Week - 02 Introduction to OOP

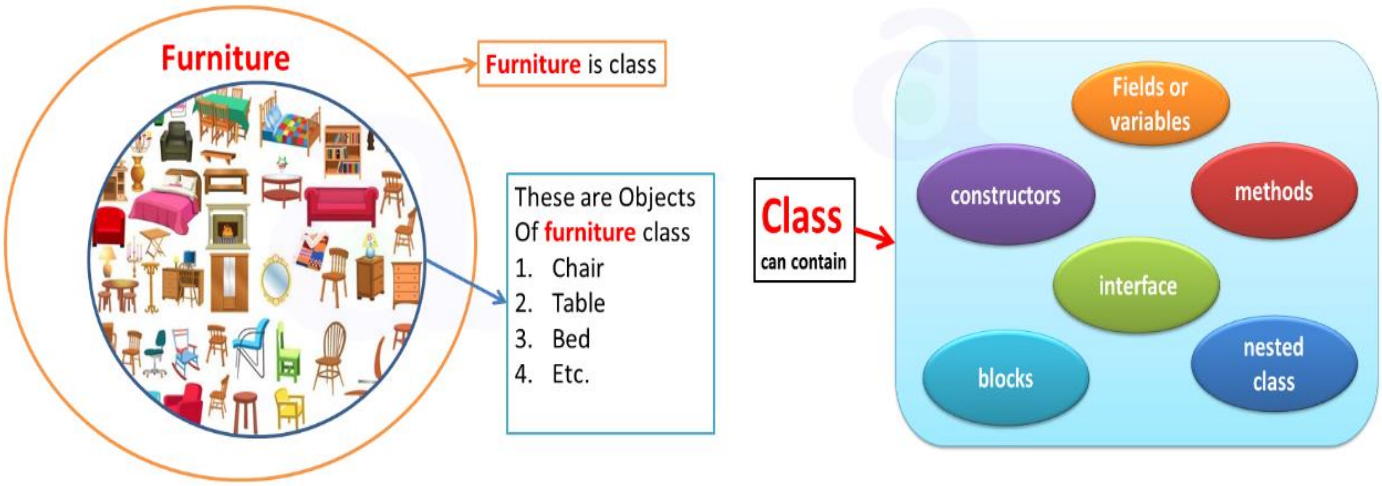
INTRODUCTION

- Classes and objects are the fundamental components of OOP's.
- JAVA is a true object-oriented language and therefore the underlying structure of all java programs is classes.
- In Java, data items are called *fields* and the functions are called *methods*

BUILDING BLOCKS

Defining A Class

A class is an entity that determines how an object will behave and what the object will contain. In other words, it is a blueprint or a set of instruction to build a specific type of object.



Syntax to declare a class:

```
class Class_name [ extends Superclass_name]
{
   [ variable declaration; ]
   [ methods declaration; ]
}
```

Adding Variables

Data is encapsulated in a class by placing data fields inside the body of the class definition. These variables are called instance variables because they are created whenever an object of the class is instantiated.

```
class Box
{
        int length;
        int width;
        int depth;
}
```

The class Box contains three integer type instance variables.

Adding Methods

Methods are declared inside the body of the class but immediately after the declaration of instance variables. The general form of method declaration is

```
type methodname (parameter-list)
{
        method-body;
}
```

```
class Box
{
        int length, width, depth;
        void getData(int x,int y,int z)
        {
        length=x;
        width=y;
        depth=z;
        }
        int boxVolume()
        {
        int vol=length*width*depth;
        return(vol);
        }
}
```

Distinction between a class and an object

- A class creates a new data type that can be used to create objects. That is, a class creates a logical framework that defines the relationship between its members. When you declare an object of a class, you are creating an instance of that class. Thus, a class is a logical construct.
- An object has physical reality. (That is, an object occupies space in memory.) It is important to keep this distinction clearly in mind.

