Object-Oriented-Programming #LE3-4

Public, Private and Protected:

- **Private:** So, if we put Private important any data type variable, we can not really access it, outside the file itself.

```
private int num;
```

// we can not really access it outside the class file itself

Note:

The only way to access the private stuff using getter and setter. getter gets the value, setter sets the value.

getter

```
public int getNum(){
    return num;
}
```

setter

```
public void setNum(int Num){
    this.num = num;
}
```

Now, you can use `obj.getNum();` method outside to access the Private Num variable. in outside file now we can use the private stuff

and we can also use the private int num method outside packages like. ⬇

~~package~~ com.Bigoy.

```
import com.Bigoy.access.A          // importing the package and file itself

A a = new A(10,"Bigoy");           // creating ob out of the class
a.getNum();
                                   // then using getNum() method to actually use the data
```

- **public** : public means we can access the data, anywhere, outside the class file itself.

```
public int num;                    // Now we can access the integer type number variable anywhere outside the class file.
```

like this ⬇

```
A a = new A(10,"Bigoy");
```

Note:
when we doesn't mention anything before the int num type. we can only access it inside the package, not outside it.

```
int num;                           // we can only used, access it inside the package inwhich it's class file present.
```

when the access modifier is	class	Package	Sub class (Same pkg)	Sub class (diff pkg)	World (diff pkg but not sub class)
public	+	+	+	+	+
protected	+	+	+	+	
no modifier	+	+	+		
private	+				

■ When to use which modifier:

- private: private is for sensitive data that you don't access to
  * we can access this via getter and setter method, which can be public
- no modifier: use no modifier for those variable you don't want be used outside that particular package.
- Protected: I don't want my variable to be use outside package, but I only wanted to be access outside the package if it's in the subclass
- public: we will gonna use public when we want the variable to be used from anywhere

that's why main function we use public as we want it to access from anywhere

Note: only the Subclass can access the protected member in the base class, when it's in different package.

Example:

Suppose you have a className A, where inside of it you got variable

```
int num;
```

Now, only the subclass of different package can access the members in the protected class, when it's outside. like this ⬇

Subclass of same & different Package

```
import com.Bijoy.access.A
public class Subclass extends A{
public Subclass(int num, String name){
    super(num, name); }
public static void main(String[] args){
Subclass obj = new Subclass(45, "Bijoy");
int n = obj.num; }
}
```

```
A obj = new A(45, "Bijoy");
int n = obj.num;   // This will not work
```

// If this works there is no difference b/w public & protected

As, A don't have idea about the child class es the you can put it anywhere.

the dependency weathen 'A' class is going to be access on the different package on not is dependent on the subclass, because 'A' doesn't know anything about, if there is any class which is extending it or not

- Also, if a subclass extends another sub-class(which contain protected stuff) then that subsubclass can also access the protected data.

example:

```
class SubSubclass extends Subclass{
public SubSubclass(int num, String name){
    super(num, name);
                                        }
public static void main(String[] args){
SubSubclass obj = new SubSubclass(45,"Bijay")
int n = obj.num;
                                        }
                                        }
```

Note:

→ we can use protected in the subclass of the same package and subclass of the different package

→ Also different package but not subclass where protected can not be used

example:

```
public class NotSubclass{ //doesn't extending class A
}
```

```
public static class NotSubclass {
public static void main(String[] args){
    NotSubclass obj = new NotSubclass();
    int n = obj.num;  // will not work
```

→ you will only able to use protected member in a different package only when you are accessing it with a subclass, not even the class itself ('A') but subclass.

