Packages in Java

Packages: Putting Classes Together

- One of the main features of OOP is its ability to **reuse the code already created**.
- One way is to inheritance – limited to reusing the classes within the program.
- What if we need to use classes from other programs?
- This can be accomplished in java by using **“Packages”**
- Packages are similar to “class libraries” in other languages.

Packages in java

- Packages are java’s way of grouping a variety of classes and/ or interfaces together.

- The grouping is usually done according to functionality.

- Packages act as containers for classes.

- Examples of packages:

  lang, **awt**, **util**, **applet**, **javax**, **swing**, **net**, **io**, **sql**,etc.

Advantages of packages

1. The classes contained in the package can be **easily reused**.

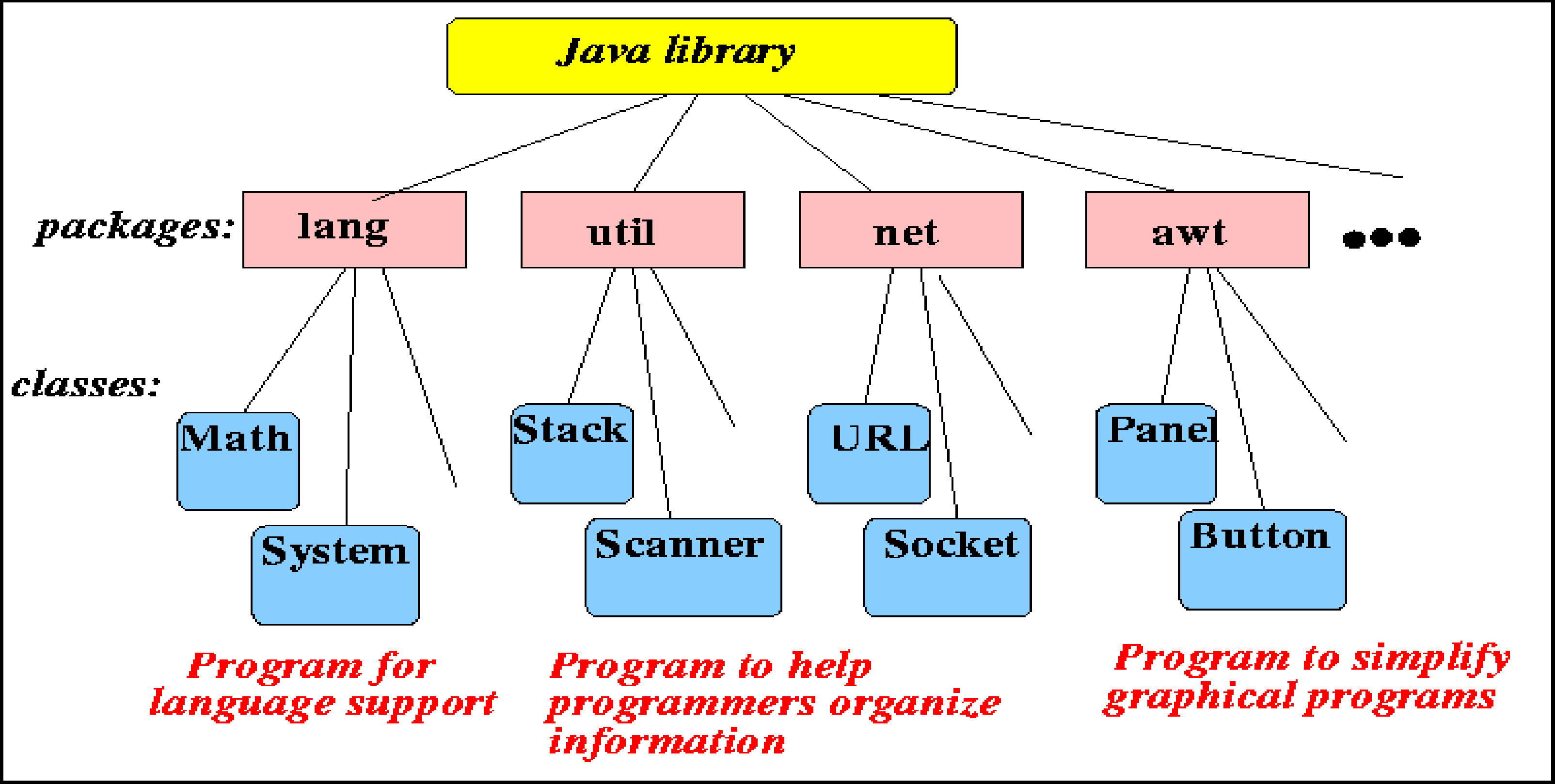
2. **Two classes** in two different packages **can have the same name**.

