

INTRODUÇÃO A LÓGICA

Curso Superior Tecnológico em

Sistemas para Internet

Semana4 – Conversões de tipos e Entrada de dados via teclado

Instituto Federal do Tocantins

Prof. Helder – helder@ifto.edu.br

Casting (Conversão)

No java é possível converter um tipo e dado para outro tipo, desde que eles sejam compatíveis. Esse processo chama-se cast ou casting.

Casting é mais aplicado a tipos primitivos. Veja que muitos tipos primitivos são subconjunto de outros tipos primitivos.

Casting só é necessário quando tentamos colocar dados de um grupo (ex. tipo int) em uma variável do tipo byte(int grupo byte subgrupo). Se fizer o processo contrário não é necessário o cast.

Exemplo de cast: long x =3; int y = (int) x;

Palavra reservada **import**

Como na construção de Código java é utilizado classes que não foi o programador que as criou é preciso especificar no Código onde encontrar essas classes no disco durante o processo de compilação, por isso se faz uso da palavra reservada import.

A maiora das classes que é usada para construção de programas está disponível para o programador e são copiadas para o computador do programador quando é instalada o JDK (Kit de Desenvolvimento Java).

Quando o JDK é instalado ele cria uma pasta chamada java e dentro desta pasta é criada outras pastas onde são inseridos os códigos das classes de forma categorizada.

Palavra reservada **import**

```
1 import java.util.Scanner;
2
3 //Exemplo de aplicação interagindo com o teclado
4 public class Prog4 {
    Run | Debug
5     public static void main(String[] args) {
6         Scanner dd;
7         dd = new Scanner(System.in);
8         byte x;
9
10        System.out.print("Digite um número: ");
11        x = dd.nextByte();
12        System.out.println("O numero digitado foi: "+x);
13    }
14 }
```

No exemplo do Código acima está se usando a classe Scanner na linha 6.

Para poder fazer uso dessa classe é preciso informar na linha 1 onde o compilador deve encontrar o Código essa classe.

Palavra reservada **import**

```
1 import java.util.Scanner;
2
3 //Exemplo de aplicação interagindo com o teclado
4 public class Prog4 {
    Run | Debug
5     public static void main(String[] args) {
6         Scanner dd;
7         dd = new Scanner(System.in);
8         byte x;
9
10        System.out.print("Digite um número: ");
11        x = dd.nextByte();
12        System.out.println("O numero digitado foi: "+x);
13    }
14 }
```

Na linha 1 do código cada palavra separada por ponto são pastas, sendo assim java e util são pastas e Scanner é a classe Java.

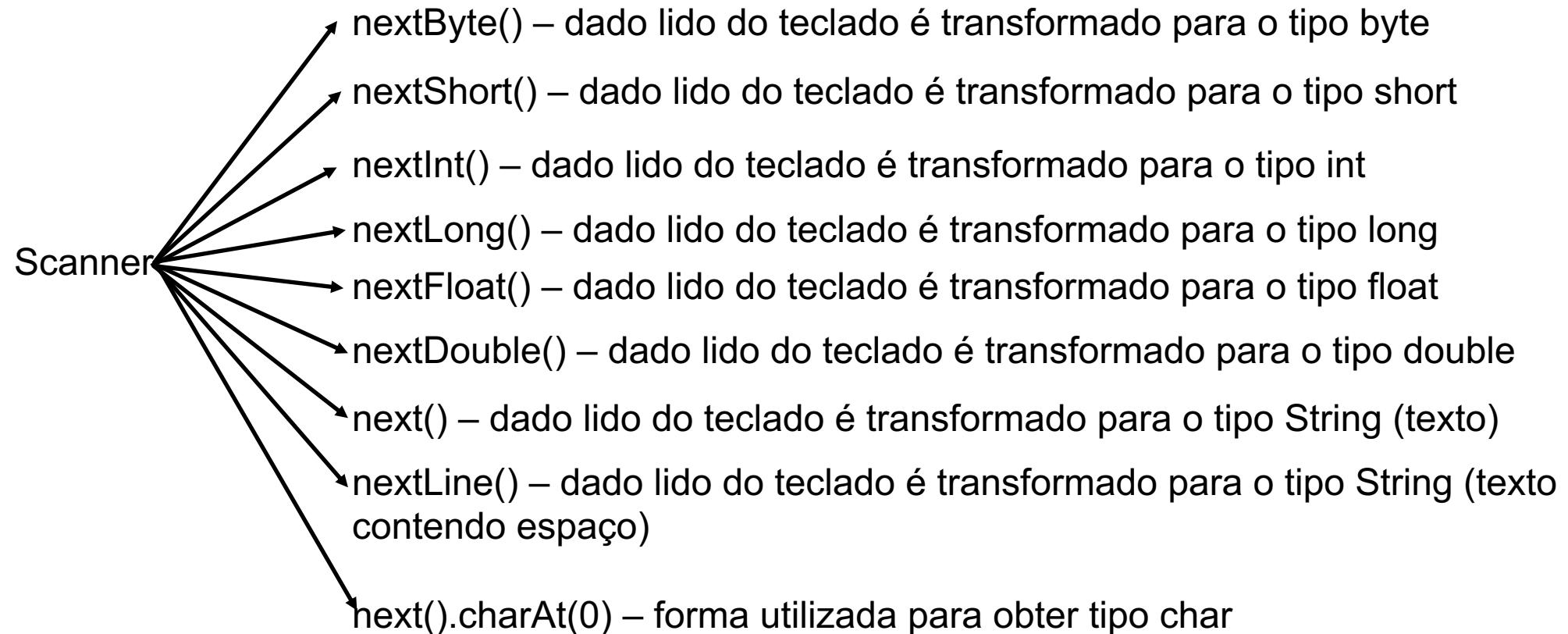
Portanto dentro da pasta java existe uma pasta chamada util e dentro da pasta util se encontra o arquivo Scanner.java. É isso que é informado ao compilador java.

