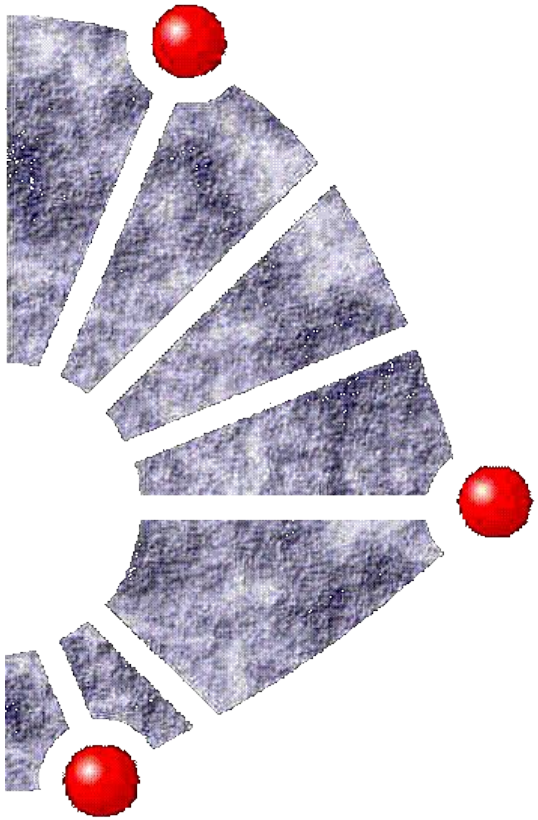
# Java Básico



**AULA 1**

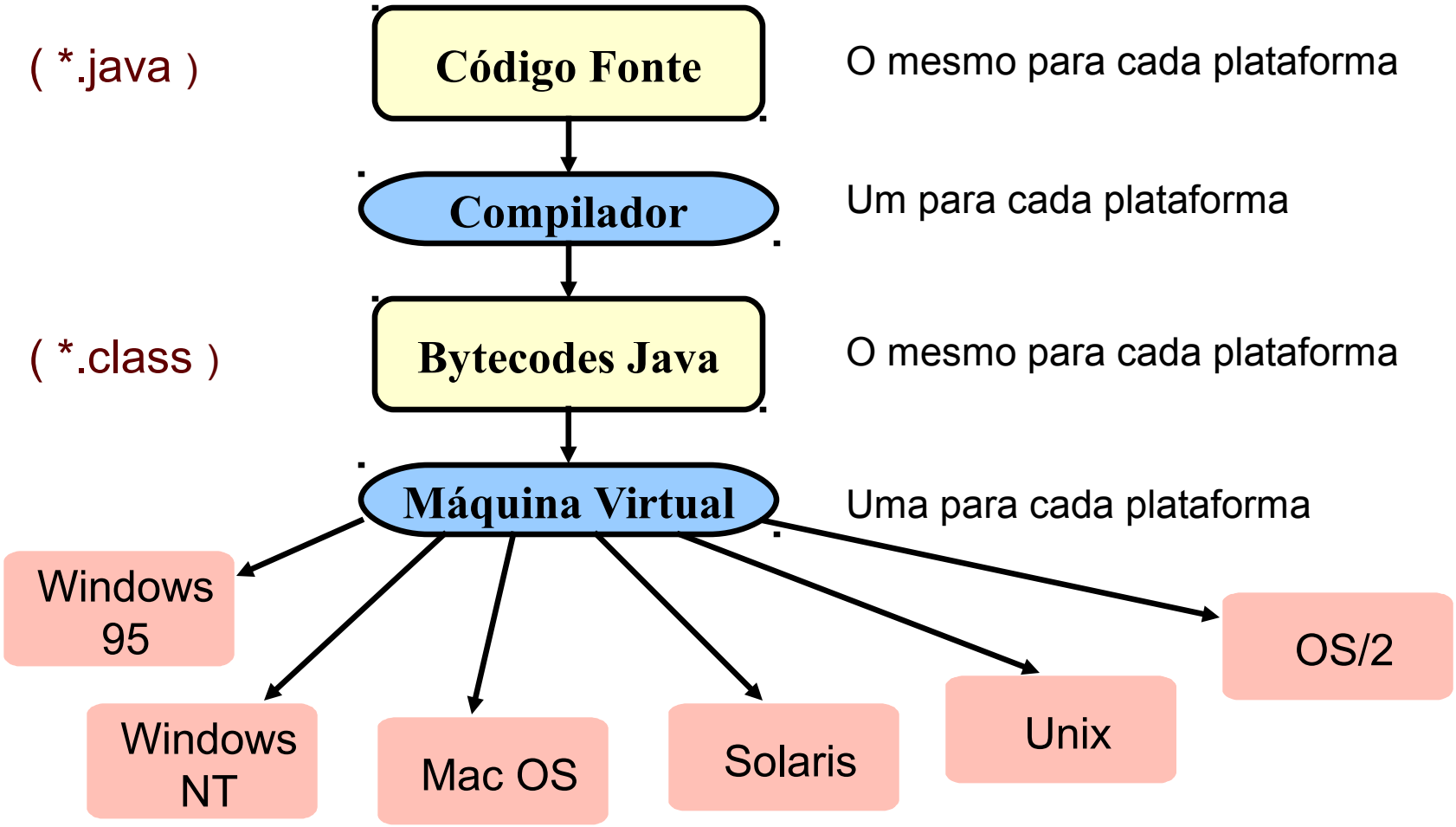
# O que é Java ?

