Java interview questions

What is class in Java?

1. Class is not a real-world entity. It is just a template or blueprint or prototype from which objects are created.
2. Class does not occupy memory.
3. Class is a group of variables of different data types and a group of methods.

A Class in Java can contain:

* Data member
* Method
* Constructor
* Nested Class
* Interface

What is Object in Java?

* It is a fundamental concept of Object-Oriented Programming and represents real-life entities
* When an object of a class is created, the class is said to be **instantiated**. All the instances share the attributes and the behaviour of the class. But the values of those attributes, i.e. the state are unique for each object. A single class may have any number of instances.

What is Method in Java?

* The**method in Java** or Methods of Java is a collection of statements that perform some specific task and return the result to the caller. A Java method can perform some specific task without returning anything. Java Methods allow us to **reuse** the code without retyping the code. In Java, every method must be part of some class that is different from languages like C, C++, and Python.

What is Constructors in Java?

* In Java, 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 the constructor, memory for the object is allocated in the memory. It is a special type of method that is used to initialize the object. Every time an object is created using the new() keyword, at least one constructor is called..
* Constructors must have the same name as the class within which it is defined it
* Constructors do not have return type

**Types of Constructors in Java**

* Default Constructor
* Parameterized Constructor
* Copy Constructor

Access Modifiers:

There are four types of access modifiers available in Java:

1. Default – No keyword required
2. Private
3. Protected
4. Public

### ****1. Default Access Modifier****

When no access modifier is specified for a class, method, or data member – It is said to be having the **default** access modifier by default i.e. having default access modifiers are accessible **only within the same package**.

**2. Private Access Modifier**

The private access modifier is specified using the keyword **private**. The methods or data members declared as private are accessible only **within the class** in which they are declared.

* Any other **class of**the **same package will not be able to access** these members or methods.

**3. Protected Access Modifier**

The **protected access modifier** is accessible within package and outside the package but through inheritance only.

The protected access modifier can be applied on the data member, method and constructor. It can't be applied to the class.

It provides more accessibility than the default modifier.

**4. Public Access Modifier:** The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

Object-oriented programming

These **concepts**aim to implement real-world entities in programs.

* Abstraction
* Encapsulation
* Inheritance
* Polymorphism

**Encapsulation:**

Encapsulation is a fundamental principle in object-oriented programming that combines data and related behaviors (methods) into a single unit called a class. It involves hiding the internal details of the class and providing controlled access to the class's data through methods. We can use setter and getter methods to set and get the data in it.

It provides you the **control over the data**. Suppose you want to set the value of id which should be greater than 100 only, you can write the logic inside the setter method. You can write the logic not to store the negative numbers in the setter methods.

It is a way to achieve **data hiding** in Java because other class will not be able to access the data through the private data members

public class Person {

    private String name;

    private int age;

    public String getName() {

        return name;

    }

    public void setName(String name) {

        this.name = name;

    }

    public int getAge() {

        return age;

    }

    public void setAge(int age) {

        if (age >= 0) {

            this.age = age;

        }

    }

}

public class Main {

    public static void main(String[] args) {

        Person person = new Person();

        person.setName("John");

        person.setAge(25);

        System.out.println("Name: " + person.getName());

        System.out.println("Age: " + person.getAge());

    }

}

### Advantages of Encapsulation in Java

**Data Hiding**

**Increased Flexibility:**

**Reusability**

### Disadvantages of Encapsulation in Java:

1. Can lead to increased complexity, especially if not used properly.
2. Can make it more difficult to understand how the system works.
3. May limit the flexibility of the implementation.

Polymorphism in Java

What is Polymorphism in Java?

Polymorphism is considered one of the important features of Object-Oriented Programming. Polymorphism allows us to perform a single action in different ways. In other words, polymorphism allows you to define one interface and have multiple implementations. The word “poly” means many and “morphs” means forms, so it means many forms.

**Real-life Illustration Polymorphism**: A person at the same time can have different characteristics. Like a man at the same time is a father, a husband, and an employee. So, the same person possesses different behavior in different situations. This is called polymorphism.

**Types of Java polymorphism**

In Java polymorphism is mainly divided into two types:

* Compile-time Polymorphism
* Runtime Polymorphism

Compile-Time Polymorphism

It is also known as static polymorphism. This type of polymorphism is achieved by function overloading

**Method Overloading**

When there are multiple functions with the same name but different parameters then these functions are said to be **overloaded**. Functions can be overloaded by changes in the number of arguments or/and a change in the type of arguments.

[Runtime Polymorphism](https://www.geeksforgeeks.org/dynamic-method-dispatch-runtime-polymorphism-java/)

Runtime polymorphism in Java, also known as dynamic method dispatch, allows objects of different classes to be treated as objects of a common superclass or interface during program execution. It allows you to write more flexible and reusable code by invoking overridden methods based on the actual type of the object at runtime.

Create a base class or interface: Define a base class or interface that contains a method to be overridden by subclasses

public class Animal {

public void makeSound() {

System.out.println("The animal makes a sound.");

}

}

Create derived classes: Inherit from the base class or implement the interface and override the method with specialized implementations.

public class Dog extends Animal {

@Override

public void makeSound() {

System.out.println("The dog barks.");

}

}

public class Cat extends Animal {

@Override

public void makeSound() {

System.out.println("The cat meows.");

}

}

Use polymorphism: Create objects of the derived classes and assign them to variables of the base class or interface type. Invoke the overridden method on these objects, and the appropriate implementation will be called based on the actual object type

Animal animal1 = new Dog();

Animal animal2 = new Cat();

animal1.makeSound(); // Output: "The dog barks."

animal2.makeSound(); // Output: "The cat meows."

