**OBJECT ORIENTED PROGRAMMING**

***Department of Software Engineering***

#### Public Relations, MUET, SZAB Campus Khairpur Mir's - College & University | Facebook - 9,847 PhotosMentor:

##### Engr. Abdul Hafeez Babar [hafeez.babar@muetkhp.edu.pk](mailto:hafeez.babar@muetkhp.edu.pk)

***Mehran University of Engineering and Technology, SZAB Campus Khairpur Mir’s***

# Encapsulation in Java



## Java Package

* A java package is a group of similar types of classes, interfaces and sub-packages.

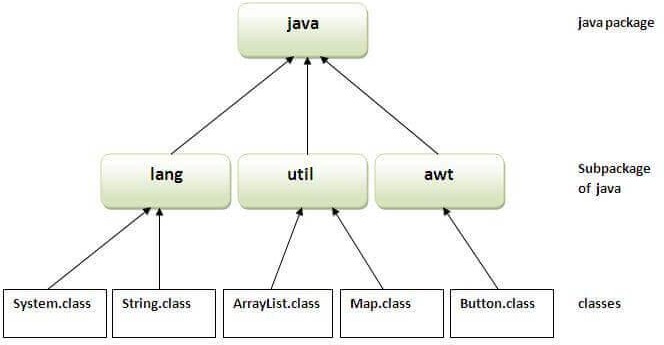


* Package in java can be categorized in two form, built-in package and user-defined package.
* There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

##### Advantage of Java Package



* 1. Java package is used to categorize the classes and interfaces so that they can be easily maintained.
  2. Java package provides access protection.
  3. Java package removes naming collision.



**Example:**



The package keyword is used to create a package in java.

//save as Simple.java package mypack; public class Simple{

public static void main(String args[]){

System.out.println("Welcome to package");

}

}

###### *How to compile java package*

* If you are not using any IDE, you need to follow the syntax given below:

javac -d directory javafilename

**For example**

javac -d . Simple.java

#### How to run java package program?



* You need to use fully qualified name e.g. mypack.Simple etc to run the class.

**To Compile:** javac -d . Simple.java

**To Run:** java mypack.Simple Output:Welcome to package

The -d is a switch that tells the compiler where to put the class file i.e. it represents destination. The . represents the current folder.

**How to access package from another package?**



###### *There are three ways to access the package from outside the package:*

1. import package.\*;
2. import package.classname;
3. fully qualified name.

### Using packagename.\*



* + If you use package.\* then all the classes and interfaces of this package will be accessible but not subpackages.
  + The import keyword is used to make the classes and interface of another package accessible to the current

package.

##### Example:

//save by A.java package pack; public class A{

public void msg(){System.out.println("Hello");}

}

//save by B.java package mypack; import pack.\*;

class B{

public static void main(String args[]){

A obj = new A();

obj.msg();

}

}

**Output:** Hello

### Using packagename.classname



If you import package.classname then only declared class of this package will be accessible.

##### Example:

//save by A.java package pack;

public class A{

public void msg(){System.out.println("Hello");}

}

//save by B.java package mypack; import pack.A;

class B{

public static void main(String args[]){

A obj = new A();

obj.msg();

}

}

**Output:** Hello

### Using fully qualified name



* + If you use fully qualified name then only declared class of this package will be accessible. Now there is no need to import. But you need to use fully qualified name every time when you are accessing the class or interface.

##### Example:

//save by A.java package pack; public class A{

public void msg(){System.out.println("Hello");}

}

//save by B.java package mypack; class B{

public static void main(String args[]){

pack.A obj = new pack.A();//using fully qualified name obj.msg();

}

}

**Output:** Hello

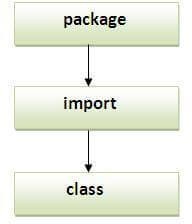
##### *Note:* If you import a package, subpackages will not be imported.

* If you import a package, all the classes and interface of that package will be imported excluding the classes and



interfaces of the subpackages. Hence, you need to import the subpackage as well.

***Note:*** Sequence of the program must be package then import then class



### Subpackage in java



* + Package inside the package is called the subpackage. It should be created to categorize the package further.

##### Example:

package pack.task; class Simple{

public static void main(String args[]){ System.out.println("Hello subpackage");

}

}

**To Compile:** javac -d . Simple.java

**To Run:** java pack.task.Simple

**Output:**Hello subpackage



## Access Modifiers in Java

###### *There are two types of modifiers in Java:*



1. access modifiers
2. non-access modifiers.