- Java é uma linguagem de programação
  - simples, segura e orientada a objetos
  - interpretada e dinamicamente otimizada por meio de bytecodes
  - neutra em relação a arquitetura e sistema operacional
  - utiliza mecanismos de garbage-collection e multi-threads
  - fortemente tipada e com mecanismos de gerenciamento de excessões *built-in*
  - utilizada para a construção de sistemas distribuídos e dinamicamente extensíveis
  - resultado de quase 15 anos de tentativas em se desenvolver uma linguagem de programação mais eficiente e simples para a construção de softwares mais robustos e confiáveis

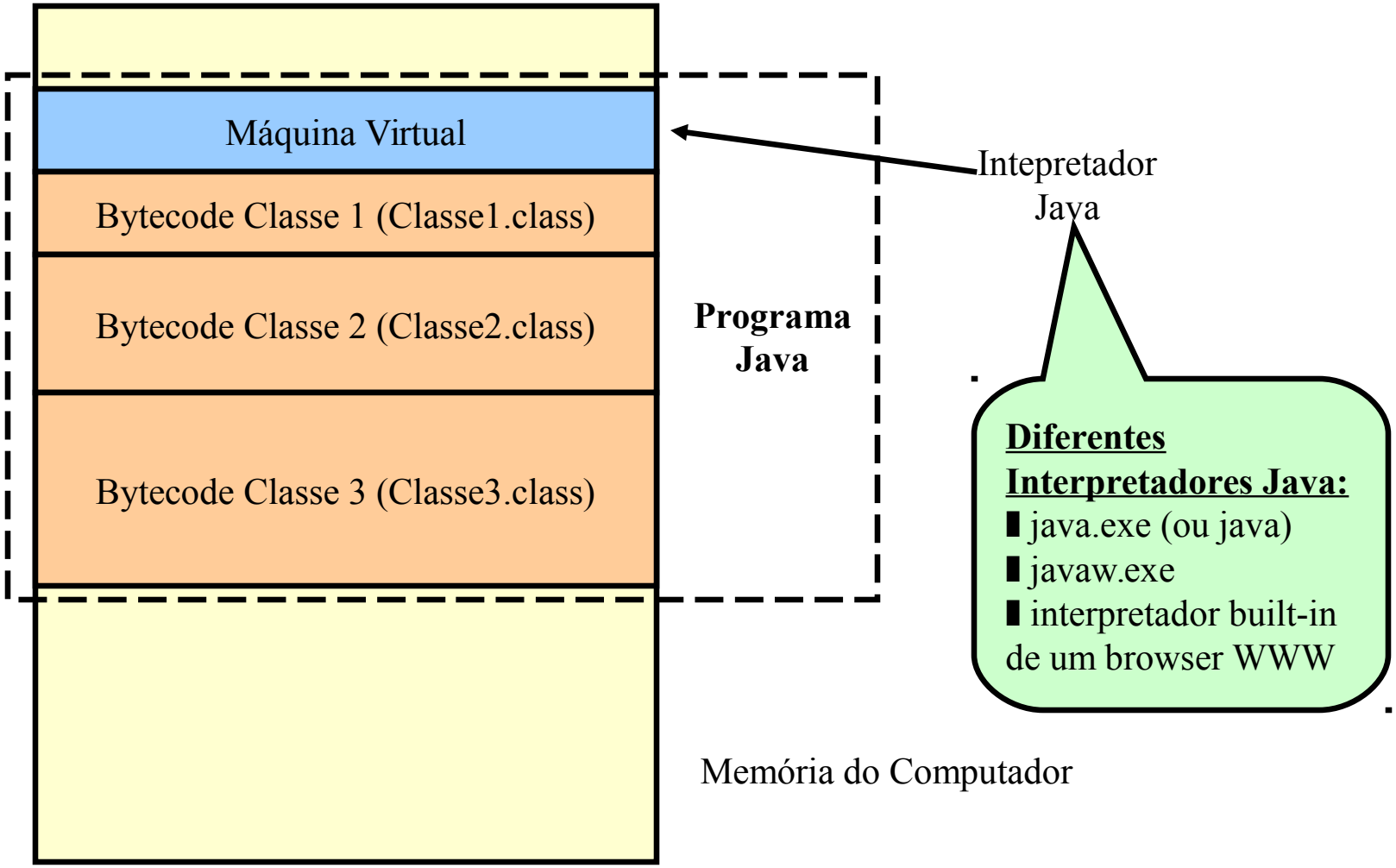
# O que é Java ?

- Java é melhor pelo que não tem do que pelo que tem
  - variáveis globais
  - goto's
  - ponteiros (ou apontadores)
  - alocação de memória
  - tipos de dados dependentes de plataforma
  - conversão de tipos de dados
  - arquivos de "header"
  - pré-processamento
- Java é uma linguagem para a Web
  - desenvolvimento direcionado para a criação de programas que pudessem ser carregados de servidores Web e executados com segurança em diferentes plataformas

# A Plataforma Java: Bytecodes e Máquinas Virtuais

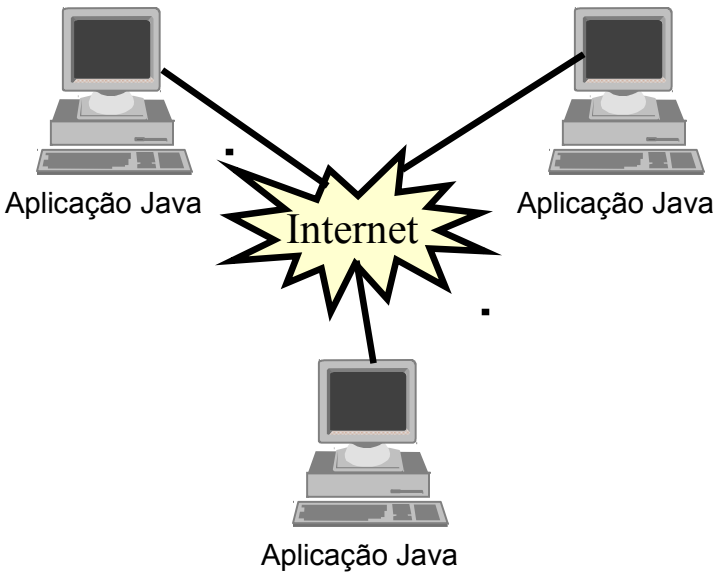


# A Plataforma Java: Bytecodes e Máquinas Virtuais

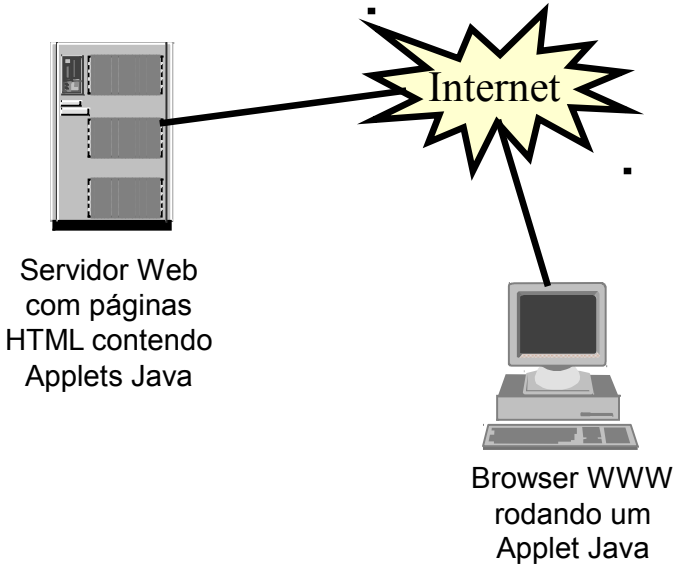


# Aplicações e Applets

- Aplicações
  - programas locais ou distribuídos rodando em um servidor local




- Applets
  - programas carregados via WWW por meio de um browser Web e executados localmente



