VIPS Technical Campus
योग: कर्मसु कौशलम्
IN PURSUIT OF PERFECTION

# SCHOOL OF ENGINEERING AND TECHNOLOGY

# Course : OBJECT ORIENTED PROGRAMMING

Paper Code: AIML-202,IIOT-202,AIDS-202

Faculty : Dr. Shivanka

Assistant Professor

VIPS

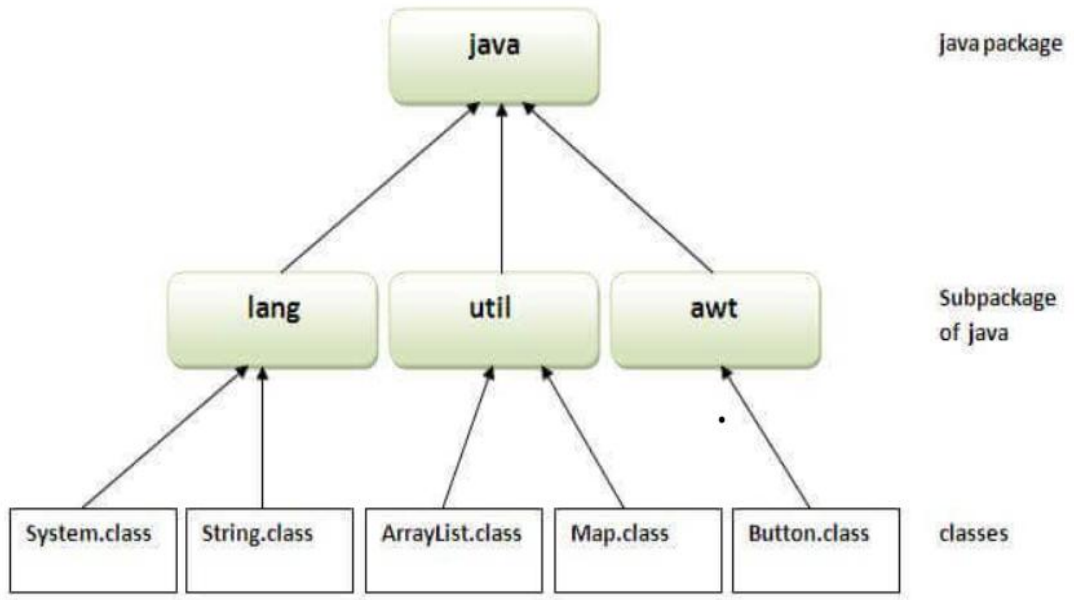
## Java Package

- 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.

# Java Package

- 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.

## Concept

Packages are nothing more than the way we organize files into different directories according to their functionality, usability as well as category they should belong to .

**A Java package is a Java programming language mechanism for organizing classes into namespaces.**

Java source files belonging to the same category or providing similar functionality can include a **package** statement at the top of the file to designate the package for the classes the source file defines.

