JAVA NOTES

Core Java (Only Theory part)

Q1. What is Java?

Java is the high-level, object-oriented, robust, secure programming language, platform-independent, high Performance , Multithreaded, and portable programming language. It was developed by James Gosling in June 1991. It can also be known as the platform as it provides its own JRE and API.

Constructor

- ✓ Constructors should have same name as that of class.
- ✓ Whenever an object is created constructor is being called.
- ✓ Constructors are permanently void and hence they cannot return any value.
- ✓ Because it is void we can only use keywords in it.
- ✓ Supplying values to constructor, then yes we can create multiple constructors.
- ✓ In constructor, method name and class name can be same.

Constructor Overloading

Here we create more than one constructor in the same class provided that they have different number of arguments or different types of arguments.

JDK - It helps us to compile .java file to .class file.

JRE - JRE helps us to run .class file.

New Keywords

- ✓ It sends request to the class to create object.
- ✓ New keywords mandatorily call constructors.
- ✓ Once object is created, then it gets its address and stores that in a reference variable.

This Keyword

- ✓ It is a special reference variable that holds object address. This keyword creates automatically.
- ✓ This keyword points to current object running in the program.
- ✓ We cannot use This keyword inside static method.
- ✓ Using This keyword we can call constructor.

Constructor Chaining

When we call a constructor from another constructor using This keyword then it is called as Constructor Chaining.

Instance Variables

Instance variables in java are non-static variable which are define in a class outside any method, constructor or a block.

Static Variable

- ✓ Static Variables can be accessed using class name.
- ✓ Static variables can be accessed by static & non-Static method
- ✓ Static Variables is like a global variable and is available to all method.

Non-Static Variable

- ✓ Non-static variables can be accessed using instance of a class.
- ✓ Non-Static variables cannot be accessed inside a static method.
- ✓ Non-Static variable is a like local variable and they can be accessed through only instance of a class.

Inheritance

Here we inherit the member from parent class to child class with an intension of reusing it.

Packages

Packages are nothing but folder created in java where programs can be stored in an organized manner.

Packages resolve naming convention problem.

Advantages of Inheritance

- ✓ Inheritance minimizes the identical code as it allows sharing of the common code among other subclass.
- ✓ Inheritance makes the code flexible to change.
- ✓ With the help of inheritance, we can override the methods of the base class.

Polymorphism

Here we can develop a feature in a way that it can take more than one form.

Polymorphism is applicable only on method.

There are two ways we can achieve polymorphism-

i)Overriding

ii) Overloading

Overriding

Here we inherit a method from parent class and modify its logic in child class by once again creating same method in child class.

Advantages of Polymorphism

Overriding

If we inherit 10 methods but some methods logic if we modify then the option to do that is Overriding.

@Override

@Override annotation checks whether overriding is happening or not if not then it report an error.

Q2. Can we override static method?

Ans- No. In java static members are never inherited and if inheritance cannot happen then overriding also cannot happen because Overriding is add on features of Inheritance.

Overloading

Developing more than one method in the same class provided that they have different number of arguments or different types of arguments then it is called as Overloading.

Type Casting/Data Type

Converting particular data type into required data type is called as type casting.

There are two types-

I. Auto Up casting
II. Explicit Down casting

Auto Up Casting

Converting smaller data type to bigger data type is called as Auto Up Casting. During Auto up casting data loss should not happen.

Explicit Down Casting

Here we convert bigger data type to smaller data type. During explicit down casting, there are chances of data loss might happen.

Class Up Casting

Here we store child class object address into parent class reference variable.

Class Down Casting

Here we store parent class object address into child class reference variable.

Run-Time-Polymorphism

In run-time-polymorphism we perform overriding with class upcasting.

Interface

Interface can consist of only incomplete method in it.

Q3. Can I create static incomplete method in an interface?

Ans- Interface does not support incomplete static method because overriding of that is not possible.

Q4. What is abstraction?

Ans- Hiding of implementation details is called as abstraction. The way we achieve this in java is by using interface and abstract class.

Abstract Keyword

When applied on a method it defines that the method is incomplete.

In an interface we can create incomplete method e1 without using abstract keyword. Uses of abstract keyword here is optional.

When abstract keyword is applied on a class then it means class is incomplete.

Note: In Java, at interface level multiple inheritance is possible but at class level it is not possible.

Q5. What is Marker Interface?

Ans- An empty interface is called as marker interface.

Final Keywords

- ✓ If we make variable final then its value cannot be changed.
- ✓ If we make static/no-static variable final then initialization is mandatory.
- ✓ If we make a method final then overriding is not allowed.
- ✓ If we make class as final then inheritance of that class is not allowed.

Q6. Explain Java 8 new features.

Ans – **Default Keyword:** Default keyword was introduced in version 8 of java using which we can develop complete method in an interface.

Functional Interface: It should consist of only one incomplete method in it.

In a functional Interface we can have any number of default methods but incomplete method should be only one.

Lamdas Expression: The advantage of lamdas expression is we can reduce number of times of code.

Note:

- As we can access non-static member of the class using lamdas expression, it is also **functional programming language** since 1.8version of java.

- When we create object to access non static member it becomes **object oriented programming language**. We create object in java using new keyword.

Abstract Class

- An abstract class can consist of both complete and incomplete method.
- To define incomplete method in abstract class usage of abstract keyword is mandatory.
- In an abstract class, we can create main method.
- Creating object in abstract class is not allowed.
- Abstract classes do not support multiple inheritances.
- In an abstract class we can create static variable as well as no static variable.

Q. What is Data Hiding?

Ans- Here we make variable private so that it can't be accessed outside the class.

Unary Operator

In java, the unary operator is an operator that can be used only with an operand. It is used to represent the positive or negative value, increment/decrement the value by 1 and complement a Boolean value.

Scanner Class

Whenever a user wants to give input via keyboard, in java there is inbuilt class Called as a Scanner Class.

Encapsulation

Bundling of data with methods which operate on that data avoiding direct access to the variable is called Encapsulation.

To avoid direct access to the variable we makes variable private and to operate on those variable we create getter and setter.

Differences between interface and abstract class.

Interface	Abstract Class
Interface can contain only abstract methods.	Abstract class can contain both complete and incomplete method.
Every variable in an interface is static and final	Abstract class can be 0 to 100% incomplete.
Interface support multiple inheritance.	Abstract class do not support multiple inheritance
Interfaces are 100% incomplete.	
An interface can consist of main method in version 8 of java onward.	

Access Specifier:

Variable/Method:

- If we make variable/method private then it can be accessed only in same class.
- If we make variable/method default then can be accessed in same class and same package only.
- If we make variable/method protected then can be accessed in same class and same package and different package only through inheritance.
- If we make variable/method public then can be accessed in same class and same package and different package also.

Class:

- A class can't be private /protected.
- If a class is public then it can be accessed in same package/different package.
- If a class is default then it can be accessed in same package only.

Constructor:

- If we make constructor private then its object can be created only in same class but in different class.
- If we make constructor default then its object can be created in same package but not in different package.
- If we make constructor protected then its object can be created in same package but not in different package.
- If we make constructor public then its object can be created in same package and in different package also.

Access Specifier(VIP)

Private: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.

Default: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.

Protected: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.

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

IIB- Instant Initialization Block

- IIBs are executed when objects are created.
- No. of times we create an object, same no. of times IIB will be called.
- IIBs are used to initialize all the instance variable in one place and that give us better readability of the code.
- We can initialize both static and non static variable inside IIB.

SIB- Static Initialization Block

- SIB runs before main method and it does not require any invoking statement.
- We can not initialize non static variable inside SIB.
- We can create an object inside SIB.

Super Keyword

- Using Super Keyword, we can access the member of parent class.
- Using super keyword, we can access static and non static member both.
- Super keyword can not be used inside static context.
- We can use super keyword only when inheritance is happening otherwise we cannot use super keyword.

File Handling:

Exist(), delete(), createNewFile(), mkdir()(to create new folder), length()(to checks character in a file), list()(to give all file name In the given path), FileReader(to read file content),.

Mutable: Mutable is something wherein the class object properties keeps on changing.

Exception:

Whenever a bad input is given then it stops the program updroply and that is called as exception.
To handle exception in java we use try and catch block.
When any line of code in try block causes exception then try block
create exception object and that exception object address is given
To catch block . Catch block will now suppress the exception and once the exception is suppressed, the further code will continue to execution .
To get exact line number where exception occurs we use printStackTrace.

Types of exception:

i)**Run time exception (Unchecked Exception):** If we get exception while running .class file then it is called as run time exception. e.g.
-Arithmetic Exception
-NullPointer Exception
-Numberformat Exception
-ArrayIndexOutOf Bound Exception
-classCasting Exception

ii)Compile time exception (checked exception): If an exception occurs when .java file is converted to .class file then it is called as compile time exception .e.g.
-SQL Exception
-IO Exception
-FileNot Found Exception
-ClassNotFound Exception

Q. Which is super most class in java?

Ans- Super most class in java is object.

Array:

Array in java is a special object with continuous block of memory to store collection of data in it.
In java, an array of length zero can be created but no value can be stored in it.
It is not mandatory to initialize array. If not initialized then depending on data type default value get stored in it.

Main method Signature:

Args in main method is a variable which is a user defined. It is a method argument of the type array and it can have any name.
String args in main method is used to receive command line arguments.

Immutable: Immutable class once its object is created then its state can not be alter.

Steps to create immutable class:

-create a final class.
-Set the values of the properties using only constructor.
-Make the properties as final
-Do not provide any setters for these properties.

Note:

The area where these immutable objects are being created that area is called as **String Constant Pool**.

Trim- to remove blank space

ValueOf method converts given data type such as int, long, float, double, Boolean and char array to string.

Threads in Java:

Multi-tasking done at program level is called as threads.
The main purpose of thread is to improve the performance of the application by reducing time.
There are two ways we can build threads
i)Build-in thread
ii)User-defined thread

Thread Synchronized:
When two threads are operating on common data, the data might get corrupted because of multi-tasking.
To make thread operate one after another, we use synchronize keyword wherein the thread has acquired the lock can only execute the block where as other thread would be in wait status.
Only when the first thread release the lock the other thread will get the opportunity to acquire the lock and execute the block.

Thread Priority:
It decide which thread is going to run first and which thread will run later.
If we set the priority then it is a request made to the thread scheduler where there is no assurity that it will be approve and process.
The minimum thread priority is 1, maximum thread priority is 10 and the normal thread priority is 5.
However we can set the thread priority with a number anything between 1 to 10.

Thread Pool:
Thread pool are useful when is needed to limit the number of threads running in our application at the same time. This will help us to improve the performance of the application.
Instead of starting new thread for every task execute can currently, the task can be passed to a thread pool.
A thread pool contain collection of threads as soon as the pool has an ideal thread, the task is assign to one of them and execute. Threads pool are often used in server. Each connection arriving at server via network is rapped as task and passed on a thread pool. The thread in thread pool will process the request on the connection concurrently. This is how we can use existing thread instead of creating new thread and there by improve the performance in term of execution.

Enum: Enum is collection of constant.
Wrapper Class- Here, the value are stored in object. The processing of storing the value inside an object is called as Wrapping or **boxing**.
Reading the value from the object is called as **unboxing**.
Finalize is a method present inside object class.
Garbage collection logic is implemented in Finalize method.

Throws Keyword: Throws keyword is applied on a method if any exception occurs in the method then the exception will be passed on to the calling statement of the method.

Throw Keyword: Throw keyword helps us to create customized exception as per the requirement of the developer.

Regular Expression:

\\s – It is used in regular expression to give white space in a given string.
\\S- It give us all the thing other than white space in a given string.
\\d – It will search for digit only.
\\D – It gives everything except digit in a given string.
\\w – it will give lower case letter, upper case letter and digit only.
\\W – It give all thing except lower case, upper case and digit.
Char* - It give zero occurrences or group of occurrence of a particular character.
Char+ - It gives only group occurrence of a particular character.
Char? – It give us zero occurrence or single occurrence of a particular character.

Tokenizer :The Tokenizer class allows us to break a String into tokens. It is simple way to break a String. It is a legacy class of Java.

Cloning: The process of creating the replica of a particular object by copying the content of one object completely into another object.

Annotation

@Override annotation checks whether overriding is happening or not. If not it reports error.
@SuppressWarning annotation is used to suppress warning used by the compiler.
@Deprecated annotation marks that this method is deprecated so compiler prints warning. It informs user that it may be removed in the future version. So, it is better not to use such method.

BufferedReader: BufferedReader is use to increase the performance and can also read the data line by line.

Q. What are marker interface and where is used?

Ans – During Serialization, we have marker interface called as serializable, until and unless class does not implemented that we can't store the object state into the file.

JDBC: JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute the query with the database. It is a part of JavaSE (Java Standard Edition). JDBC API uses JDBC drivers to connect with the database.

Q. What do you mean by CRUD operation?

Ans- CRUD stands for Create/insert Read/Retrieve Update Delete.
CRUD operation is a basic principal for any developer to interact with database.

Finally

Finally block is an extension of try, catch. Regardless of exception happens or not, finally will execute.
We can also write only try and finally.

Q. Give practical example where finally keyword is used.

Ans- When we write jdbc code closing of data base connection, I can do that in finally block.

Collection in java

Collection: collection stores group of object in it. In java, collection is a framework which has readily available logic to deal with different data structure.
ArrayList: Internally it is implemented as dynamic array.
Initial size of arraylist is 0
When we exceed the initial size, automatically arraylist size increases by 1.5 times. ArrayList maintain insertion order. It can consist of duplicate elements.
Advantages of ArrayList- Reading of data would give us best performance .
Disadvantages of ArrayList – Insertion of data in between of the arraylist will result in performance.
Note: In JDK, linked list is internally implemented as doubly linked list.

HashTable

Hashing: Hashing is a technique where we are representing any entity in the form of integer and it is done in java using a hashcode method.
HashCode – Hashcode is a method present inside object class in java.
HashTable – HashTable is an associated array where in the value store as a key value pair.
Initial size of a hashtable is 16 byte when load ratio becomes 75% that is out of 16 twelve elements are injected into the table then the size of table becomes double automatically.
HashTable is Synchronized.
Collision- When two values are being store in the same index number then it is called as collision.
To solve this problem in hashtable we store the data as list mapped to the same index number.

Set – set is an interface and does not maintain any insertion order. It can't contain duplicate values.
HashSet – HashSet uses hashtable internally. It uses hashing to inject the data into the data base. It will contain only unique elements. It does not maintain insertion order. This class permit the null elements.

Q. Difference between hashset & linked Hashset.
Ans- HashSet does not maintain insertion order but linkedhashSet maintain insertion order.
LinkedHashSet: LinkedHashSet maintain insertion order. It can contain only unique elements.
TreeSet: TreeSet contains unique elements only. It sorts the data in ascending order.
HashMap: HashMap internally uses hashtable. To inject data into hashtable, it uses hashing technique. A hashmap stores the data as key value pair. Hasmap is not synchronized.

Comparator

Comparator is actually an interface comparator interface is used to order the objects of user-defined classes.
Comparator is an interface that compares to object if in sorting object 1 comes first then object 2 then it will return negative value but if while sorting object 2 comes first and then object 1 then it will return positive value. If both objects are same, it will return zero(0).
JDK-9 : A new features jshell was added. That help us quickly build the java code in it.
In JDK-10: var data type was covered.

Serialization: Serialization is a mechanism of converting the state of an object into a byte-stream.

Deserialization: Deserialization is the reverse process where the byte-stream is used to recreate the actual java object in memory.

Collection & Collections?

Q. Why String is immutable in java?

Ans-The immutable string means it cannot be modified once it is created.
String is immutable in Java because of the security, synchronization and concurrency, caching, and class loading. The reason of making string final is to destroy the immutability and to not allow others to extend it.

JAVA NOTES

Advance Java (Only Theory part)

OOPs In java

Object-Oriented Programming System (OOPs) is a programming concept that works on the principles of abstraction, encapsulation, inheritance, and polymorphism. It allows users to create objects they want and create methods to handle those objects. The basic concept of OOPs is to create objects, re-use them throughout the program, and manipulate these objects to get results.

Abstractions: Hiding of implementation details is called as abstraction. The way we achieve this in java is by using interface and abstract class

Encapsulation: Bundling of data with methods which operate on that data avoiding direct access to the variable is called Encapsulation.

Inheritance: Here we inherit the member from parent class to child class with an intension of reusing it.

Polymorphism: Here we can develop a feature in a way that it can take more than one form. Polymorphism is applicable only on method.

Q. What are Servlet?

Ans- Servlet is a Java class that extends httpservlet and is responsible to perform backend coding.

Q. What is Inter Servlet Communication?

Ans- When one servlet calls another servlet using request dispatcher concept then it is called as Inter Servlet Communication (ISC).

Note: request.get and request.set will work only when request.dispatcher is used.

Session Variable: Once we store the data in session variable then that we can access across the application.

JSP- Java Servlet Page

JSP helps us to embed partial java code inside html.

JSP Tags are-

i)Scriplet Tag

<% Java code……%>

ii) Decleration Tag

<%! Java code….%>

iii)Expression Tag

<%= java code…….%>

iv)Directive Tag

Q. Mention implicits object of JSP.

Ans- request,response,out,session are implicit objects of jsp.

MVC (Model View Controller) Architecture:

MVC architecture is a three layered architecture wherein all HTML code, JSP code is kept in **view layer**. **Controller layer** is responsible to interact with a View, take the data from view and gives it to model layer. Business logical implementation/ database implementation is done in **model layer**, the output of the model layer is given to controller and that data further controller gives it to view. This architecture helps us to maintain the application in easy manner.

MVC architecture flow-

View →Controller →model→Controller→view

Q. Where did you achieved polymorphism?

Ans – After inheriting incomplete method, I was overriding child class to complete that model and here I achieved polymorphism.

Q. What is abstraction? Where have you used this in practical?

Ans- Hiding of implementation details is called as abstraction and the way we achieve this in java is by using interface and abstract class.

In my project, I was developing an interface layer DAOService , where I defined all the incomplete method and implemented that in the class DAOServiceImpl wherein all database related queries where written down.

Q. What is Inheritance? Give practical example.

Ans- Here we inherit the member from parent class to child class with an intension of reusing it.

In my project, I Created an interface DAOService and from here I inherited incomplete method to class DAOServiceImpl.

Servlet Life Cycle:

For the first time, when we start Tomcat init() method will run and it will run only once.

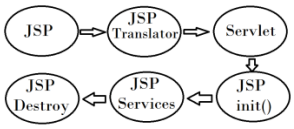
Then services method (doGet,doPost) are called and it can run any number of times.

Finally, when tomcat runs destroy method is called, the servlet life cycle comes to end.

Q. What are application servers?

Ans- Application server are responsible to run technologies like servlet, jsp, ejp(enterprise java bean), spring boot etc.

JSP Life Cycle:



JSP translator will translate JSP file into servlet.

This servlet will consist of three important methods:

i)-jsp int () – This method will run only once after tomcat started.

ii)-jsp Services () – This method can be called any number of times depending on business requirement.

iii)-jsp destroy method() – This is called by tomcat and this is last method to run.

When this method runs then it means jsp life cycle come to an end.

Q. What are WEB Servers?

Ans- Web Servers are responsible to run static application. Examples of server are –IIS, NGinx, google web server etc.

Q. Where does Maven download the Jar file?

Ans – Maven download the jar file in M2 folder.

Q. What does pom.xml file do?

Ans- pom.xml file instruct Maven Dependencies to download jar file.

Q. From where does Maven download dependencies?

Ans – It has own website called as MVN repository.

Note: Objects in SpringBoot are term as Bean.

Q. What is Dependency Injection?

Ans- Spring boot ideally creates object during runtime and this object address injected to reference variable. This technique is called as Dependency Injection.

Q. Which version of hibernate you were using in your project?

Ans- Hibernate core 5.6v

Web –Services:

Web Services help us to integrate heterogeneous and homogeneous application.

There are two ways we can implement web-services-

SOAP(Simple Object Access Protocol)- Here we exchange the data between application using xml file. Implementation of SOAP is difficult because it is complex to parse xml file.

REST(REpresentational State Transfer)- Here we exchange the data between application using JSON(JavaScript Object Notation) object.

The content of JSON object stored as a key value paired. It is easy to implement web-services using JSON. REST also supports exchanging of data using xml file.

Micro-Services:

Here we break bigger application into smaller mini project and then we establish the communication between these application using web-services.

Monolithic application- Here we put all the code in one place and then we host that on the same server.

Singleton Design Pattern: Here we create a class such that only one object of that class can be created throughout the execution.

The controller Layer is a protocol interface which exposes application functionality as RESTful web services.

The repository layer abstracts persistence operations: find (by id or other criteria), save (create, update) and delete records.

The service layer contains our business logic. It defines which functionalities we provide, how they are accessed, and what to pass and get in return.

Q. What is RESTFful web services?

Ans-

Annotation:

@Autowired- @Autowired helps us to perform dependency injection.

@Controller- @Controller helps us to define controller layer in our spring boot application.

@RequestMapping – @RequestMapping maps form-url with controller method.

@ModelAttribute- @ModelAttribute maps the form data to entity class object.

@RequestParam- @RequestParam maps form data with controller method arguments.

@RestController- @RestController help us to define web-services layer in our project.

@GetMapping- @GetMapping get all the data from DB and maps it JSON object.

@PostMapping- @PostMapping maps the JSON object content to DB.

@PutMapping- @PutMapping help us to update data in DB using web-services.

@DeleteMapping- @DeleteMapping help us to delete the record in DB using web-services.

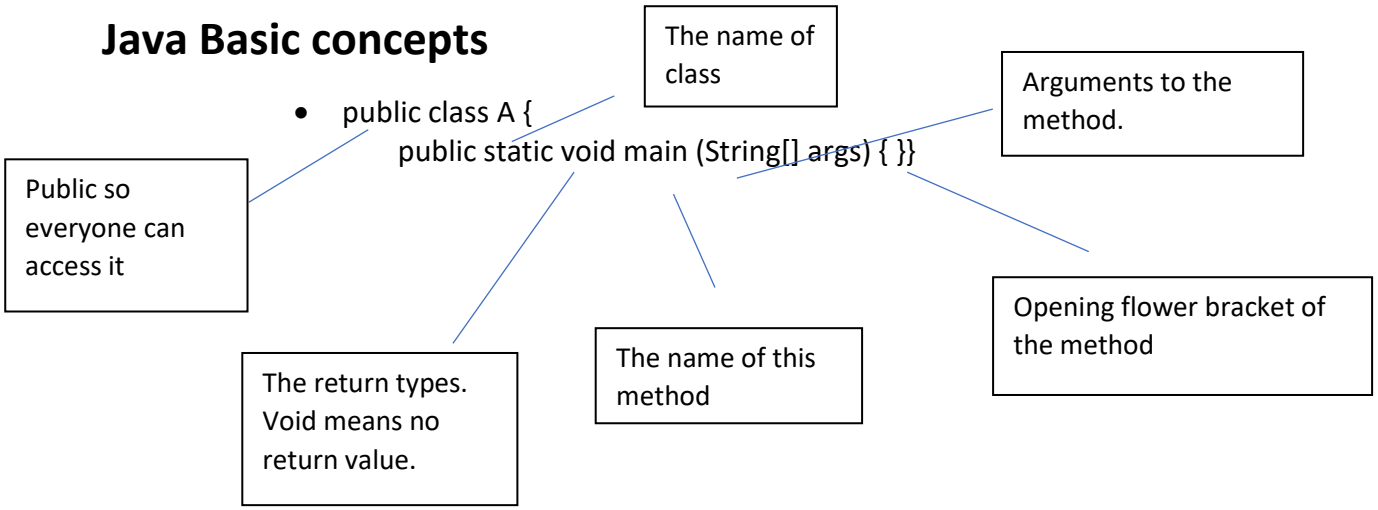
@Service - @Service help us to define service layer in our project.

JPA Annotation:

@Entity- @Entity help us to map the java class with database table.
@Table – When java class name and database table name are not same then we use it to map.
@Id- @Id maps the entity class variable with primary key column of database.
@GeneratedValue- @GeneratedValue helps us to auto-increment the value while storing the data in the database.
@Column- When the entity class variable name and the database column name are not same then this annotation help us to map it.

Pankaj Sir Academy

Java Basic concepts



- Class keyword – It generates object
- Object – Instance of class (child of class)
- { } – flower bracket symbols define boundary

NEWKEYWORD

- It sends request to the class so that it can create objects
- Once object is created **new keyword** get the object **address** and stores in **reference** variable
- Example

```
public class A {
        public static void main (String[] args) {
                A a1 = new A();
                System.out.println(a1);

                A a2 = new A();
                System.out.println(a2);
        }}
```

GARBAGECOLLECTORINJAVA

- Garbage collector helps us to manage memory in java efficiently and effectively by regularly removing unused object from memory and thus avoiding memory overflow.
- In other word, Garbage collector identifying which objects are **in use** and which are **not**, and **deleting** the unused objects.
- An in-use object, or reference object means that some of program still maintain a pointer to that object.

STATICANDNONSTATIC

- Example
  - ```
    public class B {
        int x = 10; // non static
        static int y = 100; //static
        public static void main(String[] args){
            B b1 = new B();
            System.out.println(b1.x); // non static
            System.out.println(B.y); //static

        }}
    ```

NONSTATIC

- Non static variables are created outside method inside class without static keyword
- Only after creating object, we can access non static variable
- In the below program variable x is access, without creating object and hence we get error.
  - Error

    ```
    public class B {
        int x = 10; // non static
        public static void main(String[] args){
        System.out.println(x); //Error
        }}
    ```

  - Correct program
    ```

```
public class B {
    int x = 10; // non static

public static void main(String[] args){

    B b1 = new B();
    System.out.println(b1.x); //10

    }}
```

- Non static variable is also called as instance variable.
- Non static main original value will not change only xerox copy will change as shown in below example
  - ```
    public class A {
        int x = 10;
        public static void main(String[] args){
            A a1 = new A();
            a1.x = 1000;
            System.out.println(a1.x);
            A a2 = new A();
            System.out.println(a2.x);
            }}
    ```

STATIC

- Static variables are created inside class outside method, using static keyword.
- These variables belong to class.
- Do not create object to access static variables.
- Example one
  - ```
    public class A {
        static int x = 10;
        public static void main(String[] args){
        System.out.println(A.x);
    }}
    ```
  - Static variable value can be changed as shown in the below example
    - ```
      public class A {
          static int x = 10;
          public static void main(String[] args){
          System.out.println(A.x); // 10
      ```
      ```

```
A.x = 1000;
System.out.println(A.x); //1000
}}
```

METHOD

- non static
    - ```
      public class A {
          public static void main(String[] args)
      {
          A a1 = new A();
          a1.test();
          }
          public void test() {
          System.out.println(100);
          }}
      ```

- static
    - ```
      public class A {
          public static void main(String[] args){
              A.test();

          }

          public static void test() {
              System.out.println(100);
          }}
      ```

STACKANDHEAP

- Program execution flow in maintain in stack
- Object are created in heap memory
- In stack, where methods invocations and variables live
- In heap, where all objects live

VARIABLES

LOCALVARIABLE

- These variables are created inside method and should be used only within created method, if used outside created method than, it will give you error as shown in the below example
  - ```
    public class A {
    public static void main (String[] args) {
            int id = 10; // Local variable
            System.out.println(id);
            A a1 = new A ();
            a1.test ();
       }
    public void test() {
            System.out.println(id); // Error (cannot used local
                                                variable without static
            }}
    ```

  - ```
    public class A {
            public static void main (String [] args) {
                    int id = 10;
                    System.out.println(id);
                    A a1 = new A ();
                    a1.test ();
                    }
            public void test () {
                    int x = 10;
                    System.out.println(x);
                    }}
    ```

- Without initialized local variable if used, then we get an error.
  - ```
    public class A {
            public static void main (String[] args) {
                    int x; // not initialized
                    System.out.println(x); // Error
    ```
    ```

STATICVARIABLE

- It created inside class, but outside method using static keyword.
- These variables have global access, which means it can be used anywhere in the program.
- These variables can be excess in three ways
    - A.x (Class Name A)
    - X (Direct variable name)
    - A1.x (Wrong way, but not give error)

- Example
    - ```
      public class A {
            static int id = 100;
            public static void main(String[] args) {
                  System.out.println(A.id);
                  A a1 = new A();
            }}
      ```

    - ```
      public class B {
            static int x1 = 10; // static global variable
            public static void main(String[] args) {
                  int x2 = 10; // local variable
                  System.out.println(x2);
                  System.out.println(B.x1);
            }
            public void test() {
                  System.out.println(x2); //Error because local variable
                  System.out.println(B.x1); // global variable
            }}
      ```

- Static **int** default value is 0.
- It is not mandatory to initialized. If we do not initialize then, depending on the data type auto initialization will take place.
    - ```
      public class A {
            static int id; //0
            public static void main (String[] args) {
                  System.out.println(A.id);}}
      ```
      ```

NONSTATICVARIABLE

- Non static variables are created inside class, but outside method without static keyword.
- These variables are also called as instance variables.
- Only after creating object this variable can be exist.
- It is not mandatory to initialized not static variables.
- If it does not initialize than depending on data type default value get store into it.
- Example

```
public class A {
	int id;
	public static void main(String[] args) {
		A a1 = new A();
		System.out.println(a1.id);
	}}
```

  - Example default value of int is Zero
  - Local variable name and non-static variable name can be same.
  - Static variable and local variable names can be same as show in below example.
    -
```
public class A {
		int id; // id name
		public static void main(String[] args) {
			int id = 10; // id name
			A a1 = new A();
			System.out.println(a1.id);
			System.out.println(id);
		}}
```

- Local variable names can be same provided they are created inside different method.
  -
```
public class A {
		int id = 10;
		public static void main(String[] args) {
		int id = 10;
		System.out.println(id); //10
		test();
	}
	public static void test() {
```

```
        int id = 100;
        System.out.println(id); //100
    }}
```

- Local reference variables are created inside the method and should be used only within created method. If used outside created method it will give you an error as shown in below example.
  - ```
    public class A {
        public static void main(String[] args){
            A a1 = new A();
            System.out.println(a1);
            a1.test();
            }
            public void test() {
                System.out.println(a1); //Error
            }}
    ```

- Static reference methods are created outside the method but inside class using static keyword. This variable has global exist and can be used anywhere in the program
- If we do not initialize static variable, then by default null value will be store into it.
  - ```
    public class B{
        static B b1; //  Null
        public static void main(String[] args) {
            System.out.println(b1);
        }}
    ```
  - ```
    public class C{
        static C c1;
        public static void main(String[] args) {
            c1 = new C();
            System.out.println(c1);
            c1.test();
        }
        public void test() {
            System.out.println(c1);
        }}
    ```
    ```

VARTYPE

- In version 10 of java var type is introduce.
- When a variable is created with the type var, then it means you can store any kind of value in it.
- Var type allocate the data type to the variable dynamically, depending on the value to store in the data type.
  - ```
    public class A{
        public static void main(String[] args) {
            var x1 = 10;
            var x2 = 10.3;
            var x3 = true;
            var x4  =  "Prateek";
            var x5 = new A();
            System.out.println(x1);
            System.out.println(x2);
            System.out.println(x3);
            System.out.println(x4);
            System.out.println(x5);

        }}
    ```

- A variable with the type var cannot be static and non-static, it can only be local variable
  - ```
    public class B {
        static var x1; // Error
        var x2; // Error
        public static void main(String[] args) {
            var x3; //Correct
        }}
    ```

- Method argument cannot be of the type var, the below program shows error.
  - ```
    public class C {
        public static void main(String[] args) {
            C a1 = new C();
            a1.test(100);
        }}
        public void test (var x){ //Method argument
        (Local Variable)
    ```
    ```

```
        System.out.println(x);
    }}
```

- Var is not a keyword in java and hence variable name in java can be var as shown in below example.
  - ```
    public class A{
        public static void main(String[] args) {
            int new = 10; // new is a keyword
            (Error)
            var var = 10; // variable name can be
            var (Correct)
            System.out.println(var);
    }}
    ```

METHODS

VOID

- If a method is Void, then it means it cannot return any value, and hence the below program through error
  - ```
    public class A{
            public static void main(String[] args){
            }
            public void test() {
                return 100; //Error void not return
                            any value
            }}
    ```

METHODRETURNTYPE

- ```
  public class A{
      public static void main(String[] args) {
          A a1 = new A();
          int x = a1.test();// x is local
          variable
          System.out.println(x);
      }
      public int test() {
          int x = 10;
          return x; }}
  ```
  ```

- ```
  public class A{
      public static void main(String[] args) {
          A a1 = new A();
          int x = a1.test();
          System.out.println(x);}
      public int test() {
          int x = 10;
          return x;
      }
      public void test1() {
          A a2 = new A();
          a2.test2();
      }
      public void test2() {
          System.out.println(100);
      }}
  ```

RETURN

- It should be used only inside void methods.
- It is optional to be used.
- It returns control back to calling statement
- Example
  - ```
    public class A {
        public static void main(String[] args) {
            A a1 = new A();
            a1.test();
            System.out.println();
        }
        public void test() {
            System.out.println(100);
            return; // optional to used return
            System.out.println(500); //Un reachable
            code Error Never Run
        }}
    ```

RETURNVALUE

- It should be used only inside not a void method
- It is mandatory to be used inside not a void method
- Its return control and value back to method calling statement.