# Executando Applets

- Inclusão de applets ao conteúdo de documentos HTML:

```
<APPLET code="myapplet.class"
  codebase="http://www.unicamp.br/dir/"
  width=400 height=300> ...
</APPLET>
```

- Navegadores restrigem ações que applets podem executar
  - código vindo pela rede pode não ser confiável
- Restrições têm por objetivo:
  - evitar obtenção de informação privilegiada
  - evitar execução de ações destrutivas
  - forçar a utilização de mecanismos de segurança
- Modelo de execução em *sandbox*

# Restrições de Segurança em Applets

- Sistema de Arquivos Local
  - leitura ou escrita em arquivos não é permitida
- Operações em Rede
  - restritas à conexão com servidor do applet
- Facilidades do Sistema
  - criação ou término de processos fica indisponível
- Janelas Seguras
  - funções de desenho restritas à área da janela
- Applets confiáveis
  - mecanismo de assinatura digital permite que applets originários de fontes confiáveis não estejam restritos ao modelo sandbox
    - usuário determina em quem confia

# API's e Frameworks do Java

- Alguns Frameworks do Java
  - Java Applet Framework - criação de Applets Java
  - Java Commerce Framework - transações monetárias seguras
  - Java Enterprise Framework - acesso a bancos de dados e objetos em ambientes distribuídos (incluindo CORBA)
  - Java Server Framework - desenvolvimento de servidores
  - Java Media Framework - gráficos, vídeo, áudio e animação
  - Java Security Framework - autenticação, assinaturas digitais e encriptação
  - Java Beans Framework - Modelo de componentes nativo Java
- Objetos encontram-se distribuidos em "packages"
  - importados para o código fonte utilizando a instrução "import"

# Pacotes da Plataforma Java Padrão

| | |
|---|---|
| java.applet | Provê as classes necessárias para a criação e gerenciamento de applets |
| java.awt | Contém classes para a criação de interfaces com o usuário e manipulação de gráficos e imagens |
| java.beans | Contém classes relacionadas com o desenvolvimento de Java Beans |
| java.io | Provê objetos para a manipulação de entrada e saída de dados em Streams, via serialização e por meio do sistema de arquivos |
| java.lang | Provê uma série de classes que são fundamentais para o design da linguagem Java |
| java.math | Provê classes que realizam aritmética inteira de precisão arbitrária (BigInteger) e aritmética decimal de precisão arbitrária (BigDecimal), bem como funções matemáticas diversas |
| java.net | Provê classes para a implementação de aplicações de rede |
| java.rmi | Provê classes para a manipulação de Invocações Remotas de Métodos (RMI) |
| java.security | Provê as classes e interfaces para o framework de segurança do Java |

# Pacotes da Plataforma Java Padrão

| | |
|---|---|
| java.sql | Provê classes para a manipulaçao de bancos de dados via JDBC |
| java.text | Provê classes e interfaces para a manipulação de textos, datas, números e mensagens |
| java.util | Contém o framework "Collections", classes de coleções legadas, um modelo de eventos, facilidades de manipulação de datas e tempo, mecanismos de internacionalização e um conjunto de classes utilitárias de propósito geral (string tokenizer, gerador de números aleatórios, arrays de bits, etc…) |
| javax.accessibility | Provê objetos para a integração entre componentes de interface com o usuário e tecnologias assistivas para o acesso a estes componentes. |
| javax.swing | Provê um conjunto de componentes "lightweight" (totalmente em Java), que dentro da maior conveniência possível devem funcionar de maneira semelhante em todas as plataformas. |
| org.omg.CORBA | Provê o mapeamento da tecnologia OMG/.CORBA em uma API do Java, incluindo um ORB (Object Request Broker) totalmente funcional |

# Exemplo de uma Classe Java

Indica que o método não manipula instâncias da classe, podendo assim ser invocado sem se referir a uma instância em particular

Nome da classe

Indica nenhum tipo de dado. O método não retorna nenhum valor.

Método *main* indica que a classe pode ser utilizada para iniciar uma aplicação A execução da aplicação começará pela invocação desse método

Indica que o método poderá ser chamado de qualquer lugar, dentro ou fora da classe.

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

Encapsula a funcionalidade do sistema.

Variável estática, membro da classe *System*

Método do objeto *out*. Equivalente à função printf () em C, sucedido por um "carriage return" (\n) no final.

Array de objetos tipo *String* passados como parâmetros pelo sistema operacional em tempo de execução

# Exemplo de uma Classe Java

Mecanismo de Herança

Implementação de Interfaces

Definição de Atributos

```
class MyClass extends OtherClass implements MyIface {
  int x;
  float y;
  char z;
  public int method1 (int a, int b) {
    int c;
    c = a + b;
    return(c);
  }
  public void method2 () {
    System.out.println("Fim");
  }
}
```

Definição de Métodos

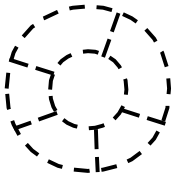
Lista de Parâmetros do Método

Atributo Local ao Método

# Classes e Objetos

## Classe

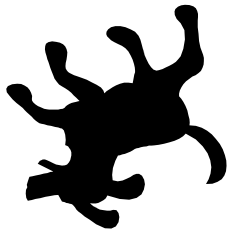
generalização
de características
(atributos e
comportamentos)



**CACHORRO**

raça,
cor

comer,
latir



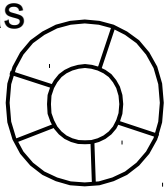
**CLASSE JAVA**

```
class Cachorro {
   String raça;
   Integer cor;
   public void comer(Comida c){

   ...
   }
   public void latir(Modo x) {
   ...
   }
}
```

## Objeto

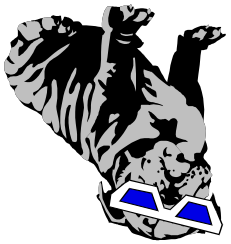
Instância particular
de uma classe.
Distinguível por suas
características
específicas.



