Nesne Yönelimli Programlama

Encapsulation (Kapsülleme)

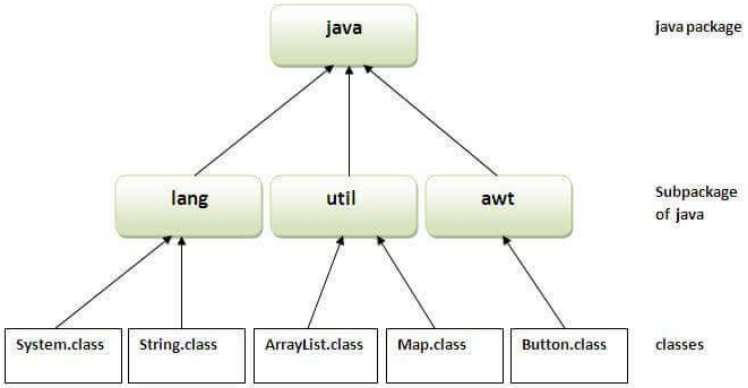
Hüseyin Ahmetoğlu

Java Package

- Java paketi, benzer türde sınıflar, arabirimler ve alt paketlerden oluşan bir gruptur.
- Java'daki paket, yerleşik paket ve kullanıcı tanımlı paket olmak üzere iki şekilde kategorize edilebilir.
- Java, lang, awt, javax, swing, net, io, util, sql gibi birçok yerleşik paket vardır.

Java Paketlemenin Avantajı

- 1) Java paketi, sınıfları ve arayüzleri kolayca korunabilmeleri için kategorize etmek için kullanılır.
- 2) Java paketi erişim koruması sağlar.
- 3) Java paketi adlandırma çakışmasını ortadan kaldırır.



Örnek

```java
//save as Simple.java
package mypack;
public class Simple{
 public static void main(String args[]){
    System.out.println("Welcome to package");
   }
}
```

Pakete başka bir paketten nasıl erişilir?

- import package.*;
- import package.classname;
- fully qualified name.

packagename.*

- Package. * Kullanırsanız, bu paketin tüm sınıflarına ve arayüzlerine erişilebilir ancak alt paketlere erişilemez.
- Import anahtar sözcüğü, başka bir paketin sınıflarını ve arayüzünü mevcut paket için erişilebilir hale getirmek için kullanılır.

```
//save by A.java
package pack;
public class A{
  public void msg(){System.out.println("Hello");}
}
```

```
Output:Hello
```

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

class B{
  public static void main(String args[]){
   A obj = new A();
   obj.msg();
  }
}
```

packagename.classname

- Package.classname dosyasını içe aktarırsanız, bu paketin yalnızca beyan edilen sınıfına erişilebilir.

```
//save by A.java

package pack;
public class A{
  public void msg(){System.out.println("Hello");}
}
```

```
Output:Hello
```

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

class B{
  public static void main(String args[]){
   A obj = new A();
   obj.msg();
  }
}
```

fully qualified name

- Tam nitelikli ad kullanırsanız, bu paketin yalnızca beyan edilen sınıfına erişilebilir. Artık içe aktarmaya gerek yok. Ancak sınıfa veya arayüze her eriştiğinizde tam nitelikli ad kullanmanız gerekir.

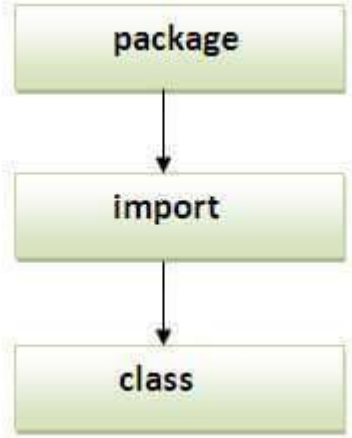
```
//save by A.java
package pack;
public class A{
  public void msg(){System.out.println("Hello");}
}
```

```
//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
```

Not

- Bir paketi içe aktarırsanız, alt paketlerin sınıfları ve arabirimleri hariç olmak üzere bu paketin tüm sınıfları ve arabirimi içe aktarılacaktır. Bu nedenle, alt paketi de içe aktarmanız gerekir.



Java'da alt paket

- Paketin içindeki paket, alt paket olarak adlandırılır. Paketi daha fazla kategorize etmek için oluşturulmalıdır.
- Bir örnek alalım, Sun Microsystem, System, String, Reader, Writer, Socket vb. Gibi birçok sınıf içeren java adında bir paket tanımladı. Bu sınıflar belirli bir grubu temsil eder, örneğin Reader ve Writer sınıfları Input / Output işlemi içindir, Socket ve ServerSocket sınıflar ağ iletişimi vb. içindir. Sun, java paketini lang, net, io vb. Gibi alt paketlere ayırdı ve Girdi / Çıktı ile ilgili sınıfları io paketine, Server ve ServerSocket sınıflarını net paketlere vb.

```
package com.javatpoint.core;
class Simple{
  public static void main(String args[]){
   System.out.println("Hello subpackage");
  }
}
```

Access Modifiers

- Java'da iki tür değiştirici vardır: erişim değiştiriciler ve erişilemeyen değiştiriciler.
- Java'daki erişim değiştiricileri, bir alanın, yöntemin, yapıcının veya sınıfın erişilebilirliğini veya kapsamını belirtir. Alanların, yapıcıların, yöntemlerin ve sınıfın erişim düzeyini üzerine erişim değiştiriciyi uygulayarak değiştirebiliriz.
- Dört tür Java erişim değiştiricisi vardır:
- **Private**: Bir **Private** değiştiricinin erişim düzeyi yalnızca sınıfın içindedir. Sınıf dışından erişilemez.
- **Default**: **Default** bir değiştiricinin erişim düzeyi yalnızca paketin içindedir. Paket dışından erişilemez. Herhangi bir erişim seviyesi belirtmezseniz, bu **Default** olacaktır.
- **Protected**: **Protected** bir değiştiricinin erişim düzeyi paketin içinde ve alt sınıf aracılığıyla paketin dışındadır. Alt sınıf yapmazsanız paket dışından erişilemez.
- **Public**: Bir **Public** değiştiricinin erişim düzeyi her yerdedir. Sınıf içinden, sınıf dışından, paket içinden ve paket dışından erişilebilir.

