A screenshot of a computer

Description automatically generated  
A screenshot of a computer

Description automatically generated  
==: Equal to
!=: Not equal to
<: Less than
<=: Less than or equal to
>: greater than
>= greater than or equal
5. Logical Operators
&&: Logical And
||: Logical or
!: Logical not
6. Ternary operator
condition ? if true : if false

Stack and heap memory:
Stack Memory only stores the local variables of the method
Each method have its own stack memory
Heap memory is a separate memory where it stores the object, and the address of the object is stored in the respective stack memory variable of the object.


Program to understand stack and heap memory

        class calculator {
Int num = 10;

Public int add(int num1, int num2){
Return num1+num2;
}
}

Public static void main (int a[]){
Int n1 = 10;
Int n2 = 20;
Calculator c1 = new calculator();
C1.add(n1, n2);
}


  
A white paper with black text and black text

Description automatically generated  
Obj1.name = "Smart Phone";

Mobile obj2 = new Mobile();
Obj2.brand = "Samsung";
Obj2.price = 1000;
Obj2.name = "Phone";

SOP(obj1.name);
SOP(obj2.name);
}
}

Output: Phone
    Phone

Since name is static variable and shared across all the objects, its printed as Phone for both obj1 and obj2.


Static Methods:
Static methods are the members of a class like non-static methods.
The static methods has access to only static variable, that means we cannot access non-static variables inside the static methos. The reason behind this  non-static variables contains different value for different objects but the static method called by class name, so the static method will not understand which object's value should be considered
We can solve the above issue by passing object as a parameter to the static method and then we can use the object reference to use non-static variables.

Ex: 
Class Mobile {
String brand;
Double price;
Static String name;

Public void show(){
SOP(brand + "-" + price + "-" + name);
}

Public static void show1(Mobile obj){
SOP(obj.brand + "-" + obj.price + "-" + obj.name);
}
}

Class Demo{
Public static void main(String a[]){
Mobile obj1 = new Mobile();
Obj1.brand = "Apple";
Obj1.price = 1500;
Obj1.name = "Smart Phone";

Mobile obj2 = new Mobile();
Obj2.brand = "Samsung";
Obj2.price = 1000;
Obj2.name = "Phone";

//call show method
Obj1.show();
  
Obj1.show();
Obj2.show();

//call show1 mothod(static)
Mobile.show1(obj1);
Mobile.show1(obj2);

}
}

Explanation: in the above program, show method is a non-static method and it is called by object reference and it can access all the static and non-static variables, but the method show1 is a non-static method and it is called by class name so it cannot access non-static variables until the object is passed as a parameter to the method.


Static block:

Static block is used to initialise the static variables of a class.
Static block is called only once irrespective of how many objects we create.
Class is loaded only once when we are creating an object of a class for the first time, at that time static block is executed. That means when we are creating second object again for the same class, then since class is already loaded while creating first object so class will not load again and hence same with the static block.

Ex:
Class Mobile {
String brand;
Double price;
Static String name;

Static {
Name = "Phone";
}
}

Getters and setters method:

If a class is having private variables which may be a static or not-static then we cannot directly access the those variables, instead we should use these getters and setters method to get and set the value for those variables.

Program:

Class Human {
String name = "Charan";
Private Int age = 27;
Private Double salary = 10000.00;

Public int getAge(){
Return age;
}

Public void setAge(int a){
Age = a;
  
Age = a;
}
Public double getSalary(){
Return salary;
}

Public void setSalary(double sal){
Salary = sal;
}

}

Class demo {
Public static void main (String a[]) {
Human hum = new Human();
//directly access name variable since it’s a public variable.
SOP(h1.name);
hum.setAge(28);
SOP(hum.getAge());
SOP(hum.getSalary());
}
}


This keyword:

This keyword refers to the current object.

Ex:
Class human {
Private Int age;
Public void setAge(int age){
This.age = age;
}
}


Constructor:

Constructor are the special methods of a class used to initialise the instance variables.
Name must be same as the class name
It wont return anything
Each time object is created, constructor will be called by default.
If we don’t write constructor method, java will call constructor behind the scene.

