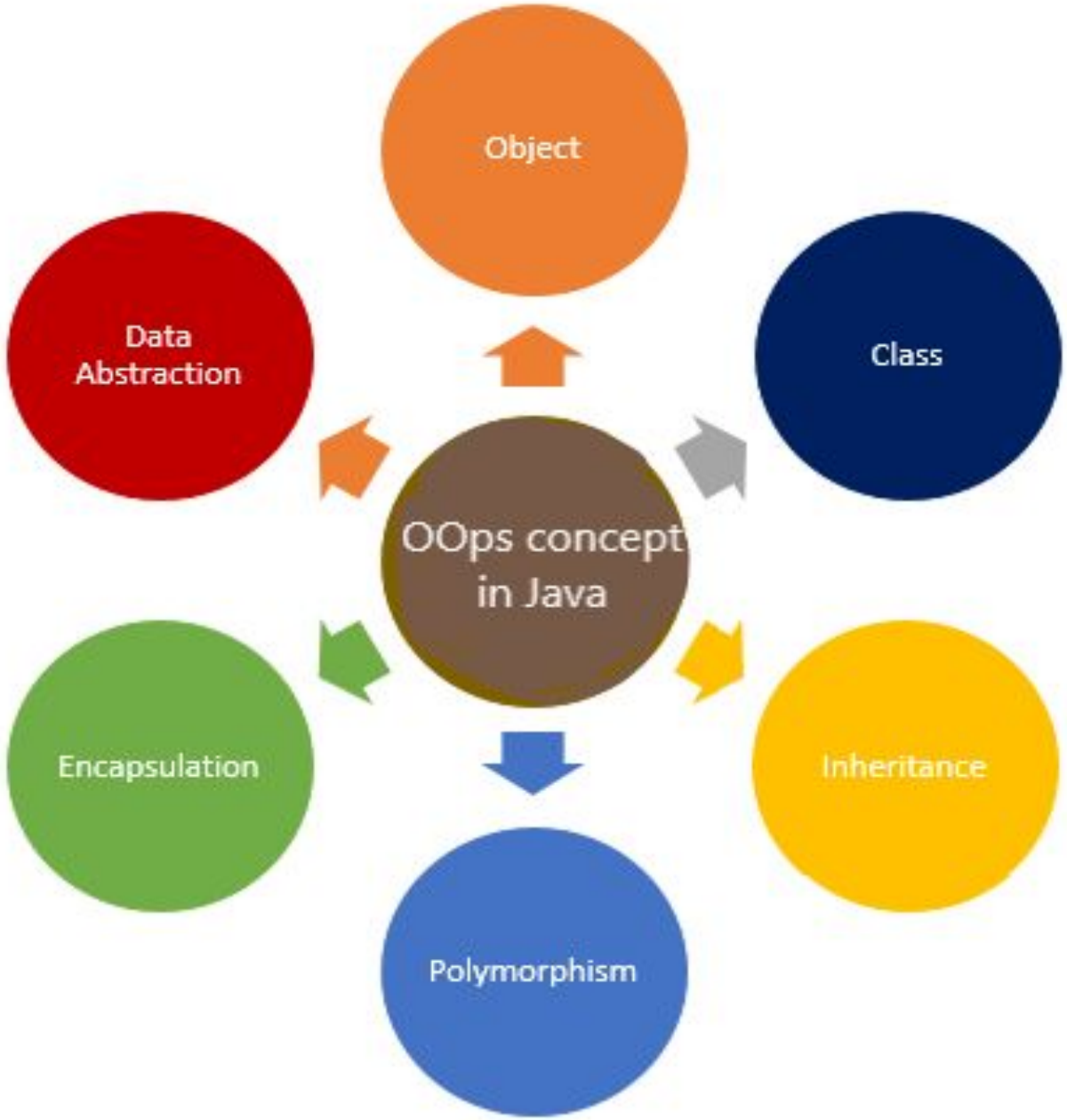
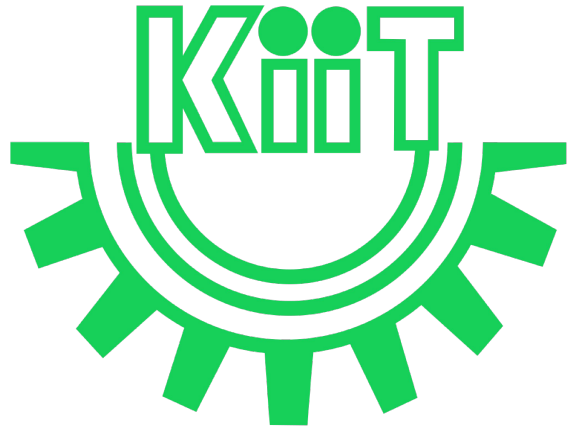


