SNS COLLEGE OF ENGINEERING

Kurumbapalayam (Po), Coimbatore – 641 107

An Autonomous Institution

Accredited by NAAC – UGC with 'A' Grade

Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

23CSB101

OBJECT ORIENTED PROGRAMMING

Access Specifiers and Static members

By

M.Kanchana

Assistant Professor/CSE

Access Specifiers

Access specifiers are used to specify the visibility and accessibility of a class constructors, member variables and methods.

Types:

1. Public
2. Private
3. Protected
4. Default (package)

Access Specifiers

Public (anything declared as public can be accessed from anywhere):

A variable or method declared/defined with the public modifier can be accessed anywhere in the program through its class objects, through its subclass objects and through the objects of classes of other packages also.

Private (anything declared as private can't be seen outside of the class):

The instance variable or instance methods declared/initialized as private can be accessed only by its class. Even its subclass is not able to access the private members.

Access Specifiers

Protected (anything declared as protected can be accessed by classes in the same package and subclasses in the other packages):

The protected access specifier makes the instance variables and instance methods visible to all the classes, subclasses of that package and subclasses of other packages.

Default (can be accessed only by the classes in the same package):

The default access modifier is friendly. This is similar to public modifier except only the classes belonging to a particular package knows the variables and methods.

Access Specifiers

	PRIVATE	DEFAULT	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

Packages and Access Specifiers

Creating Own Package

- Choose a package name (e.g., mypackage1).
- Add the **package keyword** at the top of the Java file.
- Save the file inside a folder with the same name as the package.

package **packagename;**

package **mypackage1;**
public class **FirstClass** {
    public String i = "I am public variable";
    protected String j = "I am protected variable";
    private String k = "I am private variable";
    String r = "I don't have any modifier";
}

Packages and Access Specifiers

Compilation Command:

javac -d . mypackage1\FirstClass.java

-d . → Saves the compiled .class file inside the package directory mypackage1.

D:\JAVA Programs\
├── mypackage1\
│ ├── FirstClass.class

Run:

java mypackage1.FirstClass

Packages and Access Specifiers

```
package mypackage2;
import mypackage1.FirstClass;

class SecondClass extends FirstClass
{
    void method() {
System.out.println(i);
System.out.println(j);
System.out.println(k);
System.out.println(r);
}
public static void main(String arg[]) {
      SecondClass obj = new SecondClass();
      obj.method();
}
```

Packages and Access Specifiers

·if SecondClass.java depends on FirstClass.java, so we need to set the classpath (cp)

javac -d . -cp . mypackage2\SecondClass.java

-cp . tells Java to look in the current directory (.) for other compiled classes

D:\JAVA Programs\
├── mypackage1\
│ ├── FirstClass.class
├── mypackage2\
│ ├── SecondClass.class

Run:

java mypackage2.SecondClass

Packages and Access Specifiers

·if SecondClass.java depends on FirstClass.java, so we need to set the classpath (cp)

javac -d . -cp . mypackage2\SecondClass.java

-cp . tells Java to look in the current directory (.) for other compiled classes

D:\JAVA Programs\
├── mypackage1\
│ ├── FirstClass.class
├── mypackage2\
│ ├── SecondClass.class

Run:

java mypackage2.SecondClass

Packages and Access Specifiers

.

mypackage2\SecondClass.java:7: error: k has private access in FirstClass
System.out.println(k);

mypackage2\SecondClass.java:8: error: r is not public in FirstClass; cannot be accessed from outside package
System.out.println(r);

MCQ

.

What is the error in the code above regarding access specifiers, and how can it be fixed?

```
public class Car {
    private String model;  // 'model' is private

    public Car(String model) {
        this.model = model;
    }

    public void displayModel() {
        System.out.println("Car model: " + model);
    }
}
```

```
public class TestCar {
    public static void main(String[] args)
{
        Car car = new Car("Toyota");

System.out.println("Model: " + car.model);
        car.displayModel();
    }
}
```

MCQ

.

```
public class Car {
    private String model;  // 'model' is private
    public Car(String model) {
        this.model = model;
    }
        public String getModel() {
        return model;
    }

    public void displayModel() {
        System.out.println("Car model: " +
model);
    }
}
```

```
public class TestCar {
    public static void main(String[]
args) {
        Car car = new Car("Toyota");
  System.out.println("Model: " +
car.getModel());
        car.displayModel();
    }
}
```

Static Members

- Static Members are variables and methods that belong to a static or non-static class rather than to the objects of the class.