- Example
  - ```
    public class B {
        public static void main(String[] args) {
        B b1 = new B();
        int val = b1.test();
        System.out.println(val);}
    public int test() {
        return 1;
    }}
    ```

  - ```
    public class C {
        public static void main(String[] args) {
            C c1 = new C();
            c1.test(10,10.3f,'a',"Prateek",true);
        }
        public void test(int x1, float x2, char x3,
        String x4, boolean x5) {
            System.out.println(x1);
            System.out.println(x2);
            System.out.println(x3);
            System.out.println(x4);
            System.out.println(x5);
        }}
    ```

  - ```
    public class C {
        public static void main(String[] args) {
            C c1 = new C();
            c1.test(10,20,30,40,50);
        }
        public void test(int... x){ // int… x it can
                                       hold value
            System.out.println(x[0]);
            System.out.println(x[1]);
            System.out.println(x[2]);
            System.out.println(x[3]);
            System.out.println(x[4]);}
    ```

CONSTRUCTOR

- A constructor should have same name as that of class
- Whenever it creates object constructor will be called automatically
    ```

- Constructors are permanently void

```
public class A {
	A(){
		System.out.println(100);
	}
	public static void main(String[] args) {
		A a1 = new A();
		A a2 = new A();
	}}
```

- Output is 100 and 100
- Because constructors are void and cannot return any value
  -
```
public class A {
	A(){
		System.out.println(100);
		return 100; // Error
	}
```

- Method helps us to break code into smaller reusable module.
- Creating constructor with arguments
  -
```
public class A {
	A(int x, char y){
	System.out.println(x);
	System.out.println(y);
	}
		public static void main(String[] args) {
		A a1 = new A(1000, 'a');
	}
```

CONSTRUCTOROVERLOADING

- We create more than one constructor in same class provided they have different number of arguments or different type of arguments

I.
```
public class A {
		A(){
			System.out.println(1);
		}
		A(int x){
			System.out.println(x);
		}
```

```
A(int y, int x){
     System.out.println(y);
     System.out.println(x);
}
public static void main(String[] args) {
     A a1 = new A();
     A a2 = new A(100);
     A a3 = new A(1000,2000);
}}
```

II.
```
public class A {
     A(int x){ // args = 1, Type = int
          System.out.println(x);
     }
     A(char x){ // args = 1, type = char
          System.out.println(x);
     }
     A(float x){ //args = 1.0, type = float
          System.out.println(x);
     }
     public static void main(String[] args) {
          A a1 = new A(100);
          A a2 = new A('a');
          A a3 = new A(10.3f);
     }}
```

- If you are a developer than install jdk, because that gives us a compiler and run time environment. but if you are customer that install only jre which can only run .Class files.

DEFAULTCONSTRUCTOR

- When we do not create a constructor than during compilation in dot class file automatically no args empty body constructor are created and this constructor is called as default constructor.
- Constructor automatically get created (default constructor)
  - ```
    public class A {
         A(){ // automatically created A(){ }
         }
    ```
    ```

```
    public static void main(String[] args) {
        A a1 = new A();
}}
```

- It is applicable only, for an object that is created without argument.
- When an object a value is created than as a programmer explicitly, we have to create matching constructor in the class. Or else we get an error as show in the below example.
    - ```
      public class D {
          public static void main(String[] args) {
          D d1 = new D(100); //Error because object
                             with argument is there.
                             Not created constructor
          }}
      ```

NEWKEYWORD

- It sends request to the class to create object
- It should mandatorily call constructor
- It will get object address and stores that in reference variable

THISKEYWORD

- this keyword is a special reference variable that automatically gets created
    - ```
      public class A {
          public static void main(String[] args) {
              A a1 = new A();
              System.out.println(a1);
              a1.test();
          }
          public void test() {
              System.out.println(this);
          }
      ```

- This keyword point to the current object running in the program.
    - ```
      public class A {
          public static void main(String[] args) {
      ```
      ```

```
        A a1 = new A();
        System.out.println(a1);
        a1.test();

        A a2 = new A();
        System.out.println(a2);
        a2.test();

        A a3 = new A();
        System.out.println(a3);
        a3.test();
    }
    public void test() {
        System.out.println(this);
    }}
```

- Using **this** key word, we can access non static variable
  - ```
    public class A {
        int x = 10; //non static variable
        public static void main(String[] args) {
            A a1 = new A(); // a1 is local variable
            to main
            System.out.println(a1.x);
            a1.test();
        }
        public void test() {
            System.out.println(this.x); //access
            non static variable
        }}
    ```

- **this** keyword another example
  - ```
    public class A {
        public static void main(String[] args) {
            A a1 = new A(); // a1 is local variable
            to main
            a1.test1();
        }
        public void test1() {
            this.test2();
        }
        public void test2() {
            System.out.println(100); }}
    ```

LIMITATIONOFTHISKEYWORD

- **this** keyword cannot be used inside static method and hence below program shows error
  - ```
    public class A {
        public static void main(String[] args) {
            A a1 = new A(); // a1 is local variable
            to main
            System.out.println(this); //Error
        }
        Public static void test() {
            System.out.println(this); //Error
        }}
    ```

- Using this key word to can access static members but we should never do that
  - ```
    public class A {
        static int x = 10;
        public static void main(String[] args) {
            A a1 = new A();
            a1.test();
        }
        public void test() {
            System.out.println(this.x);
        }}
    ```

- The below program we are not writing this,x in test method. If we do not write then compiler than automatically converts x to this.x and this is applicated only non-static member.
  - ```
    public class A {

        int x= 10;
        public static void main(String[] args) {
            A a1 = new A();
            a1.test();}
        public void test() {
            System.out.println(x); // this.x
        }}
    ```
    ```

THISMETHOD

- It is used to call a constructor and this calling should happen only from another constructor and always should be first statement.
  - ```
    public class A {

        A(){

        System.out.println(10);
        }
        A(int x){
        System.out.println(2000);
        this(); // It is used to call a constructor
             //and this calling should happen only
             from another constructor,
             //and always should be first statement
        System.out.println(1000);
        }
        public static void main(String[] args) {
             A a1 = new A(100);

        }
        public void test() {
             this(); //cannot use to call a
                       constructor inside method.
             System.out.println();
        }}
    ```
  - ```
    public class A {
    A(){
         this (100);
    }
    A(int x){
         System.out.println(x);
    }
    public static void main(String[] args) {
         A a1 = new A();
    }
    ```
- When we call one constructor from another constructor than it forms a flow of execution that forms chain like structure.
  - ```
    public class A {
    ```
    ```

```
A(int x, int y){
System.out.println(100);
}
A(int x){
     this (10,20);
}
A(){
     this(100);
}
public static void main(String[] args) {
     A a1 = new A();
}}
```

- ```
  public class B {
       int x; // non static
       public static void main(String[] args) {
            B b1 = new B();
            b1.test();
       }
       public void test() {
            int x = 10; // local variable
            this.x = x; // access not static
            System.out.println(this.x);
       }}
  ```

- ```
  public class B {
       int  x;
       B(int x){
            this.x = x;
       }
       public static void main(String[] args) {
            B b1 = new B(100);
            System.out.println(b1.x);
       }
       public void test() {
       }
  ```

INSTANCEINITIALISATIONBLOCK (IIB)

- IIB are executed when objects are created
- Number of times we create an object same number of times IIB will be called
- IIB are used to initialise all the instance variable in one and that gives us better readability of the code
- Example when object is not created, no output
    - ```
      public class A {
      {
          System.out.println("From IIB");
      }
      public static void main(String[] args) {
          // Object is not created
      })
      ```

- Example when object is created and output is From IIB
    - ```
      public class A {
      {
          System.out.println("From IIB");
      }
      public static void main(String[] args) {
          A a1 = new A();
      }}
      ```

- Always when object is created, IIB execute first because first initialise the variable than loaded to the object.
    - ```
      public class A {
          {
              System.out.println("From IIB");
          }
          A(){
              System.out.println("From
                                  Constructor");
          }
          public static void main(String[] args)
          {
              A a1 = new A();
          }}
      ```
      ```

- Output is first executed Start Main, then, From IIB, From Constructor and End Main.

```
public class A {
        {
            System.out.println("From IIB");
        }
        A(){
            System.out.println("From
                                Constructor");
        }
        public static void main(String[] args)
        {
            System.out.println("Start Main");
            A a1 = new A();
            System.out.println("End Main");
        }}
```

- When object is created, IIB is used practically, to initialise the variable

```
public class A {
        int i;
        {
            i = 20;
            System.out.println(i);
        }
    public static void main(String[] args) {
            A a1 = new A();
        }}
```

- We can initialize both static and non-static variables inside IIB

STATICINITILISATIONBLOCK (SIB)

- SIB is created with static keyword with the brackets.
- SIB do not need to create object to run.
- SIB runs before main method and it does not require invoking statement.
- Example

```
public class A {
static
{
```

```
        System.out.println("From SIB");
    }
    public static void main(String[] args) {

    }}
```

- ```
  public class A {
      static int i;
      static
      {
          i = 20;
          System.out.println(i);
      }
      public static void main(String[] args)
      {
          System.out.println("From Main");
      }}
  ```

- SIB is created inside the class
- SIB runs only once.
- When ever there is SIB in sequence, the first one is going to run first and then the next one.
  - ```
    public class A {
        static {
            System.out.println("From SIB -1");
        }
        static {
            System.out.println("From SIB -2");
        }
        public static void main(String[] args)
        {
            System.out.println("From Main");
        }
    ```

- We cannot initialize non-static variables inside SIB.

INHERITANCE

- It is applicable only on non-static member
- It is applicable on objects
- Here we inherit from parent class to child class with intension to reusing it.
- Example
  - ```
    public class A {
        int x = 10;
        public static void main(String[] args) {
            A a1 = new A();
        }}

    public class B extends A {
        public static void main(String[] args) {
            B b1 = new B();
            System.out.println(b1.x);
        }}
    ```
  - ```
    public class A {
        int x = 10;
        public void test() {
            System.out.println(100);
        }
            }
    public class B extends A {
        public static void main(String[] args) {
            B b1 = new B();
            System.out.println(b1.x);
            b1.test();
        }}
    ```
  - ```
    public class A {
        public void test1() {
            System.out.println(1000);
        }}

    public class B extends A {
        // test1, test2
        public void test2() {
            System.out.println(100);
    ```
    ```

```
}}

public class C extends B {
    // test1, test2, test3
    public void test3() {
        System.out.println(10);
    }
    public static void main(String[] args) {
        C c1 = new C();
        c1.test1();
        c1.test2();
        c1.test3();
}}
```

- Java does not support multiple inheritance at class level, but multiple inheritance can be performed at interfaces level

PACKAGES

- Package are nothing but folders created in java to store program in organized manner
- If you want to access a class present in different package than importing of that class is important.
- Examples
  - Importing a package class with another package class

```
package p1;
    public class A {
}

package p1;
    public class B extends A {

}

package p2;
import p1.A;
    public class C extends A {

}
```

- Non -sub classes, there is no link between them, these are individual Classes.

```
package p1;
     public class A {
          public int x = 10;
     }

package p1;

     public class B {
          public static void main(String[] args){
               A a1 = new A(); // non-sub class
               System.out.println(a1.x);
          }}

package p2;
import p1.A;
     public class C{
          public static void main(String[] args){
               A a1 = new A();
               System.out.println(a1.x);
          }}
```

- You cannot give package name in capital letter or keywords.

```
package p1.p2.p3; //class A will be created at
                        last folder
     public class A {
     }

package p4;
import p1.p2.p3.A;
     public class B {
          public static void main(String[] args){
               A a1 = new A();
          }}
```

- Star means that all classes are importing in one class.

```
     package p1;
          public class A {

     }
```

```
package p1;
     public class B {

}

package p2;
import p1.*;
     public class C {
          public static void main(String[] args){
               A a1 = new A();
               B b1 = new B();
          }}
```

- Another way to access without star symbol

```
package p1;
     public class A {

     }

package p1;
     public class B {

     }

package p2;
     public class C {
          public static void main(String[] args){
               p1.A a1 = new p1.A();
               p1.B b1 = new p1.B()
     }}
```

- If package names are different, we can create same class names.

```
package p1;
     public class A{
     public A() {
               System.out.println("P1");
          }
     }
package p1;
     public class A{
          public A() {
               System.out.println("P1");
```

```
        }
    }
package p3;
    public class B {
        public static void main(String[] args)
    {
            p1.A a1 = new p1.A();
            p2.A a2 = new p2.A();
        }
    }
```

- While you creating a constructor, if you using void, it will become a method. Both examples to given in below.
  - it's a Method Example

```
package p2;
    public class A {
        void A() { // It a method, no longer a
                   constructor
        System.out.println("P2");
        }
    public static void main(String[] args) {
         A a1 = new A();
         a1.A(); // method is call, print P2
        }}
```

  - It's a Constructor example

```
package p2;
public class A {
    A() { // It a constructor
        System.out.println("P2");
    }
    public static void main(String[] args) {
        A a1 = new A(); // constructor is
                        call, print P2
    }}
```

ACCESSSPECIFIER

It helps to restrict the visibility of constructors,methods,variables,class and package and Other data members

Private, default, protected and public

	private	default	protected	public
In same class	**Yes**	**Yes**	**Yes**	**Yes**
In same package sub class	**No**	**Yes**	**Yes**	**Yes**
In same package non sub class	**No**	**Yes**	**Yes**	**Yes**
Different package sub class	**No**	**No**	**Yes**	**Yes**
Different package non sub class	**No**	**No**	**No**	**Yes**

Private

- If you make variable/ method private then it can be accessed only in same class.

Default

- If you make a variable/method default then it can be accessed only in package but not in different package.

Protected

- If you make a variable/method protected then it can be accessed only in package and in different package only through inheritance

Public

- If you make a variable/method public than it can be accessed in same package and in different package as well

CLASS

- A class can be public or default, but it cannot be private or protected
- If a class is public then it can be accessed in same package and in different package as well
- If you make a class default, then it can be accessed only in same package

Private constructor

- If you make constructor private then its object can be created in same class only but not in different class, different package.

Default constructor

- If you make a default constructor then its object can be created in same class and in same package but not in different package

Protected constructor

- If you make a constructor protected then it object can be created in same class and in same package but not in different package.

Public

- If you make a constructor public then its object can be created in same class and same package and in different package as well

SUPERKEWORD

- Using super keyword, we can access the members of parent class.
- Example
  - ```
    public class A {
            int i =10;
        }
        public class B extends A {
            public static void main(String[] args) {
                B b1 = new B();
                b1.test();
            }
            public void test() {
                System.out.println(super.i);
            }}
    ```

- Using super keyword, we can access static and non-static member both.
  - ```
    public class A {
        static int j=10;
        }
        public class B extends A {
            public static void main(String[] args) {
                B b1 = new B();
                b1.test();
            }
        public void test() {
                System.out.println(super.j);
            }}
    ```

- Super keyword cannot be used inside static methods or static context.
  - ```
    public class A {
        static int j=10;
                }
        public class B extends A {
            public static void main(String[] args) {
                B b1 = new B();
                b1.test();
            }
        public static void test() {
            System.out.println(super.j); //Error
            }}
    ```
    ```

- Using super keyword, we can call constructor of parent class, but then we should use super keyword in child class constructor and it should be very first statement.
  - ```
    public class A {
        A(){
            System.out.println("A");
        }}
    public class B extends A {
        B(){
            super();
        }
        public static void main(String[] args) {
        B b1 = new B();
            }}
    ```

- Only When calling a parent class constructor, it should be first statement.
  - ```
    public class A {
        A(){
            System.out.println("A");
        }}
    Public class B extends A {
        B(){
            System.out.println("B");
                super(); //Error because of second \
                         statement.
            }
        public static void main(String[] args) {
            B b1 = new B();
            }}
    ```

- If we don't keep super keyword inside child class constructor, then compiler will automatically place the supper keyword such that it can call only no arcs constructor of parent class.
  In other words, Whenever we created a constructor without anything in it, the compiler automatically keep super() keyword.
  - ```
    public class A {
        A(){
            System.out.println("A");
            }}
    ```
    ```

```
public class B extends A {
    B(){
       // Super keyword created by compiler.
       }
    public static void main(String[] args) {
         B b1 = new B();
         }}
```

- If you do not create child class constructor without argument, then compiler will automatically place no arcs constructor with super keyword.
  - ```
    public class A {
        A(){
             System.out.println("A");
        }}
    public class B extends A {
        B(){
           System.out.println("B");
           }
        public static void main(String[] args) {
             B b1 = new B();
             }}
    ```

- Super keyword is kept in every child class constructor with or without argument
  - ```
    public class A {
        A(){
             System.out.println("A");
          }}
    public class B extends A {
        B(int i){
             System.out.println(i);
            }
        public static void main(String[] args) {
             B b1 = new B(100);
           }}
    ```

- If in a parent class there is only constructor with arguments then as a programmer, we should explicitly write super keyword in child class constructor.

- ```
  public class A {
      A(int i){
          System.out.println(i);
          }}
  public class B extends A {
      B(){
          super(100); // explicitly to write super
                        keyword
          System.out.println("B");
       }
  public static void main(String[] args) {
          B b1 = new B();
        }}
  ```

- Super keyword is not automatically kept when there are only constructor with argument in parent class, whereas supper keyword will be placed automatically when in the parent class there is a constructor with no argument.
  - ```
    public class A {
        A(int i){
            System.out.println(i);
        }
        A(){
            System.out.println("From Test");
        }}
    public class B extends A {
        B(){

            System.out.println("B");
        }
        public static void main(String[] args) {
            new B();
        }}
    ```

- Every child constructor will have super keyword in the beginning, that automatically added, but that super keyword call only constructor without argument

  - ```
    public class A {
        A(int i){
            System.out.println(i);
    ```
    ```

```
        }
        A(){
            System.out.println("From Test");
        }}
public class B extends A {
        B(){
            System.out.println("B");
        }
        B(int i){
            System.out.println(i);
        }
        public static void main(String[] args) {
            new B();
            new B(100);}}
```

.

- We cannot use **this** keyword and **super** keyword in the same constructor to call another constructor as either of the statement a become second statement then we will get an error.
  - ```
    public class A {
        A(int i){
            System.out.println(i);
        }
        A(){
            System.out.println("From Test");
        }}
    public class B extends A {
        B(){
            System.out.println("B");
        }
        B(int i){
            this(); super(); // Error
            System.out.println(i);
        }
        public static void main(String[] args) {
            new B(100);
        }}
    ```

- If your child class constructor consist of this keyword then in that constructor super keyword will be not automatically placed.
  - ```
    public class A {

        A(int i){
    ```

```
            System.out.println(i);
        }
        A(){
            System.out.println("From Test");
        }}
    public class B extends A {
        B(){
            System.out.println("B");
        }
        B(int i){
            this();
            System.out.println(i);
        }
        public static void main(String[] args) {
            new B(100);
        }}
```

POLYMORPHISM

- If we develop a feature such that it can take more than one form depending on the situation.
- There are two types overriding and overloading

Feature of Polymorphism

- Overriding
- Overloading

METHODOVERRIDING

- Overriding of variable cannot be done, overriding is applicable only on methods
- Example - There we Inherited method in child class, then we modify logical of inherit a method in the child class, by once again creating a feature in same signature.

```
▪ public class A {
        public void test() {
            System.out.println(100);
```

```
    }}
public class B extends A{
    public static void main(String[] args) {
        B b1 = new B();
        b1.test(); // Inheritance
    }
    public void test() {
        System.out.println(500); // Overriding
    }}
```

- ```
  public class A {
      public void test1() {
          System.out.println(100);
      }
      public void test2() {
          System.out.println(200);
      }}
  public class B extends A{
      public void test2() {
          System.out.println(500);
      }
      public static void main(String[] args) {
          B b1 = new B();
          b1.test1();
          b1.test2();
      }}
  ```

- ```
  public class A {
      public void test() {
          System.out.println(100);
      }}
  public class B extends A{
      @Override
      public void tests() {
          System.out.println(500);
      }}
  ```

- @Override annotation was introduced in version 5 of java. This annotation helps us to check weather overriding in happing or not
- @SupressWarnings will helps us to suppress warning message in the programme (CTRL + 1)

- ```
  public class A {
      public static void main(String[] args) {
          @SuppressWarnings("unused")
          int x = 10;
      }}
  ```

- When @SupressWarnings is applied to a method then it supresses unused variable warning of complete methods
  - ```
    public class A {
        public static void main(String[] args) {
            @SuppressWarnings("unused")
            int x = 10;
            int y = 20;
        }}
    ```

- Deprecated
  - ```
    public class A {
        public static void main(String[] args) {
            A a1 = new A();
            a1.protective(); //out dated feature but it
                               will work
            a1.destructive(); // latest feature
            }
        @Deprecated
        public void protective() {
            System.out.println("Helps People");
        }
        public void destructive() {
            System.out.println("Cause Problem");
        }}
    ```

METHODOVERLOADING

- If we created more than one method with the same name is the same class provided, they have different number of arguments or different type of arguments
  - ```
    public class Email {
        public void sendEmail() {
            System.out.println("Send Email");
    ```

```
    }
    public void sendEmail(String attachment) {
        System.out.println("Send Email with
                                    Attachment");
    }
    public static void main(String[] args) {
        Email e = new Email();
        e.sendEmail();
        e.sendEmail("D:\test.png");
    }}
```

- We can create more than one main method
  - ```
    public class A {
        public static void main(String[] args) {
            System.out.println(100);
            A.main(args);
        }
        public static void main() {
            System.out.println(200);
        }}
    ```

INTERFACE

- Always incomplete
- An interface cannot consist of complete method, and hence the below program through error.
  - ```
    public interface A {
        public void test() { //Error
            }
        }
    ```
  - ```
    package p1;

    public interface A {
        public void test();
        }
    package p1;

    public class B implements A{
        @Override
        public void test() {
            System.out.println(100);
    ```
    ```

```
}
public static void main(String[] args) {
     B b1 = new B();
     b1.test();
}}
```

- To achieve good designing
- When abstract key applied on method, it means method is incomplete
- In interfaces to define incomplete methods, it is not mandatory to used abstract keyword, its optional.
  - ```
    public interface A {
        public abstract void test1(); //optional
        public void test2();
      }
    ```

INTERFACES

- In java interfaces support multiply inheritance
- A class can implement more than one interface, as shown in the below example.
  - ```
    public interface A {
        }
    public interface B{

        }
    public class C implements A,B {

        }
    ```

- We can use extends and implements together but ensure that extends use first and then implements
  - ```
    public interface A {
        }
    public class B{

        }
    public class C extends B implements A {
        }
    ```
    ```

Covered so far

- Class to class----- extends
- Interface to class ------- implements
- Interface to interface------ extend

FINALKEYWORD

- If you make a variable final then changing its value is not allowed
  - ```
    public class A {
        public static void main(String[] args) {
            final int x = 10;
            x = 100; // Error, cannot change x
                     value because of final keyword.
            System.out.println(x);
    }}
    ```

- If you make static / non-static variable final then initialization is mandatory. Or else you will get an error as shown in below example
  - ```
    public class A {
        final int x;
        final static int y;
        public static void main(String[] args) {

        }}
    ```

- If you make method final then overriding of that method is not allowed
  - ```
    public class A {
        final public void test() {

        }
    public class B extends A{
        @Override
        public void test(){     //Error
        }}
    ```

- If you make class as final then inheritance is not allowed
  - ```
    public class A {

        }
    ```

```
public class B extends A{

}
```

Continuation of interface

- Every variable in an interface by default is static and final, for example.
  - ```
    public interface A {
        int id = 10;
        final static String city = "Bangalore";
    }
    ```

- An object for an interface cannot be created
  - ```
    public interface A {
        int id = 10;
        final static String city = "Bangalore";
        }
    public class B{
        public static void main(String[] args) {
            System.out.println(A.id);
            System.out.println(A.city);
            A a1 = new A(); //Error cannot create
                                objects
        }}
    ```

- An object of interface cannot be created but the reference variable can be created.
  - ```
    public interface A {
        int id = 10;
        final static String city = "Bangalore";
        }
    public class B{
        public static void main(String[] args) {
            new A();
            A a1;
            }}
    ```
    ```

CANWEOVERRIDESTATICMETHOD

- Static member in java is never inherit. In the below example B.x get converted to A.x and B.test() get converted to A.x and hence it print to output 10 and 1000
  - ```
    public class A {
            static int x = 10;
            public static void test(){
                System.out.println(1000);
                }}
    public class B extends A{
            public static void main(String[] args) {
                System.out.println(B.x); // A.x
                B.test();
                }}
    ```

- Static methods are never inheritance and hence overriding of that is never possible, hence the below example through error. (For overriding, we need to do inheritance)
  - ```
    public class A {
            public static void test(){
                System.out.println(100);
                }}
    public class B extends A{
            @Override
            public static void test() { //Error because
                                        static method
                System.out.println(500);
            }}
    ```

CANWECREATESTATICMETHODININTERFACE

- Static incomplete methods in an interface cannot be created because overriding of that is not allowed.
  - ```
    public interface A {
            public static void test(); //Error because
                                    overriding is not allowed
        }
    public class B implements A{

        }
    ```
    ```

MARKERINTERFACE

- Empty interfaces created in java are called as marker interface.
  - ```
    public interface A{
        //Empty
    }
    public class B implements A {
        //Empty
    }
    ```

JAVA8FEATURES

FUNCTIONALINTERFACE

- A functional Interface should consist of exactly one incomplete- abstract method in it
  - ```
    @FunctionalInterface
    public interface A {
        public void test(); //Incomplete method
    }

    public class B implements A{
        @Override
        public void test() {
            System.out.println(100);
        }
        public static void main(String[] args) {
            B b1 = new B();
            b1.test();
        }}
    ```

- The below program throws an error because Functional Interface cannot have zero abstract method
  - ```
    @FunctionalInterface
    public interface A {
        }
    ```

- A Functional Interface cannot have more than one abstract method and hence the below program throws an error
  - ```
    @FunctionalInterface
    public interface A { //Error
    ```

```
        public void test1();
        public void test2();
}
```

DEFAULTKEYWORD

- Was introduced in version 8 of java
- Using default keyword, we can create complete method in an interface.
- Example
  - ```
    public interface A {
        default void test1() {
            System.out.println(100);
        }
        default void test2() {
            System.out.println(200);
        }}
     public class B implements A{
        public static void main(String[] args) {
            B b1 = new B();
            b1.test1(); //100
            b1.test2(); //200
        }}
    ```

CANWEDOMULTIPLEINHERITANCEFORFUNCTIONALINTERFACE

- Yes, if the parent interfaces are functional and child one only a interface then we can do multiple inheritance for functional interface.
  - ```
    @FunctionalInterface
    public interface A {
        public void test1();
        public void test2();
    }}
    @FunctionalInterface
    public interface B{
        public void test3();
    }
    @FunctionalInterface
    public interface C extends A {

    }
    ```

- A function interface should consist of exactly one incomplete method but can consist of any number of complete methods, as shown in the below example
  - ```
    @FunctionalInterface
    public interface A {
        default void test1() { //complete method
            System.out.println(100);
    }
        default void test2() {
            System.out.println(200);
        }
        public void test3(); //incomplete
    }
    public class B implements A {
        public static void main(String[] args) {

        }
        @Override
        public void test3() {
            }}
    ```

LAMDAEXPRESSION

- It reduces number of lines of code during development
- Till version 7 of java, you had to create an object using new keyword, but in version 8 the object is getting created automatically with lambda expressions. Because of this feature java is also called as functional programming language.
- Non static member able to access without creating object with the help of lambda expression.
- Examples
  - ```
    @FunctionalInterface
    public interface A {
        public void test11();
    }
    public interface B {
        public static void main(String[] args) {
            A a1 = ()-> { // -> lambda reduce line of
                                code
    ```

```
                System.out.println(100);
            };
            a1.test11(); // 100
}}
```

- ```
  @FunctionalInterface
  public interface A {
      public void test11(int x);
  }

  public interface B {
      public static void main(String[] args) {
          A a1 = (int x) ->{
              System.out.println(x);
          };
          a1.test11(100);
  }}
  ```

- ```
  @FunctionalInterface
  public interface A {
      public void test11(int x);
  }

  public interface B {
      public static void main(String[] args) {
      A a1 = (int x, int y)->{
          System.out.println(x);
          System.out.println(y);
      };
      a1.test11(10,20);
      }}
  ```

ABSTRACT

- An abstract class can consist of both complete and incomplete methods in it
- To define incomplete method in an abstract class it is mandatory to use abstract keyword
- You can create a main method in an abstract class.
- For example

```
abstract class A {
    public void test1() {
        //complete
    }
    public abstract void test2(); //incomplete
    public void test3(); //Error because abstract is
                         missing
    public static void main (String[] args) {
    }
```

- An object of abstract class cannot be created, but the reference variable of abstract class can be created.
- Example
  - ```
    A a1 = new A(); // cannot created abstract class
    A a1; // can be created reference variable
    ```
- We can create static and non-static variable in an abstract class
  - ```
    abstract class A {
        static int x1 = 10;
        int x2 = 20;
        }
    ```

- Abstract class does not support multiply inheritance.
  - ```
    public abstract class A {
    }
    public abstract class B {

    }
    public abstract class C extends A,B{
        // multiple inheritance not allowed
    }
    ```

ABSTRACTION

- Abstraction means incomplete.
- In abstraction we hide implementation details and way we achieve in java is using interfaces and abstract class
    ```

Java 8

- As per version 8 of java an interface can consist of main method as well as complete static method.

DIFFERENCEBETWEENINTERFACESANDABSTRACT

Interfaces	Abstract
• 100 % incomplete	• 0 % to 100 % incomplete
• Supports multiple inheritance	• Does not support multiple inheritance
• All variable by default a final and static	• Variable can be static / non-static

DATAHIDING

- There we make a variable private so that variable cannot be access outside the class.
  - ```
    public class A {
        private int x = 10;
    }
    public class B extends A{
        public static void main(String[] args) {
            B b1 = new B();
            System.out.println(b1.x); // cannot access
                                      b1.x because of private
        }}
    ```
    ```

ENCAPSULATION

- Encapsulation refers to bunding of data with the methods that operator on that data, or restricting direct access to the variable.
- Here we make a variable private, to restrict its direct access
- Publicly define getters and setters' methods are created to access these variables indirectly.