Classe Scanner

A classe Scanner é uma classe que possibilita que a aplicação interaja com o teclado. Pode-se pensar que a classe Scanner é o teclado para a aplicação.

Como a linguagem java é fortemente tipada, ou seja, quando se cria uma variável já se define qual tipo de dado uma variável pode armazenar, portanto quando se usa a classe Scanner ela deve ter a capacidade de ler os dados inseridos pelo usuário via teclado e transformá-lo para um tipo específico conforme instrução java. A classe Scanner possui vários métodos para proporcionar isso.

Classe Scanner – métodos ler teclado



Acesso a atributos ou métodos de Classe/Objeto

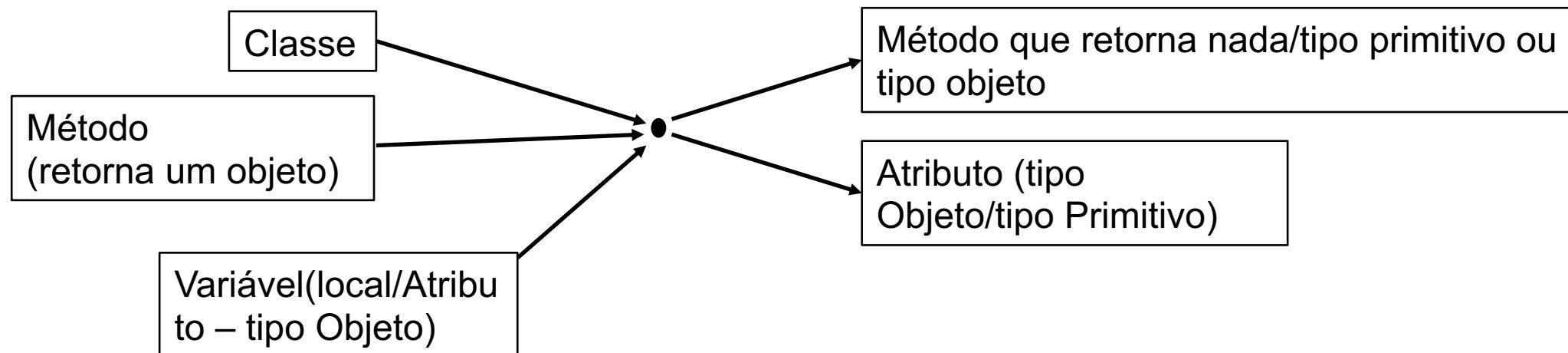
Alguns atributos ou métodos de uma classe só podem ser acessados via o objeto da classe outros podem ser acessados via o nome da classe.

Métodos ou atributos que foram declarados usando a palavra reservada **static** permitem que sejam acessados via o nome da classe. Na ausência desta o seu conteúdo só pode ser acessado via objeto.

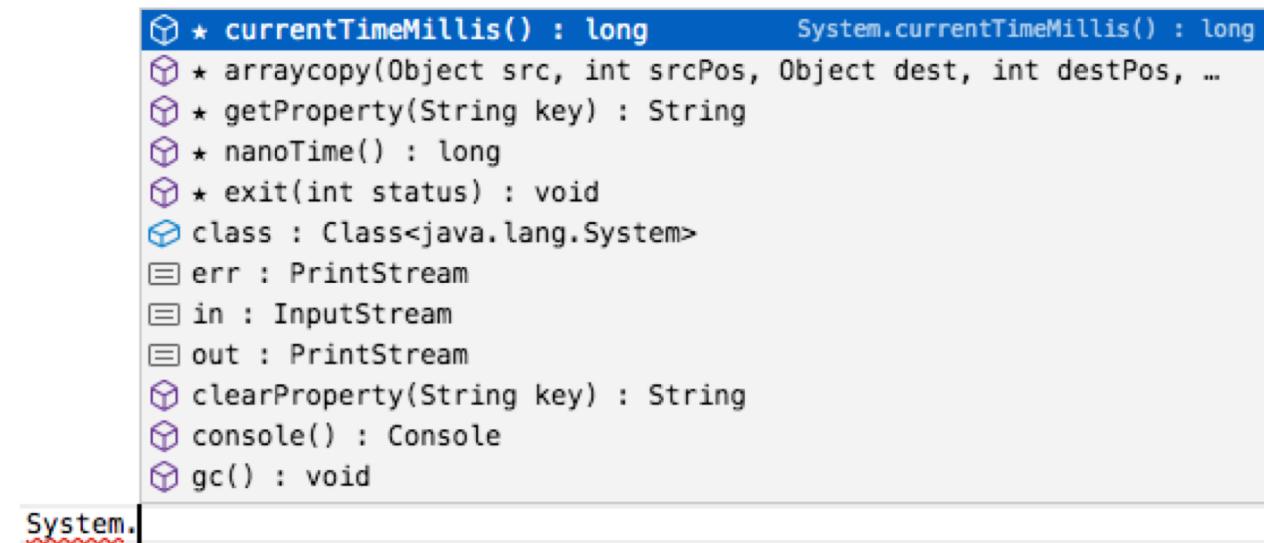
Quando o atributo ou método da classe for criado usando a palavra reservada static estes são chamados atributo/método de classe. Quando são criados não usando a palavra reservada static são chamados atributo ou método de objeto.

Acesso a atributos ou métodos de Classe/Objeto

Para acessar atributos ou métodos de uma classe ou objeto usa-se o ponto (.). Tudo o que estiver a direita do ponto pertence àquilo que estiver a esquerda do ponto.



