Universidade da Região da Campanha
Semana Acadêmica da Informática

Introdução à linguagem Java

Alexsander da Rosa <alexsand@urcamp.tche.br>
http://www.urcamp.tche.br/~alexsand/

Bagé-RS, 03 de novembro de 1999

Sumário

1. O fenômeno Java

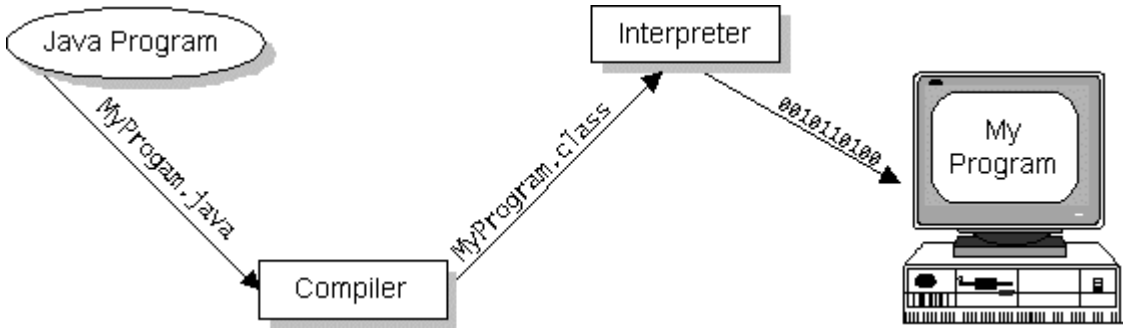
Em 23/05/1995, John Gage, diretor da *Sun Microsystems*, e Marc Andreessen, VP da *Netscape*, anunciaram que a linguagem já era realidade e seria incorporada ao *Navigator*.

Patrick Naughton, Mike Sheridan, e James Gosling criaram a linguagem em 1991, no "*Green Project*" da Sun.

Uma equipe de 13 pessoas (o "*Green Team*") se trancou num escritório na Sand Hill Road em Menlo Park. Eles cortaram as comunicações com a Sun e trabalharam 18 meses sem parar.

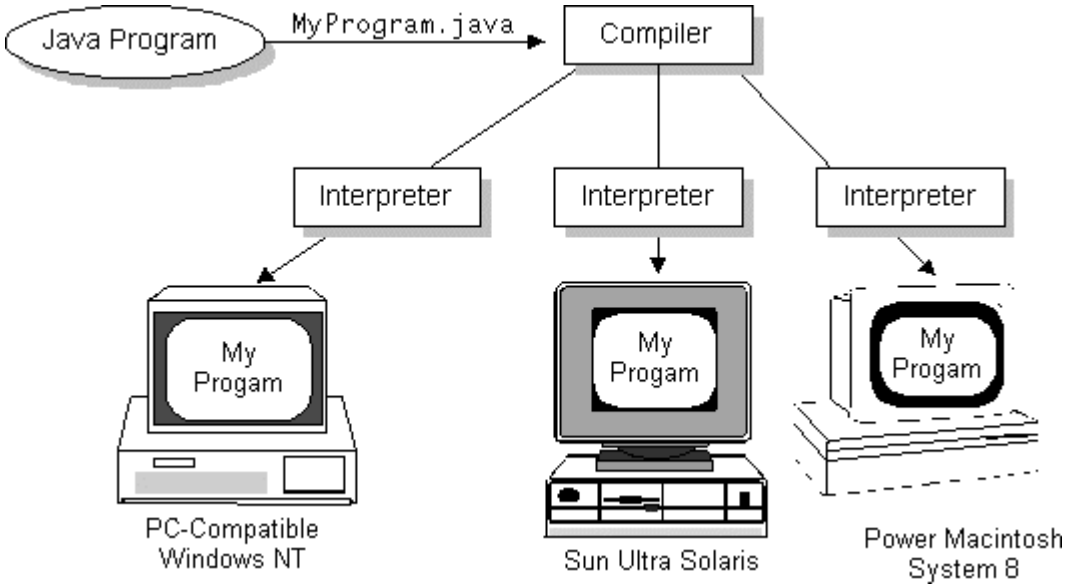
Java é tanto uma linguagem de programação de alto nível quanto uma plataforma.

Como linguagem, Java é uma linguagem orientada a objetos, independente de arquitetura, portável, robusta, segura, interpretada, distribuída, etc.



Java é tanto compilada quanto interpretada. O compilador transforma o programa em *bytecodes*, que são transformados em linguagem de máquina pelo interpretador.