In the above example, both **animal1** and **animal2** are declared as **Animal** type, but at runtime, their actual object types are **Dog** and **Cat**, respectively. When the **makeSound()** method is invoked, the overridden version of the method in the respective subclass is called, demonstrating runtime polymorphism.

Advantages of Polymorphism in Java:

1. Increases code reusability by allowing objects of different classes to be treated as objects of a common class.
2. Improves readability and maintainability of code by reducing the amount of code that needs to be written and maintained

### Disadvantages of Polymorphism in Java

1. Can make it more difficult to understand the behavior of an object, especially if the code is complex.
2. This may lead to performance issues, as polymorphic behavior may require additional computations at runtime.

# Inheritance in Java

In Java, Inheritance means creating new classes based on existing ones. A class that inherits from another class can reuse the methods and fields of that class. In addition, you can add new fields and methods to your current class as well.

## Why Do We Need Java Inheritance?

* **Code Reusability:**The code written in the Superclass is common to all subclasses. Child classes can directly use the parent class code.
* **Method Overriding:**[Method Overriding](https://www.geeksforgeeks.org/overriding-in-java/) is achievable only through Inheritance. It is one of the ways by which Java achieves Run Time Polymorphism.
* **Abstraction:**The concept of abstract where we do not have to provide all details is achieved through inheritance. [Abstraction](https://www.geeksforgeeks.org/abstraction-in-java-2/)only shows the functionality to the user.

// Java program to illustrate the

// concept of inheritance

// base class

class Bicycle {

// the Bicycle class has two fields

public int gear;

public int speed;

// the Bicycle class has one constructor

public Bicycle(int gear, int speed)

{

this.gear = gear;

this.speed = speed;

}

// the Bicycle class has three methods

public void applyBrake(int decrement)

{

speed -= decrement;

}

public void speedUp(int increment)

{

speed += increment;

}

// toString() method to print info of Bicycle

public String toString()

{

return ("No of gears are " + gear + "\n"

+ "speed of bicycle is " + speed);

}

}

// derived class

class MountainBike extends Bicycle {

// the MountainBike subclass adds one more field

public int seatHeight;

// the MountainBike subclass has one constructor

public MountainBike(int gear, int speed,int startHeight)

{

// invoking base-class(Bicycle) constructor

super(gear, speed);

seatHeight = startHeight;

}

// the MountainBike subclass adds one more method

public void setHeight(int newValue)

{

seatHeight = newValue;

}

// overriding toString() method

// of Bicycle to print more info

@Override public String toString()

{

return (super.toString() + "\nseat height is "

+ seatHeight);

}

}

// driver class

public class Test {

public static void main(String args[])

{

MountainBike mb = new MountainBike(3, 100, 25);

System.out.println(mb.toString());

}

}

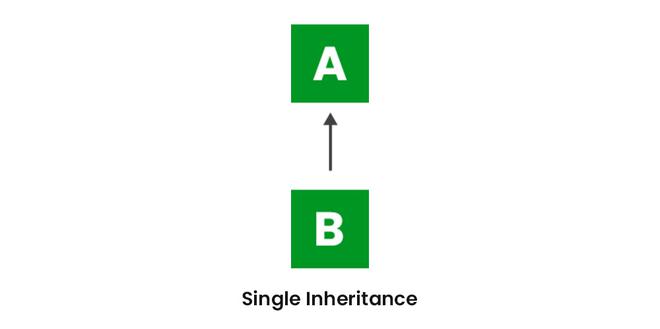
**Java Inheritance Types**

Below are the different types of inheritance which are supported by Java.

1. Single Inheritance
2. Multilevel Inheritance
3. Hierarchical Inheritance
4. Multiple Inheritance
5. Hybrid Inheritance

### ****1. Single Inheritance****

In single inheritance, subclasses inherit the features of one superclass. In the image below, class A serves as a base class for the derived class B.



// Java program to illustrate the

// concept of single inheritance

import java.io.\*;

import java.lang.\*;

import java.util.\*;

// Parent class

class one {

public void print\_geek()

{

System.out.println("Geeks");

}

}

class two extends one {

public void print\_for() { System.out.println("for"); }

}

// Driver class

public class Main {

// Main function

public static void main(String[] args)

{

two g = new two();

g.print\_geek();

g.print\_for();

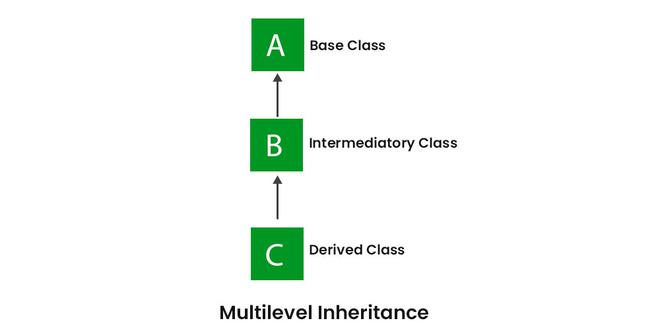
g.print\_geek();

}

}

### ****2. Multilevel Inheritance****

In Multilevel Inheritance, a derived class will be inheriting a base class, and as well as the derived class also acts as the base class for other classes. In the below image, class A serves as a base class for the derived class B, which in turn serves as a base class for the derived class C. In Java, a class cannot directly access the[grandparent’s members](https://www.geeksforgeeks.org/g-fact-91/).



