Conceitos e Fundamentos de Java

Prof. Dr. Lucas C. Ribas

Disciplina: Programação Orientada a Objetos

Departamento de Ciências de Computação e Estatística

unesp
UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"

IBILCE / UNESP - CÂMPUS DE SÃO JOSÉ DO RIO PRETO

Agenda

- Tecnologia Java
- Variáveis
- Comandos e Comentários
- Coletor de Lixo
- Entrada e saída padrão
- Java *vs* C++

unesp

Tecnologia Java

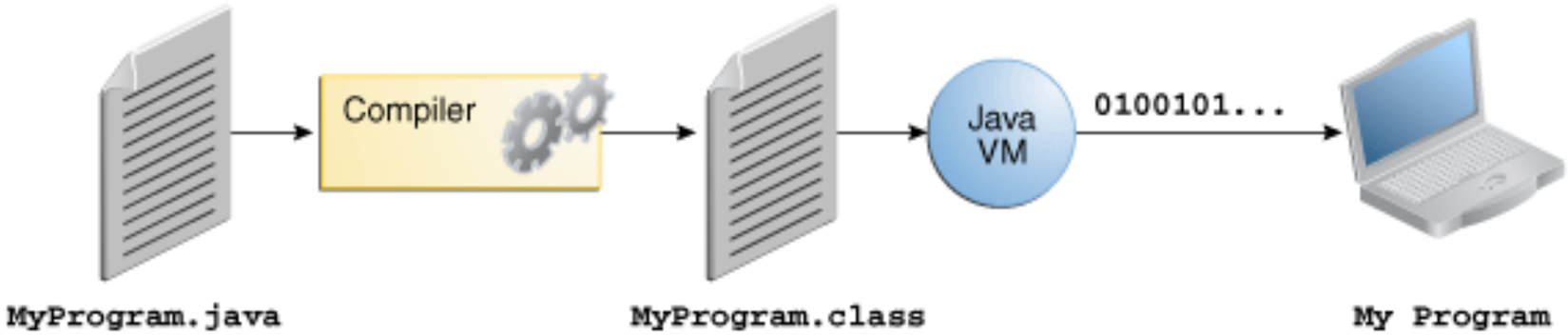
- A tecnologia Java é considerada tanto uma **linguagem** de programação como uma **plataforma**
- Java como linguagem de programação
  - Propriedades **http://www.oracle.com/technetwork/java/langenv-140151.html**
  - Códigos fonte são criados em arquivos texto com extensão *.java*
  - Após compilação, gera-se um arquivo *.class*
    - Bytecodes (linguagem de máquina da Java Virtual Machine, JVM)
  - Código compilado é interpretado pela JVM

3

unesp

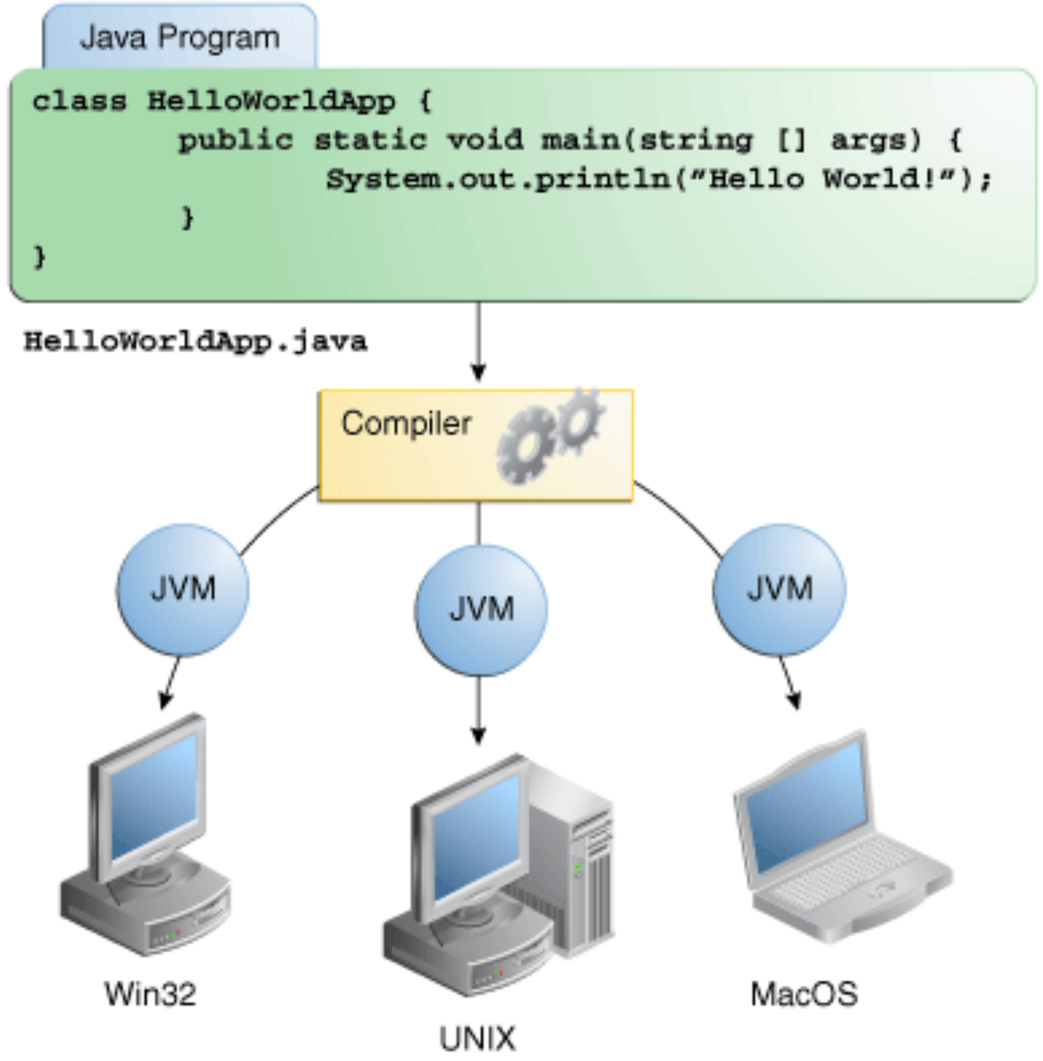
Tecnologia Java

- Comandos importantes
  - Compilação: *javac*
  - Execução: *java*
  - Documentação: *javadoc*