CS20004: Object Oriented Programming using Java

Lec-8

In this Discussion . . .

- Drawbacks of procedure oriented programming
- Classes & Objects
  - Class outline
  - Methods
- Taking inputs from user
  - Scanner Class
  - BufferedReader Class
- Constructor
- Garbage collection
- this keyword
- References



Procedural Programming

- decomposition around operations
- operation are divided into smaller operations
- Drawbacks:
  - data is given a second- class status when compared with operations
  - difficult to relate to the real world: no functions in real world
  - difficult to create new data types: reduces extensibility
  - programs are difficult to debug: little restriction to data access

Procedural Programming

- decomposition around operations
- operation are divided into smaller operations
- Drawbacks:
  - programs are hard to understand: many variables have global scope
  - programs are hard to reuse: data/functions are mutually dependent
  - little support for developing and comprehending really large programs
  - top- down development approach tends to produce monolithic programs

  [Monolithic:- an application that is made up of one large codebase that includes all the application components, such as the frontend code, backend code, and configuration files.]

Class Outline: Classes & Objects

- In object-oriented programming technique, we design a program using objects and classes.

  Object: Real world objects are things that have both:

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

  ➔ **Behavior:** represents the behavior (functionality) of an object.

  For Ex- (i) Dog is an object whose state can include: name, color, breed while its behavior can be sitting, barking, wagging tail, running, etc.

  (ii) Pen is an object, whose state can include: Its name (like Reynolds), color (like white), while its behavior can be writing.

Classes & Objects

- 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.

  For ex- A dog of specific breed for ex- German Shepherd, Akita, Golden retriever, etc. can be an object of the class Dog.

- An object is a real-world entity. An object is a runtime entity.
- The object is an entity which has state and behavior.

Java Class

- 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 which isn’t physical.
- Each concept we wish to describe in Java must be included inside a class
- A class defines a new data type, whose values are objects.

Java Class

- A class contains a name, several variable declarations (instance variables) and several method declarations. All are called members of the class.

- A class in Java can contain:
  - Fields
  - Methods
  - Constructors
  - Blocks
  - Nested class and interface

Syntax:

```
class class_name
{
    //variable declarations
    datatype instance_variable_1;
    datatype instance_variable_2;
    //Method declarations
    datatype method_1(parameter list)
    {
        //statements
    }
}
```

Java class: Instance variables & Methods

- Instance variables:
  - A variable which is created inside the class but outside the method is known as an instance variable.
  - Instance variable doesn't get memory at compile time.
  - It gets memory at runtime when an object or instance is created. That's why it is known as an instance variable.
- Method: In Java, a method is like a function which is used to expose the behavior of an object.
- Note:-

The new keyword is used to allocate memory at runtime. All objects get memory in Heap memory area.

Class & Object Example-I

```
//Java Program to illustrate how to define a class and fields

class Student
{
        //defining fields

        int id;
        String name;

        //creating main method inside the Student class
        public static void main(String args[])
        {
                //Creating an object or instance
                Student s1=new Student();
                //Printing values of the object
                System.out.println(s1.id);
                System.out.println(s1.name);
        }
}
```

Class & Object Example-I

```
//Java Program to illustrate how to define a class and fields

class Student
{
        //defining fields

        int id;
        String name;


        public static void main(String args[])
        {
                //Creating an object or instance
                Student s1=new Student();
                //Printing values of the object
                System.out.println(s1.id);
System.out.println(s1.name);
        }
}
```

Defining a Student class.

