INE5603 Introdução à POO

Prof. A. G. Silva

07 de agosto de 2017

Aula anterior – Otimização de código

- Todas as linguagens têm o propósito de instruir uma máquina digital
- Linguagens de alto nível auxiliam ao programador (humano) nessa tarefa
- Um processo de tradução (interpretação ou compilação) é necessário para produzir linguagem que a máquina entenda (zeros e uns)
- Essa conversão entre linguagens é feita de modo automático, mas não de forma otimizada
- Muitos sistemas de tempo-real são usados, sendo altamente sensíveis a custos, consumo de energia e espaço de armazenamento

Aula anterior – Otimização de código (cont...)

- A otimização de código pode produzir programas:
  - mais rápidos – possibilita o uso de processadores mais lentos, com menor custo, e com menor consumo de energia
  - mais leves – menos uso de memória, menos custo e menos consumo de energia

Aula anterior – Assembly

- Programa escritos em Assembly permitem um nível de eficiência potencialmente mais elevado, porém
  - Difícil depuração (debug) e manutenção
  - Ciclo de desenvolvimento longo
  - Dependentes do processador – código não portável
- Melhor dos mundos: escrever em Assembly apenas trechos de código críticos em desempenho

Aula anterior – exemplo 1

- Eliminação de sub-expressões comuns – menos código para executar!

```
b:
        t6 = 4 * i
        x  = a[t6]
   ⟶    t7 = 4 * i
        t8 = 4 * j
        t9 = a[t8]
   ⟶    a[t7]  = t9
        t10    = 4 * j
        a[t10] = x
        goto b
```

```
b:
        t6 = 4* i
        x  = a[t6]
        t8 = 4 * j
        t9 = a[t8]
        a[t6] = t9
        a[t8] = x
        goto b
```

Aula anterior – exemplo 2

- Eliminação de "código-morto"

```
debug = 0;
...
if (debug){
   print .....
}
```

Aula anterior – exemplo 3

- Eliminação de custo de cálculos

```
     j = 0
label_XXX
     j  = j + 1
     t4 = 11 * j
     t5 = a[t4]
     if (t5 > v) goto label_XXX
```

```
     t4 = 0
label_XXX
     t4 += 11
     t5 = a[t4]
     if (t5 > v) goto label_XXX
```

Aula anterior – exemplo em Assembly

- Na arquitetura ARM, por exemplo, é possível usar *flags* ao efetuar uma operação matemática

```
int checksum_v1(int *data)
{
    unsigned i;
    int sum=0;

    for(i=0;i<64;i++)
         sum += *data++;

    return sum;
}
```

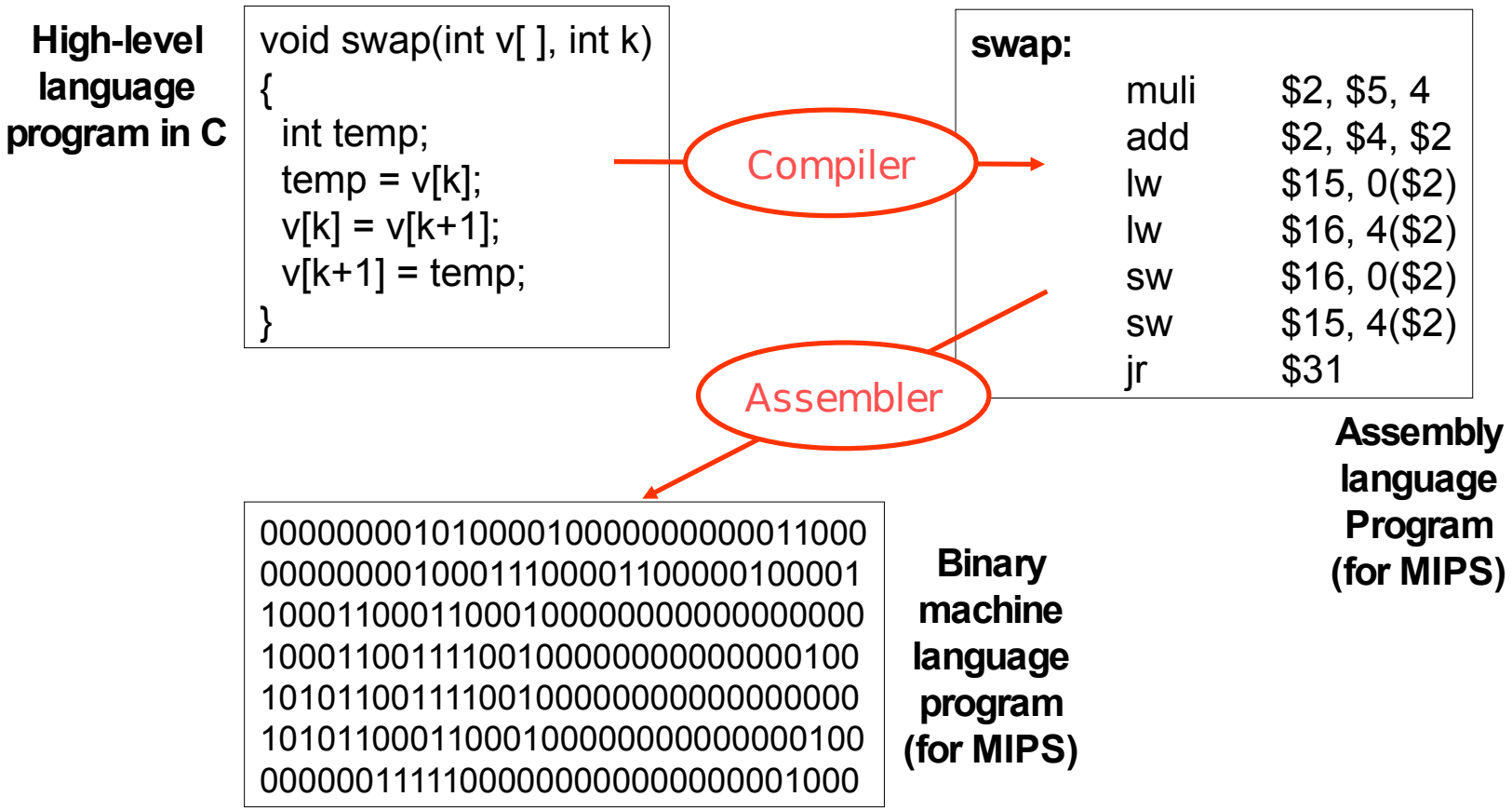
```
     MOV  r2, r0;r2=data
    MOV  r0, #0; sum=0
    MOV  r1, #0; i=0
L1  LDR  r3,[r2],#4;   r3=*(data++)
    ADD  r1, r1, #1;   i=i+1
    CMP  r1, 0x40;     cmp r1, 64
    ADD  r0, r3, r0;   sum +=r3
    BCC  L1;       if i < 64, goto L1
    MOV  pc, lr;   return sum
```

```
int checksum_v2(int *data)
{
    unsigned i;
    int sum=0;

    for(i=63;i >= 0;i--)
         sum += *data++;

    return sum;
}
```

```
     MOV  r2, r0;r2=data
    MOV  r0, #0; sum=0
    MOV  r1, #0x3f;        i=63
L1  LDR  r3,[r2],#4;   r3=*(data++)
    ADD  r0, r3, r0;   sum +=r3
    SUBS  r1, r1, #1;  i--, set flags
    BGE  L1;  if i >= 0, goto L1
    MOV  pc, lr;       return sum
```

