Capítulo 03:
Fundamentos Java

Leonardo Moura Leitão
Marcelo Gonçalves Pinheiro Quirino

Agenda

- Introdução
  - História e Filosofia do Java
  - Mágica Java: O *Bytecode*
  - Chavões Java
  - Obtendo o JDK
  - Ambiente Java
  - Primeiro Programa

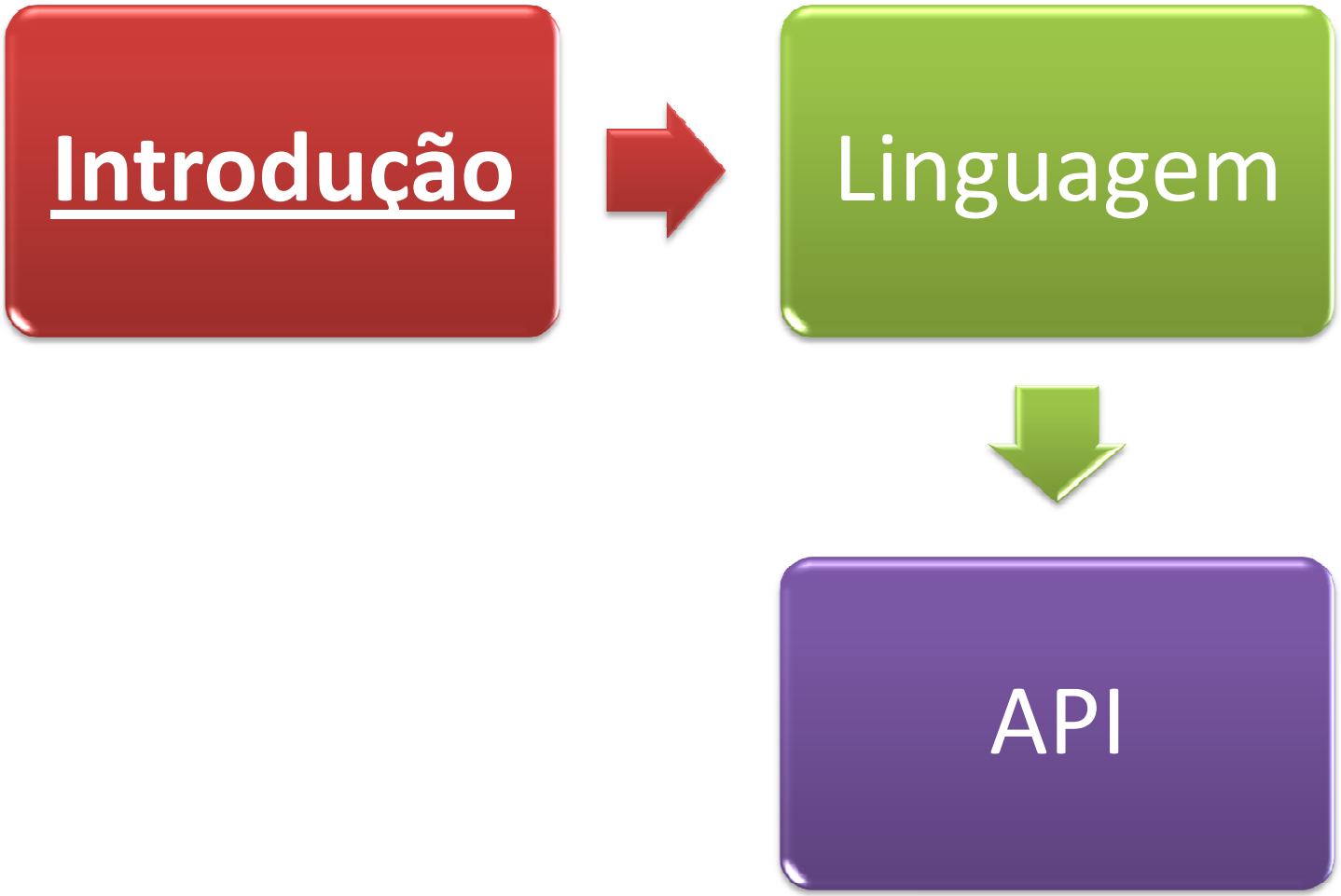
Agenda

- Linguagem Java
  - Comentários
  - Blocos e Sentenças
  - Identificadores
  - Constantes e Variáveis
  - Palavras reservadas no Java
  - Tipos Primitivos
  - Conversões (Tipos Numéricos)
  - Operadores

Agenda

- API Java
  - Arquitetura Java SE
  - Linguagem Java
  - Ferramentas
  - Tecnologias para deployment
  - Java SE API: Interface c/ Usuário
  - Java SE API: Bibliotecas de Integração
  - Java SE API: Bibliotecas Base
  - Máquina Virtual Java

Introdução Java



História e Filosofia do Java

- Java foi criado por **James Gosling**, Patrick Naughton, Chris Warth, Ed Frank, e Mike Sheridan na Sun Microsystems em 1991.

- Originalmente foi chamada de Oak, mas foi renomeada para Java em 1995.

História e Filosofia do Java

- A motivação principal do projeto foi criar uma linguagem **independente de plataforma** que pudesse ser usada em dispositivos eletrônicos como geladeiras, microondas e etc.

- Mas o futuro do Java foi altamente impactado pela Web, que também necessitava de programas portáveis.

História e Filosofia do Java

- O foco do Java passou a ser a **programação voltada para internet** e não mais para dispositivos eletrônicos.

- A internet levou o Java ao sucesso em larga escala.

História e Filosofia do Java

- Java herdou a **sintaxe do C** e **adaptou o modelo de objetos do C++**.

- Java influenciou na criação da linguagem C# da Microsoft. Na verdade, muitos dos recursos dessa linguagem foram diretamente adaptados do Java.

