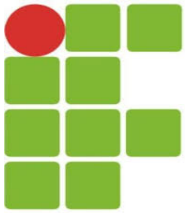


Instituto Federal

Campus Goiânia

Bacharelado em Sistemas de Informação

POO II

Prof. Dory Gonzaga Rodrigues

Ementa

- Alocação de Memória

- Garbage Collector

- Boxing, Unboxing e Wrapper Classes

- Vetor

- Matriz

Alocação de Memória em Java

A memória da Máquina Virtual do Java – JVM é dividia em 4 seções:

1. Code: Contém o bytecode do programa.

2. Stack: Contém métodos, variáveis de referência e variáveis locais.

   É sempre referenciada na ordem LIFO.

   As variáveis locais (são variáveis criadas dentro de um método incluindo os argumentos do método - parâmetros)

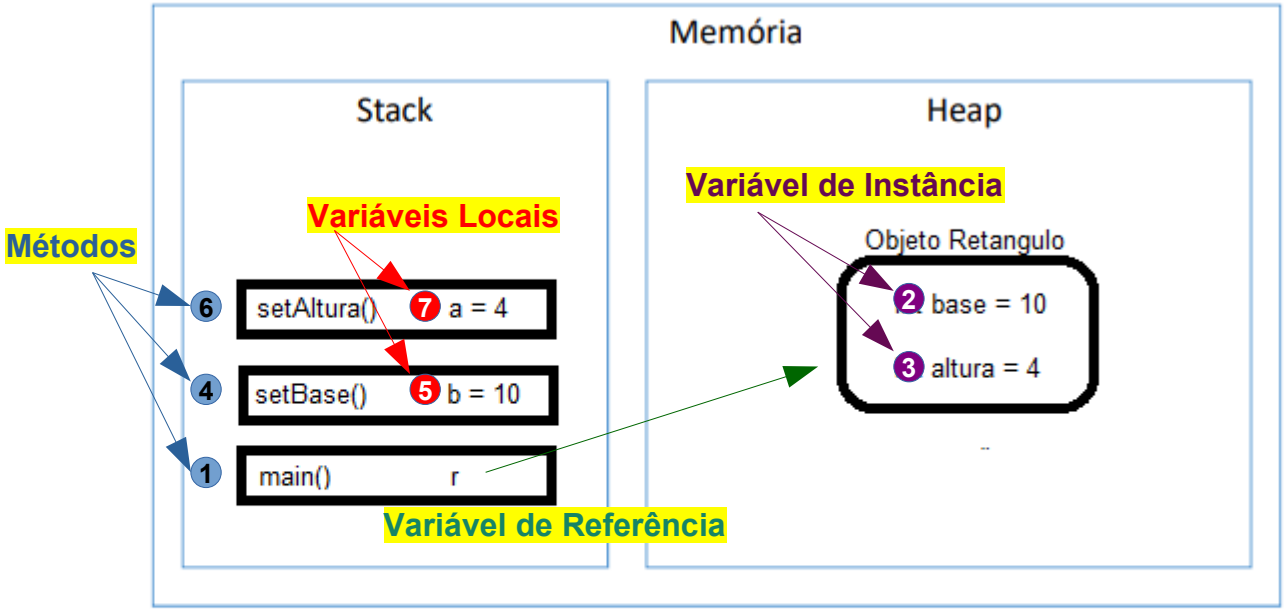
3. Heap: Contém as variáveis de instância (as propriedades dos Objetos, podem conter variáveis de referência (propriedade cujo o tipo é uma classe)

4. Static: Contém dados estáticos e métodos estáticos

Alocação de Memória em Java

Veja como funciona a alocação de memória:

```
1  package stackHeap;
2
3  public class Retangulo {
4
5      private double base;
6      private double altura;
7
8      public double getBase() {
9          return base;
10     }
11
12     public void setBase(double b) {
13         this.base = b;
14     }
15
16     public double getAltura() {
17         return altura;
18     }
19
20     public void setAltura(double a) {
21         this.altura = a;
22     }
23
24     public static void main(String[] args) {
25         Retangulo r = new Retangulo();
26         r.setBase(10);
27         r.setAltura(4);
28     }
29 }
```