field or data member or instance variable

creating main method inside the Student class

creating an object of Student class

accessing member through reference variable

Class & Object Example-I

```
//Java Program to illustrate how to define a class and fields

class Student
{
        //defining fields

        int id;
        String name;


        public static void main(String args[])
        {
                //Creating an object or instance
                Student s1=new Student();
                //Printing values of the object
                System.out.println(s1.id);
System.out.println(s1.name);
        }
}
```

```
0
null
```

Class & Object Example-II

- In real time development, we create classes and use it from another class.
- We can have multiple classes in different Java files or single Java file. If we define multiple classes in a single Java source file, it is a good idea to save the file name with the class name which has main() method.

Class & Object Example-II

```
//Java Program to demonstrate having the main method in another class
//Creating Student class.
class Student
{
        int id;
        String name;
}
//Creating another class Newclasswithmain which contains the main method
class Newclasswithmain
{
        public static void main(String args[])
        {
                Student s1=new Student();
                System.out.println(s1.id);
                System.out.println(s1.name);
        }
}
```

```
0
null
```

Classes & Objects

- **Declaring Objects:**
  - *Everything in Java is an object*
  - Declare a variable of the class type:
    - Student s1;
    - s1 = new Student();
  - Allocates memory for a Student object and returns its address:
    - Student s1 = new Student();

Classes & Objects

- **Assigning Reference variables:**
  - Assignment copies address, not the actual value
    - Student s1 = new Student();
    - Student s2 = s1;
  - Here both variables, i.e., s1 & s2 point to the same object.

Methods

- General form of a method declaration:

```
datatype method_name(parameter list)
{
        return value;
}
```

- Classes declare methods to hide their internal data structures, as well as for their own internal use
- For ex- with respect to the Student class, defining a method display():

```
class Student
{
        void display()
        {
                System.out.println(“Student ID:”+id);
                System.out.println(“Student Name:”+name);
        }
}
```

Parameterized Methods

- Parameters increase generality and applicability of a method:

For ex- Method without parameters	For ex- Methods with Parameters

int square()
{
return 10*10;
}
	int square(int i)
	{
	return i*i;
	}

```
void display(int m, String n)
{
    id = m;
    name = n;
}
```

Objects Initialization

- There are 3 ways to initialize object in Java.
  - By reference variable [Will be covered in detail in upcoming classes]
  - By method
  - By constructor [Will be covered in detail in upcoming classes]

Objects Initialization by method

- Initializing an object means storing data into the object.

```
class Student
{
	int rollno;
	String name;
	void insertRecord(int r, String n)
	{
		rollno=r;
		name=n;
	}
	void displayInformation()
	{
		System.out.println(rollno+" "+name);
	}
}

class Objectinibymethod
{
	public static void main(String args[])
	{
	Student s1=new Student();
	Student s2=new Student();
	s1.insertRecord(111,"Karan");
	s2.insertRecord(222,"Aryan");
	s1.displayInformation();
	s2.displayInformation();
	}}
```

Objects Initialization by constructor

```
class Employee{
    int id;
    String name;
    float salary;
    void insert(int i, String n, float s) {
        id=i;
        name=n;
        salary=s;
    }
    void display(){System.out.println(id+" "+name+" "+salary);}
}
public class Objectinibyconstructor {
public static void main(String[] args) {
    Employee e1=new Employee();
    Employee e2=new Employee();
    Employee e3=new Employee();
    e1.insert(101,"ajeet",45000);
    e2.insert(102,"irfan",25000);
    e3.insert(103,"nakul",55000);
    e1.display();
    e2.display();
    e3.display();
}  }
```

How to get input from user in Java: Scanner Class

Java Scanner Class

- Java Scanner class allows the user to take input from the console.
- It belongs to java.util package.
- It is used to read the input of primitive types like int, double, long, short, float, and byte.

Syntax

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

How to get input from user in Java: Scanner Class

Java Scanner Class

Syntax

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

- The above statement creates a constructor of the Scanner class having System.in as an argument.
- It means it is going to read from the standard input stream of the program. The java.util package should be import while using Scanner class.
- It also converts the Bytes (from the input stream) into characters using the platform's default charset.