Mágica Java: O *Bytecode*

- O *Bytecode* é um **conjunto de instruções** altamente otimizadas, projetadas para serem **executadas** pelo sistema de *run-time* Java, que é chamado de ***Java Virtual Machine*** (JVM).

- A Maquina Virtual Java é um **interpretador** de *bytecode*.

Mágica Java: O *Bytecode*

- Traduzir um programa em *bytecodes* torna muito mais fácil executá-lo em uma grande variedade de ambientes.

- A razão é simples: somente a **JVM** precisa ser implementada para cada **plataforma**! Uma vez que esse ambiente de execução existe, qualquer programa Java pode ser executado.

Mágica Java: O *Bytecode*

- Os detalhes da JVM podem diferir de uma plataforma para outra, mas **todos entendem o mesmo *bytecode***.

- O programa Java, por ser interpretado, é **mais seguro**, visto que a execução está **sob o controle** da JVM.

Mágica Java: O *Bytecode*

- Programas interpretados, em geral, executam substancialmente de maneira mais lenta que programas compilados...

- ...Entretanto, com Java, essa diferença não é grande por causa da tecnologia ***HotSpot***.

Mágica Java: O *Bytecode*

- HotSpot, que faz parte da JVM, provê um compilador JIT (*Just-In-Time*) para *bytecodes*.

- Esse compilador **gera um código nativo em tempo de execução** e, por gerar várias verificações e otimizações somente possíveis em tempo de execução, ele **não compila a aplicação toda** de uma vez.

Chavões Java

Chavão	Significado
Simples	Java possui um conjunto de recursos conciso e coeso que torna fácil de aprender e utilizar.
Segura	Java prover uma ambiente seguro para criação de aplicações para Internet.
Portável	Java pode executar em qualquer ambiente para o qual exita JVM (ambiente de execução).
Orientada a objeto	Java incorpora um filosofia de programação orientada a objetos moderna.
Robusta	Java incentiva uma programação livre de erros por ser uma linguagem fortemente tipada e por executar várias verificações em *run-time*.

Chavões Java

Chavão	Significado
Multi-Thread	Java possui suporte integral à programação multi-thread.
Arquitetura neutra	Java não é amarrada a nenhuma máquina ou sistema operacional.
Interpretada	Java suporta código multi-plataforma através do uso do bytecode Java.
Alto desempenho	O bytecode Java é altamente otimizado para executar de forma mais rápida.
Distribuída	Java foi projetada para sistema distribuídos, tendo com foco a internet.
Dinâmica	Um programa Java possui uma quantidade considerável de informações de tipos em tempo de execução que é usada para resolver acessos para objetos.

Obtendo o JDK

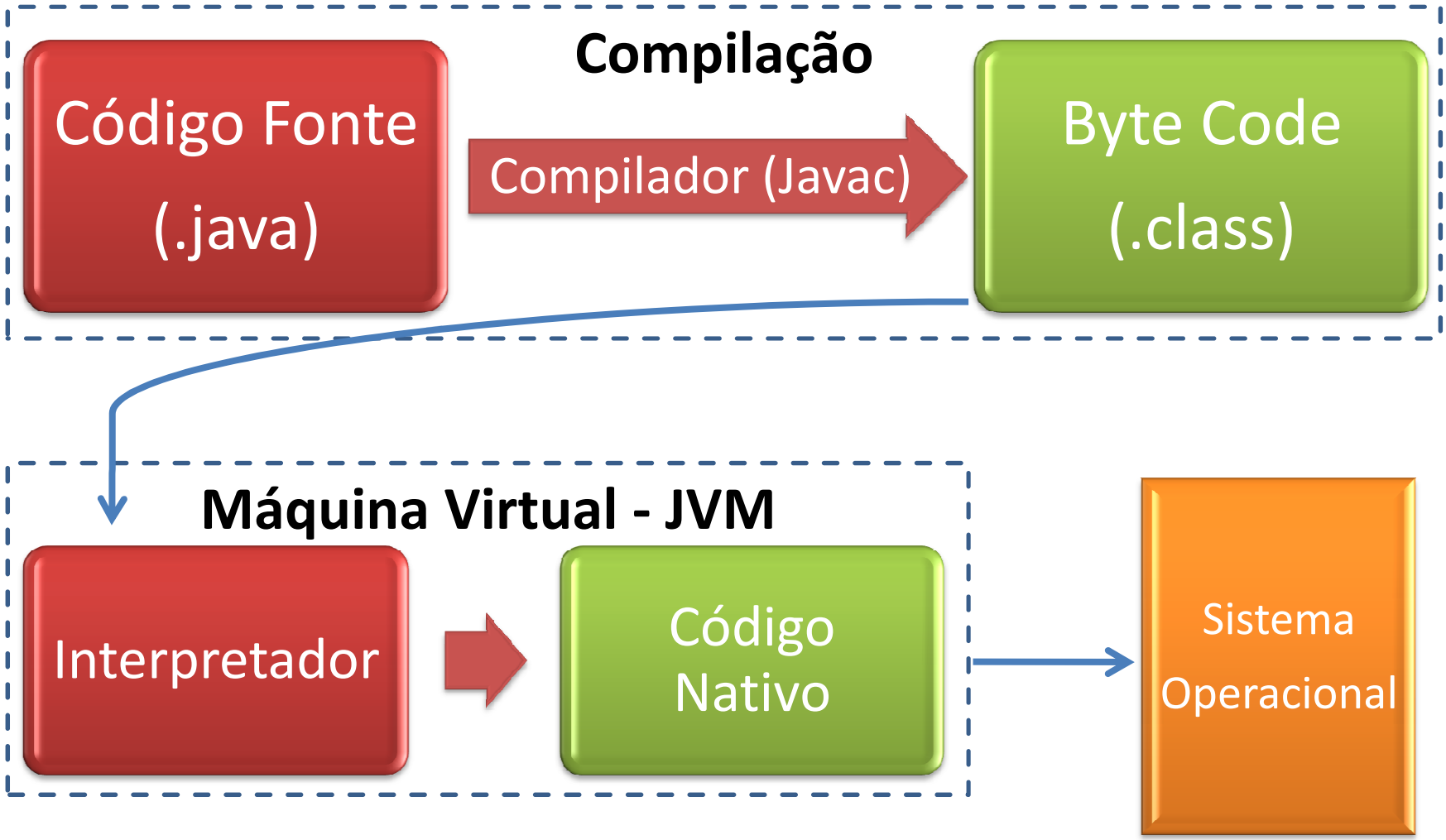
- Faça o download em www.java.sun.com e instale!

