Access Specifiers

What are access specifiers?

The access specifiers in Java specifies the accessibility or scope of a field, method, constructor, or class.

There are four types of Java access modifiers:

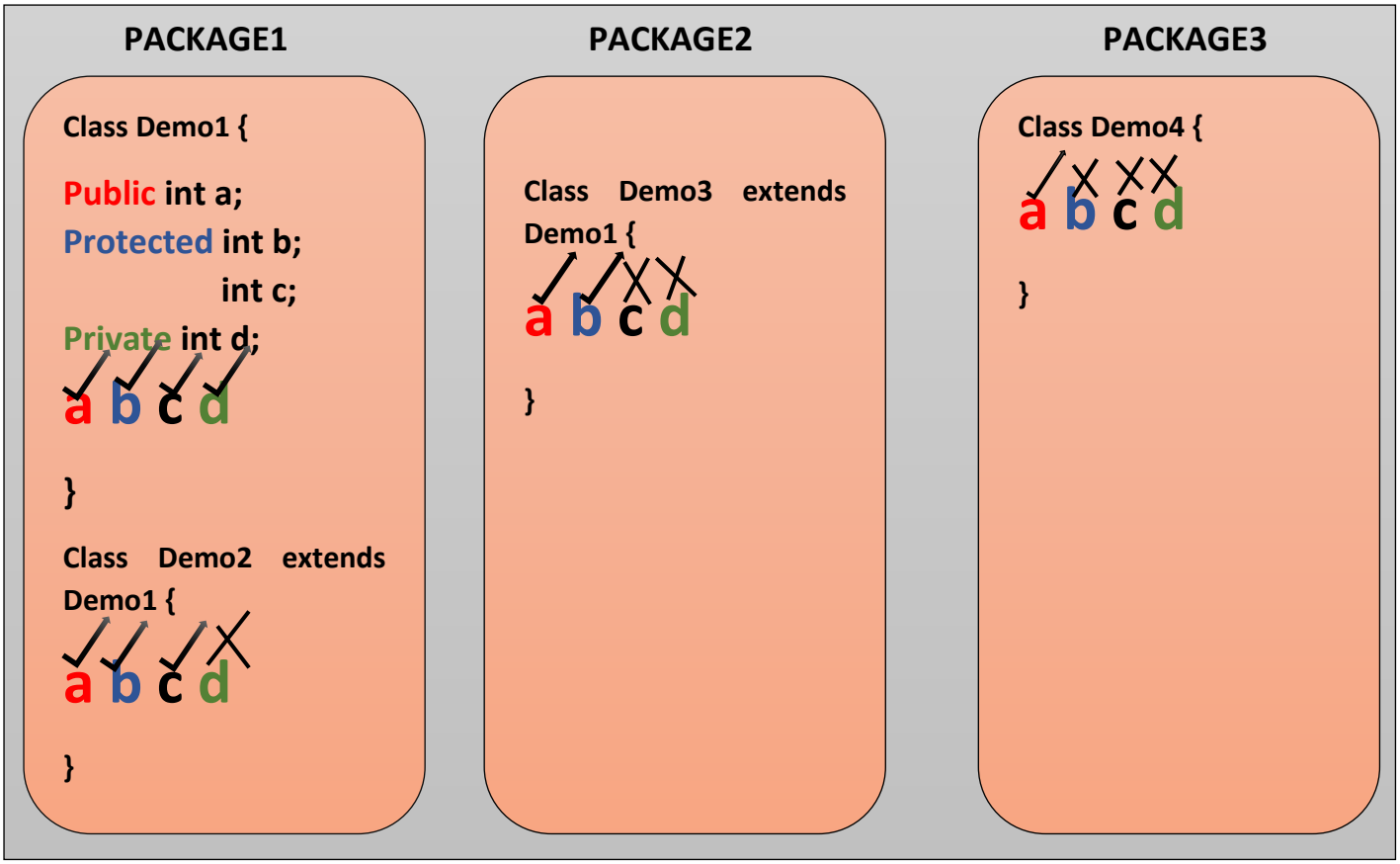
1. **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. **Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
3. **Protected**: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
4. **Public**: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

Didn't get it?? CONFUSED????

