Java

07 May 2022 13:20

James Goslin wanted a programming language Object Oriented programming language with well known syntax supported by compile time type chacking

What is reason behind to invent Java

1. **Dynamic Object model** = Means object oriented programming language having classes so by we can provide class at run time when it is required.
2. **Portable binary distribution** Java program are machine portable, its mean we can execute java program on anywhere.
3. **Safe execution Environment** - its mean **jvm** can predict all possible output of code

Java program based on object oriented programming because oop is having well known syntax and it provide compile time checking

Java: it is not just programming language, It is a software platform for building (using JDK) cross-platform(it can work multi platform) application and for safely executing (using JRE) binaries of such applications on an operating system for which it is supported. The implementation of Java platform includes support for

1. **Java Virtual Machine (JVM**) -java have class file, Each Java class file contains the binary representation of a single Java type which includes the meta-data (**compiler readable declaration means c++ there two different one is declaration means .h file and 2ns is definition .cpp file so .h file is meta-data file**) of that type along with bytecodes (machine neutral instructions) for its implemented method. The JVM manages execution of a Java application on a platform by .

(a) Loading class files into the memory when they are required by the executing application
(b) Translating bytecodes into their equivalent native machine instructions with safety checks at the time of their execution.

2. **Java Runtime Library -** It is a set of packages containing class files with types which enable a Java application to consume services offered by the following in a portable manner: .
   (a) Runtime which includes support for data-types, reflection and native-methods.
   (b) Platform which includes support for multi-threading, file i/o and network sockets.
3. **Java Programming Language -** It is a high-level programming language designed specifically for coding applications which can be executed by the JVM. It has following characteristics .
   (a) It offers C++ like but more consistent syntax based on a type system with eight primitive value types they always get pass by value and support for implementing user-defined, they get **pass by reference types.**
   (b) It is **primarily object oriented based** (c++ primarily procedure based)on common root single class inheritance model with added support for generic programming and functional programming.

- System.out.println("Hello World");
  Here there is class system and println is method inside printstream class. out is object of printstream class which declared as static variable inside System class so we can call static variable directly on class name so we called out variable as System.out (java.lang.System) but println is not static so we called it by using System.out.println
- java.lang.Object
  java.io.OutputStream
  java.io.FilterOutputStream
  java.io.PrintStream.

- Command to run java code
  -javac file.java (to compile and convert to byte code as .class file
  -javap file.class (to see class file declaration)
  -javap -c file.class(to see byte code)

JAVA PRIMITIVE TYPE

DATA Type	Allowed Value	Wrapper Class	Format
boolean	True|False	Boolean	%b
char	Single character(Unicode)	Character	%c
byte	8-bit signed integer	Byte	%d
short	16-bit signed integer	Short	%d

int	32-bit signed integer	Integer	%d
long	64--bit signed integer	Long	%d
float	32-bit signed precision real	Float	%f
double	64-bit signed precision real	Double	%f

double p = Double.parseDouble(args[0]);
Whatever string value inside args[0] convert into double value
OPEN CLOSE Principle says that make your fields private,if our any member is public we cant change public declaration. And class allowed us to add member so we can do method overloading.

- When new object created , a parameter-less constructor is implicitly defined for a class if it does not explicitly define any constructor .

Instance Method-: First, we create object and then we called a method to just initialise fields of class by using object.method called instance method.

- By using **new** operator we create object on **heap** which provide reference of object on stack.
- There is no need to provide include file, compiler provide meta-data of every declaration, So according to meta-data compiler search class.

Why java does not support default constructor.?
Because java create object on heap and default constructor create on stack and also java does not support default argument.

- In java class is not just class it is binary representation of class so every user defined(like enum)make own class file.

Java does not have mangled name concept, java has descriptor every member has own descriptor, to see descriptor put **javap -s -p classname.**

Variadic Function- A function required n number of argument and variadic method can accept variable number of arguments through its last (var-arg)
e.g. LCD, AVERAGE,GCD, SUM, etc..
Private int sum(int first, int second, int… num)
For each Style loop
For(double value : num)
Packages:- In C++ concept of namespace, but in java there is a concept package which is used for **Class** Hiding(not gives to access outside and **name collision.**
A type class **T** belonging package **P** has following characteristics:
It is accesible outside of p through its fully qualified name of **P.T** only if its defined with public modifier(in T.java**).**
- Its binary representation is loaded by default path **P/T.class.**

Dependency Inversion principle says that if your class derived from abstract, these classed not visible to outside of this class

RULE:- if we make class public, we must provide class name and file name same.
Single responsibility principle don't allow to add any member inside class so Liscov Substitution say that if you want to add any member in class so make sub class of it and add there.

Supper-:by using this keyword we can call super class constructor and any member of super class.

- **Overriding method** by defining a method whos name, parameter types and return types match with those of a method defined in the super class

IMPORT-: IMPORTING a type name from another package enables the compiler to expand unqualified name to its fully qualified name.
UPCASTING always done implicitly.
Instanceof Operator-: It will check class instance(like **emp instanceof Employee** and give true or false.

- Inheritance has "**is a"** relation.
- **Immutable Object,** If we create one object, once we initialize it we cant change fields inside class.
- **Java.lang.Object**(which is root type for all reference type) Class in java is always **super class** of every class, we don't need to defined **extends word** explicitly.

- **OBJECT IDENTITY** Two object are consider to be identical, if they refer same instance in the memory. Object **x** and **y** are identical is determined by **expression x ==y.**
- **Object Equality** Two object are consider to be equal if they refer instance of same class with matching state in memory. By using **expression x.hascode() == y.hasecode() && x.equals(y**) we can determined **object x equals to object y.** If two object are identical they have to be Equal.

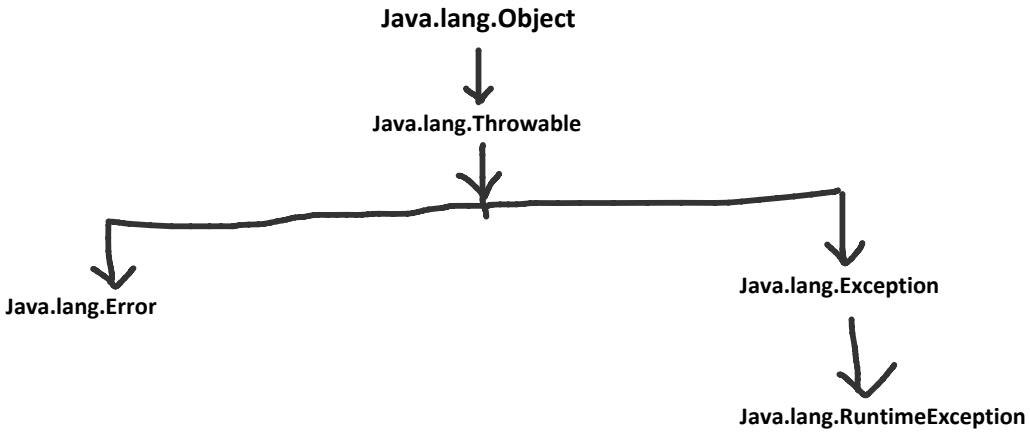
Down casting not to be done implicitly, so by using expression **CLASS obj =(class)other** we do explicitly

- ❖ If we make class as public its mean this class can accessible outside this package.
- ❖ If we make class as non public its mean this class can not accessible outside this package.

