



PCS 2302/2024
Laboratório de
Fundamentos da
Eng.de Computação

Professores:
Anarosa A.F. Brandão
Jaime S. Sichman
Reginaldo Arakaki
Ricardo L.A. Rocha
© 2009

Aula 3:

Fundamentos Prog.
Orientada Objetos (1)

Autores:
Anarosa A.F. Brandão
Jaime S. Sichman
João José Neto
Paulo S. Muniz Silva
Ricardo L. A. Rocha

v. 1.4 ago 2009

1

PCS-2302 / PCS-2024

Lab. de Fundamentos de Eng. de Computação

Aula 03

Fundamentos da Programação Orientada a Objetos (Parte 1)

Professores:

Anarosa Alves Franco Brandão (PCS 2302)

Jaime Simão Sichman (PCS 2302)

Reginaldo Arakaki (PCS 2024)

Ricardo Luís de Azevedo da Rocha (PCS 2024)

Monitores: Diego Queiroz e Tiago Matos



PCS 2302/2024
Laboratório de
Fundamentos da
Eng.de Computação

Professores:
Anarosa A.F. Brandão
Jaime S. Sichman
Reginaldo Arakaki
Ricardo L.A. Rocha
© 2009

Aula 3:

Fundamentos Prog.
Orientada Objetos (1)

Autores:
Anarosa A.F. Brandão
Jaime S. Sichman
João José Neto
Paulo S. Muniz Silva
Ricardo L. A. Rocha

v. 1.4 ago 2009

2

Roteiro

1. Motivação
2. Solução de problemas computacionais
3. Princípios da orientação a objetos (OO)
4. Conceitos básicos em OO
5. Modelagem em OO
6. Java: uma linguagem de programação OO
7. Elementos básicos da programação OO
8. Parte Experimental
 - Implementação de exemplos simples: Banco
 - Implementação do simulador MVN (parte 1)
 - Classe **Memória**

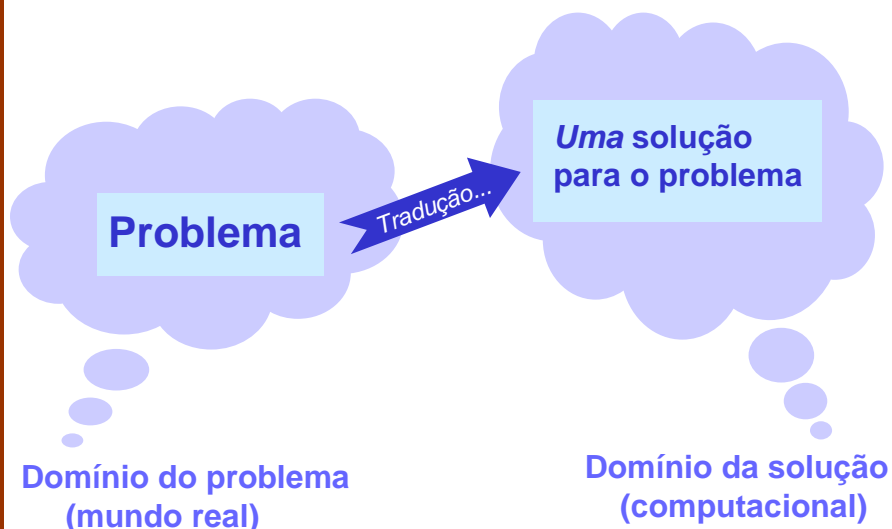


Motivação para OO

- Nosso problema imediato:
 - Implementar um simulador para a Máquina de Von Neumann apresentada em aula.
- Decisão de projeto:
 - A implementação utilizará o enfoque da orientação a objeto (OO).
- OO e a disciplina:
 - O ponto de vista adotado será o do programador.
 - Serão utilizados elementos básicos da **UML – Unified Modeling Language** para a modelagem de problemas e soluções.
 - Será utilizada a linguagem de programação Java.

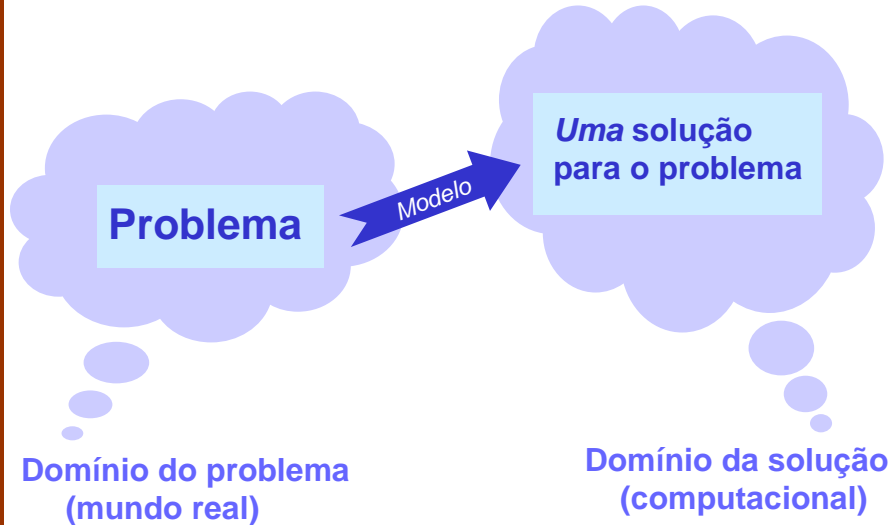


Solução de problemas computacionais (1)

