SCC0504 - Programação Orientada a Objetos

Dados: tipos, conversão, leitura, empacotadores. Static e Final. Interação de Objetos

Prof.: Leonardo Tórtoro Pereira
leonardop@usp.br

Agradecimentos ao material do prof. João do E.S. Batista Neto [1]

Tipos de Dados em Java

Category	Types	Size (bits)	Minimum Value	Maximum Value	Precision	Example
Integer	`byte`	8	-128	127	From +127 to -128	`byte b = 65;`
	`char`	16	0	$2^{16}-1$	All Unicode characters[1]	`char c = 'A';` `char c = 65;`
	`short`	16	-2^{15}	$2^{15}-1$	From +32,767 to -32,768	`short s = 65;`
	`int`	32	-2^{31}	$2^{31}-1$	From +2,147,483,647 to -2,147,483,648	`int i = 65;`
	`long`	64	-2^{63}	$2^{63}-1$	From +9,223,372,036,854,775,807 to -9,223,372,036,854,775,808	`long l = 65L;`
Floating-point	`float`	32	2^{-149}	$(2-2^{-23})\cdot 2^{127}$	From 3.402,823,5 E+38 to 1.4 E-45	`float f = 65f;`
	`double`	64	2^{-1074}	$(2-2^{-52})\cdot 2^{1023}$	From 1.797,693,134,862,315,7 E+308 to 4.9 E-324	`double d = 65.55;`
Other	`boolean`	--	--	--	false, true	`boolean b = true;`
	`void`	--	--	--	--	--

Fonte: https://en.wikibooks.org/wiki/Java_Programming/Primitive_Types

Tipos de Dados primitivos

```
byte b = 120;                       // 1 byte
short s = -10000 ;                  // 2 bytes
int i =  34034343;                  //4 bytes
long l = 1281722323120981;          // 8 bytes
float f = 9.45f;                    // 4 byes
double d = 2.343;                   // 8 bytes
char c = 'y';                       // 2 bytes - UNICODE
```

Wrappers

Wrappers

- ➔ Classe que “envolve” coisas
- ➔ Adicionam funcionalidades a classes ou tipos primitivos
- ➔ Existem *wrappers* também para trabalhar com fluxos de áudio e *buffers*, por exemplo

O que você pode fazer com um *int*?

Wrappers

- ➔ Nada, a não ser atribuir valores e ler esse valor...
- ➔ Mas você pode empacotá-lo com a classe *Integer*
- ➔ Permite transformar um *int* em *String* ou uma *String* em *int*
- ➔ Constantes de maior e menor valor possível de um *int*
- ➔ Faz *cast* para outros tipos
- ➔ Operações binárias (rotacionar bytes, ordem reversa, etc.)

O *wrapper Integer*

```
int i;
//Integer iw = new Integer(30); Descontinuado
Integer iw = 30;
i = Integer.parseInt("123");
byte b = iw.byteValue();

System.out.println(" Valor de i = "+i);
System.out.println(" Valor de iw= "+iw);
System.out.println(" Valor byte de iw = "+b);
System.out.println("Reverso de i = "+Integer.reverse(i));
System.out.println("Rotação esq. de i = "+Integer.rotateLeft(i, 1));
System.out.println("Rotação dir. de i = "+Integer.rotateRight(i, 1));
```

Conversão de tipos

Conversão

```
int i, i2;
float f, f2;
i2 = 5;
i = 10;
f = 6.5f;
f2 = 15.0f;
f = i;       // conversao por atribuicao
System.out.println(" Valor de f = "+f);
f = f2 / i2;  // conversao por promocao. i2 eh convertido para float para executar a divisao
System.out.println(" Valor de f = "+f);

// voce duvida? entao veja o exemplo abaixo..
f2 =  10.0f / 3;    // que exemplo de conversao
System.out.println(" Valor de f2 = "+f2);

i = (int) f;     // conversao por casting
System.out.println(" Valor de i = "+i);
```

Leitura e Escrita de Dados - *Scanner* e *System.out*

Escrita de Dados - *System.out*

- Para a impressão de caracteres na saída padrão usamos a classe *System*, como vimos nas aulas anteriores
- *System.out* é um objeto que contém a saída padrão
  - O objeto contém métodos como *print()*, *println()* e *printf()*
  - Pertence à classe *PrintStream*

