UNIT-II-Part-I

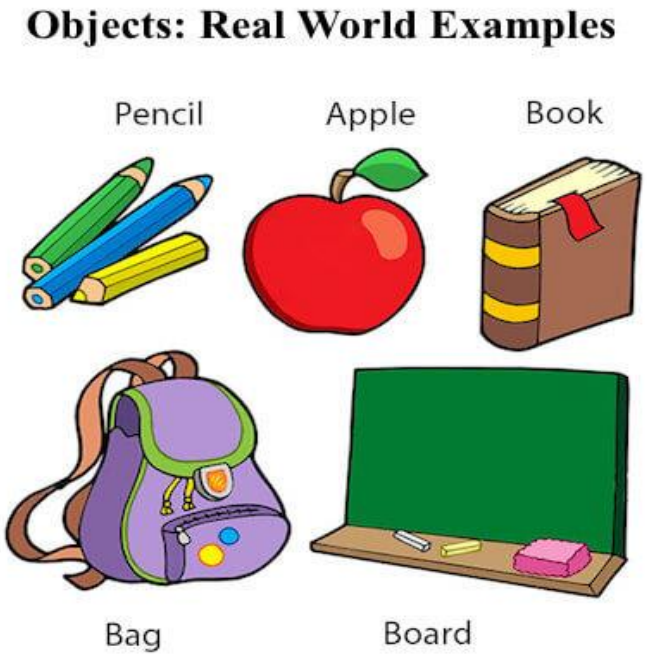
Classes:class fundamentals,Declaring objects,methods, constructors,this keywoed,garbage collection,overloading methods and constructors,recursion.

Objects and Classes in Java

An object in Java is the physical as well as a logical entity.

whereas, a class in Java is a logical entity only.

What is an object in Java



An entity that has state and behavior is known as an object e.g., chair, bike, marker, pen, table, car, etc. It can be physical or logical

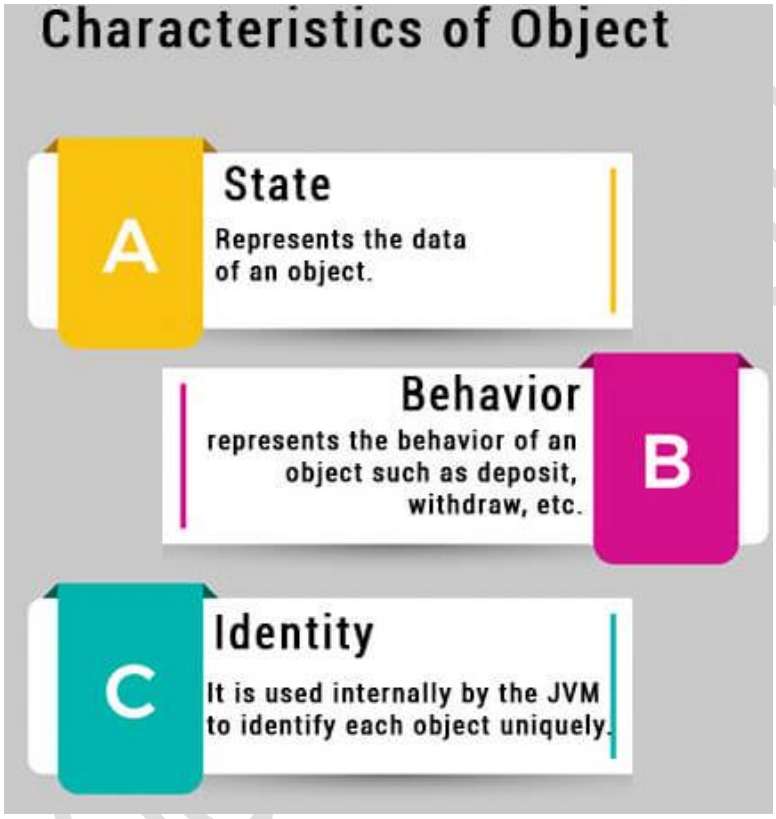
(tangible and intangible). The example of an intangible object is the banking system.