3. Package provide a way to hide classes thus preventing other programs or packages from accessing classes that are for internal use only.

4. Provide a way of separating “design” from “coding”.

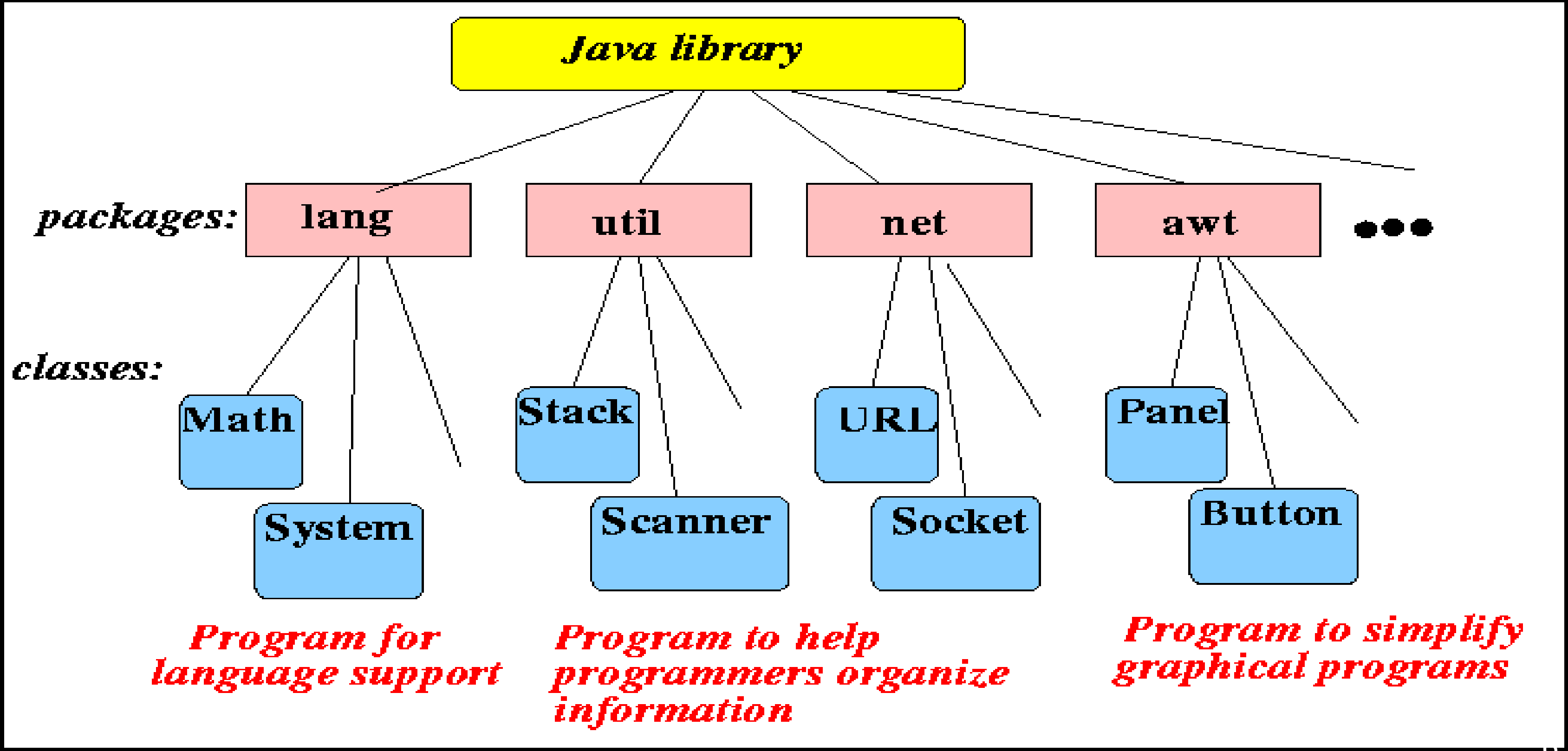
Different Types of Packages:

- There are two types of packages in Java:
  - **Built-in packages**
  - **User-defined packages**


Java library
packages:
lang
util
net
awt
•••
classes:
Math
System
Stack
Scanner
URL
Socket
Panel
Button
Program for language support
Program to help programmers organize information
Program to simplify graphical programs

Built-in packages

- They are also called as **Java *API* Packages**.
- Java API provides a large number of classes grouped into different packages according to functionality.



