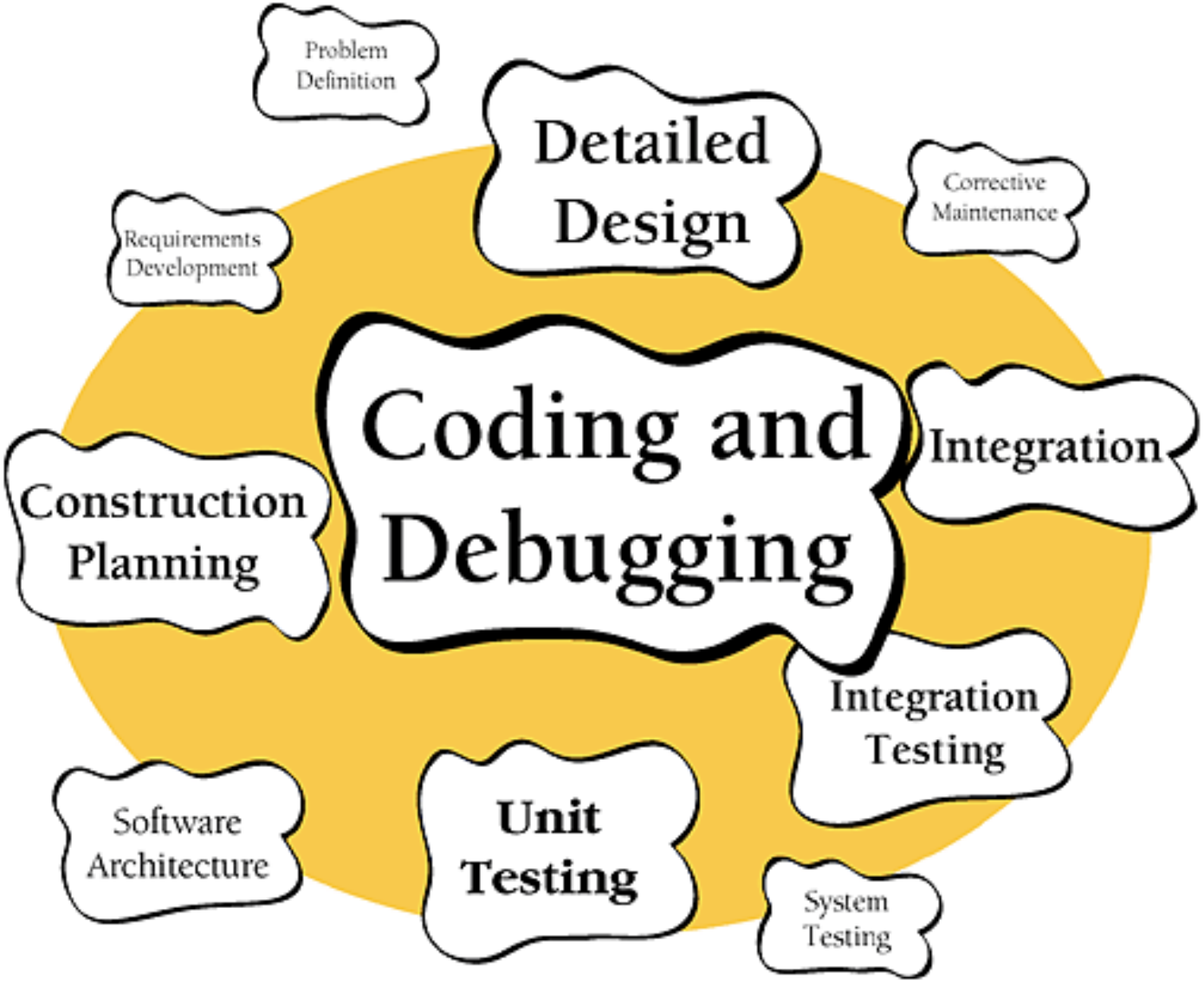
Visão Geral da Programação OO
Slides Baseados na Documentação de Jacques Sauvé (UFCG)

Universidade de Brasília

Rodrigo Bonifácio

17 de março de 2014

Programação é uma das atividades de construção de software


Problem Definition
Detailed Design
Corrective Maintenance
Requirements Development
Coding and Debugging
Integration
Construction Planning
Integration Testing
Software Architecture
Unit Testing
System Testing

Que envolve também o projeto detalhado do software, cujo objetivo é identificar boas abstrações de software