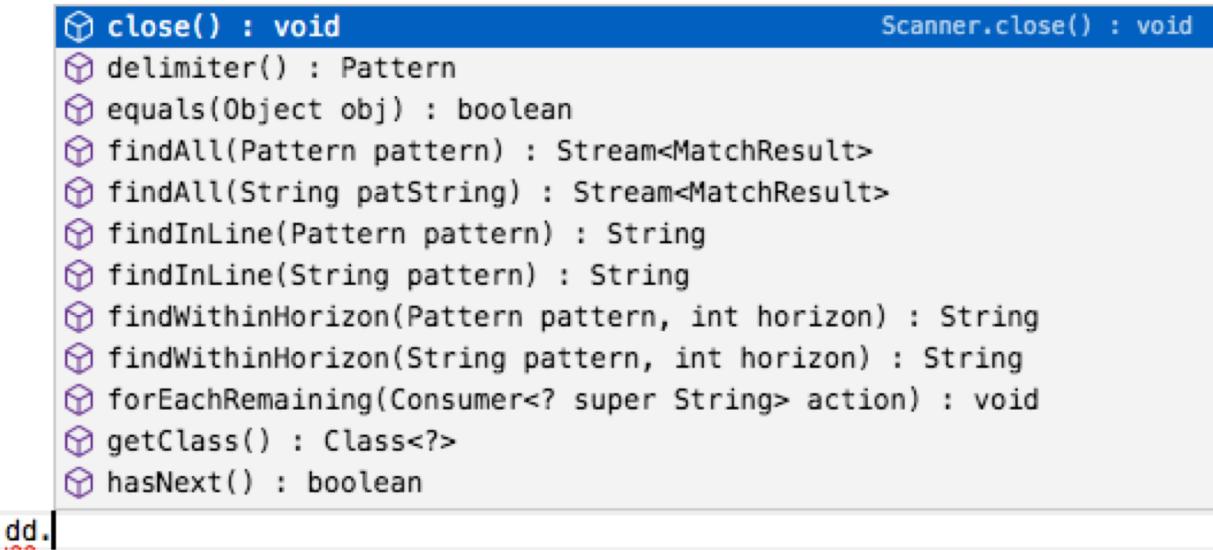
Acesso a atributos ou métodos da Classe System



No Código acima ao se digitar a palavra `System` seguida de ponto a IDE apresenta todos os atributos e métodos de classe (criados com a palavra reservada `static`). As palavras mostradas seguidas de parênteses são métodos de classe da classe `System`, já as palavras sem parênteses são atributos da classe da classe `System`.

Acesso a atributos ou métodos do objeto Scanner

```
Scanner dd;  
dd = new Scanner(System.in);
```



No Código acima ao se digitar a palavra dd seguida de ponto a IDE apresenta todos os atributos e métodos de objeto (criados sem a palavra reservada static). As palavras mostradas seguida de parênteses são métodos do objeto da classe Scanner, já as palavras sem parênteses são atributos do objeto da classe Scanner.

Interpretação das Informações - exemplos

```

System.currentTimeMillis() : long      System.currentTimeMillis() : long
System.arraycopy(Object src, int srcPos, Object dest, int destPos, ...
System.getProperty(String key) : String
System.nanoTime() : long
System.exit(int status) : void
System.class : Class<java.lang.System>
System.err : PrintStream
System.in : InputStream
System.out : PrintStream
System.clearProperty(String key) : String
System.console() : Console
System.gc() : void

```

System.

```

Scanner.close() : void
Scanner.delimiter() : Pattern
Scanner.equals(Object obj) : boolean
Scanner.findAll(Pattern pattern) : Stream<MatchResult>
Scanner.findAll(String patString) : Stream<MatchResult>
Scanner.findInLine(Pattern pattern) : String
Scanner.findInLine(String pattern) : String
Scanner.findWithinHorizon(Pattern pattern, int horizon) : String
Scanner.findWithinHorizon(String pattern, int horizon) : String
Scanner.forEachRemaining(Consumer<? super String> action) : void
Scanner.getClass() : Class<?>
Scanner.hasNext() : boolean

```

dd.

Classe System. **currentTimeMillis(): long** -> no caso está falando que esse método da Classe System chamado currentTimeMillis() não recebe nada como parâmetro e ao ser executado retorna um tipo de dado **long**;

Interpretação das Informações - exemplos

```

System.currentTimeMillis() : long      System.currentTimeMillis() : long
System.arraycopy(Object src, int srcPos, Object dest, int destPos, ...
System.getProperty(String key) : String
System.nanoTime() : long
System.exit(int status) : void
System.class : Class<java.lang.System>
System.err : PrintStream
System.in : InputStream
System.out : PrintStream
System.clearProperty(String key) : String
System.console() : Console
System.gc() : void

```

System.

```

Scanner.close() : void
Scanner.delimiter() : Pattern
Scanner.equals(Object obj) : boolean
Scanner.findAll(Pattern pattern) : Stream<MatchResult>
Scanner.findAll(String patString) : Stream<MatchResult>
Scanner.findInLine(Pattern pattern) : String
Scanner.findInLine(String pattern) : String
Scanner.findWithinHorizon(Pattern pattern, int horizon) : String
Scanner.findWithinHorizon(String pattern, int horizon) : String
Scanner.forEachRemaining(Consumer<? super String> action) : void
Scanner.getClass() : Class<?>
Scanner.hasNext() : boolean

```

dd.

Variável dd (objeto do tipo Scanner). **close(): void** -> no caso está falando que esse método de objeto (Classe Scanner) chamado close() não recebe nada como parâmetro e também não retorna nada (void);

Estudo de caso da execução do programa Java

```
import java.util.Scanner;

//Exemplo de aplicação interagindo com o teclado
public class Prog4 {
    Run | Debug
    public static void main(String[] args) {
        Scanner dd;
        dd = new Scanner(System.in);
        byte x;

        System.out.print("Digite um número: ");
        x = dd.nextByte();
        System.out.println("O numero digitado foi: "+x);
    }
}
```