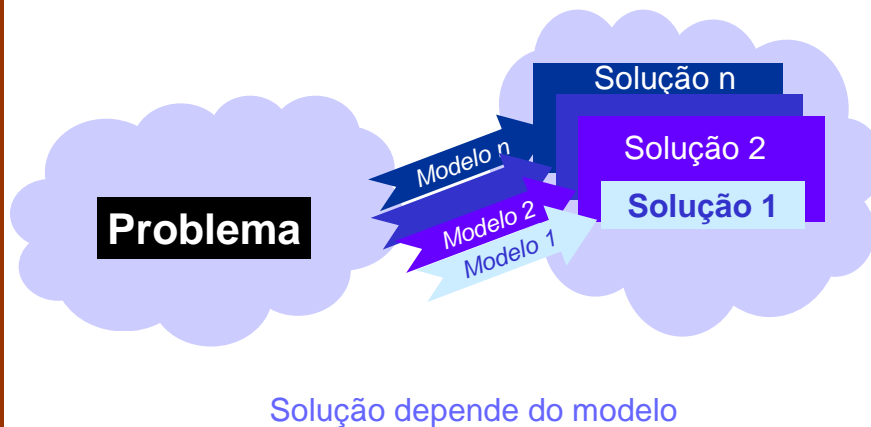




Solução de problemas computacionais (2)



Solução de problemas computacionais (3)





Solução de problemas computacionais (4)

- Dificuldade inicial:
 - Distância entre o problema no mundo real e sua solução computacional (modelo abstrato construído);
 - Quanto menor a distância, mais rápida será a construção da solução;
 - Paradigma de desenvolvimento escolhido pode facilitar a tarefa de criação do modelo da solução.



Solução de problemas computacionais (5)

- Principais paradigmas de desenvolvimento:
 - Estruturado;
 - Modelo entrada – processamento – saída;
 - Dados separados das funções.
 - Orientado a objetos (OO)
 - O mundo é composto por objetos;
 - Objetos combinam dados e funções;
 - Conceitos do problema são modelados como objetos que são associados e interagem entre si.
- Outros
 - Funcional, Lógico, etc



Princípios da Orientação a Objetos (1)

- OO busca gerenciar a complexidade dos problemas do mundo real *abstraindo* o conhecimento relevante e *encapsulando-o* em *objetos*. Para tanto utiliza-se dos seguintes fundamentos:

- Abstração;
- Encapsulamento;
- Modularidade;
- Hierarquia.



Princípios da Orientação a Objetos (1)

- OO busca gerenciar a complexidade dos problemas do mundo real *abstraindo* o conhecimento relevante e *encapsulando-o* em *objetos*. Para tanto utiliza-se dos seguintes fundamentos:

- Abstração;
- Encapsulamento;
- Modularidade;
- Hierarquia.



Princípios da Orientação a Objetos (2)

- Espécies de abstração
 - Abstração de procedimentos
 - Permite introduzir novas operações
 - Abstração de dados
 - Permite introduzir novos tipos de objetos de dados.
 - Base para a programação orientada a objeto.
 - Abstração da iteração
 - Permite iterar sobre itens em uma coleção sem revelar detalhes sobre como os itens foram obtidos.
 - Hierarquia de tipos
 - Permite abstrair de tipos de dados individuais para famílias de tipos relacionados.



Princípios da Orientação a Objetos (2)

- Espécies de abstração
 - Abstração de procedimentos
 - Permite introduzir novas operações
 - Abstração de dados
 - Permite introduzir novos tipos de objetos de dados.
 - Base para a programação orientada a objeto.
 - Abstração da iteração
 - Permite iterar sobre itens em uma coleção sem revelar detalhes sobre como os itens foram obtidos.
 - Hierarquia de tipos
 - Permite abstrair de tipos de dados individuais para famílias de tipos relacionados.



Princípios da Orientação a Objetos (3)

- Abstração de Procedimentos:
 - Uma operação com efeito bem definido pode ser tratada como atômica, mesmo que ela faça uso de outras operações de mais baixo nível;
 - Ex.: `calcularSalarioLiquido`: definida em termos das operações `obterSalarioBruto`, `calcularImposto`, `calcularDescontos`, etc.
- Mecanismos de abstração de procedimentos
 - Por parametrização;
 - Por especificação.



Princípios da Orientação a Objetos (4)

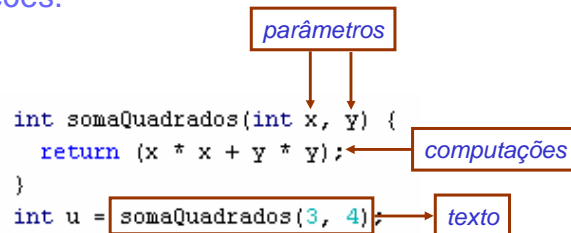
- Abstração de procedimentos por parametrização.
 - Permite, através da introdução de parâmetros, a representação de um conjunto potencialmente infinito de computações distintas com um único texto de programa, o qual é a abstração de todas as computações.

```
int somaQuadrados(int x, y) {  
    return (x * x + y * y);  
}  
int u = somaQuadrados(3, 4);
```



Princípios da Orientação a Objetos (4)

- Abstração de procedimentos por parametrização.
 - Permite, através da introdução de parâmetros, a representação de um conjunto potencialmente infinito de computações distintas com um único texto de programa, o qual é a abstração de todas as computações.



Princípios da Orientação a Objetos (5)

- Abstração de procedimentos por especificação.
 - Permite abstrair a(s) computação(ões) descrita(s) pelo corpo de um procedimento na finalidade que o procedimento foi projetado para realizar. Associa ao procedimento uma especificação de seu efeito pretendido e considera que a chamada do procedimento está baseada nesta especificação.

```

float raizQuadrada(float x) {
    // Finalidade: Calcula a raiz quadrada de um número real.
    // Pré-condição: x > 0
    // Pós-condição: Retorna uma aproximação da raiz quadrada
    // de x.
    // ... Corpo do procedimento
}

```




Princípios da Orientação a Objetos (6)

- Abstração de procedimentos por especificação.

```
float raizQuadrada(float x) {  
    // Finalidade: Calcula a raiz quadrada de um número real.  
    // Pré-condição: x > 0  
    // Pós-condição: Retorna uma aproximação da raiz quadrada  
    // de x.  
    // ... Corpo do procedimento  
}
```

- Finalidade. Descreve sucintamente a finalidade do procedimento.
- Pré-condição. Estabelece as condições que devem ser verdadeiras no início da operação.
- Pós-condição. Estabelece as condições que devem ser verdadeiras ao final da operação.