O progresso da abstração

1. Assembly: pequena abstração da máquina

O progresso da abstração

1. Assembly: pequena abstração da máquina
2. Linguagens imperativas: abstrações sobre Assembly
   - decomposição em termos da arquitetura do computador
   - não em termos do problema a ser solucionado
   - programas difíceis para escrever e manter

O progresso da abstração

1. Assembly: pequena abstração da máquina
2. Linguagens imperativas: abstrações sobre Assembly
   - decomposição em termos da arquitetura do computador
   - não em termos do problema a ser solucionado
   - programas difíceis para escrever e manter
3. Linguagens OO: abstrações associadas ao domínio do problema
4. Linguagens funcionais: $\lambda - \textit{calculus}$ como modelo computacional
5. Linguagens lógicas: lógica de predicados como modelo computacional

Cada paradigma tem sua forma de decomposição

Cada paradigma tem sua forma de decomposição, associada a maneira como abordamos para resolver um problema

- Saber decompor um sistema (design de software) em um paradigma é mais importante que dominar a sintaxe de uma linguagem específica naquele paradigma.

Cada paradigma tem sua forma de decomposição, associada a maneira como abordamos para resolver um problema

- Saber decompor um sistema (design de software) em um paradigma é mais importante que dominar a sintaxe de uma linguagem específica naquele paradigma. Dominar múltiplos paradigmas permite escolher a linguagem certa para um determinado problema.

Princípios de design e programação

Com isso, bons projetistas OO, por exemplo, possuem facilidade em transitar entre as linguagens OO; e conhecem, em profundidade, bons princípios de design e programação

Princípios de design e programação

Com isso, bons projetistas OO, por exemplo, possuem facilidade em transitar entre as linguagens OO; e conhecem, em profundidade, bons princípios de design e programação

- Abstração, Encapsulamento
- Polimorfismo
- Herança
- Reusabilidade de código

- Por outro lado, precisamos exercitar a decomposição OO usando uma linguagem. No caso deste curso, usaremos a linguagem Java.

Objetivos da aula

Introduzir a linguagem Java através de exemplos

Objetivos da aula

Introduzir a linguagem Java através de exemplos

Por que Java?

- Portabilidade

Objetivos da aula

Introduzir a linguagem Java através de exemplos

Por que Java?

- Portabilidade graças a JVM (*Write once run anywhere'*)
- Aceitação no mercado, com impacto direto em sua confiabilidade. Talvez Java, no futuro, tenha como maior força a confiabilidade (como a confiabilidade atual de linguagens como COBOL).
- Projeto da linguagem bem cuidadoso, suprindo algumas deficiências de C++

Objetivos da aula

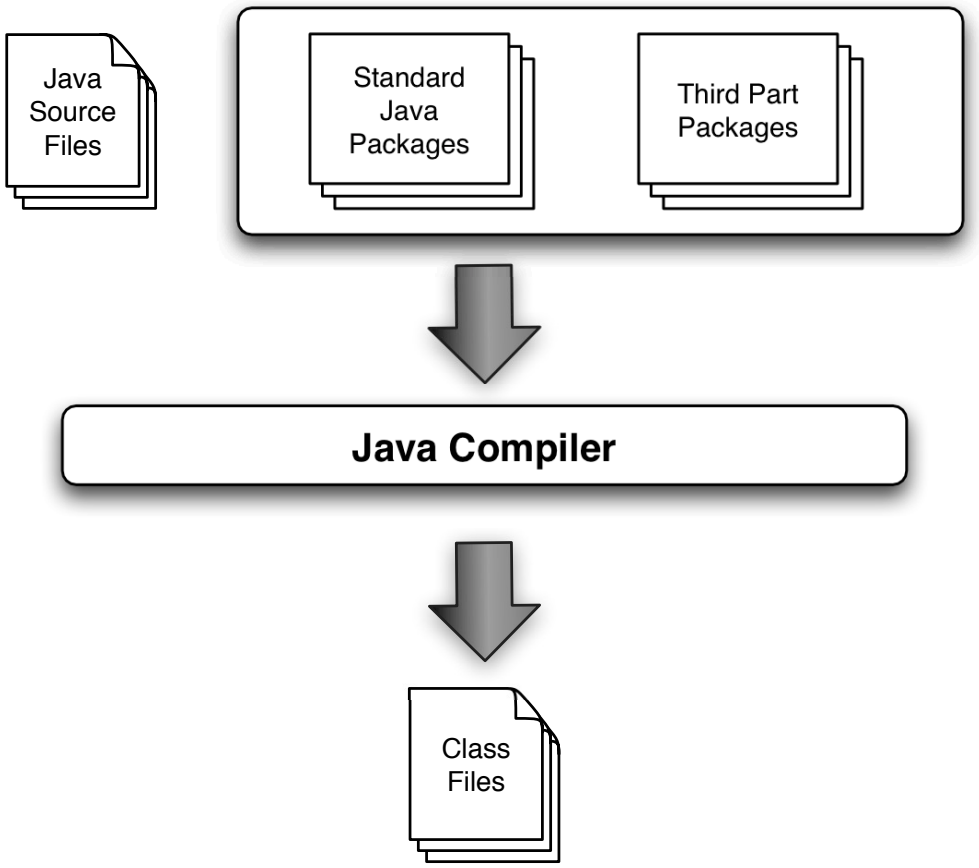
Introduzir a linguagem Java através de exemplos