```
public class A {

private int id;//shortcut (ctrl+1) to getters and
                        setters

public void setId(int id) {
     this.id = id;
}
public int getId() {
     return this.id;
}}

public class B{
     public static void main(String[] args) {
          A a1 = new A();
          a1.setId(100);
          System.out.println(a1.getId());
     }}
```

TYPECASTING

- Variable Type Casting
- Class Casting

Data Type	Bits
• **Byte**	• 1 byte
• **Short**	• 2 bytes
• **Int**	• 4 bytes
• **long**	• 8 bytes
• **float**	• 4 bytes
• **double**	• 8 bytes
• **char**	• 2 bytes
• **boolean**	• true/false

VARIABLETYPECASTING

- Here we convert particular data type into required data type
- **AUTOTYPECASTING**
  - ❖ Converting smaller data type to bigger data type is called as auto upcasting.
  - ❖ During auto up casting data lose should not happen.
  - ❖ Examples
    - ```
      public class A {
      public static void main(String[] args) {
            int x = 10; // 4 bytes
            long y = x; // 8 bytes
            System.out.println(y);

      }}
      ```
    - ```
      public class A {
      public static void main(String[] args) {
            float f = 10.3f; //4 bytes
            double d = f; // 8 bytes
            System.out.println(d);
      }}
      ```

  - ❖ In the below example there is a data loss of 0.3 value auto upcasting will not happen.
    - ```
      public class A {
            public static void main(String[] args) {
                  float f = 10.3f; //4 bytes
                  long d = f; // 8 bytes
                  System.out.println(d);
            }}
      ```

  - ❖ Upcasting get extra
    - ```
      public class A {
            public static void main(String[] args) {
                  long x = 10;
                  float y = x; // 10.0 up casting
                  System.out.println(y);
            }}
      ```

- **EXPLICITTYPECASTING**
  - ❖ Here we convert bigger data type to smaller data type
  - ❖ During explicit down casting data lose might happen.
    - ○
```
public class A {
    public static void main(String[] args) {
        long x = 10; // 8 bytes
        int y = (int)x; // explicitly
        System.out.println(y); // 10

    }}
```
    - ○
```
public class A {
    public static void main(String[] args) {
        double d = 10.3;
        float f = (float) d;
        System.out.println(f);
    }}
```
    - ○
```
public class A {
    public static void main(String[] args) {
        float f = 10.3f;
        long l = (long) f; // 0.3 data loss
        System.out.println(l);
    }}
```
    - ○
```
public class A {
    public static void main(String[] args) {
        int x = 1232323;
        byte y = (byte) x;
        System.out.println(y); //-61
}}
```
  - ❖ Converting character to integer in auto upcasting but converting integer to character is explicit down casting.
    - ▪
```
public class A {
    public static void main(String[] args) {
        char x = 'a';// up casting (auto
        casting)
        int y = x; // up casting (auto
        casting)

        int a = 97; // down casting
        (explicit)
        char b = (char) a;//down casting
                                (explicit)
```

```
            System.out.println(y);
        }}
```

CLASSCASTING

- **Class up casting**
  - Here we created parent reference variable and then we store child class objects address in it.
    - ```
      public class A {
          public static void main(String[] args) {
              A a1 = new B();
              System.out.println(a1);
              a1 = new C();
              System.out.println(a1);
          }}
      public class B extends A {

          }

      public class C extends A {

          }
      ```

- **Class down casting**
  - Example
    - ```
      public class A {
          public static void main(String[] args) {
              B b1 = new A(); //Exception
              C c1 = new A(); //Exception
          }}
      public class B extends A {

          }

      public class C extends A {

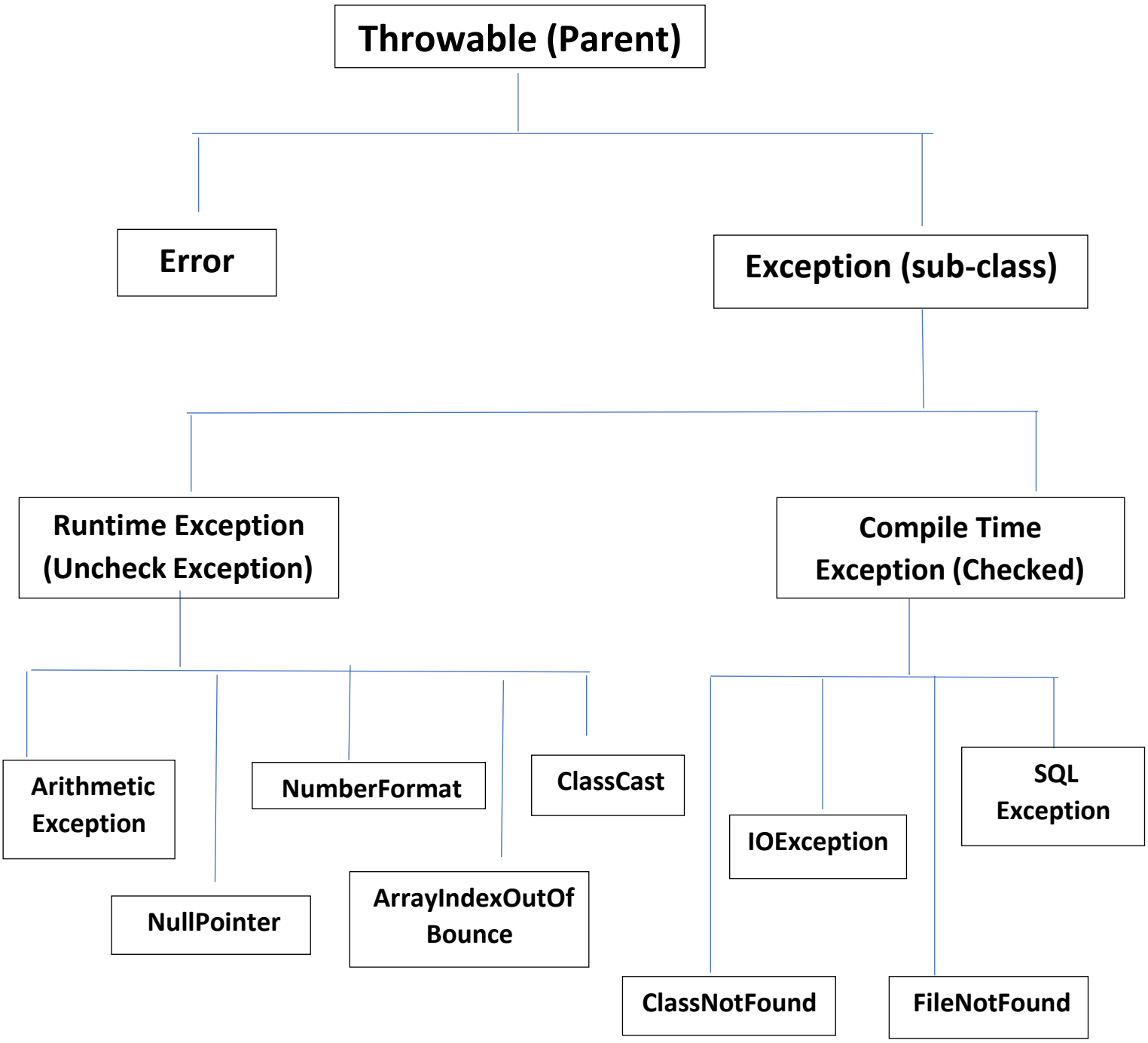
          }
      ```

EXCEPTIONHANDLING

- Whenever a bad user input in given then program execution will halt abruptly as shown in below example
  - ```
    public class A {
    ```
    ```

```
public static void main(String[] args) {
     int x = 10;
     int y = 0;
     int z = x/y; //Exception happen not execute.
     System.out.println(z);
     System.out.println("Welcome");
     System.out.println("Welcome");
     System.out.println("Welcome");
}}
```

- To handle exception we use try, catch block in java, if any line of code in try block causes exception, then try block will create exception object, and this object address is then given to catch block, then try block gives an object address to catch block. Catch block not suppresses the exception and further code will continue to flow as shown in example.
  - ```
    public class B {
        public static void main(String[] args) {
            try {
                int x = 10;
                int y = 0;
                int z = x/y;
                System.out.println(z);
            }
            catch (Exception e) {
                System.out.println(e);
            }
            System.out.println("Welcome");
            System.out.println("Welcome");
            System.out.println("Welcome");
            }}
    ```
    ```



Exception class hierarchy

- The super most class in java in object class

CHECKEDEXCEPTION

- Compile time/ checked exception happens to occur when dot java file in converted to dot class file.

UNCHECKEDEXCEPTION

- Run time exception occur when we are running dot class file

RUNTIMEEXCEPTION/UNCHECKED

- Whenever an exception occurs, from that line control directly will go to catch block and the catch block will supresses and further executed the program further.
- **AIRTHMATICEXCEPTION**
  - ✓ Whenever invalid mathematical operation is performed, we get ArithmeticException and ArithmeticException class handle the particular exception.
    - ```
      public class A {
      public static void main(String[] args) {
           try {
                int x=10;
                int y = 0;
                int z = x/y;
                System.out.println(z);
                System.out.println(100);
           } catch (ArithmeticException e) {
           System.out.println(e);
           e.printStackTrace(); // it will bring
                                the exactly exception
                                line no, where
                                exception occurs.
           }
           System.out.println("Welcome");
      }}
      ```

- **NUMBERFORMATEXCEPTION**
  - ✓ In valid string to number converser in done then it gives NumberFormat Exception
      ```

- ✓ NumberFormat Exception can only handle NumberFormat, it cannot handle other exception
- ✓ In the below example when try to convert to an integer it give an error.

```
public class B {
public static void main(String[] args) {
     try {
          String x = "10abcd"; // alpha
                                        numeric
          int y = Integer.parseInt(x);
          System.out.println(y);

     } catch (NumberFormatException e) {
          e.printStackTrace();
     }

     System.out.println(100);
}}
```

- **NULLPOINTEREXCEPTION**
  - ✓ With the Null reference variable, when you access non-static members, it gives NullPointerException

```
public class A {
static A a1; // null
int x = 10; //non-static
public static void main(String[] args) {
     try {
          System.out.println(a1.x);
                    //nullPointerException
     } catch (NullPointerException e) {
          e.printStackTrace();
     }
     System.out.println(100);
}}
```

- **ARRAYOUTOFBOUNDEXCEPTION**
  - ✓ An array can store multiple values.
  - ✓ If array out of store value, then error occurs

What is the difference between try catch ArithmeticException and try catch Exception?

- If you write try catch ArithmeticException, it can handle only ArithmeticException, but not other Exception. If you write Exception class it can handle

ARRAY

- Arrays stores collection of values in it.
- In java when we create an array, a special object gets created
- Array Index always starts with 0
- Find the size of an array, we used length attribute
- Example
  - ```
    public class A {
        public static void main(String[] args) {
            int[] x = new int[3];
            System.out.println(x);
            System.out.println(x.length); //length is
                                          non-static variable
            x[0] = 10;
            x[1] = 20;
            x[2] = 30;
            System.out.println(x[0]);
            System.out.println(x[1]);
            System.out.println(x[2]);
    }}
    ```

FORLOOP

- Example
  - ```
    public class A {
        public static void main(String[] args) {
            for (int i = 0; i<5; i++){
                System.out.println(i);
            }}
    Output – 0, 1, 2 ,3 ,4
    ```
  - ```
    public class A {
        public static void main(String[] args) {
            for (int i = 5; i>0; i--){
    ```
    ```

```
                System.out.println(i);
            }}
Output – 5, 4, 3, 2, 1
```

BREAKSTATEMENT

- Break statement only used inside loop and switch statement
- Break keyword should use on within a loop or switch statement
- In the below example break keyword helps us to exit for loop
  - ```
    public class A {
        public static void main(String[] args) {
            for (;;) { // infinite loop
                System.out.println(100);
                break; // it exit for loop
            }
        }}
    ```

- When i 3 it will break the for loop
  - ```
    public class A {
        public static void main(String[] args) {
            for (int i=0;i<5;i++) { // infinite loop
                if (i==3) {
                    System.out.println(i); //break at 3
                    break;
                }}
        }}
    ```

CONTINUEKEYWORD

- Continue keyword will send you back to for loop
  - ```
    public class A {
        public static void main(String[] args) {
            for (int i=0;i<5;i++) {
                if (i==3) {
                    continue;
                }
                System.out.println(i); // 0, 1, 2, 4
            }
        }}
    ```
    ```

LABELLEDBREAKKEYWORD

- We can apply on the block you want, for example us can apply on if, for any block you want.
- **Specific block**
  - ```
    public class A {
        public static void main(String[] args) {
            for (int i=0;i<5;i++) {
                x: if (i==3) {
                    break x;
                }
                System.out.println(i); // 0,1,2,3,4
            }
    }}
    ```

ARRAYS

- Array iteration where we printing array by using for loop
- Example
  - ```
    public class A {
        public static void main(String[] args) {
            int[] x = new int[3];
            x[0] = 10;
            x[1] = 20;
            x[2] = 30;
            for (int i = 0; i<x.length; i++) {
                System.out.println(x[i]);
            }
    }}
    ```

FOREACH

- As long as all the value is not copy it will not stop foreach loop
  - ```
    public class A {
        public static void main(String[] args) {
            int[] x = new int[3];
            x[0] = 10;
            x[1] = 20;
            x[2] = 30;
            for (int i:x) {
                System.out.println(i); //10, 20, 30
    ```
    ```

```
        }
    }}
```

- ```
  public class A {
      public static void main(String[] args) {
          int[] x = new int[4];
          x[0] = 10;
          x[1] = 20;
          x[2] = 30;
          for (int i:x) {
              System.out.println(i); //10, 20, 30 and
                                     default int value=0
          }
      }}
  ```

Assignment

- String concept - 9 lecture

WHILELOOP

- Example
  - ```
    public class A {
        public static void main(String[] args) {
            int x = 0;
            while (x<3) {
                System.out.println(x);
                x++;
            }
        }}
    ```

DOWHILE

- Example
  - ```
    public class A {
        public static void main(String[] args) {
            int x = 0;
            do {
                System.out.println(x);
                x++;
    ```
    ```

```
        }
        while (x<3);
    }}
```

SCANNERCLASS

- Input in java scanner class
  - ```
    import java.util.Scanner;
    public class A {
        public static void main(String[] args) {
            Scanner scan = new Scanner(System.in);
            System.out.print("Enter Your Name: ");
            String name = scan.next(); //Only one word
            System.out.println("Your Name is " + name);
            System.out.print("Enter Your Age: ");
            int age = scan.nextInt();
            System.out.println("Your Age is " + age);
        }}
    ```

- NextLine() for long writing
  - ```
    import java.util.Scanner;

    public class B {
        public static void main(String[] args) {
            Scanner scan = new Scanner(System.in);
            System.out.print("About YourSelf: ");
            String name = scan.nextLine();
            System.out.println(name);
            System.out.print("Enter Your Age: ");
                int age = scan.nextInt();
        }}
    ```

Example using for loop, number of iterations in a program

- ```
  import java.util.Scanner;
  public class A {
  public static void main(String[] args) {
      try {
      Scanner scan = new Scanner(System.in);
  ```
  ```

```
            for (int i=0; i<3; i++) {
            System.out.print("Enter PIN Code: ");
            int pinNum = scan.nextInt();
            if (pinNum == 1234) {
                 System.out.println("Welcome");
                 break;
            }
            else {
                 System.out.println("Invalid Pin Code");
                 if (i==2) {
                 System.out.println("Card is Blocked");
                      break;
                 }
            }
       }
       } catch (Exception e) {
            System.out.println("Invalid Input");
       }}
```

While program example

- ```
  import java.util.Scanner;


  public class B {
  public static void main(String[] args) {
       try {
            String cdn = "yes";
            Scanner scan = new Scanner(System.in);

            while (cdn.equals("yes")) {//equals compare
                                       the Strings value in
                                       this case, it is yes.
                 System.out.print("Enter the Amount: ");
                 int amount = scan.nextInt();
                 System.out.println("Please Collect Rs.
                                          "+ amount);

                 System.out.println("Do you want to
                                       continue yes or no");
                 cdn = scan.next();
  ```
  ```

```
                if (cdn.equals("no")) {
                    System.out.println("Thank You.");
                }
            }

        } catch (Exception e) {
            System.out.println("Invalid Input");

        }
    }}
```

REMOVEDUPLICATEELEMENTS

- ```
  public class A {
      public static void main(String[] args) {

      int x [] = {1,2,2,3,4,5,5,6,7,7,7,8};
      int temp [] = new int[x.length];
      int j = 0;
      for (int i=0; i<x.length-1; i++) {
          if (x[i] != x[i+1]) {
              temp[j] = x[i];
              j++;
          }
      }
      temp[j] = x[x.length-1]; //last no will print
      for (int m: temp) {
          System.out.print(m);
          }
      }
  }
  ```

How to exchange number values x and y example

- ```
  public class B {
      public static void main(String[] args) {
          int x = 10;
          int y = 20;
          int temp = x; // temp = 10
          x = y; // x = 20
  ```

```
            y = temp; // y = 10
            System.out.println("x: " + x); // x = 20
            System.out.println("y: " + y); // y = 10
        }
    }
```

BUBBLESORT

- ```
  public class A {
      public static void main(String[] args) {
          int x [] = {38,23,32,14,7};
          int temp;
          for (int i = 0; i<x.length-1; i++) {
              for (int j=0; j<x.length-1; j++) {
                  if (x[j] > x[j+1]) {
                      temp = x[j];
                      x[j] = x[j+1];
                      x[j+1] = temp;
                  }
              }
          }
          for (int i : x) {
              System.out.print(i + " ");
          }
      }
  }
  ```

2DARRAY

- Example
  - ```
    public class A {
        public static void main(String[] args) {
            int[][] x = new int[3][4]; //12
            System.out.println(x.length); //3
            System.out.println(x[0].length); //4
        }}
    ```
  - ```
    public class A {
        public static void main(String[] args) {
    ```
    ```

```
        int[][] x = new int[3][3]; //9
        x[0][0] = 10;
        x[0][1] = 20;
        x[0][2] = 30;
        x[1][0] = 40;
        x[1][1] = 50;
        x[1][2] = 60;
        x[2][0] = 70;
        x[2][1] = 80;
        x[2][2] = 90;
        for (int i=0; i<x.length;i++) {
            for(int j=0;j< x[0].length; j++ ) {
                System.out.println(x[i][j]);
            }
        }
}}
```

Compile time Exception or Checked Exception

- Depend on File handing dependence

FILEHANDLING

FILEEXIST

- Exist is a non-static method present in file class
- It will check whether the file exist in given path, if yes it returns boolean value true else false.
- Example – **delete** is a non-static method present in file class whose return value is boolean value, if the file exist it will delete and return true, but if not, there is given path, then it cannot delete and hence it returns false.

```
import java.io.File;
    public class A {
        public static void main(String[] args) {
            File f = new
File("D:\\Notes\\PSA\\test1.txt");
            System.out.println(f);

            boolean val = f.exists(); //if exists
                                        true
            System.out.println(val);
```

```
            boolean del = f.delete(); //if is not
                                      there false, is
                                      there true
            System.out.println(del);
        }}
```

CREATEFILE

- **Create** is a non-static method present in file class. If the file does not exist in the given path, then we create a new file and return boolean value true.
- If the file already existed in the given path then it will not override, create a new file hence it return false
  - ```
    public class A {
        public static void main(String[] args) {
            File f = new
                     File("D:\\Notes\\PSA\\test1.txt");
            try {
                boolean val = f.createNewFile();
                System.out.println(val);

            } catch (Exception e) {
                e.printStackTrace();
            }

        }}
    ```

CREATEFOLDERANDDELETE

- Folder – directory, if it creates folder, then return boolean value true, if it already exists then return false
- Delete – if folder delete then it returns true, if it already deleted then return false
  - ```
    import java.io.File;

        public class A {
            public static void main(String[] args) {
                File f = new
                         File("D:\\Notes\\PSA\\folder1);
                boolean val = f.mkdir();
    ```

```
        System.out.println(val);

        boolean del = f.delete();
        System.out.println(del);
    }}
```

FILELENGTH

-
```
import java.io.File;

    public class A {
    public static void main(String[] args) {
        File f = new
                File("D:\\Notes\\PSA\\test1.txt");
        long val = f.length(); //counts character
                               with space calculated
        System.out.println(val);
    }}
```

FILEHANDLINGTOGETALLARRAYNAME

- List Method –Gives all the names present in the given path, return type string array.

-
```
import java.io.File;

    public class A {
        public static void main(String[] args) {
            File f = new File("D:\\Notes\\PSA\\");
            String [] str = f.list();

            for (String s: str) {
                System.out.println(s);
            }
            System.out.println(str.length); //size
                                        of an array
        }}
```

FILEREADER

- In the below example we read the text1.txt file content.
  - ```
    import java.io.File;
    import java.io.FileReader;

    public class A {
        public static void main(String[] args) {
            try {
                File f = new
                    File("D:\\Notes\\PSA\\test1.txt");
                FileReader fr = new FileReader(f);
                for (int i = 0; i<f.length(); i++) {
                    System.out.print((char)fr.read());
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }}
    ```

- Another way to read content – we used array and type casting
  - ```
    import java.io.File;
    import java.io.FileReader;

    public class A {
        public static void main(String[] args) {
            try {
                File f = new
                    File("D:\\Notes\\PSA\\test1.txt");
                FileReader fr = new FileReader(f);
                char [] ch = new char[(int)f.length()];
                fr.read(ch);
                for (char c: ch) {
                    System.out.print(c);
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
    ```

WRITTER

- Program how to write in file

```
import java.io.File;
import java.io.FileWriter;

public class A {
     public static void main(String[] args) {
          try {
               FileWriter fw = new
                              FileWriter("D:\\Notes\\PS
                              A\\test1.txt",true);//not
                              override the file
               fw.write(100);
               fw.write("Prateek\n");
               fw.write("100");
               fw.close();

          } catch (Exception e) {
               e.printStackTrace();
          }
     }}
```

- Another way to write, write method

```
import java.io.File;
import java.io.FileWriter;

public class A {
     public static void main(String[] args) {
          try {
               FileWriter fw = new
FileWriter("D:\\Notes\\PSA\\test1.txt");
               char[]ch = {'x','y','z'};
               fw.write(ch);
               fw.close();

          } catch (Exception e) {
               e.printStackTrace();
          }
     }}
```

BUFFEREDREADER

- File reading Improve file reader
- Read the file content with the best performance
- It helps us improve file reading performance
- It has readLine method which can read the content from the file, line by line
- Example

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.nio.Buffer;

public class A {
    public static void main(String[] args) {
         try {
              FileReader fr = new
FileReader("D:\\Notes\\PSA\\test1.txt");
              BufferedReader br = new
                                  BufferedReader(fr);
              System.out.println(br.readLine());
              System.out.println(br.readLine());
              System.out.println(br.readLine());
         } catch (Exception e) {
              e.printStackTrace();
         }
    }}
```

BUFFEREDWRITTER

- Improve the file writer
- Write the file content with the best performance
- It improves file writing performance
- It has newline method, that help us to write the content in new line.
- Example

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.nio.Buffer;
```

```
 public class A {
     public static void main(String[] args) {
          try {
               FileWriter fw = new
FileWriter("D:\\Notes\\PSA\\test1.txt");
               BufferedWriter br = new
                                        BufferedWriter(fw);
               bw.write(100);
               bw.newLine();
               bw.write("testing");
               bw.newLine();

               char [] ch = {'a','q','f'};
               br.write(ch);
               bw.close();

          } catch (Exception e) {
               e.printStackTrace();
          }
     }}
```

SERIALIZATION

- In serialization we convert the object in 0 and 1 and store that into file system.

DESERIALIZATION

- In deserialization we read the file content, and form the object back.

Program Serialization

- ```
  import java.io.FileOutputStream;
  import java.io.ObjectOutputStream;

  public class B {
       public static void main(String[] args) {
            try {
  ```
  ```

```
            FileOutputStream fos = new
                            FileOutputStream("D:\\Not
                            es\\PSA\\file.ser");
            ObjectOutputStream oos = new
                             ObjectOutputStream(fos);
            A a1  = new A();
            oos.writeObject(a1);

            oos.close();
            fos.close();

        } catch (Exception e) {
            e.printStackTrace();
        }
}}
```

Deserialization

- Read the file content
- Form an object back

TRANSIENT

- During serialisation, if you want to skip any variable value to be saved then, make that variable with transient keyword.

STRINGPROGRAMS

Reverse a given String

- ```
  public class A {
  	public static void main(String[] args) {
  		String name = "Prateek Upreti";
  		for (int i = name.length()-1; i>=0; i--) {
  			System.out.print(name.charAt(i));
  		}
  	}}
  ```

Count of number of words in the given String

- ```
  public class A {
  	public static void main(String[] args) {
  		reverseString(" Prateek Upreti");
  	}
  	public static void reverseString(String name) {
  		String[] s = name.trim().split(" ");
  		System.out.println(s.length);
  		for (String x : s) {
  			System.out.println(x);
  		}
  	} }
  ```

- name.trim will remove all the white space

Opening and Closing bracket error program

- ```
  import java.util.Scanner;

  public class A {
  	public static void main(String[] args) {
  		openingClosingString();
  	}
  	public static void openingClosingString(){
  		Scanner scan = new Scanner(System.in);

  		System.out.print("Enter Opening and Closing
  ```
  ```

```
                                        Parathesis: ");
            String name = scan.next();
            int count1= 0;
            int count2 = 0;
            for (int i = 0; i<name.length(); i++) {
                if (name.charAt(i)=='(') {
                    count1++;
                }else if(name.charAt(i) == ')') {
                     count2++;
                }
            }
            if (count1 == count2) {
                System.out.println("No Error");
            }else {
                System.out.println("Error");
            }
        }
}
```

MUTABLE

- Mutable is something where in the class object property keep on changing

IMMUTABLE

- Immutable class once its object is created then it state cannot be alter

Steps to create Immutable class in java

- Create a final class
- Set the values of the properties using only constructors
- Make the properties as final
- Do not provide any setters for these properties.
- Example
  - ```
    final public class A {
        private final int age; //age cannot be altered
        private final String name; //name cannot be
                                                altered
        public A(int age, String name) {
            this.age = age;
            this.name = name;
        }
        public int getAge() { //getters
            return age;
        }
        public String getName() { //getters
            return name;
    ```
    ```

```
    }
    public static void main (String[] args) {
        A a1 = new A (26, "Prateek"); //once we
                                assigned age 26 and name,
                                then it became immutable
    }}
```

- String is Immutable (in this example we use new keyword)

```
final public class A {
    public static void main(String[] args) {
        String s1 = new String("xyz");
        String s2 = new String("xyz");
        System.out.println(s1==s2); //false because
                                object address is
                                different
        System.out.println(s1.equals(s2)); //equals
                                compares the values.
                                True.
    }}
```

- In this example we do not use new keyword to create an object

```
final public class A {
    public static void main(String[] args) {
        String s1 = "Prateek";
        String s2 = "Prateek";
        String s3 = "xyz";
        String s4 = "xyz";
        System.out.println(s1==s2); //true
        System.out.println(s3==s4); // true
        System.out.println(s1.equals(s2)); //true
    }}
```

  -
```
final public class A {
        public static void main(String[] args) {
            String s1 = new String("xyz"); // new
                                    object created

            String s2 = "xyz"; //new object created

            String s3 = "xyz"; // it can refer to
                                    s2 object because
```

```
        String s4 = new String("xyz"); // new
                                    object created
    }}
```

Indeed, it is an immutable String

- ```
  final public class A {
      public static void main(String[] args) {
          String s1 = "xyz";   //eligible to garbage
                                  collector
          String s1 = "abc"; //create new object store
                                  abc
          System.out.println(s1); // abc
      }}
  ```

STRINGCONSTANTFOOL

- Example
  - ```
    final public class A {
        public static void main(String[] args) {
            String s1 = "xyz"; //new object created
            String s2 = "xyz"; // String Constant pool
            String s3 = "abc"; // no existing abc
                               object, create new object

            String s4 = "abc" // Because of String
                               Constant pool, do not
                               create any new object
    }}
    ```

In build Method – (toLowerCase() and toUpperCase() and trim())

- **Example**
  - ```
    final public class A {
        public static void main(String[] args) {
            String str = "Prateek UpReTi";
            System.out.println(str.toLowerCase());
            System.out.println(str.toUpperCase());
    ```
    ```

```
}}
```

- Same Example with white space (trim())

```
final public class A {
    public static void main(String[] args) {
        String str = "       Prateek UpReTi";

    System.out.println(str.trim().toLowerCase());

    System.out.println(str.trim().toUpperCase());
    }}
```

In-build method – (startsWith() and endsWith() and length())

- Example of startWith() and endsWith()

```
final public class A {
    public static void main(String[] args) {
        String str = "Prateek UpReTi";
        System.out.println(str.startsWith("P"));
                                                //true
        System.out.println(str.endsWith("i"));
                                                //true
        System.out.println(str.startsWith("r"));
                                                //false
        System.out.println(str.endsWith("t"));
                                                //false
    }}
```

- Example of Length()

```
final public class A {
    public static void main(String[] args) {
        String str = "Prateek UpReTi";
        System.out.println(str.length()); // it
                            count space as well 14
    }}
```

VALUEOFMETHOD

- Value of method converts given type such as integer, long, float, double, boolean and char array to String.
- Example
  - ```
    final public class A {
        public static void main(String[] args) {
            int i = 20;
            float f = 20.21F;
            double d = 30000D;
            boolean b = true;
            char[] c = {'a','b','c'};
            String intStr = String.valueOf(i);
            String flootStr = String.valueOf(f);
            String doubleStr = String.valueOf(d);
            String booleanStr = String.valueOf(b);
            String charStr = String.valueOf(c);
            System.out.println(intStr);
            System.out.println(flootStr);
            System.out.println(doubleStr);
            System.out.println(booleanStr);
            System.out.println(charStr);
    }}
    ```

Questions

1. Write a program to find how many letters a and letter b are occurs in the given string **aabaaaababa**

2. Print this output

   1
   23
   456
   78910
   1112131415

3. Print this output

   111111 1
   111111 1111
   111111 111111
    ```

111111

4. Find duplicate element in a given array and remove it

Answers

1.
```
public class A {
    public static void main(String[] args) {
        String str = "aabaaaababa";
        int count1 = 0;
        int count2 = 0;
        for (int i = 0; i<str.length(); i++) {
            if (str.charAt(i)=='a') {
                count1++;
            }
            else if (str.charAt(i) == 'b') {
                count2++;
            }
        }
        System.out.println("A : "+ count1);
        System.out.println("B : "+ count2);
  }}
```

2.
```
public class A {
        public static void main(String[] args) {
            int number = 5;
            for (int i=0; i<number; i++) {
                for (int j=0; j<number; j++) {
                    if ((i==0 && j==0)) {
                        System.out.print("1");
                    }
                    else if(i==1 && j==0) {
                        System.out.print("23");
                    }
                    else if(i==2 && j==0) {
                        System.out.print("456");
                    }
                    else if(i==3 && j==0) {
                        System.out.print("78910");
                    }
                    else if(i==4 && j==0) {
```

```
                    System.out.print("1112131415");
                }
                else {
                    System.out.print(" ");
                }
            }
            System.out.println();
        }
    }
}
```

3.
```
public class A {
  public static void main(String[] args) {
      int number = 6;
      for (int i = 0; i<number; i++) {
          for (int j=0; j<number; j++) {
              if ((i==0 && j==2) || (i==1 && j==1) ||
   (i==1 && j==2) || (i==1 && j==3) || (i==1 && j==4))
              {
                  System.out.print("1");
              }
              else if (i==2 && j==0 || i==2 && j==1
      || i==2 && j==2 || i==2 && j==3 || i==2 && j==4||
      i==2 && j==5)
              {
                  System.out.print("1");
              }else {
                  System.out.print(" ");
                  }
          }
          System.out.println();
      }
  }}
```

4.
```
public class B {
      public static void main(String[] args) {
          int num [] = {1,1,2,3,4,5,6,6,7,8,9};
          int temp[] = new int[num.length];
          int j = 0;
```

```
        for (int i=0; i<num.length-1; i++) {
            if (num[i] != num[i+1]) {
                temp[j] = num[i];
                j++;
            }
        }
        temp[j] = num[num.length-1];
        for (int x : temp) {
            System.out.print(x);
        }}
```

THREADS

- Multi-tasking done at the program level is called as Threads.
- The main purpose of thread in to improve the performance of the application, by reducing execution time.

There are two ways we can build Threads

- ```
  public class A extends Thread { //thread has two
                                  methods run() and start()
  @Override
  public void run() { //user define thread
      for (int i=0; i<100000; i++) {
          System.out.println("Run");
      }
  }
  public static void main(String[] args){ //build in
                                                  thread
      for (int i=0; i<100000; i++) {
          System.out.println("Main");
      }
      A a1 = new A();
      a1.start(); //start inherit from thread
  }}
  ```

- ```
  public class A extends Thread{
      String name;
      A(String name){
          this.name = name;
  ```

```
        }
        @Override
        public void run() {
            for (int i=0; i<10000; i++) {
                System.out.println(this.name);
            }
        }}

    public class B {
        public static void main(String[] args) {
            A a1 = new A("xxx");
            A a2 = new A("yyy");
            A a3 = new A("zzz");
            a1.start();
            a2.start();
            a3.start();
        }}
```

RUNNABLE (Interface)

- ```
  public class A implements Runnable{ // runnable has
                                          only run()
  String name;
  A(String name){
      this.name = name;
  }
  @Override
  public void run() {
      for (int i=0; i<10000;i++) {
          System.out.println(this.name);
      }
  }
  public static void main(String[] args) { //main is
                                        also a thread,
                                        run itself
      A a1 = new A("XXX");
      Thread t1 = new Thread(a1); //second thread
      t1.start();

      A a2 = new A("YYY");
      Thread t2 = new Thread(a2); //third thread
  ```
  ```

```
        t2.start();

        for (int i=0; i<10000;i++) {
            System.out.println("Main");
        }
    }}
```

What are the drawbacks of thread?

- When two thread are operating on a common data in non-synchronized manner, it resulted data corruption.
- Example – Booking, train reservation, etc

THREADSYNCHRONIZED

- When two threads are operating on common data, the data might get corrupted because of multi-tasking.
- Two make the threads operates one after another, we used synchronized keyword, wherein the thread which has acquired the lock can only execute the code, whereas the other thread would be in wait status. Only when the first thread has released a lock and other thread will get the opportunity to acquire the lock and execute the block.
- Example – program on threads and synchronized
  - ```
    public class A {
            int balance = 0;
            public static void main(String[] args) {
                A a1 = new A();
                a1.account();
                System.out.println(a1.balance);
            }
            public void account() {
                Thread t1 = new Thread(new Runnable() {
                    @Override
                    public void run() {
                        add();
                    }
                });

                Thread t2 = new Thread(new Runnable() {
                    @Override
    ```
    ```

```
            public void run() {
                sub();}
        });

        t1.start();
        t2.start();

        try {
            t1.join();
            t2.join();
        } catch (Exception e) {
            System.out.println(e);
        }
    }

    public synchronized void add() {
        for (int i=0; i<10000; i++) {
            balance = balance + i;

        }
    }
    public synchronized void sub() {
        for (int i=0; i<10000; i++) {
            balance = balance - i;
        }
    }}
```

NOTIFYWAIT

- Example
  - ```
    public class A {

        public static void main(String[] args) {
            B b1 = new B();
            b1.start();

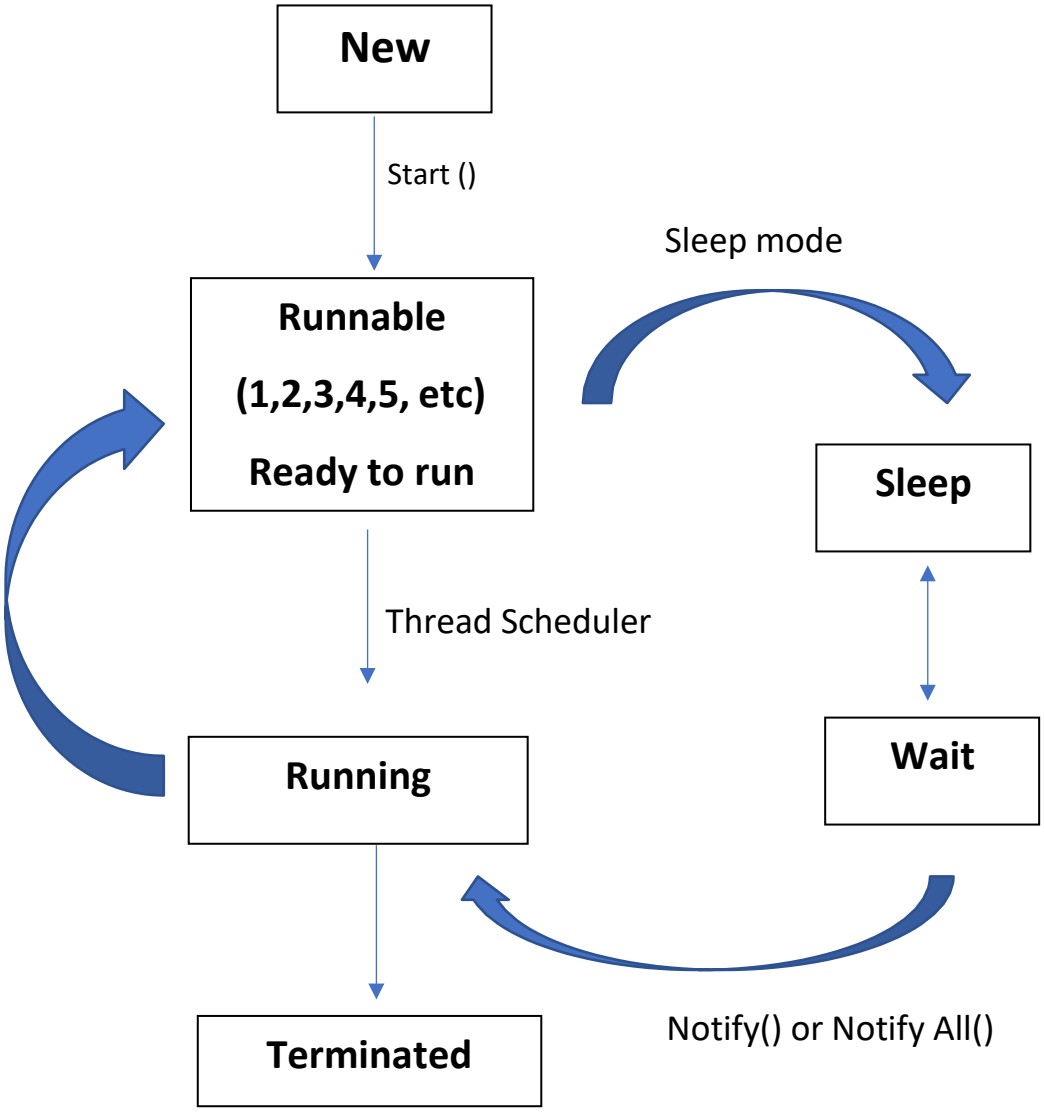
            synchronized (b1) {
                try {
                    b1.wait();
                } catch (Exception e) {
                    e.printStackTrace();}}
    ```
    ```

