Chapter-3:
Know Your Variables

Upcode Software
Engineer Team

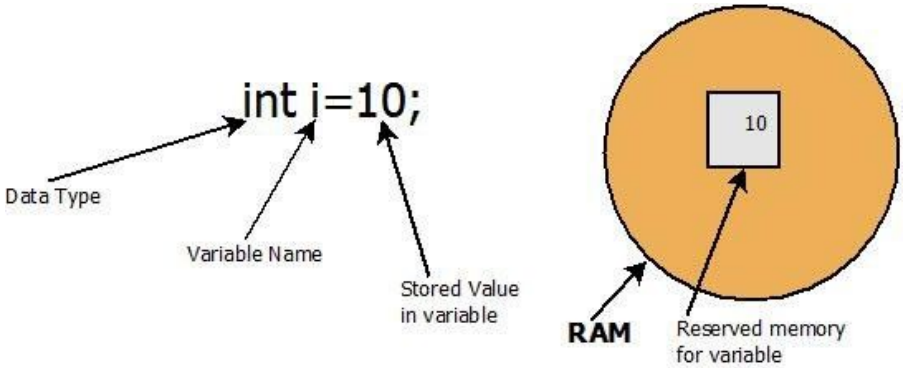
-2023-

CONTENT

1. Variables in Java
2. Primitive data type
3. Reference data type
4. What is garbage collectible heap?
5. Reference vs Primitive
6. Used materials and references

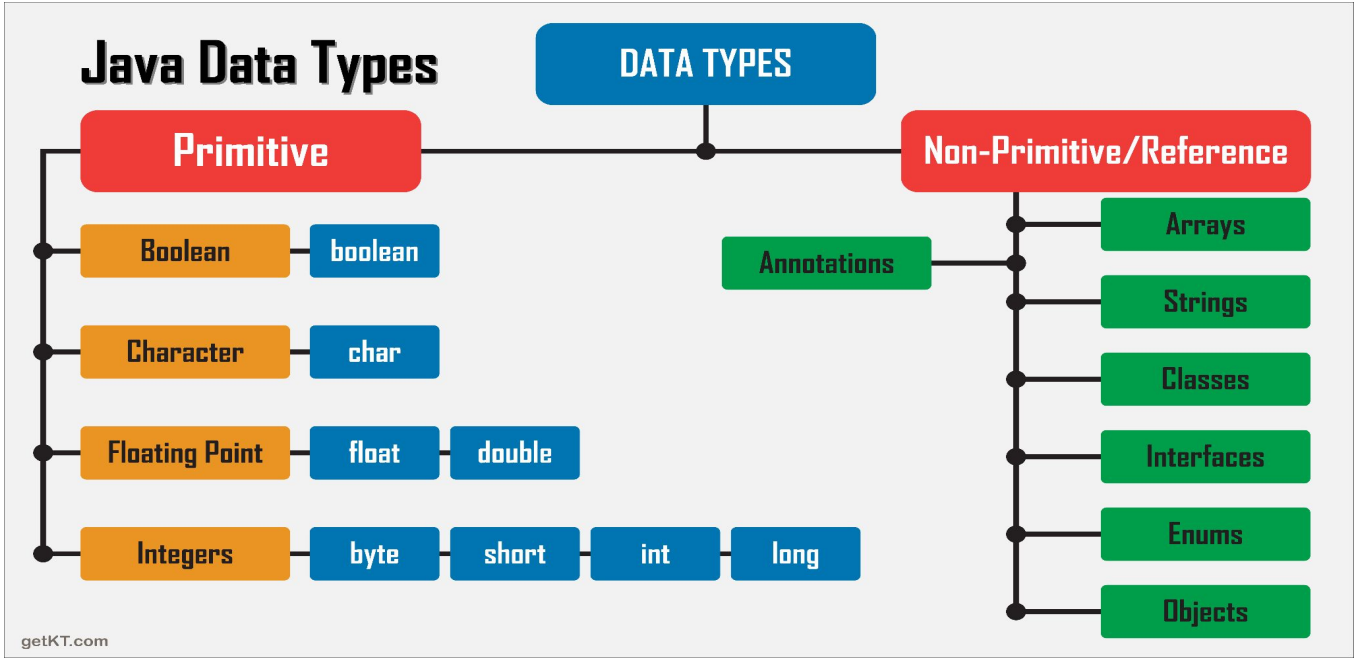
Variables in Java(1/n)

