**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.

Here, we will have the detailed learning of creating and using user-defined packages.

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.

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

File: Simple.java

**package** mypack;

**public** **class** Simple {

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

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

}

}

**Access Modifiers**

There are 4 types of java access modifiers:

* private
* default
* protected
* public

1) private access modifier

The private access modifier is accessible only within class.

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 compile time error.

**package** mypack;

**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. For example:

**package** mypack;

**class** A {

**private** A() {

System.***out***.println("hello");

}// 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.

2) default access modifier

If you don't use any modifier, it is treated as default. The default modifier is accessible only within package.

In this example, we have created two packages ‘pack’ and ‘mypack’. We are accessing the A class from outside its package, since A class is not public, so it cannot be accessed from outside the package.

//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.

3) protected access modifier

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.

Example of protected access modifier

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");

}

}

//save by B.java

**package** mypack;

**import** pack.\*;

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

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

B obj = **new** B();

obj.msg();

}

}

**Output:**

Hello

4) public access modifier

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

//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 **extends** A {

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

B obj = **new** B();

obj.msg();

}

}

**Output:**

Hello

Understanding all java access modifiers

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Options** | **Access Modifier** | **Private** | **No Modifier** | **Protected** | **Public** |
| **1** | **Same class** | Yes | Yes | Yes | Yes |
| **2** | **Same package subclass** | No | Yes | Yes | Yes |
| **3** | **Same package non-subclass** | No | Yes | Yes | Yes |
| **4** | **Different package subclass** | No | No | **Yes** | Yes |
| **5** | **Different package non-subclass** | No | No | No | Yes |

**Encapsulation in Java**

Encapsulation in java is a process of wrapping code and data together into a single unit, for example capsule i.e. mixed of several medicines.

encapsulation in java

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 fully encapsulated class.

Advantage of Encapsulation in java

By providing only setter or getter method, you can make the class read-only or write-only.

It provides you the control over the data. Suppose you want to set the value of id i.e. greater than 100 only, you can write the logic inside the setter method.

Simple example of encapsulation in java

Let's see the simple example of encapsulation that has only one field with its setter and getter methods.

//save as Student.java

**package** mypack2;

**public** **class** Student {

**private** String name;

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

}

//save as Test.java

**package** mypack2;

**class** Test {

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

Student s = **new** Student();

s.setName("vijay");

System.***out***.println(s.getName());

}

}

Output:

vijay

Option2 - Same package subclass:

**package** pack3;

**public** **class** Subclass {

**private** **int** a1;

**int** b1;

**protected** **int** c1;

**public** **int** d1;

}

**package** pack3;

**public** **class** SamePackage\_Subclass **extends** Subclass{

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

SamePackage\_Subclass n1 = **new** SamePackage\_Subclass();

// System.out.println(n1.a1);

System.***out***.println(n1.b1);

System.***out***.println(n1.c1);

System.***out***.println(n1.d1);

}

}

Option3: Same package non-subclass

**package** pack3;

**public** **class** NonSubclass {

**private** **int** a;

**int** b;

**protected** **int** c;

**public** **int** d;

}

**package** pack3;

**public** **class** SamePackage\_NonSubclass {

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

NonSubclass n1 = **new** NonSubclass();

// System.out.println(n1.a);

System.***out***.println(n1.b);

System.***out***.println(n1.c);

System.***out***.println(n1.d);

}

}

Option 4: Different package subclass

**package** pack3;

**public** **class** Subclass {

**private** **int** a1;

**int** b1;

**protected** **int** c1;

**public** **int** d1;

}

**package** pack2;

**import** pack3.Subclass;

**public** **class** DifferentPackage\_Subclass **extends** Subclass{

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

DifferentPackage\_Subclass n1 = **new** DifferentPackage\_Subclass();

// System.out.println(n1.a1);

// System.out.println(n1.b1);

System.***out***.println(n1.c1);

System.***out***.println(n1.d1);

Subclass n2 = **new** Subclass();

// System.out.println(n2.a1);

// System.out.println(n2.b1);

// System.out.println(n2.c1);

System.***out***.println(n2.d1);

}

}

Option5: Different package non-subclass

**package** pack3;

**public** **class** NonSubclass {

**private** **int** a;

**int** b;

**protected** **int** c;

**public** **int** d;

}

**package** pack2;

**import** pack3.NonSubclass;

**public** **class** DifferentPackage\_NonSubclass{

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

NonSubclass n1 = **new** NonSubclass();

// System.out.println(n1.a); //shows error as expected

// System.out.println(n1.b); //shows error as expected

// System.out.println(n1.c); //shows error as expected

System.***out***.println(n1.d); //does not show error as expected

}

}