Methods of Java Scanner

- Java Scanner class provides the following methods to read different primitives types:

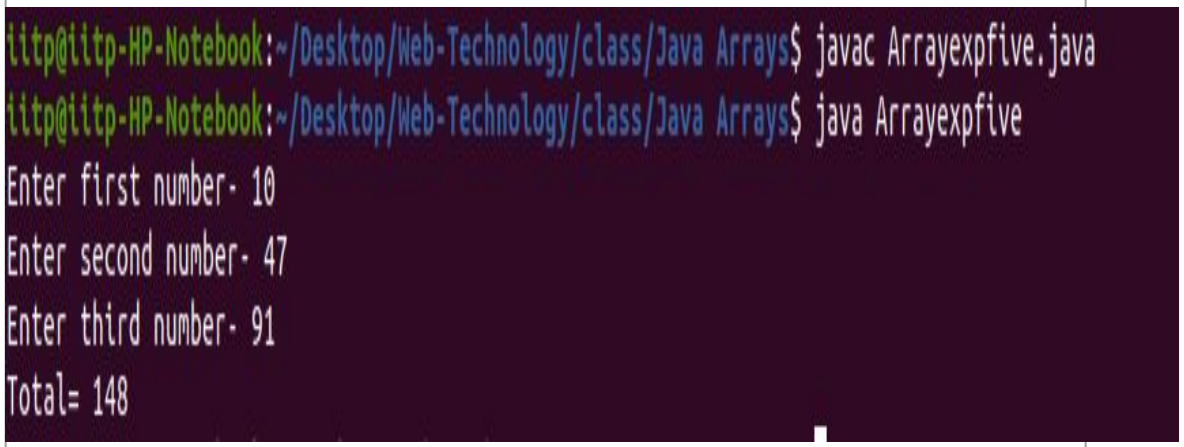
Method	Description
int nextInt()	It is used to scan the next token of the input as an integer.
float nextFloat()	It is used to scan the next token of the input as a float.
double nextDouble()	It is used to scan the next token of the input as a double.
byte nextByte()	It is used to scan the next token of the input as a byte.
String nextLine()	Advances this scanner past the current line.
boolean nextBoolean()	It is used to scan the next token of the input into a boolean value.
long nextLong()	It is used to scan the next token of the input as a long.
short nextShort()	It is used to scan the next token of the input as a Short.
BigInteger nextBigInteger()	It is used to scan the next token of the input as a BigInteger.
BigDecimal nextBigDecimal()	It is used to scan the next token of the input as a BigDecimal.

How to get input from user in Java: Scanner Class

```
import java.util.*;
class Arrayexpfive
{
public static void main(String[] args)
{
Scanner sc= new Scanner(System.in);    //System.in is
a standard input stream
System.out.print("Enter first number- ");
int a= sc.nextInt();
System.out.print("Enter second number- ");
int b= sc.nextInt();
System.out.print("Enter third number- ");
int c= sc.nextInt();
int d=a+b+c;
System.out.println("Total= " +d);
 } }
```

How to get input from user in Java: Scanner Class

```
import java.util.*;
class Arrayexpfive
{
public static void main(String[] args)
{
Scanner sc= new Scanner(System.in);    //System.in is
a standard input stream
System.out.print("Enter first number- ");
int a= sc.nextInt();
System.out.print("Enter second number- ");
int b= sc.nextInt();
System.out.print("Enter third number- ");
int c= sc.nextInt();
int d=a+b+c;
System.out.println("Total= " +d);  }  }
```



BufferedReader Class & its different Methods

- Java BufferedReader class is used to read the text from a character-based input stream.
- It can be used to read data line by line by readLine() method. It makes the performance fast. It inherits Reader class.

Method	Description
int read()	It is used for reading a single character.
int read(char[] cbuf, int off, int len)	It is used for reading characters into a portion of an array.
boolean markSupported()	It is used to test the input stream support for the mark and reset method.
String readLine()	It is used for reading a line of text.
boolean ready()	It is used to test whether the input stream is ready to be read.
long skip(long n)	It is used for skipping the characters.
void reset()	It repositions the stream at a position the mark method was last called on this input stream.
void mark(int readAheadLimit)	It is used for marking the present position in a stream.
void close()	It closes the input stream and releases any of the system resources associated with the stream.