// Java program to illustrate the

// concept of Multilevel inheritance

import java.io.\*;

import java.lang.\*;

import java.util.\*;

class one {

public void print\_geek()

{

System.out.println("Geeks");

}

}

class two extends one {

public void print\_for() { System.out.println("for"); }

}

class three extends two {

public void print\_geek()

{

System.out.println("Geeks");

}

}

// Drived class

public class Main {

public static void main(String[] args)

{

three g = new three();

g.print\_geek();

g.print\_for();

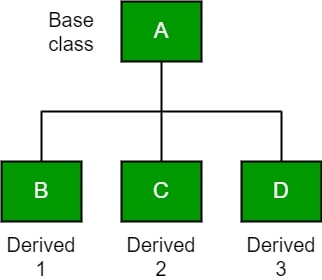
g.print\_geek();

}

}

### ****3. Hierarchical Inheritance****

In Hierarchical Inheritance, one class serves as a superclass (base class) for more than one subclass. In the below image, class A serves as a base class for the derived classes B, C, and D.



// Java program to illustrate the

// concept of Hierarchical inheritance

class A {

public void print\_A() { System.out.println("Class A"); }

}

class B extends A {

public void print\_B() { System.out.println("Class B"); }

}

class C extends A {

public void print\_C() { System.out.println("Class C"); }

}

class D extends A {

public void print\_D() { System.out.println("Class D"); }

}

// Driver Class

public class Test {

public static void main(String[] args)

{

B obj\_B = new B();

obj\_B.print\_A();

obj\_B.print\_B();

C obj\_C = new C();

obj\_C.print\_A();

obj\_C.print\_C();

D obj\_D = new D();

obj\_D.print\_A();

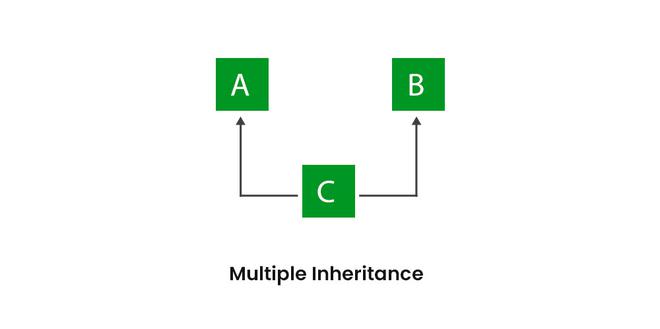
obj\_D.print\_D();

}

}

### 4. Multiple Inheritance (****Through Interfaces)****

Java does not support multiple inheritance of classes, which means a class cannot directly inherit from multiple classes simultaneously. This decision was made by the designers of Java to simplify the language and avoid some of the complexities and ambiguities that can arise from multiple inheritance.



// Java program to illustrate the

// concept of Multiple inheritance

import java.io.\*;

import java.lang.\*;

import java.util.\*;

interface one {

public void print\_geek();

}

interface two {

public void print\_for();

}

interface three extends one, two {

public void print\_geek();

}

class child implements three {

@Override public void print\_geek()

{

System.out.println("Geeks");

}

public void print\_for() { System.out.println("for"); }

}

// Drived class

public class Main {

public static void main(String[] args)

{

child c = new child();

c.print\_geek();

c.print\_for();

c.print\_geek();

}

}

**5. Hybrid Inheritance (Through Interfaces)**

It is a mix of two or more of the above types of inheritance. Since Java doesn’t support multiple inheritances with classes, hybrid inheritance is also not possible with classes. In Java, we can achieve hybrid inheritance only through [Interfaces](https://www.geeksforgeeks.org/interfaces-in-java/).

// Interface for a shape

interface Shape {

    void draw ();

}

// Base class for a vehicle

class Vehicle {

    void move () {

        System.out.println("The vehicle is moving.");

    }

}

// Derived class implementing the Shape interface

class Circle implements Shape {

    @Override

    public void draw () {

        System.out.println("Drawing a circle.");

    }

}

// Derived class inheriting from both Vehicle and implementing Shape interface

class Car extends Vehicle implements Shape {

    @Override

    public void draw () {

        System.out.println("Drawing a car.");

    }

}

public class Main {

    public static void main (String [] args) {

        Circle circle = new Circle ();

        circle.draw();

        Car car = new Car ();

        car.move();

        car.draw();

    }

}

Advantages Of Inheritance in Java

\* Code Reusability

\* Abstraction: Inheritance allows for the creation of abstract classes that define a common interface for a group of related classes. This promotes abstraction and encapsulation, making the code easier to maintain and extend.

\* Class Hierarchy

\* Polymorphism

Disadvantages of Inheritance in Java:

\* Complexity

\* Tight Coupling: Inheritance creates a tight coupling between the superclass and subclass, making it difficult to make changes to the superclass without affecting the subclass.

What is Abstraction in Java?

In Java, abstraction is achieved by [interfaces](https://www.geeksforgeeks.org/interfaces-in-java/) and [abstract classes](https://www.geeksforgeeks.org/abstract-classes-in-java/). We can achieve 100% abstraction using interfaces.

Abstraction may also be defined as the process of identifying only the required characteristics of an object ignoring the irrelevant details.

**Real-Life Example:**

*Consider a real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of a car or applying brakes will stop the car, but he does not know how on pressing the accelerator the speed is increasing, he does not know about the inner mechanism of the car or the implementation of the accelerator, brakes, etc in the car. This is what abstraction is.*

**Java Abstract classes and Java Abstract methods**

1. An abstract class is a class that is declared with an [abstract keyword.](https://www.geeksforgeeks.org/abstract-keyword-in-java/)
2. An abstract method is a method that is declared without implementation.
3. An abstract class may or may not have all abstract methods. Some of them can be concrete methods
4. A method-defined abstract must always be redefined in the subclass, thus making [overriding](https://www.geeksforgeeks.org/overriding-in-java/) compulsory or making the subclass itself abstract.
5. Any class that contains one or more abstract methods must also be declared with an abstract keyword.
6. There can be no object of an abstract class. That is, an abstract class cannot be directly instantiated with the [*new operator*](https://www.geeksforgeeks.org/new-operator-java/).