Processo de compilação

- Conversão em linguagem de alto nível para linguagem de máquina



Motaz K. Saad (2007)

Processo de compilação

Alto Nível	Baixo Nível	
Linguagem de Alto Nível	**Linguagem Assembly (*Mnemônica*)**	**Linguagem de Máquina**
val2 = val1+val2	LOAD R1, val1	0010 0001 1110
	LOAD R2, val2	0010 0010 1111
	ADD R1, R2	0001 0001 0010
	STORE R1, val2	0011 0001 1111

CÓDIGO FONTE $\Rightarrow$ TRADUTOR $\Rightarrow$ CÓDIGO OBJETO

```
val2 = val1 + val2;
```

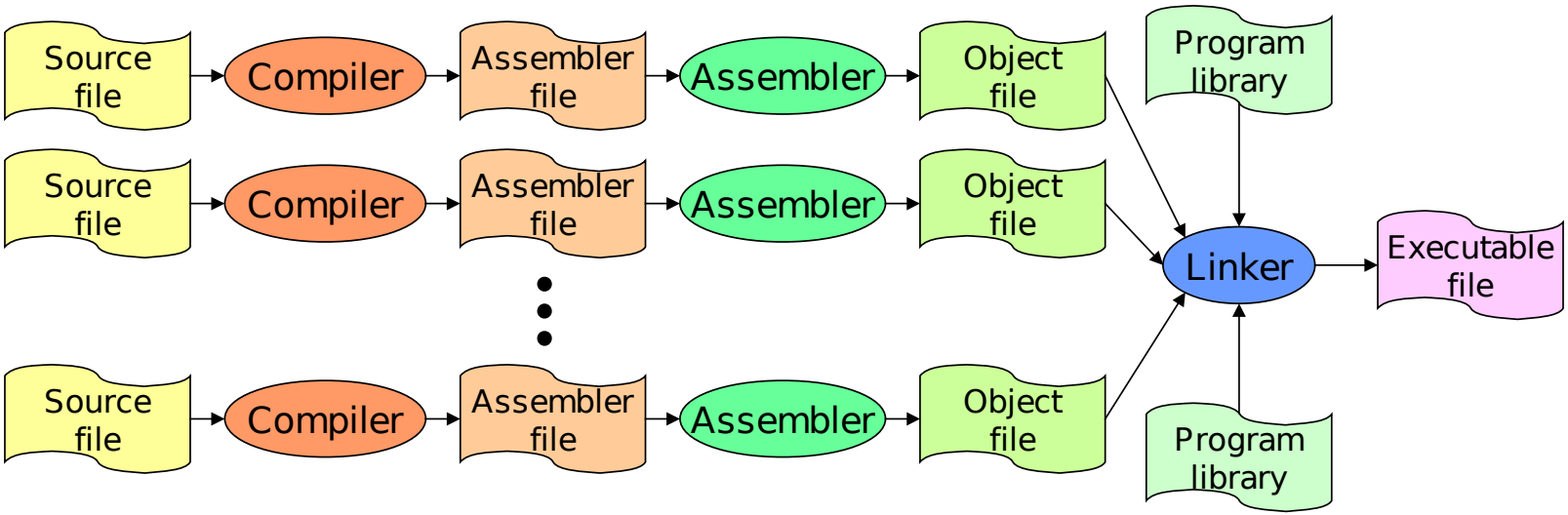
(linguagem de programação)

```
00010110111001011001011010 ...
```

("executável")

