1)Define system.out.println ?

Ans. System. out. println is a method in Java that prints a message to the standard output (typically the console) and appends a newline character. It's widely used to display messages, data, and the results of operations during the execution of a program.

1. **class** Demo
2. {
3. **public** **static** **void** main(String args[])
4. {
5. System.out.print("Hello!");
6. System.out.print("Java");
7. } }

**Output**

Hello! Java

2)Explain java thread life cycle?

Ans. In Java, a thread always exists in any one of the following states. These states are:

1. New
2. Active
3. Blocked / Waiting
4. Timed Waiting
5. Terminated

**New:** Whenever a new thread is created, it is always in the new state. For a thread in the new state, the code has not been run yet and thus has not begun its execution.

**Active:** When a thread invokes the start() method, it moves from the new state to the active state. The active state contains two states within it: one is **runnable**, and the other is **running**.

* **Runnable:** A thread, that is ready to run is then moved to the runnable state. In the runnable state, the thread may be running or may be ready to run at any given instant of time. It is the duty of the thread scheduler to provide the thread time to run, i.e., moving the thread the running state.  
  A program implementing multithreading acquires a fixed slice of time to each individual thread. Each and every thread runs for a short span of time and when that allocated time slice is over, the thread voluntarily gives up the CPU to the other thread, so that the other threads can also run for their slice of time. Whenever such a scenario occurs, all those threads that are willing to run, waiting for their turn to run, lie in the runnable state. In the runnable state, there is a queue where the threads lie.
* **Running:** When the thread gets the CPU, it moves from the runnable to the running state. Generally, the most common change in the state of a thread is from runnable to running and again back to runnable.

**Blocked or Waiting:** Whenever a thread is inactive for a span of time (not permanently) then, either the thread is in the blocked state or is in the waiting state.

**Timed Waiting:** Sometimes, waiting for leads to starvation. For example, a thread (its name is A) has entered the critical section of a code and is not willing to leave that critical section. In such a scenario, another thread (its name is B) has to wait forever, which leads to starvation. To avoid such scenario, a timed waiting state is given to thread B. Thus, thread lies in the waiting state for a specific span of time, and not forever. A real example of timed waiting is when we invoke the sleep() method on a specific thread. The sleep() method puts the thread in the timed wait state. After the time runs out, the thread wakes up and start its execution from when it has left earlier.

**Terminated:** A thread reaches the termination state because of the following reasons:

* When a thread has finished its job, then it exists or terminates normally.
* **Abnormal termination:** It occurs when some unusual events such as an unhandled exception or segmentation fault.

A terminated thread means the thread is no more in the system. In other words, the thread is dead, and there is no way one can respawn (active after kill) the dead thread.

3)What is garbage collection in java?How it is performed automatically.

Ans. Garbage collection in Java is the process by which Java programs perform automatic memory management. Java programs compile to bytecode that can be run on a Java Virtual Machine, or JVM for short. When Java programs run on the JVM, objects are created on the heap, which is a portion of memory dedicated to the program. Eventually, some objects will no longer be needed. The garbage collector finds these unused objects and deletes them to free up memory

How Does Garbage Collection in Java works?

Java garbage collection is an automatic process. Automatic garbage collection is the process of looking at heap memory, identifying which objects are in use and which are not, and deleting the unused objects. An in-use object, or a referenced object, means that some part of your program still maintains a pointer to that object. An unused or unreferenced object is no longer referenced by any part of your program. So the memory used by an unreferenced object can be reclaimed. The programmer does not need to mark objects to be deleted explicitly. The garbage collection implementation lives in the JVM.

Types of Activities in Java Garbage Collection

Two types of garbage collection activity usually happen in Java. These are:

* **\***It is said to have occurred when unreachable objects in the young generation heap memory are removed.
* It is said to have occurred when the objects that survived the minor garbage collection are copied into the old generation or permanent generation heap memory are removed. When compared to the young generation, garbage collection happens less frequently in the old generation.

4)Explain how with an example java program create and throw custom exception?

Ans. In Java, we can create our own exceptions that are derived classes of the Exception class. Creating our own Exception is known as custom exception or user-defined exception. Basically, Java custom exceptions are used to customize the exception according to user need.

Let's see a simple example of Java custom exception. In the following code, constructor of InvalidAgeException takes a string as an argument. This string is passed to constructor of parent class Exception using the super() method. Also the constructor of Exception class can be called without using a parameter and calling super() method is not mandatory.

1. // class representing custom exception
2. **class** MyCustomException **extends** Exception
3. {
5. }
7. // class that uses custom exception MyCustomException
8. **public** **class** TestCustomException2
9. {
10. // main method
11. **public** **static** **void** main(String args[])
12. {
13. **try**
14. {
15. // throw an object of user defined exception
16. **throw** **new** MyCustomException();
17. }
18. **catch** (MyCustomException ex)
19. {
20. System.out.println("Caught the exception");
21. System.out.println(ex.getMessage());
22. }
24. System.out.println("rest of the code...");
25. }
26. }

5)Explain in brief different type of access specifiers?

Ans. **Access specifiers** are the keywords like **"public"**, **"protected"**, **"default"** and **"private"** which has its special meaning in java.

It defines the access scope of the variable, methods, and classes and here the access scope means the area or space where a variable or classes or methods are accessible.

