Introduction to Java

* Java is an object-oriented programming language developed by Sun Microsystems, and it was released in 1995.
* James Gosling initially developed Java in Sun Microsystems (which was later merged with Oracle Corporation).
* Java is a set of features of C and C++. It has obtained its format from C, and OOP features from C++.
* Features Of Java
* Object-Oriented - The features of object-oriented programming are supported by Java. Its object model is straight forward and flexible.
* Platform independent – The created code is run in any other OS . (Because when Java is complied its not convert direct into MLL ,rather then its convert into Byte code (it is a Platform independent .We can upload in internet then access other OS like Linux , Mac etc .) That’s why Java is a platform independent . And it is also called **WORA**. (write one time run anywhere)
* Portable - We have the idea of portability in Java. Java allows the same software to run on various platforms.
* Simple - Because Java incorporates many C/C++ capabilities, it is simple to understand.

* Main method in Java:
* public: An access specifier, that is. Before calling the main () method, we need to use the public keyword to let the JVM know where the programme is actually being executed. Before the main () method, if we use private, protected, and default, the JVM won't be able to see it.
* static: We should call the main () method without creating an object. Static methods are the method which is invoked without creating the objects, so we do not need any object to call the main () method.
* void: In Java, every method has the return type. Void keyword acknowledges the compiler that the main () method does not return any value.
* Main (): It is a default signature which is predefined in the JVM. It is called by JVM to execute a program line by line and end the execution after completion of this method. We can also overload the main () method.
* String args[]: The main() method also accepts some data from the user. It accepts a group of strings, which is called a string array. It is used to hold the command line arguments in the form of string values.
* Class - A class is a group of objects which have common properties. It is a template or blueprint from which objects are created. It is a logical entity.
* Object - An object is an entity with state and behaviour, such as a chair, bike, marker, pen, table, or car. It could be intellectual or physical (tangible and intangible). The banking system is an illustration of an intangible entity.
* An object has three characteristics:-

1. State:- represents an object's data (value)
2. Behavior :- represents how an object behaves (or how it functions), such as when you deposit or withdraw money.
3. Identity:- Usually, a distinct ID is used to implement an object's identification. The external user cannot see the value of the ID. However, the JVM uses it internally to uniquely identify each object.

* Different way of Object Definition:-

1. An object is a real-world entity.
2. An object is a runtime entity.
3. The object is an entity which has take and behaviour.
4. The object is an instance of a class.

* Statically typed language:- Whenever we perform a operation, variable , and array or any thing to store a info. Where we spaceify the type of info. Whether it is character , String , int etc . EX- Java , C, C++

Int marks = 45; or String name = “Kimmy”;

* Dynamically typed language :- Where we not perform any data type to define the variable. EX – JavaScript , PHP , Python .

Marks = 45; or name = “Roger”;

* Variable :- A Variable is the title of h reserved region allocated in memory.
* It is a container that holds the value while the Java program is executed
* Each variable Should be given h unique name to indicate the Storage area
* A variable is assigned with a data type.
* Data Type :- Data types specify the different sizes and values that can be stored in the variable. Based on the data type of a variable, the operating system allocates memory and decides what can be stored in the reserved memory. Hence, by assigning different data types to variables, we can store integers, decimals, or characters in these variables .
* Primitive data types :- A primitive type is predefined by the language and is named by a reserved keyword .
* Non-Primitive data type :- String, Array, class .

* Type casting -- change the data one form to another form . When we change the big data type (like-double) to small data type(int), so double must be change her nature meanwhile double must change her nature into a int . Explicit type casting is possible but here the change to lost the data . When we divide int /int the result came into the int value. If the precise value the value is truncated (gone or zero).
* Implicit casting:- Implicit casting also known as widening conversion, happens automatically when a smaller data type is converted to a larger data type. Java automatically converts the smaller data type to the larger data type without any data loss. For example, if you assign an int value to a long variable, Java will implicitly cast the int value to a long value.

int i = 10;  
 long l = i ; // Implicit casting from int to long

* Explicit casting:- Explicit casting also known as narrowing conversion, occurs when you convert a larger data type to a smaller data type. It involves manually converting one data type to another data type using a type cast operator. Since the smaller data type cannot accommodate the larger data type, data loss may occur during explicit casting. Therefore, explicit casting should be used with caution.