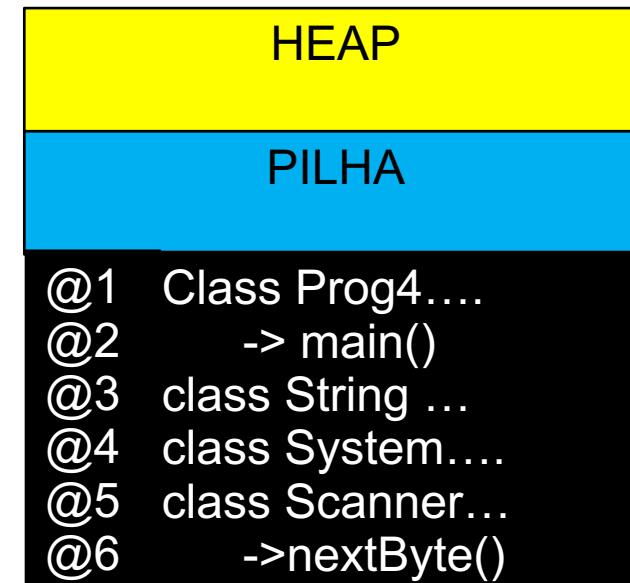
Toda classe que não pertence ao pacote `java.lang` (`System`, `String`...) precisam ser explicitamente importado para o Código usado a palavra reservada `import`.

Estudo de caso da execução do programa Java – Fase 1

```
import java.util.Scanner;

//Exemplo de aplicação interagindo com o teclado
public class Prog4 {
    Run | Debug
    public static void main(String[] args) {
        Scanner dd;
        dd = new Scanner(System.in);
        byte x;

        System.out.print("Digite um número: ");
        x = dd.nextByte();
        System.out.println("O numero digitado foi: "+x);
    }
}
```



Nesta fase – há alocação de memória para o programa (divisão lógica da mesma) e a carga do Código (do hd para a memória) a ser executado).

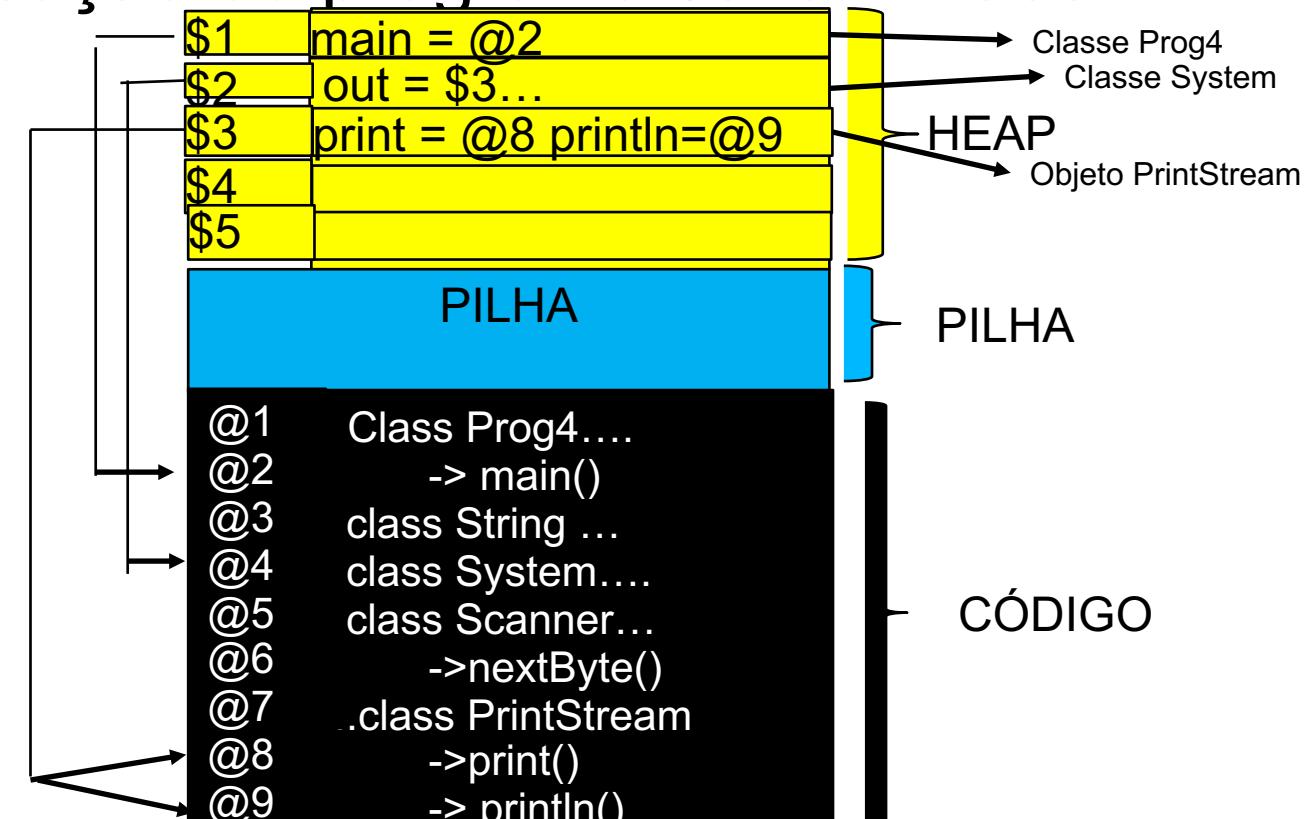
Estudo de caso da execução do programa Java – Fase 2

J Prog4.java > ...

```

1 import java.util.Scanner;
2
3 //Exemplo de aplicação interagindo com o teclado
4 public class Prog4 {
    Run | Debug
5     public static void main(String[] args) {
        Scanner dd;
6         dd = new Scanner(System.in);
7         byte x;
8
9         System.out.print("Digite um número: ");
10        x = dd.nextByte();
11        System.out.println("O numero digitado foi: "+x);
12    }
13}
14
15

```



Nesta fase – há alocação de memória para para a classe no HEAP para armazenar áquito que é de classe e os objetos que foram criados implicitamente

Estudo de caso da execução do programa Java – Fase 2

Foi reservado uma área de memória para classe **Prog4** no HEAP porque existe o método de classe, no caso **main()**, dentro da classe **Prog4**.