An object has three characteristics:

State: represents the data (value) of an object.

Behavior: represents the behavior (functionality) of an object such as deposit, withdraw, etc.

Identity: An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user



For Example, Pen is an object. Its name is Reynolds; color is white, known as its state. It is used to write, so writing is its behavior.

An object is an instance of a class. A class is a template or blueprint from which objects are created. So, an object is the instance(result) of a class.

Object Definitions:

An object is *a real-world entity*.

An object is *a runtime entity*.

The object is *an entity which has state and behavior*.

The object is *an instance of a class*.

What is a class in Java

A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity. It can't be physical.

A class in Java can contain:

Fields

Methods

Constructors

Syntax to declare a class:

```
class <class_name>
{
   field;
   method;
}
```

Object and Class Example: main within the class

In this example, we have created a Student class which has two data members id and name.

We are creating the object of the Student class by new keyword and printing the object's value.

Here, we are creating a main() method inside the class.

File: Student.java

```
class Student
{
  int id;//field or data member or instance variable
 String name;
  public static void main(String args[]){
  Student s1=new Student();//creating an object of Student
  System.out.println(s1.id);//accessing member through reference vari
able
  System.out.println(s1.name);
 }  }
```

Methods in Java

In general, a **method** is a way to perform some task.

Similarly, the **method in Java** is a collection of instructions that performs a specific task. It provides the reusability of code.

We can also easily modify code using **methods**.

What is a method in Java?

A **method** is a block of code or collection of statements or a set of code grouped together to perform a certain task or operation.

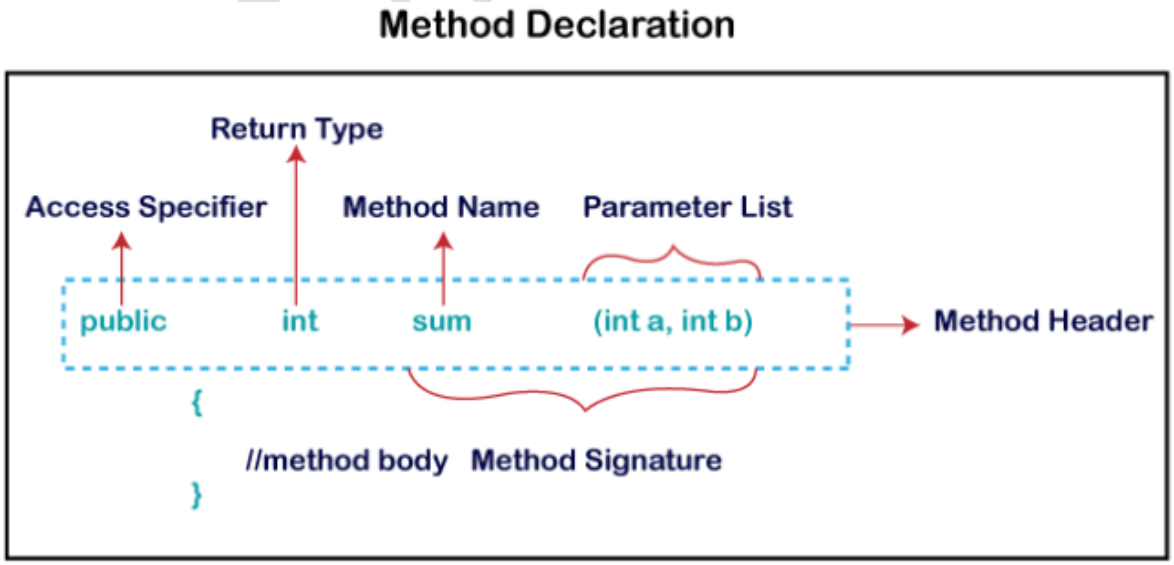
It is used to achieve the **reusability** of code.

We write a method once and use it many times.

We do not require to write code again and again.