double d = 10.5;  
int i = (int) d; // Explicit casting from double to int

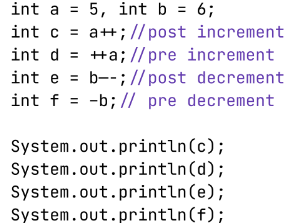
* Also important to note that when performing arithmetic operations on short values, the result is **automatically promoted** to an int. So, you may need to explicitly cast the result back to a short if you want to store it in a short variable.

short x = 10; x = (short) (x \* 5); System. out. print(x);

1. short x = 10;

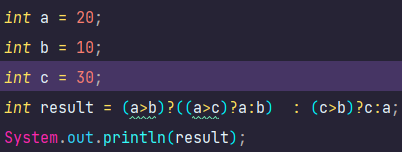
2. x = x \* 5; Show CE

3. System. out. print(x);

* Increment and Decrement ,Operator:-

Output :-



* Ternary operator (?:) :- It is a smaller version for the if-else statement. If the condition is true then the statement - 1 is executed else the statement - 2 is executed.

Syntax :- condition ? statement - 1 : statement - 2

* Loop :- Loop make it easy for us to tell the computer that a given set of instruction need to be executed repeatedly.

1. The while loop :- A while loop is a loop that runs through its body, known as a while statement, as long as a predetermined condition is evaluated as true.

Syntax :- while (condition) {

statement; }

1. The for loop :- Unlike while loop, in for loop we have 3 parts in the for header.

Syntax :- for (Initialize-statement; condition; final-expression) {

Statement }

1. The do-while loop :- In other words, the loop is executed at least once before the condition is checked.

Syntax:- do {

statement;

} while (condition);

* **OOPS Fundamental :**
* Object Creation : 1. We Can use the "new" operator to create an object.

2. There is no “delete” operator in java because destruction of useless objects is the responsibility of the garbage collector.

class Test

{

String name;

int a;

public static void main(String[] args) {

Test t = new Test();

}

}

* Behind the scenes of Object Creation:-

1. new operator is a keyword which is use to create the Object.

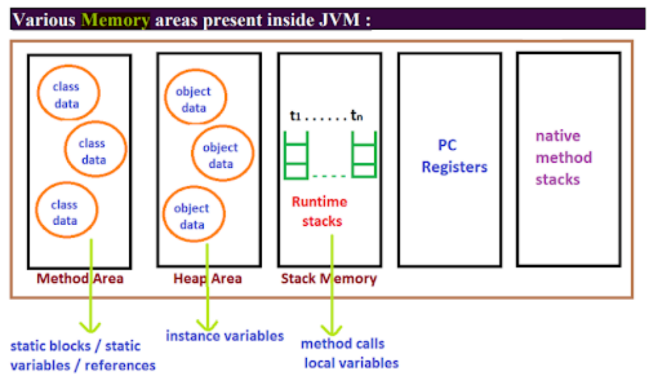
2. When we say new operator JVM would allocate memory on the heap area.

3. JVM will load the supplied class name data into the method area.

4. JVM will initialise the memory of instance variables.

5. JVM will set the default value for instance variables based on the data type . Once the memory is set then the address of the object will be stored inside the reference variable.

* Methods with memory maps (JVM data areas):-



1. class level binary data including static variables will be stored in method area.
2. Objects and corresponding instance variables will be stored in the Heap area.
3. For every method the JVM will create a runtime stack, all method calls performed by that Thread and corresponding local variables will be stored in that stack. Every entry in stack is called Stack Frame or Action record.
4. The instruction which has to execute next will be stored in the corresponding PC registers.
5. Native method invocations will be stored in native method stacks.

* Instance variables:- 1).- Instance variables will be created at the time of object creation and destroyed at the time of object destruction hence the scope of instance variables is exactly the same as scope of objects.

1. Instance variables will be stored on the heap as the part of the object.
2. Instance variables should be declared within the class directly but outside of any method or block or constructor.

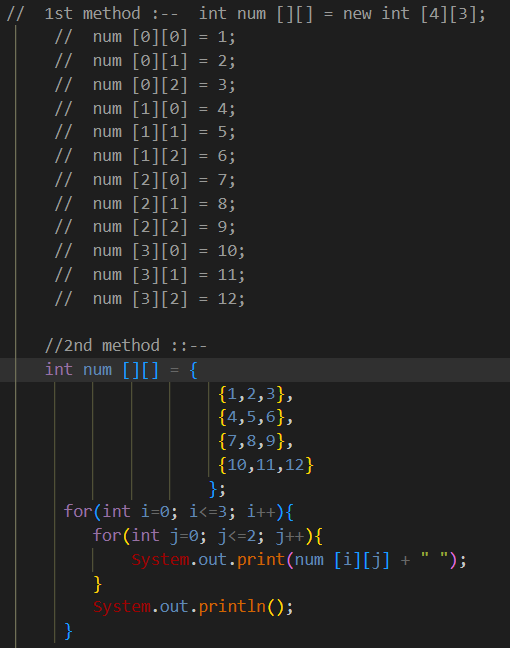
* Local variables;- 1). Sometimes to meet temporary requirements of the programmer ,we can declare variables inside a method or block or constructors such type of variables are called local variables or automatic variables temporary variables or stack variables.