- JDK fornece os dois principais programas:

  – Compilador Java (**javac.exe**)

  – Interpretador Java (**java.exe**)

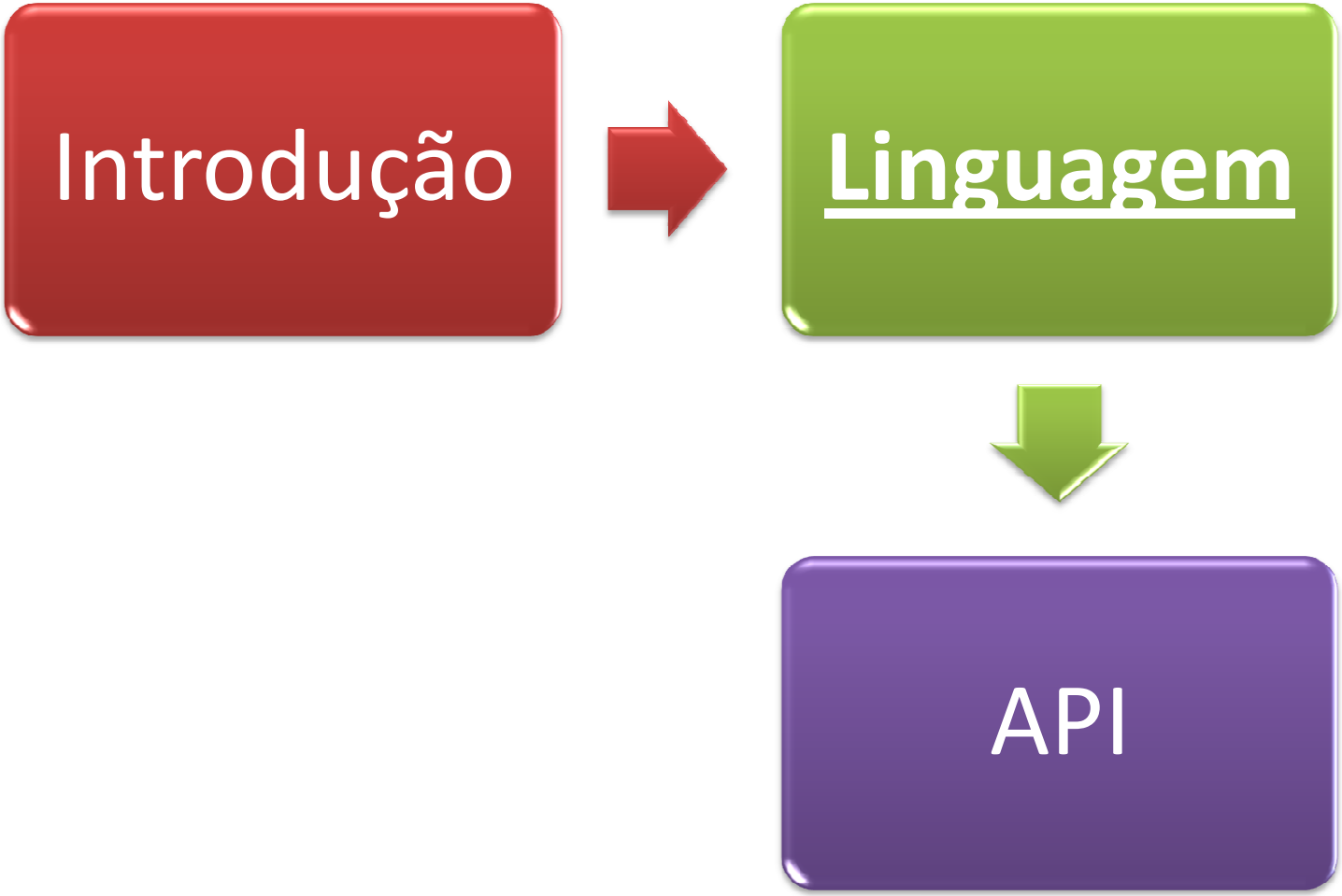
Ambiente Java



Primeiro Programa

```java
/**
 * O nome do arquivo deve se chamar NomeDaClasse.java,
 * ou seja, PrimeiroPrograma.java!
 *
 * @author Leonardo
 */
public class PrimeiroPrograma
{
        /* Um programa java começa com uma chamada
           ao método main() */
        public static void main( String[] args )
        {
                System.out.println("Primeiro programa!!!");
        }
}
```

Linguagem Java



Comentários

```
// Comentário de uma linha
```

```
/*
   Comentário de
   múltiplas linhas
 */
```

```
/**
   Comentário usando para documentação...

   @param s Descrever algo sobre o parametro
   @return Descrever algo sobre o retorno
   @throws NumberFormatException se...
   @see #parseInt(String, int)
 */
```

Blocos e Sentenças

- Uma sentença (*statement*) é uma linha de código terminada por um "**;**"

```
total = a + b + c;
```