Alocação de Memória em Java

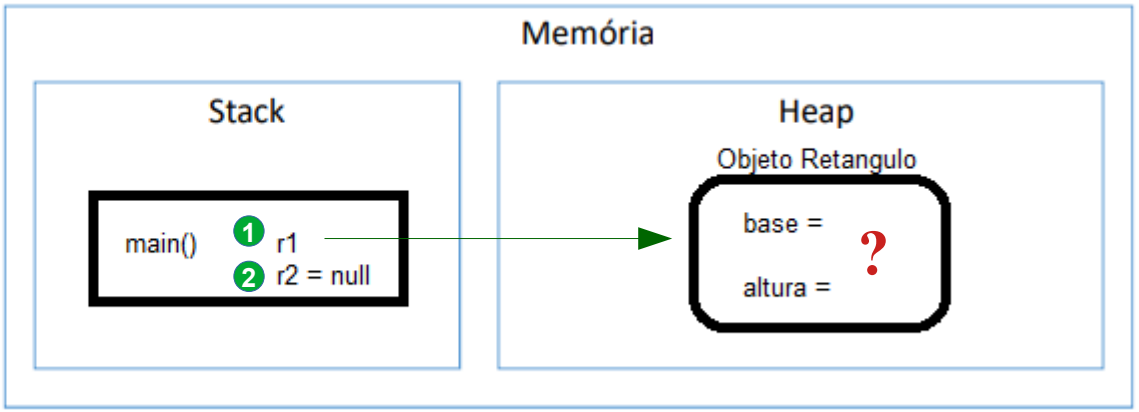
Variáveis de Referência (Ponteiro)

- Em Java a variável de Referência é um ponteiro para o objeto !

- Variáveis deste tipo são ponteiros localizados na Stack que apontam para as variáveis de instância (propriedades da classe) localizadas na memória Heap

- Variáveis de referência podem ser criadas sem uma instância da classe, ou seja, seu valor inicial é “null”.

```
1  package stackHeap;
2
3  public class Aplicacao {
4
5      public static void main(String[] args) {
6
7          Retangulo r1, r2;
8
9      1   r1 = new Retangulo();
10
11     2   r2 = null;
12
13     }
14
15 }
```



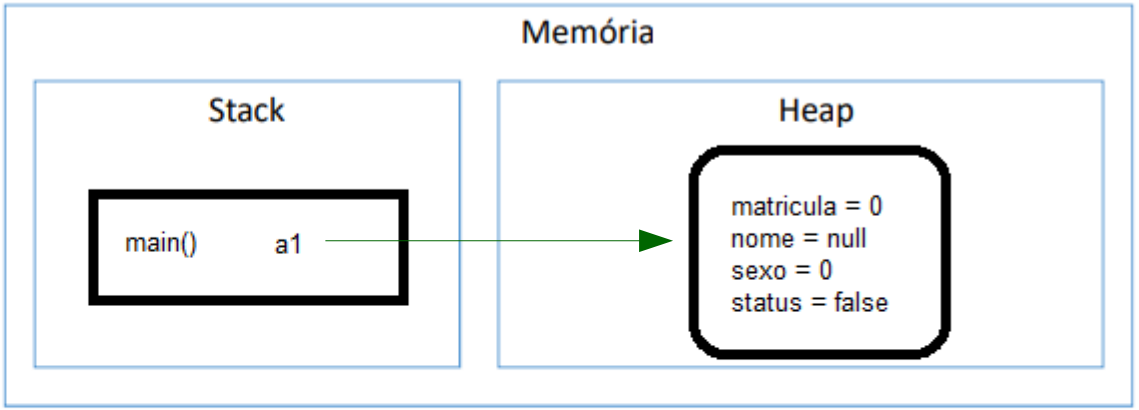
Alocação de Memória em Java

Variáveis de Referência (Ponteiro)

- Ao instanciar um objeto (comando “new”), cada propriedade do objeto que não receber valor, o JVM atribui um valor padrão de acordo com o tipo da variável.

```java
package stackHeap;

public class Aluno {

    private int matricula;
    private String nome;
    private char sexo;
    private boolean status;
    ...
}
```

```java
package stackHeap;

public class Aplicacao {

    public static void main(String[] args) {

        Aluno a1 = new Aluno();

    }

}
```



Alocação de Memória em Java

Variáveis de Tipo Primitivo

- As variáveis de tipo primitivo são aquelas que recebem um valor !

- As variáveis locais são do tipo primitivo, já as variáveis de instância podem ser do tipo primitivo ou referência;

- O JVM exige que as variáveis locais, de tipo primitivo, sejam inicializadas !

- Quais são os tipos primitivos ?