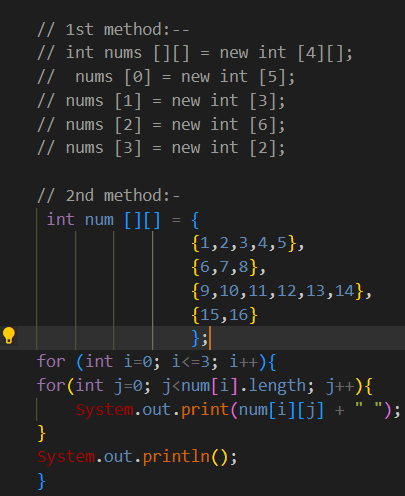
2). Local variables ,will be stored inside the stack.

* Method Overloading :- 1) Having the same name and different argument types is called method overloading. All these methods are considered as overloaded methods.

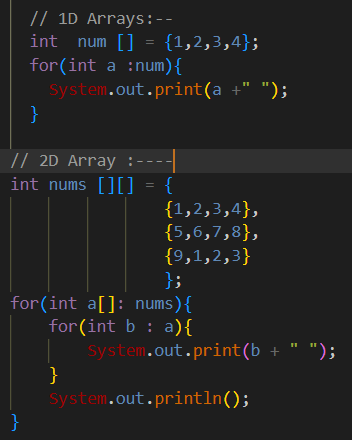
**2) Having overloading concept in 2ava reduces complexity of the programming.

1. *Automatic promotion in overloading.-*

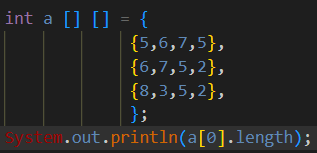
* Array :-
* 2D Array :-
* Jagged Arrays:- When we not sure about the row in matrix but sure about the column.



* Foreach/Enhanced For loop :- Syntax:- for(data\_type variable: array){

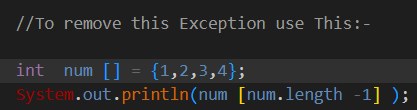


* length vs length() :- length: It is a property of the Array type class.

 FOR 2D Array Length :- when we find out in particular index [0]



length(): It is a method of String class.

* ArrayIndexOutOfBoundsException:-If we try to access the index which is not in the array range then it would result in an Exception called

“ArrayIndexOutOfBoundsException.

* **Types of String :--**

1. Immutable String :- Once if we create a String, on that String if we try to perform any operation then those changes won’t be reflected in the same object, rather a new object will be created/Such type of String is called as “Immutable String”. {Others notes are Handwritten – like String constant pool}

* Different ways of Comparison:-
* To compare 2 String in java we use following approach:-

See example in java notebook.

1. == operator:-

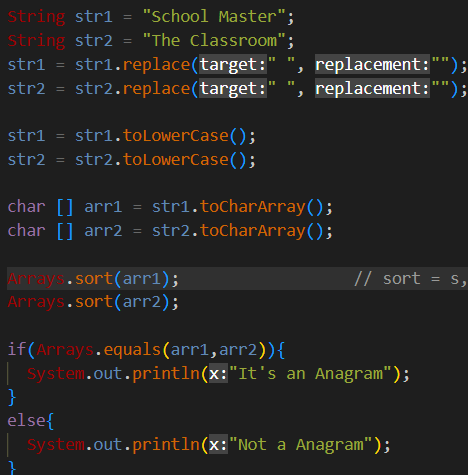
It compare the references of the Object.

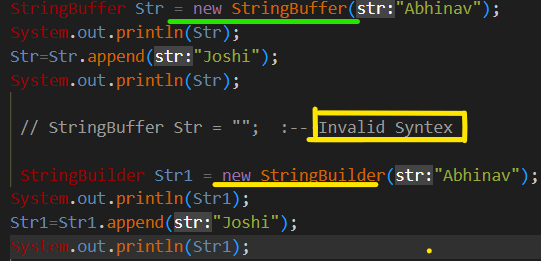
2. equals():-

It compare the content of two object.