Foi reservado uma área de memória para a classe **System** no HEAP porque existe o atributo de classe chamado **out** dentro da classe **System**.

Foi criado uma área de memória para um objeto referente a classe **PrintStream** pois o atributo chamado **out** na classe System contém um endereço para um objeto da classe **PrintStream**. Obs. Sabe-se disso olhando a documentação referente a classe **System**, isso não é óbvio apenas olhando o Código.

Essas áreas de memória no HEAP foram alocadas a partir da análise do Código a ser executado.

Estudo de caso da execução do programa Java – Fase 2

Na área de memória chamada **Código** existe o código propriamente dito no formato que a máquina virtual JAVA entende, sendo denominado **byte code**. Essa área de memória é apenas de leitura, ou seja, não é possível alterar o conteúdo dessa área de memória enquanto o programa está sendo executado.

Todo dado gerado durante a execução do Código é colocado na área de memória denominada HEAP ou na área de memória chamada PILHA.

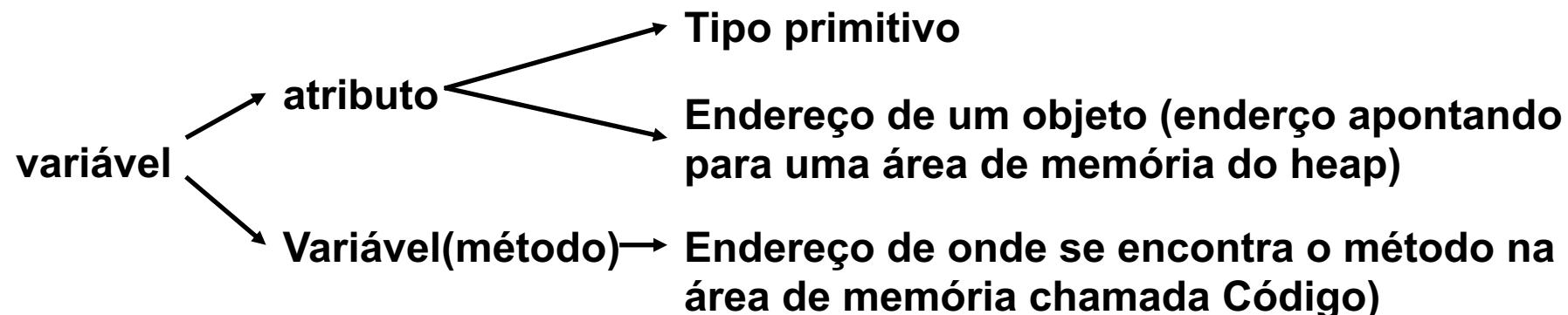
OBS: @ nos desenhos representa endereço de memória na área de Código e \$ representa endereço de memória na área HEAP. São apenas ilustrativos.

Estudo de caso da execução do programa Java – Fase 2

Conteúdo de um objeto ou classe no HEAP

Dentro da representação de um objeto ou classe no HEAP só existe variáveis (atributos) e variável que resenta o nome do método.

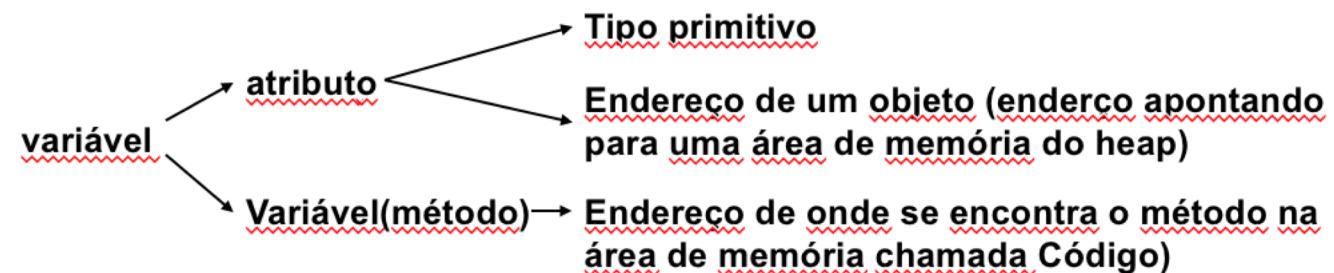
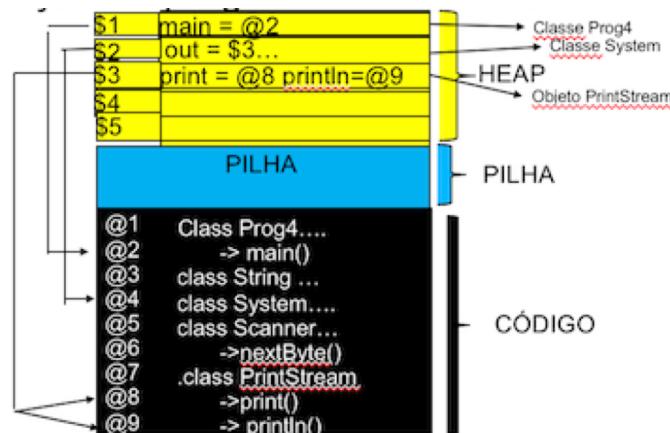
Todo nome de método construído dentro de uma classe também é uma variável. Portanto dentro de uma representação de classe ou objeto no HEAP temos:



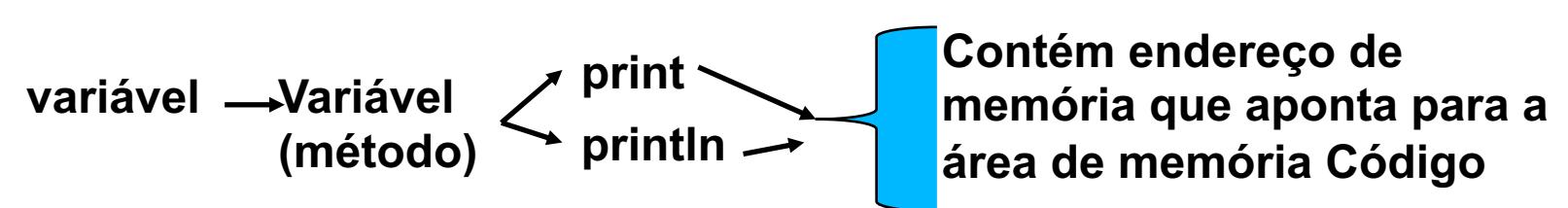
Estudo de caso da execução do programa Java – Fase 2

Conteúdo de um objeto ou classe no HEAP

Para nossa representação da classes na memória até este momento da execução temos:



Exemplo da reprentação do objeto PrintStream (Na HEAP)



Estudo de caso da execução do programa Java – Fase 3

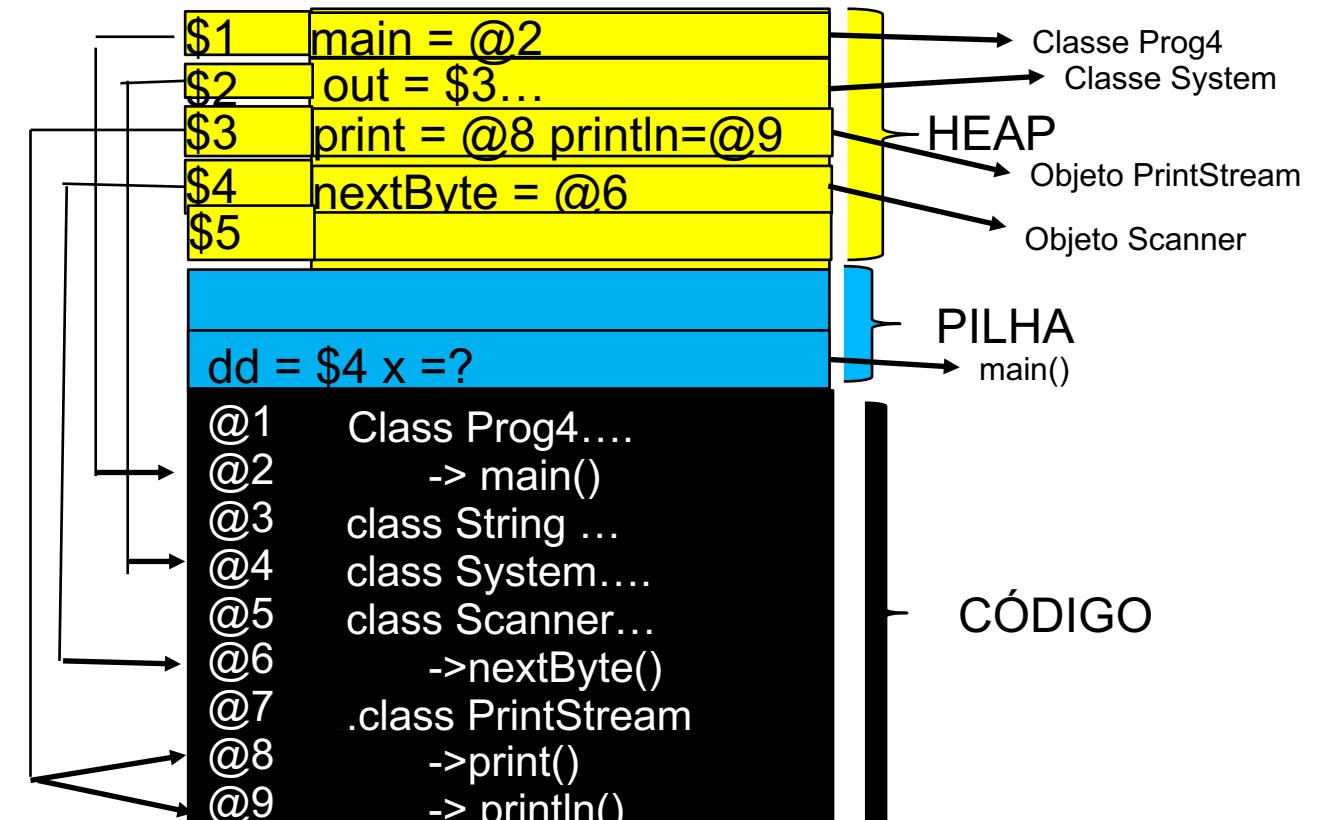
Nesta fase de execução se busca o primeiro método a ser executado, no caso é sempre o **main**. Para uma classe ser um programa executável ela obrigatoriamente deve possuir o Código main em seu bloco, caso contrário ela passaria a ser uma biblioteca a ser utilizada por outros programas, como é o caso da classe System, String, PrintStream dentre outras.

A JVM já conhece a localização física da área de memória (HEAP) onde se encontra a classe do programa a ser executar, no nosso caso o Prog4. Dentro dessa área de memória ele busca acessar a variável main, obtendo assim o endereço de memória de onde se encontra o início (na área de memória do Código) o método main.

Uma vez que encontra o método main, é alocado memória na pilha para uso exclusivo do método main. Todo dado criado via instrução contida no método main é criado nessa área de memória. Essa área de memória alocada a um método é chamado de bloco da pilha.

