CS6308- Java Programming

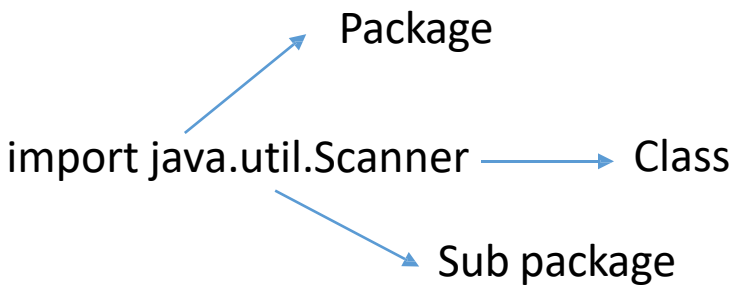
V P Jayachitra
Assistant Professor
Department of Computer Technology
MIT Campus
Anna University

Syllabus

MODULE III JAVA OBJECTS – 2	L	T	P	EL
	3	0	4	3
Inheritance and Polymorphism – Super classes and sub classes, overriding, object class and its methods, casting, instance of, Array list, Abstract Classes, Interfaces, Packages, Exception Handling				
SUGGESTED ACTIVITIES : • flipped classroom • Practical - implementation of Java programs – use Inheritance, polymorphism, abstract classes and interfaces, creating user defined exceptions • EL – dynamic binding, need for inheritance, polymorphism, abstract classes and interfaces				
SUGGESTED EVALUATION METHODS: • Assignment problems • Quizzes				

Packages

- **What is a package?**
  - Package is a mechanism to group related classes and interfaces.
- **Why packages?**
  - Organize: To group related classes and interfaces
  - Reusability: To import the required class to use
  - Encapsulation: To control accessibility of the classes and interfaces
  - Name conflicts: To avoid name conflict for same class in different packages.



Types of packages

- **User defined package:**
  - The package user create is called user-defined package.
- **Built-in package:**
  - Predefined package like java.io.*, java.lang.*

Packages

- Packages are containers for classes.
- They are used to keep the class name space compartmentalized.
  - For example, a package allows user to create a class named List, which can store in new package without concern that it will collide with some other class named List stored elsewhere.
- Packages are stored in a hierarchical manner and are explicitly imported into new class definitions.
- Without package:The name of each example class was taken from the same name space. This means that a unique name had to be used for each class to avoid name collisions.
  - Need some way to be assured that the name you choose for a class will be reasonably unique and not collide with class names chosen by other programmers.
- The package is both a naming and a visibility control mechanism.
- You can define classes inside a package that are not accessible by code outside that package

Defining a Package

- Include a package command as the first statement in a Java source file.
- Any classes declared within that file will belong to the specified package.
- The package statement defines a name space in which classes are stored.
  - Omit the package statement, the class names are put into the default package, which has no name.
- 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;

Package

- The .class files for any classes being part of mypackage must be stored in a directory called mypackage.
- i.e directory name must match the package name exactly.
- More than one file can include the same package statement.
- Create a hierarchy of packages by use of a period.
  - The general form of a **multileveled package** statement is shown here:
  - package pkg1[.pkg2[.pkg3]];
  - a package declared as package a.b.c; needs to be stored in a\b\c in a Windows environment.

Access Modifier

Public: accessible from anywhere

- Visibility: Accessible from any other class in any package.
- Usage:
  - Can be applied to classes, methods, and variables.
  - When a class is marked public, it can be accessed from any other class, regardless of the package.
  - When a method or variable is marked public, it can be accessed from any other class.
- Allowed:
  - For top-level classes, inner classes, methods, fields and constructors

Access Modifier

Private: accessible only within the same class.

- Visibility: Accessible only within the same class.
- Usage:
  - Can be applied to methods and variables.
  - A private member is not accessible from any other class, including subclasses.
  - Private methods cannot be overridden in subclasses.
  - a subclass cannot override a non-private method and make the new method private.
- Allowed:
  - For inner classes, methods, fields and constructors
- Not allowed:
  - Top-level classes

Access Modifier

Protected: access within the same package and subclasses

- **Visibility:** Accessible within the same class, subclasses, and classes in the same package**.**
- **Usage:**
  - Can be applied to methods and variables.
  - A protected member is accessible in subclasses even if they are in different packages.
  - A subclass can override a protected method or variable.
- Allowed:
  - For inner classes, methods, fields and constructors
- Not allowed:
  - Top-level classes

Access Modifier

default: access within the same package

- **Visibility:** Accessible within the same package**.**
- **Usage:**
  - When no access modifier is specified, it is considered as default
  - A default member is accessible in subclasses only if they are in same packages.
  - A subclass can override a default method or variable.
- Allowed:
  - For Top-level classes, inner classes, methods, fields and constructors

Packages and Member Access

- Classes and packages are both means of encapsulating and containing the name space and scope of variables and methods.
- Packages act as containers for classes and other subordinate packages.
- Classes act as containers for data and code.
- The class is Java's smallest unit of abstraction.
- **Java addresses four categories of visibility for class members:**
  - Subclasses in the same package
  - Non-subclasses in the same package
  - Subclasses in different packages
  - Classes that are neither in the same package nor subclasses