* Concatenation:- Concatenation is the process of combining two or more strings into a single string. This can be done in multiple ways, including using the

1. "+" operator :- Any number of string object will be added or concat.
2. concat () method :- only one object will be concat.

* Anagram String:-Two strings are said to be anagram if we can form one string by arranging the characters of another string. For example, Race and Care.

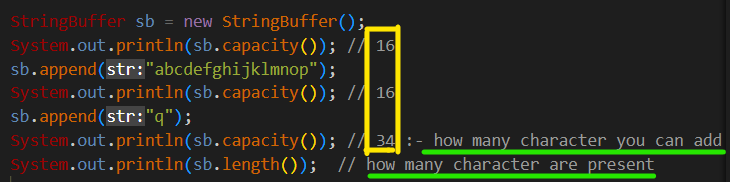
1. Mutable String:- Once if we create a String, on that String if we try to perform any operation and if those changes get reflected in the same object then such strings are called “Mutable String”{obj. are not refer a new memory}.
   * String Buffer:- a) -If the content will change frequently then it is not recommended to go for String object becoz for every new change a new Object will be created.

b) -To handle this type of requirement, we have a String Buffer/StringBuilder concept.

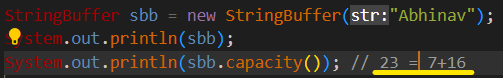
1.**StringBuffer sc = new StringBuffer(\_);**

* creates an empty String Buffer object with default initial capacity of "16".
* Once String Buffer reaches its maximum capacity a new String Buffer Object will be created.

new capacity = (current capacity+1) \* 2;



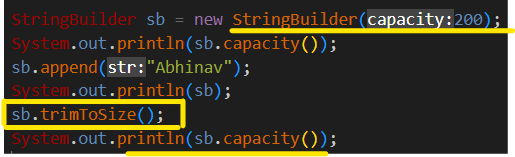
2.**StringBuffer sb=new StringBuffer(String s);**

* It creates a String Buffer object for the given String with the;-



capacity = s.length() + 16;

3. StringBuffer sc=new StringBuffer(initialCapacity);

* It creates an empty String with the specified initial capacity.



* String Buffer VS String Builder :

|  |  |
| --- | --- |
| String Buffer | String Builder |
| JDK 1.0 Version | 1. JDK 1.5 Version |
| Method Synchronized (Multithreading) | 2. Method not Synchronized (Multithreading) |
| Thread Safe | 3. At a time more than one thread can ope- rate so it is not Thread safe. |
|  | 4. Threads are not required to wait so performance is high. |

* **‘Static’ Keyword** :- The static keyword is mainly used for memory management in Java. A static keyword can be applied to variables, blocks, methods, and classes. The static keyword is used for a constant variable or a method that is the same for every instance of a class.

1. Static Variable :- If we declare any variable as static, then it is called a static variable. When a variable is declared as static, then a single copy of that variable is created and shared among all of the objects at the class level. Static variables are global variables. All instances of the class share the Case static variable.
2. Static Method:- A static method is a method that belongs to a class rather than an instance of a class. This means you can call a static method without creating an object of the class. Static methods are sometimes called class methods.

* There are a few other reasons why you might want to use static methods:
  1. You can access static methods from outside if the class in which they are defined This is not possible with non-static methods.
  2. Subclasses can override static methods, but non-static methods cannot.
  3. Static methods are executed when an instance of the class is created, whereas non-static methods are not.
* How Java Program Actually executes:-
* Class Loading :- In Java, class loading is the process of loading class files into the JVM (Java Virtual Machine) at runtime. It is responsible for loading classes from various sources, such as the file system, network, and databases, and making them available to the JVM for execution.
* The class loading process in Java is divided into three phases:

1. loading,

2. linking,

3. initialization.

* Difference with respect static and non-static members of a class
* Static:-

1. These variables are called “class variables”.
2. These variables will get memory in the method area.
3. If the value does not change from object to object then we need to use static variables.
4. Inside a static area we can access static variables only.
5. Static variables are created using static keywords.

* Non-static:-

1. These variables are called “instance variables”.
2. These variables will get memory in the heap area.
3. If the value changes from object to object then we need to use “non- static” variables.

* **OOPS IN JAVA**
* **Encapsulation:- 1.** Binding of data and corresponding methods into a single unit Is called "Encapsulation".