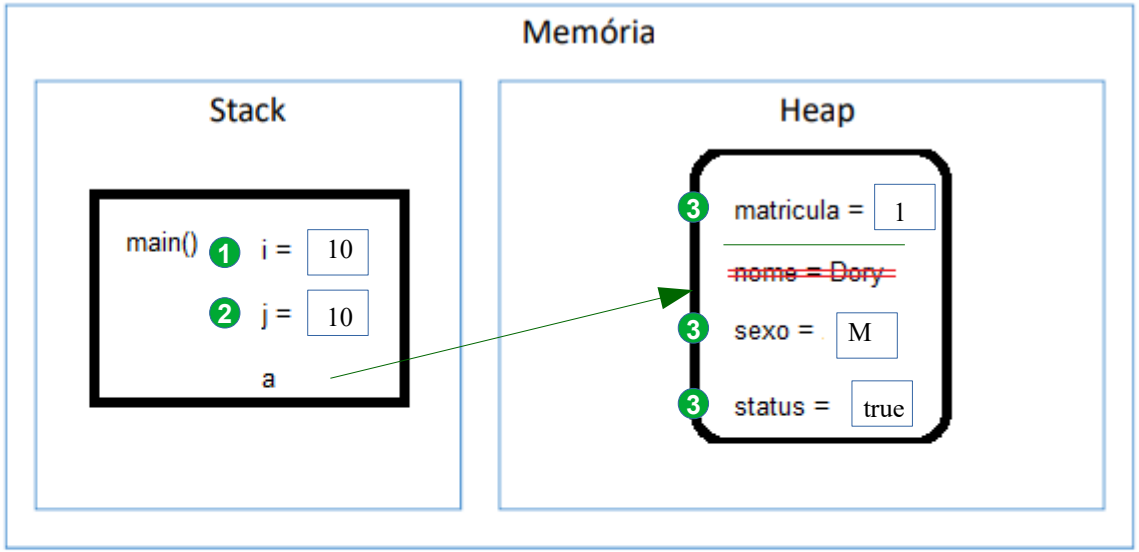
boolean	char	byte	float
		short	double
		int	
		long	

Alocação de Memória em Java

Variáveis de Tipo Primitivo

```java
package stackHeap;

public class Aluno {

    private int matricula;
    private String nome;
    private char sexo;
    private boolean status;
    ...
}
```

```java
package stackHeap;

public class Aplicacao2 {

    public static void main(String[] args) {

        double i, j;

    1   i = 10;

    2   j = i;

        Aluno a;

    3   a = new Aluno(1, "Dory", 'M', true);

    }

}
```



Garbage collector

- A linguagem Java possui gerenciamento automático de memória, controlando sua alocação e desalocação. A desalocação de memória é suportada pelo processo conhecido por Garbage Collector.

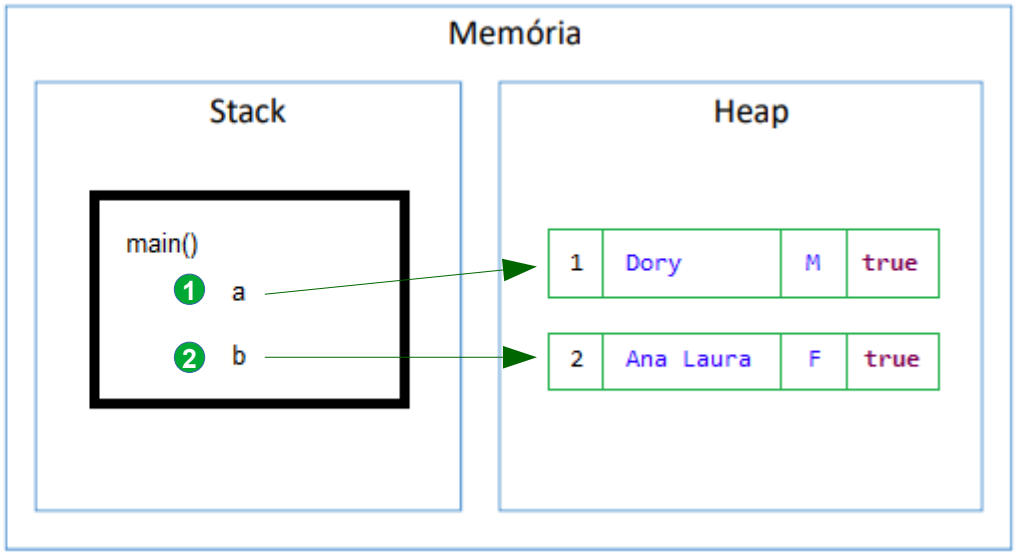
- O controle do Coletor de Lixo é feito pela JVM, que é quem decide quando ele será executado. Mas não existe nada que garanta quando o Garbage Collector será executado.