Let us try to understand these specifiers with the help of packages as shown below:

The 30th of November is known as "Computer Security Day".

JAVA PROJECT

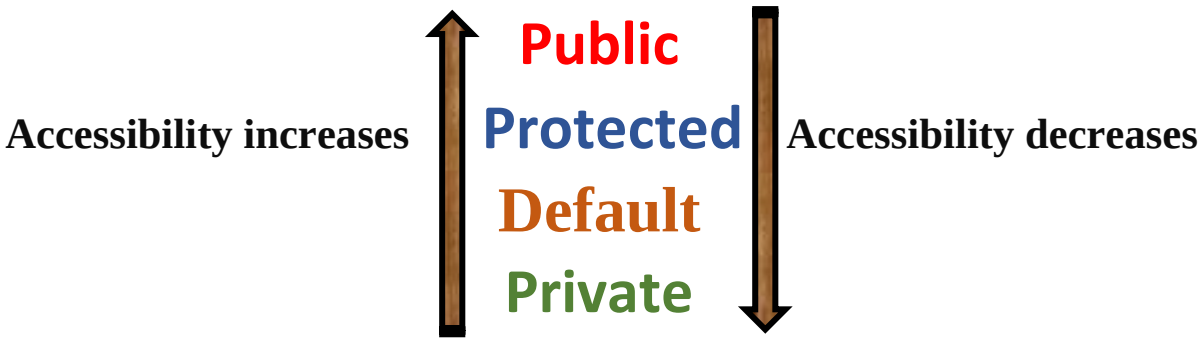


In the above java project, there are three packages. It consists of four variables:
Variable **a** whose access specifier is **Public**.
Variable **b** whose access specifier is **Protected**.
Variable **c** whose access specifier is **Default**.
Variable **d** whose access specifier is **Private.**

Let us understand the access of these variables by a simple table:

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

Accessibility:



Rules of method overriding

Rule1: The overridden method of child class must either maintain the same access modifier as the parent class method or provide greater access however, it can never reduce.

```
class Parent
{
    public void fun()
    {
        System.out.println("Parent class method");
    }
}
class Child extends Parent
{
         void fun() // access reduced to default
    {
        System.out.println("Overidden child class method");
    }
}
class  Demo
{
    public static void main(String[] args)
    {
        Child c = new Child();
        c.fun();
    }
}
```

Output:

```
Demo.java:10: error: fun() in Child cannot override fun() in Parent
                void fun()
                     ^
  attempting to assign weaker access privileges; was public
1 error
```

Rule2: **The overridden method of child class must maintain the same return types as the parent class method.**

```
class Parent
{
    public void fun()
    {
        System.out.println("Parent class method");
    }
}
class Child extends Parent
{
    public int fun() // return type changed to int
    {
        System.out.println("Overidden child class method");
        return 1;
    }
}
class  Demo
{
    public static void main(String[] args)
    {
        Child c = new Child();
        c.fun();
    }
}
```

Output:

```
Demo.java:10: error: fun() in Child cannot override fun() in Parent
    public int fun()
           ^
  return type int is not compatible with void
1 error
```

Rule3: **In the overridden method of child class the return types can be different provided between the return types, is-a relationship exists. Such return types between which is-a relationship exists is called as co-variant return types.**

```
class Plane
{

}
class CargoPlane extends Plane
{

}
class Parent
{
    public Plane fun()
    {
        System.out.println("Parent class method");
        Plane p = new Plane();
        return p;
    }
}
```

```
class Child extends Parent
{
    public CargoPlane fun()
    {
        System.out.println("Overidden child class method");
        CargoPlane cp = new CargoPlane();
        return cp;
    }
}
class  Demo
{
    public static void main(String[] args)
    {
        Child c = new Child();
        c.fun();
    }
}
```

Output: