Java Bean is a java class that is developed with the following standarads
      a. class must be public
      b. Recomended to implement Serializable interface
      c. Bean property variable should be private and non-static
      d. Every bean property should have one pair of Setter and getter method
      e. It should have zero parameter constructor directly or indirectly

There are 3 types of java beans
      a. VO class(Value Object) => To hold user inputs
      b. DTO(Data Transfer Object) => To carry the data from one layer to another layer in a project
      c. BO(Buisness Object) => To hold persistable object/persistable data.

A well designed java class should contain
      a. Overloaded constructor
      b. toString()
      c. equals() method
      d. hashcode() method
      e. setters and getters(optional)

Before Lombok API
==========
=> We need to make our java class well designed java class by the adding the above said methods manually and
   we should manually increase or decrease the setter methods and getter methods based on no of properties we
   are adding/removing.

With Lombok API
==========
=> All the above things will be taken care and generated automatically.
=> Lombok API is also called as "Project Lombok" which generates the following boiler plate of java code
      a. constructors
      b. setters and getters
      c. toString()
      d. equals()
      e. hashcode() ....
=> It is an open source api.
=> It must be configured with IDE's to make IDE's using lombok api to generate the common boiler plate code.
=> It supplies bunch of annotations for generating the common code
      Annotations::

@Setter,@Getter,@AllArgsConstructor,@NoArgsConstructor,@RequiredArgsConstructor,@ToString,@EqualsAndHashCode
            @Data(It is mix of multiple annotations) etc,....

Steps to Configure Lombok API with eclipse IDE
==========
step1: Download lombok-<ver>.jar from mvnrepository.com
step2: make sure eclipse/sts ide is downloaded
step3: create proejct in eclipse ide or sts ide by adding lombok-ver.jar file to the build path
step4: launch lombok api by clicking on lombok-ver.jar and specify the eclipseide/sts ide installation folder
         click on install/update button, click on quit installer

step5: restart eclipse ide
step6: Add one java bean class to project of eclipse ide/sts ide by lombok api annotation and observe whether code is generated or not.

Note: lombok api annotations make java compiler to generate certain code dynamicaly in the .class file
java compiler is having the ability to add code dynamically to the .class file though the instructions are not there in .java like
default consturctor generation makes java.lang.Object as default super class and etc.

Note: Lombok Retention level is source,it means these annotations will not be recorded in the .class file, but becoz of lombok
annotations instructions the code gets generated by javac compiler like setters,getters,toString() etc will be
recorded to .class file.

@Getter,@Setter
=> These annotations are applied at the class level, generates getter and setter for all field/properties.
=> If these are applied at the field level, generates getters and setters only for specific field/properties.

Eg#1.

```
@Getter
@Setter
class Student{
      private Integer sid;
      private String sname;
      private Integer sage;
}
```

Student.class
==========

```
class Student{
      private Integer sid;
      private String sname;
      private Integer sage;

      setXXX(),getXXXX(),Student(){}
}
```

Eg#2.

```
class Student{
      private Integer sid;
      private String sname;

      @Getter
      @Setter
      private Integer sage;
}
```

Student.class
==========

```
class Student{
      private Integer sid;
      private String sname;
      private Integer sage;

      setSage(),getSage(),Student(){}
}
```

```
@ToString()
==========
=>It makes the javac to generate /override toString() having the logic to display
the object data.
=>It is applicable only on the top of the class(@Target(ElementType.TYPE))

eg#1.
@ToString
public class Employee {
      private Integer sid;
      private String sname;
      private Integer sage;
      private String saddress;
}

Employee.class
==============
public class Employee {
      private Integer sid;
      private String sname;
      private Integer sage;
      private String saddress;

      Employee(){}
      public String toString(){
            prints the object data.
      }
}

What is the use of toString()?
      => It is useful to display the object data in string format
      => If we don't override this method in our class ,then java class toString()
executes and this method gives
            <fullyqualifiedclassname>@hexadecimal notation of hashCode>.
             public String toString() {
                        return getClass().getName() + "@" +
Integer.toHexString(hashCode());
             }

can we customize the logics generated by lombok api?
      Ans. not possible.

What will happen if we try to override toString() explicitly along with lombok api
code?
      => @ToString of lombok api will not give instruction to generate toString()
becoz same method is already available in .java file.
            (warning will come on top of @ToString)
                  same is applicable for @Getter,@Setter.

EqualsAndHashCode
=================
```