- “Lixo” é quando um objeto (variáveis de instância) se torna qualificado para a coleta de lixo, podendo então ser desalocado na próxima execução do GC (Garbage Collector).

Garbage collector

- O objeto “Aluno” referenciado pela variável “a” torna-se lixo quando “a” passar a apontar para outro objeto (b) ou passar a ser “null”.

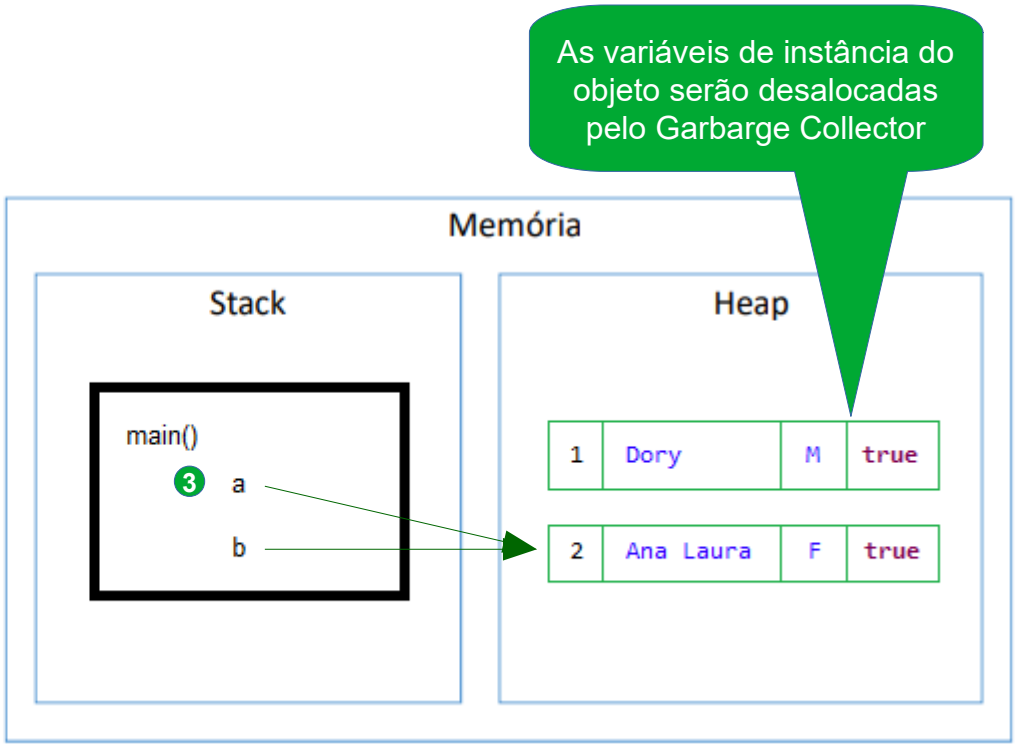
```
1  package stackHeap;
2
3  public class Aplicacao2 {
4
5      public static void main(String[] args) {
6
7          Aluno a, b;
8
9      1   a = new Aluno(1, "Dory", 'M', true);
10
11     2   b = new Aluno(2, "Ana Laura", 'F', true);
12
13         a = b ;
14
15     }
16
17 }
```



Garbage collector

- O objeto “Aluno” referenciado pela variável “a” torna-se lixo quando “a” passar a apontar para outro objeto (b) ou passar a ser “null”.

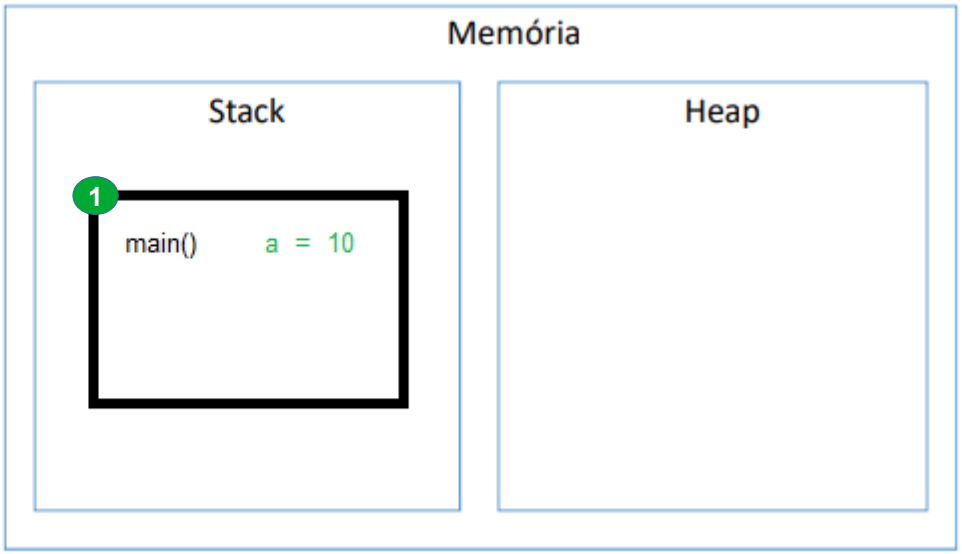
```
1  package stackHeap;
2
3  public class Aplicacao2 {
4
5      public static void main(String[] args) {
6
7          Aluno a, b;
8
9          a = new Aluno(1, "Dory", 'M', true);
10
11         b = new Aluno(2, "Ana Laura", 'F', true);
12
13     3   a = b ;
14
15     }
16
17 }
```



