Java Package

- PACKAGE in Java is a collection of classes, sub-packages, and interfaces.
- It helps organize your classes into a folder structure and make it easy to locate and use them. More importantly, it helps improve code reusability.
- Each package in Java has its unique name and organizes its classes and interfaces into a separate namespace, or name group.
- Although interfaces and classes with the same name cannot appear in the same package, they can appear in different packages. This is possible by assigning a separate namespace to each Java package.

Java Package

- Advantage of Java Package

1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.
2) Reusability: Reusability of code is one of the most important requirements in the software industry. Reusability saves time, effort and also ensures consistency. A class once developed can be reused by any number of programs wishing to incorporate the class in that particular program.
3) Java package provides access protection.
4) Java package removes naming collision.

Java Package

- Easy to locate the files.
- In real life situation there may arise scenarios where we need to define files of the same name. This may lead to “name-space collisions”. Packages are a way of avoiding “name-space collisions”.

Java Package

Predefined Packages in Java (Built-in Packages)

- Predefined packages in java are those which are developed by Sun Microsystem. They are also called built-in packages in java.
- These packages consist of a large number of predefined classes, interfaces, and methods that are used by the programmer to perform any task in his programs.

Java Package

- Java APIs contains the following predefined packages, as shown in the below figure:

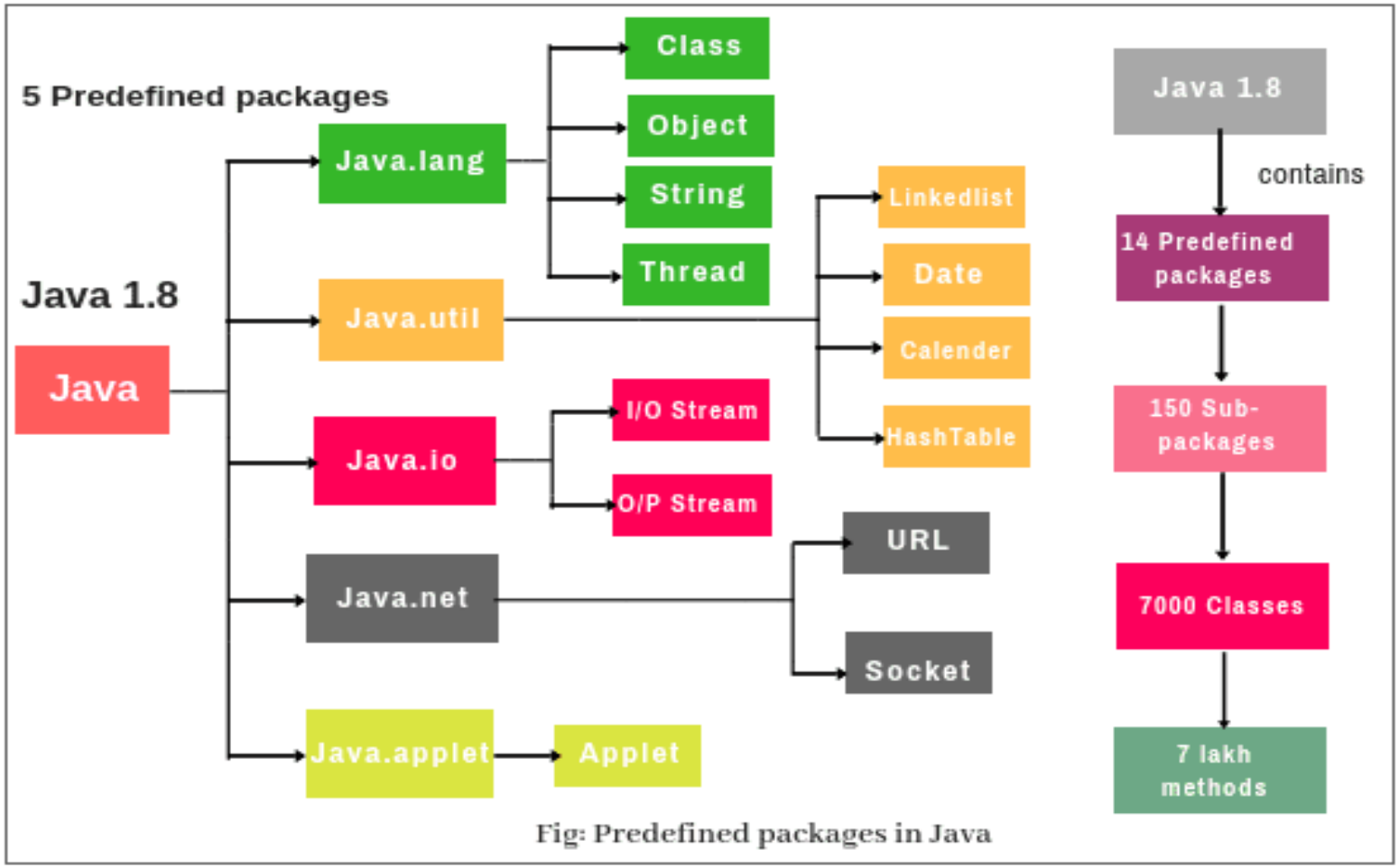


Fig: Predefined packages in Java

Java Package

- Core packages:

1. Java.lang: lang stands for language.

- The Java language package consists of java classes and interfaces that form the core of the Java language and the JVM.
- It is a fundamental package that is useful for writing and executing all Java programs.
- Examples are classes, objects, String, Thread, predefined data types, etc.
- It is imported automatically into the Java programs.

Java Package

2. Java.io: io stands for input and output.

- It provides a set of I/O streams that are used to read and write data to files.
- A stream represents a flow of data from one place to another place.

3. Java util: util stands for utility. It contains a collection of useful utility classes and related interfaces that implement data structures like LinkedList, Dictionary, HashTable, stack, vector, Calender, data utility, etc.

Java Package

4. Java.net: net stands for network. It contains networking classes and interfaces for networking operations. The programming related to client-server can be done by using this package.