// Java program to illustrate the

// concept of Abstraction

abstract class Shape {

String color;

// these are abstract methods

abstract double area();

public abstract String toString();

// abstract class can have the constructor

public Shape(String color)

{

System.out.println("Shape constructor called");

this.color = color;

}

// this is a concrete method

public String getColor() { return color; }

}

class Circle extends Shape {

double radius;

public Circle(String color, double radius)

{

// calling Shape constructor

super(color);

System.out.println("Circle constructor called");

this.radius = radius;

}

@Override double area ()

{

return Math.PI \* Math.pow(radius, 2);

}

@Override public String toString()

{

return "Circle color is " + super.getColor()

+ "and area is : " + area ();

}

}

class Rectangle extends Shape {

double length;

double width;

public Rectangle (String color, double length,

double width)

{

// calling Shape constructor

super(color);

System.out.println("Rectangle constructor called");

this.length = length;

this.width = width;

}

@Override double area () {return length \* width; }

@Override public String toString ()

{

return "Rectangle color is " + super.getColor ()

+ "and area is: " + area ();

}

}

public class Test {

public static void main (String [] args)

{

Shape s1 = new Circle ("Red", 2.2);

Shape s2 = new Rectangle ("Yellow", 2, 4);

System.out.println (s1.toString ());

System.out.println (s2.toString ());

}

}

**Advantages of Abstraction**

1.Improves code reusability and maintainability.

2. Helps to increase the security of an application or program as only essential details are provided to the user.

3.Hides implementation details and exposes only relevant information.

4. Provides a clear and simple interface to the user.

Disadvantages of Abstraction in Java:

1.Abstraction can make it more difficult to understand how the system works.

2.It can lead to increased complexity, especially if not used properly.

Interfaces in Java

interface in Java is a blueprint of a behavior. A Java interface contains static constants and abstract methods.

The interface in Java is *a*mechanism to achieve [abstraction](https://www.geeksforgeeks.org/abstraction-in-java-2/). There can be only abstract methods in the Java interface, not the method body. It is used to achieve abstraction and [multiple inheritance in Java](https://www.geeksforgeeks.org/java-and-multiple-inheritance/).

* Interfaces specify what a class must do and not how. It is the blueprint of the behavior.
* Interface do not have constructor.
* Represent behavior as interface unless every sub-type of the class is guaranteed to have that behavior.
* If a class implements an interface and does not provide method bodies for all functions specified in the interface, then the class must be declared abstract.

Why do we use an Interface?

* It is used to achieve total abstraction.
* Since java does not support multiple inheritances in the case of class, by using an interface it can achieve multiple inheritances.
* Any class can extend only 1 class but can any class implement infinite number of interface.
* It is also used to achieve loose coupling.
* Interfaces are used to implement abstraction. So the question arises why use interfaces when we have abstract classes?

The reason is, abstract classes may contain non-final variables, whereas variables in the interface are final, public and static.

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Class** | **Interface** |
| 1. | In class, you can instantiate variables and create an object. | In an interface, you can’t instantiate variables and create an object. |
| 2. | Class can contain concrete (with implementation) methods | The interface cannot contain concrete (with implementation) methods |
| 3. | The access specifiers used with classes are private, protected, and public. | In Interface only one specifier is used- Public. |

// Java program to demonstrate working of

// interface

import java.io.\*;

// A simple interface

interface In1 {

// public, static and final

final int a = 10;

// public and abstract

void display ();

}

// A class that implements the interface.

class TestClass implements In1 {

// Implementing the capabilities of

// interface.

public void display (){

System.out.println("Geek");

}

// Driver Code

public static void main (String [] args)

{

TestClass t = new TestClass();

t.display();

System.out.println(a);

}

}

Advantages of Interfaces in Java

1.In Java, multiple inheritances is not allowed, however, you can use an interface to make use of it as you can implement more than one interface.

2. Without bothering about the implementation part, we can achieve the security of the implementation.

# ‘this’ reference in Java

In Java, ‘this’ is a reference variable that refers to the current object, or can be said “this” in Java is a keyword that refers to the current object instance. It can be used to call current class methods and fields, to pass an instance of the current class as a parameter, and to differentiate between the local and instance variables. Using “this” reference can improve code readability and reduce naming conflicts.

 // Java Program to implement

// Java this reference

public class ThisExample {

int num = 10;

public ThisExample()

{

System.out.println("Inside constructor");

}

public ThisExample(int num)

{

// Invoking default constructor

this();

// Assigning the local variable num to the instance

// variable num

this.num = num;

}

void display()

{

// Invoking the method show() of the current class

this.show();

// Displaying the value of the instance variable num

System.out.println("num: " + this.num);

}

void show()

{

System.out.println("Inside show method");

}

public static void main(String[] args)

{

ThisExample obj = new ThisExample(100);

obj.display();

}

}

Static Keyword:

In Java, the keyword "static" is used to declare a member (variable or method) that belongs to the class itself, rather than to instances (objects) of the class. When a member is declared as static, it means it is shared by all instances of the class.



 Wrapper Classes:

Wrapper classes in Java are a set of classes that encapsulate primitive data types and provide object-oriented representations for them. They allow primitive types to be used as objects.

wrapper classes serve several purposes, such as enabling primitive types to be used in collections (which typically require objects), providing utility methods and functions for working with primitive types, supporting autoboxing and unboxing,

Two concepts are there in the wrapper classes namely autoboxing and unboxing.

Autoboxing – primitive type to object of the corresponding [wrapper class](https://www.geeksforgeeks.org/wrapper-classes-java/)

Unboxing – Coverting objects to primitive types

**Difference between Abstraction and Encapsulation:**

Abstraction is the method of hiding unwanted information.

Whereas encapsulation is a method to hide the data in a single entity or unit along with a method to protect information from outside

JDK

JDK is a Java Development Kit. The Java Development Kit (JDK) is a software development environment which is used to develop Java applications   It contains JRE + development tools.



