

PROJETO ARQUITETURAL

PARTE I

Projeto de Programas – PPR0001

Atividades Envolvidas

Preliminar

- Realizar a organização dos dados considerando a tecnologia que será utilizada em módulos (exemplo: definir as classes);
- Elencar as operações do sistema.
- Definir encapsulamento

Refinamento

- Definir relação entre os módulos (organização hierárquica);
- Definir a estrutura global do software

Considerações

- Diagrama de Casos de Uso → perspectiva externa ao sistema
- Colaboração entre Objetos
 - Sistema Orientado a Objetos:
 - **Aspecto Estático (estrutural) → Diagrama de classes**
 - Aspecto Dinâmico → Diagrama de Sequência
 - Modelos complementares => Um modelo adiciona detalhes ao outro

Tipos de Dados

- Com o projeto de dados pronto, escolhe-se os tipos de dados que serão utilizados para representar cada dado/informação do sistema.
 - ☐ Primitivos
 - ☐ Composto Nativo
 - ☐ Composto Próprio
- Não reinventar a roda quando não é preciso!

Tipos de Dados

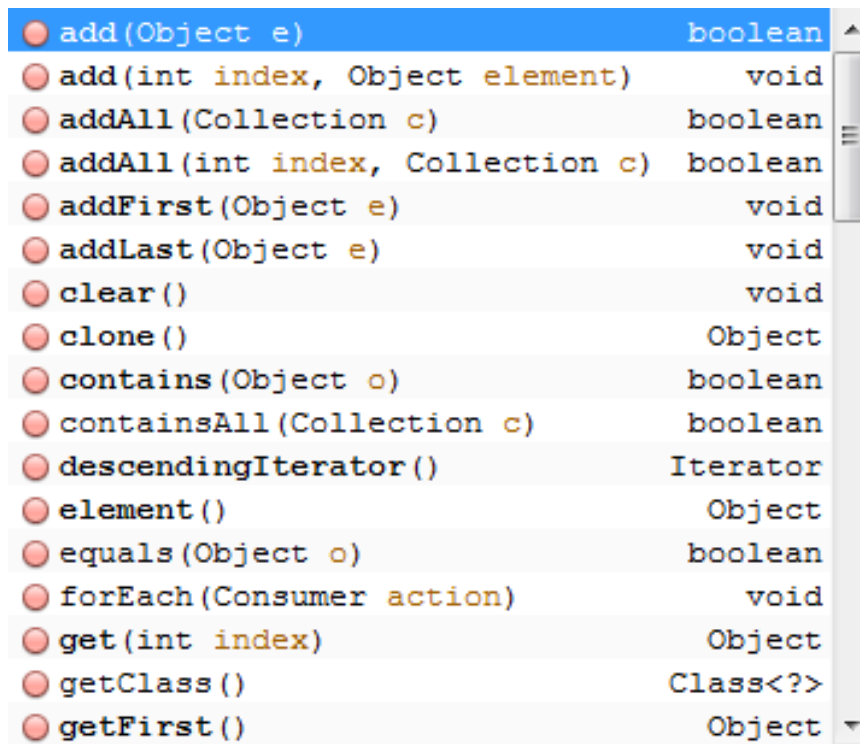
Lista Encadeada

```
LinkedList lista_1 = new LinkedList();  
LinkedList<Integer> lista_2 = new LinkedList<Integer>();  
lista_1.add( "teste" );  
lista_1.add( 23.56 );  
lista_2.add( 10 );
```

- Se nenhum tipo for especificado no “diamante” a lista poderá armazenar qualquer tipo de objeto, mas todo valor será considerado como *Object* (necessário *casting*)
- Se for especificado um tipo de dados então a lista só poderá possuir elementos daquele tipo