- Window Toolkit and Applet:

1. Java.awt: awt stands for abstract window toolkit. The Abstract window toolkit packages contain the GUI(Graphical User Interface) elements such as buttons, lists, menus, and text areas. Programmers can develop programs with colorful screens, paintings, and images, etc using this package.

Java Package

2. Java.awt.image: It contains classes and interfaces for creating images and colors.
3. Java.applet: It is used for creating applets. Applets are programs that are executed from the server into the client machine on a network.
4. Java.text: This package contains two important classes such as DateFormat and NumberFormat. The class DateFormat is used to format dates and times. The NumberFormat is used to format numeric values.
5. Java.sql: SQL stands for the structured query language. This package is used in a Java program to connect databases like Oracle or Sybase and retrieve the data from them.

Defining a Package

- This is the general form of the package statement:

```
package pkg;
```

- Here, pkg is the name of the package.
- For example, the following statement creates a package called MyPackage.

```
package MyPackage;
```

- You can create a hierarchy of packages.
- To do so, simply separate each package name from the one above it by use of a period. The general form of a multileveled package statement is shown here:

```
package pkg1[.pkg2[.pkg3]];
```

- A package hierarchy must be reflected in the file system of your Java development system

Creation of a Package

- Creating a package is a simple task as follows
- Choose the name of the package
- Include the package command as the first line of code in your Java Source File.
- The Source file contains the classes, interfaces, etc you want to include in the package
- Compile to create the Java packages

Creation of a Package

- Consider the following package program in Java:
- package firstpack;
- class PackExample{

```
public void m1(){
        System.out.println("m1 of PackExample");
}
public static void main(string args[]){
        PackExample 1 obj = new PackExample();
        obj.m1();
}
```

- }

Creation of a Package

- Here,
- To put a class into a package, at the first line of code define package firstpack
- Create a class PackExample
- Defining a method m1 which prints a line.
- Defining the main method
- Creating an object of class c1
- Calling method m1

How to compile package in Java

•Compile the application:

javac -d directory . javafilename

1.Here, javac means java compiler.

2.-d means directory. It creates the folder structure.

3..(dot) means the current directory. It places the folder structure in the current working directory.

•For example: javac -d.Example.java

// Here, Example.java is the file name.