Garbage collector

- As variáveis locais são desalocadas imediatamente após o encerramento do método a qual pertence (escopo).

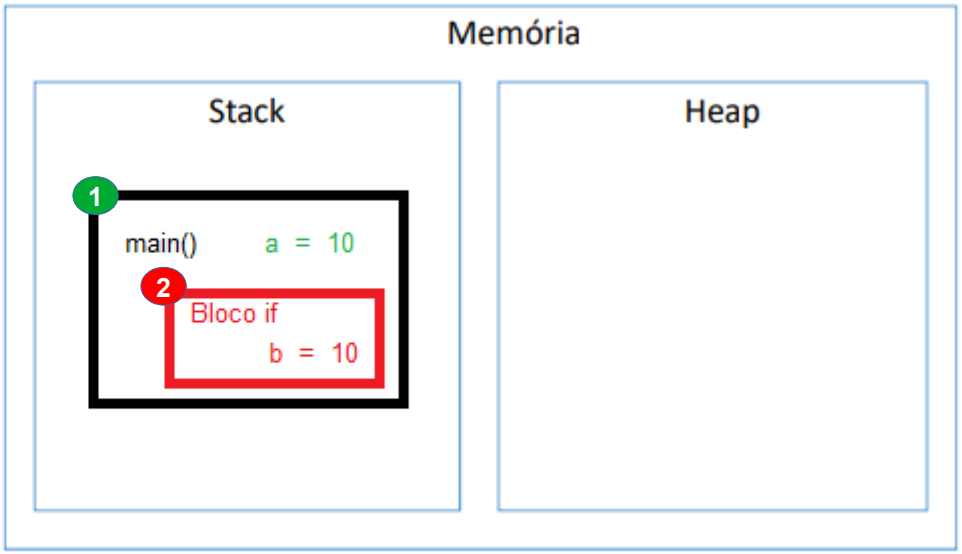
```
package stackHeap;

public class Aplicacao3 {

    public static void main(String[] args) {

        int a;

1       a = 10;

        if (a == 10 ) {

            int b = a ;

            System.out.println("B: " + b);
        }

        System.out.println("A: " + a);
    }

}
```



Garbage collector

- As variáveis locais são desalocadas imediatamente após o encerramento do método a qual pertence (escopo).

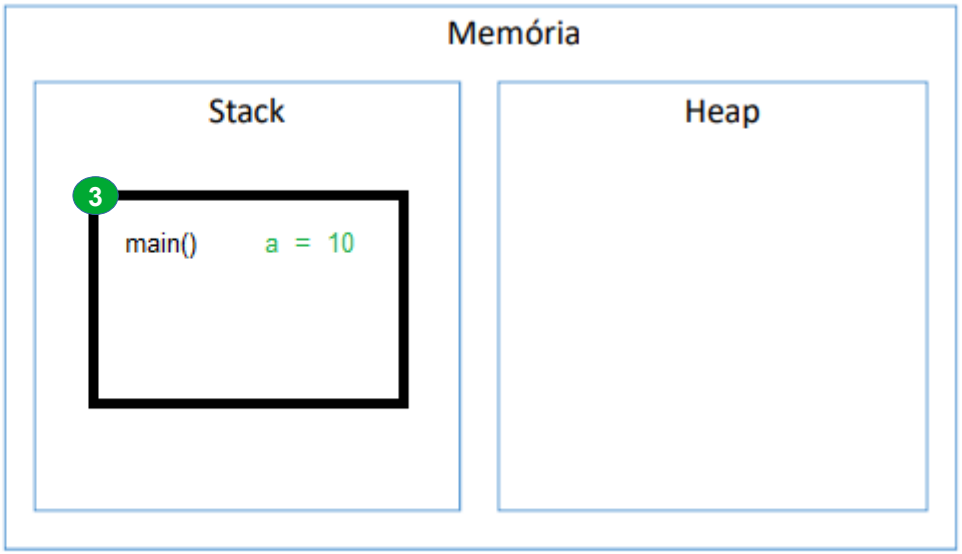
```
package stackHeap;

public class Aplicacao3 {

    public static void main(String[] args) {

        int a;

    1   a = 10;

        if (a == 10 ) {

    2   int b = a ;

            System.out.println("B: " + b);
        }

        System.out.println("A: " + a);
    }

}
```