Princípios da Orientação a Objetos (7)

- Abstração de Dados:

- Permite abstrair dos detalhes de como os dados são implementados para como os objetos se comportam.
 - Foco no comportamento fundamenta a POO.
- Permite introduzir novos **tipos** de dados
 - Os novos **tipos** de dados dependem do domínio de aplicação do programa. Por ex., **Conta**, para o domínio bancário.



Princípios da Orientação a Objetos (8)

- Abstração de Dados:

- Um **tipo** pode ser caracterizado por uma *representação de dados* e por *operações* sobre esta representação.
 - A definição de um tipo cria meios para a concretização de objetos que sigam sua especificação (instâncias de tipos)
- O acesso (externo) a um **tipo** se dá através das *operações* que realizam seu comportamento.
 - Consequência: o código usado depende apenas do comportamento especificado para o **tipo** com suas operações. Consegue-se a abstração por especificação.
 - A representação de dados fica encapsulada no tipo.



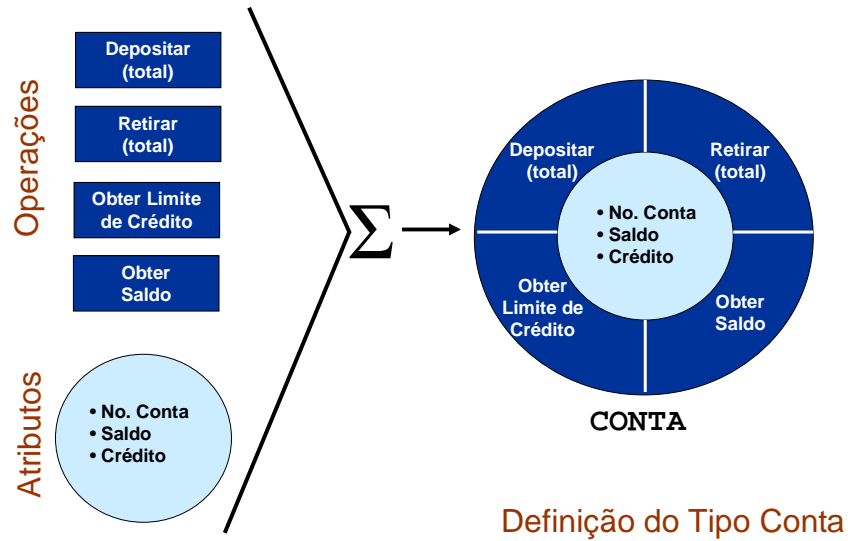
Princípios da Orientação a Objetos (9)

- Exemplos de tipos:

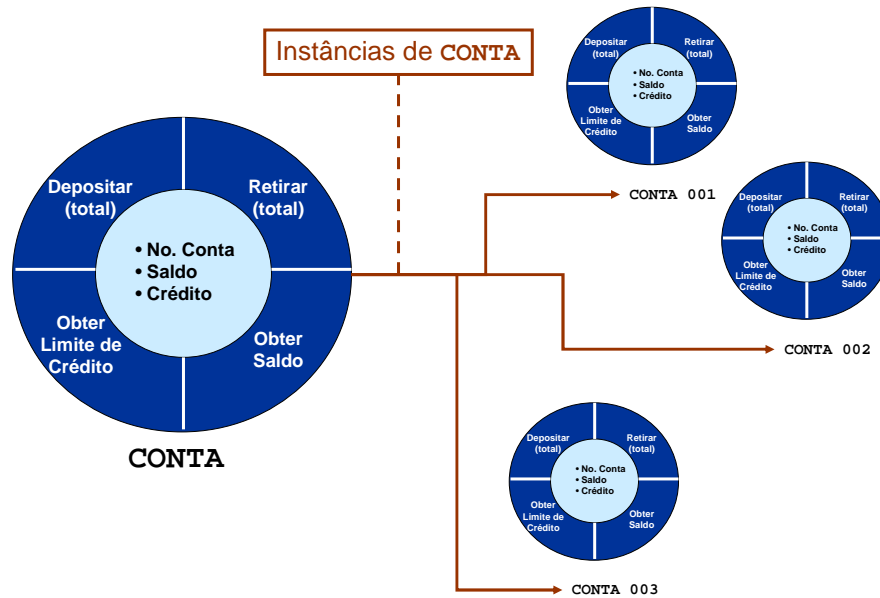
- Um **tipo pilha** é definido por suas operações **empilhar** e **desempilhar**.
 - A forma como as operações são implementadas não altera a especificação que define o tipo **pilha**!
 - O tipo **pilha** engloba a estrutura de dados pilha e sua implementação!
- Um **tipo conta** pode ser definido pelas operações **depositar**, **retirar**, **obterLimiteCredito** e **obterSaldo**



Princípios da Orientação a Objetos (10)



Princípios da Orientação a Objetos (11)





Princípios da Orientação a Objetos (12)

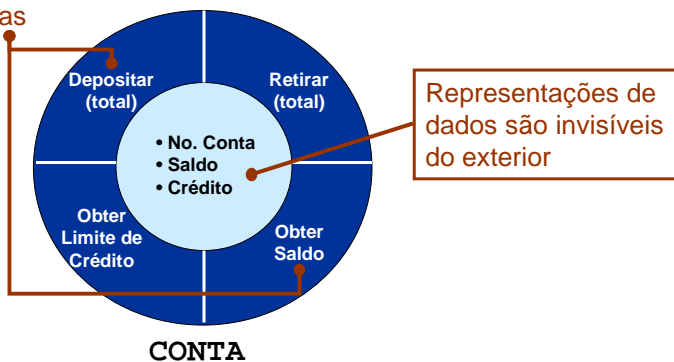
• Encapsulamento

- Separa os aspectos externos (o que faz) dos aspectos internos (como faz):
 - Aspectos externos = interface, contrato;
 - Aspectos internos = implementação.
- Complementa o pensamento abstrato
 - Ao abstrair focamos no comportamento observável do objeto
 - A implementação do comportamento é encapsulada
- Promove maior estabilidade:
 - Clientes do objeto só conhecem sua interface;
 - Podemos alterar a implementação de uma operação sem afetar o restante do sistema.



