SCC0504 – Programação Orientada a Objetos

Introdução ao Java

Luiz Eduardo Virgilio da Silva
ICMC, USP

Material em parte baseado nos slides dos professores:
Clever G. Farias (FFCLRP/USP)

Sumário

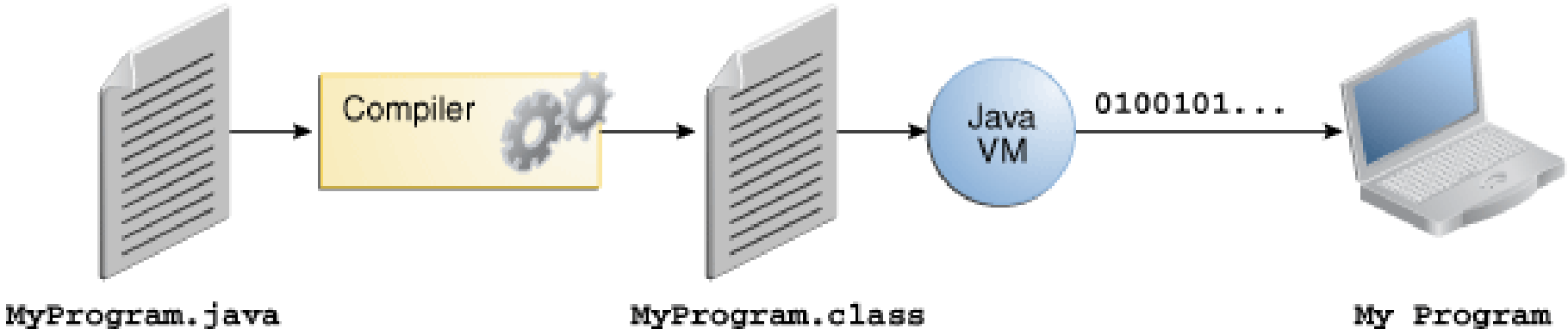
- Tecnologia Java
- Variáveis
- Comandos e Comentários
- Coletor de Lixo
- Entrada e saída padrão
- Java *vs* C++