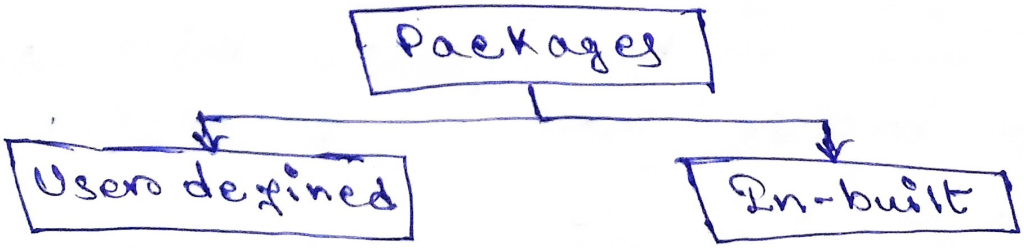
That's why we need to extending it

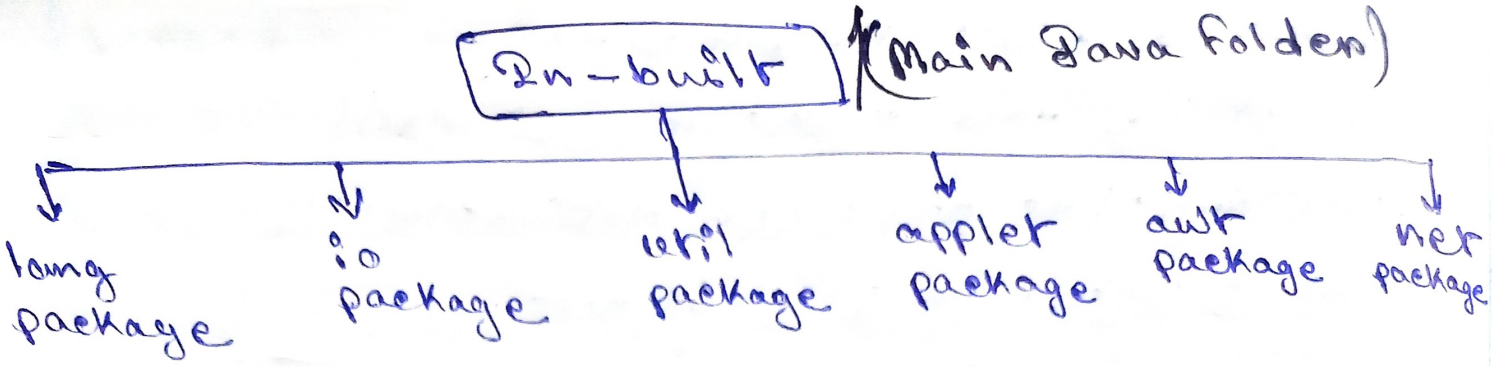
```
public class Subclass extends A {
}
// A contain protected members
```

★ In different Packages you can use protected stuff inside subclasses.

■ More about Packages :

- packages are folders that we used to categorize stuff.
- Packages are two type





- **"lang" Package:** lang package contains essential stuff of Java, Java language specific stuff.
  - Something like operations that we do, like: plus, minus or something like that, internally somewhere that might be contain in classes
  - So all the Basic language stuff go inside language package
  - we never really have to import it inside our class file, as it is automatically being imported.

- **"io" Package:** Input output Package, when we will doing file reading, those sorts of classes that used to read a file, optimizing input out for competitive programming in Java, Buffer readers and all these other things, these go inside the io package,
  - Java.io package

- "util" Package: Util contains utility classes
  - These contains data structure and stuff and Collection framework aswell, collection framework have lot of things inside of it.
- "applet" Package: used to create applets in Java.
- "awt" Package: This package is used for graphical user interface. if you wanna create a button on any Gui application.
- "net" Package: The use of these package is when we performing some networking operations.

- Object class: class object is the root of the class hierarchy. So, it's the top most class in inheritance.
  - every class that you will create that will be inheriting object class indirectly. (in java multiple inheritance is not allowed, so it will be being done internally, indirectly)
  - Object Methods: object class has few important things or methods that we use and these are . . .

and these method are:

→ "Construction" of the Object class

~~→ "toString()" method and~~

→ the "Override" Methods, and in these Override method, we have method like

- hashCode()
- equals(obj: Object)
- clone()
- toString()
- finalize()

- About toString() method: toString() gives the String representation.

```
public String toString(){
    return super.toString();
}
```

- About finalize() method: finalize() basically called when garbage Collection hits.

```
protected void finalize() throws Throwable{
    super.finalize();
}
```

- About hashCode() method: It basically gives number representation of an object.

It's actualy gives gives unique representation of number for each and individual unique object

example:

```
ObjectDemo obj = new ObjectDemo(34);
ObjectDemo obj2 = new ObjectDemo(34);

System.out.println(obj.hashCode());
System.out.println(obj2.hashCode());
```

Output: The output on number representation will be differ for each object, obj & obj2 will different for.

Although the value we have pass is the same which is integer number 34.

Representation of hashCode

```
public int hashCode(){
  return super.hashCode();
                        }
```

But if we change the return value of hashCode to. this ↓

```
public int hashCode(){
  return num;          }
```

// Now if we print it

Now, the output value will be same num itself.

the output will be ....

34
34

- hashcode() is not a number, it's a random integer value, formed via using some algorithm

- About equals() method: The equals() method compares two strings and returns true if the strings are equal, and false if not.

```
public boolean equals(Object obj){
    return super.equals(obj);
}
```

- About clone() method: The clone() method as the name implies, create an identical copy of an object.
  - Depending on the type of implementation this copy can either be a shallow or a deep copy.

- instance of Operator: The instanceof operator in Java is used to check whether an object is an instance of a particular class or not.

- ~~Eg~~. It's syntax is :

```
objectName instanceOf className;
```

// if objectName is an instance of className, the operation returns true. Otherwise, it returns false.

example: Suppose we have parent class 'A' and child class named as 'SubClass', now if we create a object out of SubClass using variable obj, like this ⬇

```
public class SubClass extends A{
public static void main(String[]args){
    SubClass obj = new SubClass(10,"Bijoy");
    int n = obj.num;
System.out.println(obj instanceof Object);
System.out.println(obj instanceof A);
System.out.println(obj instanceof SubClass);
```

// All of it will return true, as variable obj is instance of all of the three classes (object, A, SubClass).

- **getClass() Method**: getclass in Java is a method of the object class present in java.lang package
- getClass() returns the runtime class of

the object "this". This returned class object is locked by static synchronized method of the represented class.

* getClass() is a final type of method, so we can not override it.

Example:

```
public class ClassXYZ {
  public static void main (String [] args) {
    Class One object1 = new ClassOne();
    System.out.println(object1.getClass());

    String str = "Hello World";
    System.out.println (str.getclass()); }

    class ClassOne{
    }
```

Output: it will return

class com.Bijoy.access.ClassOne
class java.lang.String