Processo de edição, compilação, ligação e execução

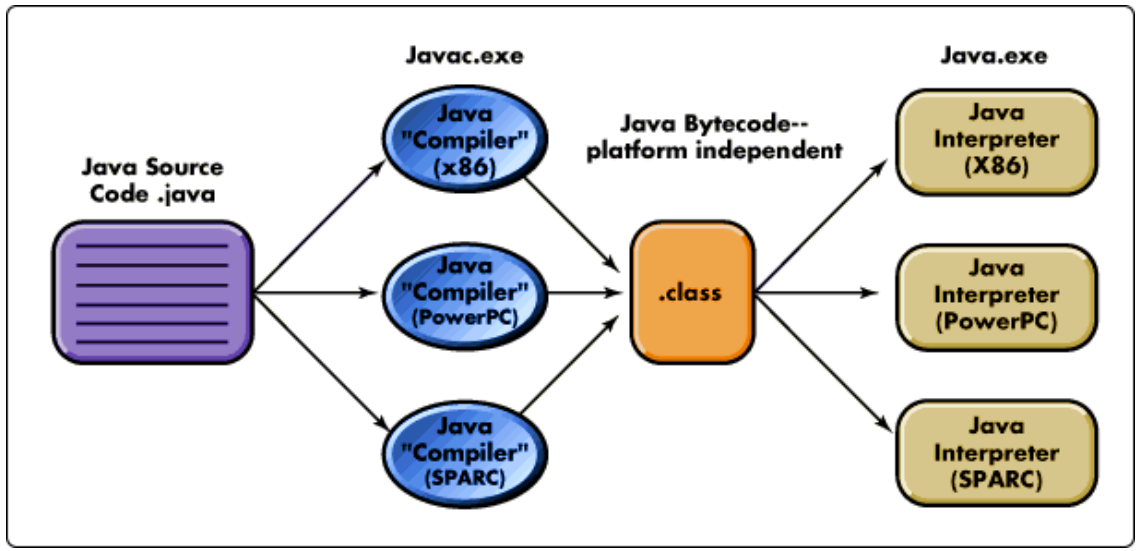
- Conversão de código-fonte em arquivos executáveis



Motaz K. Saad (2007)

Compilação/interpretação em Java

- Geração do bytecode: `javac Principal.java`
- Execução do programa: `java Principal`



http://support.novell.com/techcenter/articles/ana19970701.html

Linguagens, compiladores e interpretadores

- Compiladores são tradutores
- Na **compilação** o programa escrito na linguagem fonte é traduzido para linguagem máquina e depois ligado e carregado para ser executado
  - Exemplos: https://pt.wikipedia.org/wiki/Linguagem_compilada
- Na **interpretação** o programa fonte é traduzido e executado instrução a instrução, de modo interativo
  - Exemplos: https://pt.wikipedia.org/wiki/Linguagem_interpretada
  - Vantagens: o ciclo de escrita, execução, modificação é mais rápido
  - Desvantagens: a execução é mais lenta

Linguagens, compiladores e interpretadores

- Outras ferramentas de software úteis
  - **Depurador:** ajuda na correção de erros de execução (funções para acompanhamento e impressão de dados de programa em execução)
  - **Profilador:** ajuda na análise de gastos em tempo e uso de memória

Objetos

- Modelo (abstração) da situação real
- Objetos (entidades) integram o problema
- Significado dentro do contexto do problema
- Exemplos de objetos:
  - Um livro
  - A página de um livro
  - Uma viagem
  - Uma data
  - O clima de uma região
  - Endereço de alguém
  - Uma festa
  - Uma música
  - Um conceito sobre algo
  - Um pedido de compra
  - O salário de um funcionário
  - ...

Objetos

- Objetos devem ter alguma utilidade ou propósito
- Cada objeto tem identidade própria
  - Exemplo: cada exemplar de um mesmo livro na biblioteca é um objeto
- Em termos de programação, um objeto é uma abstração de uma entidade do mundo real
  - Própria existência, identificação, características de composição
  - E possui alguma utilidade, podendo executar determinados serviços
  - É um elemento do programa que ocupa espaço em memória após a criação

Objetos

- Em termos de programação, um objeto possui
  - Identificação (nome)
  - Características (atributos).
    - ★ Exemplo para umaLapiseira: cor predominante, peso, comprimento do grafite (simples valor), sistema de avanço do grafite (outro objeto), etc
  - Utilidade (ações ou serviços) com base em solicitações. Denomina-se comportamento ao conjunto de ações.

Objetos

- **Exemplo 1** – Objeto umaPorta para abstração da porta de uma sala:
  - Atributos (composição):
    - *Cor* (valor)
    - *Altura* (valor)
    - *Largura* (valor)
    - *Fechadura* (objeto)
  - Serviços (comportamento):
    - *Abrir*
    - *Fechar*

- **Exemplo 2** – Objeto meuRelogio para abstração, do ponto de vista do usuário, de um relógio:
  - Atributos (composição):
    - *Peso* (valor)
    - *Custo* (valor)
    - *Pulseira* (objeto)
    - *Ponteiros* (objeto)
    - *Pilha* (objeto)
  - Serviços (comportamento):
    - *Iniciar marcação de horas*
    - *Informar horas*
    - *Informar data*
    - *Corrigir horas*
    - *Trocar pilha*

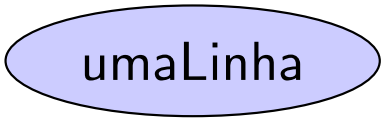
Objetos

- **Exemplo 3** – Objeto umaLinha para abstração de uma linha de ônibus:
  - Atributos:
    - *Nome da linha*
    - *Número de horários*
    - *Horários de partida*
    - *Itinerário*
  - Serviços:
    - *Incluir novo horário*
    - *Eliminar horário*
    - *Modificar itinerário*