Ex:
Class human {
Private Int age;
Public human {           // default constructor
Age = 25;
}

Public human (int age) {  //parameterised constructor
This.age = age;
}
}

  
}


Inheritance:

In java, multiple inheritance is not allowed since ambiguity problem

Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings


**This and super keyword:**

* every constructor method of a class has statement as super();

Class A {

Public A() {

Super(); // by default first statement , calls constructor of super class

//constructor of class A

}

}

Class B extend A {

//constructor of class B

Public B() {

Super(); // by default first statement , calls constructor of super class

}

//parameterized constructor

Public B (int n) {

This(); // calls default constructor

Super(); //calls super class constructor

}

}

**Method overriding:**

* having same method name in the extended class, but changing the logic in extended class.

**Package:**

* package is nothing but folder where .java files are present.
* by default package java.lang.\* will be imported in all java files.
* To include any files into a package then we should write it as

Package tool.

* To import any files present in other package to current file the we should write as

Import tools.calculator -> imports calculator file only.

Import tools.\* -> imports all the files present in tools folder

**Access Specifiers:**

These access specifiers specifies where variables can be accessed

- Public

- Protected

- private

- default

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Private | Protected | Public | Default |
| Same class | Yes | Yes | Yes | Yes |
| Same Package  Sub Class | No | Yes | Yes | Yes |
| Same Package  Non-Subclass | No | Yes | Yes | Yes |
| Different Package  Sub Class | No | Yes | Yes | No |
| Different Package  Non-subclass | No | No | Yes | No |

**PolyMorphism:**

Polymorphism means behave differently at different places based on some conditions.

Ex: we behave differently in home, office, with friends, etc..

Tyepe 1. Compile-Time polymorphism ex: overloading

2. Run-Time Polymorphism ex: Overriding

**Dynamic Method Dispatch:**

It is a best example for Run-Time Polymorphism,

A screenshot of a computer program

Description automatically generated

In the above example where class B extends class A, when object of class A created and assigned to variable of type A. But later object of class B is created and assigned to variable obj which is of type A but still works and you can see the output in the image.

The conclusion is that same variable obj behaving differently at different point of time.

**Final:**

Keyword can be used with variable, methods and class.

* The moment we use final with variable, then the value assigned to variable cannot be changed.
* If final keyword is used with class name, then nobody can extend the final class
* If final keyword is used with method name, then nobody can override that method.s

**Object Class:**

All the class that we create in java will bydefault extends Object Class.

* Object class has so many methods like toString, equals, hasCode, etc..
* We can override these object class methods.

**Type-Casting:**

2 types

1. Up-Casting
2. Down-Casting

* Incase of basic java the definition is process of converting the value of a data type into another data type

Ex:

Double n2 = 4.5;

Int n1 = (int) n2;  // n1 has value 4, we lose .5 here.

* Incase of oops world, definition is process of converting object reference

Ex: downcasting

A a = new A();

B b = new A(); // object of class A is assigned to subclass reference variable.

**Wrapper class:**

Since we have primitive data types in java which are int, double, float…we cannot call java as completely 100% oops. To overcome this we have class to convert these primitive type into object type.

1. Boxing: converting primitive type into object type
2. Unboxing: converting object type into primitive type

Ex: int n1 = 10; // primitive type

Interger n2 = new Integer(n1); // converting to object -> boxing

Int n3 = n2.intValue(); // converting back to primitive -> unboxing

**Quiz Application:**

<https://github.com/Gaurav560/quiz-console-app/tree/master/QUIZ-CONSOLE-APP>

**Abstract:**

* A method is called as abstract method if the method is declared but not defined(not implemented). Those methods are declared with abstract keyword
* A class which has abstract method will become abstract class.
* If a class is declared with abstract, it is not compulsory that it should have abstract method.
* We cannot create a object of abstract method
* If a class extends from abstract method then the extended class must implement all the abstract method or else that class also should be declared with abstract keyword.

Abstract class car {

Public abstract void drive();

Public abstract void fly();

}

Abstract Class WagonR { class UpdatedWagonR{

Public void drive(){ public void drive(){

SOP(“Driving…”) SOP(“Driving..”);

} }

} public void fly(){

SOP(“Flying”);

}

}

**Inner Class:**

Defining a class inside other class is allowed in java,

* We can make that class as a static also
* To access the method and variables of inner class,
* If inner class is a static class, then object of outer class is not needed to create the object of inner class
* If inner class is non-static, then object of outer class is needed to create the object of inner class.

A screenshot of a computer screen

Description automatically generated

**Anonymous Class:**

If I want to change the implementation of a method, but I don’t want to use override concept, then we can use these class while creating object.

* Here variable obj is not a object of class A, but object of another class.
* Anonymous class by default extended from class A.