Creating Objects

The new operator dynamically allocates memory for an object. It has this general form:

class-var **= new** ***classname*****();**

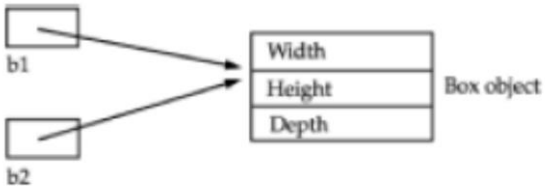
The class name followed by parentheses specifies the *constructor* for the class. A constructor defines what occurs when an object of a class is created. Java will automatically supply a default constructor example, what do you think the following fragment does?

```
Box b1 = new Box();
Box b2 = b1;
```

You might think that **b2** is being assigned a reference to a copy of the object referred to by **b1**. That is, you might think that **b1** and **b2** refer to separate and distinct objects.

However, this would be wrong. Instead, after this fragment executes, **b1** and **b2** will both refer to the *same* object. The assignment of **b1** to **b2** did not allocate any memory or copy any part of the original object. It simply makes **b2** refer to the same object as does **b1**. Thus, any changes made to the object through **b2** will affect the object to which **b1** is referring, since they are the same object.

This situation is depicted here:



Although **b1** and **b2** both refer to the same object, they are not linked in any other way.

For example, a subsequent assignment to **b1** will simply *unhook* **b1** from the original object without affecting the object or affecting **b2**. For example:

```
Box b1 = new Box();
Box b2 = b1;
// ...
b1 = null;
```

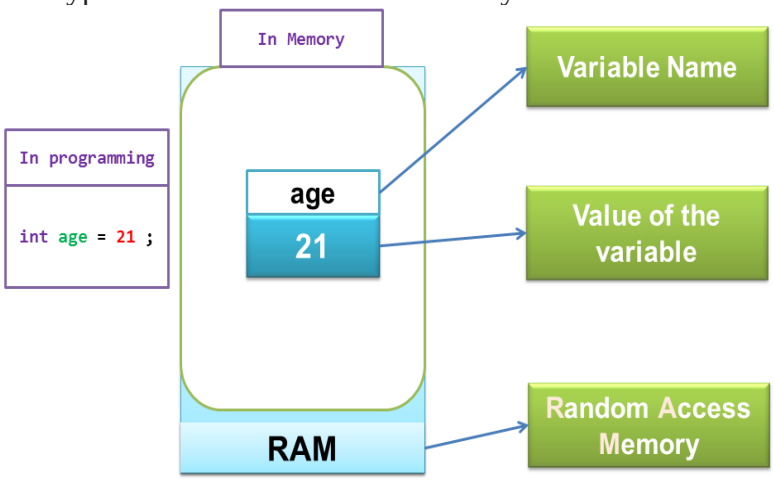
Here, **b1** has been set to **null**, but **b2** still points to the original object.

Note

When you assign one object reference variable to another object reference variable, you are not creating a copy of the object, you are only making a copy of the reference.

VARIABLES

The Value which changes throughout the execution of the program is called as Variables. A variable is a container which holds the value while the Java program is executed. A variable is assigned with a data type. Variable is a name of memory location.



The Java programming language defines the following kinds of variables:

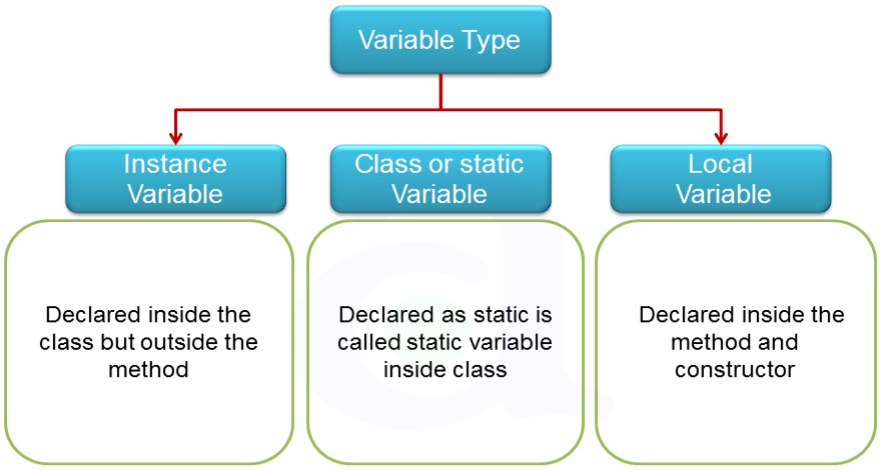
1. Instance Variables
2. Static Variables
3. Local Variables