- In Java, **Variables** are the data containers that save the data values during Java program execution.
- Variables **must have** a **type!**
- Variables **must have** a **name!**
- **Variable**, in simpler terms, is a **name** given to a **memory location.**

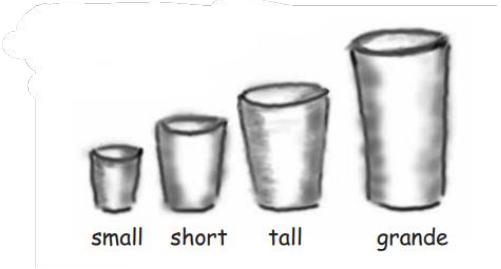


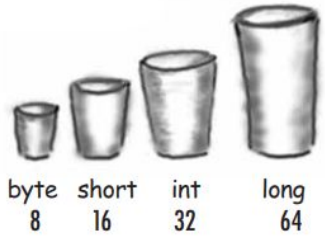
Variables in Java(2/n)

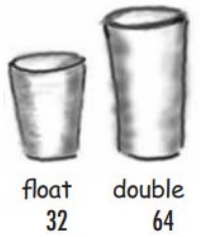
- In Java, variables come in two flavors: **primitive** and **reference**
- All the variables must **declare** with **variable type** and **name**.



Primitive data type (1/n)



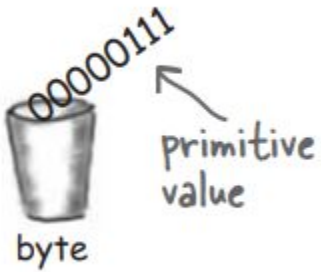




Primitive Variable

```
byte x = 7;
```

The bits representing 7 go into the variable. (00000111).

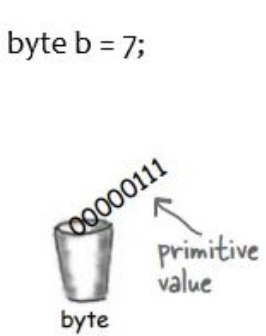


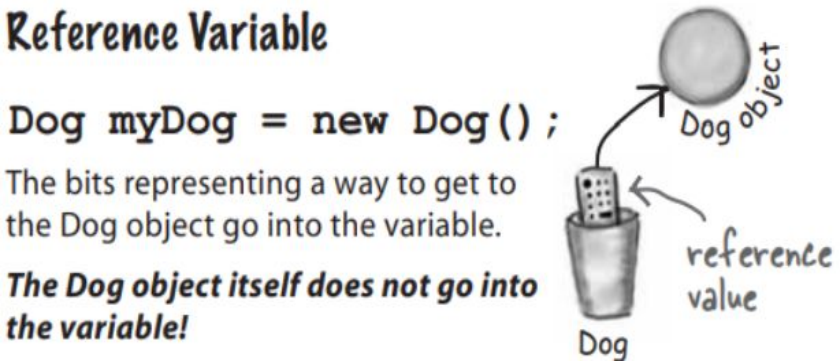
Primitive data type(2/n)

Primitive Type	Size	Minimum Value	Maximum Value
char	16-bit	Unicode 0	Unicode 2^{16}-1
byte	8-bit	-128	+127
short	16-bit	-2^{15} *(-32,768)*	$+2^{15}$-1 *(32,767)*
int	32-bit	-2^{31} *(-2,147,483,648)*	$+2^{31}$-1 *(2,147,483,647)*
long	64-bit	-2^{63} *(-9,223,372,036,854,775,808)*	$+2^{63}$-1 *(9,223,372,036,854,775,807)*
float	32-bit	32-bit IEEE 754 floating-point numbers	
double	64-bit	64-bit IEEE 754 floating-point numbers	
boolean	1-bit	`true` or `false`	
void	-----	-----	-----