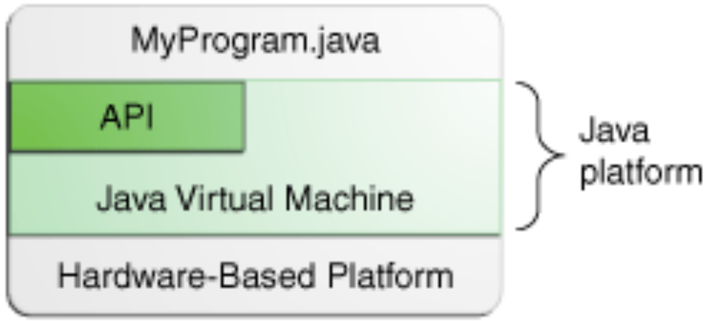
Tecnologia Java

- Portabilidade
- A JVM está disponível para diversas plataformas
- O mesmo arquivo *.class* pode ser executado em diferentes SOs

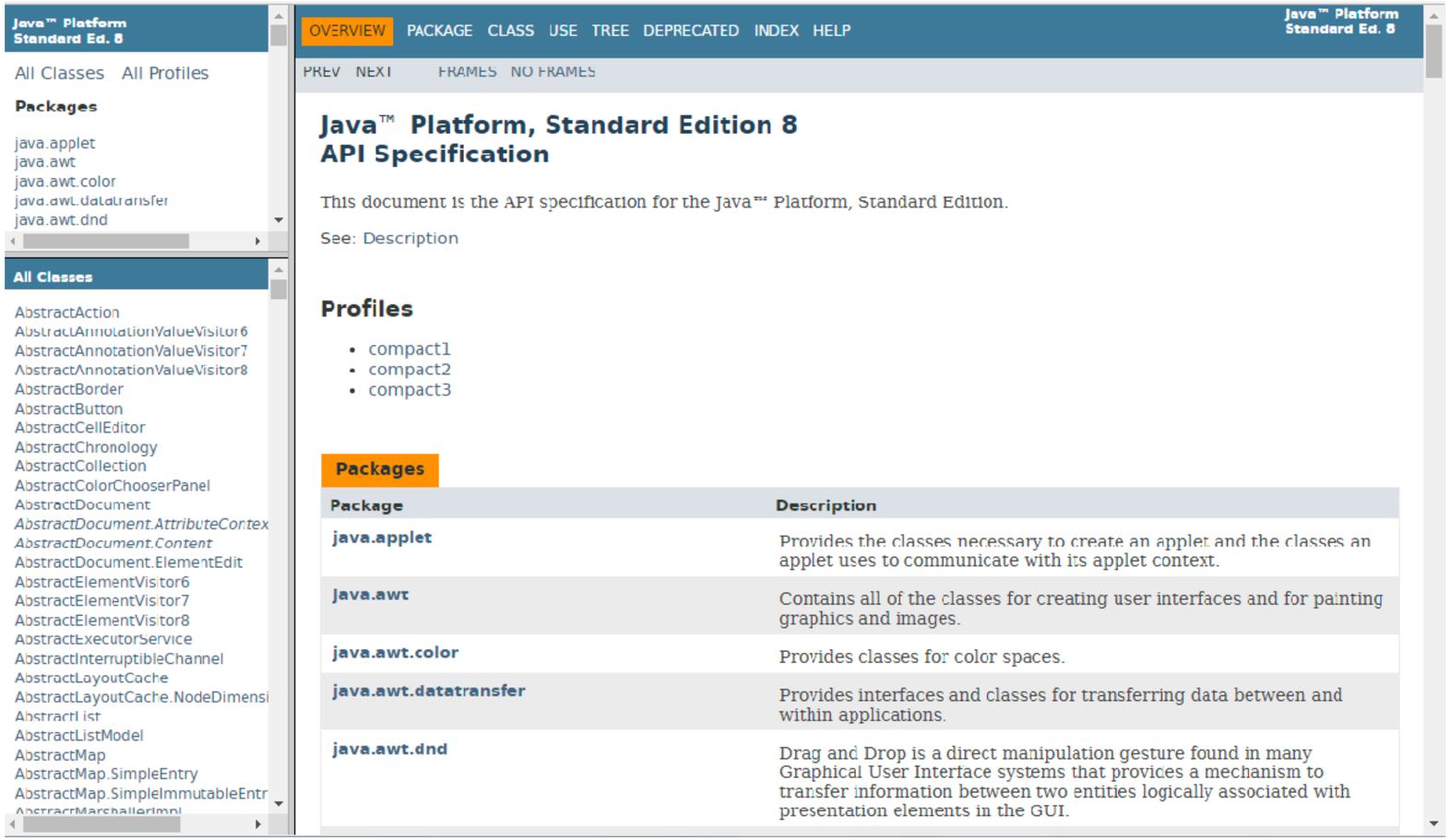


Tecnologia Java

- Java como plataforma
  - Uma plataforma é um ambiente de hardware ou software no qual um programa é executado
    - Muitas plataformas combinam hardware e software
  - A plataforma Java tem dois componentes de software
    - Java Virtual Machine (JVM)
    - Java Application Programming Interface (API)
  - A API do Java é uma coleção de componentes de softwares
    - Agrupados em pacotes