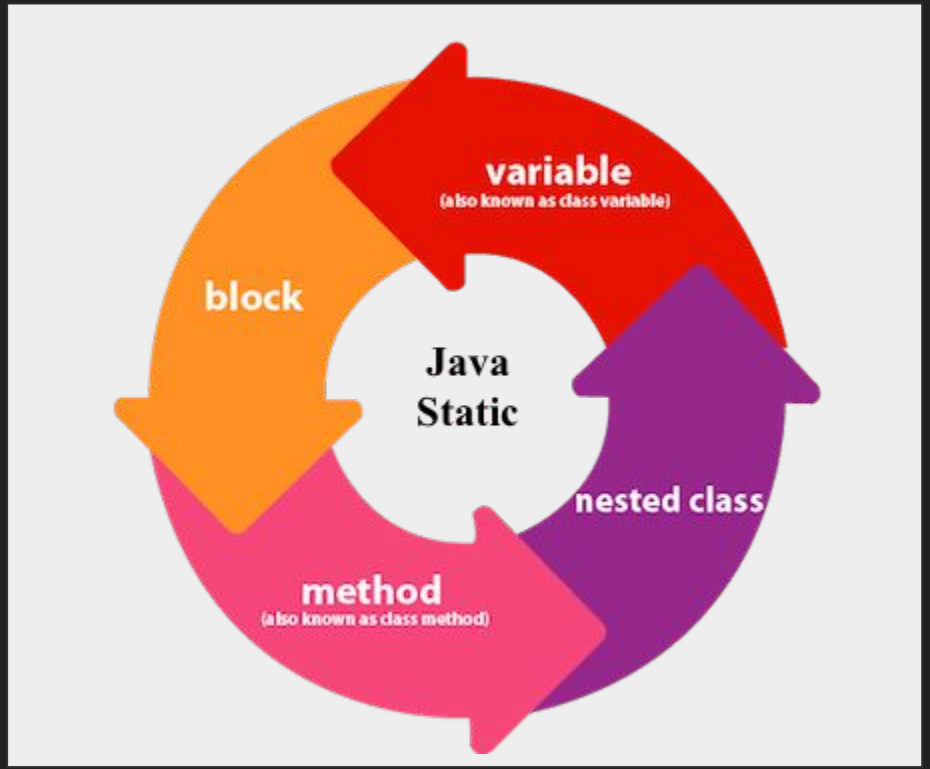
Leitura de Dados - *Scanner*

- Para fazer a leitura de dados da entrada padrão em *Java* usamos uma classe, a *Scanner*
- Essa classe recebe a entrada padrão (*System.in*)
  - Que é um objeto da classe InputStream
- Podemos umas métodos para ler linhas de entrada
  - *nextLine()*
- Ou tipos de dados
  - *nextInt()*

Scanner e System.out

```
String resp;
int i;

Scanner scan = new Scanner(System.in);

System.out.print("Entre com a string = ");
resp = scan.nextLine();

System.out.print("Entre com o inteiro= ");
i = scan.nextInt();

System.out.println(" Valor de resp = "+resp);
System.out.println(" Valor de i = "+i);
```