BufferedReader

- Syntax to create an instance of the BufferedReader class to take input from the user in Java:

```
// System.in refers to the standard input, which is the keyboard by default.

BufferedReader input = new BufferedReader(new InputStreamReader(System.in));
```

Read Input as a String using a BufferedReader in Java

```
import java.io.BufferedReader;
import java.io.InputStreamReader;

public class BufferedReaderExample {

  public static void main(String args[]){
    // Instantiate the BufferedReader class
    BufferedReader input = new BufferedReader(new
InputStreamReader(System.in));
    // Prompt the user for input
    System.out.println("Enter your name: ");
    try {
      // Read the name as a String
      String name = input.readLine();
      // Greet the user by name
      System.out.println("Hello " + name + ".");
    }
    catch (Exception e) {
      // print the exception to the console
      System.out.println("Something went wrong!");
    }
  }
}
```

Constructor

- In Java, a constructor is a block which is syntactically 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 instance variables of an object.

Constructor

- Every time an object is created using the new() keyword, at least one constructor is called.
- It is called immediately after the object is created but before the new operator completes.
- When any class has no constructor, the Java compiler provides a default constructor to automatically initializes all its instance variables with their default values.

Constructor Types & Rules for creating Constructors

- Types of constructors in Java:
  - Default constructor (No argument constructor), and
  - Parameterized constructor.
- Rules for creating Java constructor
  - Constructor name must be the same as its class name
  - A Constructor must have no explicit return type -> the default return type of a class constructor is the same class
  - A Java constructor cannot be abstract, static, final, and synchronized

Note: We can use access modifiers while declaring a constructor. It controls the object creation. In other words, we can have private, protected, public or default constructor in Java.

Java Default Constructor

- A constructor without any parameters is called "Default Constructor”.

Syntax

```
class_name( )
{


}
```

Java Default Constructor Example

- In the following example, we are creating the default constructor in the Defconstructexp class. It will be invoked at the time of object creation.

```
class Defconstructexp
{
        //creating a default constructor
        Defconstructexp()
        {
                System.out.println("Inside the default constructor");
        }

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

```
Inside the default constructor
```

Java Default Constructor Example

- The default constructor is used to provide the default values to the object like 0, null, etc., depending on the type.

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

	void display()
	{
		System.out.println(id+" "+name);
	}
	public static void main(String args[])
	{
		//creating objects
		Defconstructurstudexp s1=new Defconstructurstudexp();
		Defconstructurstudexp s2=new Defconstructurstudexp();
		//displaying values of the object
		s1.display();
		s2.display();
	}
}
```

Java Default Constructor Example

- The default constructor is used to provide the default values to the object like 0, null, etc., depending on the type.
- For the below class, we did not create any constructor so compiler provides you a default constructor. Here 0 and null values are provided by that default constructor.

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

	void display()
	{
		System.out.println(id+" "+name);
	}
	public static void main(String args[])
	{
		//creating objects
		Defconstructurstudexp s1=new Defconstructurstudexp();
		Defconstructurstudexp s2=new Defconstructurstudexp();
		//displaying values of the object
		s1.display();
		s2.display();
	}
}
```

```
0 null
0 null
```

Java Parameterized Constructor

- A constructor which has a specific number of parameters is called a parameterized constructor.
- **Need:-** The parameterized constructor is used to provide different values to distinct objects. However, we can also provide the same values too.

Java Parameterized Constructor Example

```
//Java Program to demonstrate the use of the parameterized constructor.
class Paramconstexp
{
    int id;
    String name;
    //creating a parameterized constructor
    Paramconstexp(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
    Paramconstexp s1 = new Paramconstexp(111,"Karan");
    Paramconstexp s2 = new Paramconstexp(222,"Aryan");
    //calling method to display the values of object
    s1.display();
    s2.display();
   }
}
```

Java Parameterized Constructor Example