- Documentação Online da API Java
  - https://docs.oracle.com/javase/8/docs/api/index.html



Tecnologia Java

- Principais edições do Java
  - **Java SE** (Standard Edition): Edição padrão, que permite criar programas para desktops e servidores, assim como alguns dispositivos "embedded".
  - **Java EE** (Enterprise Edition): Voltada para grandes empresas. Possui mais recursos, permite desenvolvimento de aplicativos Web;
  - **Java ME** (Micro Edition): provê suporte para dispositivos portáteis, como celulares, PDAs, conversores de sinais para TV e impressoras;
  - Outras

Variáveis e tipos primitivos

Variáveis em Java

- Variáveis (ou campos)
  - **Variáveis de instância**: únicas para cada instância
    - int velocidadeAtual;
  - **Variáveis de classe**: uma única variável compartilhada com todas as instâncias
    - Modificador *static*
    - Ex: número de marchas de uma Bicicleta static marchas = 6;
    - Poderíamos ainda adicionar o modificador *final,* caso não queiramos que seu valor seja alterado static final MARCHAS = 6;

unesp

Variáveis em Java

- Variáveis
  - Variáveis locais: escopo menor (dentro de um método)
    - `int count = 0;`
  - Parâmetros: variáveis recebidas em um método
    - `public static void main(String[] args)`

unesp

Variáveis em Java

- Nomes podem começar com uma letra, $ ou _
  - Convenções
    - Começar com uma letra
    - Evitar usar $
- Caracteres seguintes podem ser letras, números, _ ou $
  - Convenções
    - Usar nomes completos ao invés de abreviações
    - Ex: velocidade, marcha são mais intuitivos do que v, m

unesp

Nomeando Variáveis

- Se for uma única palavra, usar todas as letras minúsculas
- Se o nome consistir de mais de uma palavra, definir em maiúsculo a primeira letra das palavras subsequentes
  - Ex: velocidadeAtual, minhaMelhorTaxa, …
- No caso de constantes, capitalizar todas as letras e separar as palavras pelo underscore
  - Ex: static final NUM_MARCHAS = 6;

unesp

Nomeando Variáveis

- Exemplos:

MarchaAtual	→	marchaAtual
dia_do_ano	→	diaDoAno
MES-NASCIMENTO	→	mesNascimento
Tempoesperatotal	→	tempoEsperaTotal
Idade	→	idade

unesp

Tipos Primitivos

- Em Java há 8 tipos primitivos de variáveis
  - **byte:** 8 bits (-128 a 127);
  - **short:** 16 bits (-32.768 a 32.767);
  - **int:** 32 bits (-2^{31} a $2^{31}-1$ ou 0 a $2^{32}-1$);
  - **long:** 64 bits (-2^{63} a $2^{63}-1$ ou de 0 a $2^{64}-1$);
  - **float:** 32 bits, precisão simples (IEEE754)
  - **double:** 64 bits, precisão dupla (IEEE754)
    - https://www.h-schmidt.net/FloatConverter/IEEE754.html
  - **boolean**: true ou false. Tamanho não é bem definido.
  - **char**: 16 bits
    - '\u0000' (ou 0) a '\uffff' (ou 65535) (código Unicode do caracter)

unesp

Tipos Primitivos

- **Campos** declarados mas não inicializados, assumem valores padrões

Data Type	Default Value (for fields)
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	'\u0000'
String (or any object)	null
boolean	false

Tipos Primitivos

- Literais são representações de valores fixos

```
boolean result = true;
char capitalC = 'C';
byte b = 100;
short s = 10000;
int i = 100000;
```

Tipos Primitivos

- Literais inteiros
  - Literais **long** devem terminar com L (preferível) ou l
    - Ex: 12043L
  - Do contrário, é considerado **int**
  - Byte, short, int e long podem ser criados utilizando representação decimal, hexadecimal ou binária
    - Decimal `int decVal = 26;`
    - Hexadecimal `int hexVal 0x1a;`
    - Binário `int binVal = 0b11010;`

unesp

Tipos Primitivos

- Literais ponto flutuante
  - Literais **float** devem terminar com F ou f
  - Caso contrário são considerados do tipo **double**
    - Tipo double pode terminar com D ou d
  - A representação de pontos flutuantes pode ser feita em notação científica (terminada com E ou e)

```
double d1 = 123.4;
double d2 = 1.234e2;
float f1 = 123.4f;
```

Tipos Primitivos

- Literais numéricos (geral)
  - Os literais numéricos podem conter underscore (_) para facilitar a compreensão do código
    - Separar grupo de dígitos

```
long creditCardNumber = 1234_5678_9012_3456L;
long socialSecurityNumber = 999_99_9999L;
float pi = 3.14_15F;
long hexBytes = 0xFF_EC_DE_5E;
long hexWords = 0xCAFE_BABE;
long maxLong = 0x7fff_ffff_ffff_ffffL;
byte nybbles = 0b0010_0101;
long bytes = 0b11010010_01101001_10010100_10010010;
```

Tipos Primitivos

- Literais caracter e String
  - Literais do tipo char e String podem conter qualquer caracter Unicode UTF-16
  - Se o editor e o sistema de arquivo permitirem, é possível trabalhar diretamente com o caracter
  - Caso contrário, é possível utilizar o “Unicode Scape”
    - Exemplos
      - ‘ \u0108’ (C com acento circunflexo)
      - “S\u00ED Se\u00F1or” (Sí Señor)
  - Também há suporte para sequências de escape especiais
    - \b (backspace), \t (tab), \n (line feed), \f (form feed), \r (carriage return), \" (double quote), \' (single quote), e \\ (backslash).