Por que Java?

- Portabilidade graças a JVM (*Write once run anywhere'*)
- Aceitação no mercado, com impacto direto em sua confiabilidade. Talvez Java, no futuro, tenha como maior força a confiabilidade (como a confiabilidade atual de linguagens como COBOL).
- Projeto da linguagem bem cuidadoso, suprindo algumas deficiências de C++, mas tem se tornado uma linguagem de progresso lento (decisões melhores estão presentes em Scala, Clojure, ...).
- Aplicabilidade: software básico (compiladores, SGBDs), sitemas embarcados (controladores em geral), games, aplicações corporativas, aplicações móveis, cartões inteligentes ...

Mas não recomendada para certas classes de sistemas corporativos, pois não são críticas do ponto de vista de confiabilidade e existem alternativas mais produtivas (Groovy, Ruby, PHP, . . .)

Visão geral do processo de compilação



Programas de exemplo

- Hello, world!
- Valores mínimo e máximo
- Aposentadoria

Hello, world!

```java
package br.unb.lp.poo;

/**
 * Classe principal do programa, disponibilizando um metodo
 * principal que simplesmente imprime: Hello, world.
 *
 * Ok, esse eh o nosso primeiro programa em java. Note que
 * toda classe deve ser declarada em um pacote, no noso caso,
 * o pacote eh br.unb.lp.poo (apenas por convencao, os nomes
 * dos pacotes refletem a empresa / organizaco).
 *
 * @author rbonifacio
 */
public class HelloWorld {
    public static void main(String args[]) {
      System.out.println("Hello, world");
    }
}
```

Compilação com o Java

Programas pequenos

```
$ javac -d ../bin/ br/unb/lp/poo/HelloWorld.java
```

Compilação com o Java

Programas pequenos

```
$ javac -d ../bin/ br/unb/lp/poo/HelloWorld.java
```

Programas maiores

Uso de ferramentas como *ANT* ou *Maven*, compilando as unidades em separado, realizando a configuração do *CLASSPATH*, executando testes de unidade e demais análises de qualidade.

Compilação com o Java

Programas pequenos

```
$ javac -d ../bin/ br/unb/lp/poo/HelloWorld.java
```

Programas maiores

Uso de ferramentas como *ANT* ou *Maven*, compilando as unidades em separado, realizando a configuração do *CLASSPATH*, executando testes de unidade e demais análises de qualidade.
Vamos utilizar uma abordagem intermediária, em que o Eclipse nos apóia na maior parte das atividades de *build*.

Ao longo do curso, discutiremos as vantagens de uma decomposição OO em relação a um desenho procedural.

Algumas mais discutidas:

- Vocabulário próximo ao domínio
- Maior qualidade e produtividade
  - facilidade em compreender o software
  - facilidade em manter o software

Nada tão diferente do que vocês aprenderam em C/C++

Nada tão diferente do que vocês aprenderam em C/C++, vamos começar logo a trabalhar com POO usando Java.

- Deste ponto em diante, os slides são baseados nas notas de aula do professor Jacques Sauvé (UFCG)

Introdução a Orientação a Objetos

Pode-se pensar sobre o mundo real como uma coleção de objetos relacionados

Introdução a Orientação a Objetos

Pode-se pensar sobre o mundo real como uma coleção de objetos relacionados

Objetos do mundo bancário

- Agências
- Contas corrente, contas de poupança
- Clientes
- Caixas
- Cheques, extratos, ...