### Types of access specifiers

In java, there are four types of access specifiers and the name of these access specifiers are given below:

1. public access specifiers
2. protected access specifiers
3. default access specifiers
4. private access specifiers

Now, with the help of example, we will describe each access specifiers one by one in java.

### 1) Public access specifiers

* **"public"** is the keyword which is introduced in java.
* The access scope of the **"public"** is everywhere like in all classes and methods as well.
* If we prefixed **"public"** keyword with any class, variable or method then it can be accessed by any class or methods.

### 2) protected access specifiers

* **"protected"** is the keyword which is introduced in java.
* The access scope of the **"protected"** is not everywhere and it is accessible in the same class or its child class or in all those classes which are defined in the same package.
* If we prefixed **"protected"** keyword with any class, variable or method then it can be accessed by the same class or its child classes or all the classes which are defined in the same package.

### 3) default access specifiers

* **"default"** is the keyword which is introduced in java.
* The access scope of the **"default"** is not everywhere.
* It is not mandated to prefixed **"default"** keyword with any class, variable or method because by default class, variable or method is default public in java and it can be accessed by all those classes which are defined in same package only.

### 3) default access specifiers

* **"default"** is the keyword which is introduced in java.
* The access scope of the **"default"** is not everywhere.
* It is not mandated to prefixed **"default"** keyword with any class, variable or method because by default class, variable or method is default public in java and it can be accessed by all those classes which are defined in same package only. 

6)How many types of constructors used in java?

Ans. A constructor in [Java Programming](https://www.simplilearn.com/tutorials/java-tutorial/java-programming) is a block of code that initializes (constructs) the state and value during object creation. It is called every time an object with the help of a new () keyword is created. Even if you haven’t specified any constructor in the code, the [Java](https://www.simplilearn.com/tutorials/java-tutorial/what-is-java) compiler calls a default constructor. The default constructor is used to assign default states and values, such as 0, null, etc., to the object. The general syntax of a constructor is:

class ClassName{

ClassName(){ //creating a constructor

}}

Apart from initialization, a constructor in Java can also perform other tasks, such as calling a method, creating objects, and starting a thread. There is also a separate Constructor class in Java used to get a constructor’s internal information.

A constructor is syntactically similar to a [method](https://www.simplilearn.com/tutorials/java-tutorial/java-encapsulation), but there are several differences between the two. Firstly, although it returns the current class instance, a constructor does not have any explicit return type. Secondly, it is invoked implicitly, whereas a method is not. However, similar to a method, there are a few rules for creating a constructor in Java.

Depending on the provided parameters, Java constructors can be of two types, which are:

* No-arg constructors
* Parameterized constructors

7)Explain this keyword in java?

Ans. The this keyword refers to the current object in a method or constructor. The most common use of the this keyword is to eliminate the confusion between class attributes and parameters with the same name (because a class attribute is shadowed by a method or constructor parameter)

Example:

1. **class** Student{
2. **int** rollno;
3. String name;
4. **float** fee;
5. Student(**int** rollno,String name,**float** fee){
6. rollno=rollno;
7. name=name;
8. fee=fee;
9. }
10. **void** display(){System.out.println(rollno+" "+name+" "+fee);}
11. }
12. **class** TestThis1{
13. **public** **static** **void** main(String args[]){
14. Student s1=**new** Student(111,"ankit",5000f);
15. Student s2=**new** Student(112,"sumit",6000f);
16. s1.display();
17. s2.display();
18. }
19. }

8)Explain super keyword in java?

Ans. The super keyword refers to superclass (parent) objects.

It is used to call superclass methods, and to access the superclass constructor.

The most common use of the super keyword is to eliminate the confusion between superclasses and subclasses that have methods with the same name.

To understand the super keyword, you should have a basic understanding of [Inheritance](https://www.w3schools.com/java/java_inheritance.asp) and [Polymorphism](https://www.w3schools.com/java/java_polymorphism.asp).

Example

// Java code to show use of super

// keyword with variables

// Base class vehicle

class Vehicle {

int maxSpeed = 120;

}

// sub class Car extending vehicle

class Car extends Vehicle {

int maxSpeed = 180;

void display()

{

// print maxSpeed of base class (vehicle)

System.out.println("Maximum Speed: "

+ super.maxSpeed);

}

}

// Driver Program

class Test {

public static void main(String[] args)

{

Car small = new Car();

small.display();

}

}

9)Explain with example method overloading and method overriding in java?

Ans.

1)Method Overloading in Java

In method overloading, more than one method shares the same method name with a different signature in the class. In method overloading, the return type can or can not be the same, but we have to change the parameter because, in java, we can not achieve method overloading by changing only the return type of the method.

// Java Program to Implement

// Method Overloading

import java.io.\*;

class MethodOverloadingEx {

static int add(int a, int b) { return a + b; }

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

{

return a + b + c;

}

// Main Function

public static void main(String args[])

{

System.out.println("add() with 2 parameters");

// Calling function with 2 parameters

System.out.println(add(4, 6));

System.out.println("add() with 3 parameters");

// Calling function with 3 Parameters

System.out.println(add(4, 6, 7));

}

}

OUTPUT:

Add() with 2 parameters

10

Add () with 3 parameter

17AA3 parameters

17

10) Explan final,finally and finalize in java?

Ans. final is a keyword used in Java to restrict the modification of a variable, method, or class. finally is a block used in Java to ensure that a section of code is always executed, even if an exception is thrown. finalize is a method in Java used to perform cleanup processing on an object before it is garbage collected.