Visibility of a member outside of its declaring class

Access Modifier	Current Package	External package
Private	None	None
<default>	All	None
Protected	All	Sub-class
Public	All	All

Exception Handling -:



- In C++ we can throw any thing but in java it is not possible, in java we throw exception when class extends from java.lang.Object.
- **Checked** if we derived class from **Java.lang.Exception.**
- unChecked if we derived class from **Java.lang.RuntimeException.**
- **C**hecked exception can only occur within a try block which catches this exception OR in a method which declares it to be thrown.
- Unchecked exception occur anywhere.
- We found Checked expectation at compile time and unchecked exception at run time
- **Throws Throw.**

ABSTRACTION CLASS TEST-:

- A class defined with abstract modifier can not be instantiated, it can only be used as a super-class for other classes.
- A method defined with abstract modifier has no implementation and it must be overridden in a non-abstract subclass of this class and such a method cannot be defined in a non-abstract class.
- A method defined with final modifier cannot be overridden in a subclass of this class and as such JVM can skip dynamic binding(**Dynamic binding is so expensive**) for handling its call, means when we put method as final so jvm will knows that this method don't have any implementation in sub class. It directly execute this method

from super class.

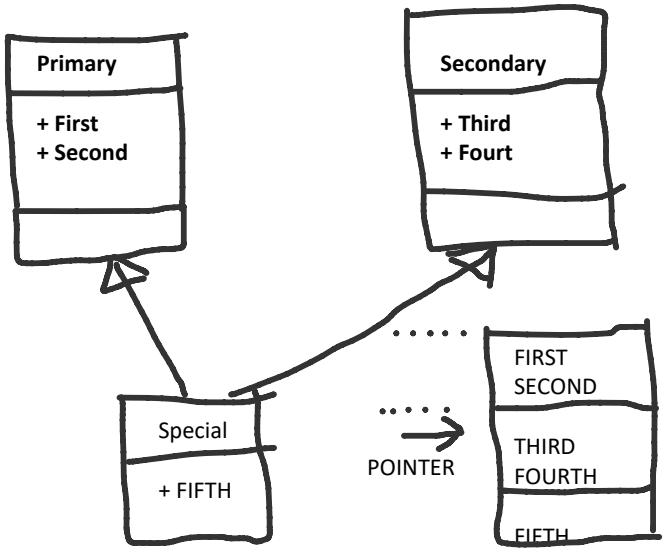
- A class defined with final modifier cannot be used as a super-class
- A field declared with final modifier cannot be reinitialized.
- In java when we override method we can increase accessibility of method(in super class method protected but we make it in sub class public but not private).
- If we don't make class public so there is no need to put constructor as public.
- If we make non-abstract class as non-public so we cant create its instance from other package .so java provide us factory class(23 design pattern to deal such situation) to make instance of non-public class.
- A class that only contains static members does not require new objects or subclasses and as such this class should define a private constructor.

Static initializer-: To initialize Static variable we **used static{ static variable=smoething;} ,** static initializer execute once after the class is loaded into the memory.

- Unix starts with 1 jan 1970 so System.currentTimeMillis()-> count seconds up to 1970 to current time.

Why java does not allow to inherit from multiple class.?

When we create object of special class, Primary class put its field at 1st position at first reference and secondary put its field at first reference but this is not possible. But in C++ there is pointer concept, when primary search its field pointer change its location and point primary's fields and when secondary search pointer change its position to secondary's fields. And Java has references and references provide current location.java can support pointer or safety.



- Suppose if secondary does not have fields, So we can provide multiple inheritance .
- But no one promise that they will not provide fields inside it .
- So overcome this condition java **introduced INTERFACE.**
- By using **Interface** java provide multiple Class inheritance as well as **Safety.**

Multiple Inheritance in Java: JVM's type system does not allow a class to inherit from multiple other classes because an instance of such a class will require multiple sub-objects (to store values of instance fields defined by corresponding super class) which complicates its runtime type access required for safe-casting and reflection. Since interface cannot define instance fields it does require a sub-object within an instance of its inheriting class and as such Java allows a class to inherit from multiple interfaces.

There are no language introduced, which provide all three properties in one language. Properties are following

1. **Strict Typing(upcasting, Downcasting,etc) - c++, JAVA**
2. **Multiple Class Inheritance -Python c++**
3. **Safe Execution -Python JAVA**

Abstract Class	**Interface**
It is a reference type which does not support instantiation but can define instance fields	It is a reference type which does not support instantiation and cannot define instance fields
It can define unimplemented instance methods using abstract modifier.	Its instance methods are implicitly abstract and implemented methods must be defined with default modifier
It can include public as well as non-public members	It can only include public members
It can define constructor which is called by its sub-classes	It cannot define a constructor
It can define final as well as non□final static fields	It can only define final static fields

It can extend exactly one other class	It can extend multiple other interfaces
A non-abstract class can inherit from a single abstract class and it must override all the abstract methods of that class	A non-abstract class can inherit from multiple interfaces and it must implement all of their abstract methods
Generally defined for segregating common state associated behavior supported by different types of objects	Generally defined for segregating common state independent behavior supported by different types of objects

- **Interface support** only Static fields, which is implicitly declare as **public static final**.
- In interface method always implicitly declare as **public abstract** .
- By using **default modifier** we can define method inside interface.

Every class, we write first extend and then we write implements. If class not inherit from abstract class then we can write implement.

We can do casting by expression :-

```
if(acc instanceof Profitable)
Profitable p = (Profitable)acc;

OR  if(acc instanceof Profitable p)
```

- **nested member class** is defined inside of another class with static modifier.
- **Inner Member Class** is defined inside of another class without static modifier.
- **inner local anonymous class** A class inside a method of a class is known as **inner local class** and a class without called **anonymous class**. A class without name inside method of class is **inner local anonymous class**

RESOURCES -: To open resources(which is controlled by OS) in java, by using constructor we can open but to close resources we have to explicitly call to **close()** function there is no concept of destructor like c++ which is call implicitly to close() function.

- Sometime what happened we write close() function but it will not called due to some **Exception** and it will go directly in try-catch block at main().
- But any condition our close() should called, so we write **try with finally block.**
- This finally block always call close() function.
- **finally block** always executes before control leaves try block.
- If we **implements** our class from **interface AutoCloseable**, means we have close method.
- There is a **try block with resources** which we used when our class is implements from **interface AutoCloseable** which allow as to close resources by expression below

  ```
  try(var a = new Auditor()){
  }
  ```
  compiler will automatially attach a finally block with a.close()
- If we define **X** class inside program class after compile JVM generate a class file name **as program$X.class .**
- **B**y using **string1.equals("rahul")** we can check that passed argument (as "rahul") is same as in **string1** .

Generic programming-:

- Object covert to any thing is done by down casting like **string ss =(sting)select(3,"ram", "shyam");.**
- **Object** is root type.

AutoBoxing - All primitive type(char, int, double,etc..) converts into its wrapper class(java.lang.Character, .Integer, .Double, etc.) object automatically and treated as reference type called as autoboxing.

OR

implicit conversion of a primitive type (double) value to an instance of its wrapper (java.lang.Double) is called auto-boxing. Auto boxing so expensive because it create instance of Double class and take double as parameter so it take more memory and also take time to execute, so it is expensive

In generic programming when we passed **String x = (String)select(s, "Sunday", 9.72);** but other end it will accept as **Object argument{**private static Object select(int index, Object first, Object second)}, if s is even this function return string, which is object type so we cast it into string but for double value it will loose **TYPE SAFETY** so compiler will not give error but at run time jvm give error.