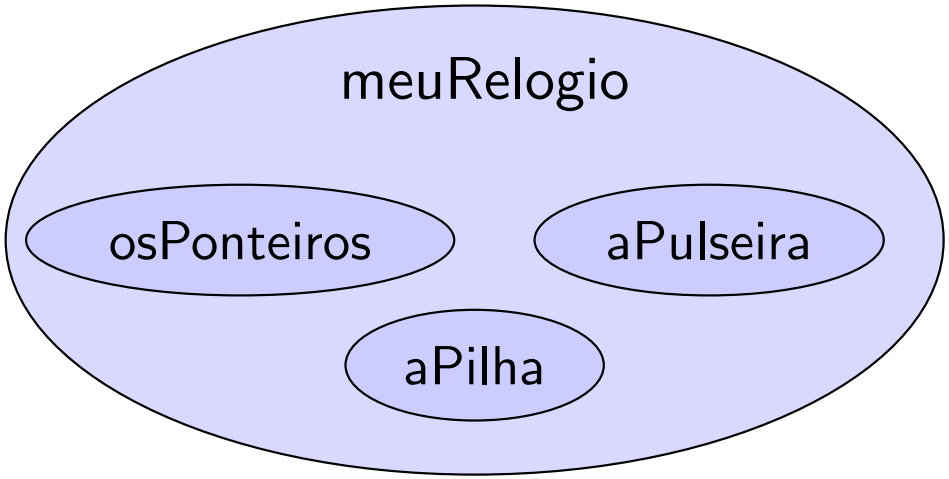
- **Exemplo 4** – Objeto umaCidade para abstração de uma cidade real:
  - Atributos:
    - *Nome*
    - *Área*
    - *Nome do prefeito*
    - *Prefeitura*
    - *Plano diretor*
  - Serviços:
    - *Mudar prefeito*
    - *Alterar plano diretor*
    - *Alterar sua área*

Representação visual

- Círculo ou elipse contendo a identificação do objeto. Exemplo:

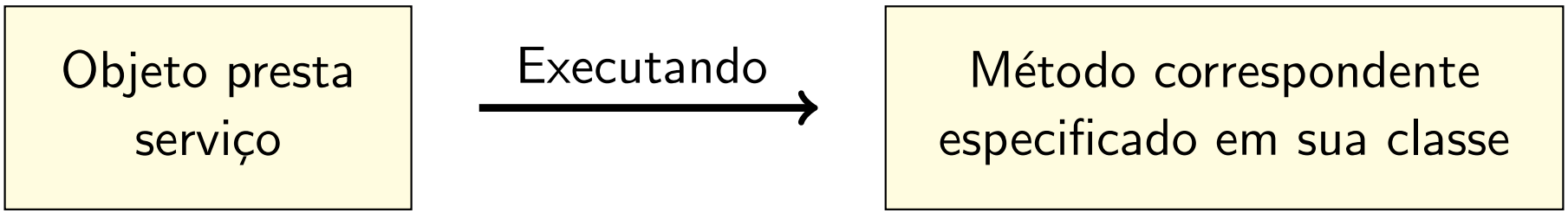


- Exemplo de atributo de objeto como um outro objeto (operação de agregação):



Classes

- Características em comum de objetos
- Exemplo: relógio de José e o relógio da Maria são objetos distintos com mesmos atributos e serviços, ou pertencentes à mesma classe Relógio
- Uma classe ("ideia") especifica quais os atributos e serviços de todos os objetos ("concreto")
- A todo serviço corresponderá a execução de uma sequência de instruções denominada método



Classes

- O processo de modelagem pode ser dividido em:
  - Análise/projeto – determinar o que é relevante para o problema
  - Implementação – expressar classes e seus relacionamentos em uma linguagem de programação
- A implementação é dependente da análise/projeto
- Na análise/projeto devem ser definidas, de maneira mais adequada possível, as classes que compõem o problema, bem como suas características específicas
- Uma definição incorreta na análise/projeto implicará em uma implementação também incorreta e um consequente custo associado