* The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

**There are four types of Java access modifiers:**



1. **Private:** The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. **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.

1. **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.

1. **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.

### Java Modifiers



* There are many non-access modifiers, such as static, abstract, synchronized, native, volatile, transient, etc.

#### Understanding Java Access Modifiers

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Access Modifier** | **within class** | **within package** | **outside package by subclass only** | **outside package** |
| **Private** | Y | N | N | N |
| **Default** | Y | Y | N | N |
| **Protected** | Y | Y | Y | N |
| **Public** | Y | Y | Y | Y |

1. **Private**



* + The private access modifier is accessible only within the class.

##### Example:

* In this example, we have created two classes A and Simple. A class contains private data member and private method. We are accessing these private members from outside the class, so there is a compile-time error.

**class** A{

**private int** data=40;

**private void** msg(){System.out.println("Hello java");}

}

**public class** Simple{

**public static void** main(String args[]){ A obj=**new** A();

System.out.println(obj.data);//Compile Time Error

obj.msg();//Compile Time Error

}

}

### Role of Private Constructor



* + If you make any class constructor private, you cannot create the instance of that class from outside the class.

##### Example:

**class** A{

**private** A(){}//private constructor

**void** msg(){System.out.println("Hello java");}

}

**public class** Simple{

**public static void** main(String args[]){ A obj=**new** A();//Compile Time Error

}

}

***Note:*** A class cannot be private or protected except nested class.

### Default



* + If you don't use any modifier, it is treated as **default** by default.
  + The default modifier is accessible only within package.
  + It cannot be accessed from outside the package. It provides more accessibility than private. But, it is more

restrictive than protected, and public.

#### Example:

//save by A.java **package** pack; **class** A{

**void** msg(){System.out.println("Hello");}

}

//save by B.java **package** mypack; **import** pack.\*; **class** B{

**public static void** main(String args[]){ A obj = **new** A();//Compile Time Error obj.msg();//Compile Time Error

}

}

* In the above example, the scope of class A and its method msg() is default so it cannot be accessed from outside the

package.

### Protected

Hello

* + The **protected access modifier** is accessible within package and outside the package but through inheritance only.
  + The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.
  + It provides more accessibility than the default modifier.

#### Example

* + - In this example, we have created the two packages pack and mypack. The A class of pack package is public, so can be accessed from outside the package. But msg method of this package is declared as protected, so it can be accessed

from outside the class only through inheritance.



//save by A.java **package** pack; **public class** A{

**protected void** msg(){System.out.println("Hello");}

}

##### OUTPUT: HELLO

//save by B.java **package** mypack; **import** pack.\*;

**class** B **extends** A{

**public static void** main(String args[]){ B obj = **new** B();

obj.msg();

}

}

### Public



* + The **public access modifier** is accessible everywhere. It has the widest scope among all other modifiers.

#### Example:

//save by A.java

**package** pack;

##### public class A{

**public void** msg(){System.out.println("Hello");}

}

//save by B.java

**package** mypack;

**import** pack.\*;

**class** B{

**public static void** main(String args[]){

A obj = **new** A(); obj.msg();

}

}

**OUTPUT: Hello**

**Java Access Modifiers with Method Overriding**



* If you are overriding any method, overridden method (i.e. declared in subclass) must not be more restrictive.

##### Example

**class** A{

**protected void** msg(){System.out.println("Hello java");}

}

**public class** Simple **extends** A{

**void** msg(){System.out.println("Hello java");}//C.T.Error

**public static void** main(String args[]){ Simple obj=**new** Simple(); obj.msg();

}

}

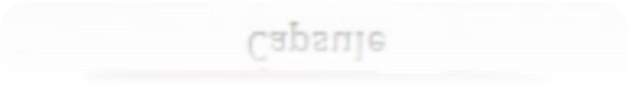
# Encapsulation in Java



* + Encapsulation in Java is a process of wrapping code and data together into a single unit, for example, a capsule which is mixed of several medicines.



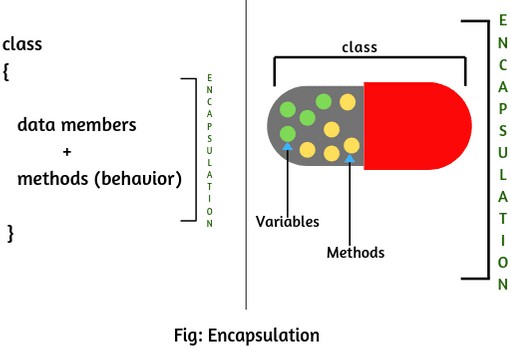
* + - We can create a fully encapsulated class in Java by making all the data members of the class private. Now we can use setter and getter methods to set and get the data in it.