Instance Variable

A variable which is declared inside the class but outside the method is called instance variable. It is not declared as static.

members of the class: the methods and variables defined within a class are called members of the class.

Why the name is instance: Variables defined within a class are called instance variables because each instance of the class (that is, each object of the class) contains its own copy of these variables. Thus, the data for one object is separate and unique from the data for another.



Static variable

A variable that is declared as static is called static variable. It cannot be local.

There would only be one copy of each class variable per class, regardless of how many objects are created from it. Static variables store values for the variables in a common memory location. Because of this common location, if one object changes the value of a static variable, all objects of the same class are affected.

Local Variable

A variable which is declared inside the methods, constructors, or blocks is called local variable. Local variables are visible only within the declared method, constructor, or block.

Declaration & Initialization of Variables

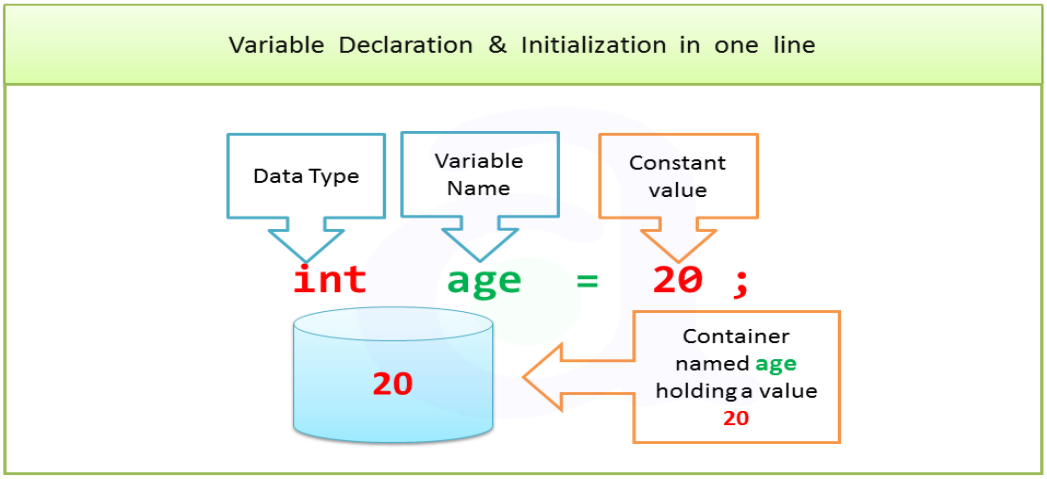
Variables are used to represent values that may be changed in the program. In Java, all variables must be declared before they can be used. The basic form of a variable declaration is shown here:

```
type identifier [ = value][, identifier [= value] ...] ;
```

- The identifier is the name of the variable. You can initialize the variable by specifying an equal sign and a value.
- Initialization expression must result in a value of the same (or compatible) type as that specified for the variable.
- To declare more than one variable of the specified type, use a comma-separated list.

example: int count; float x,y; double p1; byte b; char c1,c2,c3;

int age;
age=20; or
int age =20; float height=5.8;



int variable Declaration and Initialization

Comments in Java

In a program, comments take part in making the program become more human readable by placing the detail of code involved and proper use of comments makes maintenance easier and finding bugs easily. Comments are ignored by the compiler while compiling a code.