Objetos podem ser agrupados em classes

Contas Corrente possuem estrutura e comportamento semelhante

Objetos podem ser agrupados em classes

Contas Corrente possuem estrutura e comportamento semelhante. Cada conta corrente possui uma identificação, uma data de abertura, um saldo atual. Além disso, operações estão disponíveis para contas corrente, tais como:

- encerramento
- débito, crédito, cobrança de taxas
- obtenção do saldo atual

Objetos podem ser agrupados em classes

Contas Corrente possuem estrutura e comportamento semelhante. Cada conta corrente possui uma identificação, uma data de abertura, um saldo atual. Além disso, operações estão disponíveis para contas corrente, tais como:

- encerramento
- débito, crédito, cobrança de taxas
- obtenção do saldo atual

A observação de que objetos com mesma estrutura e comportamento dão origens às classes, que servem como um molde, um tipo definido pelo usuário que serve para a construção ou instanciação de objetos.

Objetos de uma certa classe possuem atributos

- Contas possuem número, saldo, histórico de transações
- Clientes possuem nome e endereço
- Cheque tem um valor

Objetos de uma certa classe possuem mesmo comportamento

- Clientes entram (cada vez mais raramente) nas agências
- Clientes realizam depósito e saques em contas correntes
- Clientes emitem cheques

Objetos de uma certa classe possuem mesmo comportamento

- Clientes entram (cada vez mais raramente) nas agências
- Clientes realizam depósito e saques em contas correntes
- Clientes emitem cheques

Objetos também podem estar relacionados

- Um cliente possui várias contas
- Uma conta tem um histórico de transações

Programa que manipula uma Conta Simples

```
package br.unb.cic.lp.sistemaBancario;

import junit.framework.TestCase;

/**
 * Uma classe usada para testar ContaSimples.
 * Melhor do que escrever um "main" para realizar
 * testes. Tal tipo de teste eh chamado de Teste Unitario
 *
 * @author rbonifacio
 */
public class TestContaSimples extends TestCase {
    public void testOperacaoSaque() {
      ContaSimples umaConta = new ContaSimples("rbonifacio", "123456789", 1);
      umaConta.depositar(500.0);
      assertEquals(500.0, umaConta.getSaldo());

      System.out.println("Saldo da conta antes do saque: " + umaConta.getSaldo());

      umaConta.sacar(50.0);
      assertEquals(450.0, umaConta.getSaldo());

      System.out.println("Saldo da conta apos o saque" + umaConta.getSaldo());
    }
}
```