```
            System.out.println(b1.total);
        }}

public class B extends Thread{
        int total = 0;

        public synchronized void run(){
            for (int i=0; i<1000; i++) {
                total = total+i;
            }
            notify();
        }}
```

THREADLIFECYCLE



Thread Life Cycle in Program

- ```
  public class A extends Thread {
      public static void main(String[] args) {
          A a1 = new A();
          System.out.println(a1.getState()); //built-
                                             in getState();

          a1.start();

          try {
              Thread.sleep(5000);
          } catch (Exception e) {
              e.printStackTrace();
          }

          System.out.println(a1.getState());
      }
      public void run() {
          System.out.println("TEST");
      }
  }
  ```

  ```
  Output – NEW
           TEST
           TERMINATED
  ```

THREADPRIORITY

- It decides which thread is going to run first and which thread will run later.
- If we set the priority then, it is a request made to the thread schedule which there is no assurance that it will process or approve.
- The min thread priority is 1, and max thread priority is 10 and the normal thread priority is 5, however, we can set the thread priority which the number anything between 1 to 10
- Example
  - ```
    public class A extends Thread{
        String name;
        A(String name){
    ```

```
        this.name = name;
    }
    public static void main(String[] args) {
        A a1 = new A("xxxx");
        A a2 = new A("yyyy");
        a1.setPriority(1);
        a2.setPriority(10);
        System.out.println(a1.getPriority());
        System.out.println(a2.getPriority());
        a1.start();
        a2.start();

    }
    public void run() {
        System.out.println(this.name);
    }}
```

- How to give name to thread, setting a name and getting a name

```
public class A extends Thread{
    String name;
    A(String name){
        this.name = name;
    }
    public static void main(String[] args) {
        A a1 = new A("xxxx");
        A a2 = new A("yyyy");

        a1.setName("addAmount");
        a2.setName("withdrawAmount");


        System.out.println(a1.getName());
        System.out.println(a2.getName());
        a1.start();
        a2.start();

    }
    public void run() {
        System.out.println(this.name);
    }}
```

THREADPOOL

- Thread pool are useful when you need to limit the number of threads running in your application at the same time.
- It will help us to improve the performance of that application.
- Instant of starting a new thread for every task execute concurrently, the task can be passed to a thread pool. A thread pool contains collection of threads. As soon as the pool has ideal threads, the task is assigned to one of them and then executed.
- Thread pool are often used in services. Each connection arrives at the server wire the network is wrapped as a task and it is passed on thread pool.
- The threads in the thread pool will process the request on the connection concurrently. This is how we can used existing thread insert of creating a new thread, and there by improve the perform in terms of execution

ENUM

- Enum is collection of constants
  - Example 1

```
public enum Calender {
     Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct,
     Nov, Dec
     }
 public class A{
     public static void main(String[] args) {
          System.out.println(Calender.Oct);
     }}
```

  - Example 2

```
public enum Salutation {
     Mr, Mrs, Dr, Prof
     }
public class B {
     public static void main(String[] args) {
          System.out.println(Salutation.Mr);
     }}
```

WRAPPERCLASS

- Wrapper or boxing is a process of storing the value inside an object
- Reading that value of an object is called as unboxing or unwrapping.
- Example – Here are value are stored in object, and the process of storing the value inside an object is called as wrapping or boxing. Reading the value from an object is called as unboxing.
  - ```
    public class B {
        public static void main(String[] args) {
            Integer i = 10;
            System.out.println(Integer.MAX_VALUE);
            System.out.println(Integer.MIN_VALUE);
            System.out.println(i.longValue());
            System.out.println(i.hashCode());
            System.out.println(i.SIZE);
            System.out.println(i.toString());
            System.out.println(i.doubleValue());
            System.out.println(i.byteValue());
            }}
    ```

- The main advantage of wrapper class is easy manipulating of data
  - ```
    public class B {
        public static void main(String[] args) {
            Byte b = 10;
            Short s = 20;
            Integer i = 30;
            Long l = 40L;
            Float f = 10.3F;
            Double d = 10.3;
            Character c = 'a';
            Boolean o = true;
        }}
    ```

FINALIZEDMETHOD

- Finalized is a method present inside object class
- Garbage collection logic is implemented in finalized method
- Example
  - ```
    public class A extends Object{
    ```
    ```

```
protected void finalize() {
     System.out.println(1000);
}
public static void main(String[] args) {
     A a1 = new A();
     a1 = null;
     System.gc();
}}
```

WHYJAVAISNOT100%OBJECTORIENTED

- When I can write a program only by creating an object, then it is 100 percent Object Oriented.
- I can write a program in java without creating object as well, hence java is not 100 percent Object Oriented Language

OPTIONALCLASS

- Java 8 feature
- Stream api
- Logical question on collection
- Difference between collection and collections
- String buffer and string builder
- Single ton design pattern
- Volatile keyword
- Concurrent hash map
- Vectors
- Priority meet

THROWESKEYWORD

- It is applied on a method, if any exception occurs in the method then exception will be passed on with the calling statement of the method
- Example unhandled
  - ```
    import java.io.FileWriter;

    public class A {
        public static void main(String[] args) {
            A a1 = new A();
            a1.test(); //unhandled exception
        }
        public void test() throws Exception { //only
                                        applied on method

            FileWriter fw  =  new
             FileWriter("D:\\Notes\\PSA\\test1.txt");

        }}
    ```

- Exception being handled
  - ```
    import java.io.FileWriter;

    public class A {
        public static void main(String[] args) {
            A a1 = new A();
            try {
                a1.test();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        public void test() throws Exception { //only
                                        applied on method

            FileWriter fw  =  new
             FileWriter("D:\\Notes\\PSA\\test1.txt");
        }}
    ```

- More Accurate example
  - ```
    import java.io.FileWriter;
    ```
    ```

```
public class A {
    public static void main(String[] args) throws
                                            Exception {
        A a1 = new A();
        a1.test();
    }
    public void test() throws Exception { //only
                                    applied on method
        FileWriter fw  =  new
         FileWriter("D:\\Notes\\PSA\\test1.txt");
    }}
```

- After throws keyword we can get multiple class name
- Example
  - ```
    import java.io.FileWriter;
    import java.io.IOException;
    import java.sql.DriverManager;
    import java.sql.SQLException;

    public class C {
        public static void main(String[] args)throws
                          IOException, SQLException {
            FileWriter fw = new
                 FileWriter("D:\\Notes\\PSA\\test1/tx
                                                    t"
                                                  );
            DriverManager.getConnection("","","");
        }}
    ```

THROWKEYWORD

- It helps us to create customized exception as per the requirement of the developer
- Example
  - ```
    public class InsufficientFunds extends Throwable
    {

        }

    import java.util.Scanner;

    public class B {
        public static void main(String[] args) {
    ```

```
            int balance = 500;

            Scanner s = new Scanner(System.in);
            System.out.println("Enter theAmount: ");
            int amount = s.nextInt();
            if (balance > amount) {
                System.out.println("Please collect
                                        the cash");
            }else {
                try {
                    throw new InsufficientFunds();
                } catch (InsufficientFunds e) {
                    System.out.println(e);
                    System.out.println("Low
                                            Balance");
                }
            }
        }
    }
```

REGULAEXPRESSION

- Square Brackets **[ ]** will search for anything that you enter in it.
- Flower brackets **{ }** it will read input that you enter in it.
- Example
  - ```
    import java.util.regex.Matcher;
    import java.util.regex.Pattern;

    public class A {
        public static void main(String[] args) {

            Pattern p = Pattern.compile("[0-
                                        9]{2}");

            Matcher m = p.matcher("9632882052");

            while (m.find()) {

                System.out.println(m.start()+
                   "..." + m.group());
            }}}
    ```
    ```

```
Output  0...96
        2...32
        4...88
        6...20
        8...52 //form 5 group for 2 numbers
```

- Example finding a, b and c
  - ```
    import java.util.regex.Matcher;
    import java.util.regex.Pattern;

    public class A {
        public static void main(String[] args) {
            Pattern p =
                        Pattern.compile("[abc]");

            Matcher m = p.matcher("a6b#@z9cD");
            while (m.find()) {
                    System.out.println(m.start()+
                      "..." + m.group();
            }}}
    ```

Output - 0…a
2…b
7…c

- Example finding all the small letters
  - ```
    import java.util.regex.Matcher;
    import java.util.regex.Pattern;

    public class A {
        public static void main(String[] args) {
            Pattern p = Pattern.compile("[a-
                                        z]");
            Matcher m =
                  p.matcher("a6b#@z9cDfHsKsJf");

            while (m.find()) {

                System.out.println(m.start()+
            "..." + m.group());
            }}}
    ```

```
Output - 0...a
         2...b
         5...z
         7...c
         9...f
         11...s
         13...s
         15...f
```

- Example find all the number 0 to 9
  - ```
    import java.util.regex.Matcher;
    import java.util.regex.Pattern;

    public class A {
        public static void main(String[] args) {
            Pattern p = Pattern.compile("[0-
                                        9]");
            Matcher m =
                p.matcher("a6b#@z9cDfHsKs4Jf9");

            while (m.find()) {

                System.out.println(m.start()+
                   "..." + m.group());
            }}}
    ```

```
Output   1...6
         6...9
         14...4
         17...9
```

- Example all the letter and number except symbols
  - ```
    import java.util.regex.Matcher;
    import java.util.regex.Pattern;

    public class A {
        public static void main(String[] args) {
            Pattern p = Pattern.compile("[a-zA-
                                        Z0-9]");
            Matcher m =
                p.matcher("a6b#@z9cDfHsKs4Jf9");

            while (m.find()) {
    ```

```
                System.out.println(m.start()+
            "..." + m.group());
            }
        }}
```

- Example other than lower letter a to z **(^** it mean other than)
  - ```
    import java.util.regex.Matcher;
    import java.util.regex.Pattern;

    public class A {
        public static void main(String[] args) {
            Pattern p = Pattern.compile("[^a-
                                        z]");
            Matcher m =
                p.matcher("a6b#@z9cDfHsKs4Jf9");

            while (m.find()) {

                System.out.println(m.start()+
                 "..." + m.group());
            }}}
    ```

```
Output 1...6
       3...#
       4...@
       6...9
       8...D
       10...H
       12...K
       14...4
       15...J
       17...9
```

- Other then letters, number, means only special symbol will print.
  - ```
    import java.util.regex.Matcher;
    import java.util.regex.Pattern;

    public class A {
        public static void main(String[] args) {

            Pattern p = Pattern.compile("[^a-zA-
                                        Z0-9]");
    ```

```
        Matcher m =
            p.matcher("a6b#@z9cDfHsKs4Jf9");

        while (m.find()) {

            System.out.println(m.start()+
               "..." + m.group());
        }}}
```

```
Output 3...#
       4...@
```

\\s and \\S

- Lower \\s give you while space
  - ```
    import java.util.regex.Matcher;
    import java.util.regex.Pattern;

    public class B {
        public static void main(String[] args){
            Pattern p = Pattern.compile("\\s");

            Matcher m = p.matcher("a6b @#9DE!");

            while (m.find()) {

                System.out.println(m.start() +
                    "..." + m.group());

                }}}
    ```

    ```
    Output - 3...
    ```

- Upper \\S gives us everything except white space
  - ```
    import java.util.regex.Matcher;
    import java.util.regex.Pattern;

    public class B {
        public static void main(String[] args){
            Pattern p = Pattern.compile("\\S");

            Matcher m = p.matcher("a6b @#9DE!");
    ```

```
            while (m.find()) {

                System.out.println(m.start() +
                    "..." + m.group());

                }}}
```

```
Output - 0...a
         1...6
         2...b
         4...@
         5...#
         6...9
         7...D
         8...E
         9...!
```

\\d and \\D

- \\d will gives us digits
  - ```
    import java.util.regex.Matcher;
    import java.util.regex.Pattern;

    public class B {
        public static void main(String[] args){
            Pattern p = Pattern.compile("\\d");

            Matcher m = p.matcher("a6b @#9DE!");

            while (m.find()) {

                System.out.println(m.start() +
                    "..." + m.group());

                }}}
    ```

Output -

```
1...6
6...9
```

- \\D will gives us everything except digits
  - ```
    import java.util.regex.Matcher;
    ```

```
import java.util.regex.Pattern;

public class B {
    public static void main(String[] args){
        Pattern p = Pattern.compile("\\D");

        Matcher m = p.matcher("a6b @#9DE!");

        while (m.find()) {

            System.out.println(m.start() +
                "..." + m.group());

            }}}
```

```
Output  0...a
        2...b
        3...
        4...@
        5...#
        7...D
        8...E
        9...!
```

\\w and \\W

- \\w will gives us lower case letter and upper case letters, digits only
  - ```
    import java.util.regex.Matcher;
    import java.util.regex.Pattern;

    public class B {
        public static void main(String[] args){
            Pattern p = Pattern.compile("\\D");

            Matcher m = p.matcher("a6b @#9DE!");

            while (m.find()) {

                System.out.println(m.start() +
                    "..." + m.group());

                }}}
    ```
    ```

```
Output  0...a
        1...6
        2...b
        6...9
        7...D
        8...E
```

- \\W will gives us special symbol and white space only
  - ```
    import java.util.regex.Matcher;
    import java.util.regex.Pattern;

    public class B {
        public static void main(String[] args){
            Pattern p = Pattern.compile("\\W");

            Matcher m = p.matcher("a6b @#9DE!");

            while (m.find()) {

                System.out.println(m.start() +
                    "..." + m.group());

                }}}
    ```

```
Output  3...

        4...@
        5...#
        9...!
```

Regular Expression Logic program – A name validator program

- ```
  import java.util.Scanner;
  import java.util.regex.Matcher;
  import java.util.regex.Pattern;

  public class B {
      public static void main(String[] args){
          Scanner s = new Scanner(System.in);
          System.out.print("Enter You Name: ");
  ```

```
        String str = s.next();

        int count = 0;
        Pattern p = Pattern.compile("[^a-zA-
                                    Z]");

        Matcher m = p.matcher(str);

        while (m.find()) {
            count++;
        }
        if (count != 0 || str.length()<3) {
            System.out.println("Error");
        }else {
            System.out.println("Input
                                Accepted");
        }
    }}
```

Mobile number validation program using Regular Expression

- ```
  import java.util.Scanner;

  public class A {
      public static void main(String[] args) {
          Scanner s = new Scanner(System.in);
          System.out.println("Enter Mobile Number:
                                              ");
          String mobileNum = s.next();
          String regex = "[6-9][0-9]{9}";

          if (mobileNum.matches(regex)) {
              System.out.println("Valid Number");
          }else {
              System.out.println("Invalid
                                  Number");
          }}
  }
  ```
  ```

Regular Expression Char start *

- It gives us zero occurrence or group of occurrences of a particular character
- Example

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class D {
    public static void main(String[] args) {

        Pattern p = Pattern.compile("a*");

        Matcher m =
                p.matcher("aabaaabaabaaaab");

        while (m.find()) {

            System.out.println(m.start() +
                "..." + m.group());
        }
}}
```

Regular Expression Char plus +

- It gives us one occurrence of a or group of occurrence of a
- Example

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class D {
    public static void main(String[] args) {

        Pattern p = Pattern.compile("a+");

        Matcher m =
                p.matcher("aabaaabaabaaaab");

        while (m.find()) {

            System.out.println(m.start() +
                "..." + m.group());
        }
```

```
}}
```

Regular Expression char ?

- It will give single occurrence of character
- Example, give single occurrence of group a
  - ```
    import java.util.regex.Matcher;
    import java.util.regex.Pattern;

    public class D {
        public static void main(String[] args) {

            Pattern p = Pattern.compile("a?");

            Matcher m =
                    p.matcher("aabaaabaabaaaab");

            while (m.find()) {

                System.out.println(m.start() +
                    "..." + m.group());
            }
    }}
    ```

TOKONIZER

- It is a build in method, the major purpose of StringTokenizer to split the string
- Example
  - ```
    import java.util.StringTokenizer;

    public class D {
        public static void main(String[] args) {
            StringTokenizer  str = new
    StringTokenizer("Pankaj Sir Academy");
            int count = 0;
            while (str.hasMoreTokens()) {

    System.out.println(str.nextToken());
                count++;
            }
    ```

```
            System.out.println(count);
        }}
```

- Example date
  - ```
    import java.util.StringTokenizer;

    public class D {
        public static void main(String[] args) {
            StringTokenizer  str = new
              StringTokenizer("07-03-2022",”-“);

            int count = 0;
            while (str.hasMoreTokens()) {

                System.out.println(str.nextToken
                                          ());
                count++;
            }

            System.out.println(count);
        }}
    ```

Non

- A
    ```

CLONING

- The process of creating the replica of a particular object by coping the content of one object completely into another object
- Example
  - ```
    public class E implements Cloneable {
        public static void main(String[] args){
            E e1 = new E();
            try {
                E e2 = (E) e1.clone();
                System.out.println(e1);
                System.out.println(e2);

            } catch (CloneNotSupportedException
                                            e) {
                e.printStackTrace();
            }
    }}
    ```
- Hash code - It return integer representation of an object memory address
- Example
  - ```
    public class E implements Cloneable {
        public static void main(String[] args){
            E e1 = new E();
            System.out.println(e1.hashCode());
            }}
    ```

ANNOTATION

- Do something on your behalf
- Annotation starts with @
- @SupressWarning, @Override, @Deprecated are annotation

@SuppressWarning

- Example with warning
  ```
  public class F {
      public static void main(String[] args) {
  ```

```
        @SuppressWarnings ("unused")
        int i = 10;
        int j = 100; // warning because we are
                        not using it.
        int k = 300; // warning
    }}
```

- Example with no warning

```
public class F {
    @SuppressWarnings ("unused")
    public static void main(String[] args) {
        int i = 10;
        int j = 100;
        int k = 300;
    }}
```

JDBC (Java Database Connectivity)

- **Database**
  - It can store the data permanently and securely
  - Without database we could not develop a software at all
  - Database – MySQL, Oracle, Mango DB

MySQL

- Example – basic program

```
create database db_1

use db_1

create table registration (

  name varchar(45),

  city varchar(45),

  email varchar(128),

  mobile varchar (10),

)
```

```
select * from registration

insert into registration values ('Prateek', 'Bangalore',
'prateek@gmail.com', '1234567')

drop table registration       // delete the table
```

MySQL connection in java

- Example one
  - ```
    import java.sql.Connection;
    import java.sql.DriverManager;
    import java.sql.Statement;

    public class A {
        public static void main(String[] args) {
            try {

        Connection con =
            DriverManager.getConnection("jdbc:mysql:
            //localhost:3306/db_1","root","password"
            );
                System.out.println(con);
                conn.close();

            } catch (Exception e) {
                e.printStackTrace();
            }
        }}
    ```

MySQL CRUD

- Create
- Retrieve or Read
- Update
- Delete
- Example of CRUD

  - ```
    import java.sql.Connection;
    import java.sql.DriverManager;
    import java.sql.ResultSet;
    import java.sql.Statement;
    ```

```
public class A {
  public static void main(String[] args) {
       try {

            Connection con =
                      DriverManager.getConnection("jdb
                      c:mysql://localhost:3306/db_2","
                      root","password");

            System.out.println(con);

            Statement stmt = con.createStatement();

  //Insert Query or CREATE

            stmt.executeUpdate(" Insert into student
                 values('Rahul','Upreti','Aerodynamic'
                 ,'Texas') ");

            stmt.executeUpdate(" Insert into student
                 values('Prateek','Upreti','Full
                 Stack','Bangalore') ");

            stmt.executeUpdate(" Insert into student
                 values('Jay','Sharma','BOM','Faridaba
                 d') ");

 //Delete Query
            stmt.executeUpdate(" Delete From student
                 Where firstname = 'Prateek' ");

 //Update Query
            stmt.executeUpdate("Update student Set
               course = 'Mechanical' Where
               firstname='Jay' ");

// Read Query or Retrieve
            ResultSet result = stmt.executeQuery("
                 Select * from student ");

            while(result.next()) {

                 System.out.println("First Name: " +
                      result.getString(1));

                 System.out.println("Last Name: " +
                      result.getString(2));

                 System.out.println("Course Selected:
```

```
                    " + result.getString(3));

                System.out.println("City: " +
                          result.getString(4));

                System.out.println();
            }

            con.close();

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

MySQL Java Insert Query with Scanner Class

- ```
  import java.sql.Connection;
  import java.sql.DriverManager;
  import java.sql.Statement;
  import java.util.Scanner;

  public class B {
      public static void main(String[] args) {
          try {
              Scanner scan = new Scanner(System.in);
              System.out.print("Enter Your First Name:
                                                    ");
              String firstName = scan.next();

              System.out.print("Enter Your Last Name:
                                                    ");
              String lastName = scan.next();

              System.out.print("Enter Your Course: ");
              String course = scan.next();

              System.out.print("Enter Your City: ");
              String city = scan.next();

              Connection con =
                        DriverManager.getConnection("jdb
                        c:mysql://localhost:3306/db_2","
                        root","password");

              System.out.println(con);

              Statement stmt = con.createStatement();
  ```
  ```

```
                    // Insert Query
                    stmt.executeUpdate(" Insert into student
                         values('"+firstName+"','"+lastName+"'
                         , '"+course+"', '"+city+"') ");


                    con.close();

               } catch (Exception e) {
                    e.printStackTrace();
               }
          }
     }
```

FINALLY

- Regardless of exception happen or not finally block will execute
- Database closing connection, we can use finally
- Example
  - ```
    public class E {
         public static void main(String[] args) {
              try {
                   int x = 10/0;
                   System.out.println(x);

              }finally {
                   System.out.println("Finally");
              }
         }}
    ```

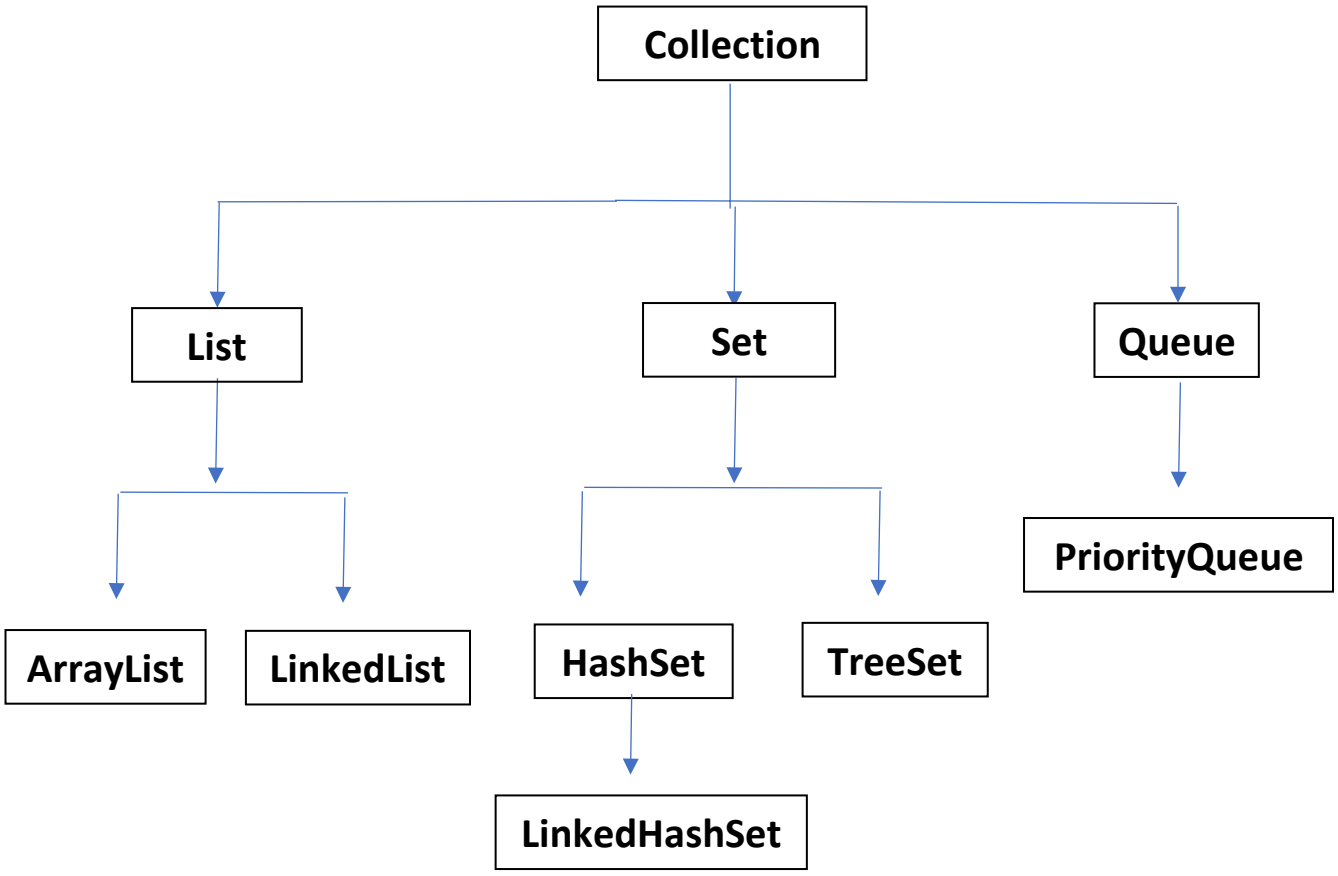
MULTIPLECATCHBLOCKISUSED

- You should start with child class and then parent class (exception e) will be last, because it is parent class.

COLLECTION

- It is nothing but collection of objects that been stored ideally in a particular entitle or area.
- Collection helps us to stored groups of objects in it.
    ```

- In java Collection is a framework (built-in methods), which is ideally readily available logic to deal with different data structures.



LIST (Interface- incomplete Method)

- Arraylist
- Linkedlist
- Vector

ARRAYLIST

- Internally it is implemented as dynamic array (size is not fixed).
- Initial size of ArrayList is 10.
- When we exceed the initial size, automatically ArrayList increases the size by 1.5 times
- ArrayList maintains insertion order.

- It can consist of duplicate elements.
- In ArrayList we can store heterogenous data in it.
- Example
  - ```
    import java.util.ArrayList;

    public class A {
        public static void main(String[] args) {
            ArrayList x = new ArrayList();
                        //default size is 10,
                        after 10, it increases
                        to 1.5 time of 10.

            x.add(10); //Boxing
            x.add(20); //Boxing
            x.add(new Integer(30)); //Boxing
            x.add("Prateek");//heterogenous data
            x.add(true); //heterogenous data
            x.add(10.5); //heterogenous data


            System.out.println(x);
        }}

    Output - [10, 20, 30, Prateek, true, 10.5]
    ```

Advantage of ArrayList

- Reading of data would give us best performance.

Disadvantage of ArrayList

- Insertion of data in between of the ArrayList, will result in worse performance.

ArrayList Examples

- Example 1
  - ```
    import java.util.ArrayList;
    ```

```
public class A {
    public static void main(String[] args) {
        ArrayList<Integer> x = new
                        ArrayList<Integer>();

        x.add(10);
        x.add(20);
        x.add(30);
        System.out.println("addMethod()= " +
                                        x);

        x.add(1,500);

        System.out.println("addMethod(Index,
                            value)= " + x);

        ArrayList<Integer> y = new
                        ArrayList<Integer>();

        y.add(300);
        y.add(500);
        x.addAll(2,y);
        System.out.println("addMethod(Index,
                        collection)= " + x);


        if(x.contains(500)) { //contains is
                            a search operation
            System.out.println("Present");
        }
        else {
            System.out.println("Not
                                    Present");
        }
        x.remove(1); //remove index 1 digit
        System.out.println("remove(index)
                        method(): "+ x);

    }}
```

Reading the data from an array list

- We using get method
- Example

```
public class A {
public static void main(String[] args) {
        ArrayList<Integer> x = new
                            ArrayList<Integer>();
        x.add(10);
        x.add(20);
        x.add(30);
        System.out.println("addMethod()= " + x);

        x.add(1,500);

        System.out.println("addMethod(Index,value)= "
                                    + x);

        ArrayList<Integer> y = new
ArrayList<Integer>();
        y.add(300);
        y.add(500);
        x.addAll(2,y);

System.out.println("addMethod(Index,collection)=
                                    " + x);


        if(x.contains(500)) { //contains is a
                                    search operation
            System.out.println("Present");
        }
        else {
            System.out.println("Not Present");
        }

        System.out.println(x.get(1));
    }}
```

ITERATOR

- Using iterator how we can read the value of an array list
- Example

```
Iterator itr = x.iterator();
        while(itr.hasNext()){
            System.out.println("Value of x:
                        " + itr.next());
            }
```

LINKEDLIST

- Singly Linkedlist
- Doubly Linkedlist
- In jdk Linkedlist is internally implemented as doubly Linkedlist
- Example

```
import java.util.LinkedList;
public class A {
    public static void main(String[] args) {
        LinkedList<Integer> x = new
                            LinkedList<Integer>();
        x.add(10);
        x.add(20);
        x.add(30);
        System.out.println("Array size: " +
                            x.size());
        System.out.println("Add method: " + x);
        x.add(1,500);
        System.out.println("Add method(index=1,
                            value=500): " + x);

        Iterator itr = x.iterator();
                while(itr.hasNext()){
                    System.out.println("Value of
                            x: " + itr.next());
        }
}}
```

- **Example Linkedlist addFirst**
  - ```
    import java.util.LinkedList;

    public class B {
        public static void main(String[] args) {
            LinkedList<Integer> x = new
                        LinkedList<Integer>();

            x.addFirst(100); // it become last
                               element
            x.addFirst(200);
            x.addFirst(300);
            x.addFirst(400); //it become first
                               element
            System.out.println(x); //
                            [400,300,200,100]
        }}
    ```

- **Example of addLast**
  - ```
    import java.util.LinkedList;

    public class B {
        public static void main(String[] args) {
            LinkedList<Integer> x = new
                        LinkedList<Integer>();
            x.addFirst(100);
            x.addFirst(200);
            x.addFirst(300);
            x.addFirst(400); //it become second
                     element because of 15line

            x.addLast(500); // it become second
                        last element because of
                        13line
            x.add(600); // it simple become last
                        element

            x.addFirst(800); // it become first
                            element
            System.out.println(x); // [800, 400,
                               300, 200, 100,
                               500, 600]
        }}
    ```

- Program using LinkedList
  - ```
    import java.util.LinkedList;

    public class Empy {
        public static void main(String[] args) {
            Employee arun = new Employee("Arun", "k",
                                                    100);

            Employee ravi = new Employee("Ravi",
                                          "kiran", 200);

            Employee santosh = new Employee("Santosh",
                                             "m", 300);

            LinkedList<Employee> empDetail = new
                                 LinkedList<Employee>();

            empDetail.add(arun);
            empDetail.add(ravi);
            empDetail.add(santosh);

            System.out.println(empDetail);

            for (Employee employee : empDetail) {

        System.out.println(employee.getFirstName());
        System.out.println(employee.getLirstName())
        System.out.println(employee.Id())
            }
        }
    }

    public class Employee {
        private String firstName;
        private String lastName;
        private int id;

        Employee(String firstName, String lastName, int
                                                    id){
            this.firstName = firstName;
            this.lastName = lastName;
            this.id = id;
        }

        public String getFirstName() {
            return firstName;
        }

        public void setFirstName(String firstName) {
            this.firstName = firstName;
    ```
    ```

```
        public String getLastName() {
            return lastName;
        }
        public void setLastName(String
                                        lastName){

            this.lastName = lastName;
    }

        public int getId() {
            return id;
    }

        public void setId(int id) {
            this.id = id;
    }

}}
```

HASING

- Hash code give an integer value and we will divide with size of an array, the result gives that index number and later we inject in hash table.
- Example
  Integer value = hashcode();
  Index number = Integer value/ size of an array
- Hashing is a technic where we represent any entity in the form of integer and it is done in java using a hash code method
- Hash code is a method present inside object class in java

HASHTABLE

- Is an associated array, where in value is store as a key, value pair

COALITION

- When two value are being stored at the same index number are called as coalition.

- To solved this problem is hash table, we store the data as list, mapped to the same index number.

SET

- It an interface
- Does not maintain any insertion order
- Cannot contain duplicate values

HASHSET

- It uses hash table internally
- It using hashing to inject the data into database
- It will contain only unique elements
- Does not maintain insertion order
- This is not synchronized
- This class permits null element
- Initial size of the hash table is 16, when the load ratio become 75% that it out of 16, 12 elements are injected into the table then, the size of the table automatically doubles.

Examples of HashSet

- **Example one** insertion order is not maintained
  - ```
    import java.util.HashSet;

    public class A {
        public static void main(String[] args) {
            HashSet<Integer> hasSet = new
                                HashSet<Integer>();
            hasSet.add(20);
            hasSet.add(30);
            hasSet.add(40);
            hasSet.add(50);