Method Declaration

The method declaration provides information about method attributes, such as visibility, return-type, name, and arguments.



Method Signature: Every method has a method signature. It is a part of the method declaration. It includes the **method name** and **parameter list**.

Access Specifier: Access specifier or modifier is the access type of the method. It specifies the visibility of the method. Java provides **four** types of access specifier:

Public: The method is accessible by all classes when we use public specifier in our application.

Private: When we use a private access specifier, the method is accessible only in the classes in which it is defined.

Protected: When we use protected access specifier, the method is accessible within the same package or subclasses in a different package.

Default: When we do not use any access specifier in the method declaration, Java uses default access specifier by default. It is visible only from the same package only.

Return Type: Return type is a data type that the method returns. It may have a primitive data type, object, collection, void, etc.

If the method does not return anything, we use void keyword.

Method Name: It is a unique name that is used to define the name of a method. It must be corresponding to the functionality of the method.

Parameter List: It is the list of parameters separated by a comma and enclosed in the pair of parentheses. It contains the data type and variable name.

Method Body: It is a part of the method declaration. It contains all the actions to be performed. It is enclosed within the pair of curly braces.

```
public class Addition
{
public static void main(String[] args)
{
int a = 19;
int b = 5;
//method calling
int c = add(a, b);   //a and b are actual parameters
System.out.println("The sum of a and b is= " + c);
}
//user defined method
public static int add(int n1, int n2)   //n1 and n2 are formal parameter
s
{
int s;
s=n1+n2;
return s; //returning the sum
}
}
```

Constructors in Java

In Java, a constructor is a block of codes similar to the method.

It is called when an instance of the class is created.

At the time of calling constructor, memory for the object is allocated in the memory.

It is a special type of method which is used to initialize the object.

Every time an object is created using the new() keyword, at least one constructor is called.

It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.

There are two types of constructors in Java: no-arg constructor, and parameterized constructor.

Rules for creating Java constructor

There are two rules defined for the constructor.

Constructor name must be the same as its class name

A Constructor must have no explicit return type

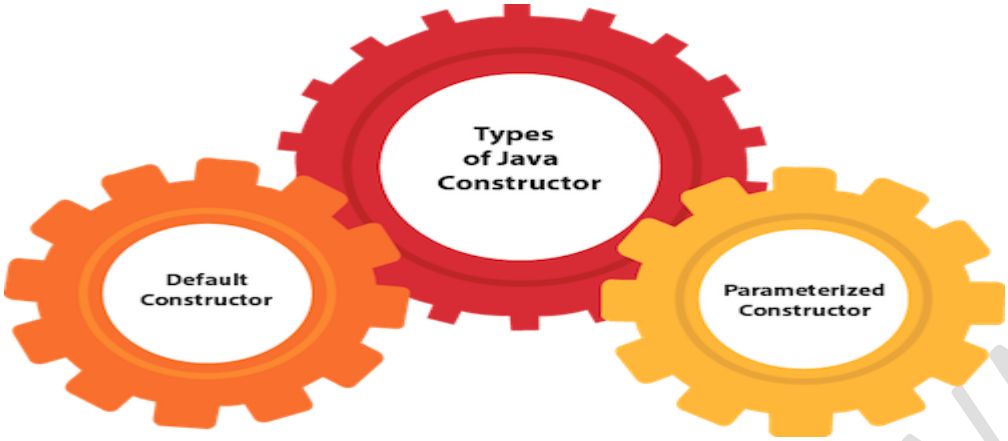
A Java constructor cannot be abstract, static, final, and synchronized

Types of Java constructors

There are two types of constructors in Java:

Default constructor (no-arg constructor)

Parameterized constructor



Java Default Constructor

A constructor is called "Default Constructor" when it doesn't have any parameter.

Syntax of default constructor:

<class_name>(){}

Example of default constructor

In this example, we are creating the no-arg constructor in the Bike class. It will be invoked at the time of object creation.

```
class Bike1
{
//creating a default constructor
Bike1()
{
System.out.println("Bike is created");
}
//main method
```

```
public static void main(String args[])
{
//calling a default constructor
Bike1 b=new Bike1();
} }
```

If there is no constructor in a class, compiler automatically creates a default constructor.