- Hence it is not necessary to create object of that class to invoke static members

this refers to the current object
static methods don't belong to an object

The static can be:
1.variable (also known as class variable)
2.method (also known as class method)
3.block
4.nested class

Static Members

Static Variables (Class Variables):

When a member variable is declared with the static keyword, then it is called static variable and it can be accessed before any objects of its class are created, and without reference to any object.

Syntax to declare a static variable:

[access_specifier] static data_type instance_variable;

Static Members

- When a static variable is loaded in memory (static pool) it creates only a single copy of static variable and shared among all the objects of the class.
- A static variable can be accessed outside of its class directly by the class name anddoesn‘t need any object.

Syntax : <class-name>.<variable-name>

Static Members

```
class Student {
    static int totalStudents = 10;
   }
public class Main {
    public static void main(String[] args) {
          System.out.println("Total Students: " + Student.totalStudents);
    }
}
```

Output

10

Static Members

Static Method:

If a method is declared with the **static** keyword , then it is known as static method.

- A static method belongs to the class rather than the object of a class.
- A static method can be invoked without the need for creating an instance of a class.
- A static method can access static data member and can change the value of it.

Static Members

Syntax: (defining static method)

[access_specifier] static Return_type method_name(parameter_list)
{
// method body
}

Syntax to access static method:

<class-name>.<method-name>

The most common example of a static member is main( ). main( ) is declared as static because it must be called before any objects exist.

Static Members

```
class calsquare {
    static int square(int number) {
        return number * number;
    }
}

public class Main {
    public static void main(String[] args) {
        System.out.println(calsquare .square(5)); // Output: 25
    }
}
```

Static Members

Static Block:

- Static block is used to initialize the static data member like **constructors** helps to initialize instance members and it gets executed exactly once, when the class is first loaded.
- It is executed before main method at the time of class loading in JVM.

Syntax:

```
class classname
{
static
{
// block of statements
}
}
```

Static Members

```
class Example {
    static int count;

       static {
        count = 100;
        System.out.println("Static block executed!");
    }
}

public class Main {
    public static void main(String[] args) {
        System.out.println(Example.count);
    }
}
```

Output

```
Static block executed!

100
```

Static Members

```
class Student {
    int rollno;
    String name;
    static String college = "SNSCE";

    static void change() {
        college = "SNSCT";
    }

    Student(int r, String n) {
        rollno = r;
        name = n;
    }

      void display() {
       System.out.println(rollno + " " + name + " " + college);
    }
}
```

```
public class TestStaticMembers {
    static {
        System.out.println("*** STATIC MEMBERS – DEMO
***");
    }

    public static void main(String args[]) {
        Student.change();

        Student s1 = new Student(111, "Karan");
        Student s2 = new Student(222, "Aryan");
        Student s3 = new Student(333, "Sonoo");


        s1.display();
        s2.display();
        s3.display();
    }
}
```

Static Members

Output:

```
*** STATIC MEMBERS – DEMO ***
111 Karan SNSCT
222 Aryan SNSCT
333 Sonoo SNSCT
```

MCQ

1. What is the keyword used to make a method belong to the class rather than an instance?

A) Public
B) private
C) static
D) protected

MCQ

1. What is the keyword used to make a method belong to the class rather than an instance?

A) Public
B) private
C) static
D) protected

MCQ

1. Can a static method access non-static variables directly?

A) Yes
B) No

MCQ

1. Can a static method access non-static variables directly?

A) Yes
B) No

MCQ

3.What will happen if we declare the main method as private?

```
class Main {
    private static void main(String[] args) {
        System.out.println("Hello");
    }
}
```

A) Compilation error
B) Runtime error
C) Hello
D) No output

MCQ

3.What will happen if we declare the main method as private?

```
class Main {
    private static void main(String[] args) {
        System.out.println("Hello");
    }
}
```

A) Compilation error
B) Runtime error
C) Hello
D) No output

Assignment

1. Can we **override** static methods in Java?
2. Can we use **this** in a static method?
3. What happens if we declare a **constructor as private**?
4. Can a **static** method **be private**?

THANK YOU