- Um bloco de código é delimitado por chaves

```
{
    total = a + b + c;
}
```

Identificadores

- São **nomes** usados para **referenciar** classes, métodos, variáveis e constantes.
- Composição:
  - Seqüência de caracteres:
    - Letras
    - Dígitos
    - underscores (_)
    - símbolos monetários ($)
    - Não pode começar por dígito
    - São "*case sensitive*"
  - Exemplos:
    - number, Number, sum_$, bingo, $$_100, mal, grüβ
  - Identificadores ilegais:
    - 48casa, all/clear, navio-escola

Constantes e Variáveis

- Variáveis

```
int ç = 0;                    // Ok!
int único = 1;                // Ok!
int _underscore = 2;          // Ok!
int $salario = 10000;         // Ok!
int 7up = 10;                 // Errado!!!
int #codigo = 1234;           // Errado!!!
int navio-escola = 1234;      // Errado!!!
```

- Constantes

```
final double VALOR_PI = 3.1416;

final double VALOR_PI;
VALOR_PI = 3.1416;     // Atribuir 1 vez (OK)
VALOR_PI = 3.14;       // 2 vez (Erro!!!)
```

Palavras reservadas no Java

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

- Modificadores de acesso
- Controle de fluxo dentro de um bloco de código
- Controle de pacotes
- Variáveis de referência
- Palavras reservadas não utilizadas
- Modificadores de classes, variáveis ou métodos
- Tratamento de erros
- Primitivos
- Retorno de um método
- Literais reservados

Tipos Primitivos (Inteiros)

Tipo	Exigência de Armazenamento	Intervalo (Inclusive)
int	4 bytes	-2.147.483.648 +2.147.483.647
short	2 bytes	-32.768 +32.767
long	8 bytes	-9.223.372.036.854.775.808L +9.223.372.036.854.775.807L
byte	1 byte	-128 +127

```
byte b = 100;
short s = 1000;
int i = 100000;
int i_hex = 0xffff;              // Hexadecimal
int i_oct = 0765;                // Octal
long l = 10000000L;
```

Tipos Primitivos (Ponto Flutuante)

Tipo	Exigência de Armazenamento	Intervalo
float	4 bytes	Aproximadamente ±3,40282347E+38F (6-7 dígitos decimais significativos)
double	8 bytes	Aproximadamente ±1,79769313486231570+308 (15 dígitos decimais significativos)

```
float f = 1.1234F;
double d = 2.123456789;
```

Tipos Primitivos (Caractere)

Tipo	Exigência de Armazenamento	Intervalo
char	2 bytes	Engloba todos os 65.536 caracteres do Unicode.

- Aspas simples são usadas para representar constantes char.
- Como o ASCII é um subconjunto do Unicode, os 128 primeiros caracteres são os mesmos do ASCII.
- O prefixo \u indica um valor Unicode e os quatro dígitos hexadecimais informam o caractere Unicode.

```
char c1 = 'a';
char c2 = '\u89ab';
```

Tipos Primitivos (Booliano)

Tipo	Exigência de Armazenamento	Intervalo
boolean	1 byte	true ou false

```
boolean b1 = true;
boolean b2 = false;
boolean b3 = 0;          // Errado!!!
boolean b4 = 1;          // Errado!!!
```

Conversões (Tipos Numéricos)

- Se qualquer um dos operandos for do tipo **double**, então, o outro será convertido para **double**.

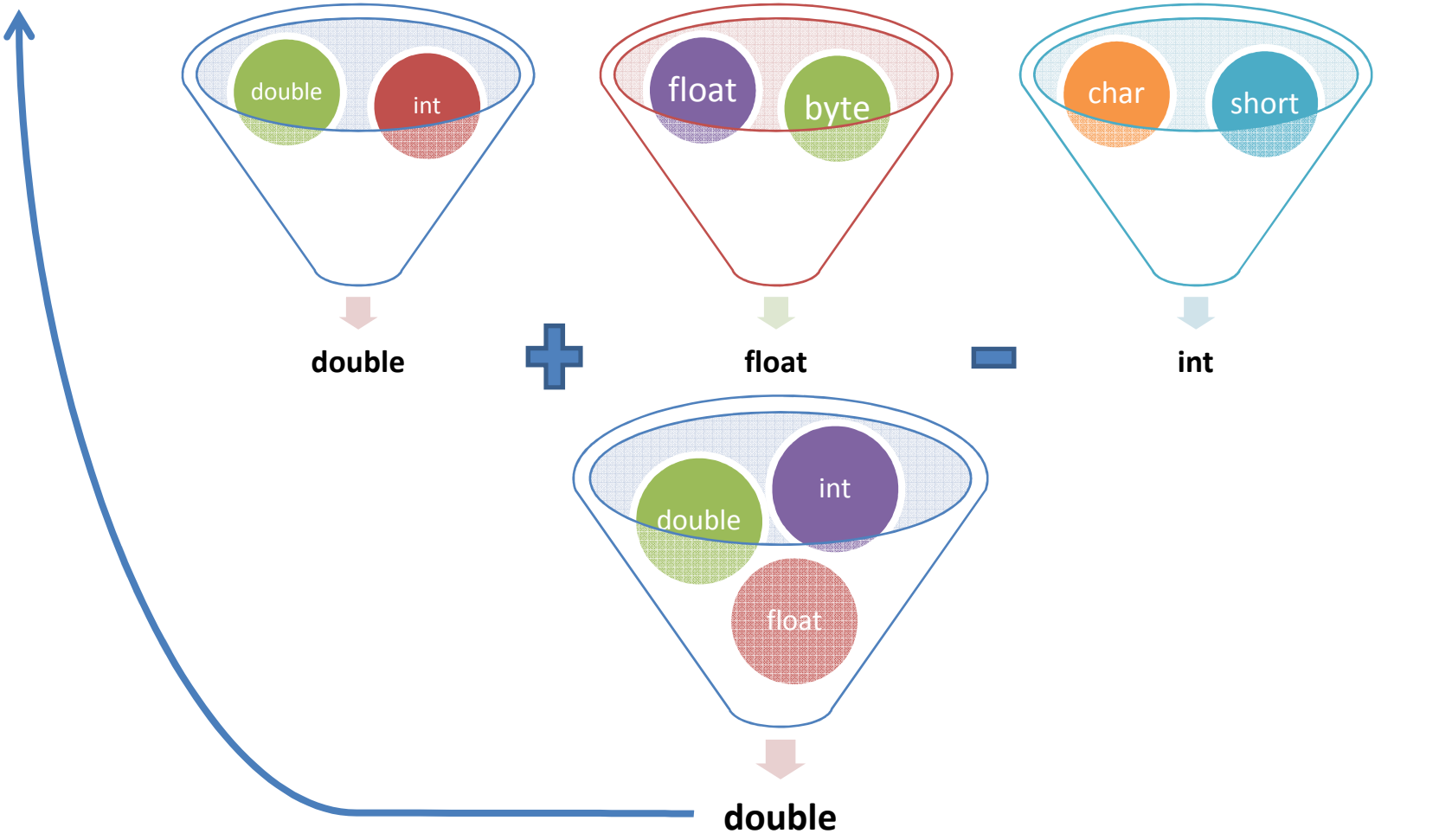
- De outro modo, se qualquer um dos operandos for do tipo **float**, o outro será convertido para **float**.