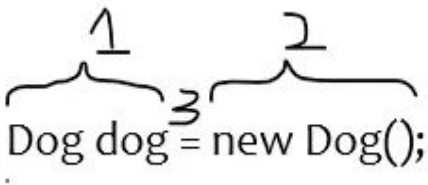
Reference data type (1/n)

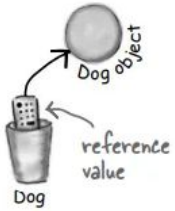
- The **object reference variable** holds bits that represent a **way to access an object.**
- **Object reference variable** like remote control of an object and It has **a type** but it doesn't have **exactly a size.**





Reference data type (2/n)

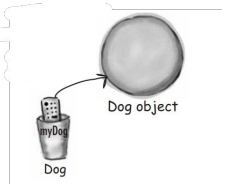




1 → Declaration —Tell the JVM to allocate space for the reference variable whose name is the dog and Whose type is Dog forever.

2 → Creation — Tell the JVM to allocate space for a new Dog object on the heap.

3 → Assignment — Assign a new Dog object to the dog reference variable.



Reference data type (3/n)

1. Declare an int array variable. An array variable is a remote control to an array object.

```
int[] nums;
```

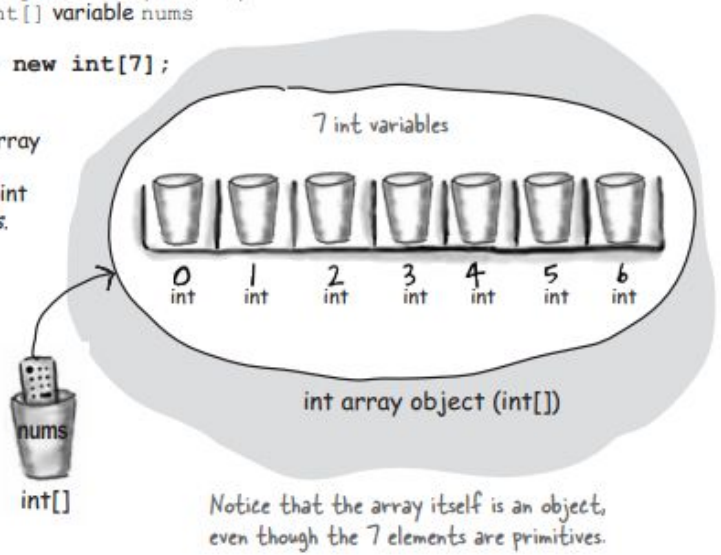
2. Create a new int array with a length of 7, and assign it to the previously-declared `int[]` variable `nums`

```
nums = new int[7];
```

3. Give each element in the array an int value.
Remember, elements in an int *array* are just int *variables*.

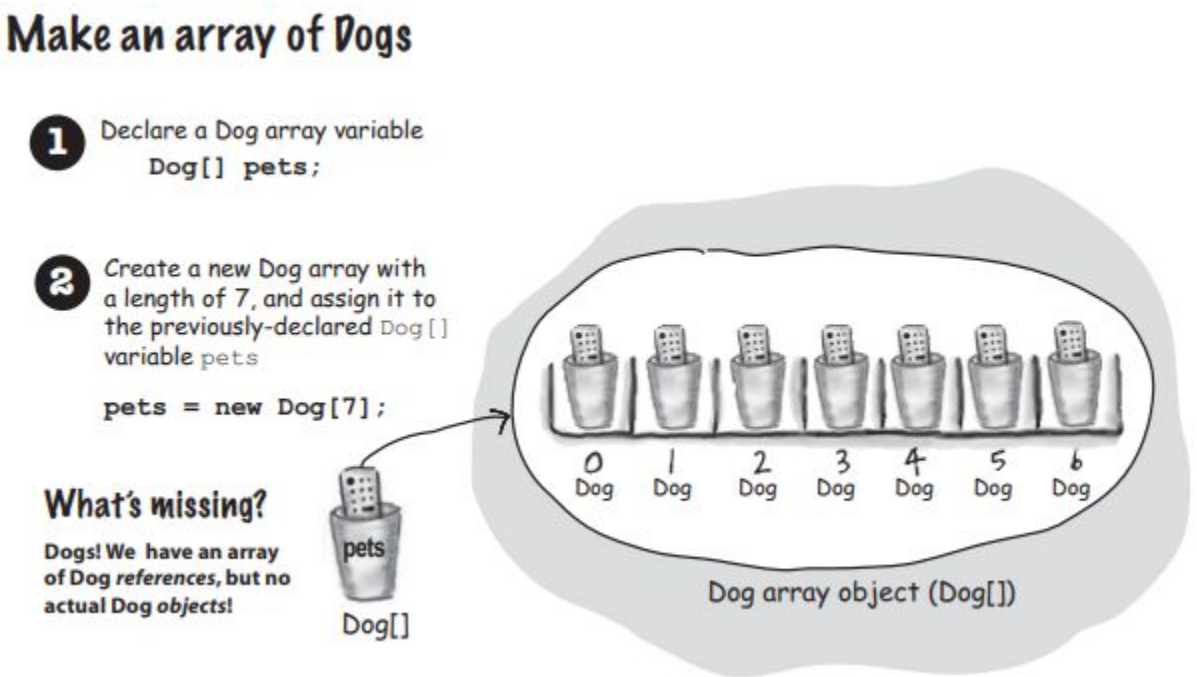
```
nums[0] = 6;
nums[1] = 19;
nums[2] = 44;
nums[3] = 42;
nums[4] = 10;
nums[5] = 20;
nums[6] = 1;
```

