reuso do software;
- Toda herança pode ser uma relação “é um”;
- Pense sempre adiante, na potencialidade de uma classe ser herdada.;

Herança

- Encontre características comuns entre as classes e empurre-as para cima na hierarquia;
- Sobrescreva métodos apropriadamente;
- Adicione novos atributos nas sub-classes, mas não redefina os herdados;

Herança

- Permita que cada classe gerencie seu próprio conteúdo; use o super para chamar o construtor da classe herdada;
- Use classes abstratas para representar conceitos gerais;
- Use os modificadores de acesso com cuidado para não violar o encapsulamento;

Polimorfismo

- Polimorfismo é um conceito de orientação objeto que nos permite criar versáteis designs de softwares

Binding (Ligação)

- Considere a seguinte chamada de método:

```
obj.facaIsto();
```

- Em algum ponto, essa chamada é ligada à definição do método;
- Em Java, esta ligação é feita dinamicamente, ou seja, em tempo de execução;
- Por isto chamamos de dynamic binding (ligação dinâmica)

Polimorfismo

- O termo polimorfismo significa “muitas formas”;
- Uma referência polimórfica é uma variável que pode se referir a diferentes tipos de objetos em diferentes instantes;
- Um método invocado através de uma referência polimórfica pode mudar de chamada para chamada;
- Todas as chamadas de método em Java são potencialmente polimórfica;

Polimorfismo

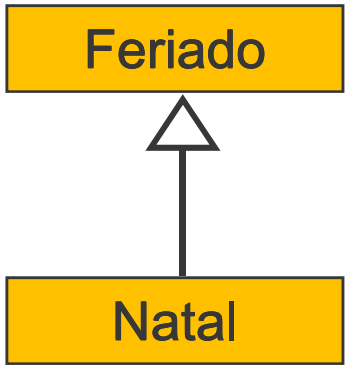
- Suponha que criamos a seguinte variável:

```
Emprego emp;
```

- Java permite a esta referência apontar a um objeto Emprego ou a qualquer outro objeto de tipo compatível;
- Essa compatibilidade pode ser estabelecida usando herança ou interfaces;
- Um uso cuidadoso de polimorfismo pode gerar elegantes e robustos softwares;

Referências e Herança

- Uma referência para um objeto pode apontar para um objeto de sua própria classe ou para um objeto relacionado a ele por herança;
- Por exemplo, se a classe Feriado é usada para derivar a classe Natal, então uma referência de Feriado pode apontar para um objeto Natal:




```
Feriado dia;
dia = new Natal();
```

Polimorfismo via Herança

- É o tipo do objeto que está sendo referenciado, não o tipo da referência, que determina qual método é chamado;
- Suponha que o classe Feriado tenha um método chamado celebrar, e que a classe Natal sobrescreve-a;
- Agora, considere a seguinte chamada:

```
dia.celebrar();
```

- Se dia refere-se a um objeto Feriado, então invoca-se a versão de celebrar de Feriado; Se dia refere-se a um objeto Natal, ele chama a versão de Natal

Verificação Dinâmica de Tipo

instanceof

- Podemos usar Casting ou o operador instanceof para fazer verificação de tipo:

```
Conta conta;
conta = new Poupanca("21.342-7");
((Poupanca) conta).renderJuros(0.01);
conta.imprimirSaldo();
```

Verificação Dinâmica de Tipo

instanceof

- Casts geram exceções quando instanceof retorna false;
- Casts são essenciais para verificação estática de tipos (compilação);

```
Conta c = this.procurar("123.45-8");
if (c instanceof Poupanca)
   ((Poupanca) c).renderJuros(0.01);
else
   System.out.print("Poupança inexistente!“);
```

Prática

1. Refatore os métodos equals de Cliente e Conta para que eles realizem verificação dinâmica de tipo (instanceof).