A computer screen shot of text

Description automatically generated

**Abstract and Anonymous:**

We can use concept of anonymous class to implement abstract methods of abstract class

A computer screen shot of code

Description automatically generated

**Interface:**

Interface are nothing but like a class, but all the methods inside the interface are by default public abstract methods. And also all the variables(member variables) are by default public and static, because its not possible to create object of a interface then there is no meaning in having non-static member variables(we can access static members by class or interface name).

* To extend interface, we should use implements keyword.

A screen shot of a computer program

Description automatically generated

* An interface can extend another interface with keyword ‘Extends’,
* A class can extend multiple interface with comma separated

**Keyword:**

Class to Class -> extends

Class to Interface -> implements

Interface to Interface -> Extends

A screen shot of a computer program

Description automatically generated

**Enum Type:**

An enum is a special "class" that represents a group of constants

* We cannot extend any other class from enum
* Enum by default extends from enum class
* Enums will contain construction and method similarly like other class.

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screenshot of a computer code

Description automatically generated

**Types of Interface:**

1. Marker Interface – has no abstract method (zero methods)
2. Functional/SAM (single Abstract Method) – has only one abstract method
3. Normal Interface – has two or more abstract methods

Functional Interface:

* Lambda expressions only works with functional interface

**Lambda expressions:**

Lambda Expressions were added in Java 8. A lambda expression is a short block of code which takes in parameters and returns a value.

A screenshot of a computer screen

Description automatically generated

**Exceptions:**

Errors we get in any language is

1.compile time error

2.Run time error

3.Logical error

Basically exceptions are run time error, we have a procedure or phenomenon of handling there errors are called as exception handling.

There are two types of statements in programming language.

1.Normal statement ex- int i=9;

2.critical statement ex: int j = i/k;

We use try and catch block to handle the exceptions

Int I = 19;

Int j = 0;

Try {

J = i/j; //critical statement

} catch (exception e){

SOP(“Something went wrong”)

}

Multiple Catch Block:

Handling different types of exception ex: null pointer exception, arrayindexboundout exception, etc…

* Exception is base class of all exception
* Exception class extends from Throwable class

A computer screen shot of a program

Description automatically generated

**Hierarchy of Exception:**

A diagram of an object

Description automatically generated

Its not necessarily to be handle Unchecked Exception, where as Checked Exception are necessary to handle.

**Throw Keyword:**

Throw keyword is used to throw an exception and the exception is handled in catch block.

**Custom Exception:**

We can create our own exception in java, refer below program for creating own exception class

* Every custom exception extends from Exception class.
* If custom exception has a message, its passed to super class constructor

A screenshot of a computer program

Description automatically generated

**Throws Keyword:**

Instead of handling the exception with try and catch block, we can use Throws keyword to ducking the exception(means ask the method which is calling this current method to handle the exception)

**User Input from Buffer reader and Scanner:**

A screen shot of a computer program

Description automatically generated

Using BufferReader

A screenshot of a computer program

Description automatically generated

Using Scanner

Finally block:

It is used commonly to be close the all the resources which are opened.

**Threads:**

A thread in Java is the direction or path that is taken while a program is being executed. Generally, all the programs have at least one thread, known as the main thread, that is provided by the JVM or Java Virtual Machine at the starting of the program's execution.

* In-order to make a normal class into thread class, the class must extended from Thread class
* Thread class had method call,start which internally calls Run method the class through which Thread class is extended.

class A extends Thread

{

    public void run()

    {

        for(int i=1;i<=100;i++)

        {

            System.out.println("Hi");

        }

    }

}

class B extends Thread

{

    public void run()

    {

        for(int i=1;i<=100;i++)

        {

            System.out.println("Hello");

        }

    }

}

class Demo {

    public static void main(String[] args) {

        A obj1=new A();

        B obj2=new B();

        obj1.start();

        obj2.start();

    }

}

**Thread Priority:**

We can set the priority for the thread so that scheduler will understand which thread needs to be encountered first.

* Priority ranges from 1 to 10 where 10 is highest priority and 1 is lowest priority.
* We can just give the priority to threads, but its finally scheduler who execute the thread based on different algorithms, like if we have a thread which takes less time will be executed first.
* We can make the thread to sleep mode by using Thread.sleep(10) \*milli seconds, when one thread is sleeping, scheduler will schedule the execution of another active thread.

<https://github.com/navinreddy20/Javacode>

**Runnable vs Thread:**

* Thread class actually implemented from Runnable interface, so instead of extending the Thread class I can implement Runnable interface
* But start method present in Thread class, so in order to start the thread, create a thread object and pass runnable object to the thread.
* We can use lambda expression to implement Runnable interface and provide define run method.