7 int variables



Notice that the array itself is an object, even though the 7 elements are primitives.

Reference data type (4/n)



Reference data type (5/n)

3 Create new Dog objects, and assign them to the array elements.
Remember, elements in a Dog *array* are just Dog reference *variables*. We still need Dogs!

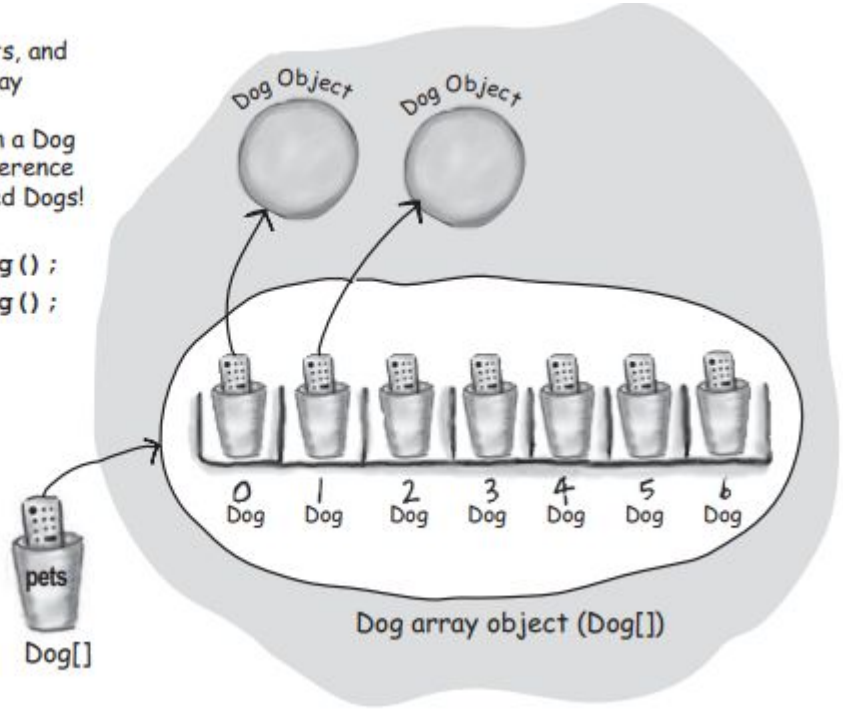
```
pets[0] = new Dog();
pets[1] = new Dog();
```

Sharpen your pencil

What is the current value of pets[2]? _____

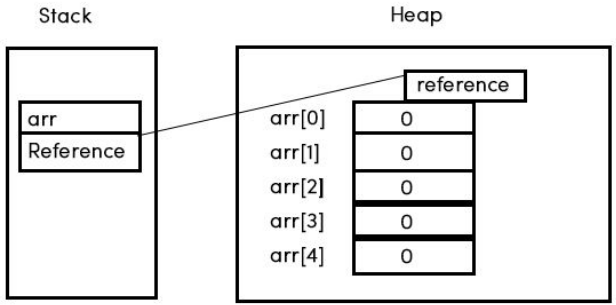
What code would make pets[3] refer to one of the two existing Dog objects?