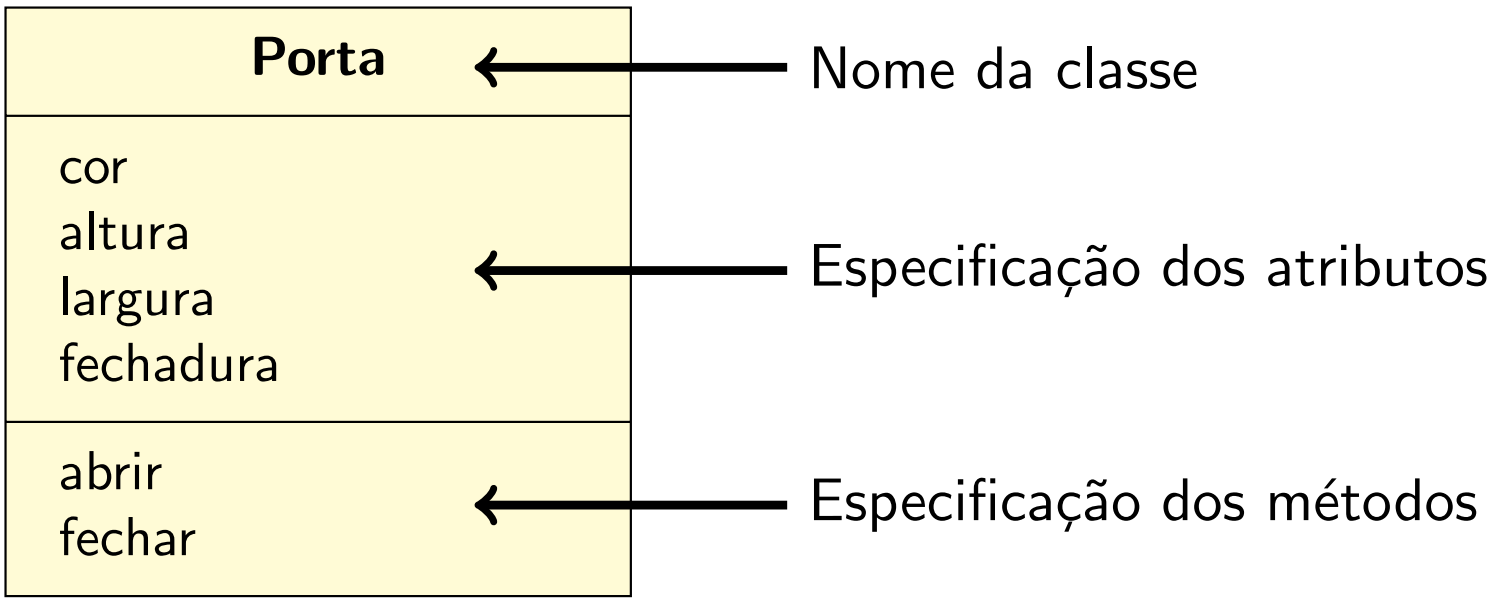
Classes

- Em suma, em uma classe, para toda instância (objeto) da mesma, define-se:
  - Quais são os seus atributos (estrutura do objeto)
  - Como são representados esses atributos (valores simples, objetos)
  - Quais métodos esse objeto poderá executar (serviços que o objeto pode prestar)

Representação de Classes

- A representação final de uma classe é um texto escrito em uma linguagem de programação

- Resultado das etapas de análise/projeto e implementação, com possível representação visual:



Aspectos de linguagem e computador

- O diagrama de classes, apesar de bastante útil para o projetista, pode não servir ao computador
- Do ponto de vista do computador, as classes e suas interações devem ser expressas como um texto em uma linguagem
- Este texto deve apresentar a organização de atributos e métodos
- Para cada método, especificar a sequência de ações a executar
- Podem ser expressas em várias linguagens de programação

Aspectos de linguagem e computador

- Utilizamos a linguagem Java desenvolvida pela Sun Microsystems em 1995
- Construção do modelo de resolução de um problema em um computador, sob a ótica da orientação a objetos
- Apresentaremos e discutiremos as características básicas e seus principais comandos e recursos de Java para resolução de problemas mais simples/comuns