=> Generates equals() and hashcode() by giving instruction to java compiler(javac)
=> It is applicable at class level(@Target(ElementType.TYPE))

What is HashCode?
 => It is the unique identity number given by jvm for every object.. and we can get by calling hashcode() method.
```
            System.out.println(c1.hashCode() + " " + c2.hashCode());
```

Can 2 objects have same hashcode?
 => If hashcode is given by jvm then it is not possible,because hashcode is generated based on hashing algorithm.
 => If we overide hashcode() method our code, generally it generates the hashcode based on the state of the object
    therefore if 2objects are having same state, then we get same hashcode for both objects.

eg::
```
@EqualsAndHashCode
@AllArgsConstructor
@ToString
public class Employee {
      private Integer sid;
      private String sname;
      private Integer sage;
      private String saddress;
}
public class TestApp {
      public static void main(String[] args) {
            Employee e1 = new Employee(10, "sachin", 49, "MI");
            Employee e2 = new Employee(10, "sachin", 49, "MI");
            System.out.println(e1.hashCode()+"==="+e2.hashCode());// 123456 ==== 123456 (because we use @EqualsAndHashCode,state of object is same)
            System.out.println(System.identityHashCode(e1));
            System.out.println(System.identityHashCode(e2));
      }
}
```

Can object have 2 hashCode?
 Ans. yes possible, one hashcode is based on the state of the object and the other one is internally generated by jvm.
        To get the hashcode generated by jvm we use a method called "System.identityHashCode(obj)".

What is the use of equals() method?
 => It is given to compare the state of 2 objects,it internally use hashcode supports also.
 => If we override equals() in our class, then it will compare the state of the object, otherwise it will execute the object class equals()
    which will compare the reference of the object.
```
            public boolean equals(Object obj) {
                        return (this == obj);
            }
```

what is the difference b/w == and equals()?
 => Both will compare the reference if both are not overriden in our class.
 => If equals() overriden,then it compares the state of the object, then == operator will compare the reference.

```
eg#1.
@EqualsAndHashCode
@AllArgsConstructor
@ToString
public class Employee {
	private Integer sid;
	private String sname;
	private Integer sage;
	private String saddress;
}

public class TestApp {
	public static void main(String[] args) {
		Employee e1 = new Employee(10, "sachin", 49, "MI");
		Employee e2 = new Employee(10, "sachin", 49, "MI");
		System.out.println(e1==e2);//false[compares the reference maintained by
jvm internally for objects]
		System.out.println(e1.equals(e2));//true [compares the state of the
object becoz overriding equals() in our class]
	}
}

Constructor Creation
====================
@NoArgsConstructor  => Generates Zero param constructor
@AllArgsConstructor => Generates parameterized constructor having all
properties/fields as fields.
				If no properties in our class then Zero param
constructor.


eg#1.
@AllArgsConstructor
@NoArgsConstructor
public class Employee {
	private Integer sid;
	private String sname;
	private Integer sage;
	private String saddress;
}
public class Employee {
	private Integer sid;
	private String sname;
	private Integer sage;
	private String saddress;

	public Employee(Integer sid,String sname,Integer sage, String saddress){
			;;;;
			;;;;
			;;;;
	}

	public Employee(){}
}

eg#2.
@AllArgsConstructor
@NoArgsConstructor
public class Employee {
```

```
}

@AllArgsConstructor
@NoArgsConstructor
public class Employee {
      //Error(becoz constructor can't be duplicated)
      public Employee(){}
      public Employee(){}
}
```

Note:Only 0-param constructor generated by compiler is called "Default constructor".
      If we write 0-param constructor and if it is generated by lombok then it is should not be called as "Default constructor".
                  eg: int a ; //instance level => default value is "0"
                      int a =0;//instance level => This is not a default value it is initialized to zero.