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

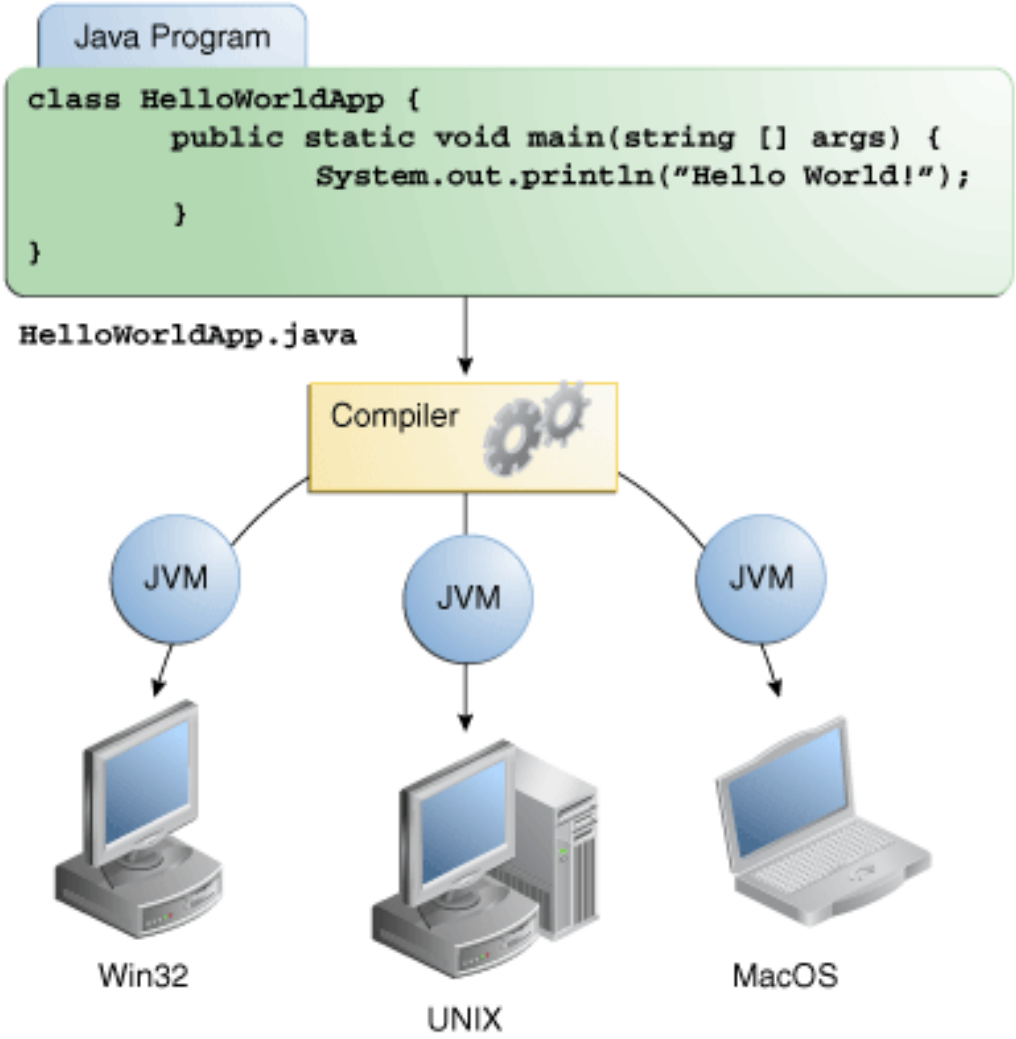
Tecnologia Java

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



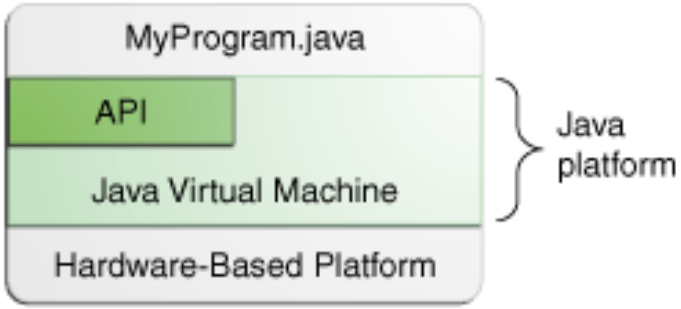
Tecnologia Java

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



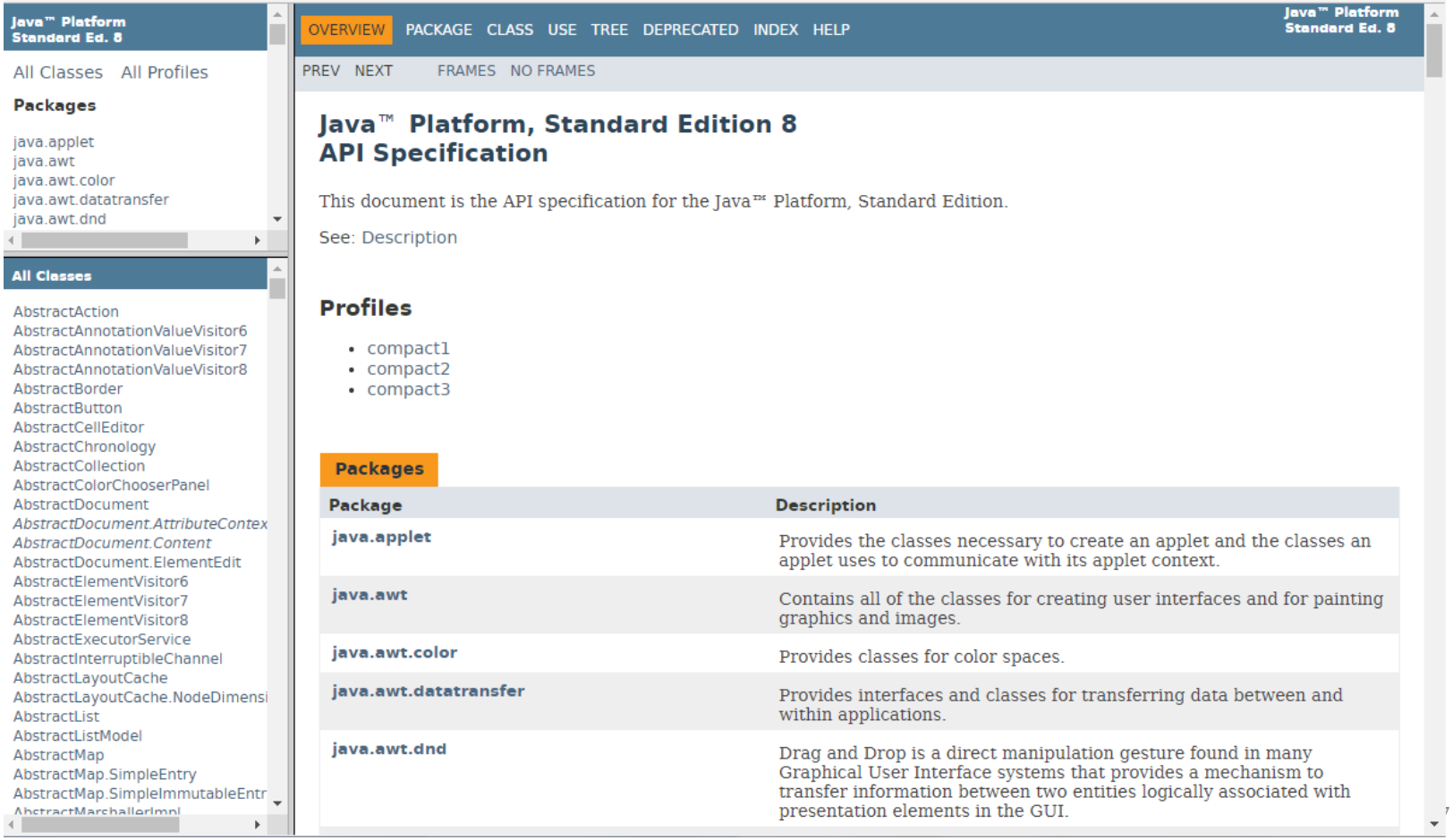
Tecnologia Java

- 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



Tecnologia Java

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



Tecnologia Java

- Apostas dos criadores
  - **Fácil de aprender**: principalmente para quem está familiarizado com C/C++
  - **Menos código**: algumas comparações mostram que um programa em Java pode ser até quatro vezes menor que o mesmo programa escrito em C++
  - **Códigos mais bem escritos**: linguagem encoraja boas práticas de programação
    - API, Garbage Collector, não há ponteiros explícitos, ...
  - **Desenvolvimento mais rápido**
  - ***Write once, run enywhere***: bytecodes (.class)
  - **Distribua mais facilmente**: com o Java Web Start é possível executar programas pelo browser

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 em Java

- Variáveis (ou campos)
  - **Variáveis de instância**: únicas para cada instância

    ```
    int currentSpeed;
    ```

  - **Variáveis de classe**: uma única variável compartilhada com todas as instâncias
    - Modificador *static*
    - Ex: número de marchas de uma Bicicleta

      ```
      static gears = 6;
      ```

    - Poderíamos ainda adicionar o modificador *final,* caso não queiramos que seu valor seja alterado