**2.** If any java class follows data hiding and abstraction then such class is referred as "Encapsulated class".

**3.** Every data member inside the class should be declared as private, and to access this private data we need to have getter and setter methods.

Encapsulation = Data Hiding + Data Abstraction.

* Advantage of Encapsulation:-

a. We can achieve security.

b. Enhancement becomes easy.

c. Maintainability and modularisation becomes easy.

d. It provides flexibility to the user to use the system very easily.

* Shadowing Problem and this keyword:- If both local variable and

instance variable have the same name inside the method then it would result in a name-clash and Jvm will always give preference for local variable. This approach is called the “Shadowing problem”.

--🡪 Show Example in java Note book/PDF.

1.Class student {

2.int age ; 🡪 Instance variable

3.Void data(int age){ 🡪 Local variable

4.age = age; --🡪 that’s called Shadowing where instance and local variable

are same . To remove this Shadowing Problem we use this

keyword . **RUN == 0; {Result**}

5. this.age = age ; { RUN= Which data we give in main class.}

* **Setters :-** Setter methods are used to set the value to the instance

variables of the class.

* Syntax for setter method :-

1. compulsory the method name should start with set.
2. it should be public.
3. return type should be void.
4. compulsorily it should have some argument.

* **Getter :-** Getter methods are used to get the value from the instance

variables of a class**.**

* Syntax for getter method:-

1. compulsory the method name should start with get.
2. it should be public.
3. return type should not be void.
4. compulsorily it should not have any argument.

**Note :-** if the property is of type Boolean then for getter method we can prefix with either “is/get".

* **Constructor**:- Main objective of the constructor is nothing but initialisation of object.
* **Note :- 1.** Constructor is a special type of method whose name is same as class name.

2. Every java has a constructor.

3. A constructor is automatically called at time of object creation.

4. A constructor never contain any return type including void.

5. The modifiers applicable for constructors are private, public protected, default.

* **Default Constructor** :- 1. For every java class constructor concept is applicable.

2. If we don’t write any constructor, then the compiler will generate a default constructor.

3.If we write at least one constructor then the compiler won’t generate any default constructor, so we say every java class will have a compiler generated default constructor or programmes written constructor but not both simultaneously.

4. Default constructor contains one line, super (). It is a call to super class constructor.

* **Super () vs this ():- 1.** The first line inside the constructor can be super ()/ this ().

2. If we are not writing anything then compiler will generate super();

* Super():- 1.It should be the first line in the constructor;



2. It should be used only in constructor;



3. It will take control to the parent class constructor.

* this () :- 1. It should be the first line in the constructor;

2. It should be used only in constructor;

3.It will make the call of the current class constructor.

* **Difference b/w super(),this():-**

1. These are constructor call.



1. These are used to invoke super call and current call constructor

directly.

1. We should use only inside the constructor that to first line otherwise we get compile time error.



* **Inheritance :- 1.**It always speaks about reusability.

**2**.Using inheritance productivity of the code can be improved.

**3**. If we use inheritance, lines of code can be reduced in the application.

**4.** In java inheritance is achieved through the "extends" keyword.

* Types of inheritance in Java :-

1. Single-level inheritance
2. Multi-level inheritance
3. Hierarchical inheritance
4. Multiple inheritance
5. Hybrid inheritance

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Access Modifier | With same class | Outside class same Package | Outside Package subclass | Outside Package non-subclass |
| Public |  |  |  |  |
| Protected |  |  |  |  |
| Default |  |  |  |  |
| Private |  |  |  |  |

* **Polymorphism** :- If one thing exists in more than one form then it is called Polymorphism. Polymorphism is a Greek word, where Poly means many and morphism means structures or forms.

1. Static Polymorphism:- If polymorphism exists at compilation time then it is called Static Polymorphism. Ex:- Overloading.

2. Dynamic Polymorphism :- If the polymorphism exists at runtime then that polymorphism is called Dynamic Polymorphism. Ex:- Overriding

Runtime vs Compile time polymorphism

* What are the differences between method overloading and method overriding?

1. Method overloading:-

● The process of extending the existing method functionality with new functionality is called Method Overloading.

● In the case of method overloading, different method signatures must be

provided to the methods

1. method overriding:-

● The process of replacing existing method functionality with new functionality is called Method Overriding.

● In the case of method overriding, the same method prototypes must be provided to the methods.

● With or without inheritance we can perform method overloading with inheritance only we can perform Method overriding.

