Access Specifier

Date: 16th Aug 2024

Access specifiers are used to represent scope of members of class (variables, methods, constructor).
In java Access specifiers are classified into 4 types

1. private
2. default
3. protected
4. public

1. **private**: If you declare any member of class as private then scope of that member remains only within the class
It cannot be access from other classes.

2. **default**: If you declare any member of class as default then scope of that member remains only within the package
It cannot be access from other packages.
There is no keyword to represent default access specifier.

3. protected: If you declare any member of class as protected then scope of that member remains only within the package that class which is present outside the package can access it by one condition ie. inheritance operation

4. public: If you declare any member of class as public then scope of that member remains thought the project.

Example1: private Access specifier

```
package AccessSpecifier;
public class Sample1
{
	private int num;  //private access specifier

	private Sample1()     //private access specifier
	{
		num=10;
	}

	private void squareOfNum()     //private access specifier
	{
		System.out.println(num*num);
	}
```

```
        public static void main(String[] args)
        {
                Sample1 s1=new Sample1();
                s1.squareOfNum();
                System.out.println(s1.num);
        }
}
```

```
package AccessSpecifier;
public class Sample2
{
        public static void main(String[] args)
        {
                Sample1 s1=new Sample1();
                s1.squareOfNum();
                System.out.println(s1.num);
        }
}
```

Example2: default Access specifier

```
package AccessSpecifier;
public class Test1
{
         int num;  //default access specifier

         Test1()    //default access specifier
        {
                num=10;
        }

  void squareOfNum()     //default access specifier
        {
                System.out.println(num*num);
        }


        public static void main(String[] args)
        {
                Test1 t1=new Test1();
                t1.squareOfNum();
                System.out.println(t1.num);
        }
}
```

```
package AccessSpecifier;
public class Test2
{
        public static void main(String[] args)
        {
                Test1 t1=new Test1();
                t1.squareOfNum();
                System.out.println(t1.num);
        }
}
```

```
package AccessSpecifier1;
import AccessSpecifier.Test1;
public class Test3
{
        public static void main(String[] args)
        {
                Test1 t1=new Test1();
        }
}
```

Example3: protected access specifiers

```
package AccessSpecifier;
public class Test5
{
        protected int a;

        protected Test5()
        {
                a=10;
        }

        protected void squareOfNum()
        {
                System.out.println(a*a);
        }

        public static void main(String[] args)
        {
                Test5 t5=new Test5();
                t5.squareOfNum();
                System.out.println(t5.a);
        }
}
```

```
package AccessSpecifier;
public class Test6
{
        public static void main(String[] args)
        {
                Test5 t5=new Test5();
                t5.squareOfNum();
                System.out.println(t5.a);
        }
}
```

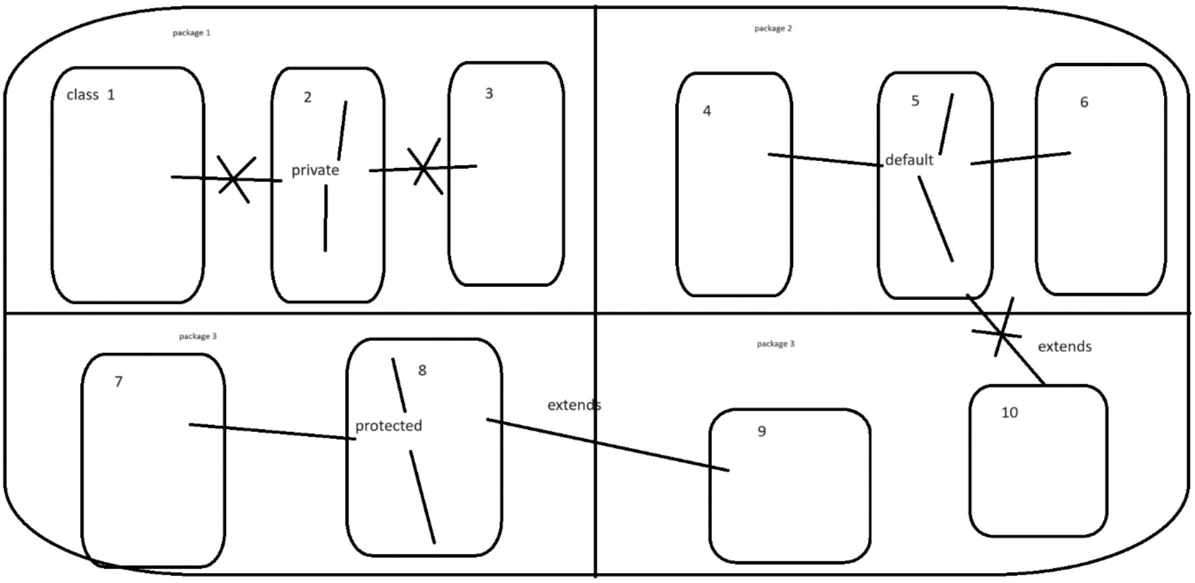
```
package AccessSpecifier1;
import AccessSpecifier.Test5;
public class Test7 extends Test5
{
        public static void main(String[] args)
        {
                Test7 t7=new Test7();
                t7.squareOfNum();
                System.out.println(t7.a);
        }
}
```