Java Parameterized Constructor

A constructor which has a specific number of parameters is called a parameterized constructor.

Why use the parameterized constructor?

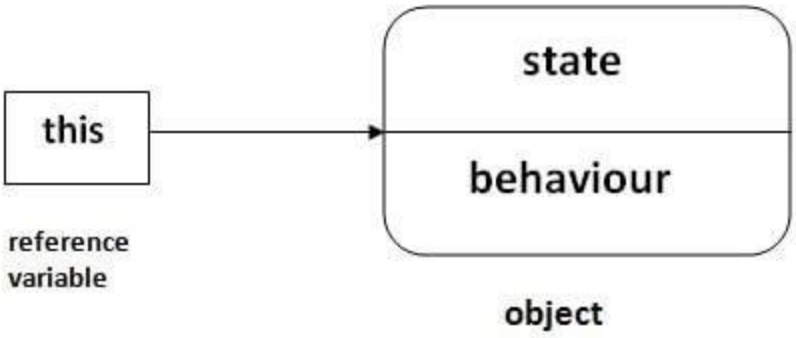
The parameterized constructor is used to provide different values to distinct objects.

```
class Student4
{
  int id;
  String name;
```

```
    //creating a parameterized constructor
    Student4(int i,String n){
    id = i;
    name = n;
    }
    //method to display the values
    void display(){System.out.println(id+" "+name);}
    public static void main(String args[])
    {
    //creating objects and passing values
    Student4 s1 = new Student4(111,"Karan");
    Student4 s2 = new Student4(222,"Aryan");
    //calling method to display the values of object
    s1.display();
    s2.display();
    }
}
```


this keyword in Java

There can be a lot of usage of **Java this keyword**. In Java, this is a **reference variable** that refers to the current object.



Usage of Java this keyword

Here is given the 6 usage of java this keyword.

this can be used to refer current class instance variable.

this can be used to invoke current class method (implicitly)

this() can be used to invoke current class constructor.

this can be passed as an argument in the method call.

this can be passed as argument in the constructor call.

this can be used to return the current class instance from the method.

1) this: to refer current class instance variable

The this keyword can be used to refer current class instance variable.

If there is ambiguity between the instance variables and parameters, this keyword resolves the problem of ambiguity.

```
class Student
{
int rollno;
String name;
float fee;
Student(int rollno,String name,float fee)
{
```

```
rollno=rollno;
name=name;
fee=fee;
}
void display(){System.out.println(rollno+" "+name+" "+fee);}
}
class TestThis1{
public static void main(String args[]){
Student s1=new Student(111,"ankit",5000f);
Student s2=new Student(112,"sumit",6000f);
s1.display();
s2.display();
}}
```

Output:

```
0 null 0.0
0 null 0.0
```

In the above example, parameters (formal arguments) and instance variables are same. So, we are using this keyword to distinguish local variable and instance variable.

Solution of the above problem by this keyword

```
class Student
{
int rollno;
```

```
String name;
float fee;
Student(int rollno,String name,float fee)
{
this.rollno=rollno;
this.name=name;
this.fee=fee;
}
void display(){System.out.println(rollno+" "+name+" "+fee);}
}

class TestThis2
{
public static void main(String args[])
{
Student s1=new Student(111,"ankit",5000f);
Student s2=new Student(112,"sumit",6000f);
s1.display();
s2.display();
}}
```

Output:

```
111 ankit 5000.0
112 sumit 6000.0
```

Method Overloading in Java

If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**.

If we have to perform only one operation, having same name of the methods increases the readability of the program.

Suppose you have to perform addition of the given numbers but there can be any number of arguments,

if you write the method such as a(int,int) for two parameters,

and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs.

So, we perform method overloading to figure out the program quickly.

Advantage of method overloading

Method overloading *increases the readability of the program*.