Princípios da Orientação a Objetos (13)

Operações públicas
(ou visíveis), manipulam
representações
encapsuladas





Princípios da Orientação a Objetos (14)

- Modularidade

- Decomposição do sistema em módulos

- Dividir para conquistar!
- Agrupamento lógico das abstrações relacionadas

- Importante ao considerar reuso do sistema (ou parte dele).

- Hierarquia

- Meio para organizar as abstrações e simplificar o entendimento do problema.

- Ordenação

Voltaremos a estes assuntos...



Conceitos básicos em OO (1)

Classes

Instâncias

Objetos

Estruturação



Associação

Composição

Herança

Métodos

Mensagens



PCS 2302/2024
Laboratório de Fundamentos da Eng.de Computação

Professores:
Anarosa A.F. Brandão
Jaime S. Sichman
Reginaldo Arakaki
Ricardo L.A. Rocha
© 2009

Aula 3:
Fundamentos Prog. Orientada Objetos (1)

Autores:
Anarosa A.F. Brandão
Jaime S. Sichman
João José Neto
Paulo S. Muniz Silva
Ricardo L. A. Rocha

v. 1.4 ago 2009

27

Conceitos básicos em OO (1)

entidades

Classes

Instâncias

Objetos

Métodos



Mensagens

Estruturação

Associação

Composição

Herança



PCS 2302/2024
Laboratório de Fundamentos da Eng.de Computação

Professores:
Anarosa A.F. Brandão
Jaime S. Sichman
Reginaldo Arakaki
Ricardo L.A. Rocha
© 2009

Aula 3:
Fundamentos Prog. Orientada Objetos (1)

Autores:
Anarosa A.F. Brandão
Jaime S. Sichman
João José Neto
Paulo S. Muniz Silva
Ricardo L. A. Rocha

v. 1.4 ago 2009

28

Conceitos básicos em OO (1)

operações

Métodos

Mensagens

Classes

Instâncias

Objetos

Estruturação

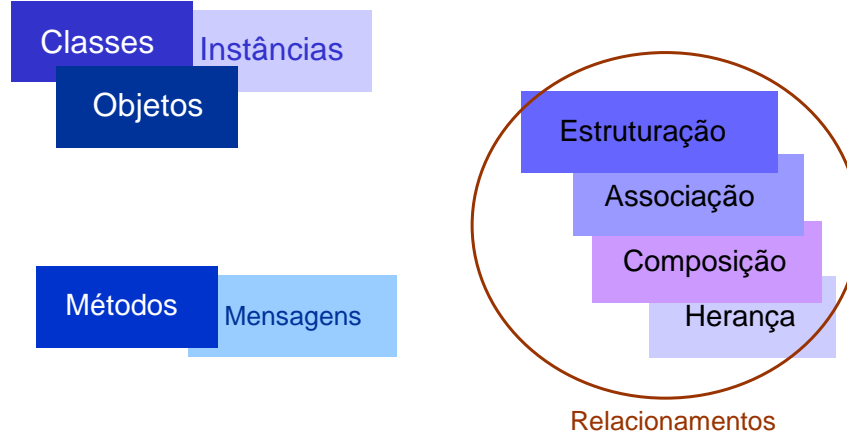
Associação

Composição

Herança



Conceitos básicos em OO (1)



Conceitos básicos em OO (2)

• Classes

- Uma classe é uma abstração de dados que define um conjunto de objetos que possuem mesma estrutura e comportamento
- Tipos definem classes





Conceitos básicos em OO (3)

Objetos

- Um objeto é uma entidade que concretiza uma abstração relevante no contexto de uma aplicação
- Objetos são instâncias de classe e possuem:
 - estado (estrutura): propriedades + valores correntes
 - comportamento: serviços que o objeto provê (operações)
 - identidade: identificador único que diferencia objetos com mesmo estado e comportamento



CONTA 001



CONTA 002



CONTA 003



Conceitos básicos em OO (4)

Métodos

- Métodos implementam operações e são invocados através de mensagens
- Normalmente vinculados aos objetos. Pode-se também ter métodos vinculados à classe (ver adiante)
- O método vinculado a um objeto tem implicitamente o objeto como argumento.
- Existem métodos construtores, que são executados ao criar um novo objeto na memória e que definem, usualmente, o estado inicial do objeto
- Outro tipo de métodos são os destrutores, que servem para limpar os espaços de memória alocados pelo objeto



Conceitos básicos em OO (5)

- Mensagens

- Toda funcionalidade do sistema é realizada pela troca de mensagens entre objetos.
- O acesso às propriedades (aos serviços) de um objeto é realizado através dos seus métodos, via troca de mensagens.