Java library
packages:
lang
util
net
awt
•••
classes:
Math
System
Stack
Scanner
URL
Socket
Panel
Button
Program for language support
Program to help programmers organize information
Program to simplify graphical programs

Java *API* Packages

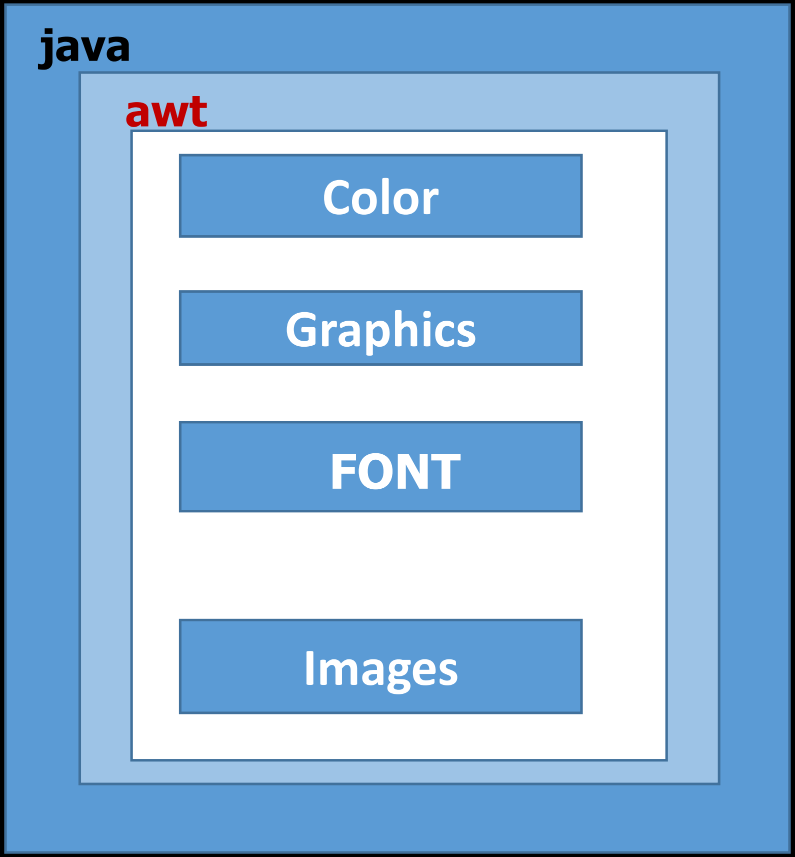
- **Java.lang:** lang stands for language.
  - This package has primary classes and interfaces essential for developing a basic Java program.
  - It consists of wrapper classes , **String, StringBuffer, StringBuilder** classes to handle strings, thread class.
- **Java.util:** util stands for utility.
  - This package contains useful classes and interfaces like Stack, **LinkedList, Hashtable, Vector, Arrays, Date and Time classes**.
- **Java.io:** io stands for input and output.
  - This package contains **streams**. A stream represents flow of data from one place to another place. Streams are useful to store data in the form of files and also to perform input-output related tasks.

Java *API* Packages

- **Java.awt:** awt stands for **abstract window toolkit**.
  - This helps to develop GUI(Graphical user Interfaces). It consists of , classes which are useful to provide action for components like **push buttons, radio buttons, menus** etc.
- **Java.net:** net stands for network.
  - Client-Server programming can be done by using this package. Classes related to obtaining authentication for network, client and server to establish communication between them are also available in java.net package.
- **Java.applet:**
  - applets are programs which come from a server into a client and get executed on the client machine on a network. **Applet** class of this package is useful to create and use applets.

Using System packages

- The package name **java** contains package **awt**, which contains various classes required for implementing GUI.



- Hierarchical representation of **java.awt** package.