- Caso contrário, se qualquer um dos operandos for do tipo **long**, o outro será convertido para **long**.

- Funciona de forma semelhante entre os tipos inteiros: **int**, **short** e **byte**.

Conversões (Tipos Numéricos)

```
total = ( d * i ) + ( f * -b ) - ( c / s );
```



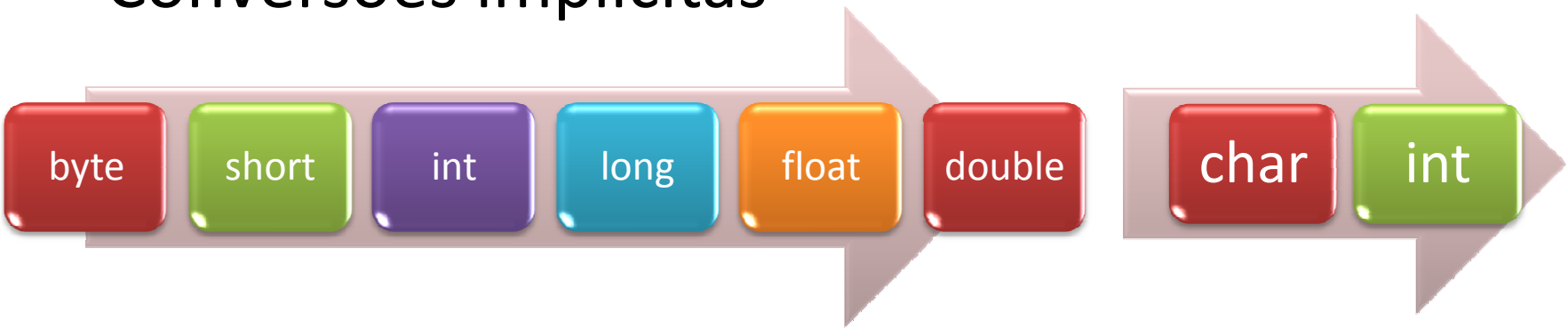
Conversões (Tipos Numéricos)

- As conversões explícitas de tipo (*casts*) podem **causar perda** de informações.

```
double d = 9.999;
int i = (int) d;                    // = 9

double d = 9.999;
int i = (int) Math.round(d);        // = 10
```

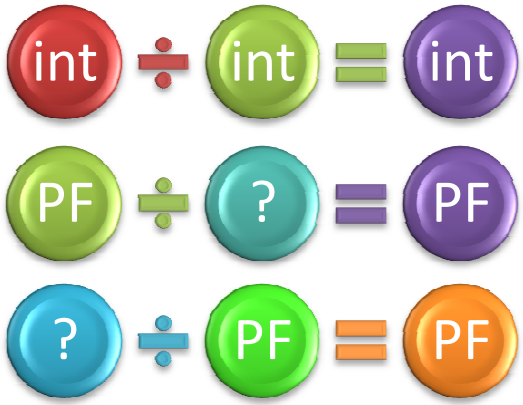
- Conversões implícitas



Operadores

- Operadores aritméticos:
  - **+** (Adição), **-** (Subtração), * (Multiplicação) e **/** (Divisão)
- **%** (Resto de uma divisão)
- **Observações**:

Detalhes sobre a divisão




```
double d = 10 / 3;                  // = 3.0
double d = 10 / 3.0;                // = 3.3333...
int i = 10 % 3;                     // = 1

int i = 5;
i += 10;      // i = i + 10         // = 15
```

Operadores

- Exponenciação
  - Não possui um operador específico.

```
double i = Math.pow(2, 3);
```

- Incremento e decremento
  - Forma pós-fixada: **i++**, **i--**
  - Forma pré-fixada: **++i**, **--i**

```
int m = 7;
int n = 7;
int a = 2 * ++m;     // agora a é 16, m é 8
int b = 2 * n--;     // agora b é 14, n é 6
```

Operadores

- Relacionais e Lógicos
  - Igualdade

```
(3 == 7) -> false
```

  - Desigualdade

```
(3 != 7) -> true
```

  - >, <, >= e <=

```
(4 > 3)  -> true
(3 < 3)  -> false
(3 >= 3) -> true
(3 <= 4) -> true
```