unesp

Arrays

Arrays

- Em Java, não basta declarar arrays, é preciso alocar a quantidade de elementos desejada

- Declaração de arrays

```
int[] anArrayOfInts;
float[] anArrayOfFloats;
string[] anArrayOfStrings;
```

  - Colchetes após o tipo é preferível do que após o nome da variável

- Criação (alocação) do array é feito através do operador **new**

```
arrayOfInts = new int[10];
arrayOfFloats = new float[15];
```

unesp

Arrays

- Arrays em Java começam do índice zero

```
arrayOfInts[0] = 100; // Inicia primeiro elemento
arrayOfInts[1] = 120;
arrayOfInts[2] = 140;
...
```

- Há uma maneira direta de declarar e inicializar um array

```
arrayOfInts[0] = {10, 15, 20, 25, 30};
```

  - Neste caso, aloca 5 posições

unesp

Arrays

- De forma similar é possível declarar e criar arrays multidimensionais
  - Considerado como array de arrays

```
int matrix[][];
float tripleArray[][][];
```

- A criação pode ser direta (junto com a declaração) ou usando o operador **new**

```
anMatrix = new int[5][8]; // opcao1
anMatrix = {{1,2,4}, {3,8,7}, {5,9,1}} // opcao2
```

Arrays

- Arrays multidimensionais não precisam ter o mesmo número de elementos para cada dimensão
  - Consequência do fato do Java tratar arrays multidimensionais como array de arrays
  - Por exemplo, em um array bidimensional, cada elemento é um array que pode ter qualquer número de elementos

```
int[][] matrix = { {1,2,4}, {3}, {5,9} };
```

unesp

Arrays

- Todo array possui uma propriedade interna *length*
  - Estrutura interna de um array
  - Definido pelo Java
  - Permite saber o tamanho do array
  - Exemplo:

```
int[] array = new int[10];
System.out.println(array.length);
```

unesp

Manipulação de Arrays

- System.arraycopy (método da classe System)
  - public static void arraycopy(Object src, int srcPos, Object dest, int destPos, int length)
- Java.util.Arrays (classe)
  - copyOfRange
  - binarySearch
  - equals
  - fill
  - sort e paralellSort

unesp

Operadores

Operadores

Operators	Precedence
postfix	`expr++ expr--`
unary	`++expr --expr +expr -expr ~ !`
multiplicative	`* / %`
additive	`+ -`
shift	`<< >> >>>`
relational	`< > <= >= instanceof`
equality	`== !=`
bitwise AND	`&`
bitwise exclusive OR	`^`
bitwise inclusive OR	`|`
logical AND	`&&`
logical OR	`||`
ternary	`? :`
assignment	`= += -= *= /= %= &= ^= |= <<= >>= >>>=`

Operadores

- Prefixo x Pósfixo

```
class PrePostDemo {
    public static void main(String[] args) {
        int i = 3;
        i++;
        System.out.println(i);    // prints 4
        ++i;
        System.out.println(i);    // prints 5
        System.out.println(++i); // prints 6
        System.out.println(i++); // prints 6
        System.out.println(i);    // prints 7
    }
}
```

unesp

Operadores

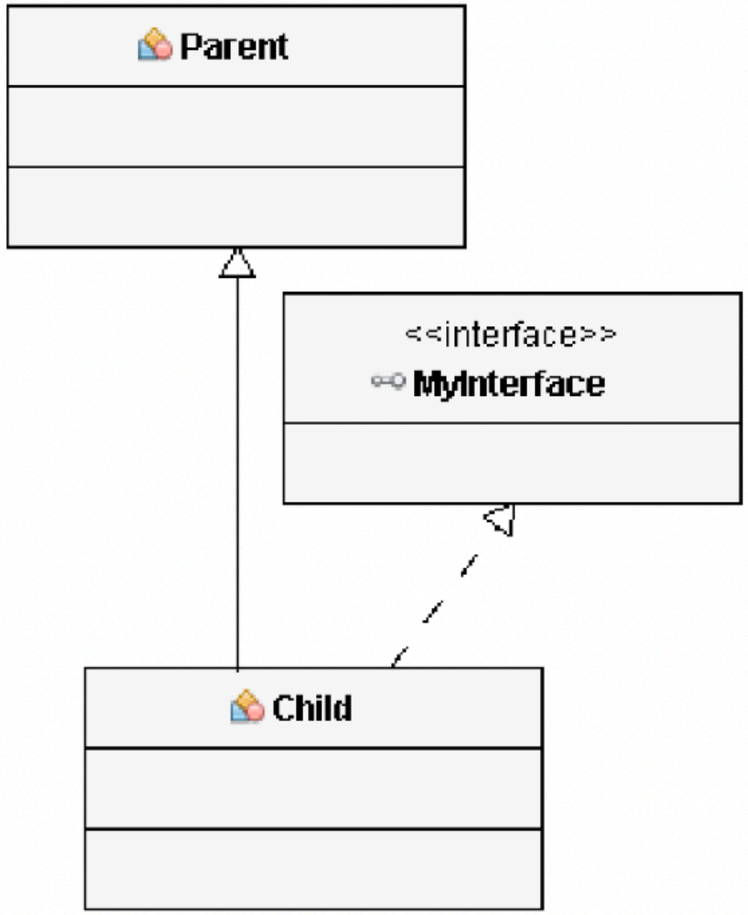
- Comparador de objetos: *instanceof*
  - Testa se um objeto é uma instância de um tipo de classe específico
  - Também usado para verificar se um objeto implementa uma interface
    - Se é instância de uma classe que implementa determinada interface
    - É uma comparação: retorna **true** ou **false**
  - Exemplo
    - objeto *instanceof* Classe

unesp

Operadores

```
Parent obj1 = new Parent();
Parent obj2 = new Child();

obj1 instanceof Parent:  true
obj1 instanceof Child:  false
obj1 instanceof MyInterface:  false
obj2 instanceof Parent:  true
obj2 instanceof Child:  true
obj2 instanceof MyInterface:  true
```



Controles de Fluxo

unesp

Controles de Fluxo

- **Tomadas de decisão**
  - if-then
  - if-then-else
  - statement ? action1 : action2
  - switch
- **Laços**
  - for
  - while
  - do-while
- **Ramificações**
  - break
  - continue
  - return

unesp

Controles de Fluxo

- **If-then-else**

```
int testscore = 76;
char conceito;
if (testscore >= 90) {
        conceito = 'A';
} else if (testscore >= 80) {
        conceito = 'B';
} else if (testscore >= 70) {
        conceito = 'C';
} else if (testscore >= 60) {
        conceito = 'D';
} else {
        conceito = 'F';
}
conceito = ‘C’;
```

unesp

Controles de Fluxo

- **statement ? action1 : action2**
  - Forma compacta do if-then-else
  - statement
    - Comando que será avaliado
  - action1
    - Ação caso *statement* seja true
  - action2
    - Ação caso *statement* seja false

```
int nota = 76;
char conceito;
nota > 50 ? conc = 'A' : conc = 'R';
```

Controles de Fluxo

- **switch**
  - Não esquecer do break
  - Tipos permitidos
    - byte
    - short
    - int
    - char
    - String
  - Classes que representam tipos primitivos: Character, Short, Byte, Integer

```
int month = 8; String monthString; switch (month) {
case 1:  monthString = "January";           break;
case 2:  monthString = "February";           break;
case 3:  monthString = "March";           break;
case 4:  monthString = "April";           break;
case 5:  monthString = "May";           break; …
default: monthString = "Invalid month";
break; }
August
```

Controles de Fluxo

- **while**

```
int count = 1;
while (count < 11) {
   System.out.println("Count is: "+ count);
   count++; }
```

- **do-while**

```
do {
   System.out.println("Count is: "+ count);
   count++; }
while (count < 11);
```

unesp

Controles de Fluxo

- for

```
for(int i=1; i<11; i++){
    System.out.println("Count is: "+ i);
}
```

```
int[] numbers = {1,2,3,4,5,6,7,8,9,10};
for(int item : numbers){
    System.out.println("Count is: "+ item);
 }
```

unesp

Controles de Fluxo

- **break**
  - Utilizado para terminar uma execução do switch, for, while, do-while
  - Se houver laço aninhado, termina o laço mais interno

```
for (i = 0; i < sequencia.length; i++) {
     if (sequencia[i] == searchfor) {
         encontrado = true;
         break;
         }
  }
```

unesp

Controles de Fluxo

- **continue**
  - Utilizado para pular a iteração atual em um for, while, do-while

```
for (i = 0; i < sequencia.length; i++) {
     if (sequencia[i] != searchfor) continue;
         cont++;

 }
```

unesp

Controles de Fluxo

- **return**
  - Utilizado para sair do método atual
  - Retorna o fluxo de controle para onde o método foi chamado
  - Pode ter ou não valor de retorno
    - Tipo do retorno deve coincidir com o tipo de retorno declarado na função
    - Pode ser tipos primitivos, arrays ou objetos

unesp

Comentários

- Há dois tipos básicos de comentários em Java
  - Comentário de linha única
    - *// Comentario*
  - Comentário de múltiplas linhas
    - */* Comentario com mais de uma linha *\/*
- Há um terceiro tipo, utilizado para documentação automatica do código
  - javadoc
  - Gera páginas html
  - Será explorada posteriormente

unesp

Entrada e Saída

unesp

Entrada e Saída Padrão

- System.in
  - Entrada padrão (java.io.InputStream)
- System.out
  - Saída padrão (java.io.PrintStream)
- System.err
  - Saída de padrão de erro (java.io.PrintStream)
- Objetos da classe System

unesp

Entrada e Saída Padrão

- System.in (java.io.InputStream)
  - Possui métodos para leitura de bytes
  - Mais fácil utilizar a classe Scanner em conjunto para fazer a leitura de tipos primitivos
    - java.util.Scanner

```
Scanner sc = new Scanner(System.in);
int i = sc.nextInt();
```

unesp

Entrada e Saída Padrão

- System.out (java.io.PrintStream)
  - Possui métodos para escrita de tipos primitivos, String e Objetos em geral

```
System.out.println(“Olá!”)
```