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

Private

- Bu örnekte, A ve Basit olmak üzere iki sınıf oluşturduk. Bir sınıf, private veri üyesini ve private yöntemi içerir. Bu private üyelere sınıf dışından erişiyoruz, dolayısıyla bir derleme zamanı hatası var.

```
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
   }
}
```

Private

- Herhangi bir sınıf yapıcısını Private yaparsanız, o sınıfın örneğini sınıfın dışından oluşturamazsınız. Örneğin:
- Bir sınıf, iç içe geçmiş sınıf dışında private veya protected olamaz

```
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
 }
}
```

Default

- Herhangi bir değiştirici kullanmazsanız, varsayılan olarak default olarak kabul edilir. default değiştiriciye yalnızca paket içinde erişilebilir. Paket dışından erişilemez. Özelden daha fazla erişilebilirlik sağlar.

```
//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
  }
}
```

Protected

- Protected erişim değiştiriciye paket içinde ve paketin dışından erişilebilir, ancak yalnızca miras yoluyla erişilebilir.
- Protected erişim değiştiricisi, veri üyesine, yönteme ve kurucuya uygulanabilir. Sınıfa uygulanamaz.
- Default değiştiriciden daha fazla erişilebilirlik sağlar.

```
//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

- Genel erişim değiştiricisine her yerden erişilebilir. Diğer tüm değiştiriciler arasında en geniş kapsama sahiptir.

```
//save by A.java

package pack;
public class A{
public void msg(){System.out.println("Hello");}
}
```

```
Output:Hello
```

```
//save by B.java

package mypack;
import pack.*;

class B{
  public static void main(String args[]){
   A obj = new A();
   obj.msg();
  }
}
```

Overriding ve Access Modifiers

- Herhangi bir yöntemi geçersiz kılıyorsanız, geçersiz kılınan yöntem (yani alt sınıfta bildirilen) daha kısıtlayıcı olmamalıdır.

```
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

- Java'da kapsülleme, kod ve verileri tek bir birime ile bir kapsül halinde sarma işlemidir.
- Sınıfın tüm veri üyelerini özel yaparak Java'da tamamen kapsüllenmiş bir sınıf oluşturabiliriz. Böylece, içindeki verileri ayarlamak ve almak için set ve get yöntemlerini kullanabiliriz.
- Size veriler üzerinde kontrol sağlar. Yalnızca 100'den büyük olması gereken id değerini ayarlamak istediğinizi varsayalım, mantığı setter yönteminin içine yazabilirsiniz. Negatif sayıların setter yöntemlerinde saklanmaması mantığını yazabilirsiniz.
- Java'da veri gizlemeyi başarmanın bir yoludur, çünkü diğer sınıflar özel veri üyeleri aracılığıyla verilere erişemeyecektir.
- Kapsülleme sınıfının test edilmesi kolaydır. Bu nedenle, birim testi için daha iyidir.

Encapsulation

```
//A Java class which is a fully encapsulated class.
//It has a private data member and getter and setter methods.
package com.javatpoint;
public class Student{
//private data member
private String name;
//getter method for name
public String getName(){
return name;
}
//setter method for name
public void setName(String name){
this.name=name
}
}
```

```
//A Java class to test the encapsulated class.
package com.javatpoint;
class Test{
public static void main(String[] args){
//creating instance of the encapsulated class
Student s=new Student();
//setting value in the name member
s.setName("vijay");
//getting value of the name member
System.out.println(s.getName());
}
}
```

Output:

```
vijay
```

Read-Only class

```
//A Java class which has only getter methods.
public class Student{
//private data member
private String college="AKG";
//getter method for college
public String getCollege(){
return college;
}
}
```

```
s.setCollege("KITE");//will render compile time error
```

Write-Only class

```
//A Java class which has only setter methods.
public class Student{
//private data member
private String college;
//getter method for college
public void setCollege(String college){
this.college=college;
}
}
```

```
System.out.println(s.getCollege());//Compile Time Error, because there is no such method
System.out.println(s.college);//Compile Time Error, because the college data member is private.
//So, it can't be accessed from outside the class
```

```
//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;
}

}
```

```java
//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("Sonoo Jaiswal");
    acc.setEmail("sonoojaiswal@javatpoint.com");
    acc.setAmount(500000f);
    //getting values through getter methods
    System.out.println(acc.getAcc_no()+" "+acc.getName()+" "+acc.getEmail()+" "+acc.getAmount());
}
}
```

Output:

```
7560504000 Sonoo Jaiswal sonoojaiswal@javatpoint.com 500000.0
```

KAYNAKLAR

- Java Tutorial | Learn Java - javatpoint. (2021, March 21). Retrieved from https://www.javatpoint.com/java-tutorial