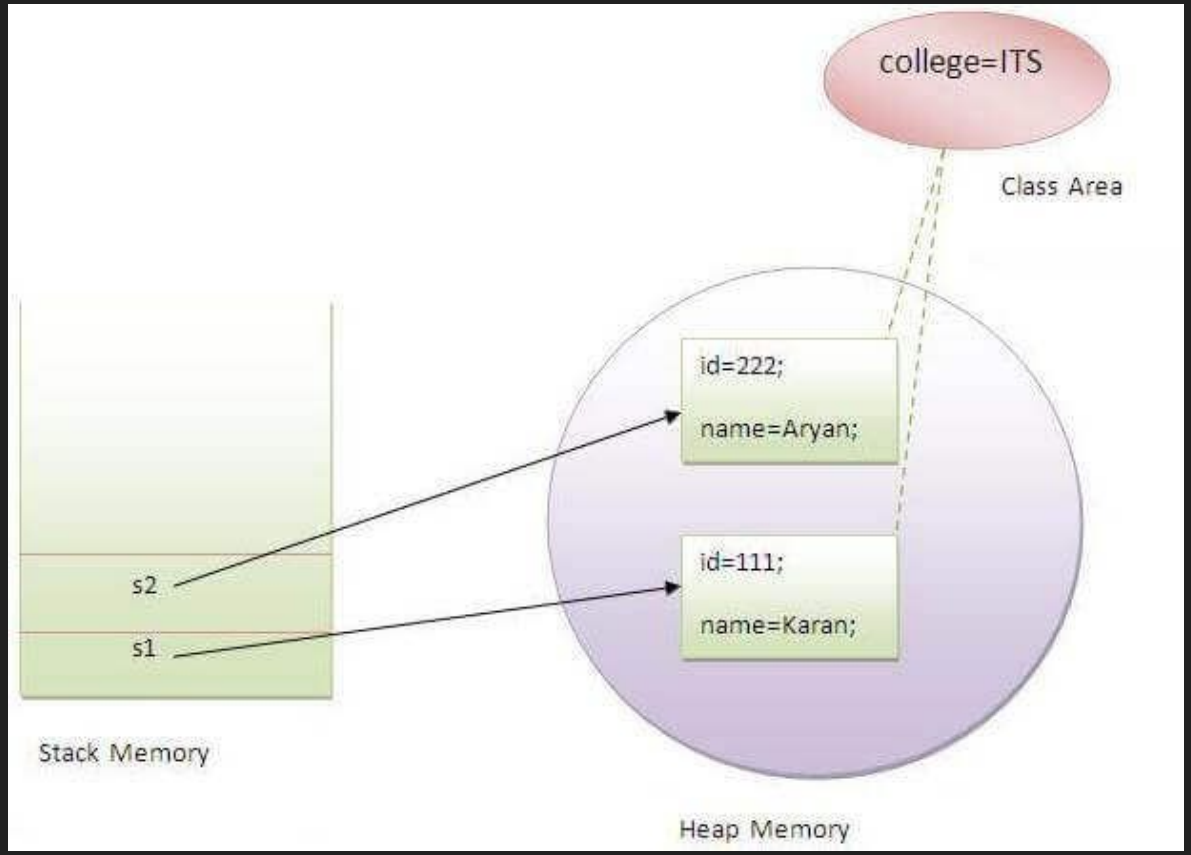
Static



Fonte: https://www.javatpoint.com/static-keyword-in-java

Variável *static*

- Propriedade em comum para TODOS OS OBJETOS de uma classe
- Ex. de uso:
  - Contadores de instâncias
    - Identificador único de objetos
  - Nome da escola para uma *classe* aluno
    - Economiza memória
- Memória na área da classe ao invés da heap



Fonte: https://www.javatpoint.com/static-keyword-in-java

Método *static*

- ➔ Método pertence à classe
- ➔ Pode ser invocado sem precisar criar uma instância da classe (objeto)
- ➔ Pode acessar atributos estáticos da classe e mudar seu valor.
- ➔ Não pode usar atributo não-estático ou chamar um método não-estático diretamente
- ➔ Não pode usar as palavras-chave *this* e *super*

Método *static*

https://www.geeksforgeeks.org/static-keyword-java/

```
class Student {
    String name;
    int rollNo;
    static String cllgName; // static variable
    static int counter = 0; // static counter to set unique roll no
    public Student(String name){
        this.name = name;
        this.rollNo = setRollNo();
    }
    static int setRollNo() { // getting unique rollNo through static variable(counter)
        counter++;
        return counter;
    }
    static void setCllg(String name){ // static method
        cllgName = name ;
    }
    void getStudentInfo(){ // instance method
        System.out.println("name : " + this.name);
        System.out.println("rollNo : " + this.rollNo);
        System.out.println("cllgName : " + cllgName); // accessing static variable
    }
}
```

Bloco *static*

https://www.geeksforgeeks.org/static-keyword-java/

- Usado para inicializar o membro *static*
- Chamada antes do método *main*

```
// Static member can be accessed before instantiating a class
class Test {
    // static method
    static void m1() {
        System.out.println("from m1");
    }
    public static void main(String[] args) {
          // calling m1 without creating
          // any object of class Test
           m1();
    }
}
```

