Chapter - 7

Inheritance:

⇒ Subclasses inherit the members (instance variables and methods) of the superclass.
⇒ Subclass extends the Superclass.

If the JVM doesn't find a version of the method in the Subclass, it starts walking back up the inheritance hierarchy until it finds a match.

What can we do, if we don't want to completely replace the superclass version and just want to add more stuff?

⇒

```
public void roam() {
        super.roam();      // This calls the inherited version of roam() then
        //my roam stuff    // comes back to do subclass specific code
}
```

Public members are inherited

Private members are not inherited

Dos and Don'ts in Inheritance:

1. **Do** use inheritance when one class is a **more specific** type of the superclass.
2. **Do** consider Inheritance when you have behaviour that should be **shared amongst multiple classes** of the same general type.
3. **Do not** use inheritance just so that you can **reuse code** from another class.
4. **Do not** use inheritance if the subclass and superclass do not pass the **IS-A test**.

Using IS-A and HAS-A

IS-A:

If class B extends class A
⇒ class B **IS A** class A.

This is true anywhere in the Inheritance Tree.
If class C extends class B
⇒ class C passes the IS-A test for both B and A.

Note: The Inheritance IS-A relationship works only in 1 direction

HAS-A:

For eg: Bathroom **HAS A** tub

Bathroom has a reference to a Tub, but bathroom doesn't extend Tub or vice-versa.
So Bathroom can have a Tub **Instance Variable.**

Polymorphism:

The reference type can be a superclass of the actual object type.

```
Animal myDog() = new Dog();
```

Anything that extends the declared Reference variable type can be assigned to the Reference variable.

For eg: Polymorphic Arrays

```
Animal[] animals = new Animal[3];  // Declaring an array of Superclass type
animals[0] = new Dog();            //You can put objects of subclass.
```

```
animals[1] = new Cat();
animals[2] = new Lion();
```

You can have polymorphic arguments and return types as well.

```
class Vet {
        public void giveShot(Animal a){
                // do anything to the Animal at the other end of the parameter a
                a.makeNoise();
        }
}

class PetOwner {
        public void start() {
                Vet v = new Vet();
                Dog d = new Dog();   // This will work because Dog and Hippo
                Hippo h = new Hippo();   // are subclasses of class Animal.

                v.giveShot(d);       // Dog's makeNoise() runs
                v.giveShot(h);       // Hippo's makeNoise() runs
        }
}
```

Overriding:

⇒ Subclass redefines one of its inherited methods when it needs to change or extend the behaviour of the method.

Instance Variables are not overridden because they don't need to be. They don't define any special behaviour, so a subclass can give an inherited instance variable any value it chooses.

Can you extend any class? Or is it like if the class is private, you can't?

⇒ There is no such thing as a private class, except the very special case called **inner class.**

A class cannot be private, but it can be non public(when you don't mark it public). And it can be subclassed only by classes in the same package.

Another thing that can stop a class from being subclassed is the keyword **modifier final**. A final class means it's the end of Inheritance.

And the third reason is if a class has private constructors.

Why make a class final?

⇒ To ensure security. For eg: String class is final.

Can a method be final, without making the whole class final?

⇒Yes, if you want to protect a specific method from being overridden.

Rules for Overriding:

1. The arguments and the return types of your overriding methods must look exactly like the overridden methods in the superclass.
2. You cannot make it less accessible, like make it private in the subclass.

The methods are the contract

Note:
Compile time: Compiler looks if you can call that method on that reference variable.
Run time: JVM looks at the object.

So, if the compiler has already approved it, it can work only if the overriding method has the same type and arguments.

Overloading a method:

It is just having 2 methods, with the same name but different argument list.
It has nothing to do with Inheritance and Polymorphism.

> The return types can be different, provided the argument list is different.
> If you change only the return type, the compiler will assume that you are trying to override. It will not be legal, until the return type is the subtype of the return type declared in the superclass.