In Java there are three types of comments:

1. Single – line comments.
2. Multi – line comments.
3. Documentation comments.

Single line (or end-of line) comment:

It starts with a double slash symbol (//) and terminates at the end of the current line. The compiler ignores everything from // to the end of the line. For example:

```
// Calculate sum of two numbers
```

Multiline Comment:

Java programmer can use C/C++ comment style that begins with delimiter /* and ends with */. All the text written between the delimiter is ignored by the compiler.

```
/*calculate sum of two numbers
and it is a multiline comment */
```

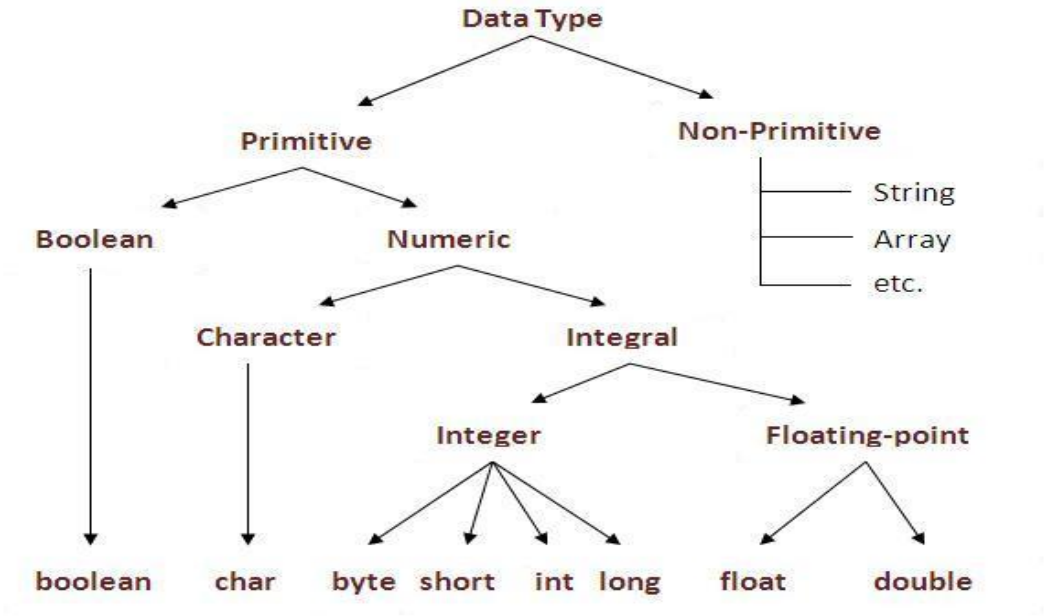
Documentation comments:

his comment style is new in Java. Such comments begin with delimiter /** and end with */.

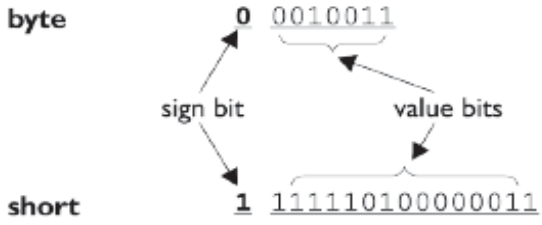
```
/** The text enclosed here will be part of program documentation */
```

DATA TYPES

Datatype is a special keyword used to allocate sufficient memory space for the data, in other words Data type is used for representing the data in main memory (RAM) of the computer.



The Sign bit for a byte



sign bit: 0 = positive
1 = negative

value bits:

byte: 7 bits can represent 2^7 or 128 different values:
0 thru 127 -or- –128 thru –1

short: 15 bits can represent 2^{15} or 32768 values:
0 thru 32767 -or- –32768 thru –1

Table below shows the primitive types with their sizes and ranges. Figure above shows

that with a byte, for example, there are 256 possible numbers (or 28). Half of these are negative, and half -1 are positive. The positive range is one less than the negative range because the number zero is stored as a positive binary number. We use the formula -2(bits-1) to calculate the negative range, and we use 2(bits-1)–1 for the positive range.