      ```
      static final GEARS = 6;
      ```

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)
    ```

Nomeando Variáveis

- Java é case-sensitive
- 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: cadence, speed, gear são mais intuitivos do que c, s, g

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: currentSpeed, myBestRate, ...
- No caso de constantes, capitalizar todas as letras e separar as palavras pelo underscore
  - Ex: static final NUM_GEARS = 6;

Nomeando Variáveis

- Exemplos

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

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)

Tipos Primitivos

- Além destes 8 tipos primitivos, a classe `String` recebe um suporte especial do Java, tornando-a parecida com um tipo primitivo.
  - Ex: String str = “This is a string example”;
  - Strings em Java são imutáveis

- Usa-se aspas duplas para Strings (“ ”) e aspas simples para char (‘ ’)

- Valor **null** pode ser atribuido a objetos mas não a tipos primitivos.

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;
      ```

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 numéricos (geral)
  - O underscore não pode ser colocado:
    - No começo ou final do número
    - Adjacente a um ponto decimal
    - Logo antes do sufixo F ou L
  - Exemplos inválidos

```
float pi1 = 3_.1415F;
float pi2 = 3._1415F;
long socialSecurityNumber = 999_99_9999_L;
int x2 = 52_;
int x4 = 0_x52;
int x5 = 0x_52;
int x7 = 0x52_;
```

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).