Operadores

- Relacionais e Lógicos – Curto circuito
  - **&&** (E), **||** (OU)

```
// Não testa o segundo operador
(false && ?) -> false
(true  || ?) -> true

// Testa o segundo operador
(true  && false) -> false
(true  && true)  -> true
(false || false) -> false
(false || true)  -> true
```

Operadores

- Bit a bit
  - **&** (E lógico), **|** (OU lógico), **^** (OU Exclusivo lógico) e **~** (Negação lógica)

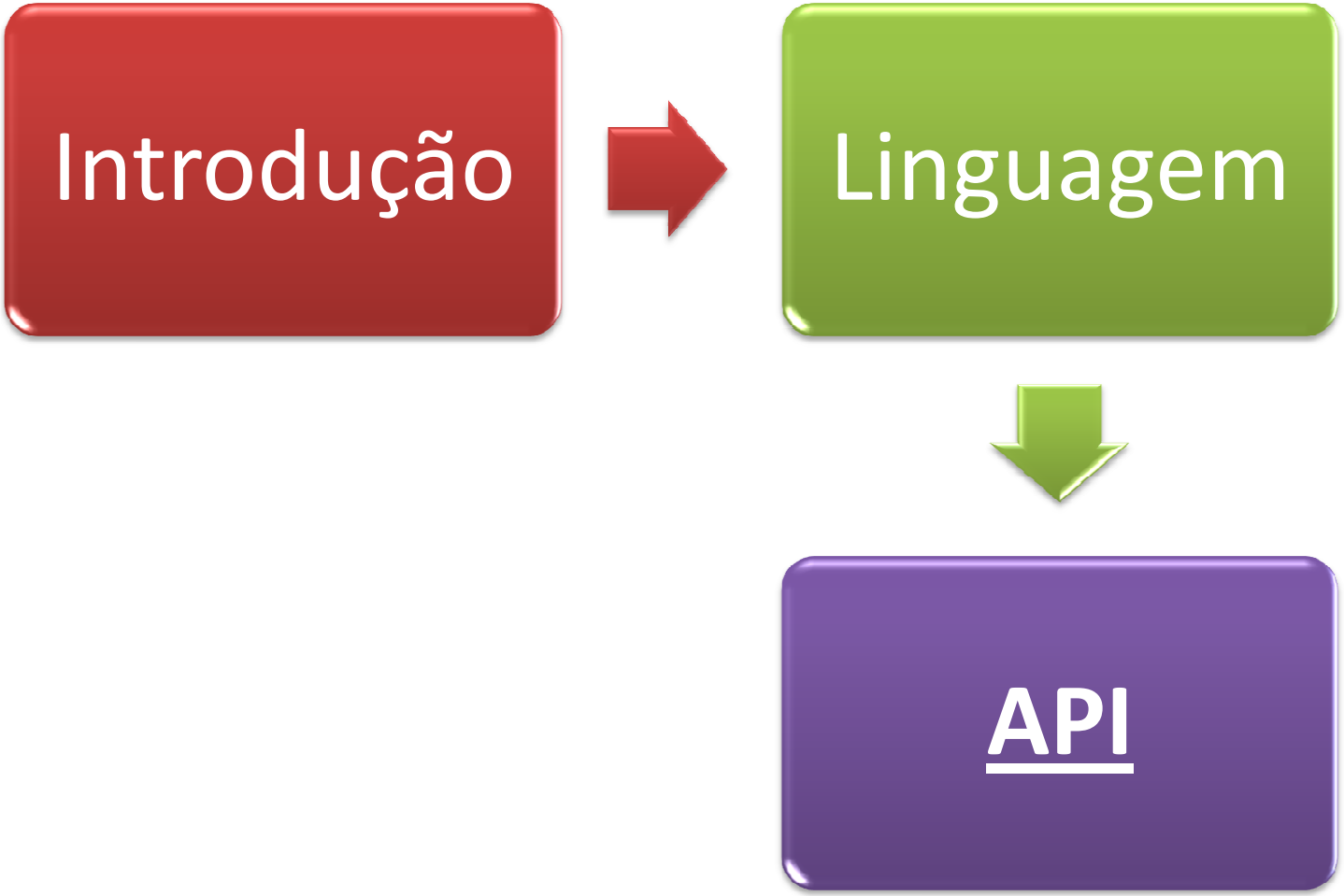
```
byte b1 = 5; // 00000101
byte b2 = 4; // 00000100

(b1 & b2) -> 00000100 = 4
(b1 | b2) -> 00000101 = 5
(b1 ^ b2) -> 00000001 = 1
(~b1)     -> 11111010 = -6
```