Type	Bits	Bytes	Minimum Range	Maximum Range
byte	8	1	-2^7	2^7-1
short	16	2	-2^{15}	$2^{15}-1$
int	32	4	-2^{31}	$2^{31}-1$
long	64	8	-2^{63}	$2^{63}-1$
float	32	4	n/a	n/a
double	64	8	n/a	n/a

Boolean type: For Boolean types there is not a range; a boolean can be only true or false.

Character type: The char type (a character) contains a single, 16-bit Unicode character. (2 bytes)

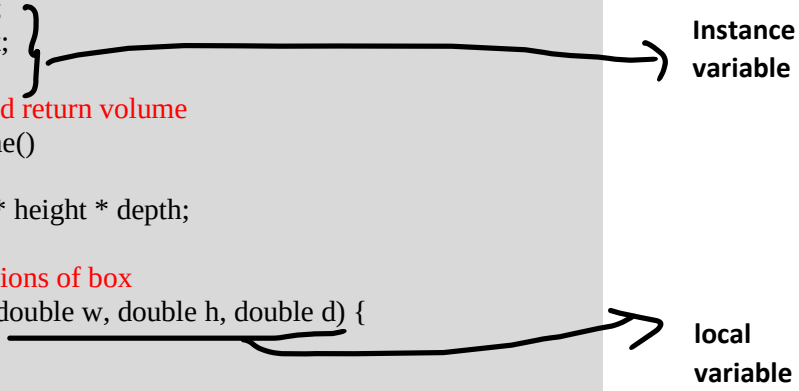
PRACTICE SESSION

1. Adding a Method to the Box Class

```
class Box
{
double width;
double height;
double depth;
// display volume of a box
void volume()
{
System.out.print("Volume is ");
System.out.println(width * height * depth);
}
}
class BoxDemo3
{
public static void main(String args[])
{
        Box mybox1 = new Box();
        Box mybox2 = new Box();
// assign values to mybox1's instance variables
        mybox1.width = 10;
        mybox1.height = 20;
        mybox1.depth = 15;
/* assign different values to mybox2's
instance variables */
        mybox2.width = 3;
        mybox2.height = 6;
        mybox2.depth = 9;
// display volume of first box
        mybox1.volume();
// display volume of second box
        mybox2.volume();
  }
 }
```

2. Adding a Method That Takes Parameters

```
class Box
{
double width;
double height;
double depth;
// compute and return volume
double volume()
{
return width * height * depth;
}
// sets dimensions of box
void setDim(double w, double h, double d) {
width = w;
height = h;
depth = d;
}
}
class BoxDemo5
{
public static void main(String args[])
{
        Box mybox1 = new Box();
        Box mybox2 = new Box();
        double vol;
// initialize each box
        mybox1.setDim(10, 20, 15);
        mybox2.setDim(3, 6, 9);
// get volume of first box
        vol = mybox1.volume();
        System.out.println("Volume is " + vol);
// get volume of second box
        vol = mybox2.volume();
        System.out.println("Volume is " + vol);
}
}
```



3. Demonstrating Static Variable

```
class StaticDemo
{
        static int a = 42;
        static int b = 99;
        static void callme()
        {
        System.out.println("a = " + a);
        }
}

class StaticByName
{
        public static void main(String args[])
        {
        StaticDemo.callme();
        System.out.println("b = " + StaticDemo.b);
        }
}
```

4. **Java Program to Demonstrate Char Primitive Data Type** [Identify and resolve issues in the given code snippet]

```
class DataTypeDemo
{
     public static void main(String args[])
     {
         char a = 'G';
         int i = 89;
         byte b = 4;

        // this will give error as number is
        // larger than byte range
        // byte b1 = 7888888955;

        short s = 56;

        // this will give error as number is
        // larger than short range
        // short s1 = 87878787878;

        // by default fraction value
        // is double in java
        double d = 4.355453532;

        // for float use 'f' as suffix as standard
        float f = 4.7333434f;

        System.out.println("char: " + a);
        System.out.println("integer: " + i);
        System.out.println("byte: " + b);
        System.out.println("short: " + s);
        System.out.println("float: " + f);
        System.out.println("double: " + d);
    }
}
```