            System.out.println(hasSet); // [50,
    ```
    ```

```
                                        20, 40, 30]
                                        insertion order
                                        not maintain
        }}
```

- **Example Two –** null value
  - ```
    import java.util.HashSet;

    public class A {
        public static void main(String[] args) {
            HashSet<Integer> hasSet = new
                                HashSet<Integer>();
            hasSet.add(20);
            hasSet.add(30);
            hasSet.add(40);
            hasSet.add(50);
            hasSet.add(null);

            System.out.println(hasSet); //[null,
                                50,20,40,30]
    ```

LINKEDHASHSET

- It maintains insertion order
- It can contain only unique elements (no duplicate values)
- Example
  - ```
    import java.util.LinkedHashSet;

    public class A {
        public static void main(String[] args) {
            LinkedHashSet<Integer> hasSet = new
                        LinkedHashSet<Integer>();
            hasSet.add(20);
            hasSet.add(30);
            hasSet.add(40);
            hasSet.add(50);

            hasSet.add(null);

            System.out.println(hasSet); //[20,
                                30, 40, 50, null]
        }}
    ```

TREESET

- It contains unique element only
- It sorts the data in ascending order
- Example
  - ```
    import java.util.TreeSet;

    public class B {
        public static void main(String[] args) {
            TreeSet<Integer> treeSet = new
                                TreeSet<Integer>();

            treeSet.add(30);
            treeSet.add(20);
            treeSet.add(100);
            treeSet.add(390);
            treeSet.add(330);
            System.out.println(treeSet); //[20,
                                30, 100, 330, 390]
        }
    }
    ```

MAP

- HashMap internally uses hash table
- To inject the data into hash table it using hashing technic
- A hash map stores the data as key, value pair
- Hash Map is not synchronized
-
- Example

```
import java.util.HashMap;
    import java.util.Map;

    public class B {
        public static void main(String[] args) {
            Map<Integer,String> studentInfo =
                        new HashMap<Integer,
                        String>(); //Map is
                                parent class
```
```

```
            studentInfo.put(100,"Prateek");
            studentInfo.put(101,"Jay");
            studentInfo.put(102,"Joshi");
            System.out.println(studentInfo);


        System.out.println(studentInfo.get(101));

        System.out.println(studentInfo.KeySet());
                        //[100, 101, 102]

        System.out.println(studentInfo.values());
                        //[Prateek, Jay, Joshi]
        }
    }
```

HASHTABLE

- It stores the content as key, value pair
- Hash Table is synchronized
- **Example**
  - ```
    import java.util.Hashtable;

    public class B {
        public static void main(String[] args) {
            Hashtable<Integer, String> hashTable
                    = new Hashtable<Integer,
                                    String>();

            hashTable.put(100, "Prateek");
            hashTable.put(101, "Jay");
            System.out.println(hashTable);
        }}
    ```

COMPARATORINTERFACE

- It compares object content
- It used to order the objects of user-defined class
    ```

- If in sorting object 1 come first and then the object 2, then it will return negative value. But if will sorting, object 2 comes first and then object 1, then it will return positive value.
- If both the object are same, then it will return zero
- Example one
  - ```
    public class A {
        public static void main(String[] args) {
            String x = "xyz";
            String y = "abc";

            System.out.println(x.compareTo(y));
                                //positive value
        }}
    ```

- Example two
  - ```
    public class Student {
        int id;
        String name;

        Student(int id, String name){
            this.id = id;
            this.name = name;
        }
    }

    import java.util.Comparator;

    public class A implements Comparator<Student>
    {
        @Override
        public int compare(Student o1, Student
        o2) {
            return o1.name.compareTo(o2.name);

        }
        public static void main(String[] args) {
            A a1 = new A();
            int val = a1.compare(new
                      Student(100, "xyz"), new
                      Student(200,"abc"));
            System.out.println(val);
        }}
    ```

GENERICS

- Generics give flexibility to dynamically allocated datatype
- **Example** where T is used, the datatype will replace by the input datatype enter in the main method.
  - ```
    public class Test {
        public static void main(String[] args) {
            B<String> a1 = new B<String>("abc");

            B<Integer> a2 = new B<Integer>(100);

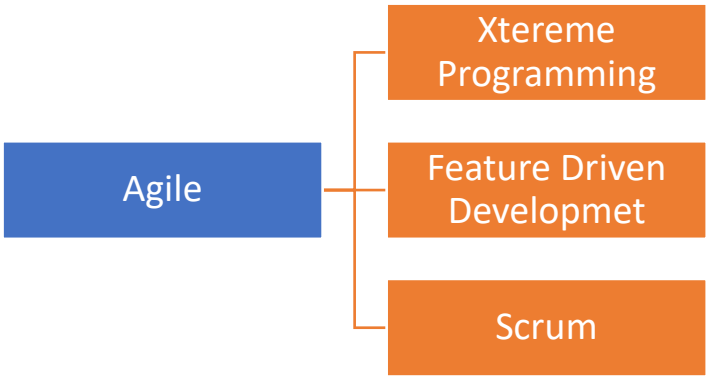
            B<Character> a3 = new
                                B<Character>('a');
        }}

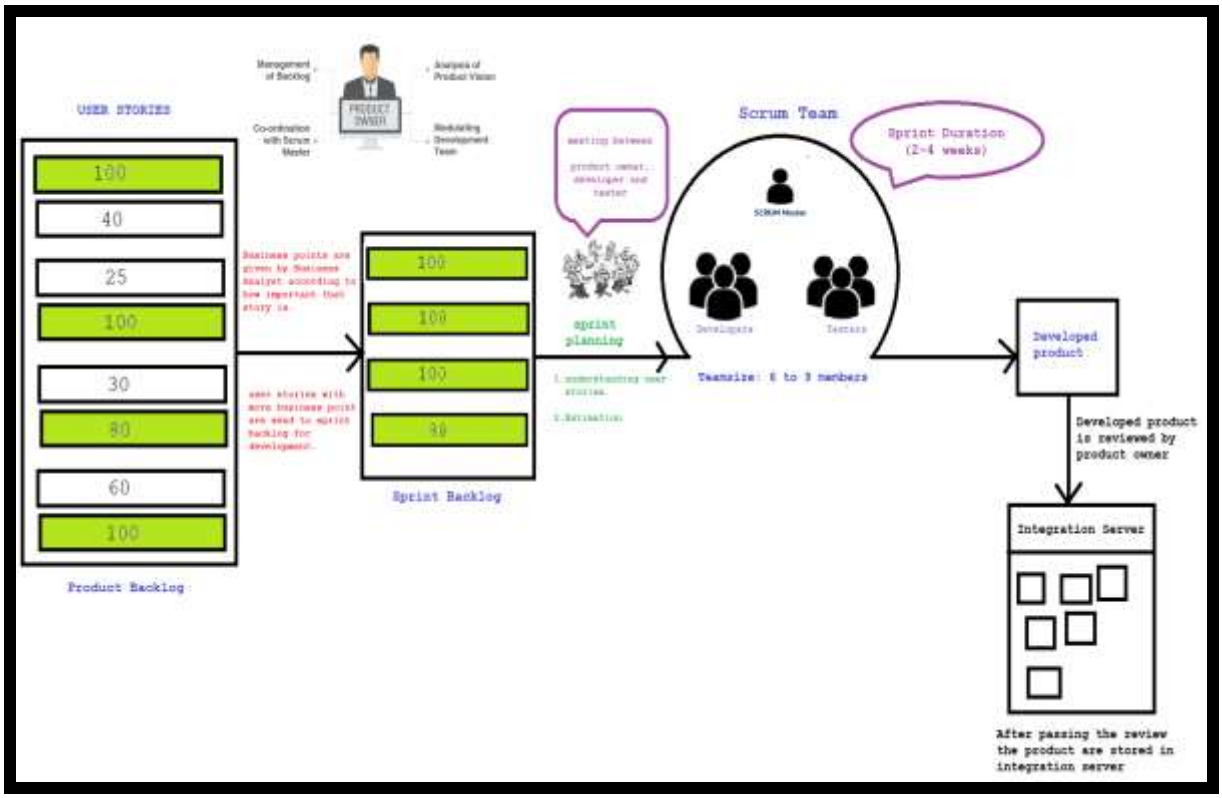
    import java.util.ArrayList;
    public class B<T> {
        T val; //T can be replace with any
                                datatype (int,
                                string, etc)

        B(T x){
            this.val = x;
            System.out.println(val);
        }
    }
    ```
    ```

AGILE SCRUM

Agile is a methodology which helps us to develop a software quickly without compromising with the quality.





Step 1.

collection of user stories.

Step 2.

Business points are given by Business Analyst to classify how important are the user stories.

Step 3.

User stories are kept in product backlog and are maintained by product owner.

Step 4.

Only those many stories are picked up and kept in sprint backlog which could be completed under max of 4 weeks or min of 2 weeks.

Step 5.

Sprint planning meeting is conducted between testers, developer, and product owner to understand user stories and estimation

Step 6.

Project is developed, and testing happens within the given duration which was estimated in sprint planning.

Step 7.

Product owner is called for review of the modules.

Step 8.

After passing the review the modules are kept in integration server.

Who are involved in scrum team?

Scrum master, developers, testers

What were you doing in sprint planning?

Understanding user stories and doing estimation

What is sprint backlog?

It consists of user stories which will be developed.

What is product backlog?

Consists of all user stories .Which need to be developed

How many minutes the stand-up meeting used to go on in your project?

15 minutes

Who are involved in stand-up meeting?

Scrum master, developers, and testers

Product owner

- He is a domain expert
- He maintains product backlog
- He also ideally conducts sprint planning

Sprint backlog

- *Here we keep only those user stories which is going to be developed now*
- *All user stories kept here be such that we can develop it within 2-4 weeks*

Sprint planning

- This meeting happens between product owner, developers and testers
- In this meeting we do following:
- Understanding user stories
- Estimation

Sprint

- It is the duration within which a working software is developed
- Sprint duration should always be between 2-4 weeks

Scrum team

- Scrum team consists of scrum master, developers, and testers
- Scrum team size should always be between 6-9.
- There will be several scrum teams in the project
- Work done by individuals scrum team after review will be integrated with the features developed by the other teams in the integration server. Then we once again test how well they are integrated

Sprint blow-up

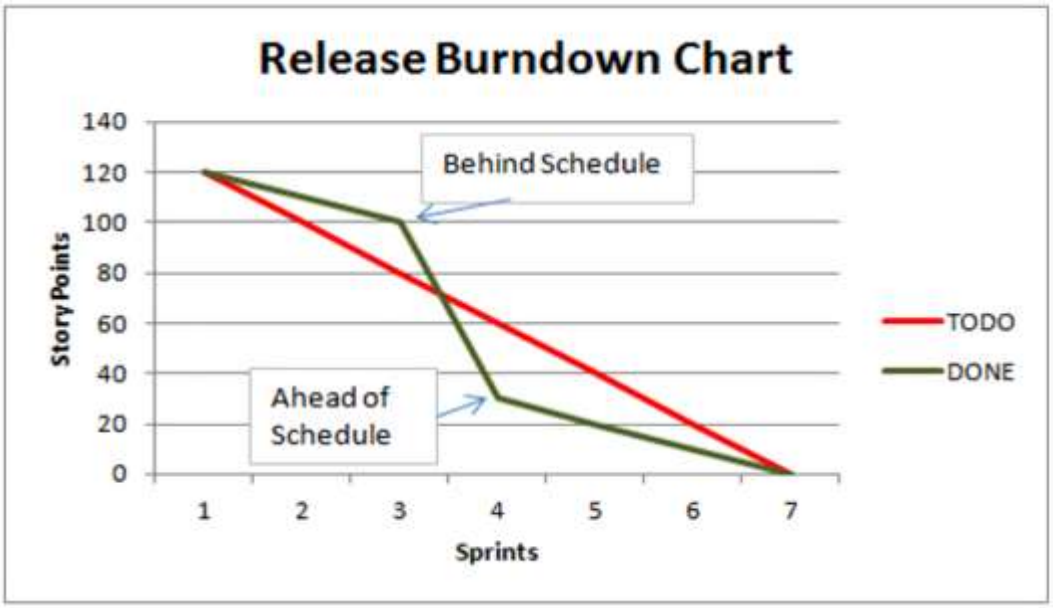
- If the work is not completed within the sprint duration, then we "Blow-up" the sprint and again start with the new sprint and new estimation.

Planning poker activity

- It is the meeting that happens between business analyst and product owner to give the business value points.
- To give business value points we use poker cards
- This classification helps us to decide which user stories should be developed first.

Job of Scrum master

1. Scrum master generates burn down chart to analyse progress of work. He can also make use of burnup chart, but it is not that popular in industry .



2. He should conduct daily stand-up meeting
   In stand-up meeting discussion on what an individual have done yesterday, what he will do today and what are the obstacles he is facing.

3. He should remove obstacle encountered by the team.

4. He should promote scrum values.

Pigs and hens in agile

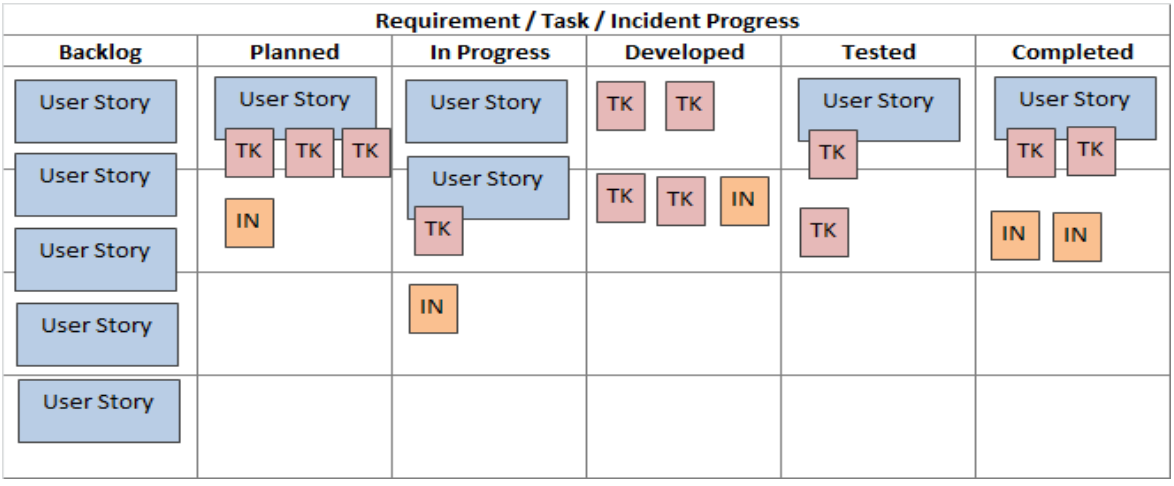
- People committed in the project are called pigs
  e.g., developers, testers, scrum masters, product owners, etc.

- The one who is not committed in the project in agile are called as hens
  e.g., managers , VP's etc

In stand-up meeting only pigs are supposed to talk while hens can only attend the meeting but should not talk during the meeting

Kanban chart

To maintain transparency in agile among the team members we prepare Kanban chart.



Advantages and disadvantages of agile

Advantages

- Helps us to develop software quickly
- Maintains transparency among the team members
- Better communication between team members
- Gives assurance of quality software

Disadvantages

- Manager loses his authority.
- We require little bit of experience to work in this process.

PARAMETERS	AGILE SOFTWARE DEVELOPMENT	WATERFALL SOFTWARE DEVELOPMENT
Classification	The software Development Life Cycle is divided into different sprints.	Software Development Life Cycle is divided into different phases
Approach	It takes an incremental and continuous iteration approach	It is sequential in order.
Flexibility	Agile is comparatively flexible to the changes.	The software development is very rigid and there is no room for change.
Testing	Testing and planning is done after each sprint	Testing is the final phase and is not done during the development stage.
Focus	Customer satisfaction and better quality product or software is the main focus in the Agile	Fast turnover of the product and meeting the deadline is the main focus.
Managerial need	There is no need for a product manager as the whole project is managed as a team.	It is a straightforward approach, where a project manager is needed to check the proper working.
Suitability	It can handle many small projects and some large projects	It can handle only one single project at a time.
Requirements	The requirements can change day today.	The requirements are collected at the beginning of the project and there no room for change.

Advance Java Notes

SERVLETLIFECYCLE

- For the first time, when we start tomcat, init method will run only once, but services method can be called any number of times, finally when the destroyed method is called, servlet life cycle comes to an end.
- In other words, the init method of servlet will run automatically only once. then doPost and doGet method can be execution any number of times. Once the complete execution of servlet done, the destroy method will run, and execution of servlet comes to end.

SERVLET

- It is a java class
- Servlet is a sub-class of http servlet
- Servlet interact with front end application

POSTANDGET

- Whenever we develop a form to submit to the data to database is Post method
- Whenever we reading the data from database and displaying that in the form is Get method

POST

- **Should be used when submit the data to database**
- **The data is not exposed in url, and hence it is more secured**
- **Upon refreshing page, we get security popup alert, avoid duplicated transactions**
- Submitting the data to the database, then use doPost
- Post is a little safer than, get because the parameters are not stored in browser history or in web server logs
- Post cannot be bookmarked

- Data will be re-submitted (the browser should alert the user that the data are about to be re-submitted.

GET

- **Should be used when we get the data from the database**
- **Data is exposed in url and hence less secured**
- **Upon refreshing page, we do not get security popup alert**
- Reading the data from database, then use doGet
- Get is less secure compared to Post, because data send is part of the URL
- Parameters remain in browser history
- Get can be bookmarked
- It is harmless for back button and reload

Web Page Program for Inserting data into Database

Java Program

```
@WebServlet("/new")
public class InsertRegistration extends HttpServlet {
     private static final long serialVersionUID = 1L;

    public InsertRegistration() {
        super();

    }
     protected void doGet(HttpServletRequest request,
     HttpServletResponse response) throws ServletException,
     IOException {

          RequestDispatcher rd =
          request.getRequestDispatcher("WEB-
          INF/lib/views/InsertRegistration.html");
          rd.forward(request, response);
     }

     protected void doPost(HttpServletRequest request,
     HttpServletResponse response) throws ServletException,
     IOException {

          String name = request.getParameter("name");
          String city = request.getParameter("city");
          String email = request.getParameter("email");
```

```
            String mobile = request.getParameter("mobile");

            try {
                  Class.forName("com.mysql.jdbc.Driver");
                  Connection con =
                  DriverManager.getConnection("jdbc:mysql://local
                  host:3306/db_1","root","password");

                  Statement stmnt = con.createStatement();
                  stmnt.executeUpdate("INSERT INTO registration
                  VALUES('"+name+"', '"+city+"', '"+email+"',
                  '"+mobile+"')");
                  con.close();

            } catch (Exception e) {
                  e.printStackTrace();
            }
      }
}
```

Html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert Page</title>
</head>
<body>
      <h2>Insert | Registration</h2>
      <form action="new" method="post">
            <pre>
                  Name <input type="text" name="name"/>
                  City <input type="text" name="city"/>
                  Email <input type="text" name="email"/>
                  Mobile <input type="text" name="mobile"/>
                  <input type="submit" value="save"/>
            </pre>
      </form>
</body>
</html>
```

Web Page Program for Reading/ Retrieve Data from Database using doGet method only

Java

```
@WebServlet("/retrieve")
public class RetrieveRegistration extends HttpServlet {
     private static final long serialVersionUID = 1L;

    public RetrieveRegistration() {
        super();

    }
     protected void doGet(HttpServletRequest request,
     HttpServletResponse response) throws ServletException,
     IOException {
          try {
               Class.forName("com.mysql.jdbc.Driver");
               Connection con =
               DriverManager.getConnection("jdbc:mysql://local
               host:3306/db_1","root","password");

               Statement stmnt = con.createStatement();

               ResultSet result = stmnt.executeQuery("SELECT *
               FROM registration");
               PrintWriter out = response.getWriter();
               out.println("<title>");
               out.println("Retrieve Page");
               out.println("</title>");

               out.println("<h2>");
               out.println("Retrieve | Registration");
               out.println("</h2>");

               out.println("<table>");

               out.println("<tr>");
               out.println("<th>");
               out.println("Name");
               out.println("</th>");

               out.println("<th>");
               out.println("City");
               out.println("</th>");

               out.println("<th>");
               out.println("Email");
               out.println("</th>");
```

```
                out.println("<th>");
                out.println("Mobile");
                out.println("</th>");
                out.println("</tr>");

                while(result.next()) {
                     out.println("<tr>");
                     out.println("<td>");
                     out.println(result.getString(1));
                     out.println("</td>");

                     out.println("<td>");
                     out.println(result.getString(2));
                     out.println("</td>");

                     out.println("<td>");
                     out.println(result.getString(3));
                     out.println("</td>");

                     out.println("<td>");
                     out.println(result.getString(4));
                     out.println("</td>");
                     out.println("<tr>");
                }

                out.println("</table>");
                con.close();

           } catch (Exception e) {
                e.printStackTrace();
           }
     }
     protected void doPost(HttpServletRequest request,
     HttpServletResponse response) throws ServletException,
     IOException {

     }
}
```

INTERSERVLETCOMMUNICATION

- When a call to one servlet is made from another servlet using RequestDispatcher then it called as Inter Servlet Communication
- When one servlet calls another servlet, it called as Inter Servlet Communication.
- Example

HTML Code

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Index</title>
</head>
<body>
     <form action="firstServlet" method="post">
          Name <input type="text" name="name"/>
          <input type="submit" value="Send"/>
     </form>
</body>
     </html>
```

Java Code (Servlet) one

```
@WebServlet("/firstServlet")

public class FirstServlet extends HttpServlet {
     private static final long serialVersionUID = 1L;
    public FirstServlet() {
        super();

    }
     protected void doGet(HttpServletRequest request,
     HttpServletResponse response) throws
     ServletException, IOException {

     }
     protected void doPost(HttpServletRequest request,
     HttpServletResponse response) throws
     ServletException, IOException {

          String name = request.getParameter("name");
          request.setAttribute("msg", name);

          RequestDispatcher rd =
          request.getRequestDispatcher("secondServlet");
          rd.forward(request, response);
     }
}
```

Java Code (Servlet) Two

```
@WebServlet("/secondServlet")

public class SecondServlet extends HttpServlet {
```

```
    private static final long serialVersionUID = 1L;
    public SecondServlet() {
        super();

    }
    protected void doGet(HttpServletRequest request,
    HttpServletResponse response) throws
    ServletException, IOException {

    }
    protected void doPost(HttpServletRequest request,
    HttpServletResponse response) throws
    ServletException, IOException {

        Object name =
        (String)request.getAttribute("msg");
        PrintWriter out = response.getWriter();

        out.println(name);
    }
}
```

SESSION

- **Once a value is store in session variable then the value can be access across the application**
- Any data stored in the session can be accessed throughout the application
- It set and get attribute is use then that data can be access by other servlet on through inter servlet communication

A login program with session

Java Servlet

```
@WebServlet("/loginServlet")
public class LoginServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    public LoginServlet() {
        super();
    }

    protected void doGet(HttpServletRequest request,
    HttpServletResponse response) throws ServletException,
    IOException {
```

```
	}
	protected void doPost(HttpServletRequest request,
	HttpServletResponse response) throws ServletException,
	IOException {
		String emailid = request.getParameter("emailid");
		String password = request.getParameter("password");

		try {
			Class.forName("com.mysql.jdbc.Driver");
			Connection con =
			DriverManager.getConnection("jdbc:mysql://local
			host:3306/login_app_db","root","password");

			Statement stmnt = con.createStatement();

			ResultSet results = stmnt.executeQuery("SELECT
			* FROM login WHERE emailid='"+emailid+"' AND
			password='"+password+"'");

			HttpSession session = request.getSession(true);

			if (results.next()) {
				session.setAttribute("emailid", emailid);

				RequestDispatcher rd1 =
				request.getRequestDispatcher("WEB-
				INF/lib/views/registerUser.html");

				rd1.forward(request, response);

			}else {
				RequestDispatcher rd2 =
				request.getRequestDispatcher("index.html")
				;
				rd2.forward(request, response);
			}

		} catch (Exception e) {
			e.printStackTrace();
		}
	}
}
```

HTML Code

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Home Page</title>
</head>
<body>
	<h2>Login Here</h2>
	<hr>
	<form action="loginServlet" method="post">
		Email ID <input type="text" name="emailid"/>
		Password <input type="text" name="password"/>
		<input type="submit" name="Login"/>
	</form>
</body>
</html>
```

JSPLIFECYCLE

- Whatever java code we write in jsp file with the help of jsp translator we translate the java code into it. After tomcat is started, jsp init method will run only once.
- JSP service method, can be called any number of times.
- Finally, when, Jsp destroyed method is called, then jsp lifecycle come to an end.

JSP (Jakarta Server Page)

- JSP helps us to embed to java code inside html
- To embed the java code inside html, we use the following tags.

SCRIPLETATG <% %>

- Inside the Scriptled tag we can create partial java code like, for loop, while loops. If conditions etc.
- What we cannot do inside a Scriptled tag is uses of excess specifier and creation of methods.
- Example

```
<%@ page language="java" contentType="text/html;
charset=ISO-8859-1"
```

```
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>New File</title>
</head>
<body>
      <h2>Prateek Upreti</h2>
      <br/>
      <%
            for (int i=0; i<10; i++){
                  out.println(i);
            }
      %>
</body>
      </html>
```

DECLARATIONTAG<%! %>

- We can create methods and variables with access specifiers
- Built in variables like out, request, response will not work in declaration tag, cannot do implicit objects
- Example

```
<%@ page language="java" contentType="text/html;
charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>New File Two</title>
</head>
<body>
   <%!
         public int x = 200;
         public int test(){
               return 100;

   }
         out.println(100); //Error cannot do build in
                           variables
   %>
   <br/>
   <%
         int val = test();
         out.println(val);
```

```
        out.println(x);
   %>
</body>
     </html>
```

EXPRESSIONTAG<%= %>

- It evaluates the expression and prints the output on the webpage
- At most you can give only one statement
- Example of expression tag and page directive tag

```
<%@ page language="java" contentType="text/html;
charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>

<%@ page import = "java.sql.DriverManager, java.util.*" %>

<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>New File Three</title>
</head>
<body>
   <%
        int x=10;

   %>

   <%=”Hello” %>
   <%=100+200%>

   <%=new Date()%>
   <%=new ArrayList<Integer>()%>
   <%=DriverManager.getConnection("".””,"") %>

</body>
</html>
```

PAGEDIRECTIVETAG <%@ %>

- Page directive tag used for importing required classes, interfaces etc

INCLUDEDIRECTIVETAG <%@ %>

- Include Directive tag help us for repeating header and footer in every Jsp or html pages.

Program using JSP, we login, Insert data and logout

HTML Code one

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Login Page</title>
</head>
<body>
     <h2>Login Here</h2>
     <form action="loginServlet" method="post">
          Email <input type="text" name="email"/>
          Password <input type="text" name="password"/>
          <input type="submit" value="Login"/>
     </form>
</body>
</html>
```

HTML Code two

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert Page</title>
</head>
<body>
     <h2>This is Registration Page.</h2>
     <form action="logoutServlet" method="post">
          <input type="submit" value="Logout"/>
     </form>

     <form action="insertServlet" method="post">
```

```
            <table>
                  <tr>
                        <td> Name </td>
                        <td><input type="text" name="name"/></td>
                  </tr>
                  <tr>
                        <td> City </td>
                        <td><input type="text" name="city"/></td>
                  </tr>
                  <tr>
                        <td> Email </td>
                        <td><input type="text" name="email"/></td>
                  </tr>
                  <tr>
                        <td> Mobile </td>
                        <td><input type="text"
                        name="mobile"/></td>
                  </tr>
                  <tr>
                        <td><input type="submit"
                        value="Send"/></td>
                  </tr>
            </table>
      </form>
</body>
</html>
```

Java Code one – login Servlet

```
@WebServlet("/loginServlet")
public class loginServlet extends HttpServlet {
      private static final long serialVersionUID = 1L;
     public loginServlet() {
         super();

     }
      protected void doGet(HttpServletRequest request,
      HttpServletResponse response) throws ServletException,
      IOException {

      }
      protected void doPost(HttpServletRequest request,
      HttpServletResponse response) throws ServletException,
      IOException {
            String email = request.getParameter("email");
            String password = request.getParameter("password");

            try {
                  Class.forName("com.mysql.jdbc.Driver");
```

```
                Connection con =
                DriverManager.getConnection("jdbc:mysql://local
                host:3306/login_app_db","root","password@");

                Statement stmnt = con.createStatement();

                ResultSet results = stmnt.executeQuery("SELECT
                * FROM login WHERE emailid='"+email+"' AND
                password='"+password+"'");

                HttpSession session = request.getSession(true);
                if (results.next()) {

                     session.setAttribute("email", email);
                     RequestDispatcher rd_1 =
                     request.getRequestDispatcher("insertRegist
                     ration.jsp");

                     rd_1.forward(request, response);
                }else {
                     RequestDispatcher rd_2 =
                     request.getRequestDispatcher("login.jsp");
                     rd_2.include(request, response);
                }
                con.close();
           } catch (Exception e) {
                e.printStackTrace();
           }
      }
}
```

Java Code two – inserting data

```
@WebServlet("/insertServlet")
public class InsertServlet extends HttpServlet {
     private static final long serialVersionUID = 1L;
    public InsertServlet() {
        super();

    }
     protected void doGet(HttpServletRequest request,
     HttpServletResponse response) throws ServletException,
     IOException {

     }
     protected void doPost(HttpServletRequest request,
     HttpServletResponse response) throws ServletException,
     IOException {
          String name = request.getParameter("name");
          String city= request.getParameter("city");
          String email = request.getParameter("email");
```

```
            String mobile = request.getParameter("mobile");
            HttpSession session = request.getSession();

            if (session.getAttribute("email") != null) {
                 try  {
                      Class.forName("com.mysql.jdbc.Driver");
                      Connection con =
                      DriverManager.getConnection("jdbc:mysql://
                      localhost:3306/login_app_db","root","passw
                      rod");

                      Statement stmnt = con.createStatement();
                      stmnt.executeUpdate("INSERT INTO
                      registration VALUES('"+name+"',
                      '"+city+"', '"+email+"', '"+mobile+"' )");

                      RequestDispatcher rd =
                      request.getRequestDispatcher("insertRegist
                      ration.jsp");

                      rd.include(request, response);

                 } catch (Exception e) {
                      e.printStackTrace();
                 }
            }
       }
}
```

Java Code three – logout Servlet

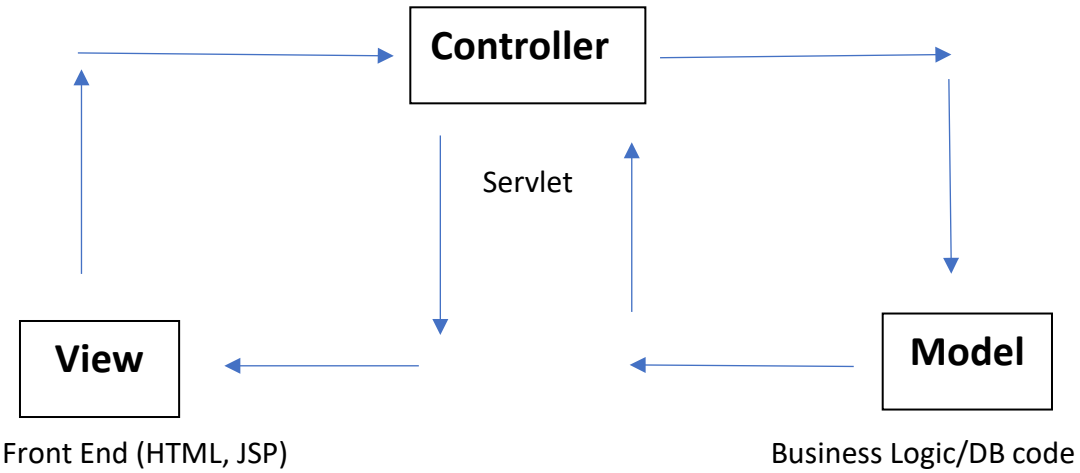
```
@WebServlet("/logoutServlet")
public class logoutServlet extends HttpServlet {
     private static final long serialVersionUID = 1L;
    public logoutServlet() {
        super();
    }
     protected void doGet(HttpServletRequest request,
     HttpServletResponse response) throws ServletException,
     IOException {

     }
     protected void doPost(HttpServletRequest request,
     HttpServletResponse response) throws ServletException,
     IOException {
          HttpSession session = request.getSession(false);
          try {
               session.invalidate();
               RequestDispatcher rd =
```

```
                request.getRequestDispatcher("login.jsp");
                rd.forward(request, response);

            } catch (Exception e) {
                e.printStackTrace();
            }
        }
}
```

MVCArchitecture (Model View Controller)



VIEW

- All the front-end code like html, css, we build that in view layer.

CONTROLLER

- Controller is responsible to read the data from view and passed on the model layer.
- It takes the data from model layer and gives it back to the view layer. Its act like a mediator between the view and model layer

MODEL

- All the business logic of the application is developed in the model layer.
- It also includes all the database related activity in it.

MVC

- In View, all the html, JSP is developed in the view layer of application
- Controller is responsible to interact with the view, receive the inputs and further passes on those input to model layer and also take the output from model layer and give it to view
- Usually. we used servlet to build controller layer
- In model layer, logical implementation or database operations are perform

Mini Project based on JSP and MVC

Com.web_app.Controllers

LoginController Servlet