Garbage collector

- As variáveis locais são desalocadas imediatamente após o encerramento do método a qual pertence (escopo).

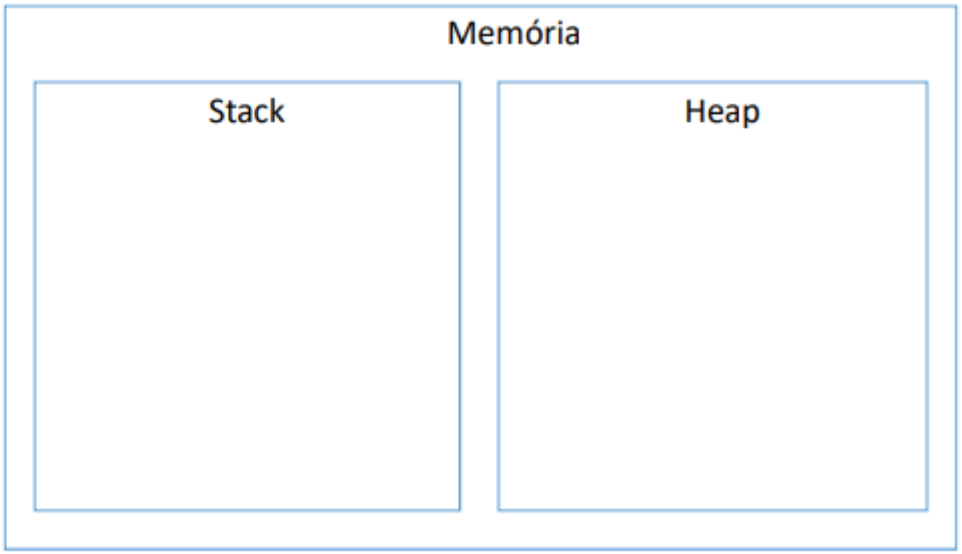
```
package stackHeap;

public class Aplicacao3 {

    public static void main(String[] args) {

        int a;

        a = 10;

        if (a == 10 ) {

            int b = a ;

            System.out.println("B: " + b);
        }

3       System.out.println("A: " + a);
    }

}
```



Garbage collector

- As variáveis locais são desalocadas imediatamente após o encerramento do método a qual pertence (escopo).

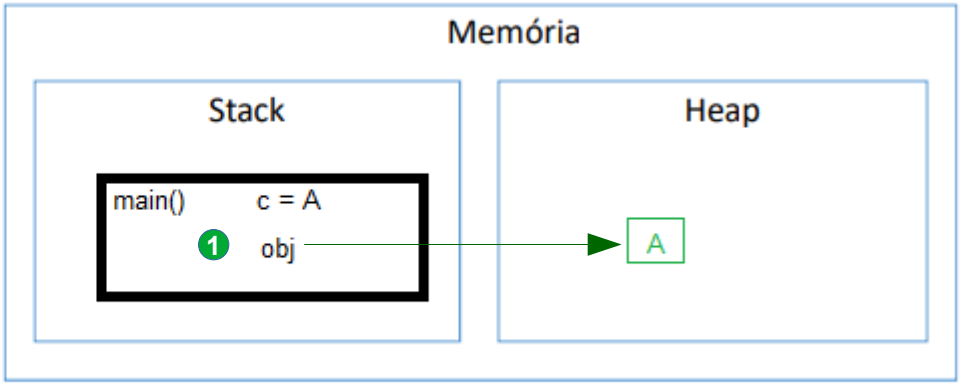
```
package stackHeap;

public class Aplicacao3 {

    public static void main(String[] args) {

        int a;

        a = 10;

        if (a == 10 ) {

            int b = a ;

            System.out.println("B: " + b);
        }

        System.out.println("A: " + a);
    }
}
```



Boxing

- É o processo de passagem do valor contido em uma variável de tipo primitivo para um variável tipo referência (objeto) compatível.

