

## \* Introducing Classes

Class defines a new datatype. This new type can be used to create objects of that type. Thus, a class is a "template" for an object, and an object is an "instance" of a class.

A class is declared by the use of class keywords.

General form of a class:-

```
class classname {  
    type instance-variable1;  
    type instance-variable2;  
  
    type method-name1(parameter-list) {  
        // body of method.  
    }  
}
```

The data, or variable, defined within a class are called Instance Variable.

When functions are <sup>defined</sup> ~~added~~ in a class body they are called methods.

Thus, the data for one object is separate and unique from the data ~~for~~ for another.

## 11 Declaring class ~~Box~~ Bon

```
class Bon {  
    double width;  
    double height;  
    double depth;  
}
```

To create a object of class Bon we need to write the following line in our main function:-

```
Bon b1 = new Bon();
```

Each time we create an instance of a class (Object), we are creating an object that contains its own copy of each instance variable defined by the class. To access these variables, we will use dot (.) operator.

e.g → `b1.width = 200;`

[dot operator can be used to access both instance variables and methods within an object]

X

What happens when we run the following command "javac filename.java"?

Step 1: Compilation → this is dependent from machine to machine

The compiler checks the java file for syntax error. If there are errors, they are reported back and compilation stops. (no class file created).



→ That's why java  
called platform  
independent.

If the syntax is correct, the compiler transcodes the human-readable Java codes into byte, a "machine independent intermediate format".

Step II: Output.


The generated byte code is stored in a .class file.

[If there are multiple class in a java file ~~there~~ then multiple .class file will also be created. When wanted to run the .class file through <sup>JVM</sup> java make sure the .class file having the main function.]

About Byte code and its execution in JVM.

Byte code is not tied to any specific machine or operating system. Byte code is executed by JVM, which is a platform-specific interpreter. Each OS has its own implementation of the JVM that translates byte code into machine specific instructions. This allows the same byte code to run ~~is~~ on different platform without modification. Java programs do not directly interact with the hardware or OS. Instead, the JVM handles the interaction, ensuring portability.

---



Difference b/w ~~new~~ reference and object.

Ans

Box b1;

~~This creates a new object of the r~~

→ This only declares a reference variable.

~~(Only reference)~~ (No memory is allocated).

→ No object is created in memory, this is an uninitialized reference variable that can be assigned to a object later.

Box b2 = new Box();

→ This creates a new object of the class in memory. (Both object and reference to the object are created).

→ Memory is allocated for the object and the constructor of the class is ~~created~~ called.

→ This points to NULL. <sup>access fields or called methods.</sup>

If we try to use it will result in a NULL POINTER EXCEPTION.

New operator.

The new operator dynamically allocates memory for an object during runtime.

classname objectname = new classname();

↓

Specifies the constructor of the class.



A constructor defines what occurs when the object of a class is created.

[If no constructor is specified in class body then java will automatically supply a default constructor]

Q What will happen when

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

→ A new Box object is created in memory. b1 holds the memory address of the new Box object.

→ The referenced variable b2 is assigned the same memory address as b1.

Now b1 and b2 point to the same Box object in memory. Any changes made to the object via b1 will be visible through b2 and vice versa.

Note:- Although b1 and b2 both refer to the same object, they are not linked in any other way. A subsequent assignment to b1 will simply unhook b1 from original object without affecting the object or affecting b2.

eg:- 

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

b2 still referring the prev. object.

[When a method uses an instance variable that is defined by its class, it does directly, without explicit reference to an object without use of the dot operator]

eg:- 

```
int vol() {  
    return height * length * width;  
}
```

---

Argument and Parameter → Placeholder /

↳ An argument is the actual value or data passed to a method ~~method~~ when it is called. variable defined in method signature

eg:- 

```
void greet(String name) {  
    S.O.P("Hello" + name);  
}
```

eg: 

```
greet("Ayan");
```

---

Constructors.

Still it would be simpler to have all of the setup done at the time of the object first created

eg:- 

```
void setValue() {  
    a = 20;  
}
```

obj1.setValue();

Why we need?

It can be tedious to initialize all the variables in a class each time an instance is created.

⇓ Potential soln

Yes, we can call methods to initialize the value of variable.

But still we have to write a ~~one~~ fucking line of code for it.



Java allows objects to initialize themselves when they are created. This initialization is done by Constructors.

A constructor initializes an object immediately upon creation. It has the same name as the class in which it resides and is syntactically similar to methods.

Once defined, the constructor is automatically called when the object is created, before the new operator completes.

```
eg:- Box() {  
    width = 10;  
    height = 10;  
    length = 10;  
}
```

### Types of Constructors:

#### (i) Default Constructors:

When we do not explicitly define a constructor for a class, then Java creates a default constructor for the class. When using this constructor, all ~~default constructor~~ non-initialized instance variables will have their default values, which are zero, null and false for numeric types, references types and boolean respectively.

## (i) Parameterized Constructors:-

```
Box(double w, double h, double l) {  
    width = w;  
    height = h;  
    length = l;  
}
```

In main func

```
Box b1 = new Box(20, 30, 40);
```

## (ii) Non Parameterized Constructors:-

```
Box() {  
    width = 10;  
    height = 20;  
    length = 30;  
}
```

## (iv) Copy Constructors:-

Creates a new object by copying the state of another object of same class.

```
class Box(Box b1) {  
    this.width = b1.width;  
    this.height = b1.height;  
    this.length = b1.length;  
}
```



## ① The "this" keyword

or constructor

Sometimes a method will need to refer to that object that invoked it. This can be used inside any method/constructor to refer to the current object.

⇓

Why using this?

→ Instance Variable  
Hiding

It is illegal in java to declare two local variables with the same name inside the same or enclosing scopes. Interestingly we can have local variables, including formal parameters to methods, which overlaps the names of the class instance variables. However when a local variable has the same name as an instance variable, the local variable hides the instance variable.

⇓

We can resolve this by using different names but we can do this by using this keyword.

Write a note on Garbage Collection in Java.

In C++ dynamically allocated objects must be manually released by use of a "delete" operator.

Java handles deallocation automatically. The technique that accomplish this is called garbage collection.

It works like this:-

When no references to an object exist, that object is assumed to be no longer needed, and the memory occupied by the object can be reclaimed. There is no need to explicitly destroy objects. Garbage collection only occurs sporadically during the execution of our program. It will not occur simply because one or more objects that are no longer use.