**Java packages can be stored in compressed files called JAR files.**

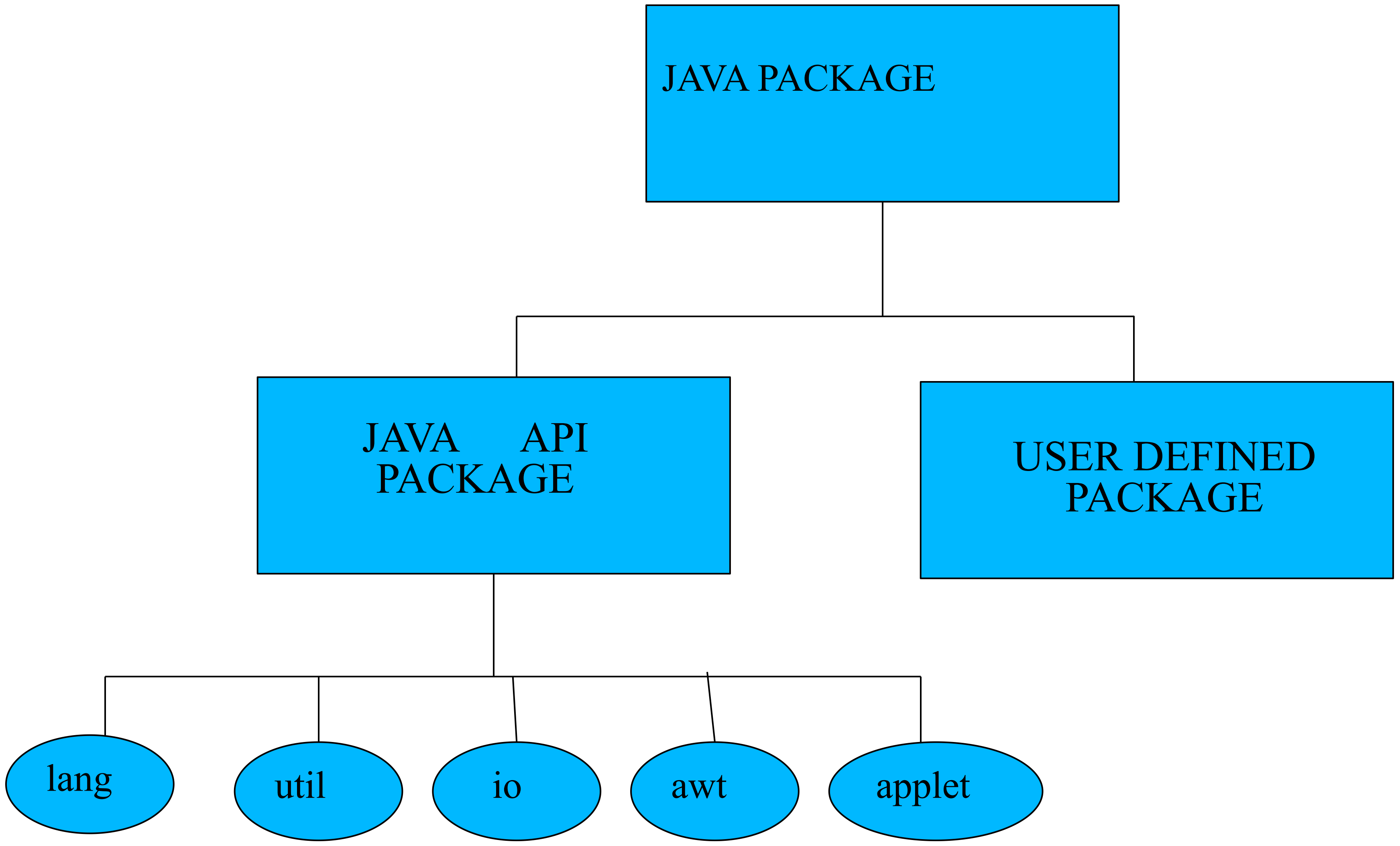
## Concept

Packaging also help us to avoid class name collision when we use the same class name as that of others.

**For example, if we have a class name called "Vector", its name would crash with the Vector class from JDK. However, this never happens because JDK uses java.util as a package name for the Vector class (java.util.Vector ).**

Understanding the concept of a package will also help us manage and use files stored in jar files in more efficient ways.

# Concept


JAVA PACKAGE
JAVA API PACKAGE
USER DEFINED PACKAGE
lang
util
io
awt
applet

## Using Packages

- To use a package inside a Java source file, it is convenient to import the classes from the package with an import statement.

```
import java.awt.*;
import java.awt.Colour;
```

- The above first statement imports all classes from the java.awt package and second will import only Colour class.

## Creation Of Jar Files

- In Java source files the package the file belongs to is specified with the package keyword .

  **package java.awt.event;**

- JAR Files are created with the jar command-line utility.

- The command **“jar cf myPackage.jar *.class”** compresses all *.class files into the JAR file *myPackage.jar*.

## Package Naming Conventions

- Packages are usually defined using a hierarchical naming pattern, with levels in the hierarchy separated by periods (.) .
- **Although packages lower in the naming hierarchy are often referred to a "subpackages" of the corresponding packages higher in the hierarchy, there is no semantic relationship between packages.**
- Package names should be all lowercase characters whenever possible.
- **Frequently a package name begins with the top level domain name of the organization and then the organization's domain and then any subdomains listed in reverse order.**
- The organization can then choose a specific name for their package.

# Simple example of java package

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**

# Simple example of java package

- The -d switch specifies the destination where to put the generated class file. You can use any directory name like /home (in case of Linux), d:/abc (in case of windows) etc. If you want to keep the package within the same directory, you can use . (dot).

❖**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.

# Creating a Package

```
package firstpack;

public class Firstclass
{

}
class Secondclass
{

}

class Thirdclass
{

}
```

1. Declare the pkg. at the beginning of the file.
2. Save the file with the same name as the class name.
3. Save it in the directory having same name as the package name and that directory should be the subdirectory of the directory where all the source files are saved.
4. Compile the file. It will create .class file in the subdirectory.

Java also support the concept of pkg. hierarchy.
Package p1.p2.p3;
It should be saved in directory p1/p2/p3.

Java pkg. file have more than one class definition. But only one class will be declared as public. Compilation will leads to independent .class files corresponding to each class.

# Using user defined Package

```
package p1;

public class ClassA
{

public void displayA()
{
  System.out.println("Class A");
}

}
```

```
import p1.ClassA;

class Pkgtest
{

public static void main(String[] args)
{
ClassA ob = new ClassA();
ob.displayA();
}

}
```

## 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.

## 1) Using packagename.*

1) 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 of package that import the packagename.*

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

```
//save by B.java
package mypack;
import pack.*;
class B{
  public static void main(String args[]){
   A obj = new A();
   obj.msg();
  }
}
Output:Hello
```

# 2.Using packagename.classname

2) Using packagename.classname

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

```
//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

## 3.Using fully qualified name

3 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.

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

## Example of package by import fully qualified

```
//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

# Thank You