_____



Reference data type (6/n)

```
int[] arr = new int[5];
```



Reference data type (7/n)

- Each element will occupy the memory space required to accommodate the values for its type, i.e.; depending on the data type of the elements, 1, 4, or 8 bytes of memory are allocated for each element. The next memory address is assigned to the next element in the array.
- Minimum object size is 16 bytes for modern 64-bit JDK since the object has 12-byte header, padded to a multiple of 8 bytes. In 32-bit JDK, the overhead is 8 bytes, padded to a multiple of 4 bytes.
- That's not specified by the Java standard and thus you should not worry about it. Technically, references are usually as big as the machine's word size, i.e. 32 bit on a 32 bit machine and 64 bit on a 64 bit machine, though some 64 bit JVMs use special magic to allow 32 bit references.

What is garbage collectible heap?(1/n)

Life on the garbage-collectible heap

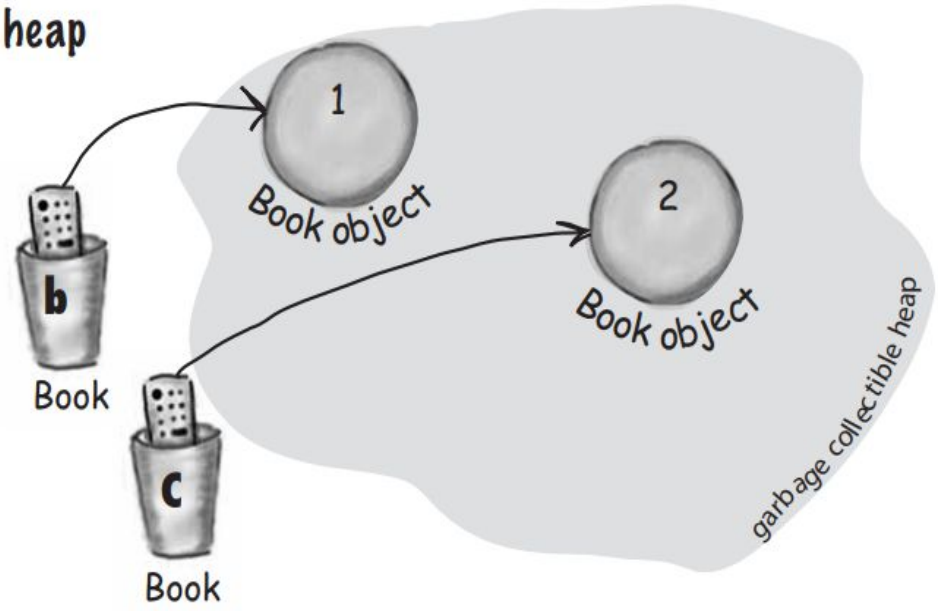
```
Book b = new Book();
Book c = new Book();
```

Declare two Book reference variables. Create two new Book objects. Assign the Book objects to the reference variables.

The two Book objects are now living on the heap.