So we want general purpose code without loosing type seafty so we used **GENERIC PROGRAMMING.**

- **generic method with type parameter T{** private static <T> T select(int index, T first, T second)**}** where T is not class it is treated as Object, So only member of java.lang.Object can be applied to it.

  For example

```
private static <T> T select(T first, T second)  {
        if(first.compareTo(second) > 0)
            return first;
        return second;
    }
```

  Here **compareTo** is not member of java.lang.Object, So compiler will not allowed and gives error.

  But

```
private static <T> T select(T first, T second) {
        if(first.hashCode() - second.hashCode() >0)
            return first;
        return second;
    }
```

  Here **hashCode()** is member of java.lang.Object and it will run.

  This above technique is called **ERASER**.

- If we want to use Others member (like compareTo) so we have to write like

```
private static <T extends Comparable<T>> T select(T first, T second) {
        if(first.compareTo(second) > 0)
            return first;
        return second;
    }
```

  Now its allow members of Comparable and Object class. But all Object which are passed by user must inherit from comparable like we passing object of interval but this interval not inherit from Comparable, So compiler will not allow and gives error

- If we want **pass interval** object and **use to compareTo** method so this interval class **must implements from Comparable** interface and we have to **override compare** to method inside interval class.
- Here <T>showing method returning Type T.
- Generic programing is design for **reference type** only.
- We can **not write** inside of **<double>** because in<> always required reference type so we will put **wrapper type<Double>.**

Heterogenous Tag<Object> a : It can have different-different data type, This Tag is not type safe.

Homogenous Tag<Double>,<Stirng>,<etc> a : same data type

Wild-Card Substitution:

When we pass argument as object type to generic method which accept argument as object type so it will work And in simple stack of object we can push any type of data.

```
private static void printStack(SimpleStack<object> store) {
        while(!store.empty())
            System.out.println(store.pop());
    }
```

When we pass argument as <String> or <Double> **type to generic method** which accept argument as object type so it will work because in simplestack of string we can not push any data type we can only push String type

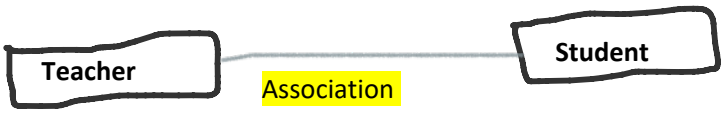
```
private static void printStack(SimpleStack<String> store) {
        while(!store.empty())
            System.out.println(store.pop());
    }
```

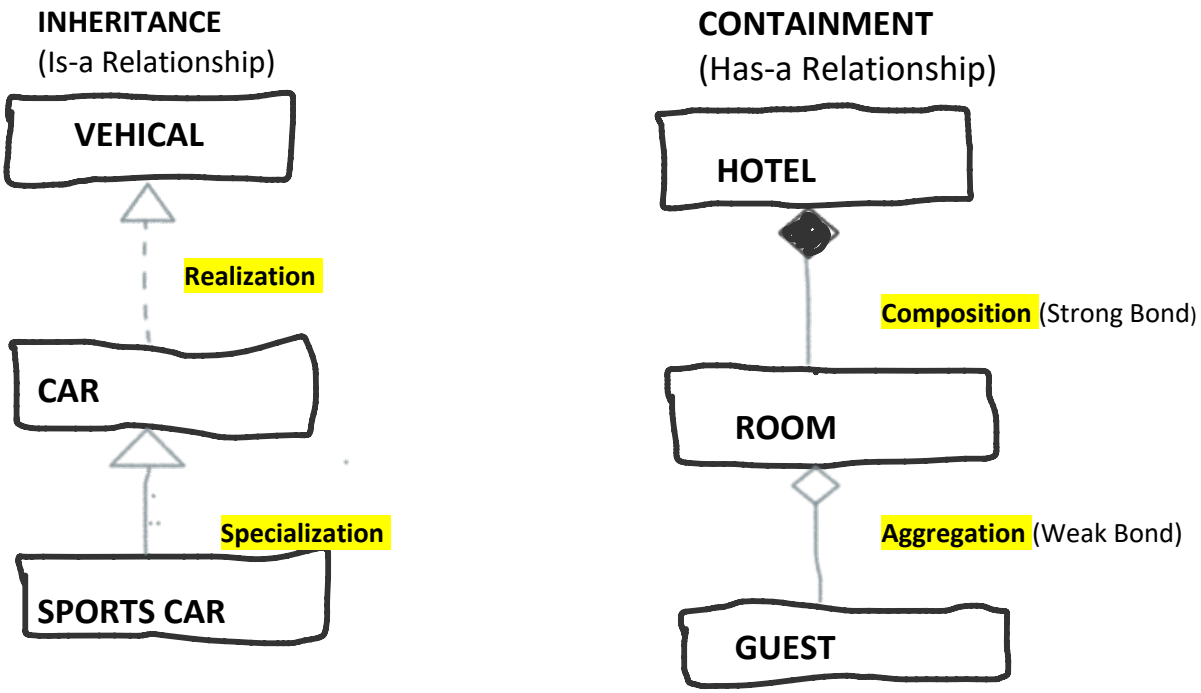
A generic type G is invariant over its type-parameter T which means G cannot be substituted by G irrespective of relationship between U and V. However following variant forms of G can be used in a declaration as

- **G<? extends U> :-** Here this <? extend U> allow to output only not input where G<V> implicitly convert into type <U>.

```
private static void printStack(SimpleStack<? extends Object> store)
            OR
    private static void printStack(SimpleStack<?> store)
```

<u>Inheritance</u> :-





Generic Collection :- It is Object of generic class which supports grouping of elements and provide facility to access those elements in type safe manner. The java.util package includes Collection interface which extends java.lang.Iterable interface to define standard methods for adding/removing elements to/from a generic collection. This interface has following sub-interfaces in the same package

1. **List<E> -** It is used to provide **Sequential Collection** whose elements can be retrieved through their indexes.It is implemented by ArrayList [1 ref/entry and O(1) indexing] and LinkedList [3 ref/entry and O(n) indexing] classes of java.util package . In ArrayList O(1) means it take time for every element **1 MS to add** but in LinkiedList O(n) means it take time for 1st element to add is **1MS** but it will take time for 16th element as 16MS . So LinkedList is not useful more.
   List method does have **add & get** method .
2. **Set<E> -** This type of collection implements which not contain any duplicate value inside it. It have some unique method in side it which shows its unique behaviour.It is implemented by HashSet [1 ref/entry and O(n) operations] and TreeSet [3 ref/entry and O(log n) operations] classes of java.util package
   in HashSet O(n) means it take time for 1st element to add is **1MS** but it will take time for 16th element as 16MS. But in TreeSet O(log n) it will take time to add 16th element as **4MS(log16 =4 log2 and log2=1)** and tree set are in sorted order, so easy to search
   In **Set** {a, a} ={a} and {a, b}={b, a} This shows, there is no indexing due to set method does not have **get** method. Hashcode and equals method always be there but treeset required compareto method which is implements through comparable inteface.

A Set is commonly implemented to contain an implementation of Map interface which groups pairs of unique keys and their associated values. This interface is implemented by HashMap [2 ref/entry and O(n) operations] and TreeMap [4 ref/entry and O(log n) operation] classes of java.util package.
HasMap implements from Map interface so it does not have add method, it have put method. If key is duplicate then it update value at last inserting key value.

