Access Control

How a member can be ~~modified~~ accessed is determined by the access modifier attached to its declaration.
Usually, you will want to restrict access to the data members of a class - allowing access only through methods.
Also there will be times when you will want to define methods that are private to a class.

Java's access modifiers are public, private and protected.
Java also defines a default access level.
Protected applies only when inheritance is involved.

⇒ When no access modifier is used, then by default the member of a class is public within its own package, but cannot be accessed outside its package.

	Class	Package	Subclass (same pkg)	Subclass (diff pkg)	World (diff pkg & not subclass)
public	✓	✓	✓	✓	✓
protected	✓	✓	✓	✓	
no modifier	✓	✓	✓		
private	✓				

```
package packageOne;
public class Base {
    protected void display(){
        System.out.println("In Base");
    }
}
```

```
package packageTwo;
public class Derived extends packageOne.Base {
    public void show(){
        new Base().display(); // Not working
        new Derived().display(); // is working
        display(); //is working.
    }
}
```

protected ~~class~~ ~~allows~~ allows access from Subclass and from other classes in same packag.

=> We can use child class to use protected members outside the package but only child class object can access it.

That is why ~~do~~ Derived class instance can access protected method in Base.

The other line creates a Base instance (~~not~~ not a derived instance)!!
Any access to protected methods of that instance is allowed only from objects of the same packag.

=> display();
allowed, because the caller, an instance of Derived class has access to protected members and fields of its subclasses even if they are in different packages.

new Derived.display();
allowed, because you call the method on an instance of Derived and that instance has access to the protected methods of its subclass.

new Base.display();
not allowed because the "caller's" (the this reference) classes is not ~~allowed~~ defined in the same package like the Base ~~class~~ Class
So this can't access the protected method.
And it doesn't matter - as we see - that the current subclasses a class from that package
That backdoor is closed :)

Remember
any time talks about a subclass having a access to a superclass member, we could be talking about the subclass inheriting the member, not simply accessing the members through a reference to an instance of the superclass.

```
class C
    protected member;

// in a different package
class S extends C {
    obj.member; // only allowed if type of obj
                   is S or subclass of S
```

The motivation is probably as follows:
If obj is an S, class S has sufficient knowledge of its internals, it has the right to manipulate its ~~memory~~ members, and it can do this safely.

If obj is not an S, its probably another subclass S2 of C, which S has no idea of.

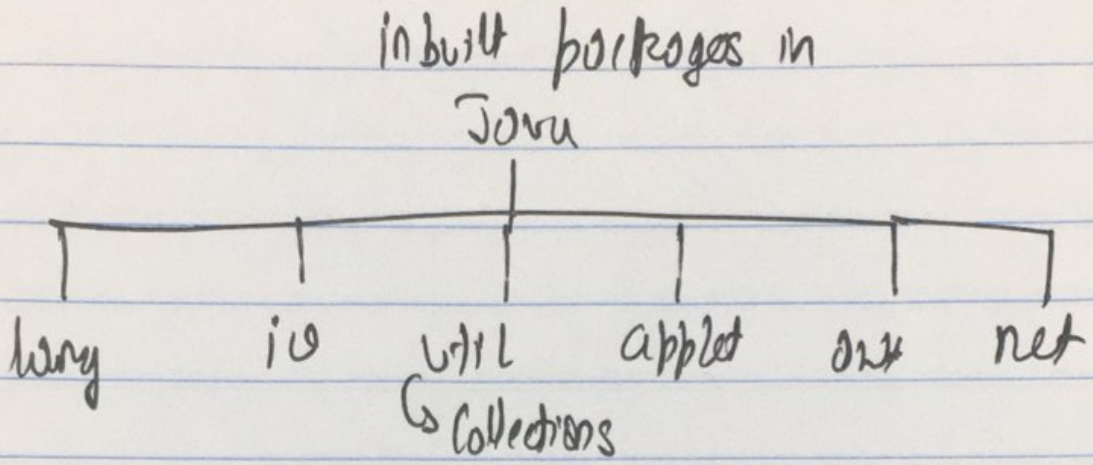
S2 may not even be born when S is written. For S to manipulate S2's protected internals is quite dangerous.

If it is allowed from S2's point of view it doesn't know who will tamper with its protected internals and how, this makes S2 Job very hard to reason about its own state.

Now if obj is O, O extends S
is it dangerous for S to access obj.member?
Not really.

How S uses member is of shared knowledge of S and all its subclasses, including S as the superclass has the right to define behaviors and O as the subclass has the obligation to accept and conform.

For easier understanding, the rule should really be simplified to require obj's (static) type to be exactly S.
After all, it's very unusual and inappropriate for subclass O to appear in S. And even if it happens, that the static type of obj is O own simplified rule can deal with it easily by upcasting: ((S)obj).member.


inbuilt packages in Java
lang
io
util
Collections
applet
awt
net