###### *The Java Bean class is the example of a fully encapsulated class.*

* Encapsulation is defined as the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates. Another way to think about encapsulation is, it is a protective shield that prevents the data from being accessed by the code outside this shield.



* Encapsulation is a programming technique that binds the class members (variables and methods) together and prevents them from being accessed by other classes. Thereby, we can keep variables and methods safes from outside interference and misuse.
* Every Java class is an example of encapsulation because we write everything within the class only that binds variables and methods together and hides their complexity from other classes.
  + In the encapsulation technique, we declare fields as private in the class to prevent other classes from accessing them directly. The required encapsulated data can be accessed by using public Java getter and setter method.



* + If the field is declared private in the class then it cannot be accessed by anyone from outside the class and hides field within the class. Therefore, it is also called data hiding.

**Realtime Example of Encapsulation in Java**



#### Realtime Example 1:

* + - School bag is one of the most real examples of Encapsulation. School bag can keep our books, pens, etc.

#### Realtime Example 2:

* When you log into your email accounts such as Gmail, Yahoo Mail, or Rediff mail, there is a lot of internal processes taking place in the backend and you have no control over it. When you enter the password for logging, they are retrieved in an encrypted form and verified, and then you are given access to your account. You do not have control over it that how the password has been verified. Thus, it keeps our account safe from being misused.

#### Realtime Example 3:



* Suppose you have an account in the bank. If your balance variable is declared as a public variable in the bank software, your account balance will be known as public, In this case, anyone can know your account balance. So, would you like it? Obviously No.
* So, they declare balance variable as private for making your account safe, so that anyone cannot see your account balance.
* The person who has to see his account balance, will have to access only private members through methods defined

inside that class and this method will ask your account holder name or user Id, and password for authentication.

* Thus, we can achieve security by utilizing the concept of data hiding. This is called Encapsulation in Java.

### How to achieve or implement Encapsulation in Java



###### *There are two important points whereby we can achieve or implement encapsulation in Java program.*

1. Declaring the instance variable of the class as private. so that it cannot be accessed directly by anyone from outside the class.
2. Provide the public setter and getter methods in the class to set/modify the values of the variable/fields.

### Advantage of Encapsulation in Java



1. By providing only a setter or getter method, you can make the class read-only or write-only. In other words, you can skip the getter or setter methods.
2. It provides you the control over the data. Suppose you want to set the value of id which should be greater than 100 only, you can write the logic inside the setter method. You can write the logic not to store the negative numbers in the setter methods.
3. It is a way to achieve data hiding in Java because other class will not be able to access the data through the private data members.
4. The encapsulate class is easy to test. So, it is better for unit testing.
5. The standard IDE's are providing the facility to generate the getters and setters. So, it is easy and fast to create an encapsulated class in Java.
6. If you don’t define the setter method in the class then the fields can be made read-only.
7. If you don’t define the getter method in the class then the fields can be made write-only.
8. It improves the maintainability of the application.

### Disadvantage of Encapsulation in Java



* + it increases the length of the code and slows shutdown execution.

### Data Hiding in Java



* Data hiding in Java is an important principle of object-oriented programming system (OOPs). It prevents to access data members (variables) directly from outside the class so that we can achieve security on data. This oops feature is called data hiding in Java.
* An outside person could not access our internal data directly or our internal data should not go out directly. After validation or authentication, the outside person can access our internal data.
* For example, after providing proper username and password, you can able to access your Gmail inbox information.

### How to achieve Data hiding programmatically?



* By declaring data members (variables) as private, we can achieve or implement data hiding. If the variables are declared as private in the class, nobody can access them from outside the class.

##### The biggest advantage of data hiding is we can achieve security.

**Key points:**

1. It is highly recommended to declare data members as private in the class.
2. A combination of data hiding and abstraction is nothing but encapsulation.

##### Encapsulation = Data Hiding + Abstraction

###### *If any component follows data hiding and abstraction, it is called an encapsulated component.*

**Tightly Encapsulated Class in Java**



* + If each variable is declared as private in the class, it is called tightly encapsulated class in Java. For tightly encapsulated class, we are not required to check whether class contains getter and setter method or not and whether these methods are declared as public or not.

