**13.What is the difference between break and**

**continue statements**

Key Break Continue

1 Functionality Break statement mainly used to terminate the enclosing loop such as while, do-while, for or switch statement wherever break is declared. Continue statement mainly skip the rest of loop wherever continue is declared and execute the next iteration.

2 Executional flow Break statement resumes the control of the program to the end of loop and made executional flow outside that loop. Continue statement resumes the control of the program to the next iteration of that loop enclosing 'continue' and made executional flow inside the loop again.

3 Usage As mentioned break is used for the termination of enclosing loop. On other hand continue causes early execution of the next iteration of the enclosing loop.

4 Compatibility Break statement can be used and compatible with 'switch', 'label'. We can't use continue statement with 'switch','lablel' as it is not compatible with them.

14.What is an infinite loop in Java? Explain with an

example.

import java.util.\*;

import java.lang.\*;

class Rextester

{

public static void main(String args[])

{

do

{

System.out.print("javaTpoint");

System.out.print(" ");

}while(true);

}

}

This program creates an infinite loop and thus, prints 'javaTpoint' infinite times.

1.Infinite loop in java refers to a situation where a condition is setup so that your loop continues infinitely without a stop.

15.What is Java String Pool?

String pool is nothing but a storage area in Java heap where string literals stores. It is also known as String Intern Pool or String Constant Pool. It is just like object allocation. By default, it is empty and privately maintained by the Java String class. Whenever we create a string the string object occupies some space in the heap memory. Creating a number of strings may increase the cost and memory too which may reduce the performance also.

16.What are constructors in Java?

2 types of constructor are there in java

1.parameterized constructor

2.default constructor

17.What are the differences between Heap and Stack

Memory in Java?

stack:

Stack memory is used to store items which have a very short life like local variables, a reference variable of objects

The stack is always reserved in a LIFO (last in first out) order

We can increase stack memory size by using JVM parameter -XSS

Variables are visible to only to owner thread

JVM will throw java.lang.StackOverFlowError

Heap Memory

Heap memory is allocated to store objects and JRE classes.

Heap memory is dynamic allocation there is no fixed pattern for allocating and deallocating blocks in memory

We can increase or decrease heap memory size by using JVM option -Xms and -Xmx

It is visible to all threads

JVM will throw java.lang.OutOfMemoryError

18.What is a package in Java? List down various

advantages of packages.

Programmers can define their own packages to bundle a group of classes/interfaces, etc.

It is a good practice to group related classes implemented by you so that a programmer can easily determine that the classes, interfaces, enumerations, and annotations are related.

Since the package creates a new namespace there won't be any name conflicts with names in other packages.

Using packages, it is easier to provide access control

It is also easier to locate the related classes.

A package is a collection of similar types of Java entities such as classes, interfaces, subclasses, exceptions, errors, and enums. A package can also contain sub-packages.

there are 2 types of pacakges 1.inbuilt 2.userdefined

`19. What is the difference between a local variable

and an instance variable?

Instance:

They are defined in class but outside the body of methods.

These variables are created when an object is instantiated and are accessible to all constructors, methods, or blocks in class.

These variables are destroyed when the object is destroyed.

It can be accessed throughout the class.

These variables are given a default value if it is not assigned by code.

It is not compulsory to initialize instance variables before use.

It includes access modifiers such as private, public, protected, etc.

LOCAL variable:

They are defined as a type of variable declared within programming blocks or subroutines.

These variables are created when a block, method or constructor is started and the variable will be destroyed once it exits the block, method, or constructor.

These variables are destroyed when the constructor or method is exited.

Its access is limited to the method in which it is declared.

They are used to decreasing dependencies between components I.e., the complexity of code is decreased.

These variables do not always have some value, so there must be a value assigned by code.

It is important to initialize local variables before use.

It does not include any access modifiers such as private, public, protected, etc.

20. What is final keyword in Java?

The final keyword in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

variable

method

class

The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only.

21.What are access modifiers in Java?

The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

22.Define a Java Class.

In object-oriented programming, a class is a basic building block. It can be defined as template that describes the data and behaviour associated with the class instantiation. Instantiating is a class is to create an object (variable) of that class that can be used to access the member variables and methods of the class.

A class can also be called a logical template to create the objects that share common properties and methods.