**Race Condition(Threads and mutatble):**

* Whenever we work with threads, its not a recommended to use mutable variables
* We use join method to join two threads so that main method waits for all joined threads to complete the execution.
* Inorder to make the method to wait for another thread when one thread is using we should use synchronized

Public void synchronized count (int count) {

Count++

}

<https://github.com/navinreddy20/Javacode/blob/main/19.5%20Race%20Condition.java>

**Thread States:**

**New** -> Whenever we create a thread ex: Thread t1 = new Thread();

**Runnable** -> soon after t1.start() method, thread will go into Runnable state

**Running** -> once scheduler schedules the execution thread will go into Running state.(run method)

**Waiting**->if we stop execution of thread using sleep or wait method, thread will go waiting state and in order to resume the execution we should use notify method and thread will go Runnable state first then to Running upon availability of CPU time with scheduler

**Dead->** once after the execution thread will automatically go dead state, or we can force by using stop() method.

**Collections:**

There are three types, it belogs to java.util package

1.collection API

2.collection

3.collections

Collections has there own advantage and disadvantage over the arrays.

| **Sr. No.** | **Key** | **Arrays** | **Collection** |
| --- | --- | --- | --- |
| 1 | Size | Arrays are fixed in size i.e once the array with the specific size is declared then we can't alter its size afterward. | The collection is dynamic in size i.e based on requirement size could be get altered even after its declaration. |
| 2 | Memory Consumption | Arrays due to fast execution consumes more memory and has better performance. | Collections, on the other hand, consume less memory but also have low performance as compared to Arrays. |
| 3 | Data type | Arrays can hold the only the same type of data in its collection i.e only homogeneous data types elements are allowed in case of arrays. | Collection, on the other hand, can hold both homogeneous and heterogeneous elements. |
| 4 | Primitives storage | Arrays can hold both object and primitive type data. | On the other hand, collection can hold only object types but not the primitive type of data. |
| 5 | Performance | Arrays due to its storage and internal implementation better in performance. | Collection on the other hand with respect to performance is not recommended to use. |

Collection

Dequeue

LinkedHashSet

HashSet

ArrayList

LinkedList

Queue

Set

List

**List:**

* List can have duplicate values in it. And also support index of the element
* It also allows null elements
* ArrayList, LinkedList, Stack, Vector….

**Set:**

Set can only have unique values in it and does not support index.

**Map:**

In Java, Map Interface is present in [java.util](https://www.geeksforgeeks.org/java-util-package-java/) package represents a mapping between a key and a value. Java Map interface is not a subtype of the [Collection interface](https://www.geeksforgeeks.org/collections-in-java-2/). Therefore it behaves a bit differently from the rest of the collection types.

* **No Duplicates in Keys**: Ensures that keys are unique. However, values can be duplicated.
* **Null Handling**: Allows one null key in implementations like [HashMap](https://www.geeksforgeeks.org/java-util-hashmap-in-java-with-examples/)and [LinkedHashMap](https://www.geeksforgeeks.org/linkedhashmap-class-in-java/" \t "_blank)**and a**llows multiple null values in most implementations.
* **Thread-Safe Alternatives**: Use [ConcurrentHashMap](https://www.geeksforgeeks.org/concurrenthashmap-in-java/" \t "_blank)for thread-safe operations also, wrap an existing Map using Collections.synchronizedMap() for synchronized access.

There are interfaces that extend the Map implementation namely [SortedMap](https://www.geeksforgeeks.org/sortedmap-java-examples/" \t "_blank) and [NavigableMap](https://www.geeksforgeeks.org/navigablemap-interface-in-java-with-example/" \t "_blank).

**A diagram of a map hierarchy

Description automatically generated**

**Comparator vs Comparable:**

these are used while we use sort method.

**Foreach loop:**

How it works is, forEach method of a collection required a consumer class, and class can be anonymous.

Ex:

Consumer<Integer> com = n-> SOP(n);

**Stream:**

* Java stream simplifies the processing of collections and sequences of data. It enables to perform complex operations of data by using filter, map, reduce,sort, etc…
* Stream is used only once and after the use stream is closed and you cannot use it.

**Parallel Stream:**

**Optional Class in Java:**

* Introduced in java 1.8
* This class is used to avoid the null pointer exception.