##### Example:

public class Account

{

private double balance;

public double getbalance()

{

return balance;

}

}

**Q. Which of the following classes are tightly encapsulated?**



##### Example:

class A

{

private int x = 20;

}

class B extends A

{

int y = 50;

}

class C extends A

{

private int z = 10;

}

class P

{

int a = 10;

}

class Q extends P

{

private int b = 20;

}

class R extends Q

{

private int z = 30;

}



**Getter and Setter Method in Java**

### Getter method in Java



* A method which is used to retrieve/get the value of a variable or return the value of the private member variable is called getter method in Java.
* This method is also known as accessor method. For every private variable, we should create a getter method.
* Depending on the access level giving to the variable, we can set the access modifier of its getter method. If we

declare instance variables as private, we will have to add public getter methods for each one.

##### Syntax:

public returnType getPropertyName()

If the returnType is boolean, the getter method should be defined by convention as follows: public boolean isPropertyName()



**Setter method in Java**

### Setter method in Java



* + A method which is used for updating or setting the value of a variable is called setter method in Java.

This method is also known as mutator method.

* + By using the setter method, we can modify the value of a variable. Just like with the getter method, we should create a setter method for every variable in the class.

#### Syntax:

public void setPropertyName(dataType propertyValue)



## Naming convention of Getter and Setter method

### Naming convention of Getter and Setter method



The naming of setter and getter can be done by using the Java bean naming convention. getXxx() and setXxx() where Xxx is the name of the variable.

##### Example:

If the variable is the type of int, you will declare like this:

private int number; then appropriate getter and setter will be like this:

public getNumber() { }

public void setNumber(int number) { }

Similarly,

If the variable is a type of boolean then

private boolean rich; public getRich() { } public setRich() { }

##### Example:

package gettersetterProgram;

public class Addition

{

private int x, y, z; public void setX(int a){

x = a;

}

public void setY(int b) { y = b;

}

public void setZ(int c) { z = c;

public class MySum {



public static void main(String[] args)

{

Addition a = new Addition(); a.setX(10);

a.setY(20);

a.setZ(30);

a.sum();

}

}

}

void sum() {

int s = x + y + z; System.out.println("Sum: " +s);

}

}

##### Output:

Sum: 60

* Thus, the setter method is used to update or set the value of a variable whereas the getter method is used to read the value of a variable.

**Example**



/\* File name : EncapTest.java \*/ public class EncapTest {

private String name; private String idNum; private int age;

public int getAge() { return age;

}

public String getName() { return name;

}

public String getIdNum() { return idNum;

}

public void setAge( int newAge) {

age = newAge;

}

public void setName(String newName) { name = newName;

}

public void setIdNum( String newId) { idNum = newId;

}

}

The variables of the EncapTest class can be

accessed using the following program:

/\* File name : RunEncap.java \*/ public class RunEncap {

public static void main(String args[]) { EncapTest encap = new EncapTest(); encap.setName("James"); encap.setAge(20); encap.setIdNum("12343ms");

System.out.print("Name : " + encap.getName() + " Age : " + encap.getAge());

}

} **Output**

Name : James Age : 20

##### Example:

//A Account class which is a fully encapsulated class.

//It has a private data member and getter and setter methods. class Account {

//private data members private long acc\_no; private String name,email; private float amount;

//public getter and setter methods public long getAcc\_no() {

return acc\_no;

}

public void setAcc\_no(long acc\_no) {

this.acc\_no = acc\_no;

}

public String getName() { return name;

}



public void setName(String name) { this.name = name;

}

public String getEmail() {

return email;

}

public void setEmail(String email) { this.email = email;

}

public float getAmount() {

return amount;

}

public void setAmount(float amount) { this.amount = amount;

}

}

**Example: Continue…**



//A Java class to test the encapsulated class Account. public class TestEncapsulation {

public static void main(String[] args) {

//creating instance of Account class

Account acc=new Account();

//setting values through setter methods acc.setAcc\_no(7560504000L); acc.setName(“Waqeel Ahmed”); acc.setEmai[l(“waqeel@gmail](mailto:waqeel@gmail.com).c[om")](mailto:waqeel@gmail.com); acc.setAmount(500000f);

//getting values through getter methods

System.out.println(acc.getAcc\_no()+" "+acc.getName()+" "+acc.getEmail()+" "+acc.getAmount());

}

}

##### Output:

7560504000 Waqeel Ahme[d waqeel@gmail.com](mailto:waqeel@gmail.com) 500000.0