unesp

Coletor de Lixo

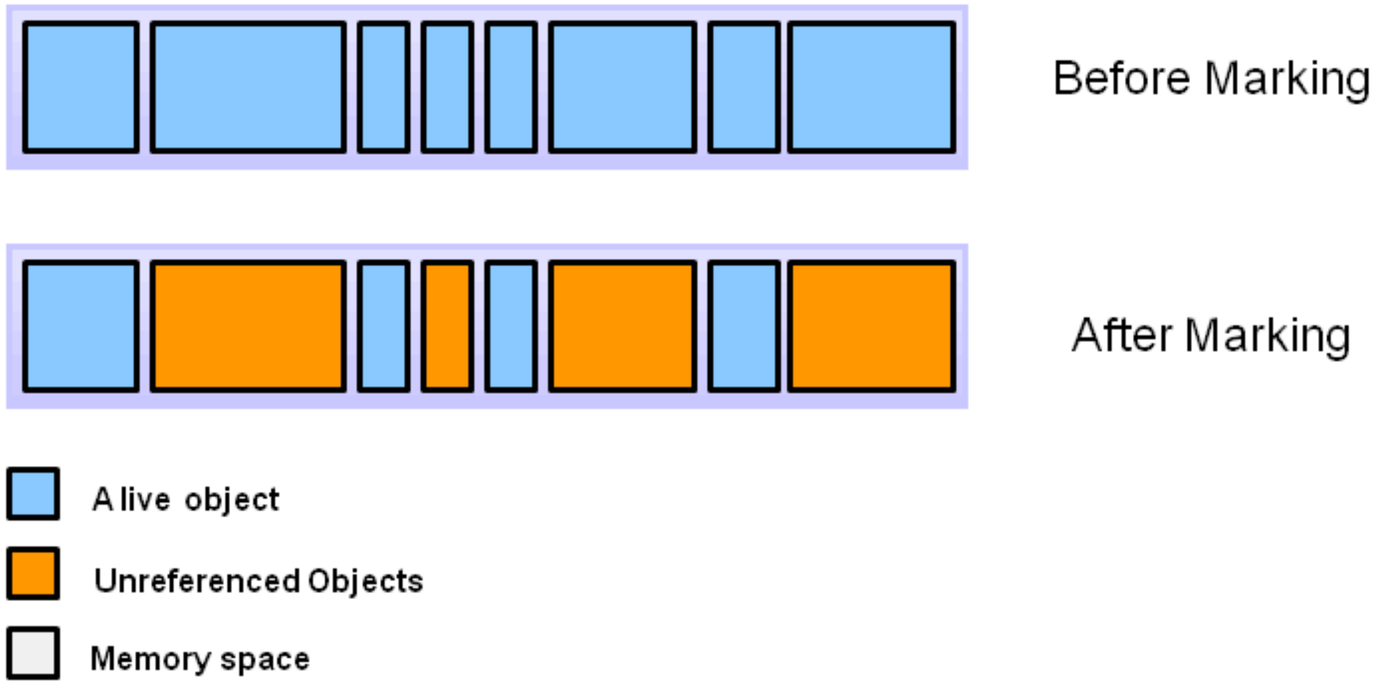
Coletor de Lixo

- Java possui um coletor de lixo (*garbage collector*) para limpeza de memória
  - Identifica os objetos na memória que não estão mais em uso
  - Objetos que não estão em uso são aqueles que não são referenciado por nenhuma parte do programa
  - A memória ocupada por estes objetos pode ser liberada
    - Marcação de que pode ser utilizada
- Assim, em Java, o processo de desalocação de memória é feito automaticamente, pelo *garbage collector*

unesp

Coletor de Lixo

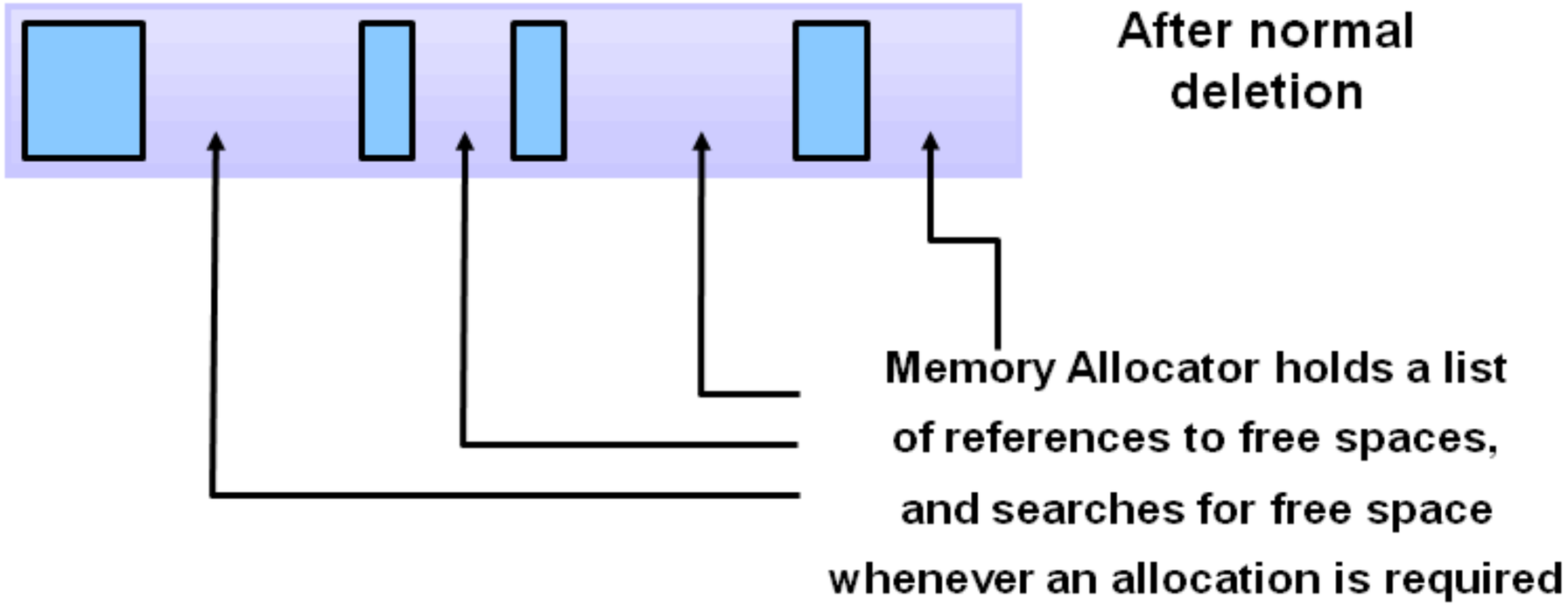
- Passo 1: marcação
  - Para cada objeto, verifica-se se há uma referência
  - Pode ser bastante custoso