For example, an Employee class may contain all the employee details in the form of variables and methods. If the class is instantiated i.e. if an object of the class is created (say e1), we can access all the methods or properties of the class.

23..What is an object in Java and how is it created?

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. However, it is used internally by the JVM to identify each object uniquely.

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.

24.What is Object Oriented Programming?

OOPs refers to languages that uses objects in programming. Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism etc in programming. The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.



.25.Differentiate between the constructors and methods in Java?

| **Sr. No.** | **Key** | **Constructors** | **Methods** |
| --- | --- | --- | --- |
| 1 | Purpose | Constructor is used to create and initialize an Object . | Method is used to execute certain statements. |
| 2 | Invocation | A constructor is invoked implicitly by the System. | A method is to be invoked during program code. |
| 3 | Invocation | A constructor is invoked when new keyword is used to create an object. | A method is invoked when it is called. |
| 4 | Return type | A constructor can not have any return type. | A method can have a return type. |
| 5 | Object | A constructor initializes an object which is not existent. | A method can be invoked only on existing object. |
| 6 | Name | A constructor must have same name as that of the class. | A method name can not be same as class name. |
| 7 | Inheritance | A constructor cannot be inherited by a subclass. | A method is inherited by a subclass. |

26.What is the difference between this() and super() in Java?

|  |  |
| --- | --- |
| **this()** | **super()** |
| The this() constructor refers to the current class object. | The super() constructor refers immediate parent class object. |
| It is used for invoking the current class method. | It is used for invoking parent class methods. |
| It can be used anywhere in the parameterized constructor. | It is always the first line in the child class constructor. |
| It is used for invoking a super-class version of an overridden method. | It is used for invoking a super-class version of an overridden method. |

27.Differentiate between static and non-static methods in Java.

| **Points** | **Static method** | **Non-static method** |
| --- | --- | --- |
| **Definition** | A **static method** is a method that belongs to a class, but it does not belong to an instance of that class and this method can be called without the instance or object of that class. | Every method in java defaults to a non-static method without a **static** keyword preceding it. **non-static** methods can access any **static** method and **static** variable also, without using the object of the class. |
| **Accessing members and methods** | In the **static** method, the method can only access only static data members and static methods of another class or same class but cannot access non-static methods and variables. | In the **non-static** method, the method can access static data members and static methods as well as non-static members and methods of another class or same class. |
| **Binding process** | The static method uses compile-time or early binding. | The non-static method uses runtime or dynamic binding. |
| **Overriding** | The static method cannot be overridden because of early binding. | The non-static method can be overridden because of runtime binding. |
| **Memory allocation** | In the **static** method, less memory is used for execution because memory allocation happens only once because the static keyword fixed a particular memory for that method in ram. | In the **non-static** method, much memory is used for execution because here memory allocation happens when the method is invoked and the memory is allocated every time when the method is called. |

28.What is a classloader in Java?

Java ClassLoader is an abstract class. It belongs to a **java.lang** package. It loads classes from different resources. Java ClassLoader is used to load the classes at run time. In other words, JVM performs the linking process at runtime. Classes are loaded into the JVM according to need. If a loaded class depends on another class, that class is loaded as well. When we request to load a class, it delegates the class to its parent. In this way, uniqueness is maintained in the runtime environment. It is essential to execute a Java program.

29.What is Polymorphism?

**Polymorphism in Java** is a concept by which we can perform a single action in different ways. Polymorphism is derived from 2 Greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.

There are two types of polymorphism in Java: compile-time polymorphism and runtime polymorphism. We can perform polymorphism in java by method overloading and method overriding.

If you overload a static method in Java, it is the example of compile time polymorphism. Here, we will focus on runtime polymorphism in java.

**Runtime polymorphism** or **Dynamic Method Dispatch** is a process in which a call to an overridden method is resolved at runtime rather than compile-time.

Q30.What is runtime polymorphism or dynamic method dispatch?

* Runtime polymorphism or Dynamic Method Dispatch is a process in which a call to an overridden method is resolved at runtime rather than compile-time. In this process, an overridden method is called through the reference variable of a superclass.

Q31.What is abstraction in Java?

Data **abstraction** is the process of hiding certain details and showing only essential information to the user.  
Abstraction can be achieved with either **abstract classes** or [**interfaces**](https://www.w3schools.com/java/java_interface.asp)

The abstract keyword is a non-access modifier, used for classes and methods:

* **Abstract class:** is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).