* **Abstract keyword and Abstraction:**

1. The abstract keyword is used to achieve abstraction in Java. It is a non- access modifier which is used to create abstract class and method.
2. The role of an abstract class is to contain abstract methods. However, it may also contain non-abstract methods.
3. The method which is declared with an abstract keyword and doesn't have any implementation is known as an abstract method.{Read some note book}

* *Abstract Class and Abstract Methods :-*

1. In Java applications, if we declare any abstract class with abstract methods,

then it is convention to implement all the abstract methods by taking sub classes.

1. To access the abstract class members, we have to create an object for the subclass and we have to create a reference variable either for abstract class or for the subclass.
2. If we create reference variables for abstract class then we are able to access only abstract class members, we are unable to access subclass own members.
3. If we declare a reference variable for subclass then we are able to access both abstract class members and subclass members.

* **Final class :-** If a class is marked as final, then the class won’t participate in inheritance ,if we try to do so then it would result in “Compile Time Error”.

Ex: String, String Buffer, Integer ,Float,......

* Final variable:- If a variable is marked as final, then those variables are treated as compile time constants and we should not change the value of those variables. If we try to change the value of those variables then it would result in “Compile Time Error”.
* Final method:- If a method is declared as final then those methods we can’t override, if we try to do so it would result in “Compile Time Error”.
* **Interfaces:-1.**Interface is a Java Feature, it will allow only abstract methods.

**2.** In Java applications, for interfaces, we are able to create only reference variables, we are unable to create objects.

**3.** In the case of interfaces, by default, all the variables are public static final. **4**. In the case of interfaces, by default, all the methods are public and abstract.

**5.**In Java applications, constructors are possible in classes and abstract classes but constructors are not possible in interfaces.

**6.** Interfaces will provide more shareability in Java applications when compared with classes and abstract classes.

* **Important key points of Interface:-**

1. Whenever we are implementing an interface compulsory for every method of that interface we should provide implementation otherwise we have to declare class as abstract in that case child class is responsible to provide

implementation for remaining methods.

1. Whenever we are implementing an interface method, it should be declared as public, otherwise we will get compile time error.
2. In Java applications, it is not possible to extend more than one class to a

single class but it is possible to extend more than one interface to a single

interface.

1. In Java applications, it is possible to implement more than one interface into a single implementation class.

* **Abstract class VS Interface :-**
* Abstract class ;-

1. If we are talking about implementation but not completely (partial

implementation) then we should go for abstract class.

2. Every method present inside abstract class need not be public and abstract.

3. There are no restrictions on abstract class method modifiers.

4. Every abstract class variable need not be a public static final.

5. Inside the abstract class we can take constructor.

6. Because abstract class is a class that means we cannot achieve multiple inheritance .

* Interface :-

1. If we don't' know anything about implementation just we have requirement specification then we should go for an interface.
2. Every method present inside the interface is always public and abstract

whether we are declaring or not.

1. We can't declare interface methods with the modifiers private, protected, final, static, synchronized, native, strict.
2. Every interface variable is always a public static final whether we are

declaring or not following modifiers. private, protected, transient, volatile.

1. Inside the interface we can't take constructor.

* **JAVA8 Features over Interfaces. :-**

1. Default Methods in Interfaces

2. Static Methods in Interfaces

3. Functional Interfaces

* Default Methods in interfaces:--

1. In general, if we declare abstract method in an interface then we have to implement all that interface method in more no. of classes with variable implementation part.

2. In the above context, if we require any method implementation common to every implementation class with fixed implementation then we have to implement that method in the interface a3 default method.

3.to declare default method in interfaces we have to use the "default" keyword in method syntax like access modifier.

**NOTE:** It is possible to provide more than one default method within a single interface.

* **Static methods in Interfaces:-**

1. Up to JAVA7 version, static methods are not possible in interfaces but from JAVA 8 version static methods are possible in interfaces in order to improve shareability.
2. If we declare static methods in the interfaces then it is not required to declare any implementation class to access that static method, we can use directly interface name to access static method.

NOTE: If we declare static methods in an interface then they will not be available to the respective implementation classes, we have to access static methods by using only interface names not even by using interface reference variable.

Example :-

Interface I {

static void m1(){

System. out. Print ln ("m1-1"); } }

class Test {

public static void main(String args [] ){

I.m1(); } }

* **Functional Interface:-**

1. If any Java interface allows only one abstract method then it is called a "Functional Interface"
2. To make any interface as Functional Interface then we have to use the following annotation just above of the interface. @FcnctionalInterface.