Conceitos básicos em OO (6)

- Estruturação

- Refere-se a forma como os objetos se relacionam
- Tipos de relacionamentos mais comuns
 - Associação
 - conexão entre classes que representa a existência de interações entre objetos;
 - Composição e Agregação
 - Formas especiais de interações entre objetos que modelam relacionamentos do tipo todo-parte;
 - Herança (especialização/generalização)
 - Define hierarquias de classes



Conceitos básicos em OO (6)

- Estruturação

- Refere-se a forma como os objetos se relacionam
- Tipos de relacionamentos mais comuns

- Associação

- conexão entre classes que representa a existência de interações entre objetos;

- Composição e Agregação

- Formas especiais de interações entre objetos que modelam relacionamentos do tipo todo-parte;

- Herança (especialização/generalização)

- Define hierarquias de classes



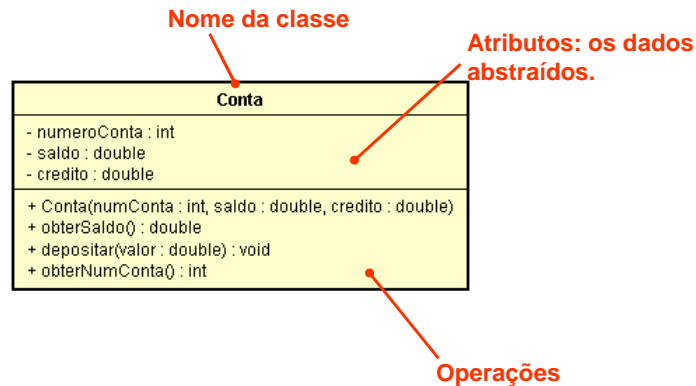
Modelagem em OO (1)

- Para facilitar a descoberta das abstrações de dados e a forma como elas se relacionam foram criadas linguagens de modelagem OO.
- UML (Unified Modeling Language) é o padrão atual destas linguagens
 - Neste curso usaremos um pequeno subconjunto de elementos desta linguagem para descrevermos a solução OO para o problema da MVN
 - Este subconjunto permitirá a construção de *diagramas de classes*



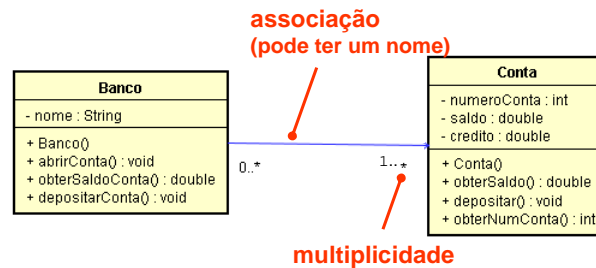
Modelagem em OO (2)

- Exemplo de descrição de uma classe em UML



Modelagem em OO (3)

- Exemplo de descrição de relacionamento entre classes em UML



- A multiplicidade é uma restrição que indica quantos objetos estão envolvidos em um relacionamento.
 - As multiplicidades mais utilizadas são: 1 (há um objeto), 0..1 (não há objeto ou há um único objeto) e 0..* ou 1 .. * (zero/um ou muitos objetos).



Java: uma linguagem de programação OO (1)

- Existem várias boas referências para o aprendizado de Java. Por exemplo, em português:
Deitel, H.M.; Deitel, P.J. *Java Como Programar*. 6a. Edição. Pearson Education do Brasil, 2005.
- Um sítio com referências para tutoriais introdutórios e informações interessantes sobre Java é:
 - www.oopweb.com/Java/Files/Java.html
 - Em particular, Eckel, B. *Thinking in Java* (3rd Edition): www.oopweb.com/Java/Documents/ThinkingInJava/VolumeFrames.html
- N.B. A disciplina não é um “curso de Java”! Java é utilizada como uma linguagem OO para resolver os problemas da disciplina.
 - Na disciplina, utilizaremos apenas um pequeno subconjunto de características de Java.



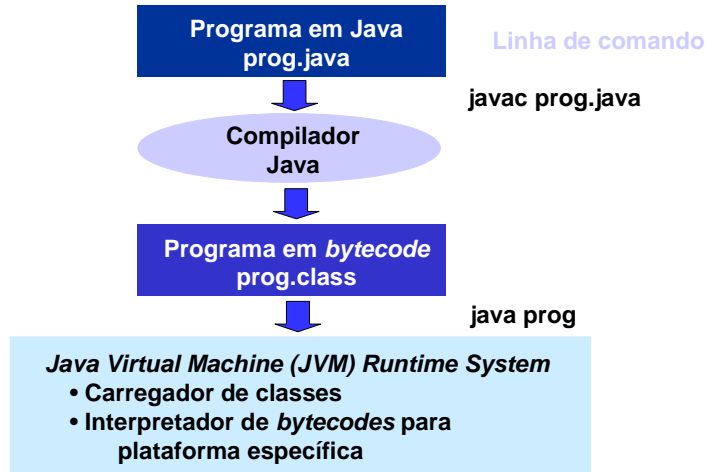
Java: uma linguagem de programação OO (2)

- A ferramenta de desenvolvimento utilizada na parte prática da disciplina é *DrJava*, uma ferramenta bem simples mas eficaz para os propósitos da disciplina.
<http://www.drjava.net>
- A versão instalada no laboratório é a versão estável e atual: **drjava-stable-20080106-0744**.
 - Existem versões em *.jar e *.exe
- A documentação do usuário de DrJava pode ser encontrada no sítio.
- Para executar o DrJava é necessário ter instalado o JDK, que pode ser encontrado em: <http://www.java.sun.com>



Java: uma linguagem de programação OO (3)

- Visão simplificada do ambiente Java:



Java: uma linguagem de programação OO (4)

- Especificação de abstração de dados em Java
 - Novos tipos também são definidos por classes ou interfaces.
 - Classe:
 - Nome da classe = nome do tipo.
 - Conjunto de métodos = implementação das operações.
 - Conjunto de campos = implementação dos atributos.
 - Criação (iniciação) de novas instâncias da classe = construtor(es).
 - Instância da classe = Objeto
 - Uma vez criado um objeto, ele pode ser usado pela chamada de seus métodos. Esta chamada é também denominada de 'envio de mensagem'. Se tivermos um objeto `obj` com um método público (visível do exterior) `metodo()`, sua chamada tem a sintaxe:


```
obj.metodo()
```