References: 2

Objects: 2



What is garbage collectible heap?(2/n)

```
Book d = c;
```

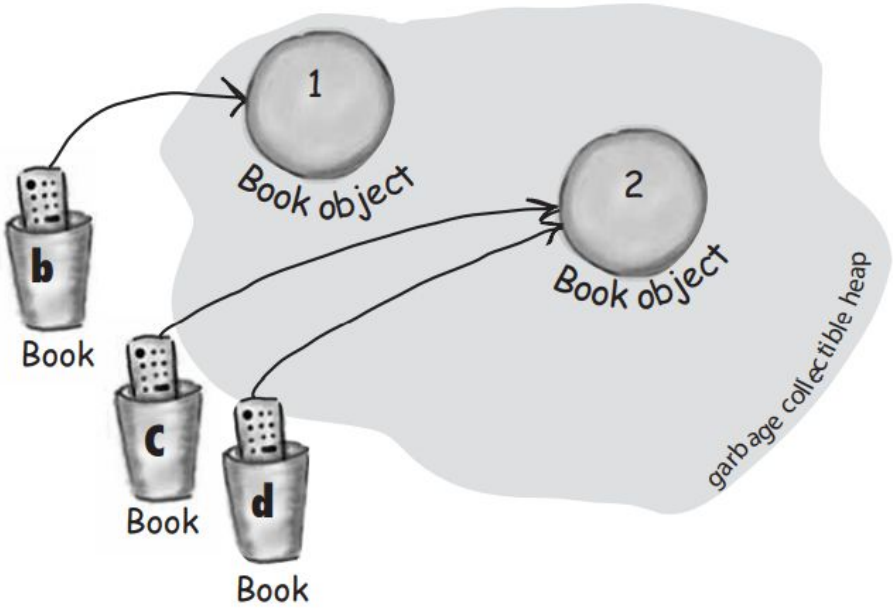
Declare a new Book reference variable. Rather than creating a new, third Book object, assign the value of variable ***c*** to variable ***d***. But what does this mean? It's like saying, "Take the bits in ***c***, make a copy of them, and stick that copy into ***d***."

Both *c* and *d* refer to the same object.

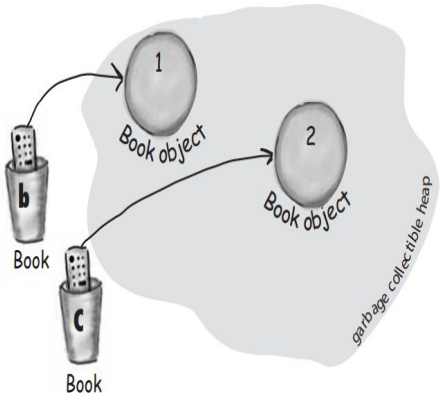
The *c* and *d* variables hold two different copies of the same value. Two remotes programmed to one TV.

References: 3

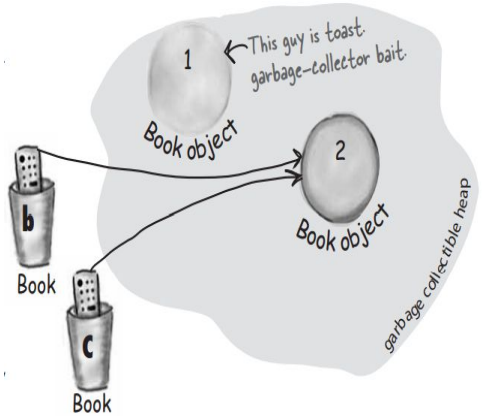
Objects: 2



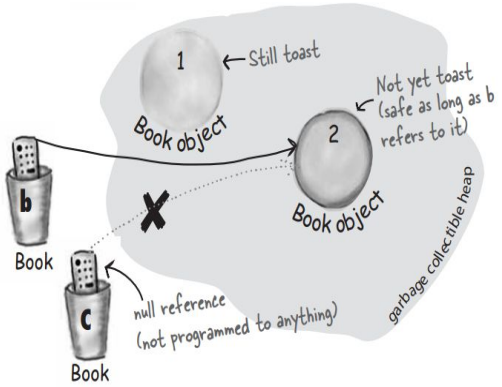
What is garbage collectible heap?(3/n)




```
Book b = new Book();
Book c = new Book();
```




```
b = c;
```




```
c = null;
```

Reference vs Primitive

Reference vs Primitive Data Types

Reference Type	Primitive Type
It is not pre-defined except the String.	It is pre-defined in Java.
All reference type begins with Uppercase letter.	All primitive type begins with a lowercase letter.
Non-primitive types have all the same size.	The size of a primitive type depends on the data type.
It is used to invoke or call methods.	We cannot invoke the method with a primitive type.
It can be null.	It cannot be null. It always has value.
Examples of reference data types are class, Arrays, String, Interface, etc.	Examples of primitive data types are int, float, double, Boolean, long, etc.
JVM allocates 8 bytes for each reference variable, by default.	Its size depends on the data type.
Example: Demo d1;	**Example**: int num=78;

Thank you!

Presented by

Jamshid Erkinov

(jamshiderkinov19992206@gmial.com)