eGd mg sir

My name pradip giri

I belong to west Bengal, I currently pursuing mca form the oxford college of eng bengalore.

I have done my bsc computer se from Midnapore college west Bengal

I have keen interest c , java dbms as well as I have done my project on it.

Thank u , that all about my self

In 5 yr, I would like to see myself in a leading position more responsibility ,knowledge and skils.

I would be more mature regarding my pfersonality and taking decision for this organization

And encourage my junior to give their best

**Do you have any questions for me?**

Thank you for this opportunity, I want to ask you that what are the growth opportunities for me if I got selected?

# How much salary do you expect?

Sir, I am a fresher and yes salary is a matter for me but my first priority is to gain knowledge and improve my skills, so I happily accepted whatever you offered.

# 

# Who has inspired you in your life and why?

# My father inspired me a lot because he regularly work hard to pay my college & hostel fees and provide a better lifestyle & his only wish to see me well settled in my life.

# Why java independent language

Java is well known for its Platform independence.

While you write a Java code and feed it to the compiler a .class file will be generated. Now this .class file/Byte code common for all kind of system whether its Linux or Mac or Windows.  Now to run this .class file, we need JRE (which is actually implementation of JVM). SO Mac will have its own version of JRE, Widows will have different and Linux has different one too. But one thing to notice here is that --- the input for all these , means the .class file is same. This is actually Platform Independence.

# C vs java

# Call by value and call by reference

## Difference between Class and Object in Java

There are the following differences between class and object in java. They are as follows:

1. A class is a user-defined data type whereas an object is an instance of class data type.

2. A class generates objects whereas an object gives life to a class.

3. Classes do not occupy memory location but objects occupy memory location.

4. Classes cannot be manipulated due to not available in the memory location but objects can be manipulated.

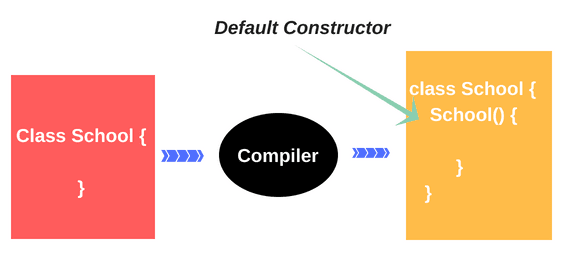
# A **constructor in java** is a block of code, similar to a [method](https://www.scientecheasy.com/2020/06/java-methods.html/) that is used to initialize the state of an object in a class through a new operator.

# it should have same name at that class

# It will executed when a object created

# he constructor should not have any return type even void also because if there is a [return type](https://www.scientecheasy.com/2020/06/return-type-in-java.html/) then [JVM](https://www.scientecheasy.com/2021/03/what-is-jvm.html/) would consider as a method, not a constructor.

## Default Constructor in Java with Example Program

A constructor that has no parameter is known as default constructor in Java. When a class does not declare a constructor, Java compiler automatically creates a constructor for that class. This constructor is called default constructor.[](https://www.scientecheasy.com/2020/06/constructor-in-java.html/)

We cannot pass any argument to default constructor. That’s why it is known as a no-argument constructor in java. It does not do anything but it allows to create instance of class.

## Parameterized Constructor in Java

A constructor that takes 1 or more parameters is known as parameterized constructor in java. The parameterized constructor is used to provide different values to distinct objects.

An example of parameterized constructor is as follows:

Person(String name, int age) {

// Constructor code.

}

To call the parameterized constructor, we pass arguments while creating the object. Therefore, parameterized constructor is also called argument constructor.

**Constructor vs Method**

|  |  |  |
| --- | --- | --- |
| **SN** | **Constructor** | **Method** |
| 1. | Constructor is a special type of method that is used to initialize the state of an object. | Method is used to expose the behaviour of an object. |
| 2. | It has no return type even void also. | It has both void and return type. |
| 3. | If we don’t provide any constructor in the class, Java Compiler provides a default constructor for that class. | Method is not provided by the compiler in any case. |
| 4. | Constructor name must be the same as name of the class. | Method name may or may not be the same name as the class name. |
| 5. | The purpose of a constructor is to create an object of a class. | The purpose of a method is to execute the functionality of the application. |
| 6. | They are not inherited by subclasses. | They are inherited by subclasses. |

# Constructor Overloading in Java | Example Program

**Constructor overloading in Java** means to define multiple [constructors](https://www.scientecheasy.com/2020/06/constructor-in-java.html/) of a class but each one must have a different signature.