Operadores

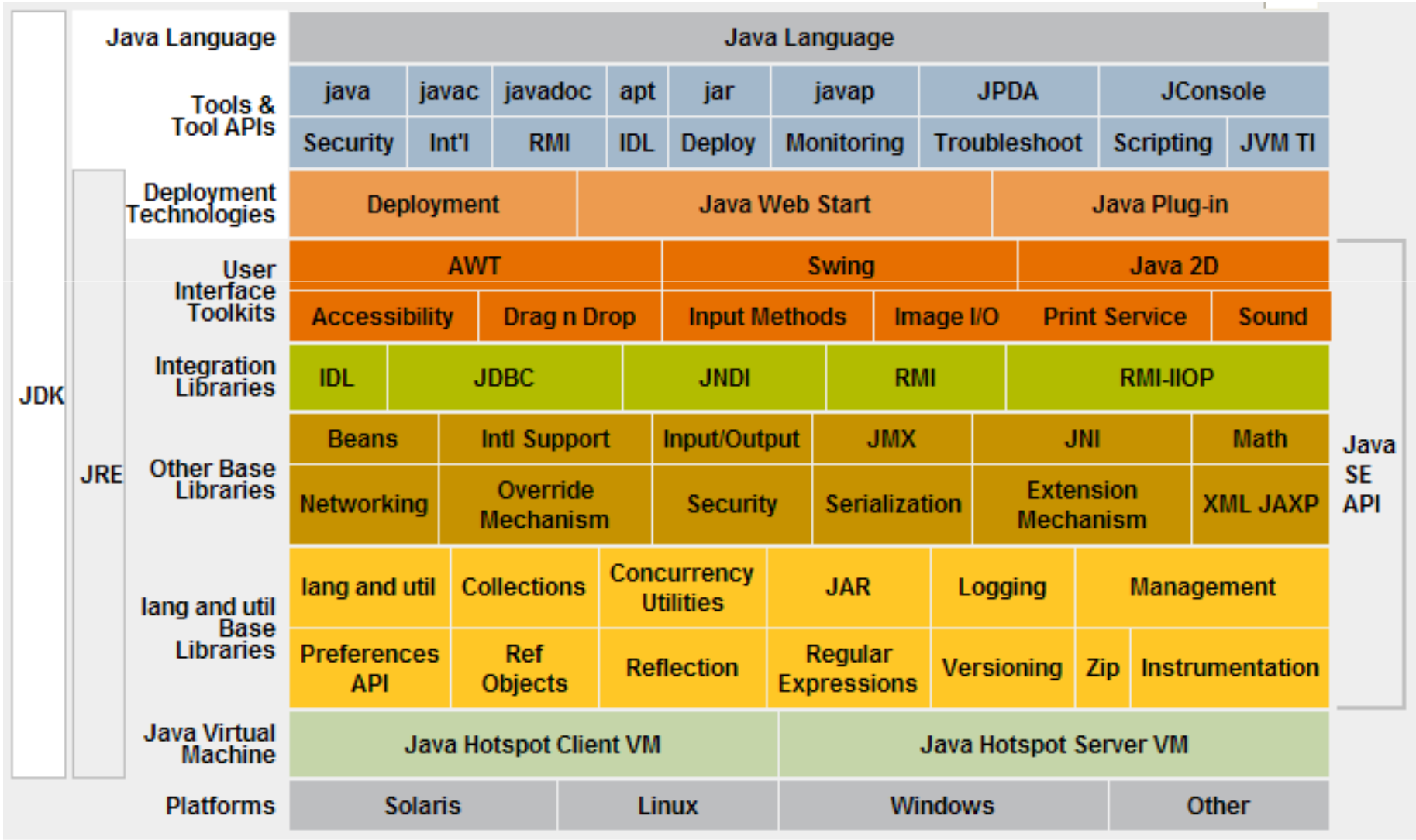
Tipo de Operadores	Operadores
Operadores Sufixais	[] . (parâmetros) x++ x--
Operadores Prefixais	++x --x +x –x ~ !
Criação e Type Cast	new (Type)
Operadores Multiplicativos	* / %
Operadores Aditivos	+ -
Operadores de Shift	<< >> >>>
Operadores Comparativos	< <= > >= instanceof
Operadores de Igualdade	== !=
Operadores Bit a Bit	& ^ |
Operadores AND e OR	&& ||
Operadores Ternários	?:
Atribuições	= += -= *= /= <<= >>= >>>= &= ^= |=

API Java



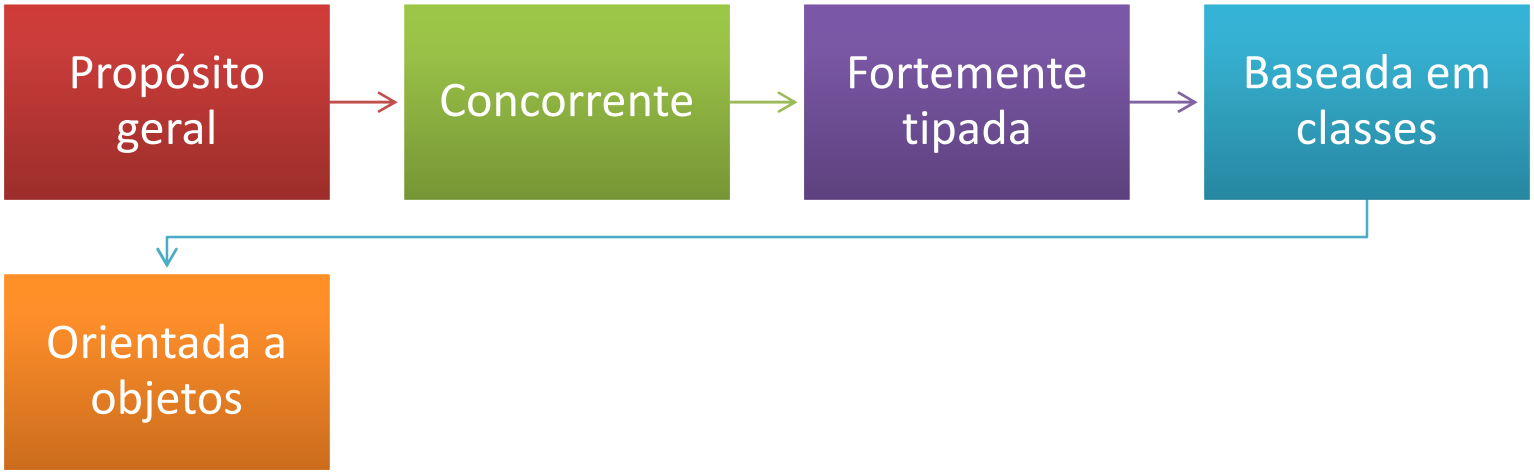
Arquitetura Java SE

Mais informações em: **http://java.sun.com/javase/technologies/index.jsp**



Linguagem Java

- A linguagem de programação Java é...