```
@WebServlet("/loginController")
public class LoginController extends HttpServlet {
     private static final long serialVersionUID = 1L;
    public LoginController() {
        super();
    }
     protected void doGet(HttpServletRequest request,
     HttpServletResponse response) throws ServletException,
     IOException {

     }
     protected void doPost(HttpServletRequest request,
     HttpServletResponse response) throws ServletException,
     IOException {
          String email = request.getParameter("email");
          String password = request.getParameter("password");

          DAOService service = new DAOServiceImpl();
          service.connectDB();

          boolean active = service.loginDB(email, password);
          HttpSession session = request.getSession(true);

          if(active==true) {
               session.setAttribute("email", email);
               session.setMaxInactiveInterval(10);

               RequestDispatcher rd_1 =
               request.getRequestDispatcher("WEB-
               INF/views/home.jsp");
               rd_1.forward(request, response);

          }else {
               request.setAttribute("msg", "Invalid
               Email/Password");
               RequestDispatcher rd =
               request.getRequestDispatcher("login.jsp");
               rd.include(request, response);
          }
     }
}
```

LogoutController

```
@WebServlet("/logoutController")
public class LogoutController extends HttpServlet {
     private static final long serialVersionUID = 1L;
    public LogoutController() {
        super();
    }
     protected void doGet(HttpServletRequest request,
     HttpServletResponse response) throws ServletException,
     IOException {
          HttpSession session = request.getSession(false);
          session.invalidate();
          RequestDispatcher rd =
          request.getRequestDispatcher("login.jsp");
          rd.include(request, response);
     }
     protected void doPost(HttpServletRequest request,
     HttpServletResponse response) throws ServletException,
     IOException {

     }
}
```

CreateController

```
@WebServlet("/createController")
public class CreateController extends HttpServlet {
     private static final long serialVersionUID = 1L;
    public CreateController() {
        super();
    }
     protected void doGet(HttpServletRequest request,
     HttpServletResponse response) throws ServletException,
     IOException {

               RequestDispatcher rd =
               request.getRequestDispatcher("WEB-
               INF/views/create.jsp");
               rd.forward(request, response);

     }
     protected void doPost(HttpServletRequest request,
     HttpServletResponse response) throws ServletException,
     IOException {
     try {
          HttpSession session = request.getSession(false);
```

```
            session.setMaxInactiveInterval(10);
            if (session.getAttribute("email") != null) {
                String fname = request.getParameter("fname");
                String lname = request.getParameter("lname");
                String email = request.getParameter("email");
                String city = request.getParameter("city");
                String mobile = request.getParameter("mobile");

                DAOService service = new DAOServiceImpl();
                service.connectDB();

                service.createDB(fname, lname, email, city,
                mobile);

                request.setAttribute("msg", "Registration
                Saved");

                RequestDispatcher rd =
                request.getRequestDispatcher("WEB-
                INF/views/create.jsp");

                rd.forward(request, response);

            }else {
                RequestDispatcher rd =
                request.getRequestDispatcher("login.jsp");
                rd.forward(request, response);
            }
        } catch (Exception e) {
            request.setAttribute("msg", "SessionTimeOut. Please
            Login in again.");

            RequestDispatcher rd =
            request.getRequestDispatcher("WEB-
            INF/views/create.jsp");
            rd.forward(request, response);
            e.printStackTrace();
            }
        }
}
```

RetrieveController

```
@WebServlet("/retrieveController")
public class RetrieveController extends HttpServlet {
     private static final long serialVersionUID = 1L;
    public RetrieveController() {
        super();
    }
```

```
	protected void doGet(HttpServletRequest request,
	HttpServletResponse response) throws ServletException,
	IOException {
		HttpSession session = request.getSession(false);
		if (session.getAttribute("email") != null) {

			DAOService service = new DAOServiceImpl();
			service.connectDB();

			ResultSet result = service.retrieveDB();
			request.setAttribute("result", result);

			RequestDispatcher rd =
			request.getRequestDispatcher("WEB-
			INF/views/retrieve.jsp");
			rd.forward(request, response);
		}else {
			RequestDispatcher rd =
			request.getRequestDispatcher("login.jsp");
			rd.forward(request, response);
		}


	}
	protected void doPost(HttpServletRequest request,
	HttpServletResponse response) throws ServletException,
	IOException {

	}
}
```

DeleteController

```
@WebServlet("/deleteController")
public class DeleteController extends HttpServlet {
	private static final long serialVersionUID = 1L;
    public DeleteController() {
        super();
    }
	protected void doGet(HttpServletRequest request,
	HttpServletResponse response) throws ServletException,
	IOException {
		HttpSession session = request.getSession(false);
		if (session.getAttribute("email") != null) {
			String fname = request.getParameter("fname");
			DAOService service = new DAOServiceImpl();
			service.connectDB();

			service.deleteDB(fname);
```

```
                ResultSet result = service.retrieveDB();
                request.setAttribute("result", result);

                RequestDispatcher rd =
                request.getRequestDispatcher("WEB-
                INF/views/retrieve.jsp");
                rd.forward(request, response);
           }else {
                RequestDispatcher rd =
                request.getRequestDispatcher("login.jsp");
                rd.forward(request, response);
           }
     }
     protected void doPost(HttpServletRequest request,
     HttpServletResponse response) throws ServletException,
     IOException {

     }
}
```

UpdateController

```
@WebServlet("/updateController")
public class UpdateController extends HttpServlet {
     private static final long serialVersionUID = 1L;
    public UpdateController() {
        super();
    }
     protected void doGet(HttpServletRequest request,
     HttpServletResponse response) throws ServletException,
     IOException {
          HttpSession session = request.getSession(false);
          if (session.getAttribute("email") != null) {

                String fname = request.getParameter("fname");
                String email = request.getParameter("email");
                String city = request.getParameter("city");
                String mobile = request.getParameter("mobile");

                request.setAttribute("fname", fname);
                request.setAttribute("email", email);
                request.setAttribute("city", city);
                request.setAttribute("mobile", mobile);

                RequestDispatcher rd =
                request.getRequestDispatcher("WEB-
                INF/views/update.jsp");
                rd.forward(request, response);
           }else {
```

```
                RequestDispatcher rd =
                request.getRequestDispatcher("login.jsp");
                rd.forward(request, response);
            }

        }
        protected void doPost(HttpServletRequest request,
        HttpServletResponse response) throws ServletException,
        IOException {
            HttpSession session = request.getSession(false);
            if (session.getAttribute("email") != null) {

                String fname = request.getParameter("fname");
                String email = request.getParameter("email");
                String city = request.getParameter("city");
                String mobile = request.getParameter("mobile");

                DAOService service = new DAOServiceImpl();
                service.connectDB();

                service.updateDB(fname, email, city, mobile);

                ResultSet result = service.retrieveDB();
                request.setAttribute("result", result);
                RequestDispatcher rd =
                request.getRequestDispatcher("WEB-
                INF/views/retrieve.jsp");
                rd.forward(request, response);
            }else {
                RequestDispatcher rd =
                request.getRequestDispatcher("login.jsp");
                rd.forward(request, response);
            }
        }
}
```

homeController

```
@WebServlet("/homeController")
public class HomeController extends HttpServlet {
     private static final long serialVersionUID = 1L;
    public HomeController() {
        super();
    }
     protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException {
            HttpSession session = request.getSession(false);
            if (session.getAttribute("email") != null) {
```

```
                RequestDispatcher rd =
request.getRequestDispatcher("WEB-INF/views/home.jsp");
                rd.include(request, response);
          }else {
                RequestDispatcher rd =
request.getRequestDispatcher("login.jsp");
                rd.forward(request, response);
          }
     }
     protected void doPost(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException {

     }
}
```

Com.web_app_Models

DAOService.Java

```
import java.sql.ResultSet;

public interface DAOService {
     public void connectDB();
     public boolean loginDB(String email, String password);
     public void createDB(String fname, String lname, String
     email, String city, String mobile);
     public ResultSet retrieveDB();
     public void deleteDB(String fname);
     public void updateDB(String fname, String email, String
     city, String mobile);
}
```

DAOServiceImlp.Java

```
public class DAOServiceImpl implements DAOService{
     private Connection con;
     private Statement stmnt;
     private ResultSet result;
     @Override
     public void connectDB() {
          try {
                Class.forName("com.mysql.jdbc.Driver");
                con =
                DriverManager.getConnection("jdbc:mysql://local
                host:3306/login_app","root","password");
                stmnt = con.createStatement();
```

```
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @Override
    public boolean loginDB(String email, String password) {
        try {
            result = stmnt.executeQuery("SELECT * FROM
            login WHERE email='"+email+"' AND
            password='"+password+"'");
            return result.next();
        } catch (Exception e) {
            e.printStackTrace();
        }return false;
    }

    @Override
    public void createDB(String fname, String lname, String
    email, String city, String mobile) {
        try {
            stmnt.executeUpdate("INSERT INTO registration
            VALUES('"+fname+"','"+lname+"','"+email+"','"+c
            ity+"','"+mobile+"')");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @Override
    public ResultSet retrieveDB() {
        try {
            result = stmnt.executeQuery("SELECT * FROM
            registration");
            return result;
        } catch (Exception e) {
            e.printStackTrace();
        }return null;
    }

    @Override
    public void deleteDB(String fname) {
        try {
            stmnt.executeUpdate("DELETE FROM registration
            WHERE fname='"+fname+"'");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @Override
```

```
     public void updateDB(String fname, String email, String
city, String mobile) {
          try {
               stmnt.executeUpdate("UPDATE registration SET
               email='"+email+"',city='"+city+"',mobile='"+mob
               ile+"' WHERE fname='"+fname+"'");
          } catch (Exception e) {
               e.printStackTrace();
          }
     }
}
```

WEBAPP Login.jsp Page

```
<%@ page language="java" contentType="text/html; charset=ISO-
8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<link rel="stylesheet" href="css/loginStyle.css">
<title>Login Page</title>
</head>
<body>
     <form action="loginController" method="post">
     <div class="container">
     <h1>Company Login Page</h1>
     <p>Please enter your email and password.</p>
    <hr>

          <label><b>Email</b></label>
          <input type="text" placeholder="Enter Email"
           name="email" required>

          <label><b>Password</b></label>
          <input type="text" placeholder="Enter Password"
           name="password" required>

     <hr>
          <button type="submit"
          class="loginbtn">Login</button>

      <hr>
          <p>
          <%
               if (request.getAttribute("msg") != null){
                    out.println(request.getAttribute("msg"));
               }
```

```
            %>
            </p>
        </div>

</form>
</body>
</html>
```

WEBAPP CSS Folder

LoginStyle.css

```
@charset "ISO-8859-1";
body {
  font-family: Arial, Helvetica, sans-serif;
  background-color: white;
}

* {
  box-sizing: border-box;
}

/* Add padding to containers */
.container {
  padding: 16px;
  background-color: white;
}

/* Full-width input fields */
input[type=text], input[type=password] {
  width: 100%;
  padding: 15px;
  margin: 5px 0 22px 0;
  display: inline-block;
  border: none;
  background: #f1f1f1;
}

input[type=text]:focus, input[type=password]:focus {
  background-color: #ddd;
  outline: none;
}

/* Overwrite default styles of hr */
hr {
  border: 1px solid #f1f1f1;
  margin-bottom: 25px;
}

/* Set a style for the submit button */
.loginbtn {
```

```
  background-color: #04AA6D;
  color: white;
  padding: 16px 20px;
  margin: 8px 0;
  border: none;
  cursor: pointer;
  width: 35%;
  opacity: 0.9;
}

.loginbtn:hover {
  opacity: 1;
}

/* Add a blue text color to links */
a {
  color: dodgerblue;
}
```

createStyle.css

```
@charset "ISO-8859-1";
body {
  font-family: Arial, Helvetica, sans-serif;
  background-color: white;
}

* {
  box-sizing: border-box;
}

/* Add padding to containers */
.container {
  padding: 16px;
  background-color: white;
}

/* Full-width input fields */
input[type=text], input[type=password] {
  width: 100%;
  padding: 15px;
  margin: 5px 0 22px 0;
  display: inline-block;
  border: none;
  background: #f1f1f1;
}

input[type=text]:focus, input[type=password]:focus {
  background-color: #ddd;
  outline: none;
}
```

```css
/* Overwrite default styles of hr */
hr {
  border: 1px solid #f1f1f1;
  margin-bottom: 25px;
}

/* Set a style for the submit button */
.registerbtn {
  background-color: #04AA6D;
  color: white;
  padding: 16px 20px;
  margin: 8px 0;
  border: none;
  cursor: pointer;
  width: 45%;
  opacity: 0.9;
}

.registerbtn:hover {
  opacity: 1;
}

/* Add a blue text color to links */
a {
  color: dodgerblue;
}
```

MenuStyle.css

```css
@charset "ISO-8859-1";
ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
  overflow: hidden;
  background-color: #333;
}

li {
  float: left;
}

li a {
  display: block;
  color: white;
  text-align: center;
  padding: 14px 16px;
  text-decoration: none;
```

```
}

li a:hover {
  background-color: #111;
}
```

RetrieveStyle.css

```
@charset "ISO-8859-1";
#customers {
  font-family: Arial, Helvetica, sans-serif;
  border-collapse: collapse;
  width: 100%;
}

#customers td, #customers th {
  border: 1px solid #ddd;
  padding: 8px;
}

#customers tr:nth-child(even){background-color: #f2f2f2;}

#customers tr:hover {background-color: #ddd;}

#customers th {
  padding-top: 12px;
  padding-bottom: 12px;
  text-align: left;
  background-color: #04AA6D;
  color: white;
}
```

UpdateSyle.css

```
@charset "ISO-8859-1";
body {
  font-family: Arial, Helvetica, sans-serif;
  background-color: white;
}

* {
  box-sizing: border-box;
}

/* Add padding to containers */
.container {
  padding: 16px;
  background-color: white;
}
```

```css
/* Full-width input fields */
input[type=text], input[type=password] {
  width: 100%;
  padding: 15px;
  margin: 5px 0 22px 0;
  display: inline-block;
  border: none;
  background: #f1f1f1;
}

input[type=text]:focus, input[type=password]:focus {
  background-color: #ddd;
  outline: none;
}

/* Overwrite default styles of hr */
hr {
  border: 1px solid #f1f1f1;
  margin-bottom: 25px;
}

/* Set a style for the submit button */
.updatebtn {
  background-color: #04AA6D;
  color: white;
  padding: 16px 20px;
  margin: 8px 0;
  border: none;
  cursor: pointer;
  width: 45%;
  opacity: 0.9;
}

.updatebtn:hover {
  opacity: 1;
}

/* Add a blue text color to links */
a {
  color: dodgerblue;
}
```

VIEWS Folder JSP Page

Home.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ include file="menu.jsp" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Home Page</title>
</head>
<body>
     <form action="homeController" method="post">

          <img src="band.jpg" alt="The Band"
          style="width:100%">
          <h1 class="w3-center">Linkin Park</h1>

          <p>Linkin Park is an American rock band from Agoura
          Hills, California. The band's current lineup
          comprises vocalist/rhythm guitarist/keyboardist Mike
          Shinoda, lead guitarist Brad Delson, bassist
          Dave Farrell, DJ/turntablist Joe Hahn and drummer
          Rob Bourdon, all of whom are founding members.
          Vocalists Mark Wakefield and Chester Bennington are
          former members of the band. </p>

          <p>Formed in 1996, Linkin Park rose to international
          fame with their debut studio album, Hybrid Theory
          (2000), which became certified Diamond by the
          Recording Industry Association of America (RIAA).
          Released during the peak of the nu metal scene, the
          album's singles' heavy airplay on MTV led the
          singles "One Step Closer", "Crawling" and "In the
          End" all to chart highly on the Mainstream Rock
          chartp </p>

     </form>
</body>
</html>
```

Create.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-
8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ include file="menu.jsp" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<link rel="stylesheet" href="css/createStyle.css">
<title>Registration Page</title>
</head>
<body>
     <form action="createController" method="post">
     <div class="container">
     <h1>Registration Open</h1>
     <p>Please enter your details correctly.</p>
    <hr>

         <label><b>First Name</b></label>
         <input type="text" placeholder="Enter First Name"
          name="fname" required>


         <label><b>Last Name</b></label>
         <input type="text" placeholder="Enter Last Name"
          name="lname" required>

         <label><b>Email</b></label>
         <input type="text" placeholder="Enter Email ID"
          name="email" required>

         <label><b>City</b></label>
         <input type="text" placeholder="Enter City Name"
          name="city" required>


         <label><b>Mobile</b></label>
         <input type="text" placeholder="Enter Mobile No"
          name="mobile" required>

     <hr>
          <button type="submit"
          class="registerbtn">Registration</button>

      <hr>
          <p>
           <%
               if (request.getAttribute("msg")!=null){
                    out.println(request.getAttribute("msg"));
```

```
                }
            %>
           </p>
      </div>

      </form>
</body>
</html>
```

Menu.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-
8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<link rel="stylesheet" href="css/menuStyle.css">
<title>Menu Page</title>
</head>
<body>
      <ul>
        <li><a class="active"
           href="homeController">Home</a></li>
        <li><a href="createController">Registration</a></li>
        <li><a href="retrieveController">View
        Registration</a></li>
        <li><a href="logoutController">Logout</a></li>
      </ul>
</body>
</html>
```

Retrieve.jsp

```
<%@page import="java.sql.ResultSet"%>
<%@ page language="java" contentType="text/html; charset=ISO-
8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ include file="menu.jsp" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<link rel="stylesheet" href="css/retrieveStyle.css">
<title>Company Registration Data</title>
</head>
<body>
```

```
      <form action="retrieveController" method="post">
            <h1>A Company Registration Data</h1>
            <table id="customers">
              <tr>
                <th>First Name</th>
                <th>Last Name</th>
                <th>Email</th>
                <th>City</th>
                <th>Mobile</th>
                <th>Delete</th>
                <th>Update</th>
              </tr>
            <% ResultSet result = (ResultSet)
            request.getAttribute("result");
                 while(result.next()){%>

                 <tr>
                      <td><%=result.getString(1) %></td>
                      <td><%=result.getString(2) %></td>
                      <td><%=result.getString(3) %></td>
                      <td><%=result.getString(4) %></td>
                      <td><%=result.getString(5) %></td>
                      <td><a
                           href="deleteController?fname=<%=resul
                           t.getString(1) %>">Delete</a></td>
                      <td><a
                           href="updateController?fname=<%=resul
                           t.getString(1)%>&email=<%=result.getS
                           tring(3)%>&city=<%=result.getString(4
                           )%>&mobile=<%=result.getString(5)%>">
                           Update</a></td>
                 </tr>
                 <% } %>


            </table>
      </form>
</body>
</html>
```

Update.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-
8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ include file="menu.jsp" %>
<!DOCTYPE html>
<html>
<head>
```

```
<meta charset="ISO-8859-1">
<link rel="stylesheet" href="css/updateStyle.css">
<title>Update Page</title>
</head>
<body>
     <form action="updateController" method="post">
     <div class="container">
     <h1>Update Registration</h1>
     <p>Please update your details.</p>
    <hr>

          <label><b>First Name</b></label>
          <input type="text" placeholder="Enter First Name"
           name="fname"
           value=<%=request.getAttribute("fname")%> readonly>

          <label><b>Email</b></label>
          <input type="text" placeholder="Enter Email ID"
           name="email"
           value=<%=request.getAttribute("email")%>>

          <label><b>City</b></label>
          <input type="text" placeholder="Enter City Name"
           name="city" value=<%=request.getAttribute("city")%>>

          <label><b>Mobile</b></label>
          <input type="text" placeholder="Enter Mobile No"
           name="mobile"
           value=<%=request.getAttribute("mobile")%>>

     <hr>
          <button type="submit"
          class="updatebtn">Update</button>
     </div>

     </form>
</body>
</html>
```

PREPAREDSTATEMENT

- Better performance when compare to the traditional approach
- Values can be injected dynamically into SQL statements

MAVENDEPENDENCIES

- Help us to download the project dependencies

Eclipse (Spring Initializer)

- Does not have the option to directly create the spring boot projects
- Spring Initializer create the format for eclipse

JUNITFRAMEWORK

- **I used Junit framework to perform unit testing**
- Old framework
- Junit is frame work that help us to perform unit testing
- Example
  - ```
    import org.junit.jupiter.api.Test;
    public class A {

        @Test
        public void test1() {
            System.out.println("From test 1");
        }
        @Test
        public void test2() {
            int x = 10/0; //error
            ArithmeticException, it will crashed
            System.out.println("From test 2");
        }
    }
    ```

  - **Run evetimes (@before and @after)**

    ```
    import org.junit.After;

    import org.junit.Before;
    import org.junit.Test;

    public class A {
    ```
    ```

```
	@Test
	public void test1() {
		System.out.println("From test 1");
	}

	@Test
	public void test2() {
		System.out.println("From test 2");
	}
	@Before
	public void test3() {
		System.out.println("From before test");
	}
	@After
	public void test4() {
		System.out.println("From after test");
	}
}
```

Output

```
From before test
From test 1
From after test
From before test
From test 2
From after test
```

- **Run only one time**

```
import org.junit.AfterClass;
import org.junit.BeforeClass;
import org.junit.Test;

public class A {

	@Test
	public void test1() {
		System.out.println("From test 1");
	}
	@Test
	public void test2() {
		System.out.println("From test 2");
	}
	@BeforeClass
	public static void beforeclass() {
		System.out.println("Before class");
	}
	@AfterClass
	public static void afterclass() {
		System.out.println("After class");
```

```
    }
}
```

Output

```
Before class
From test 1
From test 2
After class
```

Test Ng

- Popular in testing
- Example

```
import org.testng.annotations.AfterClass;
import org.testng.annotations.AfterTest;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.BeforeTest;
import org.testng.annotations.Test;

public class A {
    @Test
    public void test1(){
        System.out.println("From Test 1");

    }
    @BeforeTest
    public void beforeTest() {
        System.out.println("Before Test ");
    }
    @BeforeClass
    public void beforeClass() {
        System.out.println("Before Class");
    }
    @AfterTest
    public void afterTest() {
        System.out.println("After Test");
    }
    @AfterClass
    public void afterClass() {
        System.out.println("After Class");
    }
    @Test
    public void test2() {
        System.out.println("From Test 2");
    }
}
```

Output

```
Before Test
Before Class
From Test 1
From Test 2
After Class
After Test
```

@DATAPROVIDER

- In which data is kept
- Example

```
import org.testng.annotations.DataProvider;
import org.testng.annotations.Test;

public class B {
    @Test(dataProvider = "getData")
    public void test(String username, String
password) {
        System.out.println(username);
        System.out.println(password);
        System.out.println("_____");
    }
    @DataProvider
    public Object[][] getData(){
        Object[][] obj = new Object[2][2];
        obj [0][0] = "prateek1";
        obj [0][11] = "password1";
        obj [1][0] = "prateek2";
        obj [1][1] = "password2";
        return obj;
    }
  }
```

XMLTAG

- Skipping a particular test by using exclude tag in xml
- Example
    - `import org.testng.annotations.Test;`

```
public class A {
	@Test
	public void test1() {
		System.out.println("From test 1");
	}
	@Test
	public void test2() {
		System.out.println("From test 2");
	}
	@Test
	public void test3() {
		System.out.println("From test 3");
	}
	@Test
	public void test4() {
		System.out.println("From test 4");
	}}
```

- ```
  <?xml version="1.0" encoding="UTF-8"?>
  <!DOCTYPE suite SYSTEM "https://testng.org/testng-
  1.0.dtd">
  <suite name="Suite">
    <test thread-count="5" name="Test">
      <classes>
        <class name="java_testNg_2.A">
            <methods>
                <exclude name="test1"></exclude>
            </methods>
        </class>
      </classes>
    </test> <!-- Test -->
  </suite> <!-- Suite -->
  ```
  ```

JPA (Jakarta Persistence API)

- **JPA** describe the management of relational data in application using java.
- JPA is a concept, standard or is an interface for persistence providers to implements
- Hibernate, Eclipselinks, Toplink, Apache OpenJPA are such implementations of JPA

More about JPA

- JPA is a Jakarta EE application programming interface specification that describes the management of relational data in enterprise java application

HIBERNATEJPA

- Hibernate is an object relational mapping tool (ORM) tool for jaca programming language.
- It provides a framework for mapping object to relational database

Spring boot CRUD operation, using testing module only.

Student.java

```
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Student {
	@Id
	@GeneratedValue(strategy=GenerationType.IDENTITY)
	private long id;
	private String sname;
	private String scourse;
	private int sfee;

	public long getId() {
		return id;
	}
	public void setId(long id) {
		this.id = id;
	}
	public String getSname() {
		return sname;
	}
	public void setSname(String sname) {
		this.sname = sname;
	}
	public String getScourse() {
		return scourse;
	}
	public void setScourse(String scourse) {
		this.scourse = scourse;
	}
	public int getSfee() {
		return sfee;
	}
	public void setSfee(int sfee) {
		this.sfee = sfee;
	}
```

Application properties

```
spring.datasource.url =
jdbc:mysql://localhost:3306/student_app
spring.datasource.username=root
```

```
spring.datasource.password=password
```

Testing

```
@SpringBootTest
class CrudOperationApplicationTests {

	@Autowired
	StudentRepository studentRepo;

	@Test
	void insertStudentRecord() {
		Student student = new Student();
		student.setSname("Prateek");
		student.setScourse("Full Stack");
		student.setSfee(1000);
		studentRepo.save(student);
	}
	@Test
	public void updateStudentRecord() {
		Optional<Student> findById =
		studentRepo.findById(1L);
		Student student = findById.get();
		student.setSfee(5000);
		student.setSname("Aakash");
		studentRepo.save(student);
	}
	@Test
	public void deleteStudentRecord() {
		studentRepo.deleteById(3L);
	}
	@Test
	public void fetchStudentRecord() {
		Optional<Student> findById =
		studentRepo.findById(2L);
		Student student = findById.get();
		System.out.println(student.getId());
		System.out.println(student.getSname());
		System.out.println(student.getScourse());
		System.out.println(student.getSfee());
	}
}
```

SPRINGIOC (Inversion of control)

- It is a core of spring frame work which take cares of objection creation, complete life cycle of objection creation till objection destruction.
- Major task - @Autowired create an object and object destruction, in short life-cycle
- It is a core of spring framework which basically takes care of the life cycle of starting from object creation to objection destruction

SPRINGANNOTATIONS

@Autowired

- It performs dependence injection by creating object during runtime

JPAAnnotations

@Entity

- This annotation helps us to map java class with the database table
- It is present in javax.persistence package and hence it is a JPA implementation

@Table

- This annotation should be used when the entity class name and the database table name are not same

@Id

- This annotation defines which entity class variable is map to the primary key column of database table

@GeneratedValue

- This annotation helps us to auto generate the value and store that into the database table

- Example – Auto generating student id, every time a new record is inserted into the table

@Column

- This annotation should be used, when entity class variable name and the database column name are not same.

Sequence to create web pages

- Database Schema
- Entity Class
- Configure application.properties files
- Create Repository
- Create Controller layer
- Create view

MySQL code

create database **location_db**

use **location_db**

create table **location (**id int **PRIMARY KEY , c**ode varchar(45), name varchar(45), type varchar(45))

select * from **location**

Embed

```
<dependency>
      <groupId>org.apache.tomcat.embed</groupId>
      <artifactId>tomcat-embed-jasper</artifactId>
       <scope>provided</scope>
</dependency>
```

What is data hiding

- Encapsulation is use to hide the values or state of structured data object inside a class

Runtime Polymorphism

- Runtime polymorphism or dynamic method dispatch is a process in which a call to an overridden method is called through the reference variable of a superclass

SPRING TOOL SUITE

Project Starts with MySQL Driver, Spring Data JPA, Spring Web

POM.xml

```
<dependency>
	<groupId>org.apache.tomcat.embed</groupId>
	<artifactId>tomcat-embed-jasper</artifactId>
	<scope>provided</scope>
</dependency>

<dependency>
	<groupId>mysql</groupId>
	<artifactId>mysql-connector-java</artifactId>
	<scope>runtime</scope>
</dependency>

<dependency>
	<groupId>javax.servlet</groupId>
	<artifactId>jstl</artifactId>
	<scope>provided</scope>
</dependency>
```

Application.Properties

```
spring.datasource.url=jdbc:mysql://localhost:3306/location_db
spring.datasource.username=root
spring.datasource.password=password

server.servlet.context-path=/location_app
spring.mvc.view.prefix=/WEB-INF/jsps/
spring.mvc.view.suffix=.jsp
```

Com.project.Entities

Location.java

```
import javax.persistence.Entity;
import javax.persistence.Id;

@Entity
public class Location {
	@Id
	private long id;
	private String code;
	private String name;
	private String type;

	public long getId() {
		return id;
	}
	public void setId(long id) {
		this.id = id;
	}
	public String getCode() {
		return code;
	}
	public void setCode(String code) {
		this.code = code;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public String getType() {
		return type;
	}
	public void setType(String type) {
		this.type = type;
	}}
```

Com.project.Repositories

LocationRepository.java (Interface)

```
package com.project_one.Repositories;

import org.springframework.data.jpa.repository.JpaRepository;

import com.project_one.Entities.Location;

public interface LocationRepository extends
JpaRepository<Location, Long> {

}
```

Com.project.Controllers

LocationController.java

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import com.location_web_app.repositories.LocationRepositories;

@Controller
public class LocationController {

	@Autowired
	LocationRepository locationRepo;

	@RequestMapping("/showLocation")
	public String showLocation() {
		return "showLocation";
	}

	@RequestMapping("/saveLoc")
	public String saveLoc(@ModelAttribute("location")
	Location location, ModelMap modelMap) {

		locationRepo.save(location);

		System.out.println(location.getId());
		System.out.println(location.getCode());
		System.out.println(location.getName());
		System.out.println(location.getType());
```

```
		modelMap.addAttribute("msg","Location Saved");
		return "showLocation";
	}

	@RequestMapping("/viewLocation")
	public String viewLocation(ModelMap modelMap) {

		List<Location> location = locationRepo.findAll();
		modelMap.addAttribute("location",location);
		return "viewLocation";
	}

	@RequestMapping("/deleteLocation")
	public String deleteLocation(@RequestParam("id") Long id,
	ModelMap modelMap){

		locationRepo.deleteById(id);
		List<Location> location = locationRepo.findAll();
		modelMap.addAttribute("location",location);
		return "viewLocation";
	}

	@RequestMapping("/updateLocation")
	public String updateLocation(@RequestParam("id") Long id,
	ModelMap modelMap) {

		Optional<Location> findById =
		locationRepo.findById(id);

		Location location = findById.get();
		modelMap.addAttribute("id",location.getId());
		modelMap.addAttribute("code",location.getCode());
		modelMap.addAttribute("name",location.getName());
		modelMap.addAttribute("type",location.getType());
		return "updateLocation";
	}
	@RequestMapping("/updateLocationData")
	public String
	updateLocationData(@ModelAttribute("location") Location
	location, ModelMap modelMap) {

		location.setCode(location.getCode());
		location.setName(location.getName());
		location.setType(location.getType());
		locationRepo.save(location);
		modelMap.addAttribute("updatedMsg","Location Record
		Is Updated");
		return "updateLocation";
	}
}
```