Tipos de Dados

Lista Encadeada



add(Object e)	boolean
add(int index, Object element)	void
addAll(Collection c)	boolean
addAll(int index, Collection c)	boolean
addFirst(Object e)	void
addLast(Object e)	void
clear()	void
clone()	Object
contains(Object o)	boolean
containsAll(Collection c)	boolean
descendingIterator()	Iterator
element()	Object
equals(Object o)	boolean
forEach(Consumer action)	void
get(int index)	Object
getClass()	Class<?>
getFirst()	Object

Métodos Principais:

- add => adiciona elemento
- addFirst => adiciona no inicio
- addLast => adiciona no fim
- clear => limpa a lista
- contains => verifica existência
- get => pegar um elemento
- pop => remove elemento
- remove => remove elemento
- size => tamanho da lista
- sort => ordena lista

Tipos de Dados

ArrayList

```
ArrayList lista_3 = new ArrayList();  
ArrayList<Double> lista_4 = new ArrayList<Double>();  
lista_3.add( lista_4 );
```

- ArrayLists possuem os mesmos métodos das listas encadeadas. O que se altera é o seu funcionamento interno. Entretanto, para esta disciplina não vamos nos importar com isso.

Tipos de Dados

Árvores / Conjunto - Set

```
Set<Integer> arvore = new TreeSet();  
arvore.add( 5 );  
boolean eh_elemento = arvore.contains( 2 );
```

- Árvores geralmente são úteis quando deseja-se maior desempenho em determinadas funções;
- Árvores são implementadas como conjuntos, a princípio; logo, não possuem elementos repetidos.

Tipos de Dados

Árvores / Conjunto - Set

add(Integer e)	boolean
addAll(Collection<? extends Integer> c)	boolean
clear()	void
contains(Object o)	boolean
containsAll(Collection<?> c)	boolean
equals(Object o)	boolean
forEach(Consumer<? super Integer> action)	void
getClass()	Class<?>
hashCode()	int
isEmpty()	boolean
iterator()	Iterator<Integer>
notify()	void
notifyAll()	void
parallelStream()	Stream<Integer>
remove(Object o)	boolean
removeAll(Collection<?> c)	boolean
removeIf(Predicate<? super Integer> filter)	boolean

Métodos Principais:

- add => adiciona elemento
- clear => limpa a lista
- contains => verifica existência
- remove => remove elemento
- size => tamanho da lista

Estruturas de Dados

Como Iterar pelos elementos?

```
Iterator it = arvore.iterator();  
while( it.hasNext() ){  
    System.out.println( it.next() );  
}
```

Método 1 – Usando um Iterator

```
for( Integer a : arvore ){  
    System.out.println(a);  
}
```

Método 2 – utilizando um for

```
Consumer<Integer> cons = (Integer v) -> System.out.println(v);  
arvore.forEach( cons );
```

Método 3 – utilizando um “foreach” com *Consumer* (Java 8)