```
//Java Program to demonstrate the use of the parameterized constructor.
class Paramconstexp
{
    int id;
    String name;
    //creating a parameterized constructor
    Paramconstexp(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
    Paramconstexp s1 = new Paramconstexp(111,"Karan");
    Paramconstexp s2 = new Paramconstexp(222,"Aryan");
    //calling method to display the values of object
    s1.display();
    s2.display();
   }
}
```

```
111 Karan
222 Aryan
```

finalize() method

- finalize method is invoked just before the object is destroyed
- finalize() method in Java is used to release all the resources used by the object before it is deleted/destroyed by the Garbage collector.
- finalize is not a reserved keyword, it's a method.
- Once the clean-up activity is done by the finalize() method, garbage collector immediately destroys the Java object.

finalize() method

- Java Virtual Machine(JVM) permits invoking of finalize() method only once per object.
- Implemented when the usual way of removing objects from memory is insufficient, and some special actions has to be carried out
- Implemented inside a class as:

protected void finalize() {.....}

- finalize() is a method of the Object class in Java. The finalize() method is a non-static and protected method of java.lang.Object class.

finalize() method

- In Java, the Object class is superclass of all Java classes. Being an object class method finalize() method is available for every class in Java.
- Hence, Garbage Collector can call finalize() method on any Java object for clean-up activity.
- Once object is finalized JVM sets a flag in the object header to say that it has been finalized, and won't finalize it again. If user tries to use finalize() method for the same object twice, JVM ignores it.

Garbage collector

- Java considers unreferenced objects that are not being used by any program execution or objects that are no longer needed, as garbage.
- Garbage collection is the process of destroying unused objects and reclaiming the unused runtime memory automatically. By doing this memory is managed efficiently by Java.
- Garbage collection is carried out by the garbage collector.

Garbage collector

- The garbage collector is a part of Java Virtual Machine(JVM).
- Garbage collector checks the heap memory, where all the objects are stored by JVM, looking for unreferenced objects that are no more needed. And automatically destroys those objects.
- Garbage collector calls finalize() method for clean up activity before destroying the object.
- Java does garbage collection automatically; there is no need to do it explicitly, unlike other programming languages.

Garbage collector

- The garbage collector in Java can be called explicitly using the following method:

```
System.gc();
```

  - System.gc() is a method in Java that invokes garbage collector which will destroy the unreferenced objects.
  - System.gc() calls finalize() method only once for each object.

How does finalize() method work with Garbage Collection?

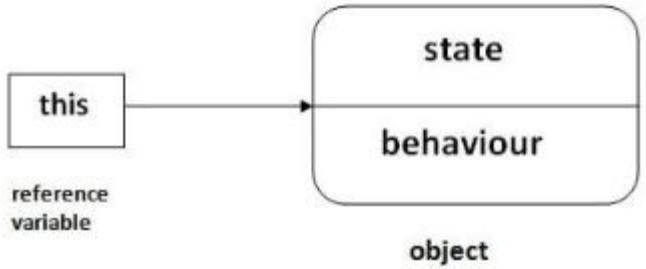
- The JVM calls the garbage collector to delete unreferenced objects. After determining the objects that have no links or references, it calls the finalize() method which will perform the clean activity and the garbage collector destroys the object.

```
public class Demo
{
    public static void main(String[] args)
    {
      String a = "Hello World!";
      a = null; //unreferencing string object
    }
}
```

- For the program, when the String object a holds value Hello World! it has a reference to an object of the String class.
- But, when it holds a null value it does not have any reference.
- Hence, it is eligible for garbage collection. The garbage collector calls finalize() method to perform clean-up before destroying the object.

this keyword

- In Java, this is a reference variable that refers to the current object.



- Usages of 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.
- In order to find out the importance of this keyword, let us first understand what are the problems which can occur, if we do not use it.

1) this: to refer current class instance variable

-

```
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 This1
{
	public static void main(String args[ ])
	{
		Student s1=new Student(111,"ankit",5000f);
		Student s2=new Student(112,"sumit",6000f);
		s1.display();
		s2.display();
	}
}
```

1) this: to refer current class instance variable