- So in this way, you must compile application if the application contains a package statement.
- To run the compiled class that we compiled using above command, we need to specify package name too. Use the below command to run the class file.
- java firstpack.PackExample

Importing of Packages

- To import java package into a class, we need to use java import keyword which is used to access package and its classes into the java program.
- Use import to access built-in and user-defined packages into your java source file so that your class can refer to a class that is in another package by directly using its name.
- There are 3 different ways to refer to any class that is present in a different package:

1. without import the package - fully qualified name.
2. import package with specified class - import package.classname;
3. import package with all classes - import package.*;

Importing of Packages

1. Accessing package without import keyword

- If you use fully qualified name to import any class into your program, then only that particular class of the package will be accessible in your program, other classes in the same package will not be accessible.
- For this approach, there is no need to use the import statement.
- But you will have to use the fully qualified name every time you are accessing the class or the interface.
- This is generally used when two packages have classes with same names. For example: java.util and java.sql packages contain Date class.

Importing of Packages

- Example
- In this example, we are creating a class A in package pack and in another class B, we are accessing it while creating object of class A.
- //save by A.java
- package pack;

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

Importing of Packages

- Example
- //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();
- }
- }

Importing of Packages

2. Import the Specific Class

- Package can have many classes but sometimes we want to access only specific class in our program in that case, Java allows us to specify class name along with package name.
- If we use import packagename.classname statement then only the class with name classname in the package will be available for use.

Importing of Packages

- Example
- package pack;
- public class Demo {
- public void msg() {
- System.out.println("Hello");
- }
- }
- class Test {
- public static void main(String args[]) {
- Demo obj = new Demo();
- obj.msg();
- }
- }

Importing of Packages

3. Import all classes of the package

- If we use packagename.* statement, then all the classes and interfaces of this package will be accessible but the classes and interface inside the sub-packages will not be available for use.

- The import keyword is used to make the classes of another package accessible to the current package.

Importing of Packages

- Example
- package learnjava;
- public class First{
- public void msg() {
- System.out.println("Hello");
- }
- }
- import learnjava.*;
- class Second {
- public static void main(String args[]) {
- First obj = new First();
- obj.msg();
- }
- }

Access Control

- Classes and packages are both means of encapsulating and containing the name space and scope of classes, variables and methods:

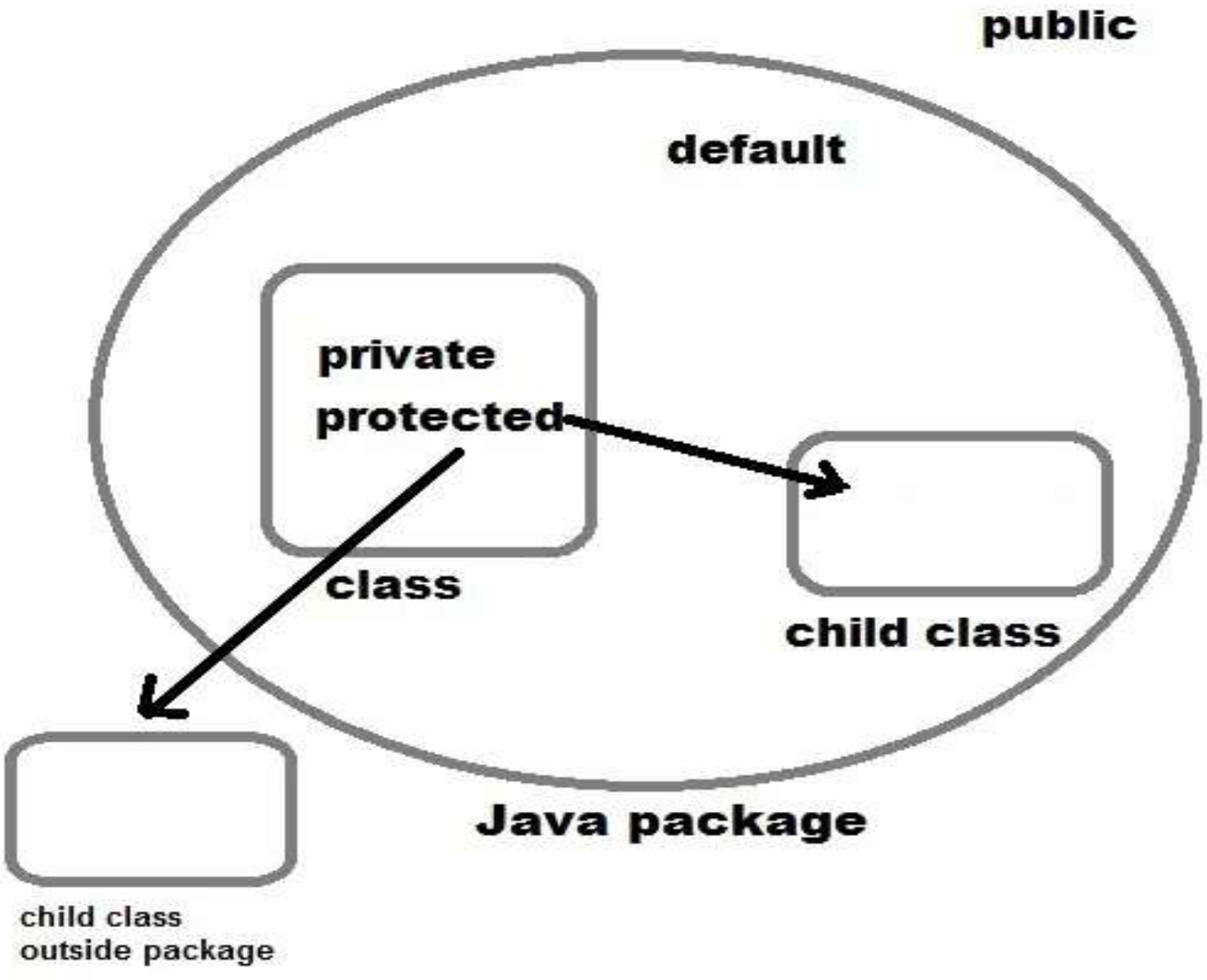
  1) packages act as a container for classes and other packages

  2) classes act as a container for data and code
- Access control is set separately for classes and class members.

Access Control

- Access modifiers are keywords in Java that are used to set accessibility.
- An access modifier restricts the access of a class, constructor, data member and method in another class.
- Java language has four access modifier to control access level for classes and its members.
- Default: Default has scope only inside the same package
- Public: Public has scope that is visible everywhere
- Protected: Protected has scope within the package and all sub classes
- Private: Private has scope only within the classes

Access Control



Access Control: Classes

- Two levels of access:
- 1) A class available in the whole program:

```
public class MyClass { … }
```

- 2) A class available within the same package only:

```
class MyClass { … }
```

Access Control: Members

- Four levels of access:

  1) a member is available in the whole program:

  ```
  public int variable;
  public int method(…) { … }
  ```

- 2) a member is only available within the same class:

  ```
  private int variable;
  private int method(…) { … }
  ```

Access Control: Members

3) a member is available within the same package (default access):

```
int variable;
int method(…) { … }
```

- 4) a member is available within the same package as the current class, or within its sub-classes:

```
protected int variable;
protected int method(…) { … }
```

- The sub-class may be located inside or outside the current package.

Access Control Summary

- Complicated?
- Any member declared public can be accessed from anywhere.
- Any member declared private cannot be seen outside its class.
- When a member does not have any access specification (default access), it is visible to all classes within the same package.
- To make a member visible outside the current package, but only to subclasses of the current class, declare this member protected.

Table: Access Control

	private	default	protected	public
same class	yes	yes	yes	yes
same package subclass	no	yes	yes	yes
same package non-sub-class	no	yes	yes	yes
different package sub-class	no	no	yes	yes
different package non-sub-class	no	no	no	yes

Example: Access 1

- Access example with two packages p1 and p2 and five classes.
- A public Protection class is in the package p1.
- It has four variables with four possible access rights:

```
package p1;
public class Protection {
        int n = 1;
        private int n_pri = 2;
        protected int n_pro = 3;
        public int n_pub = 4;
```

Example: Access 2

```
    public Protection() {
        System.out.println("base constructor");
        System.out.println("n = " + n);
        System.out.println("n_pri = " + n_pri);
        System.out.println("n_pro = " + n_pro);
        System.out.println("n_pub = " + n_pub);
    }
}
```

- The rest of the example tests the access to those variables.