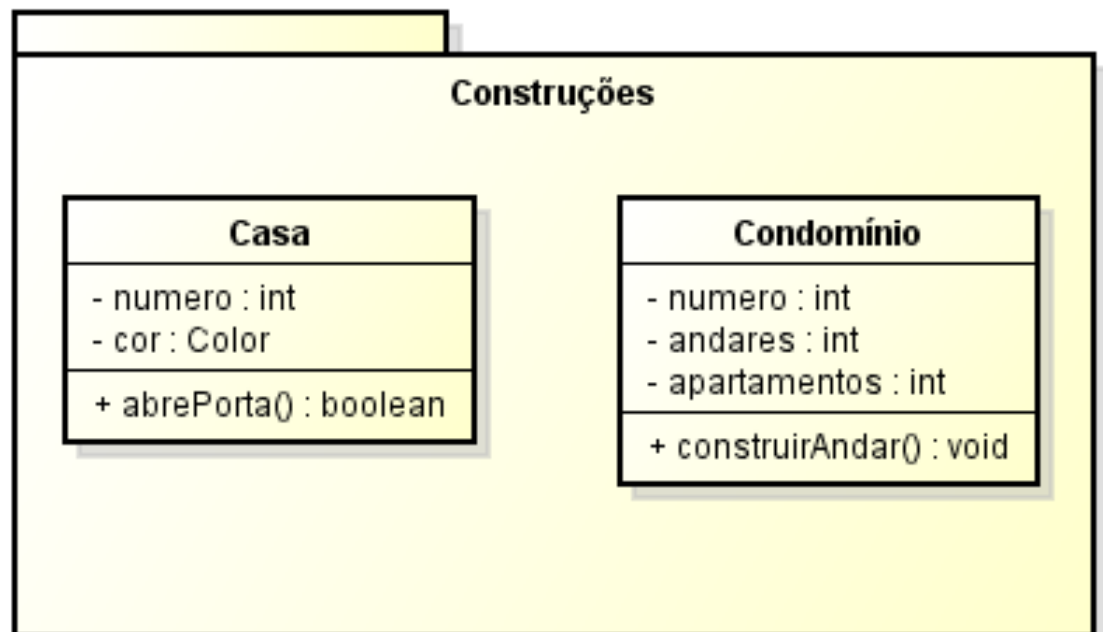
Encapsulamento

- Preocupação na Orientação a Objetos é o encapsulamento:
 - O que só eu posso usar?
 - O que todo mundo pode usar?
- Em C não há como restringir, qualquer um pode alterar o valor de um atributo de uma estrutura
- Em orientação a objetos nós temos os níveis de encapsulamento:
 - **Public:** Todos podem acessar
 - **Private:** Pode ser alterado apenas na classe
 - **Protected:** Apenas classes do mesmo pacote podem alterar (e que herdam)
 - **Package:** (sem modificador) classes do mesmo pacote podem alterar

Encapsulamento

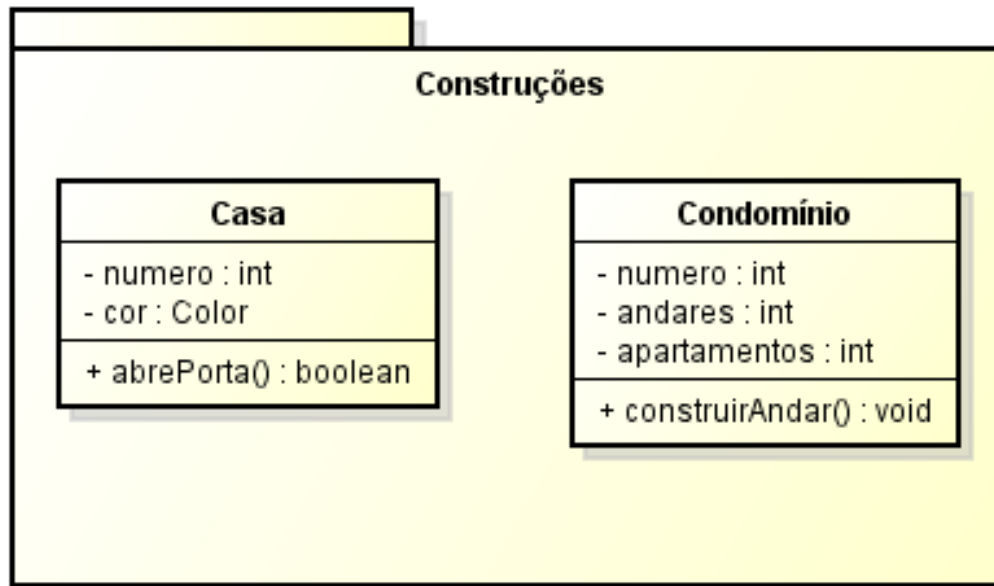
- Preocupação na Orientação a Objetos é o encapsulamento:

- **Public:** +
- **Private:** -
- **Protected:** #
- **Pacote:** ~



Pacotes

- Conceito de Orientação a Objetos utilizado para a organização das classes e demais elementos
 - Agrupar classes de funcionalidades similares ou relacionadas
 - Pacote → Pasta (Diretório)



Estilos e Padrões Arquiteturais

Estilos Arquiteturais:

- Definição de meios para selecionar e apresentar blocos de construção de arquitetura.