**RAMBO**

bulldog
cinza

comer caviar
latir forte



**OBJETO JAVA**

```
...
Cachorro Rambo;
Rambo.comer(Caviar);
Rambo.latir(Forte);
...
```

# Elementos Básicos do Java

- O Objeto Principal
  - deve implementar o método "main"
    - `public static void main(String[] args)`
  - chamado a partir do sistema operacional
- Excessões
  - mecanismo utilizando os operadores `try-catch/throws`
- Objetos Especiais
  - super e this
- Classes Internas (Inner Classes)
  - utilizadas para a definição de classes de escopo limitado
  - utilizadas para a derivação de *listeners* e *adapters* especializados
- Interfaces, Classes e Classes Abstratas

# Elementos Básicos do Java

- Modificadores de Classes
  - public, abstract, final
- Modificadores de Atributos
  - static, final, transient, volatile
- Modificadores de Métodos
  - public, protected, private, abstract, static, final, synchronized, native
- Keywords Especiais
  - throw, synchronized
- Listeners e Adapters
  - Listener - interface (precisa ser implementada)
  - Adapter - classe padrão implementando um listener (precisa ser estendida)

# Identificadores e Keywords

- Identificadores
  - Nomes para:
    - Variáveis, Métodos, Classes, Interfaces e Pacotes
  - Maiúsculas e minúsculas
    - 1o caracter: letra, "_", "$"
    - seguintes: anteriores + digitos (0 a 9)
- Keywords

| | | | | |
|---|---|---|---|---|
| abstract | continue | for | new | switch |
| boolean | default | goto | null | synchronized |
| break | do | if | package | this |
| byte | double | implements | private | threadsafe |
| byvalue | else | import | protected | throw |
| case | extends | instanceof | public | transient |
| catch | false | int | return | true |
| char | final | interface | short | try |
| class | finally | long | static | void |
| const | float | native | super | while |

# Literais

- Literais
  - Literais Inteiros, Literais de Ponto Flutuante, Literais Booleanos, Literais de Cadeias de Caracteres, Literais de Um Caracter
- Literais de Um Caracter

| | |
|---|---|
| Barra invertida (backslash) | \\ |
| Continuação de Linha | \ |
| Retrocesso (backspace) | \b |
| Carriage Return (CR) | \r |
| Alimentação de página (FF) | \f |
| Tabulador (tab) | \t |
| Nova línha (CR+LF) | \n |
| Aspas simples | \' |
| Aspas duplas | \" |
| Caracter Unicode | \udddd |
| Caracter Octal | \uddd |

# Operadores

**Aritméticos e de conversão**

| | |
|---|---|
| i,f | + |
| i,f | - |
| i,f | * |
| i,f | / |
| i,f | % |
| i,f | ++ |
| i,f | -- |
| i | - |
| i | ~ |

| | |
|---|---|
| i | & |
| i | \| |
| i | ^ |
| i | << |
| i | >> |
| i | >>> |

**De comparação**

| | |
|---|---|
| i,f | == |
| i,f | != |
| i,f | < |
| i,f | <= |
| i,f | > |
| i,f | >= |

**Lógicos**

| | |
|---|---|
| b | & |
| b | \| |
| b | ^ |
| b | && |
| b | \| |
| b | ! |
| b | == |
| b | != |
| b | ?: |

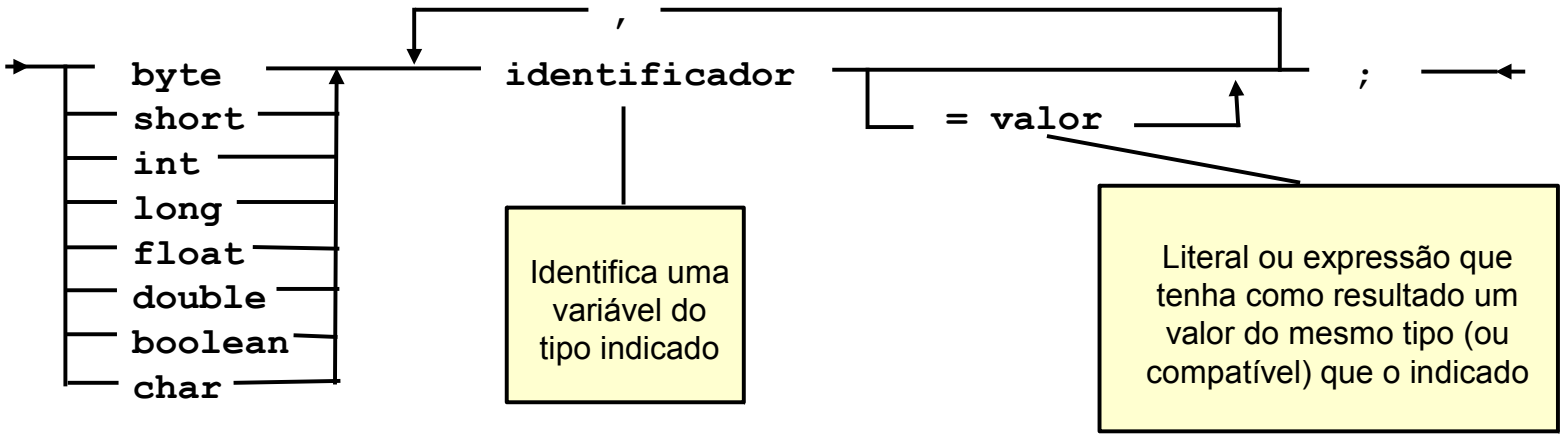
**De atribuição**

| |
|---|
| = |
| += |
| -= |
| *= |
| /= |
| %= |
| &= |
| \|= |
| ^= |

Tipos válidos de operandos
i = Inteiro (byte, small, int, long)
f = Ponto Flutuante (float, double)
b = Lógico (boolean)

# Definições Básicas da Linguagem

- Separadores e Comentários
  - { } ; , :
  - /* comentários no código do programa */
  - // comentários até o final da linha
  - /** comentários que utilizam o mecanismo de documentação javadoc */
- Declaração de variáveis com tipos de dados primitivos