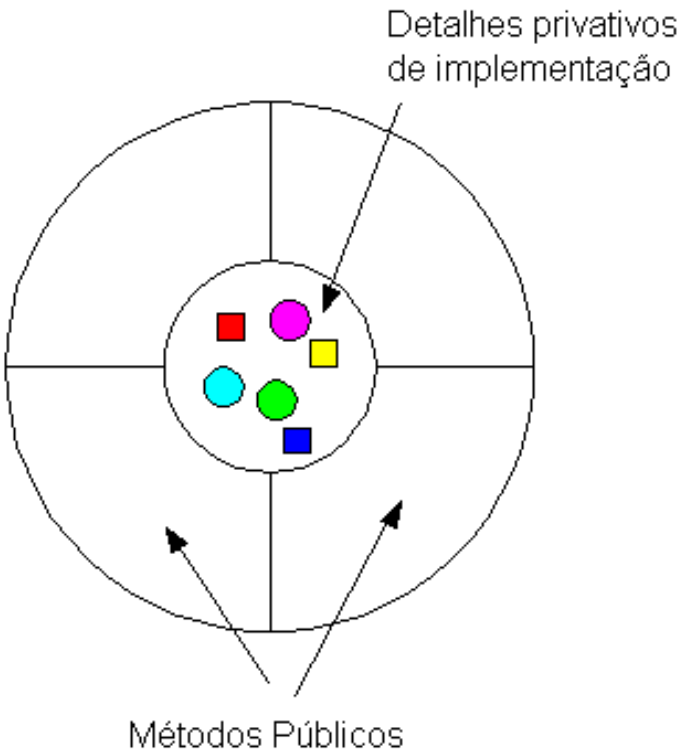
Abstração e Encapsulamento

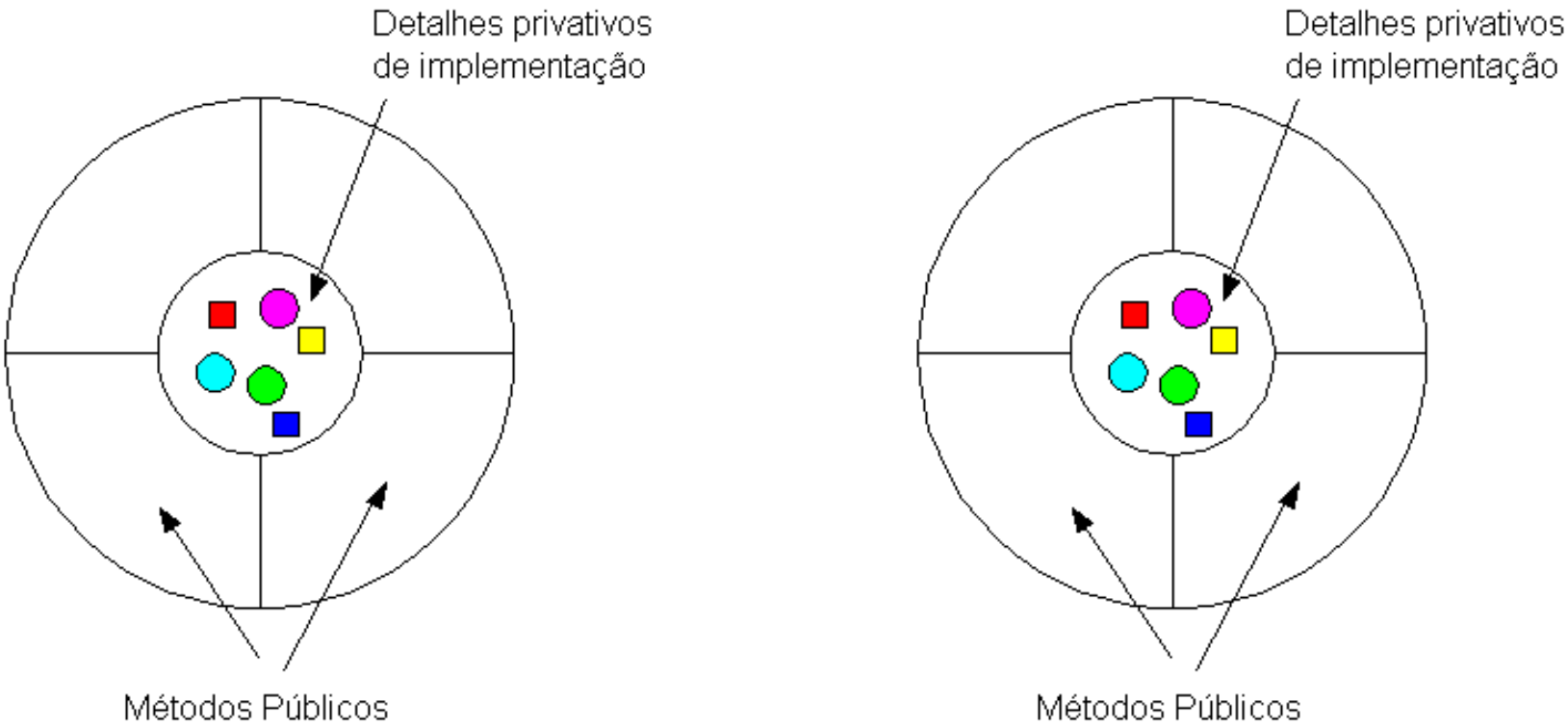
Foi possível (re)usar a classe `ContaSimples` sem conhecermos os detalhes como essa classe está implementada.

Abstração e Encapsulamento

Foi possível (re)usar a classe `ContaSimples` sem conhecermos os detalhes como essa classe está implementada.
Ou seja, a classe `ContaSimples`

- abstrai detalhes, oculta informações que não são relevantes para o reúso de comportamento de contas simples
- encapsula dados (que representam o estado das contas) e comportamento que permite alterar o estado das instâncias de contas simples


Detalhes privativos
de implementação
Métodos Públicos



Fonte: Notas de aula do Prof. Jacques Sauvé (UFCG)

Mais exemplos

- Transferência entre contas
- Gerenciamento de múltiplas contas

Próximas aulas

- Polimorfismo por subtipo
- Polimorfismo parametrizado (*generics*)
- Tratamento de Exceções