- É normalmente compilada para um conjunto de instruções (***bytecode***) binárias definidas na especificação da JVM.

Ferramentas

java

- Executa uma aplicação Java
- Inicia JVM, carrega classe especificada e executa o método **main()**

javac

- Ler o código escrito em Java
- Compila
- Gera um arquivo de ***bytecode*** .class

javadoc

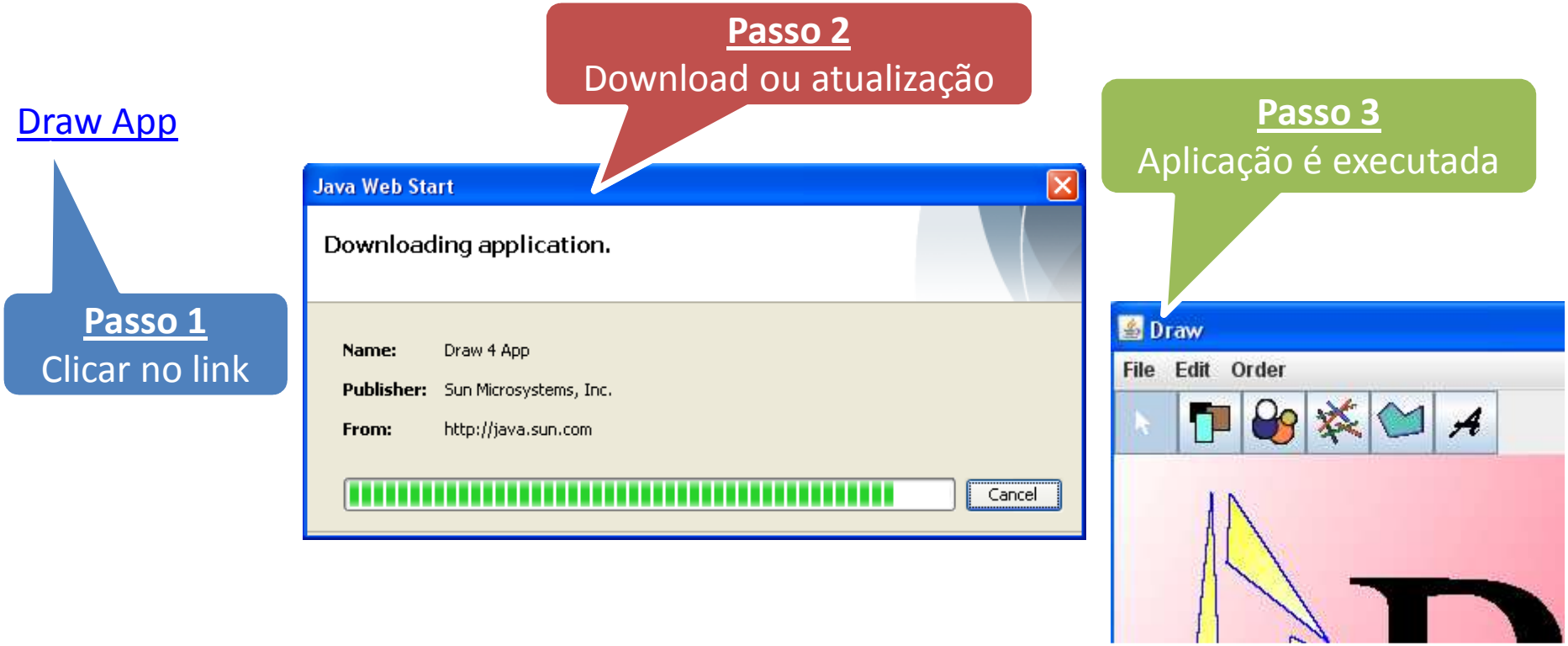
- Ler os comentários de documentação
- Gerar um conjunto de páginas em HTML descrevendo as classes, métodos, atributos...

JAR

- Utilitário usado para geração do Java ARchive (JAR).

Tecnologias para *deployment*

- Java Web Start
  - Tecnologia para instalação de aplicações desktop com um único clique a partir do browser.



Java SE API: Interface c/ Usuário

- Abstract Window Toolkit (AWT)
  - Usada para programação de GUI.
  - Possui muitos código nativo (dependente do SO)
  - Mais leve
- Swing
  - Usada para programação de GUI.
  - 100% construído com código Java
  - Mais pesado

Java SE API: Bibliotecas de Integração

- *Java Database Connectivity* (JDBC)
  - Prover para a linguagem Java um acesso universal à **banco de dados**.
  - JDBC está localizada nos pacotes: **java.sql** e **javax.sql**.
- *Java Naming and Directory Interface* (JNDI)
  - Prover mecanismos comuns para acesso a serviços de nomes e diretórios a partir de aplicações escritas em Java.

Java SE API: Bibliotecas Base

- **java.lang** e **java.util**
  - Prover as funcionalidades básicas usadas por quase todos os programas.
- **Collections Framework**
  - Define uma arquitetura unificada para representar e manipular coleções.
- **Reflection (java.lang.reflect)**
  - Permite um código Java obter informações sobre atributos, métodos e construtores das classes carregadas.

Máquina Virtual Java

- **Compilação adaptativa**
  - A aplicação é executada em modo interpretado, isto é, não é feita **nenhuma otimização inicial**.
  - Enquanto a aplicação roda, são feitas **estatísticas de runtime**, verificando quais são os **pontos mais executados** do programa (hotspots).
  - Estes trechos (hotspots), então, são **fortemente otimizados**.

Máquina Virtual Java

- **Alocação de memória e coletor de lixo**
  - Prover uma alocação rápida de memória para objetos.
  - Oferece um coletor de lixo (responsável por liberar memória) muito eficiente e rápido.