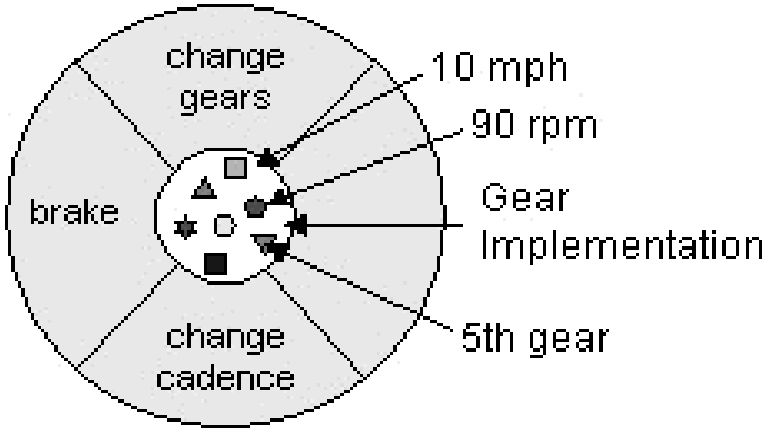
Os *bytecodes* são o que torna possível o *slogan* criado pela Sun: "*Write once, run anywhere*".



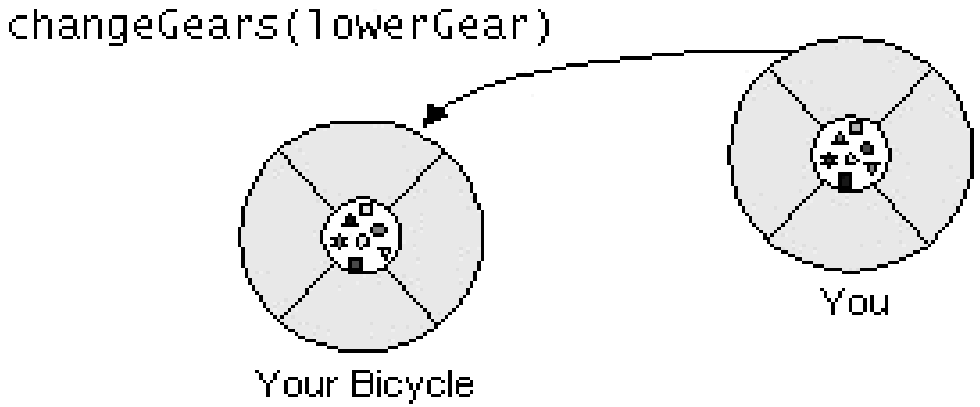
Como plataforma, Java compreende uma JVM (Java Virtual Machine) e uma API (Java API).

2. Programação Orientada a Objetos

Conceitos básicos: objetos, mensagens, classes e herança.



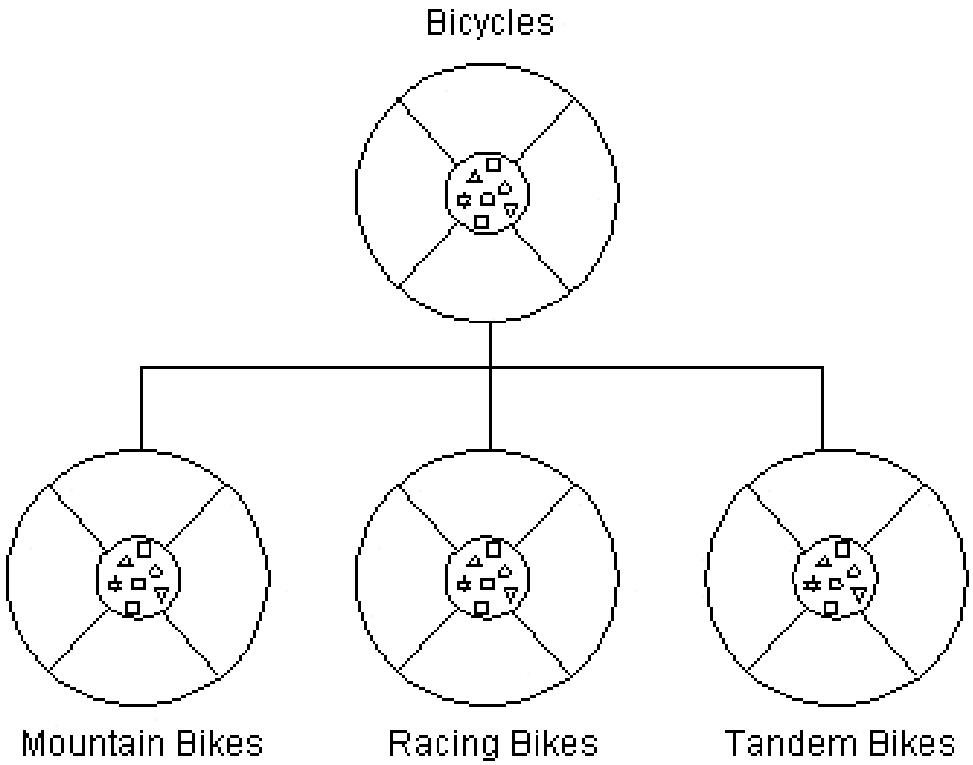
Objeto é um conjunto de variáveis e métodos relacionados, e tem *estado* e *comportamento*. As variáveis armazenam seu estado e seus métodos implementam seu comportamento.