# Estruturas de Controle

sentença1 ; Sentença

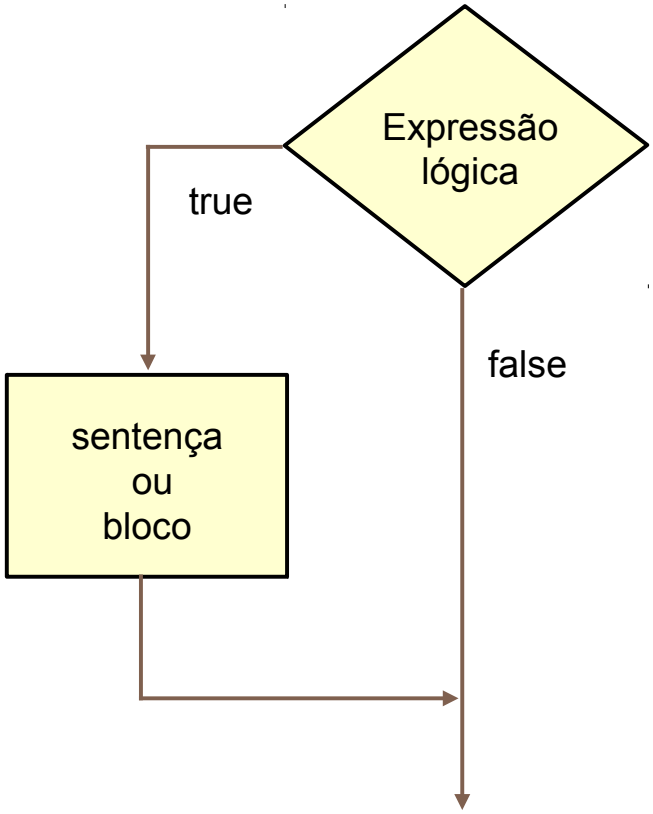
```
{
sentença2 ;
sentença3;
...
}
```

Bloco

```
{
  {
   sentença4;
   sentença5:
  }
  sentença6;
  ...
}
```

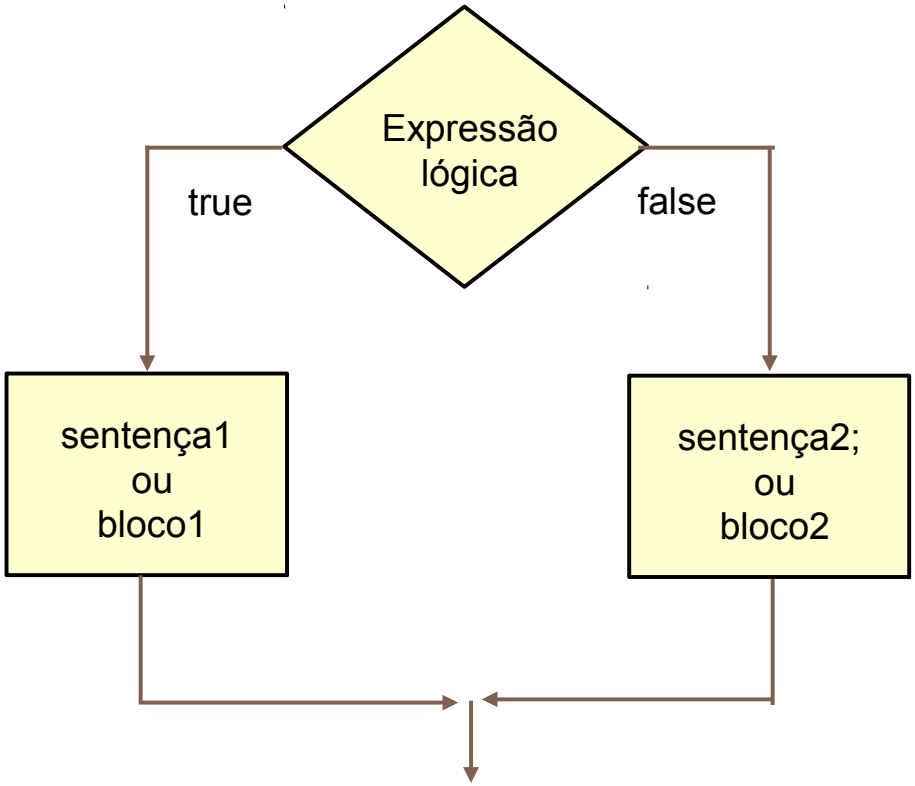
Blocos Aninhados

```
if (ExpLogica)
   sentença; (ou bloco)
```