Coletor de Lixo

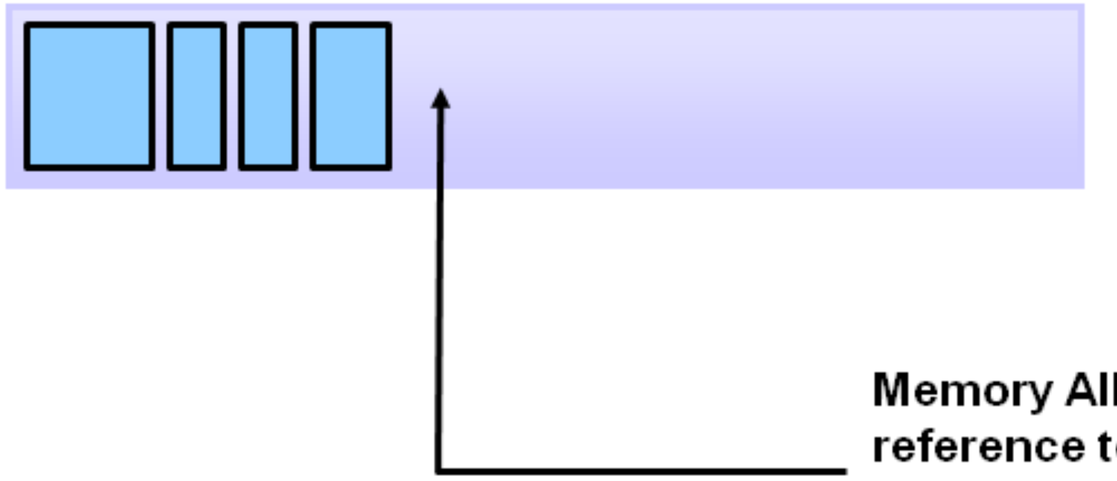
- Passo 2: deleção normal
  - Objetos referenciados são mantidos
  - Alocador de memória possui ponteiros para blocos livres

49



Coletor de Lixo

- Passo 2a: deleção com compactação
  - Além de remover objetos sem referência, é possível compactar os objetos remanescentes

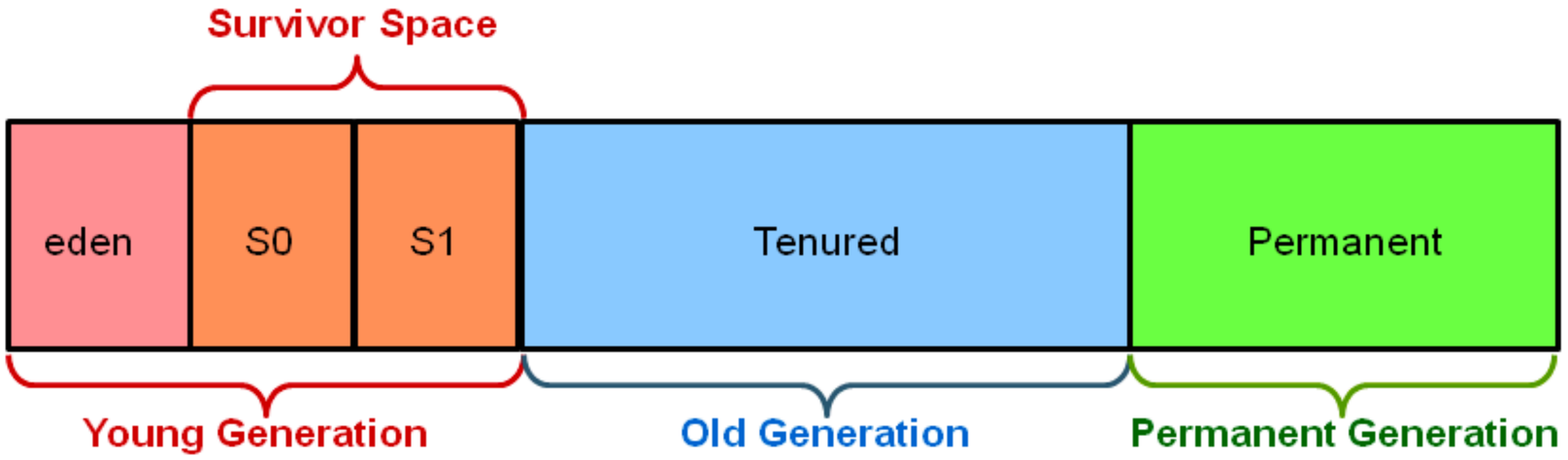


Coletor de Lixo

- Na prática, como destacado, esse processo é muito lento e ineficiente
  - Especialmente para um número muito grande de objetos
- Análises empíricas mostram que a maioria dos objetos são de vida curta
  - Esse conhecimento permitiu melhorar a JVM, e por consequência, o processo de GC
- A memória heap é dividida em três partes pela JVM
  - Young, Old e Permanent Generation
  - JVM Generations

Coletor de Lixo

- Objetos são mantidos inicialmente no eden
  - Minor GC é chamado quando eden é preenchido
  - São realocados em S0 e S1
- Objetos envelhecem
  - Após certo limiar de tempo, são transferidos para Ternured