Example4: public access specifiers

```
package AccessSpecifier;
public class Test11
{
        public int a;

        public Test11()
        {
                a=20;
        }

        public void squareOfNum()
        {
                System.out.println(a*a);
        }

        public static void main(String[] args)
        {
                Test11 t11=new Test11();
                t11.squareOfNum();
                System.out.println(t11.a);
        }
}
```

```
package AccessSpecifier;
public class Test12
{
        public static void main(String[] args)
        {
                Test11 t12=new Test11();
                t12.squareOfNum();
                System.out.println(t12.a);
        }
}
```

```
package AccessSpecifier1;
import AccessSpecifier.Test11;
public class Test13
{
        public static void main(String[] args)
        {
                Test11 t13=new Test11();
                t13.squareOfNum();
                System.out.println(t13.a);
        }
}
```



1. What are Access modifiers in java?
2. Why are access modifiers used?
3. How many types of modifiers in Java?
4. Which access modifier is also known as Universal access modifier?
   -> Public
5. Which is the most restrictive access modifier in Java?
   -> Private
6. Which is the least restrictive access modifier in Java?
   -> Public
7. Can we have a private constructor in Java?
8. What is the role of private constructor in Java?
9. Which access modifiers can be used with a class?
   -> Public and Default.
7. Can we declare a class as private?
   -> No
9. Can we declare a class as protected?
   -> No

10. Types of modifiers in java?
    1: access modifiers
    2: non-access modifiers

11. What are non-access modifiers in Java?
    -> There are four non-access modifiers in Java. They are as follows:

    1: static
    2: final
    3: abstract
    4: synchronized

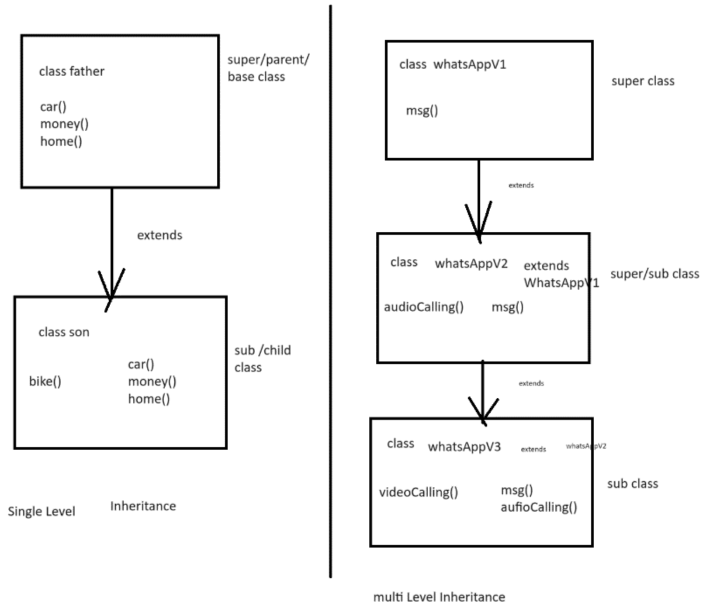
    1) Static: This modifier is used to check that a member is a class member or instance member.
    If you declare a class as static, this class will be executed first.

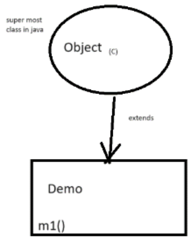
    2) Final: Final is a keyword that is used to restrict the users. In other words,
    it is used to restrict further modification of a class, field, or method. If a class is declared as ‘final’,
    the class cannot be subclassed.

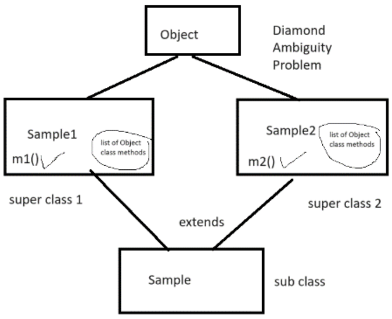
    3) Abstract: Abstract is a keyword that is used with a class or a method. An abstract class or abstract method is used for further modification.
    If a class is declared as ‘abstract’, the class cannot be instantiated.

    4) Synchronized: It is used to achieve thread safeness. Only one thread can enter in a synchronized method or block at a given time.

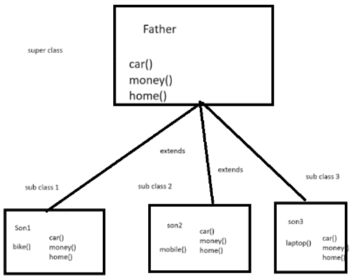
Images of Inheritance: -







3: multiple Inheritance



4: Hierarchical Inheritance