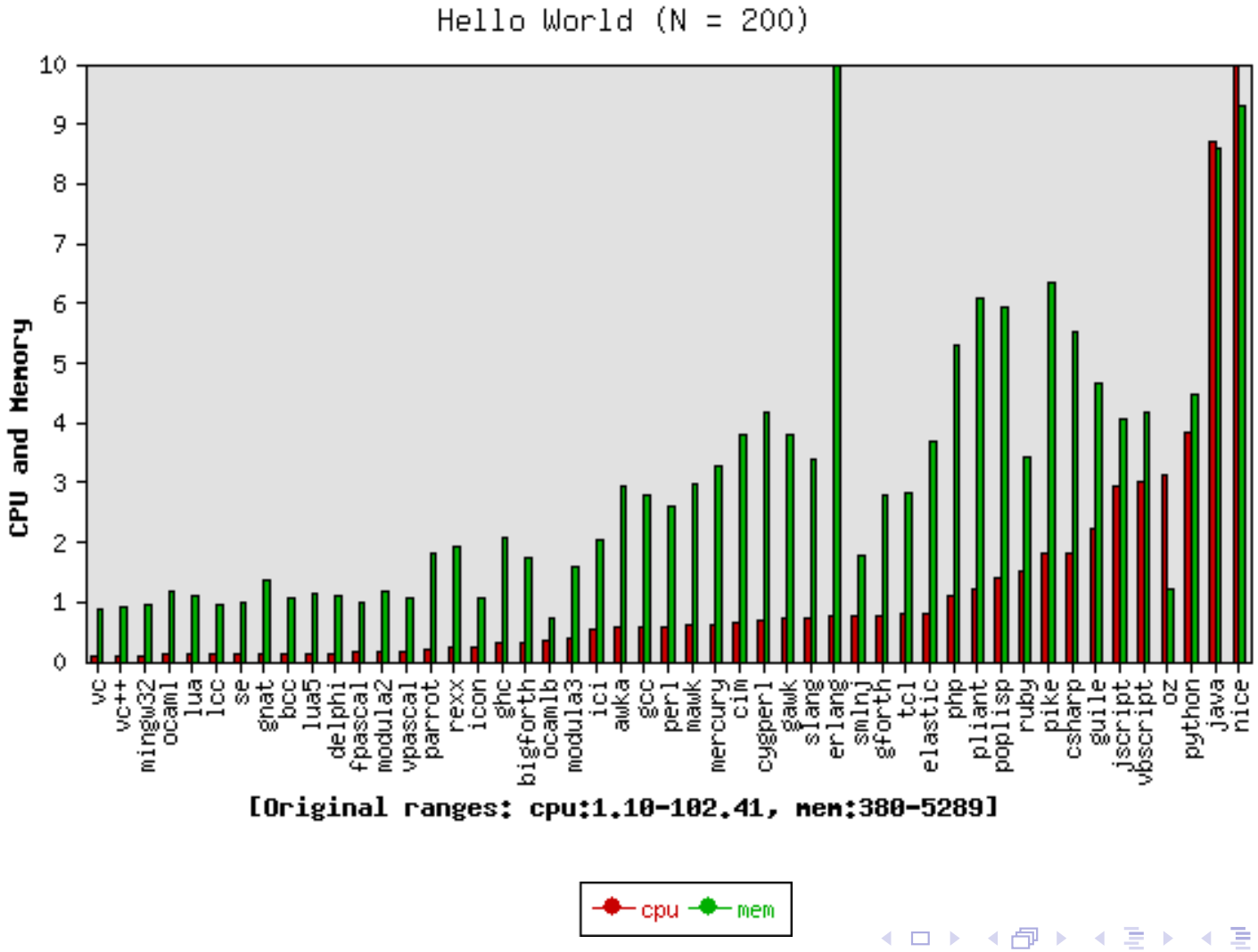
Exemplos iniciais em Java

- Método `main`

```
public class OlaMundo {
    /*
     * Metodo que executa o programa
     * public = e' visto em qualquer lugar da aplicacao
     * static = e' iniciado automaticamente pela JVM, sem
         precisar de uma instancia, sendo comum a todos
        os objetos da mesma classe
     * void = metodo sem retorno (retorno vazio)
     * main = nome do metodo (este e' obrigatorio),
        recebe como parametro um array de String.
     * String[] args = array de argumentos que podem ser
        repassados na chamada do programa.
     */
    public static void main(String[] args) {
        System.out.println("Ola, Mundo!"); //Imprime na
            tela a frase
    }
}
```

Comparação de desempenho

- Hello World – http://dada.perl.it/shootout/hello.html



Algoritmo

- Partes de um algoritmo
  - **Entrada de dados** – informações necessárias à execução, fornecidas em tempo de execução ou embutidas, por interação com o usuário ou por arquivos
  - **Processamento de dados** – avaliação de expressões algébricas, relacionais e lógicas, assim como estruturas de controle (condição e/ou repetição)
  - **Saída de dados** – resultados de processamento enviados a dispositivos de saída (monitor, impressora, ou memória)

Algoritmos (I)

- Qualquer problema de computação pode ser resolvido executando uma série de ações em uma ordem específica

- Um algoritmo é um procedimento para resolver um problema em termos
  - das ações a executar e
  - da ordem em que essas ações executam

  Considere o algoritmo para se levantar e ir ao trabalho:
  - (1) Levantar da cama; (2) tirar o pijama; (3) tomar banho; (4) vestir-se; (5) tomar café da manhã; (6) dirigir o carro até o trabalho
  - Imagine a troca de ordem destes passos

- Especificar a ordem em que as instruções (ações) são executadas em um programa é chamado controle de programa

Algoritmos (II)

- Sequência finita de passos, em um encadeamento lógico, que leva à execução de uma tarefa

- Claro e preciso. Por exemplo: "somar dois números":
  - Ler primeiro número, atribuindo-o a A
  - Ler segundo número, atribuindo-o a B
  - Somar A com o número B, atribuindo este resultado a C

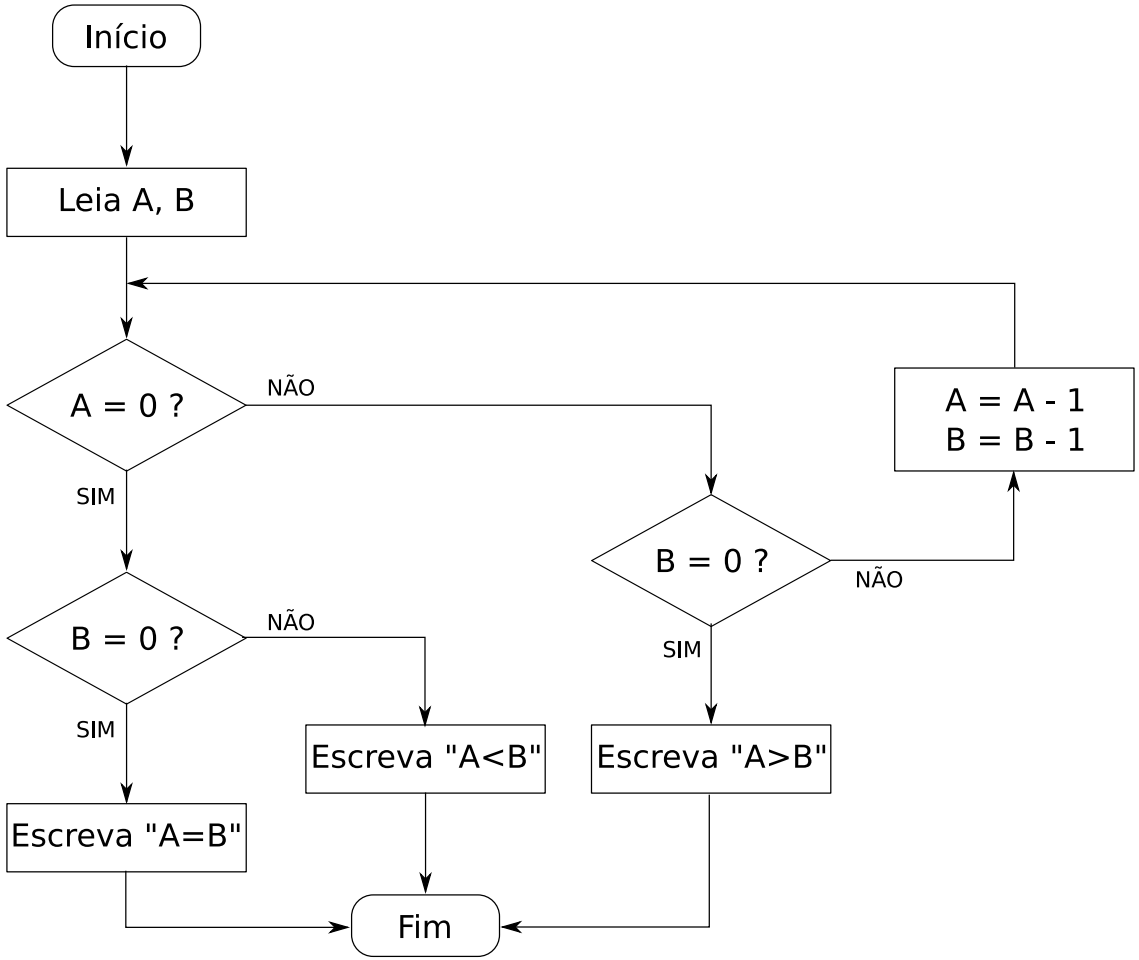
- Um motorista que necessita efetuar a troca de um pneu furado segue uma rotina para realizar essa tarefa:
  - (1) Verifica qual pneu está furado; (2) Posiciona o macaco para levantar o carro; (3) Pega o estepe; (4) Solta os parafusos; (5) Substitui o pneu furado; (6) Recoloca os parafusos; (7) Desce o carro; (8) Guarda o macaco e o pneu furado