Using System Packages

- Classes stored in packages can be accessed in two ways

1. Use the **fully qualified class name** of class that we want to use. This is done by using package name containing the class name using dot operator.
   - Example :

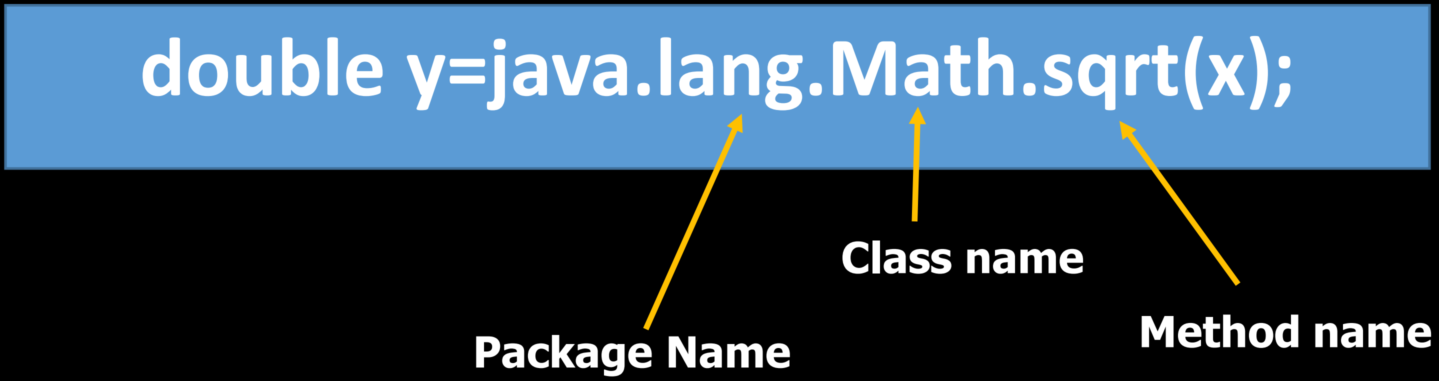
     import java.awt.Color;

2. In many situations we might use class name in number of places in the program or we may like to use many classes of that package then-
   - Example:

     import java.awt.*;

Naming Conventions

- Package names are written in lower case to avoid conflict with the names of classes or interfaces.
- All class names begin with uppercase letters and methods with lower case letters.



```
double y=java.lang.Math.sqrt(x);
```

User-Defined packages:

- The users of the Java language can also create their own packages.
- They are called **user-defined packages**.
- User-defined packages can also be imported into other classes and used exactly in the same way as the Built-in packages.

Creating Packages

- **package packagename;** **//to create a package**
- **package packagename.subpackagename**;
  - //to create a sub package within a package.

  e.g.: package pack;
- The first statement in the program must be package statement while creating a package.
- While creating a package except instance variables, declare **all the members and the class itself as public** then only the public members are available outside the package to other programs.

A program to create a package pack with Addition class.

```
package pack;
public class Addition
{
        private double d1,d2;
        public Addition(double a, double b)  {
        d1 = a;
        d2 = b; }
 public void sum()
 {  System.out.println ("Sum is : " + (d1+d2) );   }
}
```

- Compile and save **.java and .class file in pack** directory (folder).

Using Package

- Write a java program to use the already created package.

```
import pack.Addition;
class Use
{
	public static void main(String args[])
	{
		Addition A=new Addition(10.5,5.2);
		A.sum();
	}
}
```

- Compile and run Use.java program

Steps to create a package in java

1. Declare the package at the beginning of a file using form

   package *packagename;*

2. Define a class that is be put in the package and declare it **public.**
3. Create a subdirectory under the directory where main source files are stored.
4. Store the listing as the classname.java file in the subdirectory created.
5. Compile the file. This creates **.class** file in the subdirectory.

- Access Protection

	Public	Protected	Friendly	Private protected	Private
Same class	Yes	Yes	Yes	Yes	Yes
Subclass in same package	Yes	Yes	Yes	Yes	No
Other classes in same package	Yes	Yes	Yes	No	No
Subclasses in other package	Yes	Yes	No	[✓ Yes]	No
Non subclasses in other package	Yes	No	No	No	No

Static Import

- Eliminates the need of qualifying a static member with the class name.
- The static import declaration is similar to that of import.
- We can use static import statement to import static members from classes and use them without qualifying class name.

import **static** package-name.sub-package-name.**class-name.Staticmember-name;**

Or

import static package-name.subpackage-name. class-name.*;

Program Without Static Import

```
import  java.lang.Math.*;
class Test{
    public static void main(String[] args)
    {
        System.out.println(Math.sqrt(4));
        System.out.println(Math.pow(2, 2));
        System.out.println(Math.abs(6.3));
    }
}
```

Program With Static Import

```
import static java.lang.Math.*;
class Test2 {
    public static void main(String[] args)
    {
         out.println(sqrt(4));
        out.println(pow(2, 2));
        out.println(abs(6.3));
    }
}
```