RUNTIME

- **Callback Function**-> It is a mechanism of Passing a function of a specific type(Like return type and argument type) to a function, in order to allow this function to indirectly call that function. Java does **not support pointer** because pointer are **not type safe**. Java provides synthetical support for implementing callbacks through.
  1.
  2. **Functional Interface** - It is an interface containing **exactly one** abstract method defined to be used as a type for a method reference which is compatible with that abstract method. It is **automatically implemented** at runtime to invoke the method referred by its identifier.
     Immutable Object-> A Object doesn't have any **set** method, who's state can not be change. .
- **Functional Programing**(Declarative Type)->

  - **Record** is reference type, which instantiation produce a value like immutable object. Compiler convert a record type into a class type, containing private final instance fields, a parametrized constructor, getter method of fields and override toString, hashCode and equals method.
  - **Arrays.stream(object**) convert object type into stram.
  - **.filter(lambda function)** use to provide filtration.
  - **.map** convert type object into stream object.
  - **.foreach** it take one argument and return void

Runtime

25 June 2022 01:56

Reflection -> It is a mechanism to **examine(INTROSPECT)** the structure(**Like what is name of class, no of fields, name and type of fields**) of its **own class Objec**t at **runtime** called as ***Reflection***. Java is first language, provides support for Reflection.

In java the **meta-data** of type **T** is **exposed** by JVM through an instance of ***java.lang.Class<T>***. And ***instance*** of *java.lang.Class<T> referred by expression **T.class.***

E.g. *Interval i = new Interval(3,45);*

I is Instasce of Interval class

*Classes is store in memory as Object of **java.lang.Class<Obj>***

Interval() = new java.lang.Class<Inteval>

Interval() is instance of java.lang.Class<Inteval>

Representational form(like Json, Xml) are use to represent data and transfer the data, we cant use object to transfer data because, object has required environment to obtain data.

1. When we have object as **Obj** of classs

➢ To get **class nam**e we used

```
Class<?> c = Obj.getClass();
c.getName()
```

➢ To get **fields details**

```
var f : c.getDeclaredFields()
```

- To know name of fields-> **f.getName()**
- To make private fields accessecible-> **f.setAccessible(true)**
- To get value of fields -> **f.get(Obj)**

2. The fully-qualified name N of a class or an interface

Class c = Class.forName(N); //if class not found ->**classnotFoundException**

- To create instance of class at runtime known as **Dynamic Activation**,

```
Object policy = c.getConstructor().newInstance(); //here parameter less constructor will call
Object policy = c.getConstructor(double.class).newInstance(2.5); //here parametrized constructor will call who's taking one argument as double type. if method not found ->methodNotFoundException
```

- To get method at runtime which taking 3 actual arguments, first argument always be this type . double and int type are other parameter

```
Method scheme = c.getMethod(args[2], double.class, int.class);// args[2]-> MATCHING method name
```

- To call this method

```
float rate = (float)scheme.invoke(policy, p, n); //late binding
```

The class will be loaded by the built-in class loader of JVM which will throw java.lang.ClassNotFoundException if it cannot find the class file corresponding to name N. The e built-in class loader loads the binary representation of type p.T from path p/T.class. java.class.path property indicate this path by default to the **current directory** and always includes the **Java runtime library.** And we can set our class path by using **java -Djava.class.path=../ReflectionTest1 Program OR export CLASSPATH=.:lib**

STATIC BINDING-> If we know the method is declare in employee class with **final** modifier and we call it so this binding known as Static Binding.

DYNAMIC BINDING-> If we know the method is **not declare** with **final** modifier in employee class and we call it so this method called from employee class or its sub class but at runtime specific from specific class which method we know at compile time so this binding is known as Dynamic Binding.

LATE BINDING-> If we know full qualified name of class and we get the class at Runtime and we will create

instance of that class at runtime and we call the method, comes at runtime of same class, known as Late Binding.

Annotation

User defined modifier called as Annotation. It is defined as an interface along with following retention policies.

- **SOURCE -** The annotation appears in the source code of its declaration target but it is discarded by the compiler(like comments, @Override). Such an annotation can only be examined at compile time using its annotation provider.
- **CLASS -** The annotation is inserted by the compiler into the binary representation of its declaration target within the class file but is discarded by the runtime. Such an annotation can only be examined at compile-time using its annotation provider.
- **RUNTIME -** The annotation is loaded into the memory by the **class-loader** along with the meta-data of its declaration target. Such an annotation can be **examined at runtime** using reflection.

- Interface notation defined by **@interface interface_name**. Inside we can declare method having default value , this method can not allowed parameter but it can return.

```
E.g
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface Annotation_name {
    int value() default 5;   // value is special member of annotation interface
}
```

- By default we can apply annotation anywhere on class or on class member.

```
@Annotation_name(value=5) or (5)                    @Annotation_name
    public float common(double amount, int period)   OR   public class EducationLoan {}
```

- Provide restriction to apply annotation -> **@Target(ElementType.METHOD)**
- To provide Retention of annotation we used **RetentionPolicy.** If we set **RetentionPoilcy.SOURCE**, compiler will remove it at compile time. If we set **RetentionPoilcy.CLASS**, loader will not allow it at runtime time. If we set **RetentionPoilcy.RUNTIME**, loader will load it at runtime time.
- To apply annotation on program main on method file we used like

```
Annotation_name md = method.getAnnotation(Annotation_name.class);
    int m = md != null ? md.value() : 10;   // Here we can use int m as annotation
```

NATIVE METHOD

Java is portable language. We can use java language at any machine. And java is not machine specific so it is not powerful language, and this machine specific language are use to link with java we used **NATIVE-METHOD** to make java powerful.

1. **Purpose**
   (a) to increase the performance of a Java program by skipping the excessive verification done by the JVM.
   (b) to reuse non-Java code available on the platform including system services which are not exposed by the Java runtime library.
2. **Disadvantages**
   (a) It can compromise the execution safety ensured by the JVM to protect a program from crashing at runtime.
   (b) It can limit the portability of a program to different platforms supported by Java.

Step 1-> compile code with **javac -h . Program.java**

Open program.h

Focus on

JNIEXPORT jlong JNICALL Java_Program_gcd
(JNIEnv *, jclass, jlong, jlong);

Here we see Java_Program_gcd function has 4 parameter and one #include <jni.h> which is not part of C

This <jni.h> file not available in c library it is in java library. Compiler look into c library to change path we used->**export CPATH=~/jdk-16.0.1/include/:~/jdk-16.0.1/include/linux/**

- Make a C file, include "program.h" and put above method (Java_Program_gcd(JNIEnv* env, jclass cls, jlong first, jlong second){ }
- To compile above C file(PIC= position independent code) -> **cc -shared -fPIC -o libfile_name.so file_name.c**
- To load .so file in library -> System.loadLibrary("shared_file_name");
- To look share object in current directory ->java -Djava.library.path=. Program gcd 48 72 **or** export LD_LIBRARY_PATH= (. Used for current directory or we can give any path)
- In java processing of array is taking more time because to get element of array, it process whole array every time. take array of string and convert into double

```
Double[] values =  Array.stream(args)
                   .skip(1)  // to skip first argument in args[0]
                   .mapToDouble(Double::parseDouble)
                   .toArray();
```

- Inside java.h file java gives **jdoubleArray values** which is useless to **C file.**c array are in sequence but java arrays is not sequential. Sequence of array in java only JVM knows. So JNIEnv* env which deal with JVM.
- Here env -> [JNIEnv] -> (it contain a table of all ENV function)
  - To get length of array -> **jint length = (*env)->GetArrayLength(env,values)**;
  - To allocate memory -> jdouble* items =malloc(n * sizeof(jdouble));
  - To put elements from jdoubleArray value to jdouble items but (*env)->GetDoubleArrayRegion(env, values, 0, n, items);

    We telling here to env **copy** from Array values elements **to 0 upto nth** into items Array.
  - After performing task

Platform

27 June 2022 23:15

Thread-Test

Concurrency : It is programming support offered by the platform for simultaneously executing multiple blocks of code. It is used for

1. **Asynchrony -** In which user call the procedure, between execution of procedure and return of procedure , user can resume other execution in order to increase the responsiveness of the running program, this is known as asynchrony.
2. **Parallelism** - Long running iterations distribute on separate processors(or cores) available on the hardware in order to increase the performance of the running program.