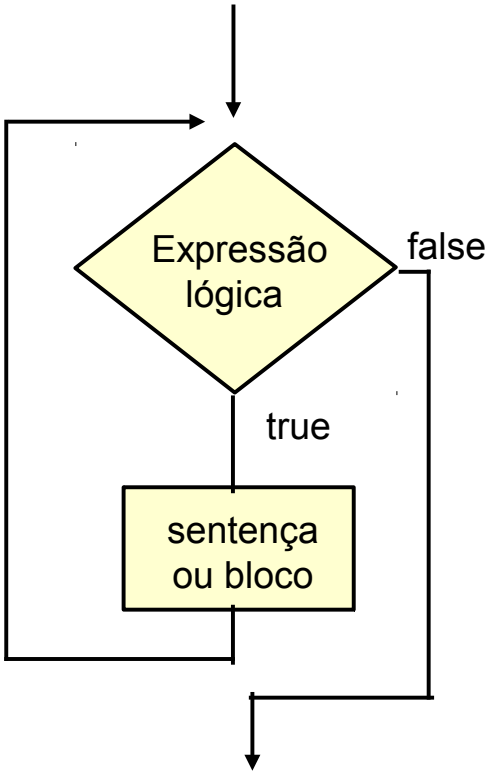


# Estruturas de Controle

```
if (ExpLogica)
  sentença1; (ou bloco1)
else
  sentença2; (ou bloco2)
```

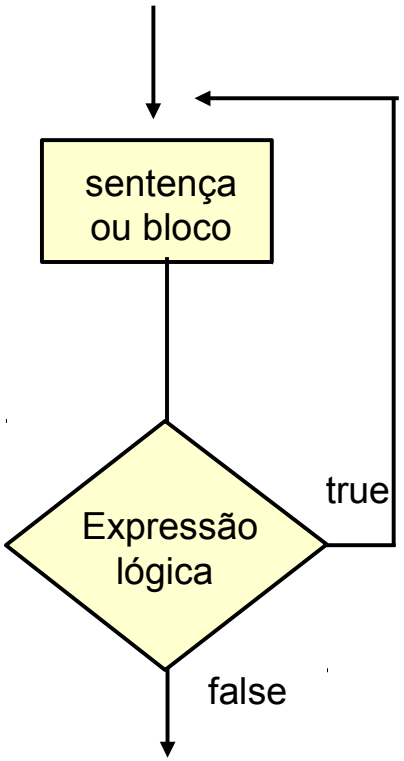


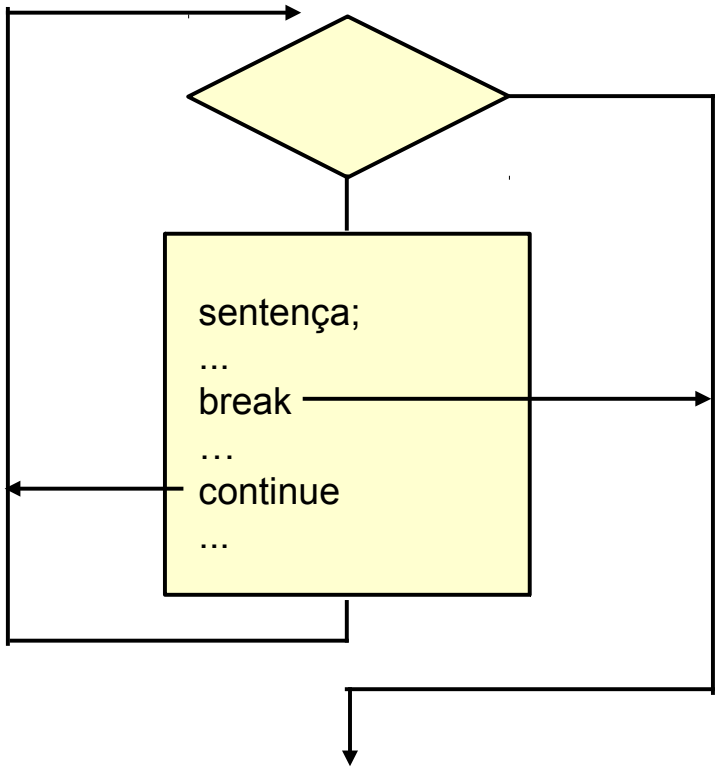

```
while (ExpLogica)
  sentença; (ou bloco)
```



# Estruturas de Controle

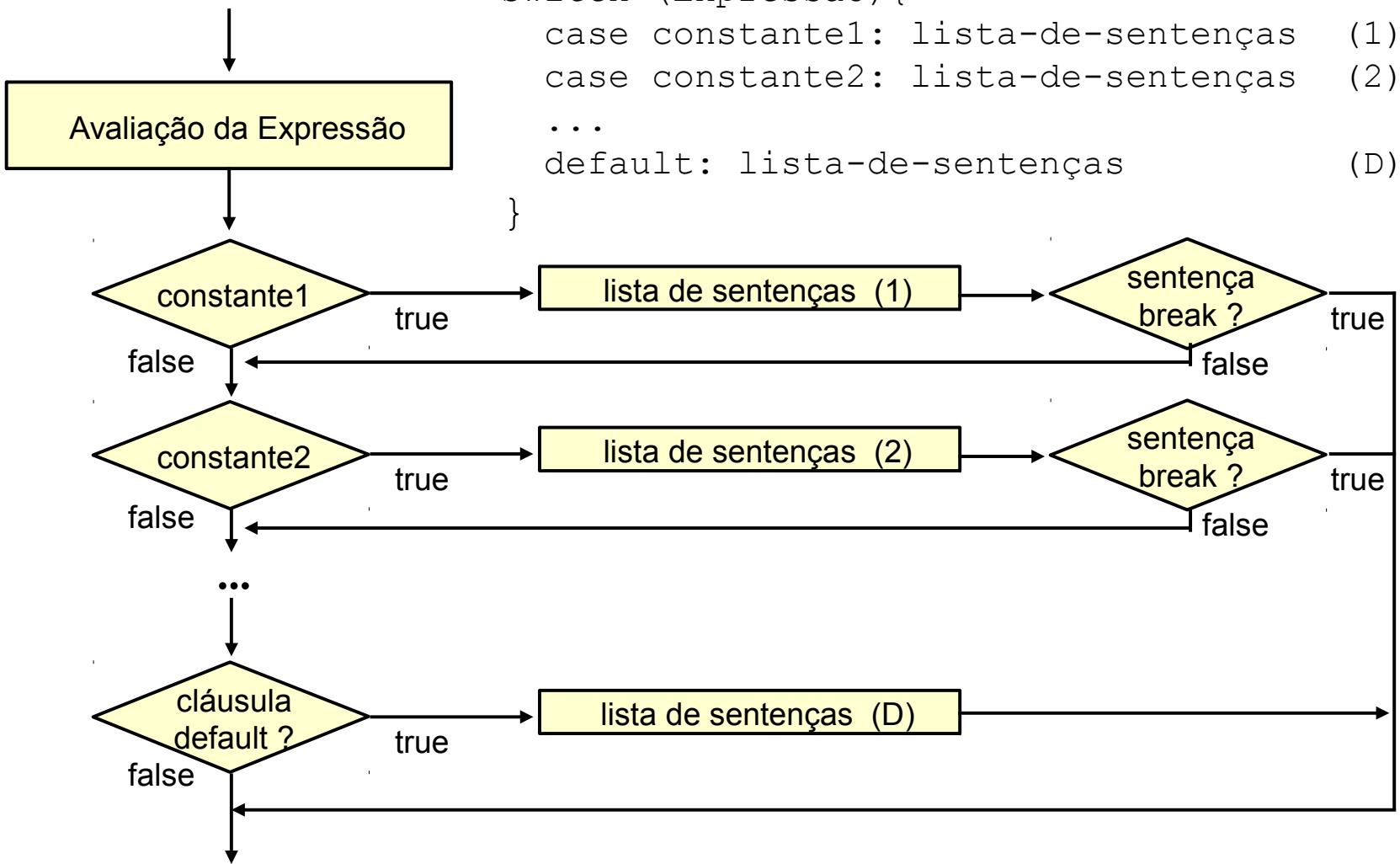
```
do
   sentença; (ou bloco)
while (ExpLogica);
```



