Java & JEE Training

Day 11 – OOPs with Java Contd.

MindsMapped Consulting

Java & JEE Training

Review of OOPs concepts (Last class)

MindsMapped Consulting

Review…

- Difference between interfaces and abstract classes?
- When to use which?
- Multiple inheritance by implementing multiple interfaces is allowed. Why? Does it not cause ambiguity?

Point to ponder...

- Can we override static methods?

Point to ponder…

- Can we override static methods? No

Point to ponder…

- Can we override static methods? No
- What is Method Hiding? How is it different from Method Overriding?

Agenda

- ✓ Packaging Classes & Access Modifiers

Java & JEE Training

Packaging classes in Java

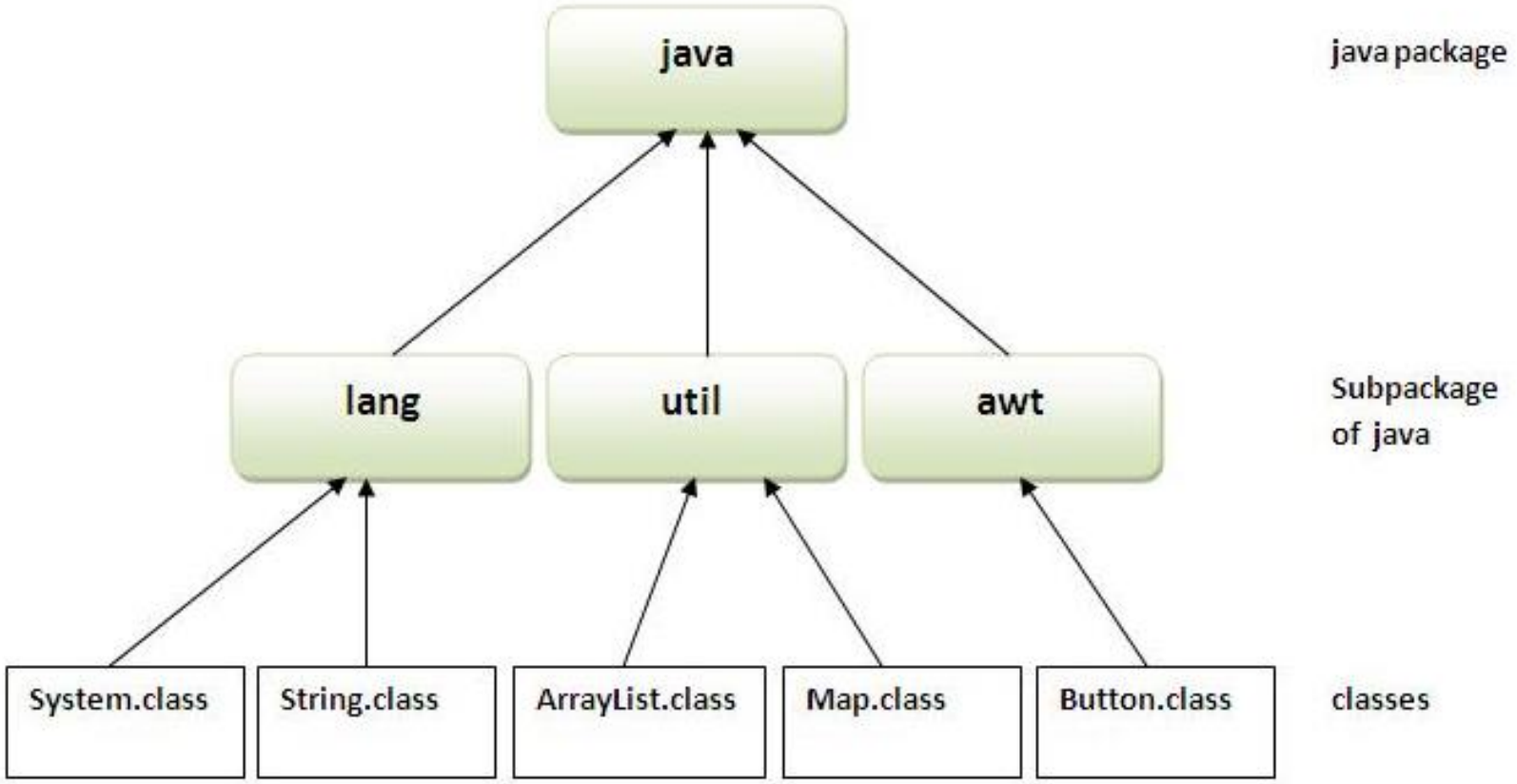
MindsMapped Consulting

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.

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

Java Package



Java Package Example

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

Accessing package

There are three ways to access the package from outside the package.

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

Accessing package: 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.

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

//save as B.java
package mypack;
import pack.*;

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

Accessing package: Using packagename.classname

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

```
//save as A.java
package pack;
public class A{
  public void msg(){System.out.println("Hello");}
}
//save as B.java
package mypack;
import pack.A;
class B{
  public static void main(String args[]){
   A obj = new A();
   obj.msg();
  }
}
```

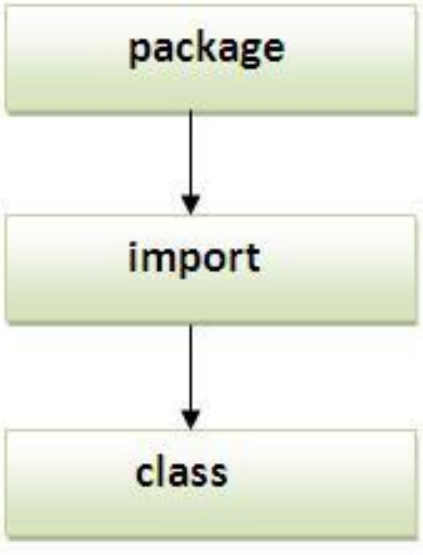
Accessing package: Using fully qualified name

- 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.
- It is generally used when two packages have same class name e.g. java.util and java.sql packages contain Date class.

```
//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();
  }
}
```

Packages.. rules

- **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.
- Java program structure



- The standard of defining package is domain.company.package e.g. com.mindsmapped.bean or org.wikipedia.dao.

More rules...

- **There can be only one public class in a java source file and it must be saved by the public class name.**

```
//save as C.java otherwise Compile Time Error

class A{}
class B{}
public class C{}