Src/main/webapp/WEB-INF/jsps

showLocation.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-
8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
     <a href="viewLocation">Click To all Location</a>
     <h2>Location Page</h2>
     <form action="showLoc" method="post">
          ID <input type="text" name="id"/>
          Code <input type="text" name="code"/>
          Name <input type="text" name="name"/>
          Type:
          Urban <input type="radio" name="type"
value="urban"/>
          Rural <input type="radio" name="type"
value="urban"/>
          <input type="submit" value="Save"/>
     </form>
     ${msg}
</body>
</html>
```

viewLocation.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-
8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"
%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>View Page</title>
</head>
<body>
     <a href="showLocation">Click To all Location</a>
     <h2>View Page</h2>
     <table border="1">
          <tr>
               <th>ID</th>
               <th>Code</th>
```

```
                <th>Name</th>
                <th>Type</th>
            </tr>

            <c:forEach items="${location}" var="location">
                <tr>
                    <th>${location.id}</th>
                    <th>${location.code}</th>
                    <th>${location.name}</th>
                    <th>${location.type}</th>
                </tr>
            </c:forEach>

    </table>
</body>
</html>
```

updateLocation.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-
8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"
%>
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Update Page</title>
</head>
<body>
    <a href="viewLocation">Click To all Location</a>
    <form action="updateLocationData" method="post">
    <h2>Update Page</h2>
        ID <input type="text" name="id" value="${id}"
readonly/>
        Code <input type="text" name="code"
value="${code}"/>
        Name <input type="text" name="name"
value="${name}"/>
        Type: Urban <input type="radio" name="type"
value="urban" value="${type}"/>
        Rural <input type="radio" name="type" value="rural"
value="${type}"/>
        <input type="submit" value="Update"/>
    </form>
    ${updatedMsg}
</body>
</html>
```

WEBSERVICES

- Web services are used to integrated heterogenous and homogenous applications

Exposing of Web Services

- Application exposing its data is called as exposing of web service

Consuming Web Services

- Application which consumes the data expose by other software's

Difference Between Application Server vs Web Server

Application Server

- If you want to build dynamic application using technology like servlet, jsp, EJB, spring boot etc.
- We use application servers like tomcat, Jboss, Class fish.

Static web server

- If you want to build static application using technology like html, css, JavaScript etc then we use web server like Nginx, IIS, Apache HTPC server.

Maven

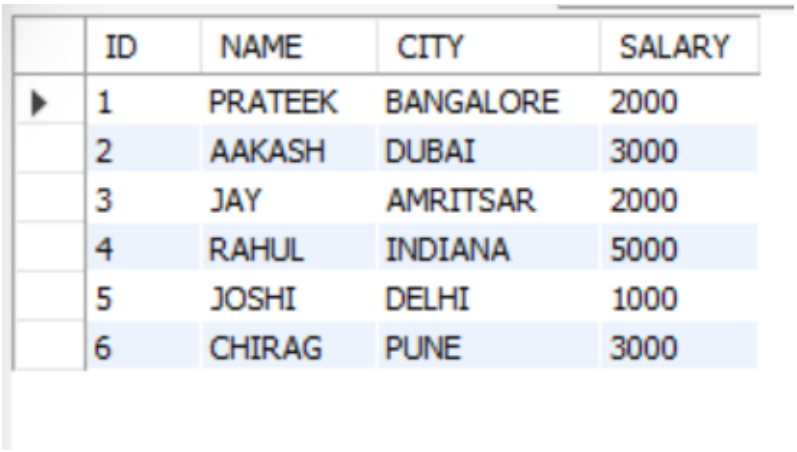
- Help us to download dependence

SQL Notes

CREATE Table EMPLOYEES

CREATE TABLE EMPLOYEES (ID INT (45), NAME VARCHAR (45), CITY VARCHAR (45), SALARY INT (45))

SELECT * FROM EMPLOYEES



ID	NAME	CITY	SALARY
1	PRATEEK	BANGALORE	2000
2	AAKASH	DUBAI	3000
3	JAY	AMRITSAR	2000
4	RAHUL	INDIANA	5000
5	JOSHI	DELHI	1000
6	CHIRAG	PUNE	3000

Select only ID and Name

- SELECT **ID**, **NAME** FROM EMPLOYEES

1 PRATEEK
2 AAKASH
3 JAY
4 RAHUL
5 JOSHI
6 CHIRAG

READ the Employees name whose salary is 1000

- SELECT **NAME** FROM EMPLOYEES **WHERE** SALARY=1000
- Output = **'JOSHI'**

Read the EMPLOYEES name whose Salary is equal or greater than 3000

- SELECT **NAME** FROM EMPLOYEES **WHERE** SALARY>=3000

- Output = 'AAKASH', 'RAHUL', CHIRAG

Employees name whose salary is 5000 and 2000

- SELECT **NAME** FROM EMPLOYEES **WHERE** SALARY **IN** (2000, 5000)
- Output -

PRATEEK
JAY
RAHUL

- **In** Operator is used when, you want to perform equals with more than one data.

Employees Id whose name is Prateek and Joshi

- SELECT **ID** FROM EMPLOYEES **WHERE** NAME **IN** ("PRATEEK", "JOSHI")
- Output -

1
5

Golden Rules Structure

SELECT -→ FROM → WHERE -→ GROUP BY-→ HAVING -→ ORDER BY

- You cannot alter the order, although you can skip the keyword
- For example

SELECT -→ FROM -→ ORDER BY

Employees Id sorted in descending order

- SELECT **ID** FROM EMPLOYEES **ORDER BY** ID DESC
- Output -

6
5
4
3
2
1

- In SQL to **Sort** the data we use **ORDER BY** command. **DESC** for Descending and **ASC** for Ascending

Filter the record using having command, and give employees Id whose name is Jay

- SELECT **ID** FROM EMPLOYEES **HAVING** NAME ="JAY"

Sort the employees name in descending order

- SELECT **NAME** FROM EMPLOYEES **ORDER BY** ID DESC
- Output -

CHIRAG
JOSHI
RAHUL
JAY
AAKASH
PRATEEK

Sort the employees Id in ascending order

- SELECT **ID** FROM EMPLOYEES **ORDER BY** ID ASC

Built-in Function in SQL

Give me the max salary from the table

- SELECT **MAX**(SALARY) FROM EMPLOYEES
- **Output – 5000**

Minimum salary from the employee

- SELECT **MIN**(SALARY) FROM EMPLOYEES
- **Output** – 1000

Give the Average of all the salaries

- **SE**LECT **AVG**(SALARY) FROM EMPLOYEES
- Output **- 2666.6667**

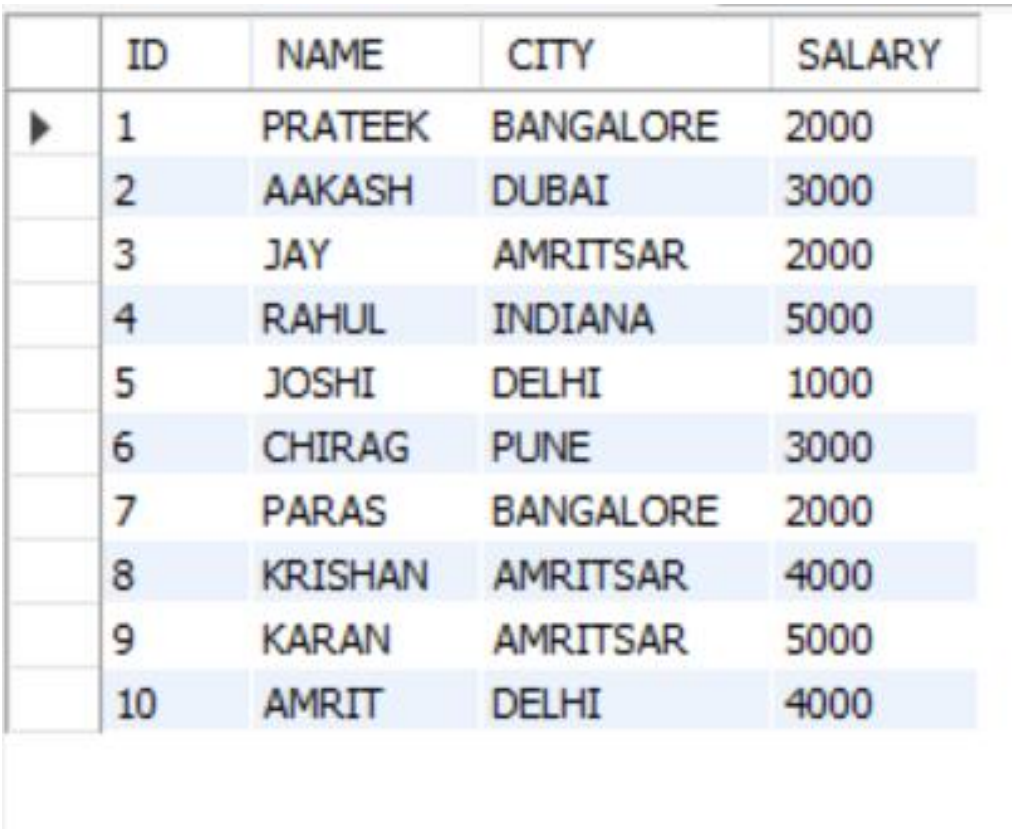
Sum of all salaries

- SELECT **SUM**(SALARY) FROM EMPLOYEES
- **Output** - 16000

Count the number of employees in your company

- SELECT **COUNT**(ID) FROM EMPLOYEES
- Output - 6

NEW TABLE EMPLOYEES



ID	NAME	CITY	SALARY
1	PRATEEK	BANGALORE	2000
2	AAKASH	DUBAI	3000
3	JAY	AMRITSAR	2000
4	RAHUL	INDIANA	5000
5	JOSHI	DELHI	1000
6	CHIRAG	PUNE	3000
7	PARAS	BANGALORE	2000
8	KRISHAN	AMRITSAR	4000
9	KARAN	AMRITSAR	5000
10	AMRIT	DELHI	4000

Count number of employees city wise

- SELECT **COUNT**(ID), NAME, CITY FROM EMPLOYEES **GROUP BY** CITY
- **Output** -

2	PRATEEK	BANGALORE
1	AAKASH	DUBAI
3	JAY	AMRITSAR
1	RAHUL	INDIANA
2	JOSHI	DELHI
1	CHIRAG	PUNE

Group the employees based on city and then count city wise and print the count in ascending order.

- SELECT **COUNT**(CITY), NAME, CITY FROM EMPLOYEES **GROUP BY** CITY **ORDER BY** COUNT(ID) **ASC**

- Output-

1	AAKASH	DUBAI
1	RAHUL	INDIANA
1	CHIRAG	PUNE
2	PRATEEK	BANGALORE
2	JOSHI	DELHI
3	JAY	AMRITSAR

Count number of cites and sort the count in ascending manner

- SELECT **COUNT**(CITY), CITY FROM EMPLOYEES GROUP BY CITY **ORDER BY** CITY **ASC**
- Output –

3	AMRITSAR
2	BANGALORE
2	DELHI
1	DUBAI
1	INDIANA
1	PUNE

Group the candidates based on their age

- **SELECT count**(age) **FROM** table **GROUP BY** age;
- **GROUP BY –** if 12 people age is 24, it will group 12 people into one group

Group the city after counting where the city name is Bangalore

- **S**ELECT **COUNT**(CITY), CITY FROM EMPLOYEES **GROUP BY** CITY **HAVING** CITY="BANGALORE"
- Output- 2 BANGALORE

Convert all the Employees name in Upper Case and Lowe Case

- **SELECT Ucase**(name) **FROM** employees
- **SELECT Lcase**(name) **FROM** employees

- **Output -**

prateek
aakash
jay
rahul
joshi
chirag
paras
krishan
karan
amrit

Top two Record

- SELECT * FROM EMPLOYEES ORDER BY ID ASC **LIMIT** 2
- Output -

1	PRATEEK	BANGALORE	2000
2	AAKASH	DUBAI	3000

First record from the table

- SELECT * FROM EMPLOYEES **ORDER BY** ID **ASC** LIMIT 1
- Output 1 PRATEEK BANGALORE 2000

‘

Last two record

- SELECT * FROM EMPLOYEES **ORDER BY** ID **DESC** LIMIT 2
- Output -

10	AMRIT	DELHI	4000
9	KARAN	AMRITSAR	5000

In city column show only three letters

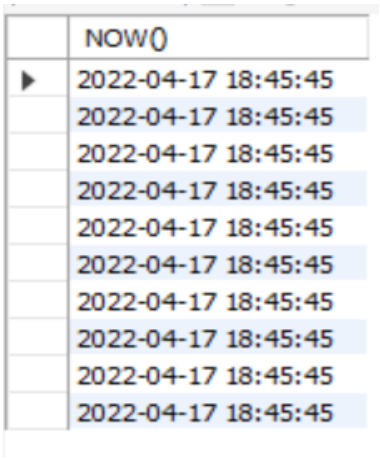
- SELECT **MID**(CITY,**1,3**) FROM EMPLOYEES

- Output -

BAN
DUB
AMR
IND
DEL
PUN
BAN
AMR
AMR
DEL

Print current system time

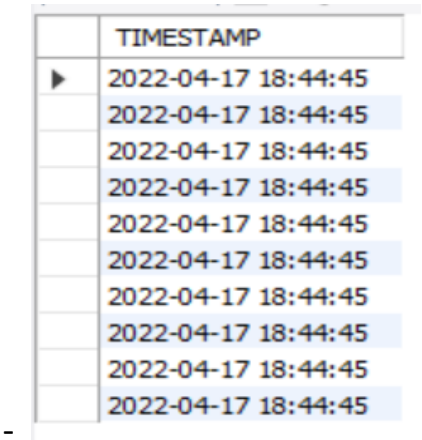
- SELECT **NOW ()** FROM EMPLOYEES
- Output -



	NOW()
▶	2022-04-17 18:45:45
	2022-04-17 18:45:45
	2022-04-17 18:45:45
	2022-04-17 18:45:45
	2022-04-17 18:45:45
	2022-04-17 18:45:45
	2022-04-17 18:45:45
	2022-04-17 18:45:45
	2022-04-17 18:45:45
	2022-04-17 18:45:45

Print timestamp AND change the name Using AS

- SELECT **NOW** () **AS** TIMESTAMP FROM EMPLOYEES
- Output



	TIMESTAMP
▶	2022-04-17 18:44:45
	2022-04-17 18:44:45
	2022-04-17 18:44:45
	2022-04-17 18:44:45
	2022-04-17 18:44:45
	2022-04-17 18:44:45
	2022-04-17 18:44:45
	2022-04-17 18:44:45
	2022-04-17 18:44:45
	2022-04-17 18:44:45

-

Wild Cards

- LIKE (“%A”, “A%” OR “%A%”)
- “%A” – END Letter
- “A%” – Start Letter
- “%A%” – Every Letter
- UNDER SCORE _ _ _

Name ends with N

- SELECT **NAME** FROM EMPLOYEES **WHERE** NAME **LIKE** "%N"
- Output KRISHAN
  KARAN

Name of the employees which consist of letter P

- SELECT **NAME** FROM EMPLOYEES **WHERE** NAME **LIKE** "%P%"
- Output – PRATEEK
  PARAS

Name of employees that starts with letter R

- SELECT **NAME** FROM EMPLOYEES **WHERE** NAME **LIKE** "R%"
- Output - RAHUL

Name of the employees which consist of three letters

- SELECT **NAME** FROM EMPLOYEES **WHERE** NAME **LIKE** "_ _ _"
- Output - JAY

Employees name which consists of five letters

- SELECT **NAME** FROM EMPLOYEES **WHERE** NAME **LIKE** "_ _ _ _ _"

- Output -

RAHUL
JOSHI
PARAS
KARAN
AMRIT

Employees name ends with letter Y and consist of three letters

- SELECT **NAME** FROM EMPLOYEES WHERE **NAME LIKE** "__Y"
- Output – JAY

Delete a record from employees record where ID = 7

- **DELETE** FROM EMPLOYEES **WHERE** ID = 7

Delete a record from employees where ID is 6 and 3

- **DELETE** FROM EMPLOYEES **WHERE** ID **IN** (3,6)

Delete a record from employees where NAME is Prateek

- **DELETE** FROM EMPLOYEES **WHERE** NAME="PRATEEK"

DROP the table employees

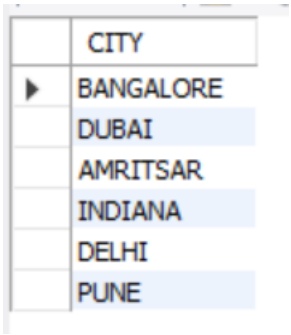
- **DROP** TABLE EMPLOYEES

DISTINCT Record from table

- Remove duplicates

Print only unique city name from the table

- SELECT **DISTINCT** (CITY) **AS** CITY FROM EMPLOYEES
- Output



	CITY
▶	BANGALORE
	DUBAI
	AMRITSAR
	INDIANA
	DELHI
	PUNE

Distinct salary from employees table in ascending order

- SELECT **DISTINCT** (SALARY) FROM EMPLOYEES **ORDER BY** SALARY **ASC**

- Output - 1000
  2000
  3000
  4000
  5000

Concat employees name and salary and then print output

- SELECT **CONCAT** (NAME, SALARY) FROM EMPLOYEES

‘

Separate employees name and salary with an under-score and then Concat the content of these follows

- SELECT **CONCAT** (NAME,"_", SALARY) FROM EMPLOYEES

Remove white space from the left side and the right side of employee’s name and then print name

- LTRIM REMOVE LEFT SIDE
- RTRIM REMOVE RIGHT SIDE
- **TRIM ROMOVE BOTH SIDES**
- SELECT LTRIM(NAME) FROM EMPLOYEES

- SELECT RTRIM(NAME) FROM EMPLOYEES
- SELECT **TRIM**(NAME) FROM EMPLOYEES

Remove the white space from name, and salary from both left and right side and then Concat them separated with underscore.

- SELECT **CONCAT**(**TRIM**(NAME), "_", **TRIM**(SALARY)) **AS** NAME_SALARY FROM EMPLOYEES **LIMIT** 4
- Output

	NAME_SALARY
▶	PRATEEK_2000
	AAKASH_3000
	JAY_2000
	RAHUL_5000

Concat the name and salary and print the output in descending order

- SELECT **CONCAT** (NAME, "_", SALARY) **AS** NAME_SALARY FROM EMPLOYEES **ORDER BY CONCAT** (NAME, SALARY) **DESC LIMIT** 4
- Output

RAHUL_5000
PRATEEK_2000
PARAS_2000
KRISHAN_4000

Sort the employees name in ascending order

- SELECT NAME FROM EMPLOYEES **ORDER BY** NAME **ASC LIMIT** 5
- Output

AAKASH
AMRIT
CHIRAG
JAY
JOSHI

Update the record where employees name is Jay to Mritunjay

- **UPDATE** EMPLOYEES **SET** NAME = 'MRITUNJAY' **WHERE** NAME='JAY'
- Output - 'MRITUNJAY'

Update the employee’s name Joshi to Nitish where ID is 5

- **UPDATE** EMPLOYEES **SET** NAME = 'NITISH' **WHERE** ID=5
- Output - NITISH

Update ID 6 to 7

- **UPDATE** EMPLOYEES **SET** ID=7 **WHERE** ID=6
- Output – ID 7 become 6

Replace the city Bangalore to Bengaluru in the table

- **UPDATE** EMPLOYEES **SET** CITY="BENGALURU" **WEHRE** CITY="BANAGLORE"

Delete a record from the table where ever your find Karan

- **DELETE** FROM EMPLOYEES **WEHER** NAME="KARAN"
- WHERE finds KARAN, it deleted from the table

Delete the record from the table whose ever salary is 3000

- **DELETE** FROM EMPLOYEES **WHERE** SALARY=3000

Update everyone salary to 100

- **UPDATE** EMPLOYEES **SET** SALARY=**SALARY**+200
- Output - Everybody salary increase by 200

Decrease everyone salary to 500

- **UPDATE** EMPLOYEES **SET** SALARY=**SALARY**-500

Add a column to the existing table

- **ALTER** TABLE EMPLOYEES **ADD** EMAIL VARCHAR (45)
- OUTPUT-

	ID	NAME	CITY	SALARY	EMAIL
▶	1	PRATEEK	BANGALORE	2000	NULL
	2	AAKASH	DUBAI	3000	NULL
	3	JAY	AMRITSAR	2000	NULL
	4	RAHUL	INDIANA	5000	NULL
	5	JOSHI	DELHI	1000	NULL

Add a column at first position

- **ALTER** TABLE EMPLOYEES **ADD** EMAIL VARCHAR (45) **FIRST**;

Add the column email after name column

- **ALTER** TABLE EMPLOYEES **ADD** EMAIL VARCHAR (45) **AFTER** NAME;

Delete a column email from the table

- **ALTER** TABLE EMPLOYEES **DROP** EMAIL

Delete a column with the called NAME

- **ALTER** TABLE EMPLOYEES **DROP** NAME

Find the length of each and every employee's name in the table

- **SELECT LENGTH**(NAME), NAME **FROM** EMPLOYEES
- NOTE – White space also calculate.
- Output

3	JAY
5	RAHUL
5	JOSHI
6	AAKASH
7	PRATEEK

INTERVIEW QUESTION

TABLE

	ID	NAME	CITY	SALARY
▶	1	PRATEEK	BANGALORE	2000
	2	AAKASH	DUBAI	3000
	3	JAY	AMRITSAR	2000
	4	RAHUL	INDIANA	5000
	5	JOSHI	DELHI	1000
	6	CHIRAG	PUNE	3000
	7	PARAS	BANGALORE	2000
	8	KRISHAN	AMRITSAR	4000
	9	KARAN	AMRITSAR	5000
	10	AMRIT	DELHI	4000

Sub-query – Find out second max salary

- SELECT **MAX**(SALARY) **AS** SECOND_MAX_SALARY FROM EMPLOYEES **WHERE** SALARY **<** (SELECT **MAX**(SALARY) FROM EMPLOYEES)
- Output

SECOND_MAX_SALARY
4000

Third Max Salary

- SELECT **MAX**(SALARY) AS THIRD_MAX_SALARY FROM EMPLOYEES WHERE SALARY **< (**SELECT MAX(SALARY) **AS** SECOND_MAX_SALARY FROM EMPLOYEES WHERE SALARY **< (**SELECT MAX(SALARY) **AS** FIRST_MAX_SALARY FROM EMPLOYEES**))**
- Output

THIRD_MAX_SALARY
3000

Employees name who got incentives

- SELECT **DISTINCT**(NAME) FROM EMPLOYEES **WHERE** ID IN (SELECT **ID** FROM INCENTIVE)

Name of employees who are not married

- SELECT **NAME** FROM EMPLOYEES **WHERE ID** IN (SELECT **ID** FROM STATUS **WHERE** MARRITAL STATUS='NO')

Employees name WHO's are female candidates and also their salary

- SELECT **NAME**, **SALARY** FROM EMPLOYEES **WHERE ID IN** (SELECT **ID** FROM STATUS **WHERE** GENDER = "FEMALE")

Employees name who got incentive amount greater than or equal to 5000

- SELECT **NAME** FROM EMPLOYEES **WHERE ID IN** (SELECT **ID** FROM INCENTIVE **WHERE** AMOUNT >= 5000)

Employees name who is married and has got incentives

- SELECT **NAME** FROM EMPLOYEES **WHERE ID IN** (SELECT **ID** FROM INCENTIVES **WHERE ID IN** (SELECT **ID** FROM STATUS **WHERE** MARITAL_STATUS = "YES"))

Marital status of employees whose salary is greater than 10000

- SELECT **MARITIAL_STATUS** FROM STATUS **WHERE ID IN** (SELECT **SALARY** FROM EMPLOYEES **WHERE** SALARY > 10000)

Employees incentive date who's name end with letter A

- SELECT **DATE** FROM INCENTIVES **WHERE** NAME **LIKE** "%A"

Marital Status of people residing in Bangalore

- SELECT **MARITIAL_STATUS** FROM STATUS **WHERE ID IN** (SELECT **ID** FROM EMPLOYEES **WHERE** CITY = “BANGALORE”)

Employees NAME who are married and gender is male

- SELECT **NAME** FROM EMPLOYEES **WHERE ID IN** (SELECT **ID** FROM STATUS WHERE **GENGER** = “MALE”)

Store numbers in database – how much data can be stored

- TINY INT = 120 to -128
- SMALL INT = 32,765 to -32,767
- MEDIUM INT = 8,388608 to -8,388608
- INT = 2^31 - 1

Store Strings in database

- Char (4) – MAX number of letters is 4
- VARCHAR (20) – MAX limit is 20
- BLOB – 2^16 Bytes

Float, Double and Enum (Difference in Memory size)

- FLOAT
- DOUBLE
- ENUM (“F”, “M”) – Group of constants

Datatype of MySQL

- DATE = YYYY/mm/dd (Default time format)
- TIME = HH:MM: SS
- YEAR = YYYY
- TIMESTAMP – DATE AND TIME

Creating Database and table in MySQL

CREATE DATABASE **ANIMAL**

USE **ANIMAL**

CREATE TABLE **PET** (ID INT (45), NAME VARCHAR (45), OWNER VARCHAR (45), BIRTH DATE, GENDER ENUM("F","M"))

CREATE TABLE **REGISTRATION** (ID INT (45), FIRSTNAME VARCHAR (45), LASTNAME VARCHAR (45), DOB DATE, REG_TIME TIMESTAMP, LOCATION VARCHAR (45))

SELECT * FROM **PET**

SELECT * FROM **REGISTRATION**

Constrains on Table (Restrictions)

- NOT NULL – Cannot skip (Mandatory)
- UNIQUE
- PRIMARY KEY
- FORIGN KEY
- AUTO_INCREMENT – Auto updated

NOT NULL

- It you make a column NOT NULL then it means, it can consist of duplicate record but not NULL values

UNIQUE

- It can consist of UNIQUE records and also NULL values, as two NULL values can never be same

PRIMARY KEY

- It is UNIQUE and NOT NULL

ENUM

- It gives us a fixed values to be selected

SET

- A group of values and this constrains helps us to SELECT any ONE value FROM the SET or ALL the values FROM the SET or few values from the SET

Create table STUDENTS with Constrains

```
CREATE TABLE STUDENTS (
  ID INT PRIMARY KEY,
  NAME VARCHAR (45) NOT NULL,
   GENGER ENUM ("F", "M"),
   DOB DATE NOT NULL,
   PHONE VARCHAR (45) UNIQUE,
   LOCATION VARCHAR (45),
   CERTIFICATE SET ("SQL", "PYTHON","JAVA"))
```

Insert into table

```
INSERT INTO STUDENTS VALUES (
1, "PRATEEK", "M", "1918-01-19","123456", "BANGALORE","PYTHON, JAVA")
```

FOREIGN KEY

- It helps us to build the relation between two tables
- It can consist of repeated values

PRIMARY KEY AND FOREIGN KEY

S.No.	Primary Key	Foreign Key
1	A primary key generally focuses on the uniqueness of the table. It assures the value in the specific column is unique.	A foreign key is generally used to build a relationship between the two tables.
2	Table allows only one primary key.	Tables can allow more than one foreign key.
3	The primary key doesn't allow null values.	Foreign key accepts multiple null values.
4	It can identify the record uniquely in the database table.	A foreign key is a field in the table that is the primary key in another table.
5	In the primary key, the value cannot be removed from the parent table.	In this, the value can be deleted from the child table.
6	Its restriction can be completely defined on the temporary tables.	Its restriction cannot be defined on the global or local temporary tables.

PRIMARY KEY AND FOREIGN KEY CONCEPT TABLE

```
CREATE TABLE EMPLOYEE (
        ID INT AUTO_INCREMENT,
        NAME VARCHAR (45) NOT NULL,
        DEPT VARCHAR (45) NOT NULL,
        CITY VARCHAR NOT NULL,
        PRIMARY KEY (ID))