JRE

JRE is an acronym for Java Runtime Environment. It is also written as Java RTE. The Java Runtime Environment is a set of software tools which are used for developing Java applications. It is used to provide the runtime environment. It is the implementation of JVM. It physically exists. It contains a set of libraries + other files that JVM uses at runtime.



 JVM

JVM (Java Virtual Machine) is an abstract machine. It is called a virtual machine because it doesn't physically exist. It is a specification that provides a runtime environment in which Java bytecode can be executed. It can also run those programs which are written in other languages and compiled to Java bytecode.

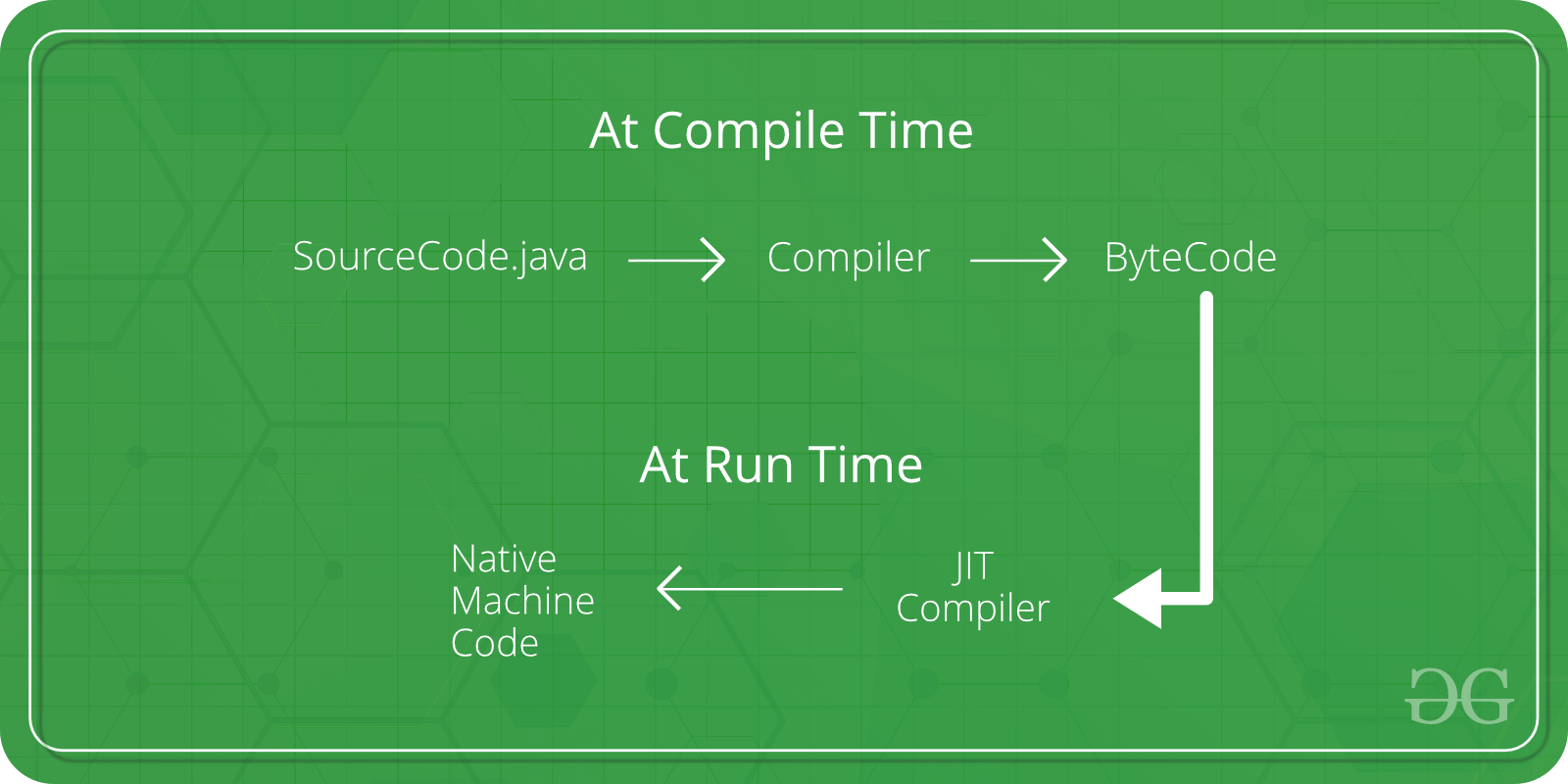
Note :

*Consider a java source file saved as ‘Example.java’. The file is compiled into a set of Byte Code that is stored in a “.class” file. Here it will be “Example.class“.*



* Java applications are called WORA (Write Once Run Anywhere). This means a programmer can develop Java code on one system and can expect it to run on any other Java-enabled system without any adjustments. This is all possible because of JVM.
* When we compile a *.java* file, *.class* files(contains byte-code) with the same class names present in *.java* file are generated by the Java compiler. This *.class* file goes into various steps when we run it. These steps together describe the whole JVM.