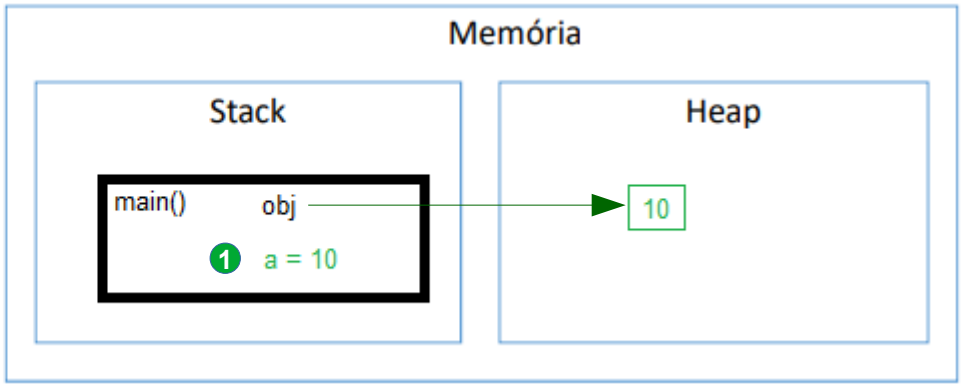
```
1  package stackHeap;
2
3  public class Aplicacao4 {
4
5      public static void main(String[] args) {
6
7          char c = 'A';
8
9      1   Object obj = c;
10
11         System.out.println(c);
12
13     }
14
15 }
```



Unboxing

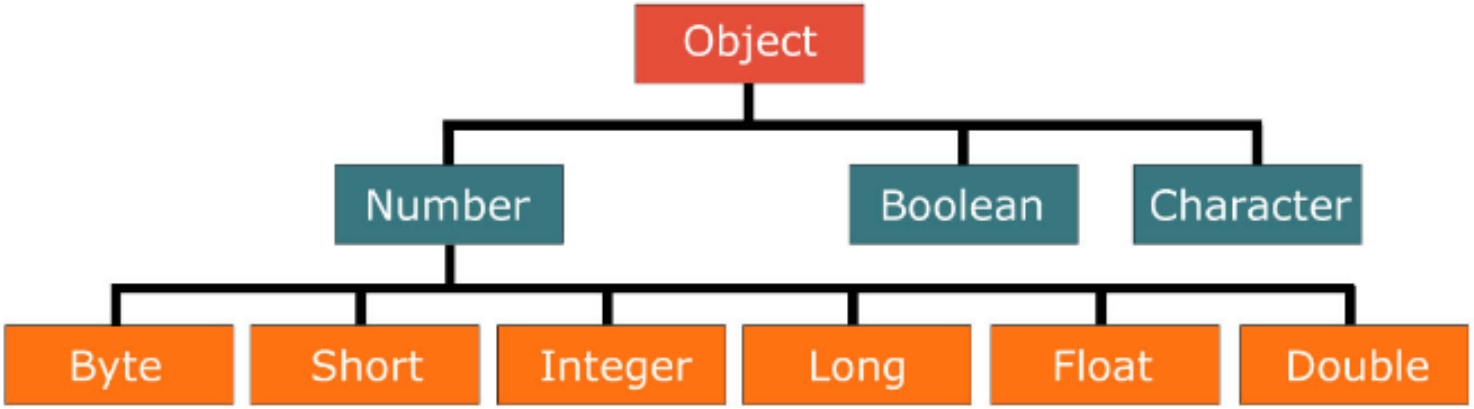
- É o processo de passagem do valor contido em uma variável de referência (objeto) para uma variável de tipo primitivo compatível.

```
1  package stackHeap;
2
3  public class Aplicacao5 {
4
5      public static void main(String[] args) {
6
7          Object obj = 10;
8
9      1   int a = (int) obj;
10
11     }
12
13 }
```



Wrapper classes

- São as classes equivalentes aos tipos primitivos!



- A operação de Boxing e Unboxing é natural na linguagem entre as variáveis do tipo primitivo e a variável tipo referência equivalente.

- Normalmente utilizamos as classes Wrapper como o tipo das variáveis de instância dos objetos que representam uma Entidade do Banco de Dados. A vantagem é que aceita o valor “null” e ainda tem os recursos de OO.

Vetores

- Vetor é uma estrutura de dados unidimensional:

    Homogênea (dados do mesmo tipo)

    Indexada (os elementos são acessados por meio de posições)

    A memória é alocada de uma só vez, em um bloco contíguo da memória.