Java: uma linguagem de programação OO (5)

- Especificação de classe em Java

```
visibilidade class NomeDaClasse {  
    // DESCRIÇÃO: Uma breve descrição do  
    // comportamento do tipo de objetos  
    // implementado pela classe.  
  
    // Construtores  
    // Especificações dos construtores:  
    // abstração procedimental por  
    // especificação.  
  
    // Métodos  
    // Especificações dos métodos: abstração  
    // procedimental por especificação.  
}
```



Java: uma linguagem de programação OO (6)

- Fragmento da especificação de uma classe

```
public class Conta {  
  
    // DESCRIÇÃO: As contas bancárias...  
  
    // Construtores  
    public Conta() {  
        // Pós: Inicia este objeto com saldo = 0.00  
    }  
  
    // Métodos  
    public void retirar(double valor) {  
        // Finalidade: Calcula o novo valor do saldo após a retirada  
        // de um determinado valor.  
  
        // Pré: O valor deve ser menor ou igual ao saldo atual  
        // acrescido do limite de crédito.  
  
        // Pós: O novo saldo é igual ao antigo saldo menos o valor  
        // da retirada.  
    }  
}
```

Apenas um deles.
Sem pré-condição
relevante.



PCS 2302/2024
Laboratório de
Fundamentos da
Eng.de Computação

Professores:
Anarosa A.F.Brandão
Jaime S. Sichman
Reginaldo Arakaki
Ricardo L.A. Rocha
© 2009

Aula 3:

Fundamentos Prog.
Orientada Objetos (1)

Autores:
Anarosa A.F. Brandão
Jaime S. Sichman
João José Neto
Paulo S. Muniz Silva
Ricardo L. A. Rocha

v.1.4 ago 2009

45

Exemplo de programa muito simples (1)

```
// Conta.java
// PCS2302/2006 - Aula 3: a classe Conta

public class Conta {
    // DESCRIÇÃO: Representação muito simples e parcial de uma conta bancária.

    // Atributos
    private int _numeroConta;    // número da conta
    private double _saldo;       // saldo corrente da conta
    private double _credito;      // limite de crédito da conta

    // Construtores
    // Construtor com valores iniciais de número da conta, saldo e crédito parametrizados.
    // 0 número da conta é fornecido pelo Banco.
    public Conta(int numConta, double saldo, double credito) {
        // Pós: Número da conta é numConta E
        // Saldo é saldo E
        // Limite de crédito é credito
        _numeroConta = numConta;
        _saldo = saldo;
        _credito = credito;
    } // fim de Conta()

    // Métodos. Faltam vários métodos. Implementar.
    public double obterSaldo() {
        // Finalidade: Obtém o saldo.
        // Pós: Retorna o saldo.
        return _saldo;
    } // fim obterSaldo()
```



PCS 2302/2024
Laboratório de
Fundamentos da
Eng.de Computação

Professores:
Anarosa A.F.Brandão
Jaime S. Sichman
Reginaldo Arakaki
Ricardo L.A. Rocha
© 2009

Aula 3:

Fundamentos Prog.
Orientada Objetos (1)

Autores:
Anarosa A.F. Brandão
Jaime S. Sichman
João José Neto
Paulo S. Muniz Silva
Ricardo L. A. Rocha

v.1.4 ago 2009

46

Exemplo de programa muito simples (2)

```
// Métodos. Faltam vários métodos. Implementar.
public double obterSaldo() {
    // Finalidade: Obtém o saldo.
    // Pós: Retorna o saldo.
    return _saldo;
} // fim obterSaldo()

public void depositar(double valor) {
    // Finalidade: Efetua um depósito.
    // Pré: valor > 0.00
    // Pós: O saldo atual é igual ao saldo anterior mais o valor
    if (valor <= 0.00)
        System.out.println("O valor depositado deve ser maior do que zero.");
    else
        _saldo += valor;
} // fim depositar()

public int obterNumConta() {
    // Finalidade: Retornar o número da conta.
    // Pós: Retorna o número da conta.
    return _numeroConta;
} // fim de obterConta()
} // fim da classe Conta
```



Exemplo de programa muito simples (3)

Teste simples para exercitar a classe Conta

```
// TesteConta.java
// PCS2302-2024: Exemplo de programa para teste da classe Conta

import java.io.*;

public class TesteConta {

    public static void main(String[] args) {
        Conta aConta = new Conta(10, 350.00, 1000.00);
        System.out.println("Saldo da conta " + aConta.obterSaldo());
        aConta.depositar(430.00);
        System.out.println("Saldo da conta " + aConta.obterSaldo());
    } // fim main()

} // fim classe TesteConta
```

O operador, new com o construtor como operando, aloca memória para o objeto aConta. Mais detalhes na próxima aula.



Elementos básicos da programação OO (1)

- Interação entre objetos

- De modo simplificado e informal, pode-se ter duas categorias de interação entre objetos: transitória e estrutural.

- Interação transitória: durante a execução de um método no objeto de origem há interação com método(s) do(s) objeto(s) de destino. Ao término da execução, o(s) objeto(s) de destino saem do escopo do objeto de origem.
- Interação estrutural: há um relacionamento semântico forte entre o objeto de origem e o(s) objeto(s) de destino. Por exemplo, a qualquer momento, pode haver invocação de método(s) do(s) objeto(s) de destino por qualquer método do objeto de origem.



Elementos básicos da programação OO (2)

- Interação transitória entre objetos

- Tem-se duas classes: **Banco** e **Conta**.

Banco tem o método **inativar** expressando o comportamento de desativar conta.

```
public class Banco {
    // Atributos ...
    // Contrutor(es) ...
    // Métodos ...

    public void inativar(Conta oConta) {
        // Finalidade: Inativar conta.
        // Pré-condição: Nenhuma.
        // Pós-condição: Conta inativada.
        oConta.inativar();
    } // fim inativar()

} // fim classe Banco
```

Normalmente implementada com um objeto da classe de destino como parâmetro.