JIT

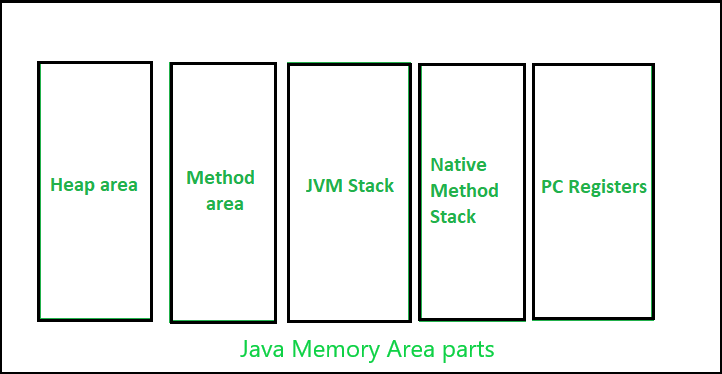


 Java Memory Management

The major concepts in Java Memory Management:

* JVM Memory Structure
* Working of Garbage Collector

Java Memory Structure:



 Stack Memory:

**Key Points:**

* It’s a temporary memory allocation scheme where the data members are accessible only if the method () that contained them is currently running.
* It allocates or de-allocates the memory automatically as soon as the corresponding method completes its execution.
* We receive the corresponding error Java. lang.[StackOverFlowError](https://www.geeksforgeeks.org/stackoverflowerror-in-java-with-examples/)by [JVM](https://www.geeksforgeeks.org/jvm-works-jvm-architecture/), If the stack memory is filled completely.
* Stack memory allocation is considered safer as compared to heap memory allocation because the data stored can only be accessed by the owner thread.
* Memory allocation and de-allocation are faster as compared to Heap-memory allocation.
* Stack memory has less storage space as compared to Heap-memory.

int main ()

{

// All these variables get memory

// allocated on stack

int a;

int b [10];

int n = 20;

int c[n];

}

**Heap Allocation:**

**Key Points:**

* We receive the corresponding error message if Heap-space is entirely full,  [java. lang.OutOfMemoryError](https://www.geeksforgeeks.org/understanding-outofmemoryerror-exception-java/" \t "_blank) by JVM.
* This memory allocation scheme is different from the Stack-space allocation, here no automatic de-allocation feature is provided. We need to use a Garbage collector to remove the old unused objects in order to use the memory efficiently.
* The processing time(Accessing time) of this memory is quite slow as compared to Stack-memory.
* Heap memory is also not as threaded-safe as Stack-memory because data stored in Heap-memory are visible to all threads.
* The size of the Heap-memory is quite larger as compared to the Stack-memory.
* Heap memory is accessible or exists as long as the whole application (or java program) runs.

**class Emp {**

**int id;**

**String emp\_name;**

**public Emp (int id, String emp\_name) {**

**this.id = id;**

**this.emp\_name = emp\_name;**

**}**

**}**

**public class Emp\_detail {**

**private static Emp Emp\_detail(int id, String emp\_name) {**

**return new Emp(id, emp\_name);**

**}**

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

**int id = 21;**

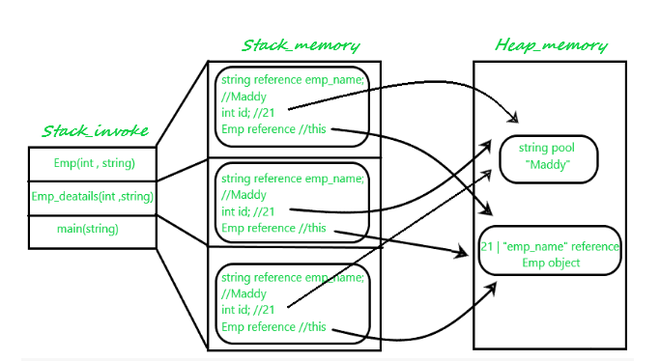
**String name = "Maddy";**

**Emp person\_ = null;**

**person\_ = Emp\_detail(id, name);**

**}**

**}**



**Key Differences Between Stack and Heap Allocations**  

1. In a stack, the allocation and de-allocation are automatically done by the compiler whereas, in heap, it needs to be done by the programmer manually.
2. Handling the Heap frame is costlier than handling the stack frame.
3. Memory shortage problem is more likely to happen in stack whereas the main issue in heap memory is fragmentation.
4. Stack frame access is easier than the heap frame as the stack has a small region of memory and is cache-friendly but in the case of heap frames which are dispersed throughout the memory so it causes more cache misses.
5. A stack is not flexible, the memory size allotted cannot be changed whereas a heap is flexible, and the allotted memory can be altered.
6. Accessing the time of heap takes is more than a stack.

1. Class (Method) Area

The class method area is the memory block that stores the class code, variable code (static variable, runtime constant), method code, and the constructor of a Java program. (Here method means the function which is written inside the class). It stores class-level data of every class such as the runtime constant pool, field and method data, the code for methods.

2. Heap

The Heap area is the memory block where objects are created, or objects are stored. Heap memory allocates memory for class interfaces and arrays (an array is an object). It is used to allocate memory to objects at run time

#### 3. Stack

Each thread has a private JVM stack, created at the same time as the thread. It is used to store data and partial results which will be needed while returning value for method and performing dynamic linking.

Java Stack stores frames and a new frame is created each time at every invocation of the method. A frame is destroyed when its method invocation completes

#### What is Garbage Collection?

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.

What is a string in Java, and how is it different from other data types?

A string in Java is a sequence of characters. It is a reference type and is immutable, meaning its value cannot be changed once created. Unlike other data types, strings have their own class (String) in Java.

How do you create a string object in Java? Provide examples.?

String str1 = new String("Hello");

String str2 = "World";

What is the difference between String, StringBuilder, and StringBuffer in Java?

The String class represents an immutable sequence of characters. StringBuilder and StringBuffer are mutable classes for creating and manipulating strings. StringBuilder is not thread-safe, while StringBuffer is thread-safe.

Can you explain the immutability of strings in Java?

Strings in Java are immutable, meaning their values cannot be changed after creation. This immutability allows strings to be safely shared and used as keys in hash-based data structures.

How can you concatenate strings in Java?

String str1 = "Hello";

String str2 = "World";

String result = str1 + ", " + str2; // "Hello, World"

What are some methods available in the String class? Explain a few of them.