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];
```

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

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} };
```

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);
```

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

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
   }
}
```

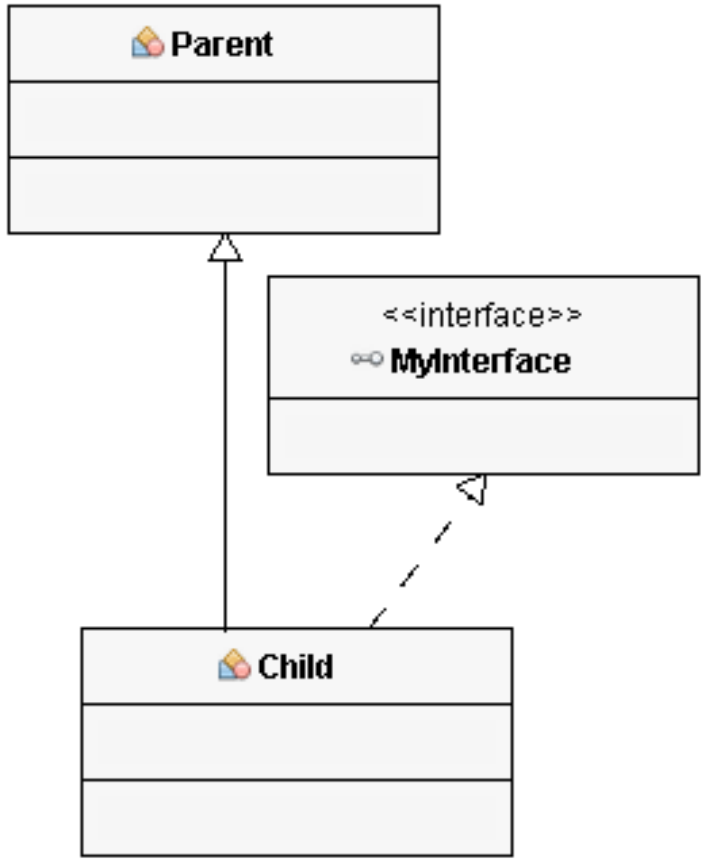
Operadores

- 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

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

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

Controles de Fluxo

if-then

```
void applyBrakes() {
    if (isMoving) {
        currentSpeed--;
    }
}
```

if-then-else

grade = 'C'

```
int testscore = 76;
char grade;

if (testscore >= 90) {
   grade = 'A';
} else if (testscore >= 80) {
   grade = 'B';
} else if (testscore >= 70) {
   grade = 'C';
} else if (testscore >= 60) {
   grade = 'D';
} else {
   grade = 'F';
}
```

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 conc;

nota > 50 ? conc = 'A' :
            conc = 'R';
```

```
conc = 'A'
```

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++;
}
```

```
Count is: 1
Count is: 2
Count is: 3
Count is: 4
Count is: 5
Count is: 6
Count is: 7
Count is: 8
Count is: 9
Count is: 10
```

do-while

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

Controles de Fluxo

for

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

```
Count is: 1
Count is: 2
Count is: 3
Count is: 4
Count is: 5
Count is: 6
Count is: 7
Count is: 8
Count is: 9
Count is: 10
```

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

Controle 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 < arrayOfInts.length; i++) {
    if (arrayOfInts[i] == searchfor) {
        foundIt = true;
        break;
    }
}
```

Controle de Fluxo

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

```
for (int i = 0; i < max; i++) {
    // interested only in p's
    if (searchMe.charAt(i) != 'p')
        continue;

    // process p's
    numPs++;
}
```

Controle 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

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

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

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();
```