Elementos básicos da programação OO (3)

- Interação estrutural entre objetos

- Tem-se duas classes: **Banco** e **Conta**.

- A classe **Banco** tem os métodos: **depositarConta**, **retirarConta** e **obterSaldoConta**. **Conta** foi vista anteriormente. Todos os métodos de **Banco** devem ter acesso a um objeto da classe **Conta**, em momentos distintos. Ademais, **Banco** deve ter acesso a vários objetos da classe **Conta**.

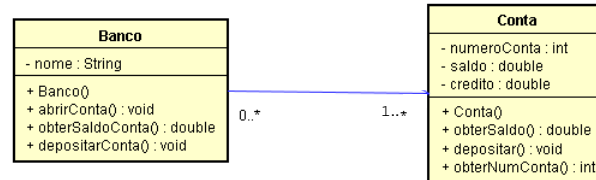
- Uma solução: implementar uma coleção de referências para objetos da classe **Conta** como campo da classe **Banco**.

- Qual a implementação da coleção de referências? Várias soluções. Por exemplo, se a semântica do problema requerer que a coleção seja ordenada, deve-se ter a implementação de uma sequência. Depende, portanto, do problema em mãos, incluindo considerações sobre o desempenho.



Elementos básicos da programação OO (4)

- Pode-se representar o relacionamento semântico entre classes, do ponto de vista estático, em um diagrama de classes UML.



Elementos básicos da programação OO (5)

- Java oferece várias alternativas para a implementação de coleções de objetos no pacote `java.util`. As *collections* (também conhecidos como contêineres), permitem armazenar e organizar objetos de vários modos para acesso eficiente.
- É muito importante estudar as coleções de Java!
- Algumas coleções disponíveis: `Set`, `List`, `ArrayList`, `Map`, `HashSet`, `HashMap`, etc.
- Como há a necessidade de mais conceitos para utilizá-las, isso será visto na próxima aula.
- Segue uma implementação parcial, muito simples e limitada da classe **Banco**, a qual tem ligações (instâncias da associação) unidirecionais para objetos **Conta**. Esta implementação tem finalidade apenas didática.



PCS 2302/2024
Laboratório de
Fundamentos da
Eng.de Computação

Professores:
Anarosa A.F. Brandão
Jaime S. Sichman
Reginaldo Arakaki
Ricardo L.A. Rocha
© 2009

Aula 3:
Fundamentos Prog.
Orientada Objetos (1)

Autores:
Anarosa A.F. Brandão
Jaime S. Sichman
João José Neto
Paulo S. Muniz Silva
Ricardo L. A. Rocha

v.1.4 ago 2009

53

Elementos básicos da programação OO (6)

```
// Banco.java
// PCS2302/2006 - Aula 3: a classe Banco
public class Banco {
    // DESCRIÇÃO: Representação incompleta muito simples de um banco.

    // Atributos
    private String _nome; // Nome do banco
    private Conta[] _asContas; // Implementação simplória, por enquanto, da ligação com as contas
    private final int MAX_CONTAS = 100; // Idem
    // 0 número da conta deve ser atribuído automaticamente pelo Banco a cada conta.
    // Devemos ter, portanto, uma variável de classe, compartilhada por todos
    // os objetos da classe, a qual é incrementada de 1 a cada instânciação de um
    // objeto Conta.
    private static int _proxNumConta = 0; // Implementação simples.

    // Construtor default
    public Banco() {
        _nome = "Banco ABC";
        _asContas = new Conta[MAX_CONTAS]; // Suporte para até 100 contas !?
    }

    // Métodos: por economia, nenhum dos métodos que segue avalia adequadamente a pré-condição.
    // Mas vocês não poderão deixar de testá-la!!!
    public void abrirConta(double saldoInicial, double limiteCredito) {
        // Finalidade: Cria uma conta.
        // Pré: saldoInicial >= 0.00 E limiteCredito >= 0.00
        // Pós: conta criada com um número gerado sequencialmente e vinculada ao banco.
        if (_proxNumConta < MAX_CONTAS) {
            _asContas[_proxNumConta] = new Conta(_proxNumConta, saldoInicial, limiteCredito); // Vincula a conta ao banco
            ++_proxNumConta;
        }
        else
            System.out.println("Atingido o numero maximo de contas.");
    } // fim abrirConta()
}
```



PCS 2302/2024
Laboratório de
Fundamentos da
Eng.de Computação

Professores:
Anarosa A.F. Brandão
Jaime S. Sichman
Reginaldo Arakaki
Ricardo L.A. Rocha
© 2009

Aula 3:
Fundamentos Prog.
Orientada Objetos (1)

Autores:
Anarosa A.F. Brandão
Jaime S. Sichman
João José Neto
Paulo S. Muniz Silva
Ricardo L. A. Rocha

v.1.4 ago 2009

54

Elementos básicos da programação OO (7)

```
public double obterSaldoConta(int numConta) {
    // Finalidade: Obtém o saldo da conta de número numConta
    // Pré: numConta existente.
    // Pós: saldo da conta de numConta.
    for (int i = 0; i < _proxNumConta; ++i) {
        if (_asContas[i].obterNumConta() == numConta)
            return _asContas[i].obterSaldo();
    }
    // 0 código que segue é inadequado, mas serve por enquanto.
    System.out.println("Conta nao encontrada.");
    return 0.0;
} // fim obterSaldoConta()

public void depositarConta(int numConta, double valor) {
    // Finalidade: Deposita um valor numa conta.
    // Pré: valor > 0.00 E numConta existente.
    // Pós: valor depositado na conta.
    for (int i = 0; i < _proxNumConta; ++i) {
        if (_asContas[i].obterNumConta() == numConta) {
            _asContas[i].depositar(valor);
            return;
        }
    }
    // 0 código que segue é inadequado, mas serve por enquanto.
    System.out.println("Conta nao encontrada.");
} // fim depositarConta
} // fim classe Banco
```



Exercícios

TYGXXA03E01_09

Os slides #45-47 e #53-54 apresentam um fragmento de programa que implementa, em Java, um modelo muito simples de conta bancária e de seu relacionamento com um banco.