Fluxograma (I)

- Um diagrama de atividades modela o fluxo de trabalho (também chamado atividade) de uma parte de um sistema de software
- Podem incluir uma parte de um algoritmo, como a estrutura de sequência ou de decisão
- Símbolos existentes
  - Estado de ação (retângulos com lados esquerdo e direito arredondados) – representa uma ação a realizar
  - Losangos – representa uma decisão sobre a continuidade do fluxo
  - Círculos pequenos (ou retângulo com cantos arredondados) – sólido com o estado inicial; sólido envolvido por uma cincunferência com o estado final
- Esses símbolos são conectados por setas de transição, que representam o fluxo da atividade, isto é, a ordem em que as ações devem ocorrer
- Ajudam a desenvolver e representar algoritmos
- Mostram claramente como as estruturas de controle funcionam

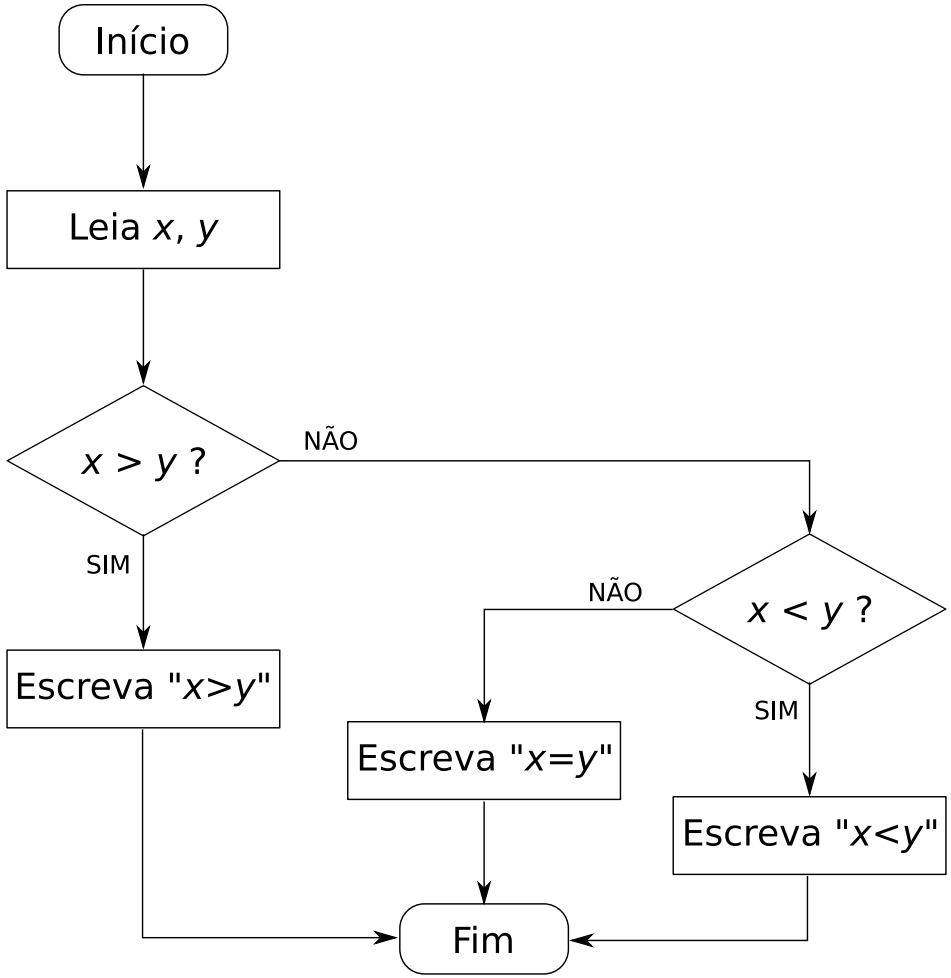
Fluxograma (II)

- Exemplo 1 – comparação entre dois inteiros, A e B, sem utilizar qualquer operador relacional



Fluxograma (II)

- Exemplo 2 – comparação entre x e y



Pseudocódigo (I)

- Pseudocódigo é uma linguagem informal que ajuda você a desenvolver algoritmos sem se preocupar com os detalhes estritos da sintaxe da linguagem Java
- Particularmente útil para desenvolver algoritmos que serão convertidos em partes estruturadas de programas Java
- Similar ao português do cotidiano (às vezes chamado de "portugol")
- Ajuda a "estudar" um programa antes de tentar escrevê-lo em uma linguagem de programação como Java
- O pseudocódigo cuidadosamente preparado pode ser facilmente convertido em um programa Java correspondente

Pseudocódigo (II)

Algoritmo 1 Comparação entre x e y

```
1:  leia x, y
2:  se x > y então
3:      escreva "x é maior"
4:  senão
5:      se x < y então
6:          escreva "y é maior"
7:      senão
8:          escreva "x e y são iguais"
9:      fim se
10: fim se
```

Programa (I)

- Cada programa é formado combinando o número de instruções de sequência, instruções de seleção (três tipos) e instruções de repetição (três tipos) conforme apropriado para o algoritmo que o programa implementa
- Podemos modelar cada instrução de controle como um diagrama de atividades
- O estado inicial e o estado final representam um ponto de entrada e um ponto de saída da instrução de controle, respectivamente
- Instruções de controle de entrada única e saída única
- Empilhamento de instruções de controle – conectam o ponto de saída de uma instrução ao ponto de entrada da instrução seguinte
- Aninhamento de instruções de controle – instrução de controle dentro de outra.