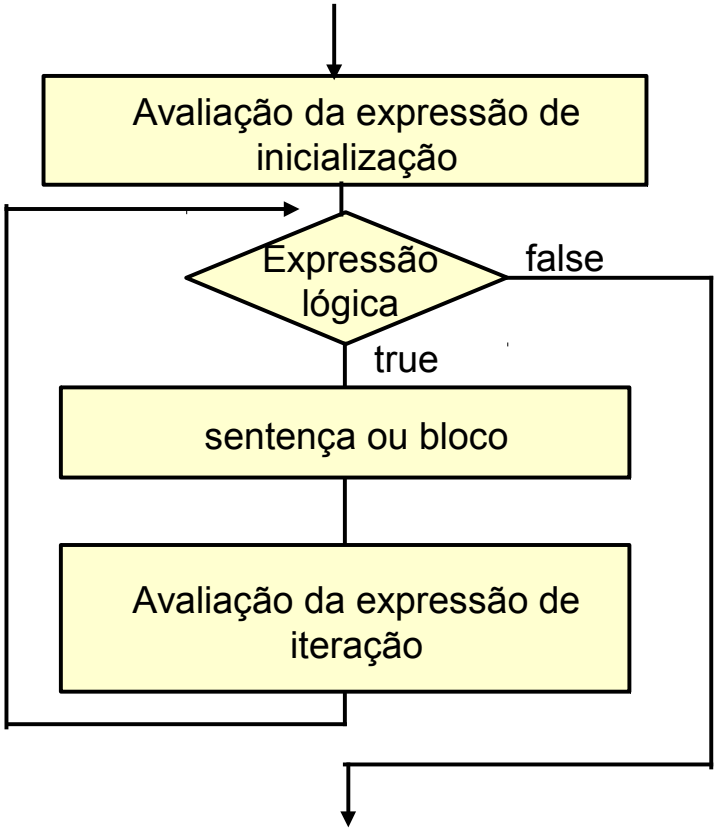
# Estruturas de Controle

```
switch (Expressão){
  case constante1: lista-de-sentenças   (1)
  case constante2: lista-de-sentenças   (2)
  ...
  default: lista-de-sentenças           (D)
}
```



# Estruturas de Controle

```
for (ExpInic; ExpLogica; ExpIter) sentença; (ou bloco)
```



# Algumas Classes Básicas

- Classe Object
  - classe raiz de todas as outras classes
  - possui métodos que são herdados por todos os outros objetos
    - equals, hashCode, clone, getClass, toString, finalize, wait, notify, notifyAll
- Classe Class
  - permite obtermos informações sobre uma determinada classe em particular
  - base para o mecanismo de reflexão do Java
- Classe ClassLoader
  - é uma classe abstrata utilizada para a definição de carregadores de classes
  - no Java, pode haver múltiplas maneiras de se carregar uma classe (e.g. URLClassLoader)

# Algumas Classes Básicas

- Classe SecurityManager
  - classe abstrata que trabalha em conjunto com ClassLoaders para implementar políticas de segurança
  - contém diversos métodos que podem sofrer override para a implementação de mecanismos customizados de segurança
- Classe Runtime
  - provê acesso ao sistema de execução do Java
  - diversas informações podem ser obtidas (memória total, memória livre, etc…), bem como alguns comandos referentes à execução podem ser solicitados (disparo do garbage collector, execução de programas externos, carregamento de bibliotecas dinâmicas, etc…
- Classe System
  - provê uma interface aos recursos do sistema operacional

