Week 8

Java Assignment

Java Access Modifiers with Method Overriding with example

Method Overriding in java and Access Modifier in Java **Method Overriding**
In any object-oriented programming language, Overriding is a feature that allows a subclass or child class to provide a specific implementation of a method that is already provided by its super-class or parent class.

When a method in a subclass has the same name, same parameters or signature and same return type(or sub-type) as a method in its super-class, then the method in the subclass is said to override the method in the super-class.

Method overriding is one of the ways by which java achieve Run Time Polymorphism. The version of a method that is executed will be determined by the object that is used to invoke it.

If an object of a parent class is used to invoke the method, then the version in the parent class will be executed, but if an object of the subclass is used to invoke the method, then the version in the child class will be executed.

In other words, it is the type of the object being referred to (not the type of the reference variable) that determines which version of an overridden method will be executed.

Access Modifiers As the name suggests, access modifiers in Java help to restrict the scope of a class, constructor, variable, method or data member. There are four types of access modifiers available in java:

- Default – No keyword required
- Private
- Protected
- Public

Method Overriding with Access Modifiers There is Only one rule while doing Method overriding with Access modifiers i.e.
If you are overriding any method, overridden method (i.e. declared in subclass) must not be more restrictive.

Access modifier restrictions in decreasing order:

- private
- default
- protected
- public

i.e. private is more restricted then default and default is more restricted than protected and so on. **Example 1:**

- Java

```
class A {
    protected void method()
    {
        System.out.println("Hello");
    }
}

public class B extends A {

    // Compile Time Error
    void method()
    {
        System.out.println("Hello");
    }

    public static void main(String args[])
    {
        B b = new B();
        b.method();
    }
}
```

Output:

```
Compile Time Error
```

In the above Example Superclass **class A** defined a method whose access modifier is **protected**. While doing method overriding in SubClass **Class B**

we didn't define any access modifier so **Default** access modifier will be used. By the rule, **Default** is **more restricted** then **Protected** so this program will give

compile time error. Instead of default, we could've used **public** which is **less restricted** then **protected**. **Example 2:**

- Java

```
class A {
    protected void method()
    {
        System.out.println("Hello");
    }
}

public class B extends A {
    public void method()
    {
        System.out.println("Hello");
    }

    public static void main(String args[])
    {
        B b = new B();
        b.method();
    }
}
```

Output:

```
Hello
```

In the above Example Superclass **class A** defined a method whose access modifier is **protected**.

While doing method overriding in SubClass **Class B** we define access modifier as **Public**. Because **Public** access modifier is **less restricted** than **Protected** hence this program compiles successfully.

2.Write a program in class uses private access control.

Private Access Modifier

- The methods or data members that are declared as private are only accessible within the class in which they are declared.
- Top level classes or interface cannot be declared as private in light of the fact that
-
  - Private signifies "just visible inside the enclosing class".
  - Protected signifies "just noticeable inside the enclosing class and any subclasses".
-
- If a class has a private constructor then you cannot create the object of that class from outside the class.
- Classes cannot be marked with the private access modifier.
- Denoting a class with the private access modifier would imply that no different class could get to it. This generally implies that you cannot utilize the class by any stretch of the imagination. In this way, the private access modifier does not take into account classes.
-

Class or Interface cannot be declared as private.

Syntax:

```
public class Clock {
private long time = 0;
}
```

Example:1

```
package p;
class A {
private void display(){
System.out.println("Edureka");
```

```
}
}
class B {
public static void main(String args[]){
A obj = new A();
//trying to access private method of another class
obj.display();
}
}
```

The output of this program is:

```
error: display() has private access in A
obj.display();
```

The private access modifier is accessible only within the class.

another example of private access modifier

In this example, we have created two classes A and Simple. A class contains private data member and private method.

We are accessing these private members from outside the class, so there is a compile-time error.

```
class A{
private int data=40;
private void msg(){System.out.println("Hello java");}
}

public class Simple{
```

```
 public static void main(String args[]){
   A obj=new A();
   System.out.println(obj.data);//Compile Time Error
   obj.msg();//Compile Time Error
   }
}
```

Role of Private Constructor

We cannot create the instance of that class from outside the class. For example:

```
class A{
private A(){}//private constructor
void msg(){System.out.println("Hello java");}
}
public class Simple{
 public static void main(String args[]){
   A obj=new A();//Compile Time Error
 }
}
```