Different ways to overload the method

There are two ways to overload the method in java

By changing number of arguments

By changing the data type

1) Method Overloading: changing no. of arguments

In this example, we have created two methods, first add() method performs addition of two numbers and second add method performs addition of three numbers.

```
class Adder

{

static int add(int a,int b)
```

```
{return a+b;
}
static int add(int a,int b,int c)
{
return a+b+c;
}
}
class TestOverloading1
{
public static void main(String[] args)
{
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(11,11,11));
}}
```

2) Method Overloading: changing data type of arguments

In this example, we have created two methods that differs in data type.

The first add method receives two integer arguments and second add method receives two double arguments.

```
class Adder
{
static int add(int a, int b)
{
```

```
return a+b;
}
static double add(double a, double b)
{
return a+b;
}
}
class TestOverloading2
{
public static void main(String[] args)
{
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(12.3,12.6));
}}
```

Constructor Overloading in Java

In Java, a constructor is just like a method but without return type. It can also be overloaded like Java methods.

Constructor overloading in Java is a technique of having more than one constructor with different parameter lists. They are arranged in a way that each constructor performs a different task. They are differentiated by the compiler by the number of parameters in the list and their types.

Example of Constructor Overloading

```
//Java program to overload constructors
```

```
class Student5
{
   int id;
   String name;
   int age;
   //creating two arg constructor
   Student5(int i,String n)
   {
   id = i;
   name = n;
   }
   //creating three arg constructor
   Student5(int i,String n,int a)
   {
   id = i;
   name = n;
   age=a;
   }
   void display()
   {
    System.out.println(id+" "+name+" "+age);
   }
   public static void main(String args[])
```

```
    {
    Student5 s1 = new Student5(111,"Karan");
    Student5 s2 = new Student5(222,"Aryan",25);
    s1.display();
    s2.display();
   }
}
```

Recursion in Java

Recursion in java is a process in which a method calls itself continuously. A method in java that calls itself is called recursive method.

It makes the code compact but complex to understand.

Syntax:

```
returntype methodname()
{
//code to be executed
methodname();//calling same method
}
```

Example

```
//Factorial of a no
public class Recursion
```

```
{
   static int factorial(int n)
   {
        if (n == 1)
          return 1;
        else
          return(n * factorial(n-1));
   }

public static void main(String[] args)
{
System.out.println("Factorial of 5 is: "+factorial(5));
}
}
```

Fibonacci Series

```
public class Recursion4
 {
   static int n1=0,n2=1,n3=0;
    static void printFibo(int count)
    {
      if(count>0)
```

```
    {
        n3 = n1 + n2;
        n1 = n2;
        n2 = n3;
        System.out.print(" "+n3);
        printFibo(count-1);
      }
   }

public static void main(String[] args)
  {
   int count=15;
    System.out.print(n1+" "+n2);//printing 0 and 1
    printFibo(count-2);//n-2 because 2 numbers are already printed
}
}
```

Output:

0 1 1 2 3 5 8 13 21 34 55 89 144 233 377

Java Garbage Collection

In java, garbage means unreferenced objects.

Garbage Collection is process of reclaiming the runtime unused memory automatically. In other words, it is a way to destroy the unused objects.

To do so, we were using free() function in C language and delete() in C++. But, in java it is performed automatically. So, java provides better memory management.

Advantage of Garbage Collection

It makes java **memory efficient** because garbage collector removes the unreferenced objects from heap memory.

It is **automatically done** by the garbage collector(a part of JVM) so we don't need to make extra efforts.