Programa (II)

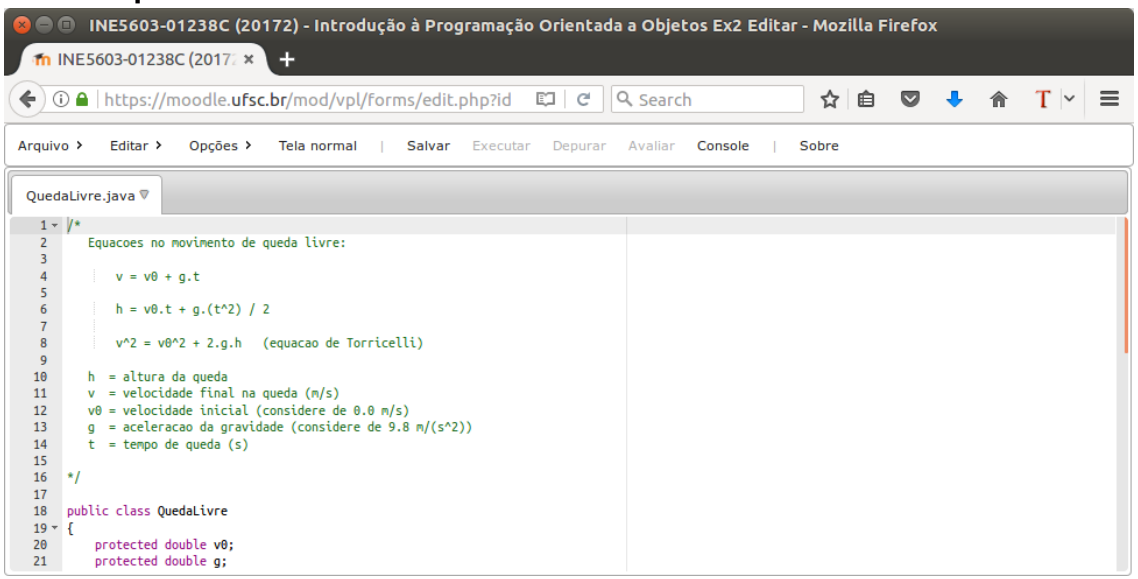
```
import java.util.Scanner;

public static void main(String[] args) {
    int x, y;  // isto e' um comentario de linha
    Scanner s = new Scanner(System.in);  //entrada de dados

    /* isto e' um comentario
       em bloco */

    System.out.println("digite x: ");
    x = s.nextInt();
    System.out.println("digite y: ");
    y = s.nextInt();
    if (x > y) {
        System.out.println("x e' maior");
    } else if (x < y) {
        System.out.println("y e' maior");
    } else {
        System.out.println("x e' y sao iguais");
    }

}
```

Estrutura básica de um programa em Java

- Para que certas funcinalidades/métodos (p. ex: entrada e saída) sejam acessíveis, é necessário incluir/importar algumas bilbiotecas (p. ex: `import java.util.Scanner;`).
- Todo programa em Java inicia sua execução pela função `main()`, escrita em uma classe com o mesmo nome do arquivo (.java).
- Instruções são finalizadas com ponto-e-vírgula.
- Os blocos de instruções são delimitados por chaves.
- Linhas de comentários são iniciadas por duas barras `//`
- Blocos de comentários são delimitados por `/*` e `*/`

Moodle & VPL

- O Moodle tem suporte à programação diretamente pelo navegador
- VPL – *Virtual Programming Lab*
  - Para conhecer mais:
    http://vpl.dis.ulpgc.es/
- Utilizaremos este ambiente para entrega de exercícios
  - Exemplo



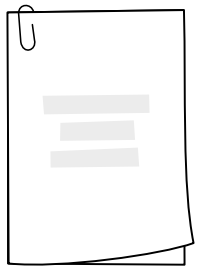
Dicas iniciais

- Termine todas as instruções com ponto-e-vírgula
- Sempre salve o programa antes de compilar
- Sempre compile o programa antes de executar
- Quando ocorrer um erro de compilação, dê um duplo clique (se for um ambiente integrado) sobre a mensagem de erro ou simplesmente identifique a posição (linha/coluna) de erro para efetuar sua correção
- Verifique também a linha anterior, que pode ser a responsável pelo erro, especialmente se faltar o ponto-e-vírgula
- Use comentários, iniciados por //, para documentar a implementação facilitando o seu entendimento

Ferramentas

- Compiladores
  - Javac, incluido na Java Development Kit (JDK) da Oracle Corporation
  - GCJ (GNU Compiler for Java) é um compilador estático parte do GCC
- Ambientes de desenvolvimento: BlueJ, JCreator, jEdit, Eclipse
- IDE completas (programas profissionais): Eclipse (IBM), JBuilder (Borland), JDeveloper (Oracle), NetBeans (Sun Microsystems)

Exercício



- Exercício 1 no Moodle: ex01.pdf (*para entrega*)