Entrada e Saída Padrão

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

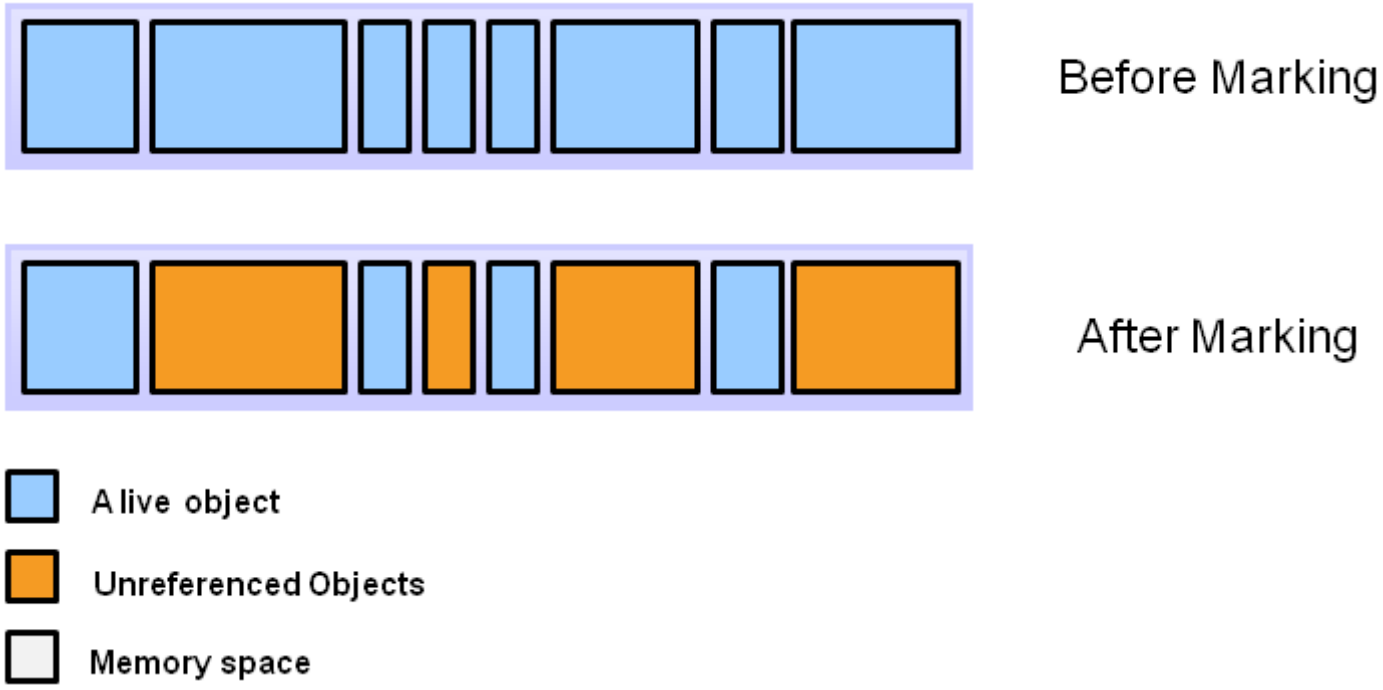
```
System.out.println(“Olá!”)
```

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*

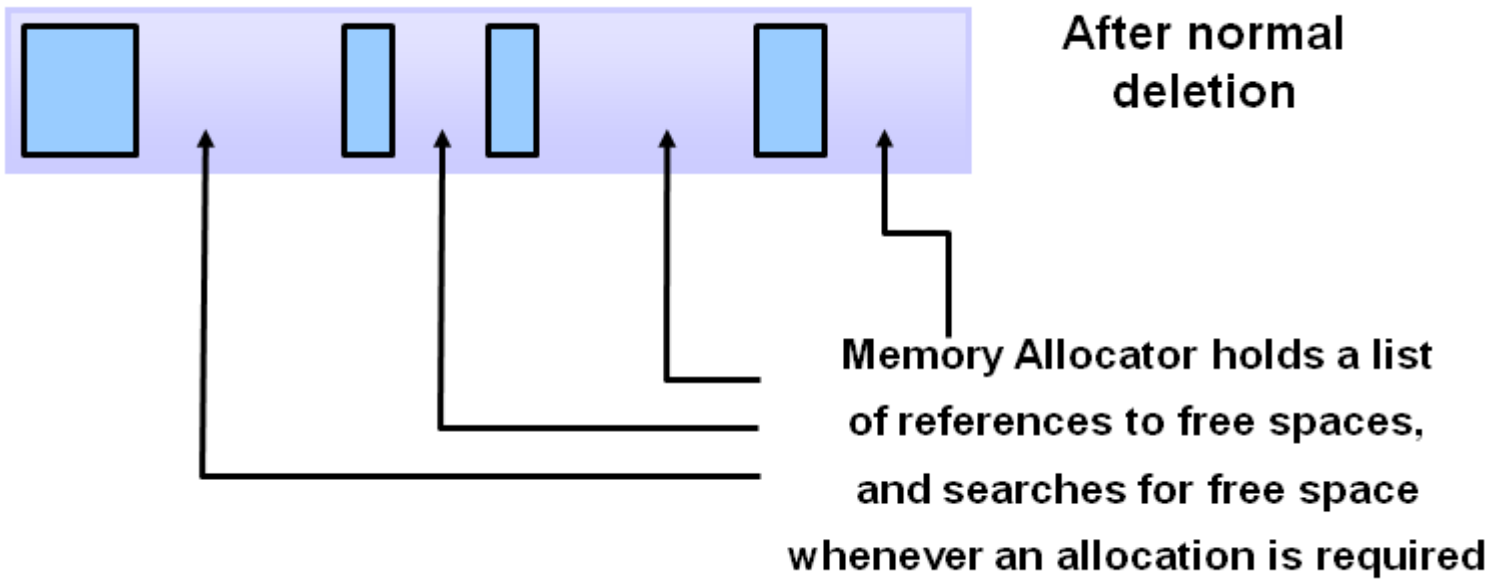
Coletor de Lixo

- 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



Coletor de Lixo

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

After normal Deletion with compacting

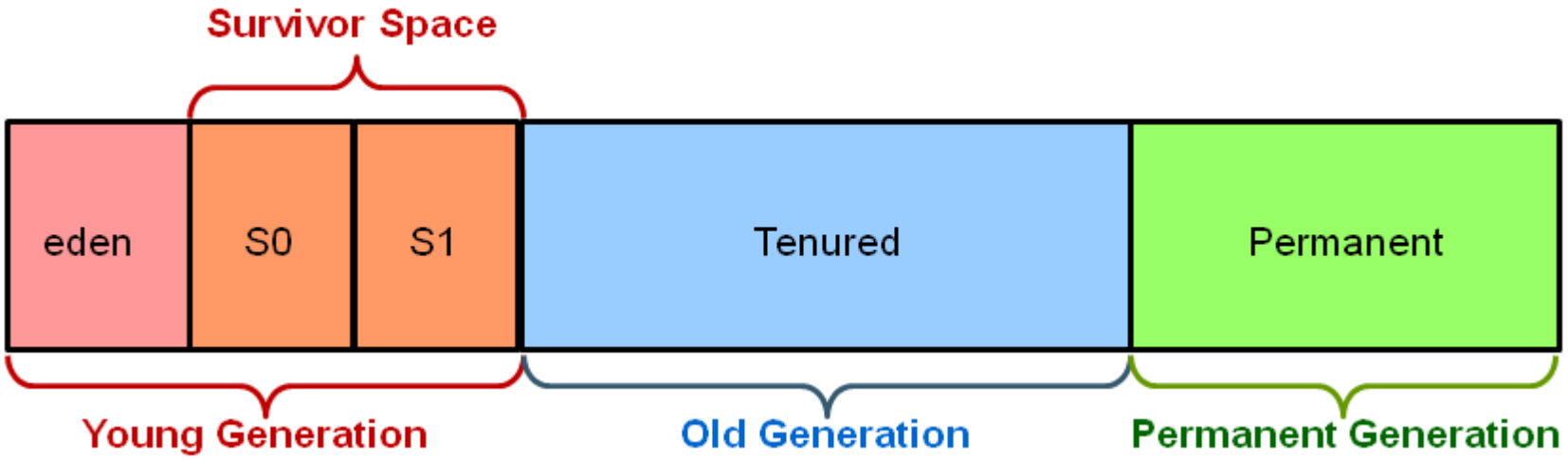
Memory Allocator holds the reference to the beginning of free space, and allocated memory sequentially then on.

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

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

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

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
- ...

Java *vs* C++

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

Resumo

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

Hello World (próxima aula)

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

Dúvidas


VIDA DE PROGRAMADOR .COM.BR
real historia;
string sender;
sender = "Andre Mendes";
#1242
QUE LEGAL, VOCÊ ENTENDE DE COMPUTADOR?
SIM, UM POUCO...
O ESSENCIAL É INVISÍVEL AOS OLHOS
ENTÃO ME EXPLICA UMA COISA...
DIGA...
O QUE É MELHOR: WINDOWS OU WI-FI?
PLOFT!