Example :-

java.lang.Runnable

java.lang.Comparable

NOTE: In Functional Interfaces we have to provide only one abstract method but we can provide any no. of default methods and any no. of static methods.

* **What is Lambda Expression (λ):-**
  1. Lambda Expression is just as anonymous(nameless) function That means the function which doesn’t have the Same, return type and access modifiers

1. Lambda Expression also known as anonymous functions or closures.

* Use of Lambda Expression:-

1. To provide the implementation of Functional interface.
2. Less coding.

* Lambda Expression Syntax: -

(Parameter-list) -> {body}

Java lambda expression is consisted of three components.

1. Argument-list: It can be empty or non-empty as well.

2. Arrow-token: It is used to link arguments-list and body of expression.

3. Body: It contains expressions and statements for lambda expression.

1. No Parameter Syntax

() -> {

//Body of no parameter lambda }

1. One Parameter Syntax

p1 -> {

//Body of single parameter lambda

}

1. Two Parameter Syntax

(p1,p2) -> {

//Body of multiple parameter lambda

}

* **Exception**: - An unwanted/expected event that disturbs the normal flow of execution of a program is called "Exception handling".

The main objective of Exception handling is to handle the exception.

{more about exception in videos and note book. like checked/unchecked exception}

* **Difference Between Exception and Error:-**
* Exceptions: Exceptions are events that occur during the execution of a program that disrupt the normal flow of the program. They represent exceptional conditions that can be handled and recovered from within the program. Exceptions are typically caused by errors in the program logic, input data, or external factors. Examples of exceptions include NullPointerException, ArrayIndexOutOfBoundsException, FileNotFoundException, etc. Exceptions are further divided into two types: checked exceptions and unchecked exceptions.  
    
  Checked Exceptions: These are exceptions that the Java compiler requires you to handle explicitly using a try-catch block or declare in the method signature using the throws keyword. Examples include IOException, SQLException, etc.  
    
  Unchecked Exceptions: These are exceptions that are not checked at compile-time, meaning the compiler does not force you to handle or declare them. They are typically runtime errors that occur due to programming errors or unexpected conditions. Examples include NullPointerException, ArrayIndexOutOfBoundsException, ArithmeticException, etc.
* Errors: Errors, on the other hand, represent serious problems that are typically beyond the control of the program. They are usually caused by external factors or issues with the environment in which the program is running. Errors are not meant to be caught or handled by the program because they generally indicate a critical condition from which it is unlikely or impossible to recover. Examples of errors include OutOfMemoryError, StackOverflowError, VirtualMachineError, etc. Errors are typically unchecked exceptions
* **Multi-Threading: -**
* Operating System: - It is a system software which runs in the background and helps the user to run other applications.
* Multitasking: - Executing several tasks simultaneously is the concept of multitasking.

There are 2 types of Multitasking.

a). Process based multitasking

b). Thread based multitasking.

* Process based multiTasking: - Executing several tasks simultaneously where each task is a separate independent process such type of multitasking is called "process-based multitasking".
* Thread based multiTasking: - Executing several tasks simultaneously where each task is a separate independent part of the same Program, is called "Thread based MultiTasking".

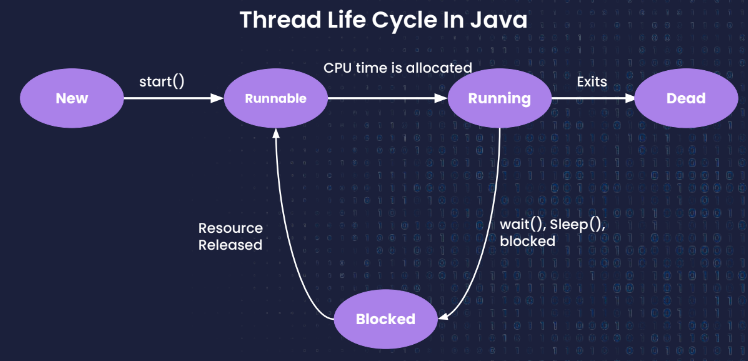
Each independent part is called a "Thread".

* What is Thread? - A Thread is a very light-weighted process, or we can say the smallest part of the process that allows a program to operate more efficiently by running multiple tasks

simultaneously.