Packages and Member Access

Class Member Access

Description	Private	Default(No modifier)	Protected	Public
Same Class	Yes	Yes	Yes	Yes
Same Package Subclass	No	Yes	Yes	Yes
Same Package Non-Subclass	No	Yes	Yes	Yes
Different Package subclass	No	No	Yes	Yes
Different Package Non-subclass	No	No	No	Yes

```java
package p1;

class DerivedClass extends p1.AccessModifier {
    DerivedClass() {
        System.out.println("derived constructor");
        System.out.println("defaultVariable = " + defaultVar);
        // System.out.println("privateVariable = " + privateVar); // private members are not accessible
        System.out.println("protectedVariable = " + protectedVar);
        System.out.println("publicVariable = " + publicVar);
    }
}
```

```java
package p1;

public class AccessModifier {
    int defaultVar = 1;
    private int privateVar = 2;
    protected int protectedVar = 3;
    public int publicVar = 4;

    public AccessModifier() {
        System.out.println("defaultVar = " + defaultVar);
        System.out.println("privateVar = " + privateVar);
        System.out.println("protectedVar = " + protectedVar);
        System.out.println("publicVar = " + publicVar);
    }
}
```

```java
package p1;
//same package but some other class
class SomeClass{
    SomeClass() {
        p1.AccessModifier obj = new p1.AccessModifier();
        System.out.println("same package constructor");
        System.out.println("defaultVar = " + obj.defaultVar);
        // System.out.println("privateVar = " + obj.privateVar); //  private members are not accessible
        System.out.println("protectedVar = " + obj.protectedVar);
        System.out.println("publicVar = " + obj.publicVar);
    }
}
```

This is file SomeClass.java:

```java
package p2;

import p1.AccessModifier;

class DifferentPackage extends AccessModifier {
    DifferentPackage() {
        System.out.println("constructor");
        // System.out.println("defaultVariable = " + defaultVar); // default members are not accessible from different packages
        // System.out.println("privateVariable = " + privateVar); //  private members are not accessible
        System.out.println("protectedVariable = " + protectedVar);
        System.out.println("publicVariable = " + publicVar);
    }
}
```

This is file DifferentPackage.java:

- The two classes defined in p2 cover the other two conditions that are affected by access control.
- The first class, Protection2, is a subclass of p1.Protection. This grants access to all of p1.Protection's variables except for n_pri (because it is private) and n, the variable declared with the default protection.
- Default only allows access from within the class or the package, not extra-package subclasses.

```java
package p2;

import p1.AccessModifier;

class CrossPackage {
    CrossPackage() {
        AccessModifier demo = new AccessModifier();
        System.out.println("constructor");
        // System.out.println("defaultVar = " + demo.defaultVar); // Not accessible
        // System.out.println("privateVar = " + demo.privateVar); // Not accessible
        // System.out.println("protectedVar = " + demo.protectedVar); // Not accessible
        System.out.println("publicVar = " + demo.publicVar);
    }
}
```

This is file CrossPackage.java

```java
package p1;

public class TestPackage {
    public static void main(String[] args) {
        p1.AccessModifier obj1 = new p1.AccessModifier();
       p1.DerivedClass obj2 = new p1.DerivedClass();
        p1.SomeClass obj3 = new p1.SomeClass();
    }
}
```

```java
package p2;

public class TestPackage2{
    public static void main(String[] args) {
        p2.DifferentPackage obj1 = new p2.DifferentPackage();
        p2.CrossPackage obj2 = new p2.CrossPackage();
    }
}
```

```java
package q1;
class ClassA {
    void display(){
        System.out.println("package q1 display method");
    }
}
```

```java
package q2;

import q1.ClassA;

 class ClassB {
    public static void main(String[] args) {
        ClassA obj=new ClassA();
        //obj.display();
    }
}
```

```java
package q1;
class ClassA {
    void display(){
        System.out.println("package q1 display method");
    }
}
```

```java
package q2;

import q1.ClassA;

 class ClassB {
    public static void main(String[] args) {
        ClassA obj=new ClassA();
        //obj.display();
    }
}
```

```
java: q1.ClassA is not public in q1; cannot be accessed from outside package
```

Importing Packages

- All of the standard classes are stored in some named package.
- Classes within packages must be fully qualified with their package name or names.
- Java includes the import statement to bring certain classes, or entire packages, into visibility.
- Once imported, a class can be referred to directly, using only its name.
- In a Java source file, import statements occur immediately following the package statement (if it exists) and before any class definitions.
- This is the general form of the import statement:
  - import pkg1 [.pkg2].(classname | *);
  - pkg1 is the name of a top-level package, and pkg2 is the name of a subordinate package inside the outer package separated by a dot (.).
  - Finally, specify either an explicit classname or a star (*), which indicates that the Java compiler should import the entire package.

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

```
class MyDate extends java.util.Date {
 }
```

Summary

- A class can have only one package declaration but it can have more than one package import statements.

```
package package4; //This should be one
import package1;
import package2;
import package3;
```

- A class inside a package while importing another package then the package declaration should be the first statement, followed by package import.

```
package abcpackage;
import xyzpackage.*;
```

- Use the fully qualified name method when both the packages have a class with the same name,

```
package1. Example obj = new package1.Example();
package2. Example obj2 = new package2. Example();
```

```
//This will throw compilation error
import package1.*;
import package2.*;
```

Summary

- To import all the classes present in package and subpackage, we need to use two import statements like this:

```
import abc.*;
import abc.foo.*;
```

- import abc.*; will only import classes Example1, Example2 and Example3 of abc package but it will not import the classes of sub package.

```
import abc.*;
```

- To import the classes of subpackage you need to import like this:,

```
import abc.foo.*;
```