**Method Reference:**

Method references, introduced in Java 8, are a concise shorthand for referring to methods or constructors. Think of them as a way to say, "Hey Java, use this method here!" without writing out the entire method call. They work hand-in-hand with lambda expressions to bring a touch of functional programming to Java.

**Constructor reference:**

Constructor reference works similar to method reference,

In order to create new object of a class by writing “new ClassName()”, we can use “ClassName::new”.

**Maven:**

Maven is a project management tool, when we build a project we need to compile, run,etc…

* It also provides all the respective libraries
* Provides plugin to compile, run,etc..

**JDBC (Java Data Base Connection):**

* We use postgresql

**Steps to connect Data base:**

1. Import package
2. Load driver
3. Register driver
4. Create connection
5. create a statement
6. Execute a statement
7. Close connection

After JDBC 4.0 load and register of driver is automatically done by the framework.

**Prepared Statement:**

PreparedStatement is a interface which is sub interface of Statement, It is used to execute parameterized query.

**Spring Framework and Spring boot:**

<https://www.techtarget.com/searchapparchitecture/definition/Spring-Framework>

* Have features
* Good community
* Good documentation

Pre-requisite:

* Core java
* Oops concept
* Thread
* Collection API
* JDBC
* Build Tool (maven, grandle, etcc)
* ORM

**IoC(Inversion of Control):**

We are inverting the control, as a programmer we create object and flow of the application. We create the object then we should handle the maintenance of the object, instead we can give this controlling the maintenance of object to someone else who is IoC. IoC container will have the objects.

* Here we use design pattern Dependency Injection(DI) to handle the objects(to perform the IoC). Injecting the dependency objects into our program.

Ex: we have a two class, Laptop and CPU, where Laptop class is having dependency on CPU object since Laptop cannot work without CPU, here DI helps to inject CPU object into Laptop class. This process is IoC

* Spring is a framework on top of the we have spring boot because, to create a spring project it requires lot of configuration just to create a hello world project, this is avoided using spring boot, which create a readymade java project.

**Application Context in Spring Framework:**

Spring supports several methods to configure the Application Context

1. Xml configuration
2. Java-based configuration
3. Annotation-based configuration

**1.xml-based configuration**

ApplicationContext context = new ClassPathXmlApplicationContext(“spring.xml”);

Alien obj = context.getBean(“idAlien”);

Where spring.xml file contains below code

<beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="  
 http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">

<bean id=”idAlien” class=”com.example.Alien”>

</beans>

**Note:** in this type, all object will be created when we call “new ClassPathXmlApplicatioContext”.

If we have two beans for the same class in xml file, two object Is created for the same class.

**Scope:**

There are two scope in spring framework

1. Singleton – no matter how many times we call getBean method only one object is created

Alien obj1 = context.getbean(“Alien”);

Alien obj2 = context.getBean(“Alien”);

Where obj1 and obj2 both referring to same object of Alien

1. Prototype – each time object will be created whenever we call getBean method.

Alien obj1 = context.getbean(“Alien”);

Alien obj2 = context.getBean(“Alien”);

Here obj1 and obj2 both are two different objects.

**Setter Injection:**

In order to set the value for the properties of a class, we use xml file like below

Class Alien {

Private int age;

//write setter and getter for age

}

**In XML file**

<bean id=”alien” class=”com.charan.Alien”>

<property name=”age” value=”21”/>

</bean>

* Bean uses setters method to set the value 21 which is mentioned in xml file.

**Reference Attribute:**

This is used to assign object to the non-primitive property

-refer sprinFramework project in git for the code

**Constructor Injection:**

We can use parameterized constructor with the help of xml file and constructor-args tag

**Create Interfaces:**

-since alien class is dependent on laptop or desktop, we can create a interface computer which will be extended to laptop and desktop.

**Autowire:**

**Autowiring** in the Spring framework can inject dependencies automatically. The Spring container detects those dependencies specified in the configuration file and the relationship between the beans. This is referred to as **Autowiring in Spring**. To enable Autowiring in the Spring application we should use[@Autowired](https://www.geeksforgeeks.org/spring-autowired-annotation/)annotation. Autowiring in Spring internally uses constructor injection. An autowired application requires fewer lines of code comparatively but at the same time, it provides very little flexibility to the programmer.

**NOTE:** incase of autowire=”byType” and we have two beans of same type, then we use primary attribute to solve the confusion of which type should be considered.

**Lazy-init bean:**

If the bean is mentioned as a lazy-init , then the object is only created when someone is trying to use it. By default object for that bean will not be created.