[Java compiler](https://www.scientecheasy.com/2021/03/java-compiler.html/) differentiates these constructors based on the number of the parameter lists and their types. Therefore, the signature of each constructor must be different.

## What Is Object-Oriented Programming in Java?

**OOPs Concepts in Java |** Object-oriented programming (OOP) in Java is a programming methodology or paradigm (model)  to design a computer program using classes and objects.

It is the most popular programming paradigm and widely used in the software industry today. It is an extension of procedural programming.

# Encapsulation in Java |

encapsulation is a programming technique that binds the class members (variables and methods) together and prevents them from being accessed by other classes.

**Realtime Example 1:**  
School bag is one of the most real examples of Encapsulation. School bag can keep our books, pens, etc.

**Realtime Example 3:**  
Suppose you have an account in the bank. If your balance variable is declared as a public variable in the bank software, your account balance will be known as public, In this case, anyone can know your account balance. So, would you like it? Obviously No.

So, they declare balance variable as private for making your account safe, so that anyone cannot see your account balance.

The person who has to see his account balance, will have to access only private members through methods defined inside that class and this method will ask your account holder name or user Id, and password for authentication.

Thus, we can achieve security by utilizing the concept of data hiding. This is called Encapsulation in Java.

## Advantage of Encapsulation in Java

There are following advantages of encapsulation in Java. They are as follows:

1. The encapsulated code is more flexible and easy to change with new requirements.  
2. It prevents the other classes to access the private fields.  
3. Encapsulation allows modifying implemented code without breaking other code that has implemented the code.  
4. It keeps the data and codes safe from external inheritance. Thus, Encapsulation helps to achieve security.  
5. It improves the maintainability of the application.  
6. If you don’t define the setter method in the class then the fields can be made read-only.  
7. If you don’t define the getter method in the class then the fields can be made write-only.

## Disadvantage of Encapsulation in Java

The main disadvantage of encapsulation in Java is it increases the length of the code and slows shutdown execution

public class Account

{

private double balance;

public double getbalance()

 {

   return balance;

 }

}

# Abstraction in Java | Abstract Class, Example

**Abstraction in Java** is another [OOPs principle](https://www.scientecheasy.com/2020/07/oops-concepts-in-java.html/) that manages complexity. It is a process of hiding complex internal implementation details from the user and providing only necessary functionality to the users.

In other words, abstraction in Java is a technique by which we can hide the data that is not required to a user.

It hides all unwanted data so that users can work only with the required data. It removes all non-essential things and shows only important things to users.

That is, every user will get the required data and will not get confused with unnecessary data.

Let’s take some realtime examples to understand the concept of java abstraction.

## Realtime Examples of Abstraction in Java

1. Let’s first take ATM machine as a real-time example. We all use an ATM machine for cash withdrawal, money transfer, retrieve min-statement, etc in our daily life.

But we don’t know internally what things are happening inside ATM machine when you insert an ATM card for performing any kind of operation.

# Polymorphism in Java | Compile Time, Runtime, Example

**Polymorphism in java** is one of the core concepts of object-oriented programming language (OOPs).

The word polymorphism is derived from two Greek words: poly and morphs. The word “poly” implies many and “morphs” means forms.

Therefore, polymorphism means “many forms”. That is one thing that can take many forms.

Polymorphism is a concept by which we can perform a single task in different ways. That is, when a single entity behaves differently in different cases, it is called polymorphism in Java.

2. The best example of polymorphism is human behavior. One person can have different behavior. For example, a person acts as an employee in the office, a customer in the shopping mall, a passenger in bus/train, a student in school, and a son at home.

3. We all use a single button to switch ON and OFF the computer.

## Static Polymorphism in Java

A polymorphism that exhibited/show during compilation is called **static polymorphism in java**. In static polymorphism, the behavior of a method is decided at compile-time.

Hence, [*Java compiler*](https://www.scientecheasy.com/2021/03/java-compiler.html/) binds method calls with method definition/body during compilation. Therefore, this type of polymorphism is also called **compile-time polymorphism in Java**.

## Dynamic Polymorphism in Java

A polymorphism that is exhibited/display at runtime is called dynamic polymorphism in java. In dynamic polymorphism, the behavior of a method is decided at runtime,

therefore, the JVM (Java Virtual Machine) binds the method call with method definition/body at runtime and invokes the relevant method during runtime when the method is called.