* **Abstract method:** can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).

32.What do you mean by an interface in Java?

An **interface in Java** is a blueprint of a class. It has static constants and abstract methods.

The interface in Java is *a mechanism to achieve [abstraction](https://www.javatpoint.com/abstract-class-in-java)*

. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple [inheritance in Java](https://www.javatpoint.com/inheritance-in-java)

.

In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.

33.What is the difference between abstract classes and interfaces?

|  |  |  |
| --- | --- | --- |
| **S.No.** | **Abstract Class** | **Interface** |
| 1. | An abstract class can contain both abstract and non-abstract methods. | Interface contains only abstract methods. |
| 2. | An abstract class can have all four; static, non-static and final, non-final variables. | Only final and static variables are used. |
| 3. | To declare abstract class abstract keywords are used. | The interface can be declared with the interface keyword. |
| 4. | It supports multiple inheritance. | It does not support multiple inheritance. |
| 5. | The keyword ‘extend’ is used to extend an abstract class | The keyword implement is used to implement the interface. |
| 6. | It has class members like private and protected, etc. | It has class members public by default. |

Q34.What is inheritance in Java?

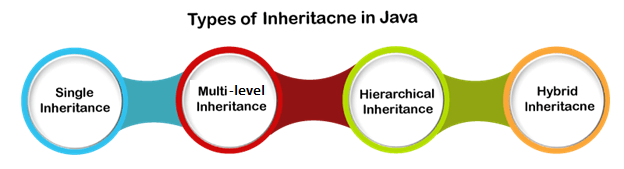
**Inheritance in Java** is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of [OOPs](https://www.javatpoint.com/java-oops-concepts)

(Object Oriented programming system).

The idea behind inheritance in Java is that you can create new [classes](https://www.javatpoint.com/object-and-class-in-java)

that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.

35.What are the different types of inheritance in Java?



In single inheritance, a sub-class is derived from only one super class. It inherits the properties and behavior of a single-parent class. Sometimes it is also known as **simple inheritance**.

In **multi-level inheritance**, a class is derived from a class which is also derived from another class is called multi-level inheritance. In simple words, we can say that a class that has more than one parent class is called multi-level inheritance. Note that the classes must be at different levels. Hence, there exists a single base class and single derived class but multiple intermediate base classes.

### **Hierarchical Inheritance**

If a number of classes are derived from a single base class, it is called **hierarchical inheritance**.

### **Hybrid Inheritance**

Hybrid means consist of more than one. Hybrid inheritance is the combination of two or more types of inheritance.

## Multiple Inheritance (not supported)

Java does not support multiple inheritances due to ambiguity

36.What is method overloading and method overriding?

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Method Overloading in Java** | **Method Overriding in Java** |
| Type of Argument | The argument type needs to be different in Method Overloading (at least the order). | The argument type needs to be the same in Method Overriding (including the order). |
| Return Type | It can be different or the same in this case. But it is a must for a user to change the parameter. | It must be the very same until the 1.4 version of Java only. After that, it only allows the Covariant return type from Java 1.5 onwards. |
| Access Modifiers | You can use any access modifier, or it can be different. | The access modifier for a subclass method must be the very same or higher than the access modifier of the superclass method. |
| Method Signatures | Every method signature must be different (with the same name) in the case of Method Overloading. | Every method signature must be the same (with the same name) in the case of Method Overriding. |
| Class | A user can generally perform method overloading within the same class. | A user can usually perform the method overriding in two of the classes through the Inheritance (considered an Is-A relationship). |
| Final/Static/Private Method | A user can easily overload it. | It is not possible for a user to override it. |
| Method Resolution | A user can always take care of it with a Java compiler based on the reference type. | A user can always take care of it with the JVM based on the runtime object. |
| Performance | The method overloading exhibits much better performance. | The method overriding usually exhibits a lesser performance. |
| Polymorphism | It is also known as the early binding, static polymorphism, or compile-time polymorphism. | It is also known as late binding, dynamic polymorphism, or runtime polymorphism. |
| Uses | The method overloading assists in raising the program’s readability. | It assists in granting the specific implementation of any method (that the superclass or parent class provides). |
| Inheritance | It may or may not be requiring inheritance. | It is always in need of inheritance. |
| Parameter | The parameter needs to be different in the case of method overloading. | The parameter needs to be the same in the case of method overriding. |

37.Can you override a private or static method in Java?