CREATE TABLE ATTENDANCE (
        ID INT NOT NULL,
        NAME VARCHAR (45),
```

DATE VARCHAR (45),

FOREIGN KEY(ID) REFERENCES EMPLOYEES(ID))

RDBMS AND DBMS

RDBMS	DBMS
Data stored is in table format	Data stored is in the file format
Multiple data elements are accessible together	Individual access of data elements
Data in the form of a table are linked together	No connection between data
Normalisation is not achievable	There is normalisation
Support distributed database	No support for distributed database
Data is stored in a large amount	Data stored is a small quantity
Here, redundancy of data is reduced with the help of key and indexes in RDBMS	Data redundancy is common
RDBMS supports multiple users	DBMS supports a single user
It features multiple layers of security while handling data	There is only low security while handling data
The software and hardware requirements are higher	The software and hardware requirements are low
Oracle, SQL Server.	XML, Microsoft Access.

Subtracting two tables in MySQL

- SELECT * FROM STUDENTS **WHERE** ID **NOT IN** (SELECT **UNIQUE** ID FROM ATTENDENCE)

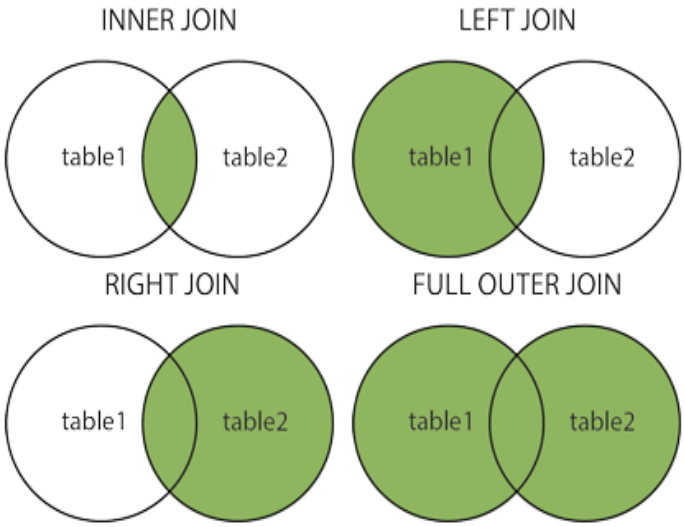
What do you mean my CRUD Operations?

- Create
- Retrieve
- Update
- Delete

JOIN

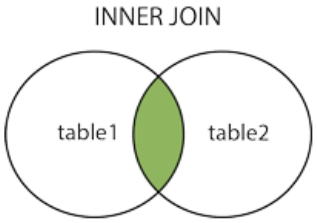
JOIN (Inner join, Left join, Right join)

- If you want to displace the content of two columns that belongs to different tables than we use JOIN operation
- Example



INNER JOIN

- INNER JOIN keyword selects all rows from both sides as long as there is match between the columns.
- Give matching record from both the table
- SELECT **EMPLOYEES**.NAME, **EMPLOYEES**.DEPT, **ATTENDANCE**.DATE FROM EMPLOYEES **INNER JOIN** ATTENDANCE **ON EMPLOYEES**.ID = **ATTENDANCE**.ID
- Example



Matching record between two table and display the class location and employees id

- SELECT **EMPLOYEES**.ID, **EMPLOYEES**.LOCATION, **ATTENEDANCE**.CLASS FROM EMPLOYEES **INNER JOIN** ATTENDACE ON **EMPLOYEES**.ID = **ATTENDANCE**.ID

Subtract two table and give all the record of employees

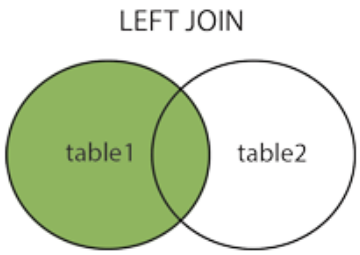
- USED **NOT IN** when subtract question are asked.

Employees id, name, dept, location, attendance_date and class only if record is common between two table

- SELECT **EMPLOYEES**.ID, **EMPLOYEES**.NAME, **EMPLOYEES**.DEPT, **EMPLOYEES**.LOCATION, **ATTENDANCE**.ATTENDEACE_DATE, **ATTENDANCE**.CLASS **WHERE** EMPLOYEES **INNER JOIN** ATTENDANCE ON **EMPLOYEES**.ID = **ATTENDANCE**.ID

LEFT JOIN

- Gives all the record of left table and only matching record from right table
- Example



Employees name, dept and attendance_date such that all the record are displaced from left table and only matching record are displaced from right table

- SELECT **EMPLOYEES**.NAME, **EMPLOYEES**.DEPT, **ATTENDANCE**.ATTENDANCE_DATE FROM EMPLOYEES **LEFT JOIN** ATTENDANCE **ON EMPLOYEES**.ID = **ATTENDANCE**.ID

Employees id, dept and class such that all the record are displaced from the left table and only matching record from right table

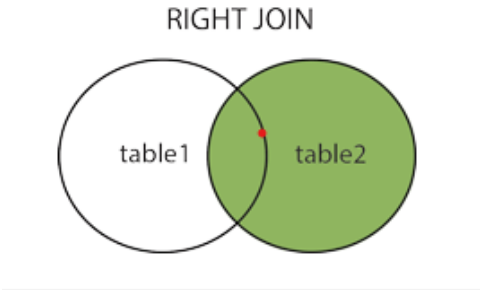
- SELECT **EMPLOYEES**.ID, **EMPLOYEES**.DEPT, **ATTENDANCE**.CLASS FROM EMPLOYEES **LEFT JOIN** ATTENDANCE **ON EMPLOYEES**.ID = **ATTENDANCE**.ID

RECORDS from table t1 and only matching record from the table t2 and output should consist of B, C, D

- SELECT **T1**.B, **T1**.C, **T2**.D FROM T1 **LEFT JOIN** T2 ON **T1**.A = **T2**.A

RIGHT JOIN

- Give us all the record from right table and only matching record from left table.
- Example



Employees name, dept, attendance_date such that all the record are pick up from right table and only matching record from the left table

- SELECT **EMPLOYEES**.NAME, **EMPLOYEES**.DEPT, **ATTENDANCE**.ATTENDANCE_DATE FROM EMPLOYEES **RIGHT JOIN** ATTENDANCE **ON EMPLOYEES**.ID = **ATTENDANCE**.ID

Attendance_date, class and employees name such that all the details of attendance should be taken into consideration but then only matching record of employees should be pick up

- SELECT **ATTENDANCE**.ATTENDANCE_DATE, **ATTENDANCE**.CLASS, **EMPLOYEES**.DEPT FROM EMPLOYEES RIGHT JOIN ATTENDANCE

Cartesian Join

- It means every single record of table 1 will be multiply with table 2

Oracle

What is Rollback and commit

COMMIT – in SQL is a transaction control language that is used to permanently save the change done in the transaction in table or database.

ROLLBACK - in SQL is a transactional control language that is used to undo the transactions that have not been saved in the database. The command is only been used to undo changes since the last COMMIT.

	COMMIT	ROLLBACK
1.	COMMIT permanently saves the changes made by the current transaction.	ROLLBACK undo the changes made by the current transaction.
2.	The transaction can not undo changes after COMMIT execution.	Transaction reaches its previous state after ROLLBACK.
3.	When the transaction is successful, COMMIT is applied.	When the transaction is aborted, incorrect execution, system failure ROLLBACK occurs.
4.	COMMIT statement permanently save the state, when all the statements are executed successfully without any error.	In ROLLBACK statement if any operations fail during the completion of a transaction, it cannot permanently save the change and we can undo them using this statement.
5.	**Syntax of COMMIT statement are:** COMMIT;	**Syntax of ROLLBACK statement are:** ROLLBACK;

What is normalisation and explain 1NF, 2NF AND 3NF

Normalization is a database design technique that reduces data redundancy and eliminates undesirable characteristics like Insertion, Update and Deletion Anomalies

1NF – First Normal Form

2NF – Second Normal Form

3NF – Third Normal Form

NAME of the employees whose birthday is in the year 2000

- SELECT NAME FROM EMPLOYEES WHERE YEAR (DOB)==2000

YEAR, MONTH AND DAY

- SELECT YEAR (NOW ())
- SELECT MONTH (NOW ())
- SELECT DAY (NOW ())

Union

- The UNION operator is used to combine the result-set of two or more select statement

1. Difference between array and arraylist

Definition	An **array** is a dynamically-created object. It serves as a container that holds the constant number of values of the same type. It has a contiguous memory location.	The **ArrayList** is a class of Java **Collections** framework. It contains popular classes like **Vector, HashTable**, and **HashMap**.
Static/ Dynamic	Array is **static** in size.	ArrayList is **dynamic** in size.
Resizable	An array is a **fixed-length** data structure.	ArrayList is a **variable-length** data structure. It can be resized itself when needed.
Initialization	It is mandatory to provide the size of an array while initializing it directly or indirectly.	We can create an instance of ArrayList without specifying its size. Java creates ArrayList of default size.
Performance	It performs **fast** in comparison to ArrayList because of fixed size.	ArrayList is internally backed by the array in Java. The resize operation in ArrayList slows down the performance.
Primitive/ Generic type	An array can store both **objects** and **primitives** type.	We cannot store **primitive** type in ArrayList. It automatically converts primitive type to object.
Iterating Values	We use **for** loop or **for each** loop to iterate over an array.	We use an **iterator** to iterate over ArrayList.
Type-Safety	We cannot use generics along with array because it is not a convertible type of array.	ArrayList allows us to store only **generic/ type, that's why it is type-safe.**
Length	Array provides a **length** variable which denotes the length of an array.	ArrayList provides the **size()** method to determine the size of ArrayList.

Adding Elements	We can add elements in an array by using the **assignment** operator.	Java provides the **add()** method to add elements in the ArrayList.
Single/ Multi-Dimensional	Array can be **multi-dimensional**.	ArrayList is always **single-dimensional**.

2. Difference between arraylist and linkedlist

Sr. No.	Key	ArrayList	LinkedList
1	Internal Implementation	ArrayList internally uses a dynamic array to store its elements.	LinkedList uses Doubly Linked List to store its elements.
2	Manipulation	ArrayList is slow as array manipulation is slower.	LinkedList is faster being node based as not much bit shifting required.
3	Implementation	ArrayList implements only List.	LinkedList implements List as well as Queue. It can acts as a queue as well.
4	Access	ArrayList is faster in storing and accessing data.	LinkedList is faster in manipulation of data.

3. Difference between hashset and linkedhashset

Property	HashSet	LinkedHashSet
Data structure	It uses a Hashtable to store the elements.	It uses a HashTable and doubly linked list to store and maintain the insertion order of the elements.

Technique to store the elements	Hashing	Hashing
Insertion Order	It does not provide any insertion order. We can not predict the order of elements.	It provides an insertion order; we can predict the order of elements.
Null elements	It allows only one null element.	It also allows only one null element.
Memory	It requires less memory.	It requires more memory than HashSet.
Performance	It provides slightly faster performance than LinkedHashSet	It provides low performance than HashSet
Synchronized	Non-synchronized	Non-synchronized
Complexity for the insertion, removal, retrieval operations	O (1)	O (1)
Declaration	HashSet obj = new HashSet();	LinkedHashSet obj = new LinkedHashSet();
Extends	AbstractSet class	HashSet class
Implements	Set interface	Set interface
Initial Capacity	16	16
Package	java.util	Java.util

Difference Between HashSet and TreeSet

Parameters	HashSet	TreeSet
Ordering or Sorting	It does not provide a guarantee to sort the data.	It provides a guarantee to sort the data. The sorting depends on the supplied Comparator.
Null Objects	In HashSet, **only an element** can be null.	It does not allow null elements.
Comparison	It uses **hashCode()** or **equals()** method for comparison.	It uses **compare()** or **compareTo()** method for comparison.
Performance	It is **faster** than TreeSet.	It is **slower** in comparison to HashSet.
Implementation	Internally it uses **HashMap** to store its elements.	Internally it uses **TreeMap** to store its elements.
Data Structure	HashSet is backed up by a hash table.	TreeSet is backed up by a Red-black Tree.
Values Stored	It allows only **heterogeneous** value.	It allows only **homogeneous** value.

Difference Between JDK, JRE, and JVM

JDK	JRE	JVM
JDK stands for Java Development Kit.	JRE stands for Java Runtime Environment	JVM stands for Java Virtual Machine
It's used to develop applications in Java	It provides Java class libraries and components to run Java code.	It executes Java byte code and provides an environment for it.
It is a platform-dependent	It is a platform-dependent	It is a platform-dependent
It primarily assists in executing codes.	It assists in creating an environment for code execution.	It is responsible for providing all these implementations.
JDK is responsible for prime development containing tools like debugging and monitoring java applications.	JRE contains class libraries and other supporting files that JVM requires to run the program.	It does not include software development tools.
JDK = JRE + Development tools	JRE = JVM + Class libraries	JVM = provides a runtime environment.

8. diffrence between method overloading and overriding.

No.	Method Overloading	Method Overriding

1)	Method overloading is used *to increase the readability* of the program.	Method overriding is used *to provide the specific implementation* of the method that is already provided by its super class.
2)	Method overloading is performed *within class*.	Method overriding occurs *in two classes* that have IS-A (inheritance) relationship.
3)	In case of method overloading, *parameter must be different*.	In case of method overriding, *parameter must be same*.
4)	Method overloading is the example of *compile time polymorphism*.	Method overriding is the example of *run time polymorphism*.
5)	In java, method overloading can't be performed by changing return type of the method only. *Return type can be same or different* in method overloading. But you must have to change the parameter.	*Return type must be same or covariant* in method overriding.

9.Diffrence between arraylist and vector

Difference between Arraylist and Vector in Java

S.No.	ArrayList	Vector
1.	ArrayList is a resizable or dynamic array.	Vectors are also a form of dynamic array.
2.	ArrayList is not synchronised.	Vector is synchronised.
3.	Syntax of ArrayList: ArrayList<T> al = new ArrayList<T>();	Syntax of Vector: Vector<T> v = new Vector<T>();

4.	If the number of elements overextends its capacity, ArrayList can increase the 50% of the present array size.	If the number of elements overextends its capacity, Vector can increase the 100% of the present array size, which means double the size of an array.
5.	It is not an inheritance class.	It is an inheritance class.
6.	It is faster than Vector.	It is slow as compared to the ArrayList.
7.	It is not a legacy class.	It is a legacy class.
8.	It prefers the Iterator interface to traverse the components.	It prefers an Enumeration or Iterator interface to traverse the elements.

9.difference between == operator and .equals() method

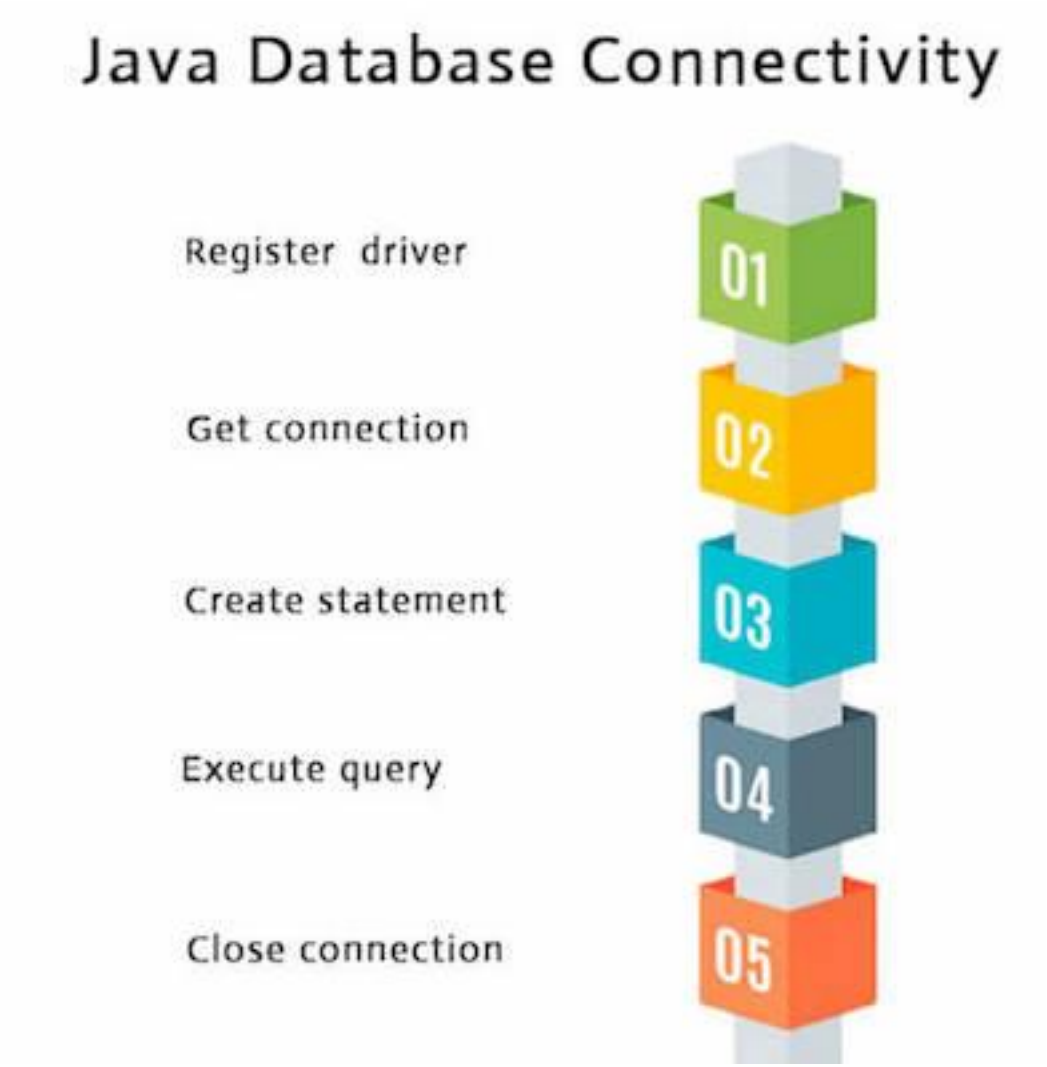
== operator	.equals() method
1.it is used for object comaparison or Reference comparasion(in object class)	1. .it is used for object comaparison or Reference comparasion(in object class)
2. .it is used for object comaparison or Reference comparasion	2.it is used to content comparison (in string)

10. difference between stack and heap

Stack	Heap
• It is a linear data structure. • Memory is allocated in a contiguous (continuous) block. • The memory for a stack is allocated and deallocated automatically	• It is a hierarchical data structure. • Memory is allocated in a random fashion.

using the instructions of the compiler. • It costs less to build and maintain a stack. • It is easy to implement. • It is fixed in size; hence it is not flexible. • Its only disadvantage is the shortage of memory, since it is fixed in size. • If all the blocks are not occupied, memory gets wasted too. • It takes less time to access the elements of a stack. • It has an excellent locality of reference.	• The memory is allocated and deallocated manually by the programmer. • It is costly to build and maintain a heap. • It is difficult to implement a heap structure. • It takes more time to access elements of a heap. • The disadvantage of heap is fragmentation of memory. • Resizing is possible in heap. • Hence, memory is not wasted. • It has adequate locality of reference.

Java Database Connectivity



Collection	Collections
Collection is an interface.	Collections is a utility class.
Belongs **java.util** package.	Belongs **java.util** package.
Collection can be used to represent a group of individual objects as a single entity.	Collections define several utility methods (like sorting , searching --) for collection objects.
Collection is the root interface from which almost all Data-structures are derived, commonly known as collection framework.	Collections class contains many static methods with which Data-structures manipulation becomes easier.

Differece between String Builder and String Buffer

StringBuffer	StringBuilder
StringBuffer operations are thread-safe and synchronized	StringBuilder operations are not thread-safe are not-synchronized.
StringBuffer is to used when multiple threads are working on the same String	StringBuilder is used in a single-threaded environment.
StringBuffer performance is slower when compared to StringBuilder	StringBuilder performance is faster when compared to StringBuffer
Syntax: `StringBuffer var = new StringBuffer(str);`	**Syntax:** `StringBuilder var = new StringBuilder(str);`

Difference Between Primary Key and Foreign Key

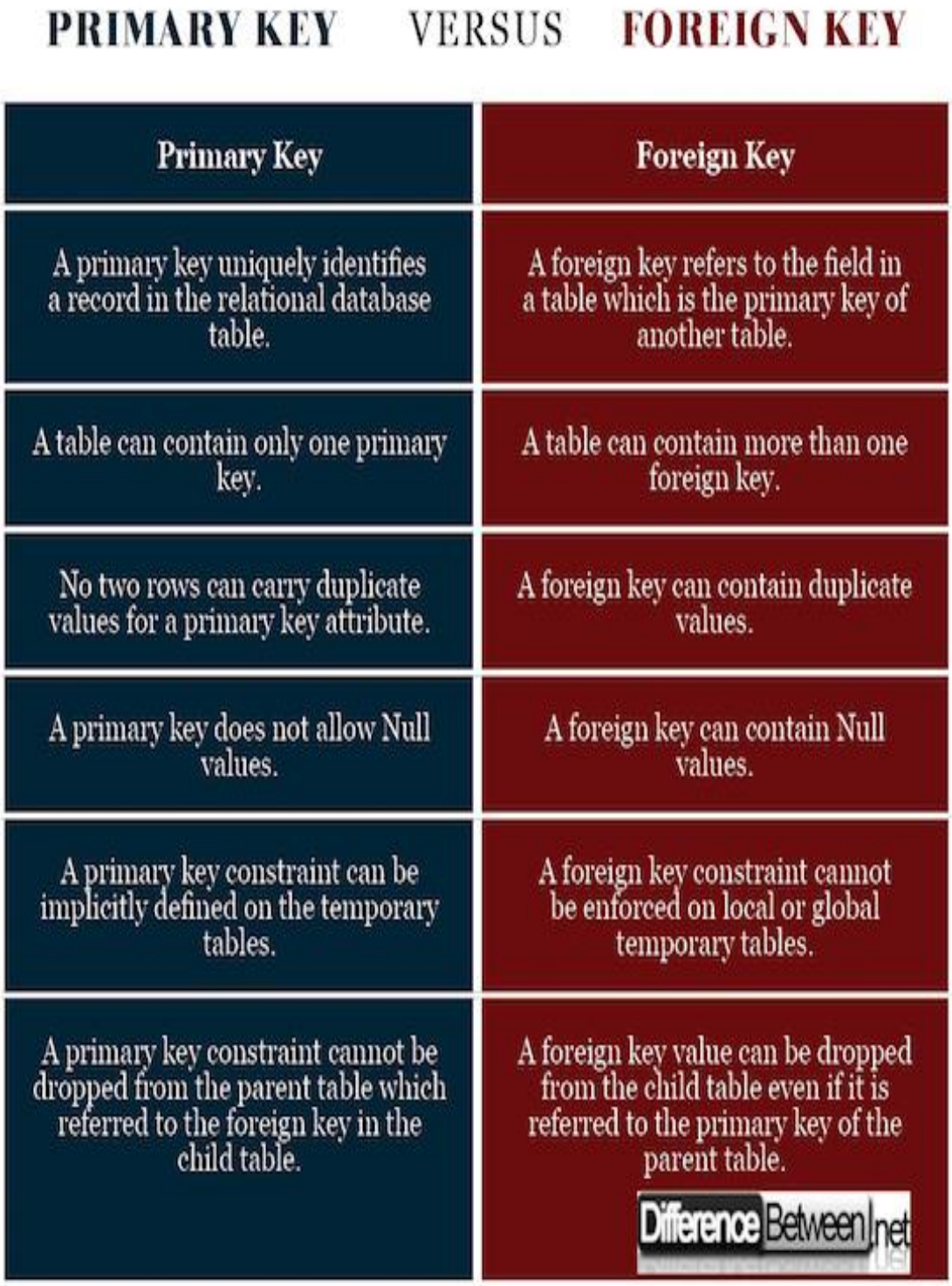
Primary Key	Foreign Key
Primary key uniquely identify a record in the table.	Foreign key is a field in the table that is primary key in another table.
Primary Key can't accept null values.	Foreign key can accept multiple null value.
By default, Primary key is clustered index and data in the database table is physically organized in the sequence of clustered index.	Foreign key do not automatically create an index, clustered or non-clustered. You can manually create an index on foreign key.
We can have only one Primary key in a table.	We can have more than one foreign key in a table.
The primary key of a particular table is the attribute which uniquely identifies every record and does not contain any null value.	The foreign key of a particular table is simply the primary key of some other table which is used as a reference key in the second table.
A primary key attribute in a table can never contain a null value.	A foreign key attribute may have null values as well.
Not more than one primary key is permitted in a table.	A table can have one or more than one foreign key for referential purposes.
Duplicity is strictly prohibited in the primary key; there cannot be any duplicate values.	Duplicity is permitted in the foreign key attribute, hence duplicate values are permitted.

Difference between hibernate and jpa

Difference between Stack and Heap Memory in Java

S.No.	Stack Memory in Java	Heap Memory in Java
1.	Stack follows the LIFO order.	Heap does not follow any order because here, the pattern of memory allocation is not fixed.
2.	It stores the entities that have a short life, like variables and methods.	Heap stores entities like objects.
3.	It is not flexible as we cannot make any changes once the memory allocation is done.	It is flexible as we can make changes here even after the allocation of memory.
4.	It is faster than heap in terms of allocation and deallocation.	It is slower than stack in terms of allocation and deallocation.
5.	The size of stack memory is small.	The size of heap memory is large.
6.	Through the JVM option -Xss, we can improve the size of a stack.	Through the JVM options -Xmx and -Xms, we can change the size of a stack.
7.	It is affordable.	It is affordable as compared to the stack.
8.	The implementation part is easy here.	The implementation part is tough here.
9.	In stack, the memory allotment is continuous.	In heap, the memory allotment is random.
10.	The allocation and deallocation are automatically performed by the compiler.	Here the allocation and deallocation are done manually.

Difference between Primary key and foreign key



Difference between Array and Collection

S.No.	Arrays	Collection Framework
1.	Arrays are fixed in size.	Collections are growable in nature.
2.	Array can contains both primitives and object type.	Collections can hold only objects but not primitive.
3.	Arrays are better than collections in performance.	Collections are not poor in performance than Arrays.
4.	Arrays can hold only homogeneous data types elements.	Collection can hold both homogeneous and heterogeneous elements.
5.	If we know the size of elements then it is good to go for arrays.	If we are not sure about size we will go for Collections.
6.	Underlying data structure is not available.	Collections are implements on a standard data structure.
7.	Example: int[] arr=new int[5];	Example: ArrayList<String> alist=new ArrayList<String>();

Difference between comparable and comparator

Comparable interface	Comparator interface
Comparable interface present in java.lang package.	Comparator interface present in java.util package.
Sort the elements according to natural sorting order.	Sort the elements according to customized sorting order.
It contains only one method i.e. compareTo()	It contains two methods compare() and equals()
CompareTo() method is responsible to sort the elements.	compare() method is responsible to sort the elements

Difference between spring and springboot

Spring Framework	Spring Boot Framework
The primary feature is Dependency Injection	Autoconfiguration is the primary feature of Spring Boot
we need to set up the server explicitly for the testing procedure.	offers embedded server such as Jetty and Tomcat, etc
For smaller tasks, developers need to write a boilerplate code	Reduction in boilerplate code
Does not provide an in-memory Database	Provide several plugins to work with embedded servers and some in-memory databases such as H2
Developers have to define dependencies manually in the pom.xml file	Starter concept in pom.xml file internally handles the required dependencies

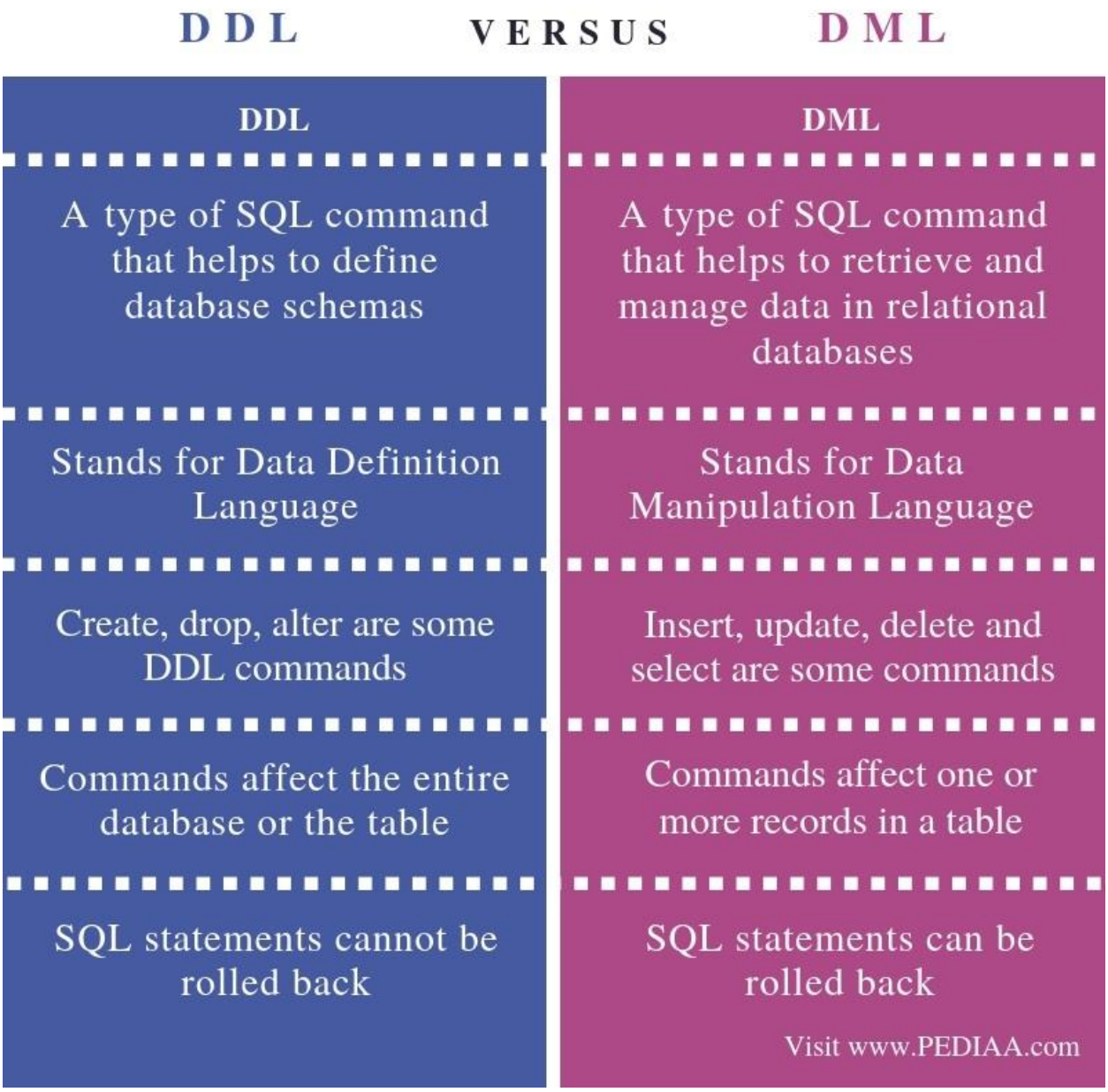
Difference between get() and load()

	get( )	load( )
1	Return value null is possible	Never returns null
2	Fast if record exists	Slow if record exists
3	Used to retrieve object (record)	Used for delete etc. operations
4	Eager fetching	Lazy fetching
5	Always hits the database	Not always hits
6	Does not return proxy object	Always returns a proxy object
7	Performance-wise is slow as it may have to make number of rounds to database to get data	Better performance. If already in the cache, the record is available to much difference
8	As it returns null if no record exist, the execution continues	It throws ObjectNotFoundException, if record not found. Execution terminates if not handled successfully

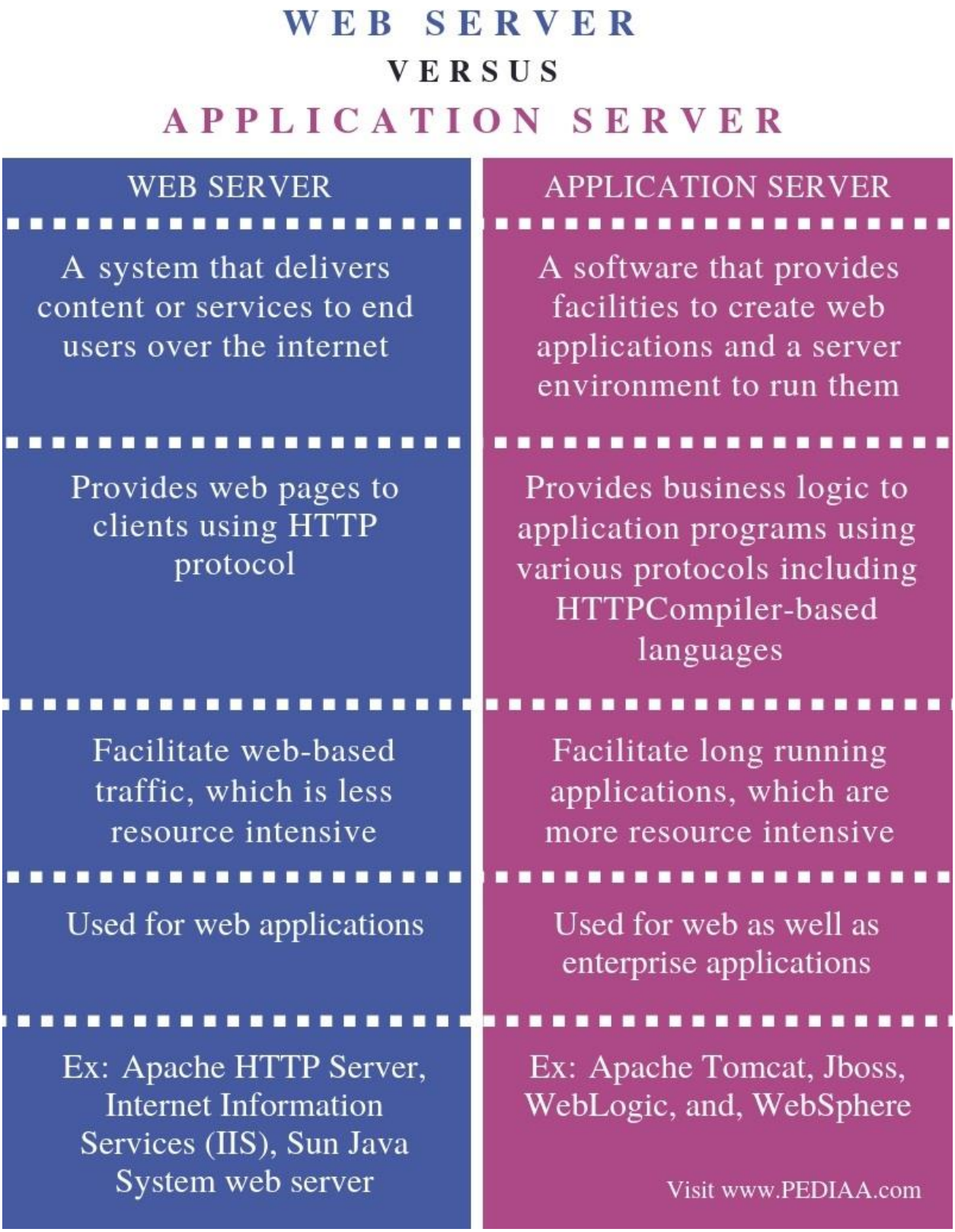
Difference between Doget and dopost

doGet()	doPost()
Parameters are appended to the URL, and sent along with the header information. So it brings up a security issues (eg. password)	Parameters are sent through Request body.
We can send maximum size of data 240 bytes.	We can sent large amount of data
Parameters are not encrypted	Parameters are encrypted
Processing is faster	Processing is slow when compared with Get()
Bookmark possible	The user can't bookmark a form submission if you use POST instead of GET.

Difference Between DDL and DML



Diffe rence between Application server and Web Server



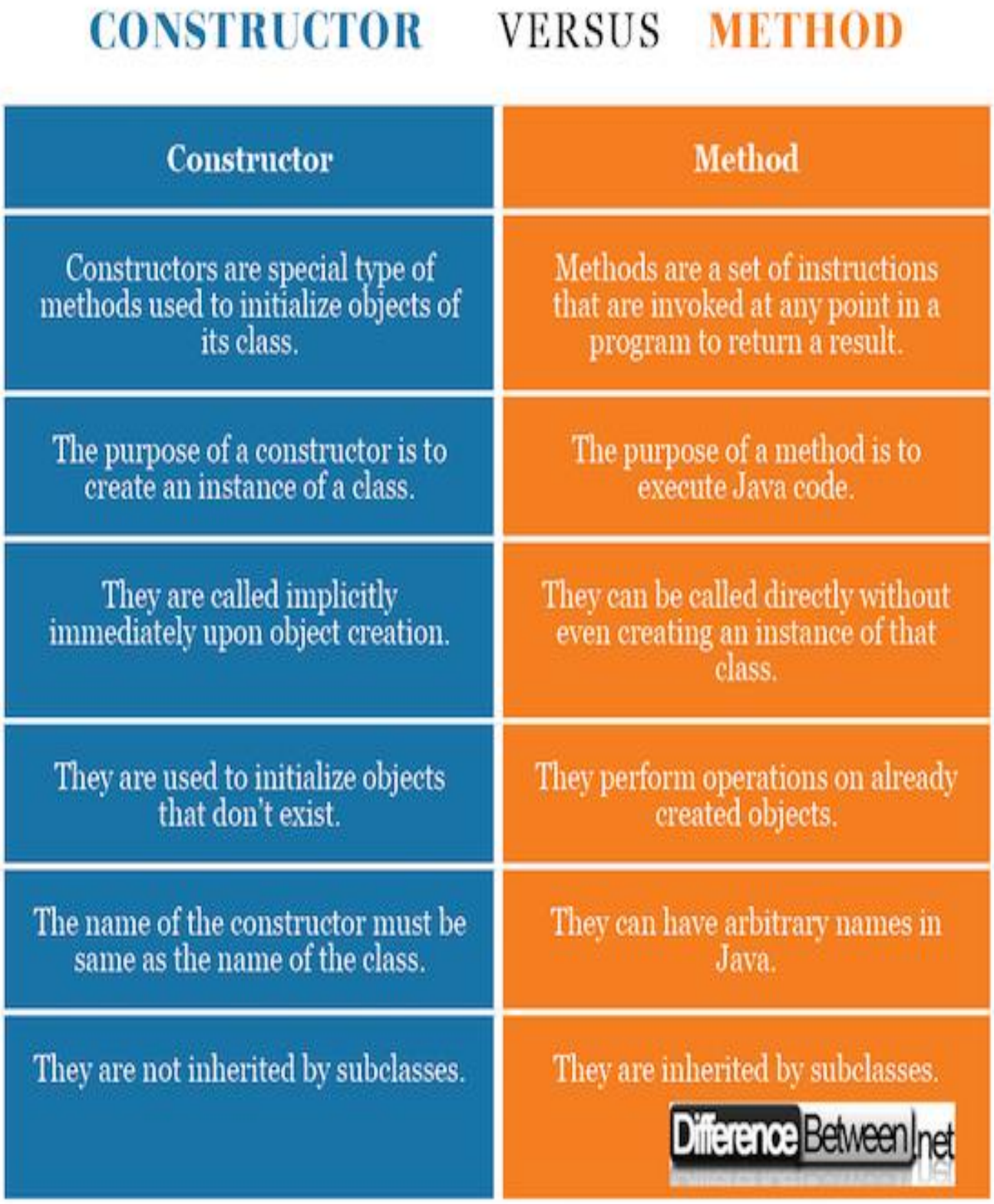
Difference between This keyword and super keyword

this()	super()
1. this() represents the current instance of a class	1. super() represents the current instance of a parent/base class
2. Used to call the default constructor of the same class	2. Used to call the default constructor of the parent/base class
3. Used to access methods of the current class	3. Used to access methods of the base class
4. Used for pointing the current class instance	4. Used for pointing the superclass instance
5. Must be the first line of a block	5. Must be the first line of a block

Difference between Throws an throw

No.	throw	throws
1)	Java throw keyword is used to explicitly throw an exception.	Java throws keyword is used to declare an exception.
2)	Checked exception cannot be propagated using throw only.	Checked exception can be propagated with throws.
3)	Throw is followed by an instance.	Throws is followed by class.
4)	Throw is used within the method.	Throws is used with the method signature.
5)	You cannot throw multiple exceptions.	You can declare multiple exceptions e.g. public void method()throws IOException,SQLException.

Difference between Constructors and methods



Difference between final,Finally,Finalize

BASIS FOR COMPARISON	FINAL	FINALLY	FINALIZE
Basic	Final is a "Keyword" and "access modifier" in Java.	Finally is a "block" in Java.	Finalize is a "method" in Java.
Applicable	Final is a keyword applicable to classes, variables and methods.	Finally is a block that is always associated with try and catch block.	finalize() is a method applicable to objects.

BASIS FOR COMPARISON	FINAL	FINALLY	FINALIZE
Working	(1) Final variable becomes constant, and it can't be reassigned. (2) A final method can't be overridden by the child class. (3) Final Class can not be extended.	A "finally" block, clean up the resources used in "try" block.	Finalize method performs cleans up activities related to the object before its destruction.

BASIS FOR COMPARISON	FINAL	FINALLY	FINALIZE
Execution	Final method is executed upon its call.	"Finally" block executes just after the execution of"try-catch" block.	finalize() method executes just before the destruction of the object.

Difference between Hashmap and hashtable

HashMap vs Hashtable

Comparison Chart

HashMap	Hashtable
HashMap is an unsynchronized Map.	Hashtable is a synchronized Map.
It is not thread-safe and cannot be shared between multiple threads without proper synchronization.	It is thread-safe and can be shared between multiple threads.
It supports all Map operations and allows multiple null values but only one null key in a collection so that it could maintain unique key properties.	It does not support null values and null keys because there is null check in the put method implementation of Hashtable.
It is much faster and better than a Hashtable in terms of performance.	It is a bit slower than a HashMap but faster than a synchronized HashMap.
HashMap performs better in a multi threaded environment.	Hashtable performs better in a single threaded environment.
It uses iterator to iterate the values of HashMap object.	It uses iterator and Enumerator to iterate the Hashtable object values.
The iterator is fail-fast meaning it throws an exception if the system fails.	The enumerator in Hashtable does not have this behavior.