The String class provides various methods such as **length()**, **charAt()**, **substring()**, **toUpperCase()**, **toLowerCase()**, **indexOf()**, and **replace()**, among others, for manipulating and accessing string data.

How do you compare two strings in Java? What is the difference between "==" and the **equals()** method?

To compare two strings in Java, you can use the **equals()** method, which checks for content equality. The **==** operator compares references for object equality. Example:

String str1 = "Hello";

String str2 = "Hello";

boolean isEqual = str1.equals(str2); // true

boolean isSameReference = str1 == str2; // true

What is the significance of the **StringPool** in Java?

The String Pool is a special memory area in Java where string literals are stored. It allows strings to be reused, saving memory. String literals with the same value share the same reference in the String Pool.

Explain the **substring()** method in the String class. Provide an example.?

The **substring()** method in the String class extracts a portion of a string based on the specified indexes. Example:

String str = "Hello, World";

String substr = str.substring(7, 12); // "World"

How can you convert a string to an integer in Java?

To convert a string to an integer, you can use the **parseInt()** method of the Integer class or the **valueOf()** method. Example:

String str = "123";

int number = Integer.parseInt(str); // 123

Can you describe the concept of string interpolation or formatting in Java?

String interpolation or formatting in Java can be achieved using the **String.format()** method or the **printf()** method. These methods allow you to substitute values into a formatted string. Example:

String name = "Alice";

int age = 25;

String message = String.format("My name is %s and I'm %d years old.", name, age);

// "My name is Alice and I'm 25 years old."

What is the purpose of the **StringBuffer** class, and when would you choose to use it over **StringBuilder**?

StringBuffer and StringBuilder are used when you need mutable string objects. Use StringBuilder when working in a single-threaded environment and StringBuffer when working in a multi-threaded environment.

What are some common performance considerations when working with strings in Java?

When working with strings in Java, consider using **StringBuilder** or **StringBuffer** for concatenation or repeated modifications, as they offer better performance than concatenating strings directly.

How does Java handle Unicode characters and string encoding?

Java uses the Unicode character set to represent characters. Strings in Java

# Collections in Java

#### What is Collection in Java

A Collection represents a single unit of objects, i.e., a group.

Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.

#### What is a framework in Java

* It provides readymade architecture.
* It represents a set of classes and interfaces.
* It is optional.

#### What is Collection framework

The Collection framework represents a unified architecture for storing and manipulating a group of objects. It has:

1. Interfaces and its implementations, i.e., classes
2. Algorithm

### Hierarchy of Collection Framework

Let us see the hierarchy of Collection framework. The **java.util** package contains all the [classes](https://www.javatpoint.com/object-and-class-in-java) and [interfaces](https://www.javatpoint.com/interface-in-java) for the Collection framework.



Explain the difference between the Collection framework and the Collections class in Java.?

The Collection framework is a set of interfaces, classes, and algorithms that provide the foundation for working with collections in Java. The Collections class, on the other hand, is a utility class that provides various static methods to perform operations on collections, such as sorting, searching, and synchronizing.

What is the difference between List, Set, and Map in Java?

List is an ordered collection that allows duplicate elements, Set is an unordered collection that does not allow duplicate elements, and Map is a collection of key-value pairs where each key is unique.

# Java ArrayList

* Java ArrayList class can contain duplicate elements.
* Java ArrayList class maintains insertion order.
* Java ArrayList class is non [synchronized](https://www.javatpoint.com/synchronization-in-java).
* Java ArrayList allows random access because the array works on an index basis.
* In ArrayList, manipulation is a little bit slower than the LinkedList in Java because a lot of shifting needs to occur if any element is removed from the array list.
* We can not create an array list of the primitive types, such as int, float, char, etc. It is required to use the required wrapper class in such cases.

What is the difference between ArrayList and LinkedList? In which scenarios would you choose one over the other?

|  |  |
| --- | --- |
| **ArrayList** | **LinkedList** |
| 1) ArrayList internally uses a **dynamic array** to store the elements. | LinkedList internally uses a **doubly linked list** to store the elements. |
| 2) Manipulation with ArrayList is **slow** because it internally uses an array. If any element is removed from the array, all the other elements are shifted in memory. | Manipulation with LinkedList is **faster** than ArrayList because it uses a doubly linked list, so no bit shifting is required in memory. |
| 3) An ArrayList class can **act as a list** only because it implements List only. | LinkedList class can **act as a list and queue** both because it implements List and Deque interfaces. |
| 4) ArrayList is **better for storing and accessing** data. | LinkedList is **better for manipulating** data. |

What is the significance of the Iterator interface in Java collections?

[**Iterable interface**](https://www.geeksforgeeks.org/iterable-interface-in-java/) is the root interface for the entire collection framework. The collection interface extends the iterable interface. Therefore, inherently, all the interfaces and classes implement this interface. The main functionality of this interface is to provide an iterator for the collections.

Can you explain the concept of generics in the Java Collection framework?

Generics in the Java Collection framework allow you to specify the type of elements that a collection can contain. They provide type safety and enable compile-time type checking to prevent type-related errors.

How do you sort elements in a List in Java? Explain the use of Comparable and Comparator interfaces.?

Elements in a List can be sorted using the **Collections.sort()** method. The Comparable interface is used to define a natural ordering for objects, while the Comparator interface allows custom sorting based on specific criteria.

What is the purpose of the hashCode() and equals() methods in the Object class? How are they used in collections?

The **hashCode()** and **equals()** methods are used for hashing and equality comparisons in collections. **hashCode()** calculates a unique integer value for an object, while **equals()** compares two objects for equality based on their content.

How does the Hashset class work internally in Java?

The important points about Java HashSet class are:

* HashSet stores the elements by using a mechanism called **hashing.**
* HashSet contains unique elements only.
* HashSet allows null value.
* HashSet class is non synchronized.
* HashSet doesn't maintain the insertion order. Here, elements are inserted on the basis of their hashcode.
* HashSet is the best approach for search operations.