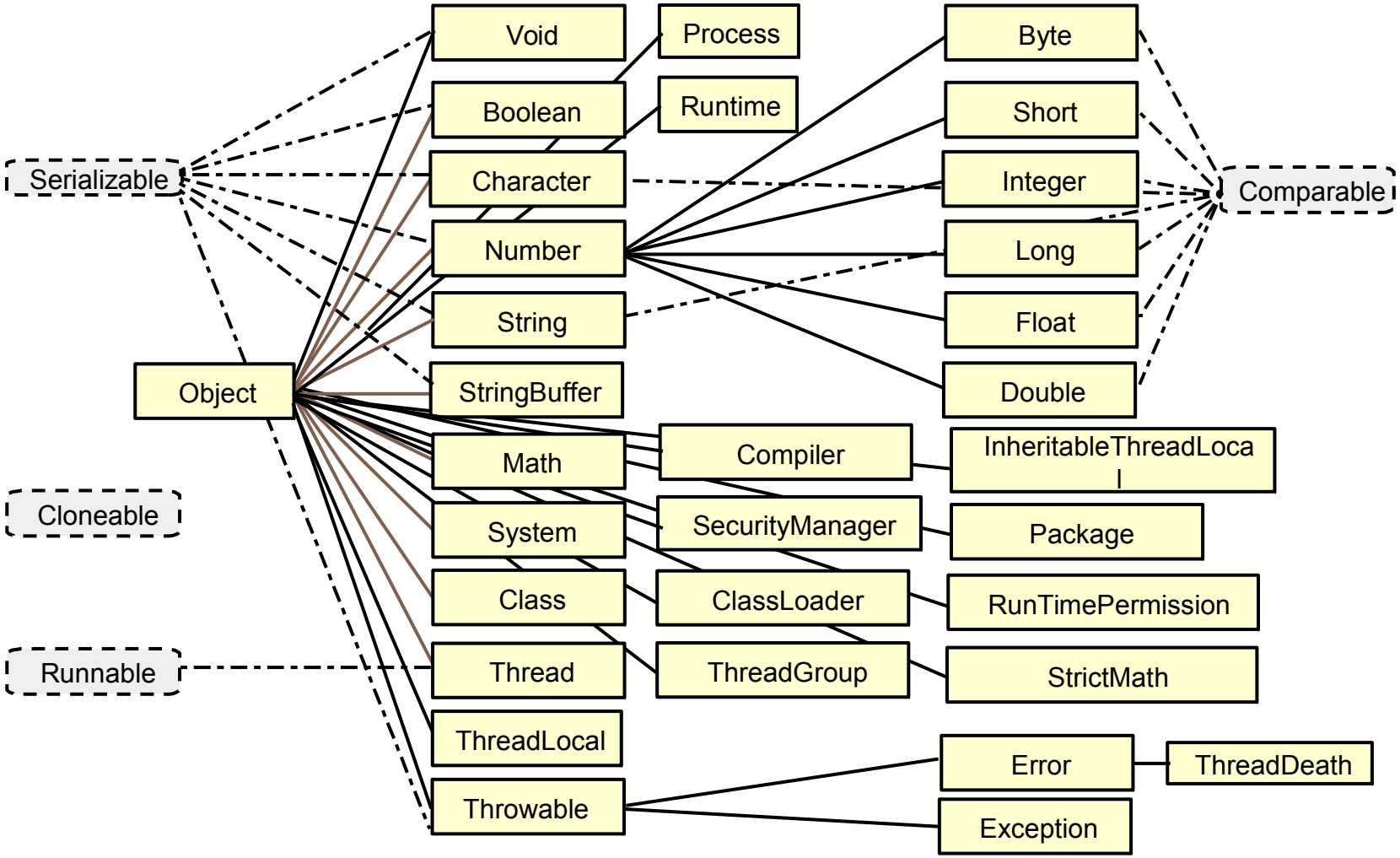
# Algumas Classes Básicas

- Wrapped Classes
  - Tipos primitivos de dados não são tratados pelo Java como sendo objetos
  - Para criar classes Java equivalentes a tipos de dados primitivos, usamos as Wrapped Classes
  - Boolean, Character, Byte, Short, Integer, Long, Double, Float, Number
- Class Math
  - classe estática que provê um conjunto extenso de métodos e constantes matemáticas
  - suporta métodos aritméticos, trigonométricos, exponenciais, logarítmicos, números aleatórios e rotinas de conversão
- Classe StrictMath
  - Variação da classe Math seguindo padrões numéricos rígidos

# Algumas Classes Básicas

- Classe String
  - permite como valores cadeias de caracteres
  - String str = “text”;
  - String str = new String(“text”);
  - objetos String são sempre constantes
- Classe StringBuffer
  - para a manipulação de Strings, como Strings são constantes, é necessário uma classe que permita essa manipulação
  - Essa classe é a classe StringBuffer
  - A classe StringBuffer opera em conjunto com o operador + para a manipulação de Strings
    - `String s = “”;`
    - `s = s + “a” + “b”;`
    - `s = new StringBuffer(“”).append(“a”).append(“b”).toString();`

# O Pacote java.lang



# Streams

- Streams
  - são sequências de bytes que viajam de uma origem a um destino por um caminho de comunicação
  - se um programa está escrevendo em um Stream, ele é uma origem
  - se ele está lendo de um Stream, ele é um destino
  - Streams podem ocorrer em transferências de memória, sistemas de arquivos, redes e outras formas de E/S
  - são o mecanismo básico para uma série de funções em Java
- Streams Básicos
  - InputStream e OutputStream
    - são subclasseados em uma rica gama de tipos de Streams, descritos no package java.io

# Ferramentas Úteis em java.util

- Classes Relacionads com Datas
  - Date, Calendar, GregorianCalendar, TimeZone, SimpleTimeZone
- Classe Random
  - permite a manipulação de números aleatórios
- Interface Enumeration
  - permite a manipulação de conjuntos indexados de valores
- Classe Vector
  - permite a manipulação de arrays que podem crescer dinamicamente
- Classe StringTokenizer
  - utilizada para a criação de parsers para objetos do tipo String
  - faz o parsing do String de acordo com um conjunto de delimitadores

# Utilizando Arrays

- Criando Arrays
  - `int[] arrayOfInts = new int[10];`
  - `int[][] arrayOfInts = new int[10][20];`
- Arrays de Tipos Primitivos e Arrays de Objetos
  - **Arrays de Tipos Primitivos:**
    - armazenam memória para os objetos
  - **Arrays de Objetos:**
    - armazenam memória para as referências aos objetos, mas os objetos precisam ser criados explicitamente
- Exemplo de Array de Objetos
  - `Integer[] arrayOfIntegers = new Integer[2];`
  - `arrayOfIntegers[0] = new Integer();`
  - `arrayOfIntegers[1] = new Integer();`

# Utilizando Coleções

- Coleção (Collection)
  - também chamada de *container*, representa um grupo de objetos, conhecidos como elementos
  - algumas coleções permitem elementos duplicados, enquanto outras não.
  - Algumas são ordenadas e outras não-ordenadas
  - Algumas podem permitir elementos duplicados (Bags ou multisets)
- Tipos de Coleções
  - Set, List, Map, SortedSet, SortedMap, HashSet, TreeSet, ArrayList, LinkedList, Vector, Collections, Arrays, AbstractCollection
- A Interface Collection
  - interface mãe na hierarquia de coleções do Java

# Utilizando Coleções

- Classe Collections
  - classe estática que provê métodos para operação ou geração de coleções
- Interface List
  - Listas são coleções ordenadas (sequências)
- Classes ArrayList e LinkedList
  - implementações da interface List
  - permitem manipulações de arrays de listas e listas ligadas
  - Classe ArrayList é muito semelhante à classe Vector
    - alguns métodos tiveram seus nomes encurtados para facilitar a operação
    - métodos em ArrayList são não-sincronizados
      - cuidado caso haja múltiplos thread acessando o mesmo ArrayList

# Usando Applets

- Quando se cria um Applet, ao invés de uma Aplicação
  - ao invés de se declarar um método estático **main**
  - deve-se fazer o objeto principal do programa estender a classe Applet (ou JApplet), fazendo o override dos métodos init, start, stop e destroy
- Classe Applet
  - provê uma interface standard entre o Applet e seu ambiente
- Classe JApplet
  - versão estendida da classe Applet para aplicações que utilizem o Swing
- O Próprio browser executando o Applet
  - se encarrega de efetuar a chamada dos métodos init, start, stop e destroy, quando necessário

# Usando Applets

- Método init()
  - chamado pelo browser para informar ao Applet que ele foi carregado no sistema
- Método start()
  - chamado pelo browser para informar ao Applet que ele deve começar sua execução
- Método stop()
  - chamado pelo browser para informar ao Applet que ele deve parar sua execução
- Método destroy()
  - chamado pelo browser para informar ao Applet que sua destruição foi solicitada e portanto ele deve destruir e/ou liberar todos os recursos que tenha alocado