```

How to put 2 public classes in a package?

If you want to put two public classes in a package, have two java source files containing one public class, but keep the package name same.

```
//save as A.java
package dummy;
public class A{}

//save as B.java
package dummy;
public class B{}
```

Static import

- The import allows the java programmer to access classes of a package without package qualification whereas the static import feature allows to access the static members of a class without the class qualification

```
import static java.lang.System.*;
class StaticImportExample{
  public static void main(String args[]){

   out.println("Hello"); //Now no need of System.out
   out.println("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

Private access modifier

- The private access modifier is accessible only within class.

```
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 constructor 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
 }
}
```

Can a class be private?

Rule:A class cannot be private or protected except nested class.

Default access modifier

If you don't use any modifier, it is treated as **default** bydefault. The default modifier is accessible only within 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
  }
}
```

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.

```
//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();
  }
}
```

public access modifier

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

```
//save as A.java
package pack;
public class A{
public void msg(){System.out.println("Hello");}
}
//save as B.java
package mypack;
import pack.*;

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

Overriding rule...

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

```
class A{
protected void msg(){System.out.println("Hello java");}
}

public class Simple extends A{
void msg(){System.out.println("Hello java");}//Compile-time 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 capsule i.e. mixed of several medicines.

- The **Java Bean** class is the example of fully encapsulated class.
- **Advantages:**
  - 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.

Java Bean Example (Best Practice)

```
//save as Student.java
package com.dummy;
public class Student{
    private String name;

    public String getName(){
            return name;
    }
    public void setName(String name){
            this.name=name
    }
}
```