Can we overload/override Constructor?
Ans. We can't override constructor, we can just overload constructor.

```
eg::
@NoArgsConstructor
public class Customer{
      public Customer(){

      }
}

public class Customer{
      //Error(becoz constructor can't be duplicated)
      public Customer(){}
      public Customer(){}
}
```

@RequiredArgsConstructor
==========
 => Allows to generate parameterized constructor involving our choice properties/fields.
 => The properties that we need to involve should be annotated with @NonNull annoatation.
 => If no properties are annotated with @NonNull then it will give Zero param constructor.

```
eg#1.
@RequiredArgsConstructor
public class Employee {
      private Integer sid;
      private String sname;
      private Integer sage;
      private String saddress;
}

public class Employee {
      private Integer sid;
      private String sname;
```

```
	private Integer sage;
	private String saddress;

	public Employee(){}
}

eg#2.
@NoArgsConstructor
@RequiredArgsConstructor
public class Employee {
	private Integer sid;
	private String sname;
	private Integer sage;
	private String saddress;
}

public class Employee {
	private Integer sid;
	private String sname;
	private Integer sage;
	private String saddress;

	public Employee(){}
	public Employee(){}
}

eg#3.
@RequiredArgsConstructor
public class Employee {

	@NonNull
	private Integer sid;

	@NonNull
	private String sname;
	private Integer sage;
	private String saddress;
}

public class Employee{
	@NonNull
	private Integer sid;

	@NonNull
	private String sname;
	private Integer sage;
	private String saddress;

	public Employee(Integer sid,String sname){.......}
}


eg#4.
@RequiredArgsConstructor
@AllArgsConstructor
@NoArgsConstructor
public class Employee {

	@NonNull
```

```
      private Integer sid;

      @NonNull
      private String sname;
      private Integer sage;
      private String saddress;
}

public class Employee {

      @NonNull
      private Integer sid;
      @NonNull
      private String sname;
      private Integer sage;
      private String saddress;

      public Employee(){}
      public Employee(Integer sid,String sname,Integer sage,String saddress){}
      public Employee(Integer sid,String sname){...}

}
```

Note: We can take constructor as private,protected,public. To get them through lombok api we can use "access" attribute of
@XxxxArgsConstructor as shown below.

```
@AllArgsConstructor(access = AccessLevel.PRIVATE)
@RequiredArgsConstructor(access = AccessLevel.PROTECTED)
@NoArgsConstructor(access = AccessLevel.PUBLIC)
public class Employee {
      ;;;;;;
}
```

Note::
=> we cannot generate constructor with var args
=> we cannot generate multiple our choice parameterized constructor at a time like 1param,2param with 3param at a time.

@Data
=====
=> It is a Combination of
@Getter,@Setter,@EqualsAndHashCode,@ToString,@RequiredArgsConstructor.

```
@Data
public class Employee {

      @NonNull
      private Integer sid;
      @NonNull
      private String sname;
      @NonNull
      private Integer sage;
      private String saddress;
}

public class Employee{
      Employee(Integer, String, Integer)
```

```
      canEqual(Object)
      equals(Object)

      getSaddress()
      getSage()
      getSid()
      getSname()

      hashCode()

      setSaddress(String)
      setSage(Integer)
      setSid(Integer)
      setSname(String)

      toString()
}


@Data
@AllArgsConstructor
public class Employee {

      @NonNull
      private Integer sid;
      @NonNull
      private String sname;
      @NonNull
      private Integer sage;
      private String saddress;
}
Note: @RequiredArgsConstructor of @Data works only when
@AllArgsConstructor,@NoArgsConstructor is not placed on top of the class.
      If u still need the effect of @RequiredArgsConstructor then place it
explicitly.

eg#2.
@Data
@AllArgsConstructor
public class Employee {

      @NonNull
      private Integer sid;
      @NonNull
      private String sname;
      @NonNull
      private Integer sage;
      private String saddress;
}

public class Employee{
            Employee(Integer, String, Integer, String)

            canEqual(Object)
            equals(Object)

            getSaddress()
            getSage()
```

```
            getSid()
            getSname()

            hashCode()

            setSaddress(String)
            setSage(Integer)
            setSid(Integer)
            setSname(String)

            toString()
}


eg#3.
@Data
@NoArgsConstructor
@AllArgsConstructor
@RequiredArgsConstructor
public class Employee {

      @NonNull
      private Integer sid;
      @NonNull
      private String sname;
      @NonNull
      private Integer sage;
      private String saddress;
}

public class Employee{
     Employee()
     Employee(Integer, String, Integer, String)
      Employee(Integer, String, Integer)


      canEqual(Object)
      equals(Object)

      getSaddress()
      getSage()
      getSid()
      getSname()

      hashCode()

      setSaddress(String)
      setSage(Integer)
      setSid(Integer)
      setSname(String)

      toString()

}
```