Polymorphism

PREVIEW / RECAP:

We have seen how Inheritance works. Now let's understand the concept of Polymorphism.

What is a Polymorphism?

Polymorphism means more than one form. That is, the same entity (method or operator or object) behaves differently in different scenarios. Polymorphism allows us to create consistent code.
Example: a superclass called Animal that has a method called animalSound(). Subclasses of Animals could be Pigs, Cats, Dogs, Birds - And they also have their own implementation of an animal sound (the pig oinks, and the cat meows, etc.):

Is it compulsory to create a Constructor in java program? Like the Main Method?

NO. It's ok if we don't create constructor in java programs. As java creates a **Default Constructor** for each class. So every class has at least 1 constructor which is **Default Constructor.**

Rules of Creating Constructor

1. Constructor name must be the same as its class name
2. A Constructor must have no explicit return type
3. A Java constructor cannot be abstract, static, final, and synchronized

Types of Polymorphism

In Java, Polymorphism can be divided into two types:

- Run-time Polymorphism
- Compile-time Polymorphism

Run-time Polymorphism

In Java, run-time polymorphism can be achieved through method overriding. Suppose the same method is created in the superclass and its subclasses. In this case, the method that will be called depends upon the object used to call the method.

E.g.

```java
abstract class Animal {
   public abstract void makeSound();
}

class Dog extends Animal {
   @Override
   public void makeSound() {
      System.out.println("Bark bark..");
   }
}

class Cat extends Animal {
   @Override
   public void makeSound() {
      System.out.println("Meow meow..");
   }
}

class Main {
   public static void main(String[] args) {
     Dog  d1 = new Dog();
      d1.makeSound();

      Cat c1 = new Cat();
      c1.makeSound();
   }
}
```

Output

Bark bark...

Meow-meow...

Compile-time Polymorphism

The compile-time polymorphism can be achieved through method overloading and operator overloading in Java.