-

```
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 This1
{
	public static void main(String args[ ])
	{
		Student s1=new Student(111,"ankit",5000f);
		Student s2=new Student(112,"sumit",6000f);
		s1.display();
		s2.display();
	}
}
```

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

1) this: to refer current class instance variable

```
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 This2
{
        public static void main(String args[ ])
        {
                Student s1=new Student(111,"ankit",5000f);
                Student s2=new Student(112,"sumit",6000f);
                s1.display();
                s2.display();
        }
}
```

The example on left side, has local variable parameters (i.e., formal arguments) and instance variables same.

Thus, we need to use this keyword to distinguish local variable and instance variable.

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

1) this: to refer current class instance variable

- However, if the local variables(formal arguments) and instance variables are different, there is no need to use this keyword.

```
class Student
{
        int rollno;
        String name;
        float fee;
        Student(int roll,String n,float f)
        {
                rollno=roll;
                name=n;
                fee=f;
        }
        void display()
        {
                System.out.println(rollno+" "+name+" "+fee);
        }
}
class This3
{
        public static void main(String args[])
        {
                Student s1=new Student(111,"ankit",5000f);
                Student s2=new Student(112,"sumit",6000f);
                s1.display();
                s2.display();
        }
}
```

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

2) this: to invoke current class method

- We can invoke the method of the current class by using the this keyword.
- If we don't use the this keyword, compiler automatically adds this keyword while invoking the method.

```
class Delhi
{
	void m()
	{
		System.out.println("Chole Bhature");
	}
	void n()
	{
		System.out.println("Tikki");
		//m();//same as this.m()
		this.m();
	}
}
class Meerut
{
	public static void main(String args[])
	{
		Delhi de=new Delhi();
		de.n();
	}
}
```

2) this: to invoke current class method

- We can invoke the method of the current class by using the this keyword.
- If we don't use the this keyword, compiler automatically adds this keyword while invoking the method.

```
class Delhi
{
	void m()
	{
		System.out.println("Chole Bhature");
	}
	void n()
	{
		System.out.println("Tikki");
		//m();//same as this.m()
		this.m();
	}
}
class Meerut
{
	public static void main(String args[])
	{
		Delhi del=new Delhi();
		del.n();
	}
}
```

```
Tikki
Chole Bhature
```

3) this() : to invoke current class constructor

- The this() constructor call can be used to invoke the current class constructor. It is used to reuse the constructor. In other words, it is used for constructor chaining.

```
class A
{
	A()
	{
		System.out.println("hello a");
	}
	A(int x)
	{
		this();
		System.out.println(x);
	}
}
class This4
{
	public static void main(String args[])
	{
		A a=new A(10);
	}
}
```

3) this() : to invoke current class constructor

- The program provided below helps in calling the default constructor from parameterized constructor:

```
class A
{
	A()
	{
		System.out.println("hello a");
	}
	A(int x)
	{
		this();
		System.out.println(x);
	}
}
class This4
{
	public static void main(String args[])
	{
		A a=new A(10);
	}
}
```

```
hello a
10
```

3) this() : to invoke current class constructor

- The program provided below helps in calling the parameterized constructor from default constructor:

```
class A
{
	A()
	{
		this(5);
		System.out.println("hello a");
	}
	A(int x)
	{
		System.out.println(x);
	}
}
class This5
{
	public static void main(String args[])
	{
		A a=new A();
	}
}
```

```
5
hello a
```

3) this() : to invoke current class constructor

- The this() constructor call should be used to reuse the constructor from the constructor. It maintains the chain between the constructors i.e. it is used for constructor chaining.

```
class Student{
int rollno;
String name,course;
float fee;
Student(int rollno,String name,String course){
this.rollno=rollno;
this.name=name;
this.course=course;
}
Student(int rollno,String name,String course,float fee){
this.fee=fee;
this(rollno,name,course);
}
void display(){System.out.println(rollno+" "+name+" "+course+"
"+fee);}
}
class This6{
public static void main(String args[]){
Student s1=new Student(111,"ankit","java");
Student s2=new Student(112,"sumit","java",6000f);
s1.display();
s2.display();
}}
```

3) this() : to invoke current class constructor

- The this() constructor call should be used to reuse the constructor from the constructor. It maintains the chain between the constructors i.e. it is used for constructor chaining.