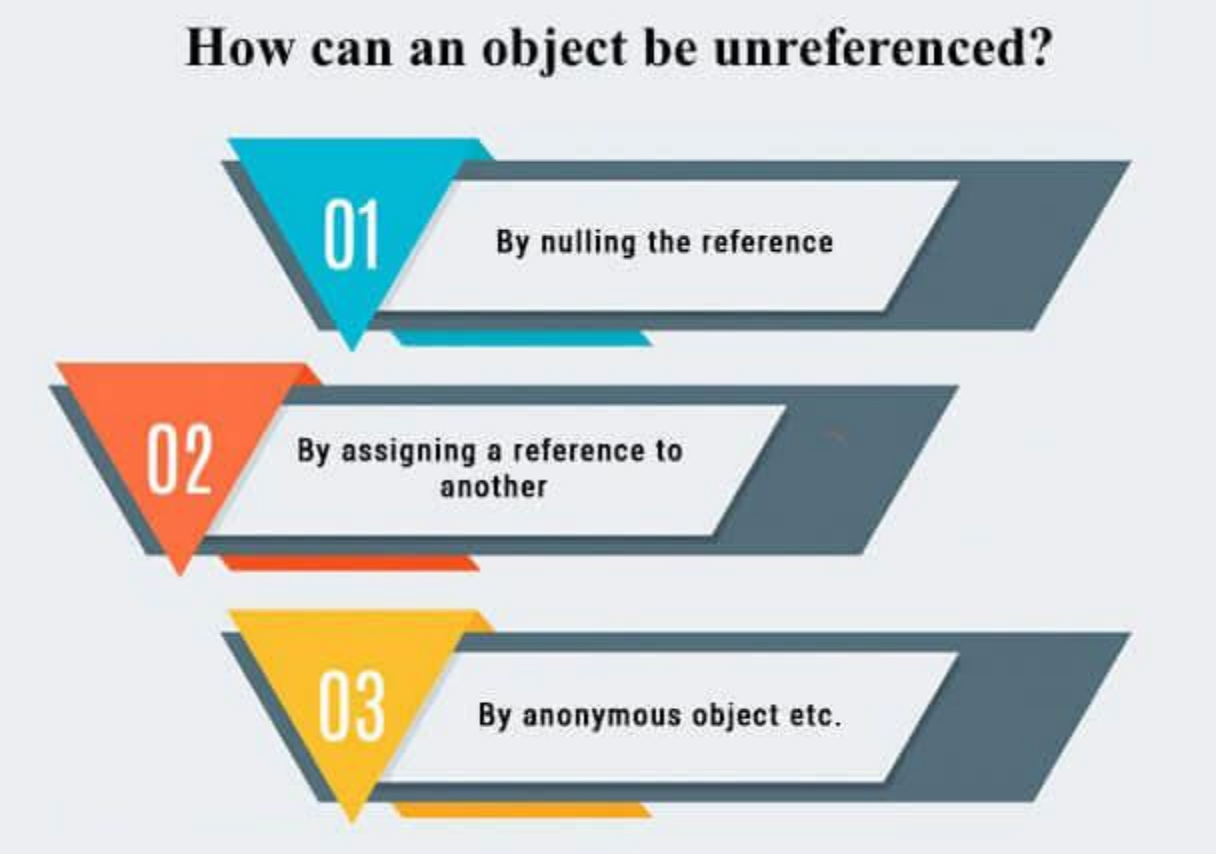
How can an object be unreferenced?

There are many ways:

By nulling the reference

By assigning a reference to another

By anonymous object etc.



1) By nulling a reference:

```
Employee e=new Employee();

e=null;
```

2) By assigning a reference to another:

```
Employee e1=new Employee();

Employee e2=new Employee();

e1=e2;//now the first object referred by e1 is available for garbage col lection
```

3) By anonymous object:

```
new Employee();
```

gc() method

The gc() method is used to invoke the garbage collector to perform cleanup processing. The gc() is found in System and Runtime classes.

```
public static void gc(){}
```

Example of garbage collection in java

```
public class TestGarbage1

{

 public void finalize()

{System.out.println("object is garbage collected");}

 public static void main(String args[])

{

  TestGarbage1 s1=new TestGarbage1();

  TestGarbage1 s2=new TestGarbage1();
```

```
  s1=null;
  s2=null;
  System.gc();
 }
}
```

Output:-

```
object is garbage collected
object is garbage collected
```