## Overloading vs Overriding in Tabular Form

A list of differences between overloading and overriding in Java is given below for quick revision in tabular form.

|  |  |  |  |
| --- | --- | --- | --- |
| **SN** | **Property** | **Overloading** | **Overriding** |
| 1 | **Argument type** | Must be different (at least order). | Must be the same (including order). |
| 2 | **Method signatures** | Must be different. | Must be the same. |
| 3 | **Return type** | Same or different. | Must be the same until Java 1.4 version only. Java 1.5 onwards, Covariant return type is allowed. |
| 4 | **Class** | Generally performed in the same class. | Performed in two classes through Inheritance (Is-A relationship). |
| 5 | **Private/Static/Final method** | Can be overloaded. | Cannot be overridden. |
| 6 | **Access modifiers** | Anything or different. | Subclass method’s access modifier must be same or higher than superclass method access modifier. |
| 7 | **Throws clause** | Anything | If child class method throws any checked exception compulsory parent class method should throw the same exception is its parent otherwise we will get compile-time error but there is no restriction for an unchecked exception. |
| 8 | **Method resolution** | Always take care by java compiler based on reference type. | Always take care by JVM based on runtime object. |
| 9 | **Polymorphism** | Also known as compile-time polymorphism, static polymorphism, or early binding. | Also known as runtime polymorphism, dynamic polymorphism, or late binding. |
| 10 | **Performance** | Better | Less |

# <https://media.geeksforgeeks.org/wp-content/uploads/Structure-vs-Union.png>

he pointer in C language is a variable which stores the address of another variable. This variable can be of type int, char, array, function, or any other pointer. The size of the pointer depends on the architecture. However, in 32-bit architecture the size of a pointer is 2 byte.

Consider the following example to define a pointer which stores the address of an integer.

1. int n = 10;
2. int\* p = &n; // Variable p of type pointer is pointing to the address of the variable n of type integer.

# C goto statement

The goto statement is known as jump statement in C. As the name suggests, goto is used to transfer the program control to a predefined label. The goto statment can be used to repeat some part of the code for a particular condition. It can also be used to break the multiple loops which can't be done by using a single break statement. However, using goto is avoided these days since it makes the program less readable and complecated.

Syntax:

1. label:
2. //some part of the code;
3. goto label;

|  |  |
| --- | --- |
| 1. **static memory allocation** | **dynamic memory allocation** |
| memory is allocated at compile time. | memory is allocated at run time. |
| memory can't be increased while executing program. | memory can be increased while executing program. |
| used in array. | used in linked list. |

Now let's have a quick look at the methods used for dynamic memory allocation.

|  |  |
| --- | --- |
| **malloc()** | allocates single block of requested memory. ptr=(cast-type\*)malloc(byte-size) |
| **calloc()** | allocates multiple block of requested memory. |
| **realloc()** | reallocates the memory occupied by malloc() or calloc() functions. |
| **free()** | frees the dynamically allocated memory. |

## malloc() function in C

The malloc() function allocates single block of requested memory.

It doesn't initialize memory at execution time, so it has garbage value initially.

It returns NULL if memory is not sufficient.

The syntax of malloc() function is given below:

1. ptr=(cast-type\*)malloc(byte-size)

## calloc() function in C

The calloc() function allocates multiple block of requested memory.

It initially initialize all bytes to zero.

It returns NULL if memory is not sufficient.

The syntax of calloc() function is given below:

1. ptr=(cast-type\*)calloc(number, byte-size

## realloc() function in C

If memory is not sufficient for malloc() or calloc(), you can reallocate the memory by realloc() function. In short, it changes the memory size.

Let's see the syntax of realloc() function.

1. ptr=realloc(ptr, new-size)

## free() function in C

The memory occupied by malloc() or calloc() functions must be released by calling free() function. Otherwise, it will consume memory until program exit.

Let's see the syntax of free() function.