Thread : It is the basic unit of concurrency within an executing program and it has following characteristics.

1. A thread refers a function which executes concurrently with other threads. Main threads started automatically when program execute. Main thread also execute concurrently with other threads.
2. A thread shares values of variables with other thread reference to the function not local function.
   System.currentTimeMillis() -> It returns time measured in milliseconds, between the current time and midnight, January 1, 1970
   1000 part -> mili sec
   1 million(10,00,000) part = one micro sec
   One Billion part is equal to 1000 million part = one neno sec

- Declare variable as **private static ThreadLocal<String > client = new ThreadLoca<>()**;
- We find **ID** of **current thread** in **Unix environment** By using **-> pthread_self()**
- We find **ID** of **current thread** in **JAVA environment** By using **-> Thread.currentThread().hashCode()** Here **currentThread()** return which thread is current thread and by overriding **hashCode()** method we get id of thread.
- In Java there are two **type** of thread **daemon** and **non-daemon**. By default all threads are non daemon thread. For daemon(Background) thread JVM is not going to wait to finished. We can make threads daemon by using **->child.setDaemon(true).**

SYNCHRONIZATION-TEST

- Wait for child to exit -> **child.join();**

synchronized block - By default When one thread enter into synchronized block it will lock the block by using monitor and it will unlock the block until the monitor will not exit from synchronized block.

When we do reading and writing data from code, we must defined synchronized. If we dont define then thread race can happened.

This.wait(); ->temporarily unlock the monitor of this object and wait for some other thread to notify the monitor of this object. In side wait we can put mili sec to wait thread if signal will not com and time complete than thread will start execution.

this.notify(); -> allow a thread waiting on the monitor of this object to resume execution.

Asynchrony-Test

- Create a ASYNCH version of method like

```
public long compute(int first, int last) {
    CODE
    }
```

ASYNCH version of above function is

```
public CompletableFuture<Long> computeAsync(int first, int last) {
        return CompletableFuture.supplyAsync(() -> compute(first, last));
    }
```

- A **comleteableFuture<>** contain a **thread pool.** Free thread(child thread, main thread) they do perform task and die But pool thread will not die.
  A pooled thread will invoke the supplied method allowing the caller thread to resume execution and acquire the result of invocation at a future time after the pooled thread has completed that invocation
- Here Free thread call the ASYNCH version of **compute function** where A pooled thread will invoke the supplied method with free(caller)thread .And make to Caller thread free to resume execution. After acquiring the result of invocation at a future time to

STREAM IO

File is a way to input-output of data. Transferring data bytes by bytes is known as **Streaming of Data.**

- **FileInputStram**->Open file for Read data if file is not exist gives exception
- **FileOutputStram** -> Create a new file and Write data if file exist truncate it and write data. In java O_EXEL not available which use to if file not available then only create new file.
- We will write both statement of FileInputStram and FileOutputStram in **try** block because open file **CLOSE** automatic.
- **int n = input.read(buffer, 0, BUF_SIZE);** ->input is file, start Reading buf_size data from 0 and put into buffer.
- **output.write(buffer, 0, n);** -> write **n bytes** from **buffer** to **output file** and start from **0 bytes.**
- **READ** and **WRITE** function done by operating system only not programming language.
- In java all byte operations return int e.g. int data = a(byte) **+**or-or*****or**/**or**^** b(byte).
  E.g. -> data[i] = **(byte)**(data[i] ^ '#'); Here we casting data into byte
- Stream io is not suitable for large size of file because we cant create big size(cant create size of buffer more then 64mb) of buffer. And also some operation like reverse of file not possible by stream io because it is read and write file **block by block** and to reverse to we required whole file to overcome it we used **MEMORY-MAP-IO**.

MEMORY-MAP-IO

Instead of making address space somewhere else, put the address of file into our address space so we can direct access it.

- **byte[] buffer** is always be **on heap** but **ByteBuffer** is address of file and it can **be anywhere**.
- **try(var channel = FileChannel.open(doc, StandardOpenOption.READ, StandardOpenOption.WRITE))** -> Describe path which we have to open and for what reason(like read write)
- **channel.lock(); ->** Lock the resources so any other program can not access it after use-> **lock.release();** so other can use it.
- **(int)channel.size(); ->** This size return as **long(64 bit valye)** so we have to cast in **int.** Java has restriction 2gb file we can mapped.
- **var buffer = channel.map(FileChannel.MapMode.READ_WRITE, 0, n);**-> mapped channel into buffer start from 0 to whole file.

Object Serialization : It is a mechanism of converting the entire state of an object (including the state of each object it references) into a stream (series) of bytes from which it can be deserialized (reconstructed). java.io.Serializable is **marker interface** there is no method inside it. It is commonly used for

1. **Persistence -** in which the object is transferred to a storage medium.
2. **Marshalling** - in which the object is transported from process to process, across process boundary.

Object Serialization in Java: The java.io package includes ObjectOutputStream/ObjectInputStream class to support serialization/deserialization of an object with following characteristic**s**

1. Its class must inherit from java.io.Serializable interface and should define static final long serialVersionUID field with a value representing the set of its non-transient instance fields.
2. It must not refer to any non-serializable object through a field which is declared without transient modifier by its class.

Transient- Object which we dont want to do serializable, we make it as **transient.**

- We cant store direct Object into fileOutputstream so we first convert Object into bytes by using **ObjectOutputStram** and by using FileOutputStram we convert bytes into stream ->**var output = new ObjectOutputStream(new FileOutputStream(doc)).**
- output.writeObject(info); -> object serialization writing object stream into file (writeObject(Object) and we passing site upcasting happening which can done implicitly)
- **var input = new ObjectInputStream(new FileInputStream(doc))** -> Here by using **FileInputStream(doc)** we convert byte data into stream and by using **ObjectInputStream** convert stream data into object.
- **(Site)input.readObject()**; ->object deserialization, reading object stream from file (returning object but we want site so down casting happening which done explicitly by using cast operator **(Site)**).
- **Java Platform Modularization System(JPMS)** provides support for grouping packages into a module to control their visibilities and for specifying their dependencies

```
module tourism.lib {
        exports tourism.api;
}
```

```
module tourism.app {
        requires tourism.lib;
}
```

- To compile both module
  javac -d dist/ --module-source-path src/ --module tourism.app
  javac -d dist/ --module-source-path src/ --module tourism.lib
- To run program file
  java -p dist -m tourism.app/tourism.ui.Program
- To run above command by using shell scripting
  Step-1 **vi run. Sh**
  Step-2 **in run.sh file put** ->java -p dist -m tourism.app/tourism.ui.Program $*
  Step-3 run by using command-> **bash run.sh**
  Step-4 dont wont to write bash put -> **chmod a+x run.sh**
  Step-5 run without bash -> **./run.sh**

Communication

29 June 2022 13:21

Files do to input output and Socket do to communication. **Socket** is common logical **interface** offered by the **platform** use to **transfer data** between separate **process** by **using** different **Network communication** schemes supported by the **operating System**. Each socket is bound to a unique **endpoint identifying** its communication type **specific address** and provides operations to send/receive data to/from other such **endpoints**.A **connection oriented socket** which supports streaming of data between endpoints is used for implementing Java supports to scheme 1. Standard protocol Unix Communication 2. INET(Internet Network communication scheme)

(A) Server - It provide service for processing or sharing data at well known endpoint address(Servers address well known by client) using following steps

1. Open a listener socket and bind it to a particular endpoint address.
2. Use the above listener socket to accept the socket connected to the endpoint which has requested this connection.
3. Use the above accepted socket to exchange data with its connected endpoint.
4. Close the above accepted socket and go to step 2.