* Thread Scheduler: - If multiple threads are waiting to execute, then which thread will execute 1st is decided by Thread Scheduler which is part of JVM.
* In the case of MultiThreading we can't predict the exact output(order of execution), only possible output we can expect.
* Diff b/w start () and run () :-
* If we call start() and separate thread will be created which is responsible to execute the run() method.
* If we call run(), no separate thread will be created, rather the method will be called just like a normal method by main thread.
* **Life cycle of a Thread:-**

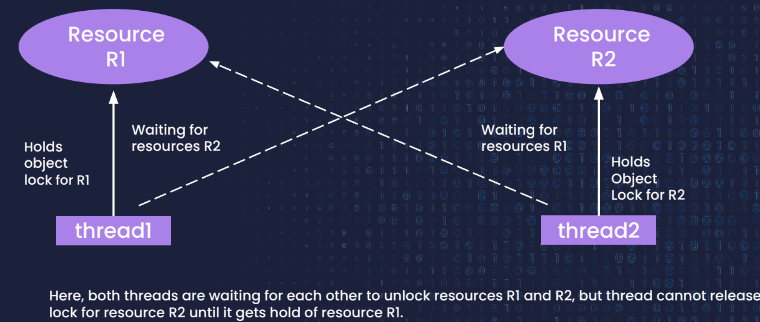
1. if Thread scheduler allocates CPU time, then we say thread entered into Running state.
2. if run () is completed by thread then we say thread entered into dead state
3. Once we create a Thread object then the Thread is said to be in a new state or born state.
4. Once we call start () method then the Thread will be entered into Ready or Runnable state
5. If Thread Scheduler allocates CPU, then the Thread will be entered into running state.
6. Once the run () method completes then the Thread will enter into dead state.

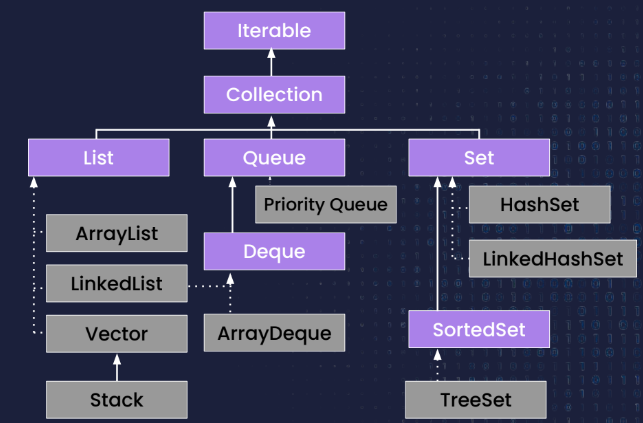


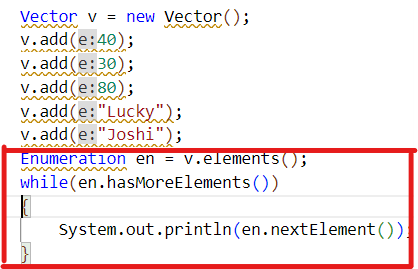
* **Synchronized keyword:-**

1. synchronized keyword is used to make the class or method thread-safe which mean only one thread can have lock of synchronized method and use it, other threads have to wait till the lock releases and anyone of them acquire that lock.
2. It is important to use if our program is running in multi-threaded environment where two or more threads execute simultaneously. But sometimes it also causes a problem which is called Deadlock.
3. The main advantage of synchronized keywords is we can resolve data inconsistency problems.
4. But the main disadvantage of synchronized keyword is it increases waiting time of the Thread and effects performance of the system.

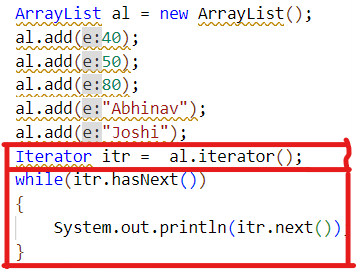
* **Deadlock: -**

1. Deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.
2. If 2 Threads are waiting for each other forever (without end) such type of situation (infinite waiting) is called dead lock.
3. There are no resolution techniques for deadlock but several prevention (avoidance) techniques are possible. Synchronized keyword are the cause for deadlock hence whenever we are using synchronized keyword, we have to take special care.
4. Long waiting for a thread where waiting ends at a certain point is called **"starvation".**

* **Collection:-** In Order to represent a group of individual object as a single entity then we need to go for Collection.
* Collection Framework :-Group of classes and interface, which can be used to represent a group of individual object as a single entity, then we need to go for "Collection Framework".
* **Collection Hierarchy:-**



Enumeration: -



Iterator: -