1. free(ptr)

# your company is working on this technology

# as I am a fresher I need a platfrom to implement my theoretical knowledge into

# practical knowledge .even your company mainly work on

# Capgemini is a global leader in consulting, digital transformation, technology and engineering services.

|  |  |
| --- | --- |
| The String class is immutable. | The StringBuffer class is mutable. |
| 2) | String is slow and consumes more memory when we concatenate too many strings because every time it creates new instance. | StringBuffer is fast and consumes less memory when we concatenate t strings. |
| 3) | String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method. | StringBuffer class doesn't override the equals() method of Object class. |
| 4) | String class is slower while performing concatenation operation. | StringBuffer class is faster while performing concatenation operation. |
| 5) | String class uses String constant pool. | StringBuffer uses Heap memory |
| No. | StringBuffer | StringBuilder |
| 1) | StringBuffer is synchronized i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously. | StringBuilder is non-synchronized i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously. |
| 2) | StringBuffer is less efficient than StringBuilder. | StringBuilder is more efficient than StringBuffer. |
| 3) | StringBuffer was introduced in Java 1.0 | StringBuilder was introduced in Java 1.5 |

# Threads : Multitasing done at the program level is call threads By using thread class : Here we inherited run method from thread class then we override it user define class To start run thread we used start method of thread class Run thread is user define thread whrer means thread dy default always main thread run first and then user define thread Which thread will executed how much time can not predicted by programmer as it is decided by the processor dependent on avability

# Interface can store only incomplete method By default every method is abs Every variable by default in interface public static and final Multiple inheritance support in interface Abs class can consist of both complete and inclompled method Every method by default not abstract we should used abstract key to spcifipe incomplete method Variable by default not final Multiple inheritance abs class not possible Inheritance btw abs class and interface possible

# Local reference variables:

# • These variables are created inside a method

# These variables can be used only within created method

# These variables are accessed directly with its name

# These variables if not initialized and used then it gives us an error

# Hasmap and hashtable

|  |  |  |
| --- | --- | --- |
| 1. | No method is synchronized. | Every method is synchronized. |
| 2. | Multiple threads can operate simultaneously and hence hashmap’s object is not thread-safe. | At a time only one thread is allowed to operate the Hashtable’s object. Hence it is thread-safe. |
| 3. | Threads are not required to wait and hence relatively performance is high. | It increases the waiting time of the thread and hence performance is low. |
| 4. | Null is allowed for both key and value. | Null is not allowed for both key and value. Otherwise, we will get a null pointer exception. |
| 5. | It is introduced in the 1.2 version. | It is introduced in the 1.0 version. |
| 6. | It is non-legacy. | It is a legacy. |

**thread in Java** simply represents a single independent path of execution of a group of statements. It is the flow of execution, from beginning to end, of a task.

When we write a group of statements in a program, these statements are executed by JVM one by one. This execution process is called thread in Java.

There is always at least one thread running internally in every program and this thread is used by JVM to execute statements in the program.

|  |  |
| --- | --- |
| * **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 bits 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. |

| [List](https://www.geeksforgeeks.org/list-interface-java-examples/) | [Set](https://www.geeksforgeeks.org/set-in-java/) | [Map](https://www.geeksforgeeks.org/map-interface-java-examples/) |
| --- | --- | --- |
| The list interface allows duplicate elements | Set does not allow duplicate elements. | The map does not allow duplicate elements |
| The list maintains insertion order. | Set do not maintain any insertion order. | The map also does not maintain any insertion order. |
| We can add any number of null values. | But in set almost only one null value. | The map allows a single null key at most and any number of null values. |
| List implementation classes are [Array List](https://www.geeksforgeeks.org/arraylist-in-java/), [LinkedList](https://www.geeksforgeeks.org/linked-list-in-java/). | Set implementation classes are [HashSet](https://www.geeksforgeeks.org/hashset-in-java/), [LinkedHashSet](https://www.geeksforgeeks.org/linkedhashset-in-java-with-examples/), and [TreeSet](https://www.geeksforgeeks.org/treeset-in-java-with-examples/). | Map implementation classes are [HashMap](https://www.geeksforgeeks.org/java-util-hashmap-in-java/), [HashTable](https://www.geeksforgeeks.org/hashtable-in-java/), [TreeMap](https://www.geeksforgeeks.org/treemap-in-java/), [ConcurrentHashMap](https://www.geeksforgeeks.org/concurrenthashmap-in-java/), and [LinkedHashMap](https://www.geeksforgeeks.org/linkedhashmap-class-java-examples/). |
| The list provides get() method to get the element at a specified index. | Set does not provide get method to get the elements at a specified index | The map does not  provide get method to get the elements at a specified index |
| If you need to access the elements frequently by using the index then we can use the list | If you want to create a collection of unique elements then we can use set | If you want to store the data in the form of key/value pair then we can use the map. |
| To traverse the list elements by using Listlterator. | Iterator can be used traverse the set elements | Through keyset, value, and entry set. |