Padrões Arquiteturais:

- Projetos de blocos de construção de arquitetura em alto nível que já foram testados e validados.

Estilos Arquiteturais

Taxonomia:

- Sistema de Fluxo de Dados: batch, pipes & filters;
- Sistemas de Chamada e Retorno: estruturado, invocação remota de procedimento, orientado a objetos e sistema em camadas;
- Componentes Independentes: processos comunicantes, invocação implícita (sistemas baseados em eventos);
- Máquinas Virtuais: Interpretador, sistema baseado em regras;
- Sistemas Centrados em Dados: banco de dados, sistemas de hipertexto e blackboards

Sistemas de Fluxo de Dados

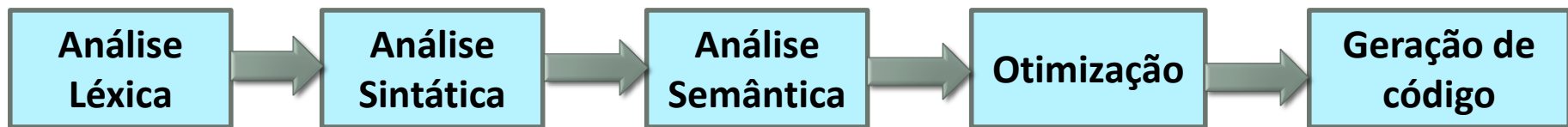
- Modelo originário de sistemas operacionais UNIX:

Pipes: conduzem a saída de um programa como uma entrada de outro programa / sistema.

Batch: execução sequencial de programas independentes (pipeline), um após o outro.

Sistema é composto por várias aplicações / processos.

Compilador



Java – Modelo Híbrido = compilação + interpretação



Sistemas de Fluxo de Dados

Vantagens:

- Rápida prototipação
- Permite reusabilidade de forma simples
- Gera boa flexibilidade (fácil adicionar, alterar ou retirar componentes)
- Eficiência no processamento

Desvantagens:

- Não é bom quando o sistema necessita muitas interações com o usuário (sistemas interativos)
- Não existe compartilhamento de dados
- Gerenciamento de erros só pode ocorrer em cada bloco isolado.

Sistemas de Chamada e Retorno

- Sistema é composto por componentes / módulos / sub-rotinas / funções que podem invocar uns aos outros, podendo gerar um resultado como retorno (comunicação).
- Sistema é composto por uma aplicação / processo que possui vários componentes internos.