Example: Access 3

- Derived class is in the same p1 package and is the sub-class of Protection.
- It has access to all variables of Protection except the private n_pri:

```
package p1;
    class Derived extends Protection {
        Derived() {
        System.out.println("derived constructor");
        System.out.println("n = " + n);
        System.out.println("n_pro = " + n_pro);
        System.out.println("n_pub = " + n_pub);
    }
}
```

Example: Access 4

- SamePackage is in the p1 package but is not a sub-class of Protection.
- It has access to all variables of Protection except the private n_pri:

```
package p1;
class SamePackage {
        SamePackage() {
                Protection p = new Protection();
                System.out.println("same package constructor");
                System.out.println("n = " + p.n);
                System.out.println("n_pro = " + p.n_pro);
                System.out.println("n_pub = " + p.n_pub);
        }
}
```

Example: Access 5

- Protection2 is a sub-class of p1.Protection, but is located in a different package – package p2.
- Protection2 has access to the public and protected variables of Protection. It has no access to its private and default-access variables:

```
package p2;
class Protection2 extends p1.Protection {
        Protection2() {
                System.out.println("derived other package");
                System.out.println("n_pro = " + n_pro);
                System.out.println("n_pub = " + n_pub);
        }
}
```

Example: Access 6

- OtherPackage is in the p2 package and is not a sub-class of p1.Protection.
- OtherPackage has access to the public variable of Protection only. It has no access to its private, protected or default-access variables:

```
class OtherPackage {
        OtherPackage() {
                p1.Protection p = new p1.Protection();
                System.out.println("other package constructor");
                System.out.println("n_pub = " + p.n_pub);
        }
}
```

Example: Access 7

- A demonstration to use classes of the p1 package:

```
package p1;
    public class Demo {
        public static void main(String args[]) {
        Protection ob1 = new Protection();
        Derived ob2 = new Derived();
        SamePackage ob3 = new SamePackage();
    }
}
```

Example: Access 8

- A demonstration to use classes of the p2 package:

```
package p2;

    public class Demo {
        public static void main(String args[]) {
        Protection2 ob1 = new Protection2();
        OtherPackage ob2 = new OtherPackage();
    }
}
```

Import Statement

- The import statement occurs immediately after the package statement and before the class statement:

```
package myPackage;
import otherPackage1;otherPackage2.otherClass;
class myClass { … }
```

- The Java system accepts this import statement by default:

```
import java.lang.*;
```

- This package includes the basic language functions. Without such functions,Java is of no much use.

Name Conflict 1

- Suppose a same-named class occurs in two different imported packages:

```
import otherPackage1.*;
import otherPackage2.*;
class myClass { … otherClass … }
```

```
package otherPackage1;
class otherClass { … }
```

```
package otherPackage2;
class otherClass { … }
```

Name Conflict 2

- Compiler will remain silent, unless we try to use otherClass.
- Then it will display an error message.
- In this situation we should use the full name:

```
import otherPackage1.*;
import otherPackage2.*;
        class myClass {
                …
otherPackage1.otherClass
        …
otherPackage2.otherClass
        …
}
```

Short versus Full References

- Short reference:

```
import java.util.*;
class MyClass extends Date { … }
```

- Full reference:

```
class MyClass extends java.util.Date { … }
```

- Only the public components in imported package are accessible for nonsub-classes in the importing code!

Example: Packages 1

- A package MyPack with one public class Balance. The class has two same-package variables: public constructor and a public show method.
- package MyPack;

```
public class Balance {
        String name;
        double bal;
        public Balance(String n, double b) {
                name = n; bal = b;
                }
        public void show() {
        if (bal<0) System.out.print("-->> ");
        System.out.println(name + ": $" + bal);
    }
}
```

Example: Packages 2

- The importing code has access to the public class Balance of the MyPack package and its two public members:

```
import MyPack.*;
class TestBalance {
      public static void main(String args[]) {
      Balance test = new Balance("J. J. Jaspers", 99.88);
      test.show();
      }
}
```