(B) Client - It consumes the service provided by the server from a random endpoint address using following steps

1. Open a socket and connect it to the endpoint address of the server.
2. Use the above socket to exchange data with its connected endpoint
3. Close the above socket

Server Socket

- UnixDomainSocket based on file concept. So we have to provide **unique name.** And if there same file name available so we will delet file and create new one by using command-> **Files.deleteIfExists(Path.of(NAME));**
- **Step1**Open listener Socket by usnig ->
  var listener=ServerSocketChannel.open(StandardProtocolFamily.UNIX);
  ServerSocketChannel -> it allow us to listener socket.
  To binding with particular(**UnixDomain**) endpoint-> listener.bind(**UnixDomainSocketAddress.of(NAME)**);
- In java we make infinite loop -> **for(; ;)**
- **Step-2** Wait for connection request and accept it ->**var client = listener.accept();**
- **Step-3** Using this client, we can read(**client.read(buffer)**) and write(**client.write(buffer)**) data into buffer
  To convert buffer to byte[] array -> **buffer.flip().array() //** Here flip is use to position marker at start otherwise it will make blank space array.
- Step-4 close the client connection -> client.close(); and return to step 1.

Client-Socket

UTF-8 It take for English character 1 byte to store and for non-english it take 2 bytes.
UTF-16 It take for All character 2 byte to store
ASCII It take for English character 1 byte and for others character it gives error.

- To set character -> Charset.forName("UTF-8");
- **Step-1** To open server ->**var server = SocketChannel.open(StandardProtocolFamily.UNIX);**
  To connect to server->**server.connect(UnixDomainSocketAddress.of(Server.NAME));**
- **Step-2**
  To write data on buffer in bytes and data in stream -> **server.write(cs.encode(text));**
  To create buffer ->**var buffer = ByteBuffer.allocate(80);**
  To read data from buffer->**server.read(buffer);**
- **Step-3** To display data-> **System.out.println(cs.decode(buffer));**

TCPIP-Server Test :-

- To Set time to automatically disconnect to server we used ->**client.setSoTimeout(20000);**
- Socket can only read **BYTES** and here inputStramReader reads bytes and convert into array of charter after it BufferedReader join array of character and make text ->**var reader = new BufferedReader(new**

InputStreamReader(input));

- Here PrintWriter converts array of character into stream and output required bytes so OutputStreamWriter converts stream into bytes ->**var writer = new PrintWriter(new OutputStreamWriter(output));**
- PrintWriter contain buffer and it gives to OutputStreamWriter but that buffer full for immediately send to OutputStreamWriter we used **writer.flush();**
- By using above pattern we can connect to only one client at time so to make connect multiple client we used child thread ->

```
for(int i = 0; i < 3; ++i){
        var child = new Thread(() -> {
            try{
                service(listener);
            }catch(IOException e){}
        });
        child.start();
    }
```

TCPIP-Client Test :-

- We used to split string over "," Operator ->**String[] parts = text.split(",");**
- To parse string into primitive type we used ->double p = **Double.parseDouble(parts[0].substring(leave_charcter));**

DATABASE

10 June 2022 23:18

Relational Data : It is a form of data, logically structured into tables as column wise which can be referenced by columns of another table to support parent-child relationship between those tables.

1. **Schema :** Type of value that can be stored into rows of that table.
2. **Constraints:** Restrictions on the values that can be stored into that table.

Transactional Data: It is a form of persistent data whose different parts can be updated within a single unit of work known as a transaction with support for a commit operation to persists the new state of data and a rollback operation to restore the data to its original state. A well behaved transaction has following ACID characteristics.

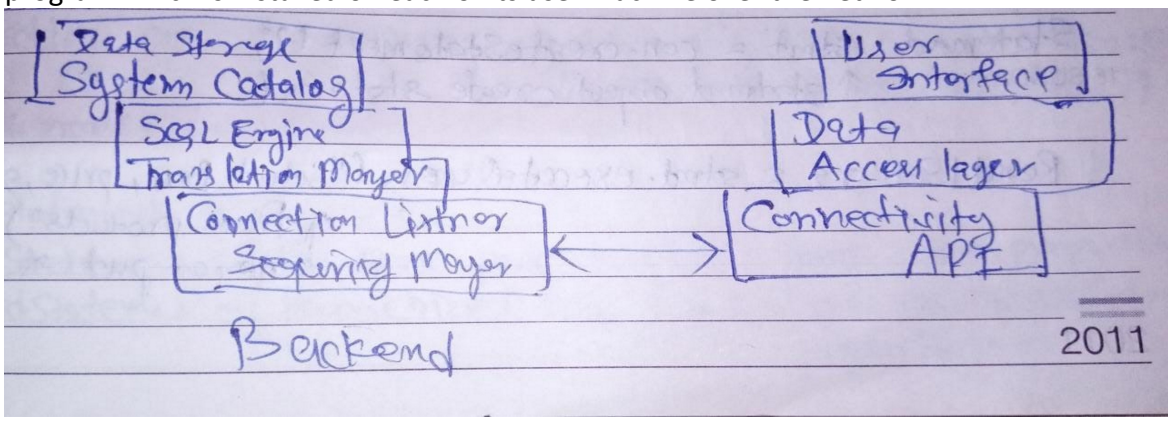
1. **Atomic** - A transaction should only perform an update whose effect can be cancelled by the rollback operation. (Like create table can not be cancelled by rollback operation).
2. **Consistent** - A transaction should rollback as soon as any of its update violates a constraint on the target data.
3. **Isolated -** A transaction should not allow any data it accesses to be updated by an operation that does not belong to its scope.
4. **Durable -** A transaction should always end either with a commit or a rollback operation.

Relational Database System: It is a software that manages persistence of relational and transactional data with support for

1. Consuming this data using an API based on (a fourth generation(compile into c then assembly then machine language and at last binary) declarative) structured query language (SQL).
2. Sharing this data in a secure manner based on authentication (confirming identity of a user) and authorization (granting permission to a user) schemes.

Database Application: It is software that automates a certain business process by performing a set of transactional operations known as the business logic on some persistent data. Its implementation is generally divided to execute as at least two separate but communicating processes with following responsibilities

1. **Backend -** It manages persistence of transactional data for the application**.** If the business logic to be changed at one placed, so it implemented by Backend and it increases **maintaiability.** It is using relational database system installed on a central machine over the network.
2. **Frontend -** It is provides a user-interface for the application. When we required to change business logic at many places so it implemented by **Frontend** to increases **Scalability.** Its commonly supported using client program which is installed on each of its user machine over the network.



If the logic not going to be changed frequently so it is implemented by Backend and logic require to be changed frequently so it is implemented by frontend. DBS also called client server architecture.