Estudo de caso da execução do programa Java – Fase 3

```
J Prog4.java > ...
1 import java.util.Scanner;
2
3 //Exemplo de aplicação interagindo com o teclado
4 public class Prog4 {
    Run | Debug
    5     public static void main(String[] args) {
        6         Scanner dd;
        7         dd = new Scanner(System.in);
        8         byte x;
        9
        10        System.out.print("Digite um número: ");
        11        x = dd.nextByte();
        12        System.out.println("O numero digitado foi: "+x);
        13    }
        14
        15
```



Cenário de execução até a linha 8 do método main

Estudo de caso da execução do programa Java – Fase 3

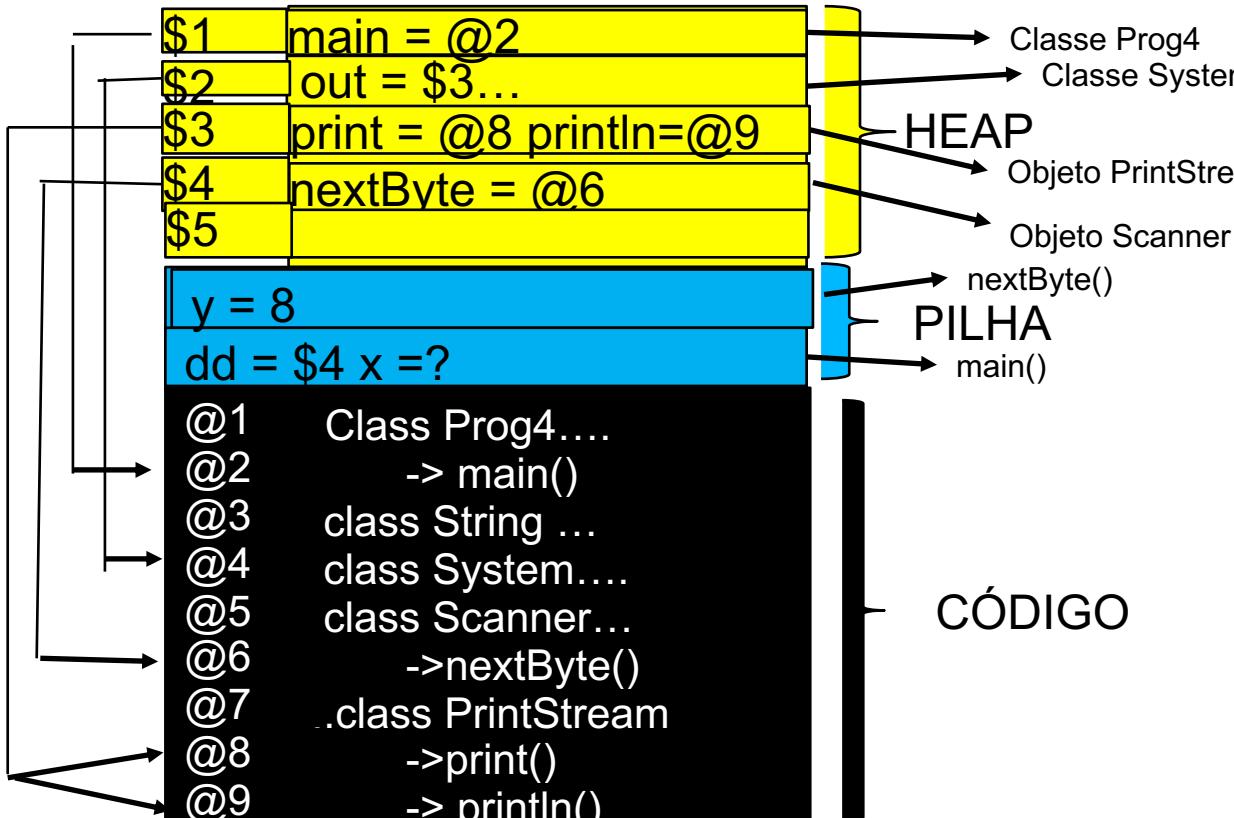
Veja que todas os dados criados até a execução da linha 8 do método main, foram criados na pilha.

Como na linha 7 existe uma instrução(**new**) para criar um objeto do tipo Scanner, o objeto foi criado na área de memória HEAP e no bloco da pilha do main foi criado uma variável chamada dd que o seu conteúdo é um endereço de memória, ou seja, o endereço de onde se localiza o objeto Scanner na memória HEAP.

Lembre-se fisicamente todo objeto fica localizado na área de memória HEAP.