How does the Hashmap class work internally in Java?

* Java HashMap contains values based on the key.
* Java HashMap contains only unique keys.
* Java HashMap may have one null key and multiple null values.
* Java HashMap is non synchronized.
* Java HashMap maintains no order.

What is the purpose of the Queue interface in Java? Provide an example of its implementation.?

The Queue interface in Java represents a collection designed for holding elements prior to processing. It follows the FIFO (First-In-First-Out) principle. An example implementation of Queue is the LinkedList class.

How can you iterate over a Map in Java?

You can iterate over a Map in Java using various methods such as keySet(), entrySet(), or values(). The keySet() method returns a Set of all keys, entrySet() returns a Set of key-value pairs, and values() returns a Collection of all values.

**Exception Handling**

**Exception Handling** in Java is one of the effective to handle the runtime errors so that the regular flow of the application can be preserved.

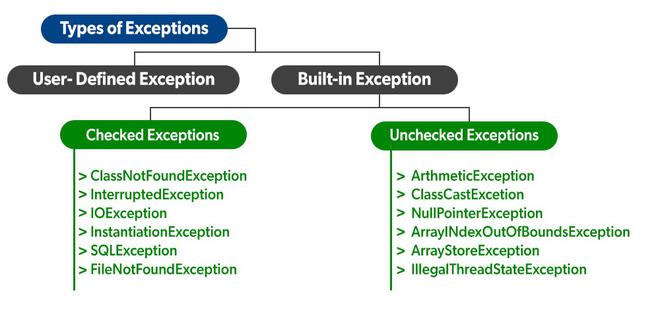
Java Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

* **Error:** An Error indicates a serious problem that a reasonable application should not try to catch.
* **Exception:** Exception indicates conditions that a reasonable application might try to catch.

Exception Hierarchy



Types of Exceptions



**Built-in Exceptions:**

* **Checked Exceptions:** Checked exceptions are called compile-time exceptions because these exceptions are checked at compile-time by the compiler.
* **Unchecked Exceptions:** The unchecked exceptions are just opposite to the checked exceptions. The compiler will not check these exceptions at compile time. In simple words, if a program throws an unchecked exception, and even if we didn’t handle or declare it, the program would not give a compilation error.

**User-Defined Exceptions:**

Sometimes, the built-in exceptions in Java are not able to describe a certain situation. In such cases, users can also create exceptions, which are called ‘user-defined Exceptions’

**1.ArithmeticException:** It is thrown when an exceptional condition has occurred in an arithmetic operation.

**2.ArrayIndexOutOfBoundsException:** It is thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.

**3.ClassNotFoundException:** This Exception is raised when we try to access a class whose definition is not found

**4.FileNotFoundException:** This Exception is raised when a file is not accessible or does not open.

**5.IOException:** It is thrown when an input-output operation failed or interrupted

**6. NoSuchFieldException:** It is thrown when a class does not contain the field (or variable) specified

**7.NoSuchMethodException:** It is thrown when accessing a method that is not found.

**8.NullPointerException:** This exception is raised when referring to the members of a null object. Null represents nothing

**9.RuntimeException:** This represents an exception that occurs during runtime.

**10.StringIndexOutOfBoundsException:** It is thrown by String class methods to indicate that an index is either negative or greater than the size of the string

**Blocks & Keywords used for exception handling**

[**try**](https://www.geeksforgeeks.org/flow-control-in-try-catch-finally-in-java/): The try block contains a set of statements where an exception can occur.

try   
{   
    // statement(s) that might cause exception   
}

**catch**: The catch block is used to handle the uncertain condition of a try block. A try block is always followed by a catch block, which handles the exception that occurs in the associated try block.

// Java program to demonstrate working of try,

// catch and finally

class Division {

public static void main(String [] args)

{

int a = 10, b = 5, c = 5, result;

try {

result = a / (b - c);

System.out.println("result" + result);

}

catch (ArithmeticException e) {

System.out.println("Exception caught:Division by zero");

}

finally {

System.out.println("I am in final block");

}

}

}

**throw**: The throw keyword is used to transfer control from the try block to the catch block.

public class ExceptionExample {

    public static void main (String [] args) {

        int age = -1;

        try {

            if (age < 0) {

                throw new IllegalArgumentException("Age cannot be negative");

            }

            System.out.println("Age: " + age);

        } catch (IllegalArgumentException e) {

            System.out.println("Exception: " + e.getMessage());

        }

    }

}

Output: Exception: Age cannot be negative

Example 2:

// Java program that demonstrates the use of throw

class ThrowExcep {

static void fun ()

{

try {

throw new NullPointerException("demo");

}

catch (NullPointerException e) {

System.out.println("Caught inside fun ().");

throw e; // rethrowing the exception

}

}

public static void main (String args[])

{

try {

fun();

}

catch (NullPointerException e) {

System.out.println("Caught in main.");

}

}

}

**throws**: The throws keyword is used for exception handling without try & catch block. It specifies the exceptions that a method can throw to the caller and does not handle itself.

// Java program to demonstrate working of throws

class ThrowsExecp {

// This method throws an exception

// to be handled

// by caller or caller

// of caller and so on.

static void fun () throws IllegalAccessException

{

System.out.println("Inside fun (). ");

throw new IllegalAccessException("demo");

}

// This is a caller function

public static void main (String args[])

{

try {

fun();

}

catch (IllegalAccessException e) {

System.out.println("caught in main.");

}

}

}

Example 2:

// Java program to illustrate throws

class tst {

public static void main (String [] args)

throws InterruptedException

{

Thread.sleep(10000);

System.out.println("Hello Geeks");

}

}

java