Vantagem:

    Acesso direto a qualquer elemento, basta informar a posição

Desvantagem:

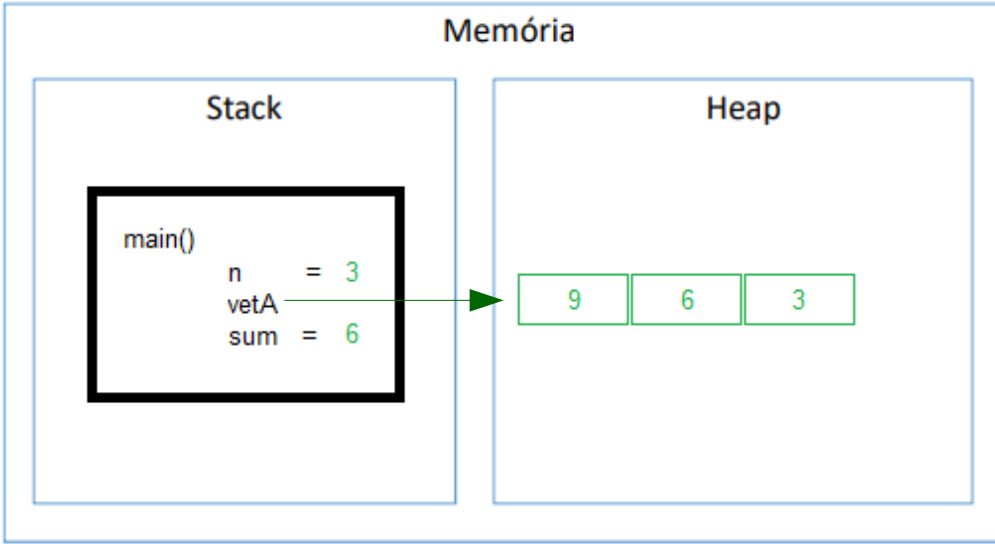
    De tamanho fixo

Vetor

Exemplo

O programa abaixo realiza a leitura de um número inteiro que irá representar a quantidade de pessoas, depois armazena os valores em um vetor do tipo double que representa a altura dessas pessoas. Ao final, calcula e apresenta a média aritmética da altura dessas pessoas.

```
1  package vetor;
2
3  import java.util.Locale;
5
6  public class Exemplo {
7      public static void main(String[] args) {
8
9          Locale.setDefault(Locale.US);
10
11         Scanner  sc = new Scanner(System.in);
12
13         System.out.println("Digite a quantidade de pessoas:");
14         int n = sc.nextInt();
15
16         double[] vetA = new double[n];
17
18         double sum = 0;
19
20         for(int i=0; i < vetA.length; i++) {
21             System.out.println("Digite a altura da pessoa: ");
22             vetA[i] = sc.nextDouble();
23             sum += vetA[i];
24         }
25
26         double media = sum / vetA.length;
27
28         System.out.printf("Média: %.2f%n", media);
29     }
30 }
```



Matriz

- Matriz é uma estrutura de dados bidimensional:

    Homogênea (dados do mesmo tipo)

    Indexada (os elementos são acessados por meio de posições)

    A memória é alocada de uma só vez, em um bloco contíguo da memória.

Vantagem:

    Acesso direto a qualquer elemento, basta informar a posição

Desvantagem:

    De tamanho fixo

Matriz

Exemplo

Um programa que realize a leitura de um número inteiro N que irá representar a dimensão de uma matriz. Depois armazene valores inteiros em uma matriz de tamanho N. Ao final, mostre a diagonal principal e a diagonal secundária da matriz.

Matriz

```java
package matriz;

import java.util.Scanner;

public class exemplo {
    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        System.out.print("Digite a dimensão da matriz: ");
        int n = sc.nextInt();

        int[][] mat = new int[n][n];

        for (int i=0; i<mat.length; i++) {
            for (int j=0; j<mat[i].length; j++) {
                sc.nextLine();
                System.out.printf("Digite o valor[%d][%d]: ", i+1 , j+1);
                mat[i][j] = sc.nextInt();
            }
        }

        System.out.print("Diagonal Principal:");
        for (int i=0; i<mat.length; i++) {
            System.out.print(mat[i][i] + " ");
        }

        System.out.println("");
        System.out.print("Diagonal Secundária:");
        for (int i=0; i<mat.length; i++) {
            for (int j=0; j<mat[i].length; j++) {
                if (i+j+1 == mat.length )
                    System.out.print(mat[i][j] + " ");
            }
        }
        sc.close();
    }
}
```