O objetos interagem entre si através da troca de *mensagens*. Se um objeto quer executar um método de outro, ele envia uma mensagem. Essa mensagem contém três itens: o *nome do objeto* destino, o *nome do método* e os *parâmetros*.

Classe é um protótipo que define as variáveis e métodos comuns a todos os objetos de um certo tipo. Um objeto pode ser visto como uma *instância* de uma classe.

A *herança* permite que classes sejam definidas a partir de outras classes, agrupando características comuns.



3. Programando em Java

Lembrando que a linguagem Java é orientada a objetos, temos que ter no mínimo uma classe em cada programa. Exemplo:

```
class HelloWorldApp {
   public static void main(String[] args) {
      System.out.println("Hello World!");
       }
   }
```

Para executar esse programa, basta digitá-lo em um arquivo com o mesmo nome da classe (ou seja, **`HelloWorldApp.java`**), compilar (usando **`javac HelloWorldApp.java`**) e depois usar o interpretador Java (digitando **`java HelloWorldApp`**).

Para criar uma classe em Java, a sintaxe é a que segue. Um mesmo programa pode conter várias classes.

```
class NomeDaClasse {
    ...
    }
```

No nosso pequeno programa-exemplo, a única classe existente se chama **`HelloWorldApp`** e é delimitada pelas chaves.

```
class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!");
         }
    }
```

A linguagem Java é muito parecida com a linguagem C++.

Programas em Java sempre começam a executar pelo método **`main`** da classe que tem o mesmo nome do arquivo.

```
class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!");
         }
    }
```

A palavra **`public`** indica que o método **`main`** pode ser chamado de qualquer objeto; **`static`** indica que este é um método de classe, e **`void`** indica que o método não retorna valor.

Os argumentos de linha de comando são passados através de um vetor de *strings* **`args`**. Isto equivale aos **`int argc`** e **`char *argv[]`** do C/C++. O número de argumentos pode ser obtido pelo atributo **`args.length`** (de **`String`**).

O programa-exemplo usa uma outra classe, **`System`**, que é parte da API (*Application Programming Interface*).

```
class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!");
         }
    }
```

O trecho **`System.out`** indica que este é o nome completo da *variável de classe* **`out`**. Essa variável aponta (ou *referencia*, em Java) para uma *instância* da classe **`PrintStream`**, que por sua vez possui um *método* chamado **`println`**.

Por exemplo, versão do ambiente Java está em:

```
System.getProperty("java.version");
```

4. Objetos e Classes em Java

O código abaixo implementa uma classe **`SimplePoint`**, que representa um ponto no espaço 2D (na tela, por exemplo).

```
public class SimplePoint {
   public int x = 0;
   public int y = 0;
   }
```

A classe definida acima contém duas *variáveis membro*, *x* e *y*. A palavra-chave **`public`** antes do tipo **`int`** indica que qualquer classe pode acessar livremente o conteúdos de *x* e de *y*.

Para criar um *objeto*, cria-se uma *instância* da classe:

```
SimplePoint P = new SimplePoint();
```

Uma classe que representa um retângulo:

```
public class SimpleRectangle {
    public int width = 0;
    public int height = 0;
    public SimplePoint origin = new SimplePoint();
    }
```

Assim como **`width`** *é um* inteiro e **`height`** *é um* inteiro, o membro **`origin`** é um **`SimplePoint`**. Por outro lado, um objeto da classe **`SimpleRectangle`** *tem um* **`SimplePoint`**.

Os conceitos de *é-um* e *tem-um* são muito importantes, pois somente um *objeto* que *é-um* **`SimplePoint`** pode ser usado onde um *objeto* do tipo **`SimplePoint`** é esperado.

Uma versão melhorada da classe **`SimplePoint`** poderia ser:

```
public class Point {
   public int x = 0;
   public int y = 0;
   // um construtor!
   public Point(int x, int y) {
      this.x = x;
      this.y = y;
      }
   }
```

Que permite criar um ponto com a linha de código:

```
Point P2 = new Point(44, 78);
```

Os valores 44 e 78 são passados para *x* e *y* respectivamente.