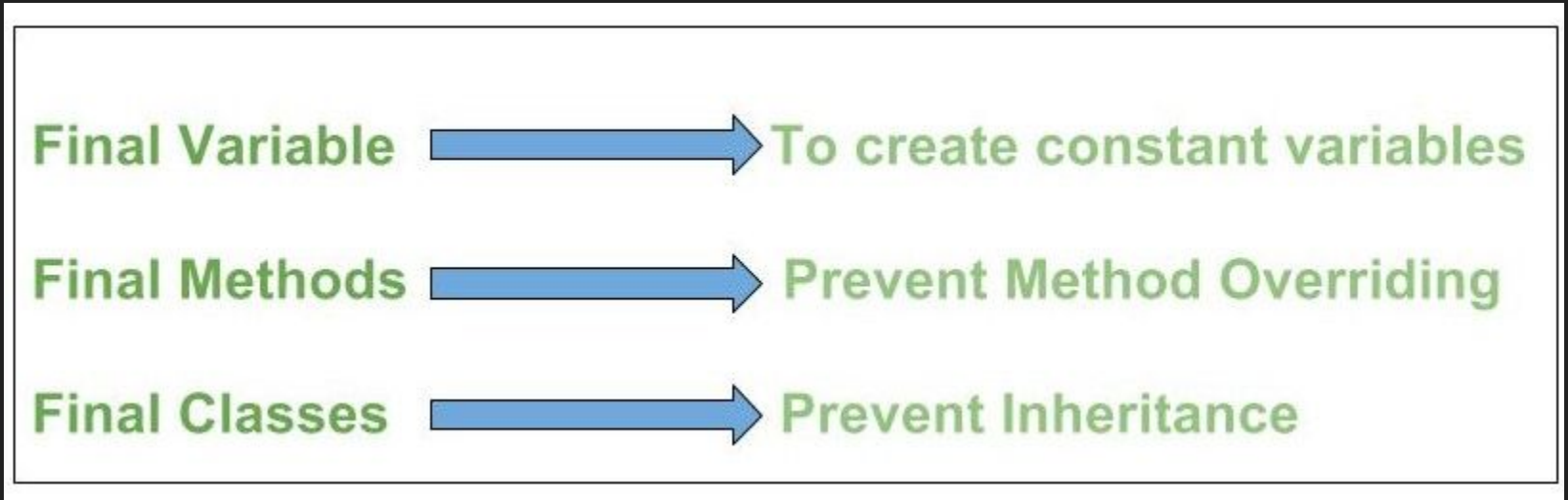
Classes Aninhadas *static*

- É possível aninhar classes usando a palavra-chave *static*
- Nós veremos isso em outra aula, junto com os conceitos e propriedades de classes aninhadas :)

Final

Final

- Assim como *static*, pode ser usada para mais de um contexto



Fonte: https://www.geeksforgeeks.org/final-keyword-java/

Variável *final*

- Em uma variável, cria uma constante
  - Valor não pode ser modificado. Só inicializado
  - Deve ser inicializado no construtor ou na declaração
    - Pode ser inicializada em blocos *static* ou só blocos de inicialização
  - Nome todo em MAIUSCULAS

Método e Classe *final*

➔ Veremos com mais detalhes os efeitos da palavra *final* em métodos e classes nas próximas aulas.

Variáveis *final*

https://www.geeksforgeeks.org/final-keyword-java/

```java
final int THRESHOLD = 5; // a final variable direct initialize
final int CAPACITY; // a blank final variable
final int  MINIMUM; // another blank final variable
static final double PI = 3.141592653589793; // a final static variable PI direct initialize
static final double EULERCONSTANT; // a  blank final static  variable

// instance initializer block for initializing CAPACITY
{
    CAPACITY = 25;
}
// static initializer block for initializing EULERCONSTANT
static{
    EULERCONSTANT = 2.3;
}
// constructor for initializing MINIMUM Note that if there are more than one
// constructor, you must initialize MINIMUM in them also
public GFG() {
    MINIMUM = -1;
}
```

Exemplo de Interação entre Objetos

Referências

[1] http://www.lcad.icmc.usp.br/~jbatista/sce537/

- Alguns exemplos das aulas 3, 4 e 5.

[2] https://www.javatpoint.com/static-keyword-in-java

[3] https://www.geeksforgeeks.org/static-keyword-java/

[4] https://www.geeksforgeeks.org/final-keyword-java/

[5] https://docs.oracle.com/javase/tutorial/java/javaOO/objects.html