* No, we cannot override private or static methods in Java. Private methods in Java are not visible to any other class which limits their scope to the class in which they are declared.

38.What do you mean by data encapsulation?

**Encapsulation in Java** is a *process of wrapping code and data together into a single unit*, for example, a capsule which is mixed of several medicines.



We can create a fully encapsulated class in Java by making all the data members of the class private. Now we can use setter and getter methods to set and get the data in it.

The **Java Bean** class is the example of a fully encapsulated class.

39. Briefly explain the concept of constructor overloading

In Java, we can overload constructors like methods. The constructor overloading can be defined as the concept of having more than one constructor with different parameters so that every constructor can perform a different task.

Consider the following [Java](https://www.javatpoint.com/java-tutorial)

program, in which we have used different constructors in the class.

### **Example**

1. **public** **class** Student {
2. //instance variables of the class
3. **int** id;
4. String name;
6. Student(){
7. System.out.println("this a default constructor");
8. }
10. Student(**int** i, String n){
11. id = i;
12. name = n;
13. }
15. **public** **static** **void** main(String[] args) {
16. //object creation
17. Student s = **new** Student();
18. System.out.println("\nDefault Constructor values: \n");
19. System.out.println("Student Id : "+s.id + "\nStudent Name : "+s.name);
21. System.out.println("\nParameterized Constructor values: \n");
22. Student student = **new** Student(10, "David");
23. System.out.println("Student Id : "+student.id + "\nStudent Name : "+student.name);
24. }
25. }

**Output:**

this a default constructor

Default Constructor values:

Student Id : 0

Student Name : null

Parameterized Constructor values:

Student Id : 10

Student Name : David

In the above example, the Student class [constructor](https://www.javatpoint.com/java-constructor)

is overloaded with two different constructors, I.e., default and parameterized.

40.Can the static methods be overloaded ?

**Can we overload static methods?**   
The answer is ‘Yes’. We can have two or more static methods with the same name, but differences in input parameters. For example, consider the following Java program.   
// filename Test.java

**public** **class** Test {

**public** **static** **void** foo() {

        System.out.println("Test.foo() called ");

    }

**public** **static** **void** foo(**int** a) {

        System.out.println("Test.foo(int) called ");

    }

**public** **static** **void** main(String args[])

    {

        Test.foo();

        Test.foo(10);

    }

}

**Output**

Test.foo() called

Test.foo(int) called

**Can we overload methods that differ only by static keyword?**   
We cannot overload two methods in Java if they differ only by static keyword (number of parameters and types of parameters is the same). See the following Java program for example. This behavior is the same in C++ (See point 2 of [this](https://www.geeksforgeeks.org/function-overloading-in-c/)). 

* Java

|  |
| --- |
| // filename Test.java  **public** **class** Test {  **public** **static** **void** foo() {          System.out.println("Test.foo() called ");      }  **public** **void** foo() { // Compiler Error: cannot redefine foo()          System.out.println("Test.foo(int) called ");      }  **public** **static** **void** main(String args[]) {          Test.foo();      }  } |

**Output:**

Compiler Error, cannot redefine foo()

Q41.A single try block and multiple catch blocks can co-exist in a Java Program. Explain.

Yes, multiple catch blocks can exist but specific approaches should come prior to the general approach because only the first catch block satisfying the catch condition is executed. The given code illustrates the same:  
  
public class MultipleCatch {  
  
public static void main(String args[]) {  
  
try {  
  
int n = 1000, x = 0;  
  
int arr[] = new int[n];  
  
for (int i = 0; i <= n; i++) {  
  
arr[i] = i / x;  
  
}  
  
}  
  
catch (ArrayIndexOutOfBoundsException exception) {  
  
System.out.println("1st block = ArrayIndexOutOfBoundsException");  
  
}  
  
catch (ArithmeticException exception) {  
  
System.out.println("2nd block = ArithmeticException");  
  
}  
  
catch (Exception exception) {  
  
System.out.println("3rd block = Exception");  
  
}  
  
}  
  
}  
  
Here, the second catch block will be executed because of division by 0 (i / x). Incase x was greater than 0 then the first catch block will execute because for loop runs till i = n and array index are till n-1.