Uma classe mais completa ficaria assim:

```
public class Rectangle {
    public int width = 0;
    public int height = 0;
    public Point origin;
    public Rectangle() { // quatro construtores
        origin = new Point(0, 0);
        }
    public Rectangle(Point p) {
        origin = p;
        }
    public Rectangle(int w, int h) {
        this(new Point(0, 0), w, h);
        }
    public Rectangle(Point p, int w, int h) {
        origin = p;
        width = w;
        height = h;
        }

//----- segue... parte 1 de 2
```

```
// um metodo para mover o retangulo
public void move(int x, int y) {
    origin.x = x;
    origin.y = y;
    }
// um metodo para calcular a area do retangulo
public int area() {
    return width * height;
    }
// limpeza (ou o destrutor)
protected void finalize() throws Throwable {
    origin = null;
    super.finalize();
    }
} //----- final, parte 2 de 2
```

Assim, as linhas de código abaixo realizam três operações: *declaração*, *instanciação* e *inicialização*.

```
Rectangle rect1 = new Rectangle(100, 200);
Rectangle rect2 = new Rectangle(new Point (44,78));
Rectangle rect3 = new Rectangle(new Point (44,78),100,200);
```

5. Criando Applets

Uma *Applet* é um pequeno programa em Java que pode ser executado de dentro de um *browser* ou através do *AppletViewer*.

Para criar uma *Applet*, é necessário criar uma classe que seja uma *subclasse* da classe **`Applet`**. Para executar a *Applet*, é preciso carregar um arquivo HTML com a *tag* **`<APPLET>`**.

```
import java.applet.Applet;
import java.awt.Graphics;

public class HelloWorld extends Applet {
   public void paint(Graphics g) {
      g.drawString("Hello world!", 50, 25);
      }
   }
```

Por exemplo, para executar a *Applet* **`HelloWorld`**:

```
<HTML>
<HEAD><TITLE> A Simple Program </TITLE></HEAD>
<BODY>
Here is the output of my program:
<APPLET CODE="HelloWorld.class" WIDTH=150 HEIGHT=25>
</APPLET>
</BODY>
</HTML>
```

O *browser* procura pela classe no mesmo diretório onde está o arquivo HTML, faz o *download*, cria uma *instância* da classe e chama os métodos *init* e *start* da *Applet*.

Os parâmetros **`WIDTH`** e **`HEIGHT`** specificam o tamanho em *pixels* da área de tela destinada à *Applet*.

O ciclo de vida de uma *Applet* consiste de quatro eventos:

```
public class Simple extends Applet {
    . . .
    public void init() { . . . }
    public void start() { . . . }
    public void stop() { . . . }
    public void destroy() { . . . }
    . . .
    }
```

O método *init* é chamado para inicializar a *Applet* toda vez que ela é carregada (ou recarregada). O método *start* começa a execução de uma *Applet*, quando ela é carregada ou quando o usuário volta para a página que contém a *Applet*. O método *stop* pára a execução da *Applet*, quando o usuário sai da página ou do *browser*. E o método *destroy* é chamado antes da descarga.

Em 1997 foram criadas as JFC (*Java Foundation Classes*), para facilitar a criação de GUIs (*Graphical User Interfaces*). Elas contém as seguintes características (JFC 1.1):

- Componentes *Swing* (como *Frames*, *Dialogs*, etc)
- Suporte a vários *look-and-feel* (aparência)
- API de acessibilidade (Braille)
- API de Java 2D (a partir do JDK 1.2)
- Suporte a *Drag-and-Drop* (a partir do JDK 1.2)

Os *browsers* mais antigos suportam apenas as Applets geradas com o JDK 1.0.2. Para verificar que versão de JDK o seu *browser* suporta clique em *Ajuda - Sobre Plug-ins*.

6. Futuro da linguagem Java

A idéia principal por trás da plataforma (linguagem+ambiente) Java é a *portabilidade*. A integração entre dispositivos diferentes, como computadores e torradeiras, saiu do campo da ficção científica diretamente para os laboratórios.

Está em desenvolvimento um processador Java (o JavaChip), que permite executar os *bytecodes* diretamente. Um chip como esse pode equipar diversos aparelhos domésticos. Já existem protótipos de sistemas de navegação para veículos, sistemas embutidos, NC's, *smart cards*, vídeo-fones, etc.

Nos negócios, a possibilidade de integrar facilmente sistemas com arquiteturas diferentes à Internet é uma grande vantagem.

Fontes de consulta na Web

http://java.sun.com (site oficial)
http://www.gamelan.com/ (código e tutoriais)
http://www.javaworld.com/ (notícias)
http://www.javareport.com/ (revista)
http://www.sys-con.com/ (Java Developers Journal)

http://www.javabr.com.br/ (Clube Java Brasil)
http://www.gojug.com.br/ (Java User Group-GO)
http://www.dfjug.org/ (JUG DF)
http://www.soujava.org.br (JUG Sucesu-SP)
http://www.gujava.cjb.net/ (JUG Sucesu-MG)
http://www.javaman.com.br (página pessoal)
http://www.cade.com.br/intjava.htm (seção do Cadê)