Estudo de caso da execução do programa Java – Fase 3

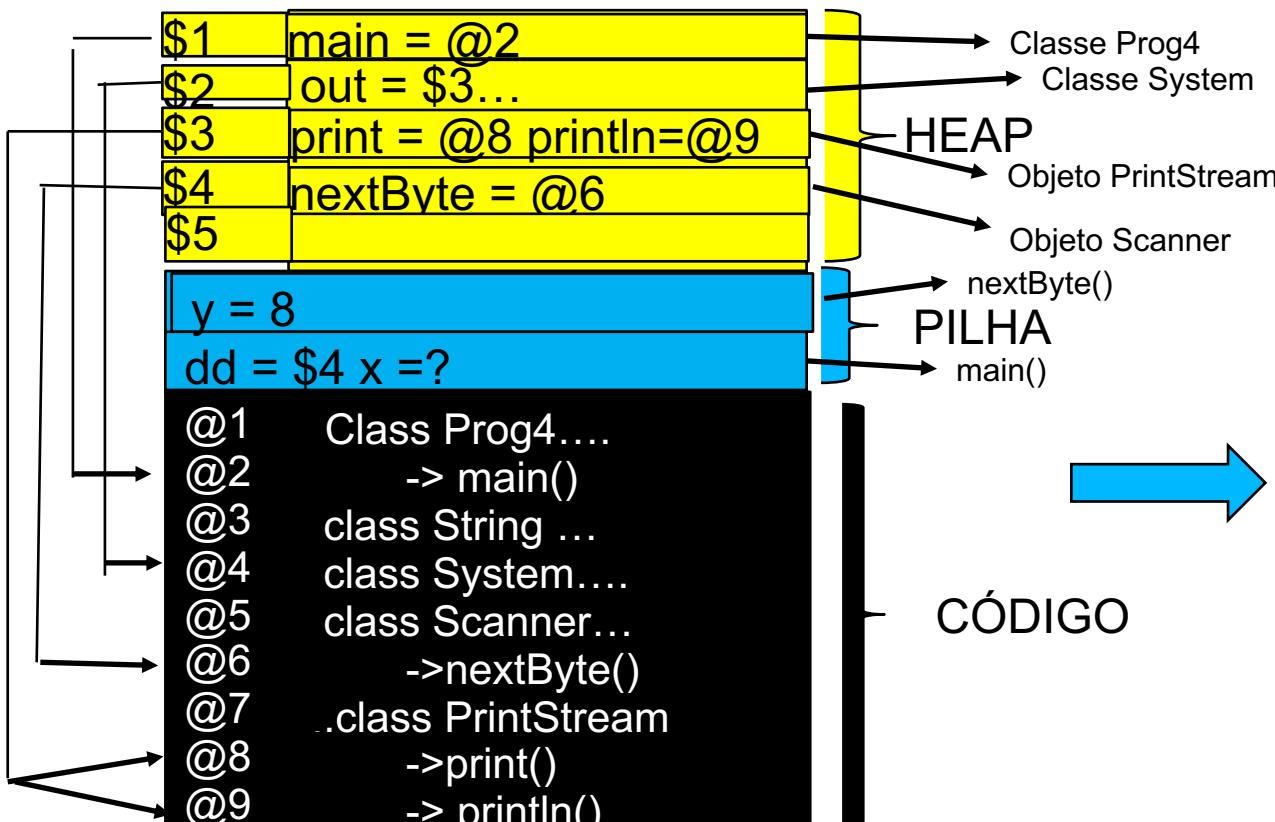
Executando a Linha 11



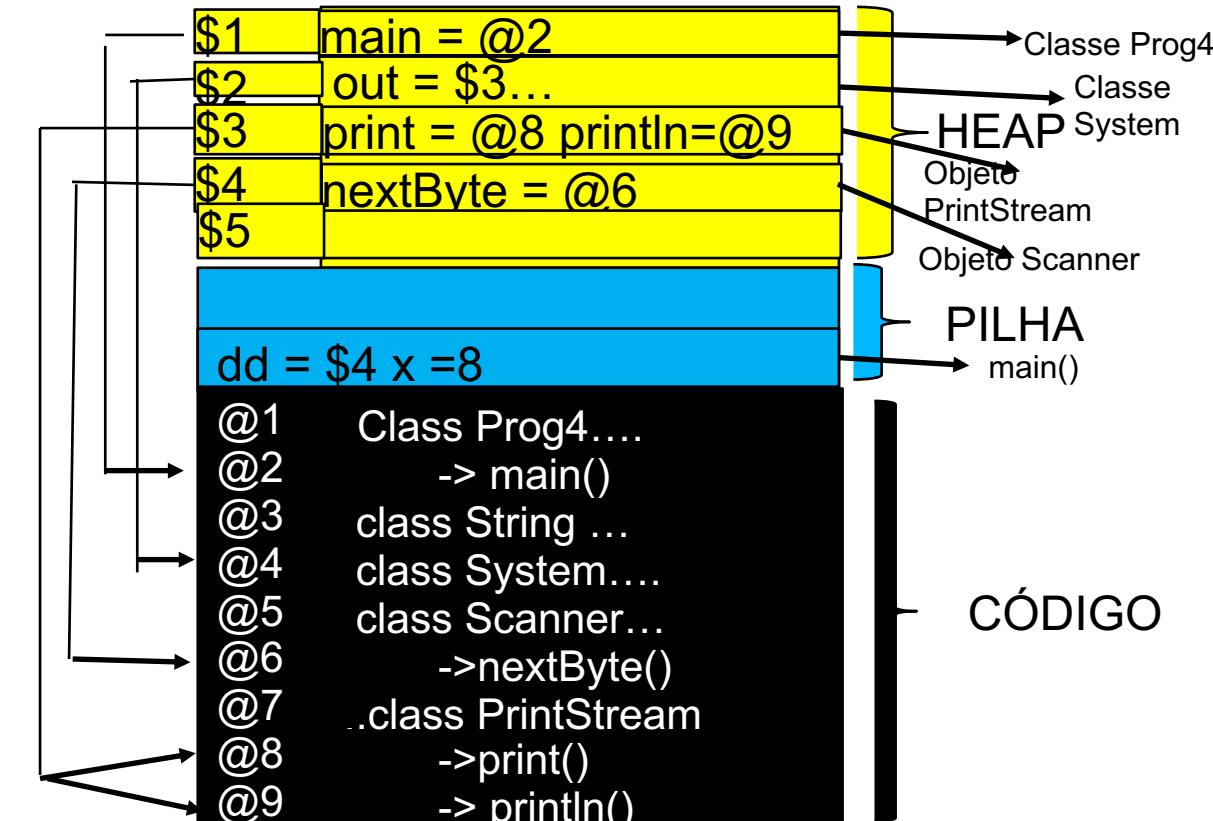
1. Ao executar a linha 11: `x = dd.nextByte();` A variável **dd** é buscada dentro da pilha (método main) ao acessá-la o tem-se acesso ao objeto Scanner (HEAP) onde se encontra a variável nextByte que contém o endereço de memória de onde começa o Código nextByte (área de Código).
2. Antes de executar o método nextByte é alocado uma área de memória na pilha onde são criados os dados conforme instrução contida no método nextByte.
3. 8 representa o valor digitado pelo usuário via teclado e armazenado em uma variável que estou chamando de y (o nome da variável não importa).

Estudo de caso da execução do programa Java – Fase 3

Executando a Linha 11



CÓDIGO



CÓDIGO

Estudo de caso da execução do programa Java – Fase 3

Executando a Linha 11

Veja que durante o processo de execução foi alocado uma área de memória na pilha para o método nextByte() e o valor de retorno do método nextByte() foi colocado na variável x do método main. A área de memória alocado para o método nextByte() foi apagada porque o método foi encerrado. Devido a isso que se diz que os dados da pilha são temporários.