Coletor de Lixo

- Mais detalhes:
  - http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html

unesp

Static

Palavra-chave *static*

- É utilizada para criar variáveis e métodos que **pertencem à classe** em si, e **não a uma instância específica dessa classe**
- Isso significa que:
  - variáveis e métodos estáticos são compartilhados entre todas as instâncias da classe
  - podem ser acessados sem a necessidade de criar um objeto da classe

unesp

Palavra-chave *static*

- **Variáveis estáticas:**
  - Uma variável estática é declarada com a palavra-chave *static* e são compartilhadas entre todas as instâncias de uma classe e podem ser acessadas diretamente através do nome da classe
  - Por exemplo, podemos declarar uma variável estática para contar o número de objetos criados de uma classe

```
public class Exemplo {
    private static int contador = 0;
    public Exemplo() {
        contador++;
    }
    public static int getContador() {
        return contador;
    }
}
```

Palavra-chave *static*

```
public class Util {
    public static double raizQuadrada(double n) {
        return Math.sqrt(n);
    }
}
```

- **Métodos estáticos:**
  - é um método que é declarado com a palavra-chave *static*
  - Métodos estáticos pertencem à classe em si e não a uma instância específica dessa classe
  - Ou seja, pode ser chamado diretamente através do nome da classe, sem a necessidade de criar um objeto da classe.
  - Podem ser úteis para criar funções de utilidade que não dependem do estado de uma instância específica da classe.
  - Por exemplo, podemos declarar um método estático para calcular a raiz quadrada de um número

unesp

Java *vs* C++

Java *vs* C++

- Java não tem ponteiros explícitos
  - Alocação e desalocação de memória é feita automaticamente
  - Não há nenhum tipo de manipulação de memória
    - Não há métodos para calcular o tamanho dos tipos de dados (primitivos ou objetos)
  - O argumento para isso é que ponteiros são uma fonte de pontenciais *bugs*
    - Eliminação de ponteiros garante mais estabilidade e simplicidade para a linguagem
  - Em C++, há método construtor e destrutor
    - Java não tem destrutor

unesp

Java *vs* C++

- Em Java não há variáveis globais
  - Ao contrário, há variáveis de classe (static)
- Java não suporta herança múltipla
  - Com o argumento de que não é necessário
  - Quando necessário, utilizar interfaces
  - Não há interface em C++
    - Herança múltipla / Classe abstrata
- Java não suporta sobrecarga de operadores
  - Isso permitiria extensões da sintaxe, o que não seria bom

unesp

Java *vs* C++

- Em Java não há cabeçalhos (.h)
  - Toda definição precisa estar dentro da classe
- Objetos em Java são criados sempre através do operador **new**
- Em Java, objetos são inicializados com **null**
- Em Java, para todo acesso a arrays há uma checagem de violação de bordas
- Em Java, toda classe é implicitamente derivada da classe Object
- …

unesp

Java *vs* C++

- Algumas similaridades entre Java e C++
  - Comentários
  - Estruturas de decisão e repetição
  - Tratamento de exceções

unesp

Resumo

- Tecnologia Java
- Variáveis
- Comandos e Comentários
- Coletor de Lixo
- Entrada e saída padrão
- Java *vs* C++

unesp

Hello World

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

unesp

Pelo terminal

- Compilando: javac **HelloWorldApp.java**
- Executando: java **HelloWorldApp**

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

unesp

Pelo terminal

- Pelo **NetBeans**: criar um novo projeto -> Java with Maven -> Java Application
- Faça a leitura do nome utilizando system.in e imprima

System.out.println(“Olá [nome]!”);

unesp

Multiplicação de Matrizes

- Faça um código que leia duas matrizes e calcule a matriz resultante da multiplicação da primeira pela segunda, caso seja possível
- Você pode pedir que o usuário informe o tamanho das matrizes inicialmente
- No final, imprima na tela a expressão calculada, no formato do exemplo abaixo:

```
2 8 1       9 3         39 34
7 5 3   *   2 2    =    88 67
            5 12
```

Operador *instanceof*

- Execute o código abaixo e responda:

Pode substituir: Integer.valueOf(123);

```
public class Exercício {
   public static void main(String[] args) {
      Integer intObj = new Integer(123);
      Long longObj = new Long(1234567890);
      Double doubleObj = new Double(12.345);
      Boolean boolObj = new Boolean(true);
      Object[] objArray = {intObj, longObj, doubleObj, boolObj};

      for (int i = 0; i < objArray.length; i++) {
         if (objArray[i] instanceof Number) {
            System.out.println(objArray[i].toString() + " é um Number.");
         } else {
            System.out.println(objArray[i].toString() + " NÃO é um Number.");
         }
      }
   }
}
```

unesp

Operador *instanceof*

- Execute o código abaixo e responda:
  - Qual a hierarquia das classes utilizadas no código?
    - Number, Integer, Long, Double, Boolean
  - Vá na documentação online da API do Java e verifique a relação entre essas classes
    - https://docs.oracle.com/javase/8/docs/api/
  - A classe **Object** é parte da hierarquia destas classes? Por quê?
  - Baseado nisto, um teste de *instanceof* da classe Object retornaria sempre true?

unesp

- A hierarquia das classes Java segue a seguinte ordem:
  - Object: é a classe raiz da hierarquia de classes Java e todas as outras classes herdam dela.
  - Number: é a classe pai das classes que representam valores numéricos, como Integer, Long e Double.
  - Integer: representa um valor inteiro.
  - Long: representa um valor inteiro maior do que o Integer.
  - Double: representa um valor decimal de ponto flutuante.
  - Boolean: representa um valor booleano (verdadeiro ou falso).
- Vale ressaltar que existem muitas outras classes na hierarquia de classes Java que não foram mencionadas nesta lista, incluindo classes para lidar com strings, datas, coleções, entre outras.

Bibliografia

- DEITEL, H. M. & DEITEL, P.J. "Java : como programar", Bookman, 2017.
- Material baseado nos slides:
  - Luiz E. Virgilio da Silva. Notas de Aula de Programação Orientada a Objetos (ICMC/USP).