Java Database Connectivity (JDBC): It provides a standard API (through java.sql package) for consuming data managed by a relational database system. Different database work with different approach, So JDBC designed with interface, When we changed classes but those same classes implemented by same interface and dependency principle says that, called method by interface not on classes. So it will work with all database system. And that jdbc implementation called **JDBC Driver.**

1. **Connection -** it opens a communication session with the database system using its URL.
   SQLITE:
   Connection con = DriverManager.getConnection("jdbc:sqlite:sales.db", **"USERNAME",** "**PASSWORD**");
   DriverManager: Connection is **interface** and DriverManager gives us connectin. It Find all jdbc given by class path and asked to every driver that do you recognised this URL(jdbc:sqlite:sales.db) and after find the suitable driver tell it to connect database.
   MySQL:
   Connection con = DriverManager.getConnection("jdbc:mysql://localhost/sales", "dbpro", "Dbpro@789");
   Oracle:
   Connection con = DriverManager.getConnection("jdbc:oracle:thin:@//iitdac.met.edu/xe", "binarybrains", "metiit");

2. **Statement -** it sends SQL commands to the database system using the connection object.
   To create statement Object (**Bridge Pattern**)
   Statement stmt = con.createStatement(); **-> This statement we used for simple SQL**
   To insert data into table Row
   PreparedStatement pstmt = con.prepareStatement("insert into orders values(?, ?, ?, ?, ?)"); **-> This statement we used for Parametrised SQL**
   This ? Mark shows that we have to add data in columns
   By using **pstmt.setInt(1, orderNo);** We add data columns first.
   CallableStatement interface is used to call the stored procedures and functions.

   We can have business logic on the database by the use of stored procedures and functions that will make the performance better because these are precompiled.

3. **ResultSet** - it fetches(rs.next()) the rows resulting from execution of a query command sent using the statement object.
   There are two type of command
   i. ResultSet rs=stmt.executeQuery("select "**columns_name**" from products");
   ii. ResultSet rs=stmt.executeUpdate("update products set stock=stock+5 where pno=" + id);
   We will fetch row by using **while(rs.next())**

To save the updated data on database we used command
con.commit();
update violates a constraint on the target data so we used command
con.rollback();

Java Persistence API(JPA): JPA allows us to write classes and Load object of those classes from database, By using JPA we dont need to change code when we change database system. JPA is used to built **Data Access Layer** through **object** .
It specifies standard support (through javax.persistence package) for accessing relational data using instances of data-context unaware Java classes. The implementation of this API (such as EclipseLink and Hibernate) is called the JPA provider and it includes support for

1. **Entity** - It is a serializable plain old Java object (POJO-> a class contain fields and accessor method for thoes

fields and C++ caled as POD->Plain Old Data) containing at least one identity field which is mapped (in META-INF/orm.xml or through annotations) along with its other basic fields to columns of a database table mapped to its class.

2. **Entity Manager** - It loads entities specified using Java persistence query language (JPQL) from their mapped tables and handles persistence of these entities into their mapped tables in a transactional manner.
3. **Entity Manager Factory** - It creates an instance of entity manager from its persistence unit (configured in META-INF/persistence.xml) which includes information required by the JPA provider to connect to the database and for reading/writing data within entities from/to tables of that database.

XML-MAPPING

- Step-1 Create **entityManaferFactory object** which configure our persistence.xml file
  ->var emf = Persistence.createEntityManagerFactory("ShopPU"); // here ShopPU is **PERSISTENCE Unit**
- Step-2 here we create object of **entityManager** which loads all data-> var em = emf.createEntityManager();
- Step-3 By using entity we create object of particular class in database by using **JPQL(Java persistence Query language. Its a declarative** language E.g. SELECT **p.price** FROM ProductEntity p WHERE stock > 10-> gives **price)**->var query = em.createQuery("SELECT p FROM ProductEntity p", ProductEntity.class);
- Above statement gives us all object of class but to find particular object we use
  ->ProductEntity product = em.find(ProductEntity.class, pno);
- If threr is any realtion between two tables or same column then we have to define relation inside orm.xml file of parent table like ->

```
<one-to-many name="orders">
    <join-column name="pno"/>
```

JPA-ANNOTATION

- Here we do mapping through annotation
- Table name mapping->

```
@Entity
@Table(name="Table_name_at_database")
```

- Table column primary key **Or** Id column mapping->

```
@Id
@Column(name="Column_name_at_database")
```

Web

01 July 2022 19:36

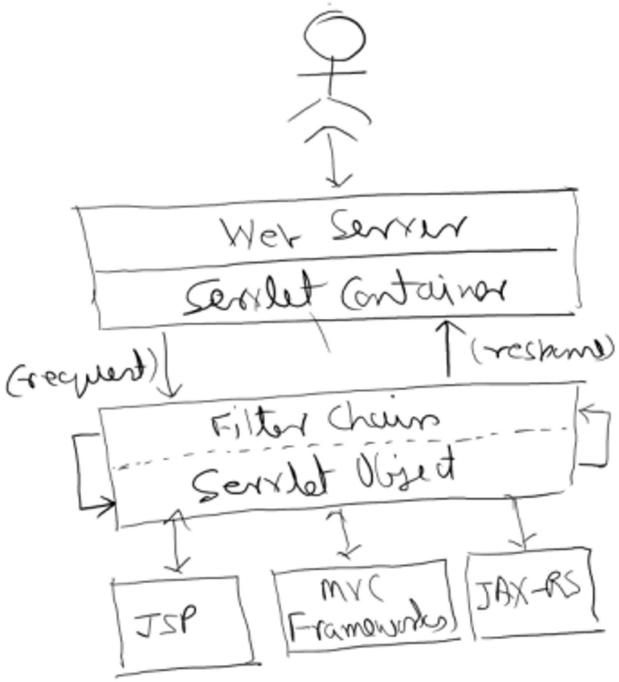
Web Application: It is a software executed by web-server to publish dynamic content over its HTTP endpoint so that this content can be presented to the user through a web-browser. Web browser publish web pages nothing but a html file and html file are **static** but web application by using Server, it generate html file at runtime and send to the browser HTTP endpoint.

Classic Web Application	**Modern Web Application**
The data is acquired and by using server side code, generate output as html file and this output is transported to the browser for presentation.	The data is acquired by the server side code where both code and data transported to the browser where it is rendered by the client-side code for presentation.
Input is received by the browser and transported to the server where it is handled by the server side code.	Input is received by the browser where it is directly handled by client side code which calls server side code if required.
Implementation requires familiarity with server-side frameworks such as Spring (Java) and ASP.NET (C#).	Implementation requires familiarity with server-side frameworks as well as client side frameworks such as Angular and React.
Application is more secure and is independent of type/version of the browser.	Application is less secure and may depend upon the type/version of the browser.
UI is less responsive because every interaction requires its re-rendering on the server.	UI is more responsive because interactions do not require any re-rendering
It uses patterns suitable for large applications with multiple web-page	It uses patterns suitable form a single page application (SPA)

Java Servlet API: It specifies standard support through (jakarta.servlet package) for implementing a Java server-side object known as a servlet which generates a response to a request of a particular type received by the web-server. A web-server (such as TomCat, GlassFish, WebLogic, WebSphere) which supports deployment of Java web-applications includes an environment called a servlet container which provides the implementation of the Servlet API and handles each request received by the server using following steps:

1. Get the servlet object mapped (in WEB-INF/web.xml or through annotation) to the URL pattern matching with the requested path initializing a new object from the corresponding servlet class (which implements jakarta.servlet.Servlet interface) if required.
2. Invoke the service method of the above servlet object passing it the current request and response objects as arguments and when this method returns, pass the content of the response

object to the web-server so that it is transported back to the requesting client.