- Estenda a funcionalidade do Banco para aceitar operações de retirada de um determinado valor. O valor máximo da retirada não pode deixar um saldo menor do que 10% do valor do limite de crédito.
- Escreva as especificações dos métodos necessários no estilo apresentado em aula. Implemente-os em Java.
- Escreva o programa de teste para exercitar todas as operações do Banco com pelo menos 3 contas.



Exercícios

TYGXXA03E02_09

Esta primeira parte da implementação do simulador MVN tem por finalidade a familiarização com a programação OO em Java.

O simulador deve carregar um programa MVN escrito em arquivo texto para a memória. Deve também poder gravar um arquivo com extensão *.show, mostrando o conteúdo da memória para verificação.

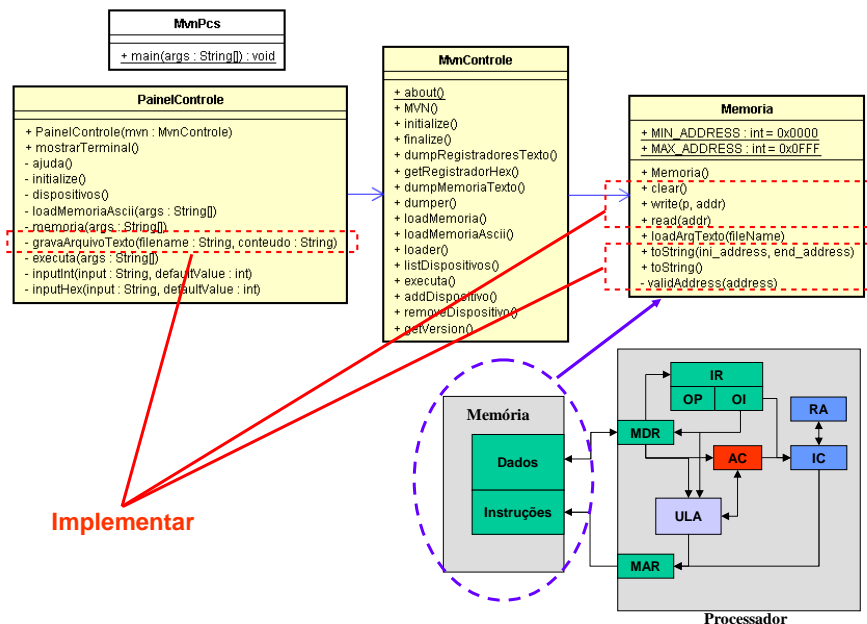


Implementação da MVN - Parte 1 (1)

- Segue o diagrama de classes parcial, em UML, do simulador MVN.
- Vocês receberão o código fonte incompleto, com especificações parciais de alguns métodos e a implementação de alguns métodos.
 - Implementar o simulador MVN de acordo com o diagrama de classes, codificando os métodos especificados e acrescentando atributos e métodos faltantes, com suas especificações e códigos das classes **Memoria** e **PainelControle**.
 - Especificar todos os métodos no estilo apresentado.



Implementação da MVN - Parte 1 (2)





Implementação da MVN - Parte 1 (3)

- Existem duas abstrações primitivas utilizadas pelo simulador: **Bits8** e **Word**.
 - Bits8 fornece a representação própria de um byte para a MVN.
 - Word implementa a abstração de uma palavra para a MVN, composta de 2 Bits8.
 - As classes que implementam Bits8 e Word são fornecidas. Notem que existem alguns métodos que podem não ser utilizados na implementação completa do simulador.
- Existe ainda outra classe **MVN** que encapsula constantes e literais globais utilizadas pelo simulador.
- O Painel de Controle ainda não segue exatamente a especificação da Aula 1. O código contém várias partes 'mortas'. Por enquanto, pode-se ignorar tais partes para não perder tempo de implementação.



Implementação da MVN - Parte 1 (4)

- Notas
 - A compreensão de determinados conceitos de OO do código de Bits8 ficará mais clara na próxima aula.
 - A utilização de **exceções** no código ficará mais clara na próxima aula.
 - Ao final da implementação (parte 3), as equipes poderão ter simuladores personalizados, mas todos deverão, mandatoriamente, atender aos requisitos especificados.
 - A atualização e limpeza do código fornecido é de responsabilidade de cada equipe.
 - Cada equipe deve gerir adequadamente as versões incrementais do seu simulador. Cada equipe é a única responsável pela manutenção dos programas.



Algumas dicas

- Leiam os comentários do código. Neles estão explicados alguns outros detalhes dos produtos que devem ser gerados
- Vejam os métodos existentes nas demais classes fornecidas (em especial a classe Bits8)
- Na classe **Integer** da biblioteca do Java há um conversor de inteiro para Hexadecimal (e vice-versa)
- Caso haja qualquer dúvida sobre uma classe do Java veja a documentação em: <http://java.sun.com/javase/6/docs/api/>.



Bibliografia Complementar

- Deitel, H.M.; Deitel, P.J. *Java - Como Programar*. 6a. Edição. Pearson Education do Brasil, 2005.
- Fowler, M. *UML Essencial* – 3a. Edição. Bookman Companhia Ed. 2004.
- Liskov, B.; Guttag, J. *Program Development in Java*. Addison-Wesley, 2001.
- Scott, M.L. *Programming Language Pragmatics*. Morgan Kaufmann, 2000.