Sistemas de Chamada e Retorno

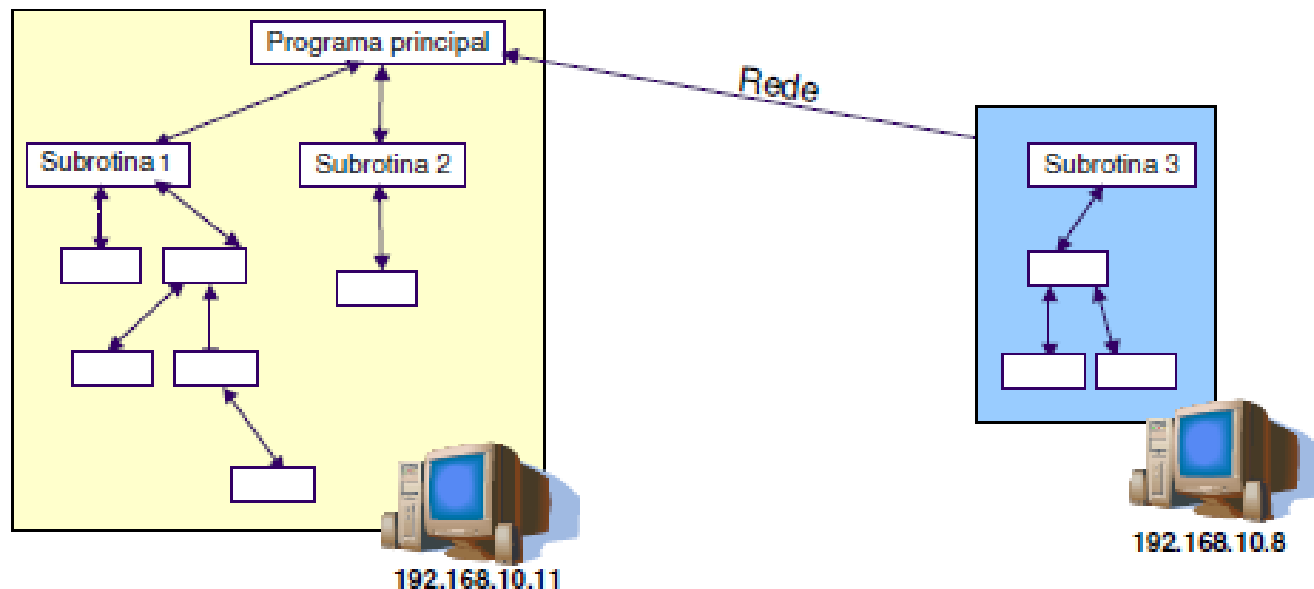
Modelo Estruturado

- O fluxo do sistema começa em uma rotina principal e move-se através de chamadas de procedimentos / funções em uma mesma aplicação e mesmo computador.
- Vantagens:
 - Desenvolvimento dos procedimentos é independente
 - Possibilita o uso compartilhado de memória
- Desvantagens:
 - Reuso em outras aplicações pode ser difícil
 - Não há proteção de variáveis

Sistemas de Chamada e Retorno

Invocação Remota de Procedimento

- O fluxo do sistema começa em uma rotina principal e move-se através de chamadas de procedimentos / funções que podem estar em outro computador ou sistema.



Sistemas de Chamada e Retorno

Invocação Remota de Procedimento

- Vantagens:
 - Desenvolvimento dos procedimentos continua independente
 - Pode gerar um ganho de desempenho quando o sistema usa mais máquinas (+ máquinas = + recursos).
- Desvantagens:
 - A comunicação via rede entre dois computadores é mais lenta – logo, procedimentos que se comunicam muito teriam uma perda de tempo significativa durante as trocas de mensagens.

Sistemas de Chamada e Retorno

Sistemas Orientados a Objetos

- Sistema modelado como um conjunto de objetos que oferecem serviços e possuem interfaces bem definidas.
- Vantagens:
 - A abstração em objetos é simples e bastante poderosa
 - Possibilita o uso compartilhado de memória
- Desvantagens:
 - Reuso em outras aplicações pode ser difícil

Sistemas de Chamada e Retorno

Vantagens:

- Rápida prototipação
- Permite reusabilidade de forma simples
- Gera boa flexibilidade (fácil adicionar, alterar ou retirar componentes)
- Eficiência no processamento

Desvantagens:

- Não é bom quando o sistema necessita muitas interações com o usuário (sistemas interativos)
- Não existe compartilhamento de dados
- Gerenciamento de erros só pode ocorrer em cada bloco isolado.

Atividade

- Cada equipe deve fazer uma versão inicial do projeto de arquitetura do seu sistema baseado no projeto de dados. Faça a escolha dos tipos de dados, encapsulamento e empacotamento.

Bibliografia

- **Básica:**

BEZERRA, E. Princípios de Análise e Projetos de Sistemas com UML. Rio de Janeiro: Campus, 2003.

PRESSMAN, R.S. Engenharia de Software. São Paulo: Makron Books, 2002.

SOMMERVILLE, I. Engenharia de Software. São Paulo: Addison Wesley, 2003.

- **Complementar:**

WARNIER, J. Lógica de Construção de Programas. Rio de Janeiro: Campus, 1984.

JACKSON, M. Princípios de Projeto de Programas. Rio de Janeiro: Campus, 1988.

PAGE-JONES, M. Projeto Estruturado de Sistemas. São Paulo: McGraw-Hill, 1988.