- **To start glassfish domain->** ~/glassfish6/bin/asadmin start-domain
- **To deploy** ->~/glassfish6/bin/asadmin deploy SimpleWebApp
- **To undeploy** -> ~/glassfish6/bin/asadmin undeploy SimpleWebApp
- It will undeploy and again deploy->touch SimpleWebApp/.reload
- **To export path of servlet container**->export CLASSPATH=.:~/glassfish6/glassfish/lib/javaee.jar
- To check server is running type on Url->**localhost:8080 //** 8080 is port number

Implements Servlet(WEB.XML Approach)

- create one directory-> **mkdir -p SimpleWebApp/WEB-INF/classes** //inside web-inf put **web.xml** file
- Inside servlet interface, five method declared -> init,getServletConfig,service,getServletInfo,destroy

- All four method will be same for all program but service method will change
  1.read data from request 2. generate response 3.output that response to client
- After compilation we should put classes inside WEB-INF/classes .
- To get object of response.getWriter() ->var out = response.getWriter();
- And whatever we write through object of response.getWriter() will send to client-> out.println("<l>");
- According to first rule it will map servlet class matching with URL if not available it will initialize object of class which inherit from servlet

HttpServlet(Annotation Approach)

- HttpServlet also comes from **import jakarta.servlet.http.*;** which having five all method but it is also have **doGet()** and **doPost()** method having parameters (HttpServletRequest request, HttpServletResponse response) so we can override.
- **To read request data ->**String guest = request.getParameter("user");
- If user put nothing and click on post get button so we redirect at home page-> **response.sendRedirect("home_page");**
- To know about page type->**response.setContentType("text/html");**
- **To Map witch class->**

```
@WebServlet("/count")
public class CLASS_NAME extends HttpServlet
```

PAUSING-FILTER

- This class implements by Filter interface.
- There are three method inside it
  1. **init(filterConfig cfg)**
  2. **doFilter(ServletRequest request, ServletResponse response, FilterChain next)**
  3. **destroy()**
- After filter we transfer request to next process->**next.doFilter(request, response);**
- All request should go through this filter class we use before filter class name as **@WebFilter("/*")** here **/*** means accessible to all request ***.html** for all html file if write **/welcome** means for welcome html only.

Java Server Pages(JSP) :It provides servlet based support for enabling a web-application to publish dynamic content by combining client-side markup (HTML) with server-side elements (expressions and tags) in a single web-page. The URL pattern of Browser side JSP page(*.jsp) is mapped to the built-in JSP servlet which execute server side code and serve to the request page.

- In side JSP we can write expression like **${ }, ${empty param.user ? 'Visitor' : param.user}**

The programming model of JSP includes support for:

1. **Java-Bean -**Java Bean are java Object. Its class **implements** through **serializable**. This class supports a **parameter-less constructor** and exposes instance properties using methods which follow standard **get-set** naming convention but for Boolean(true or false) properties **is** permitted .

- To use JavaBean we have to write-><**jsp:useBean id="counter" class="basic.web.app.CounterBean" scope="application"/>**

  Where **id** use as JavaBean object in class we have to write full class name

scope=" "

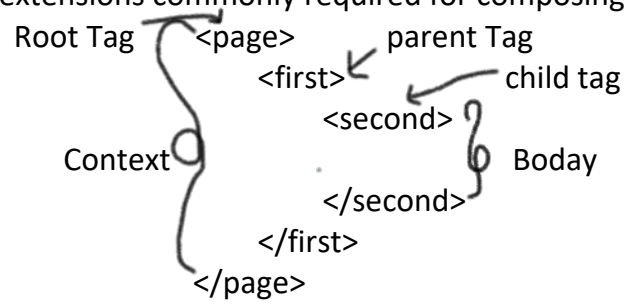
if we dont provide **scope** it will treat as **default**(page scope)means it will create object for every request .

if we provide **scope** as **session**(browser scope)means it will create one object for one user.

if we provide **scope** as **application**(application scope) means it will create object for every user.

- Use to method we dont write get and set like we wrote method **getNextCount()** to use in JSP inside expression **NextCount()**
- **To use the property** of class-><jsp:setProperty name="greeter" property="*"/> **//** * is to use all property

2. **Tag-Extension** - It is an object used specifically by a JSP page for handling a custom (user-defined) server-side markup element (tag) which is use to mapped JSP class (which implements jakarta.servlet.jsp.tagext.SimpleTag interface) along with other information in a URI identified XML document called the tag-library descriptor. The JSP standard tag library (JSTL) provides tag extensions commonly required for composing JSP pages.

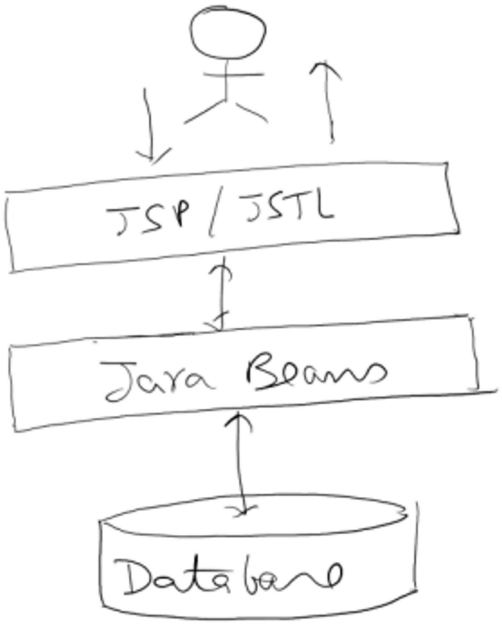


- Information inside Tag is context or body and inside tag<first key="Attribute">
- Inside jakarta.servlet.jsp.tagext.SimpleTag interface, there are five method->**doTag, setParent, getParent, setJspContext, setJspBoday.** And if we dont want to write all five method we used class extends SimpleTagSupport.
- First we implement above method and inside all set get method assign value.
- If we provide other setter method will treat as **Attribute.**
- But inside doTag() method->

  var out =context.getOut()-> it tell the context gives output this tag means in which reason this tag.

Taglib.tld file(TAG Library Descriptor)

- Tld file is description of **custom TA**G, A class of TAG to mapping we defined tld file.
- Give any name inside this tag-><**short-name>basic</short-name>**
- Provide any URL but we have to required-><**uri>http://java.met.edu/basic/web/app</uri>**
- Give name of Tag-> **<name>clock</name>**
- Give full name of tag class-> **<tag-class>basic.web.app.ClockTag</tag-class>**
- Either empty(like input tag) or scriptless(tag not contain script)-> **<body-content>empty</body-content>**
- By using **<%@ taglib prefix="my" uri="http://java.met.edu/basic/web/app" %>** inside jsp file we can use custom tag
- Search all tld file look **above uri** over there and **short name** of this **uri** is **prefix name**.
- We define this TAG by using prefix like **my:clock**
- We can use attribute(specified extra method inside class tag)-> **<my:clock format="dd/MM/yyyy hh:mm:ss a"/>**



JAX-RS (Java API for Restful Services):

- pool.setURL("jdbc:mysql://localhost/sales?useSSL=**false**"); **->** We make it **True** when we want our **username and password** in **encrypted** form.