```
class Student{
int rollno;
String name,course;
float fee;
Student(int rollno,String name,String course){
this.rollno=rollno;
this.name=name;
this.course=course;
}
Student(int rollno,String name,String course,float fee){
this.fee=fee;
this(rollno,name,course);
}
void display(){System.out.println(rollno+" "+name+" "+course+"
"+fee);}
}
class This6{
public static void main(String args[]){
Student s1=new Student(111,"ankit","java");
Student s2=new Student(112,"sumit","java",6000f);
s1.display();
s2.display();
}}
```

```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Objects and Classes$ javac This6.java
This6.java:12: error: call to this must be first statement in constructor
this(rollno,name,course);
    ^
1 error
```

3) this() : to invoke current class constructor

- The this() constructor call should be used to reuse the constructor from the constructor. It maintains the chain between the constructors i.e. it is used for constructor chaining. Call to this() must be the first statement in constructor.

```
class Student{
int rollno;
String name,course;
float fee;
Student(int rollno,String name,String course){
this.rollno=rollno;
this.name=name;
this.course=course;
}
Student(int rollno,String name,String course,float fee){
this(rollno,name,course);
this.fee=fee;
}
void display(){System.out.println(rollno+" "+name+" "+course+"
"+fee);}
}
class This6{
public static void main(String args[]){
Student s1=new Student(111,"ankit","java");
Student s2=new Student(112,"sumit","java",6000f);
s1.display();
s2.display();
}}
```

```
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Objects and Classes$ javac This6.java
iitp@iitp-HP-Notebook:~/Desktop/Web-Technology/class/Java-Objects and Classes$ java This6
111 ankit java 0.0
112 sumit java 6000.0
```

4) this: to pass as an argument in the method

- The this keyword can also be passed as an argument in the method. It is mainly used in the event handling.

```
class S2
{
        void m(S2 obj)
        {
                System.out.println("method is invoked");
        }
        void p()
        {
                m(this);
        }
        public static void main(String args[])
        {
                S2 s1 = new S2();
                s1.p();
        }
}
```

```
method is invoked
```

Application of this that can be passed as an argument: In event handling (or) in a situation where we have to provide reference of a class to another one. It is used to reuse one object in many methods.

5) this: to pass as argument in the constructor call

- We can pass the this keyword in the constructor also. It is useful if we have to use one object in multiple classes.

```
class B{
  A4 obj;
  B(A4 obj){
    this.obj=obj;
  }
  void display(){
    System.out.println(obj.data);//using data member of A4 class
  }
}

class A4{
  int data=10;
  A4(){
   B b=new B(this);
   b.display();
  }
  public static void main(String args[]){
   A4 a=new A4();
  }
}
```

5) this: to pass as argument in the constructor call

- We can pass the this keyword in the constructor also. It is useful if we have to use one object in multiple classes.

```
class B{
  A4 obj;
  B(A4 obj){
    this.obj=obj;
  }
  void display(){
    System.out.println(obj.data);//using data member of A4 class
  }
}

class A4{
  int data=10;
  A4(){
   B b=new B(this);
   b.display();
  }
  public static void main(String args[]){
   A4 a=new A4();
  }
}
```

10

6) this: keyword can be used to return current class instance

- We can return this keyword as an statement from the method. In such case, return type of the method must be the class type (non-primitive).

Syntax:

```
return_type method_name(){
return this;
}
```

6) this: keyword can be used to return current class instance

- We can return this keyword as an statement from the method. In such case, return type of the method must be the class type (non-primitive).

```
class A
{
        A getA()
        {
                return this;
        }
        void msg()
        {
                System.out.println("Hello java");
        }
}
class Test1
{
        public static void main(String args[])
        {
                new A().getA().msg();
        }
}
```

Hello java

References

1. https://www.geeksforgeeks.org/type-conversion-java-examples/
2. https://www.javatpoint.com/scope-of-variables-in-java
3. https://i.stack.imgur.com/Ij3vJ.png
4. https://www.javatpoint.com/control-flow-in-java
5. https://www.scaler.com/topics/expression-in-java/
6.