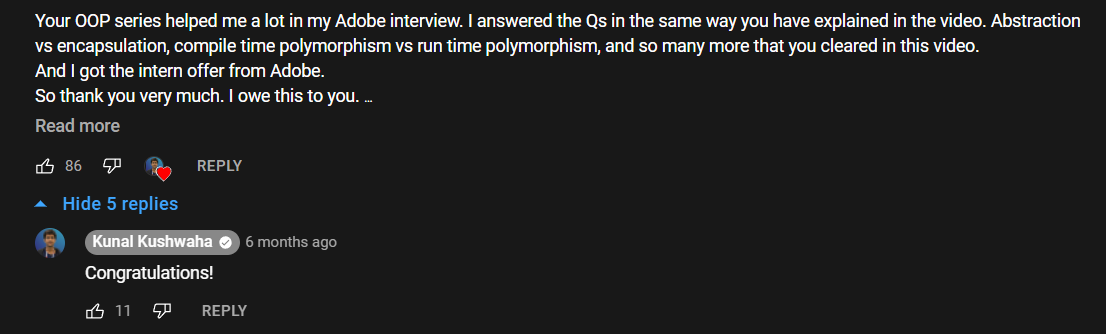
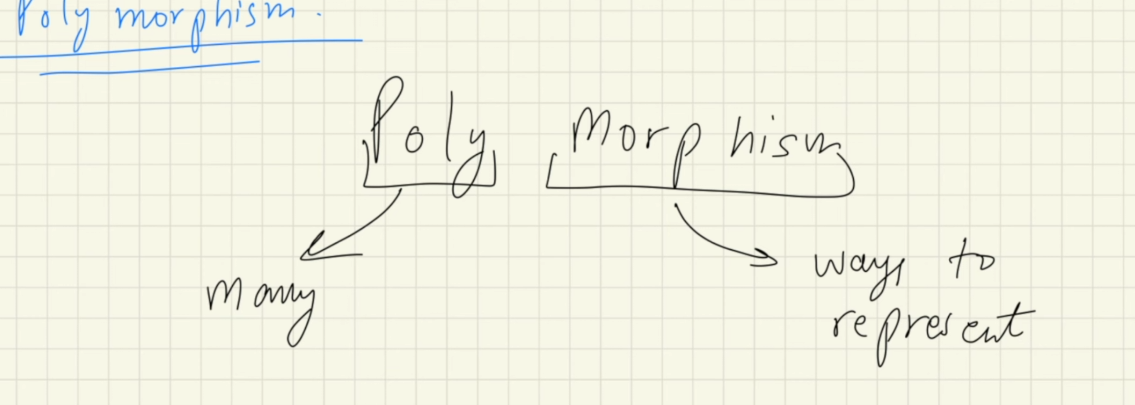
****

**Polymorphism**



Polymorphism means many ways to represent single entity, like representing the same method in different ways

**Types of polymorphism**

**Static polymorphism / Compile time polymorphism**

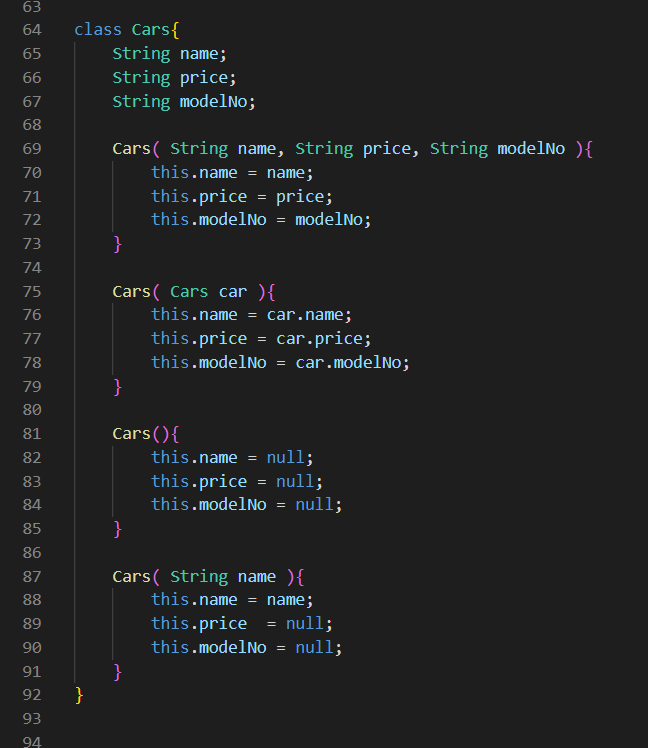
Achieved via method overloading

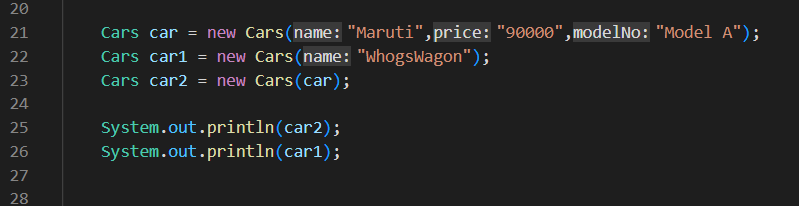
**Method overloading**

Same name but different data type of parameters or different numbers of parameters or different return type

Example : Multiple constructors

It is called compile time polymorphism, because Java itself decides which method to call during the compile time





Here in Car class we have different types of constructors

Basically a method with same name but with different number and different data types of parameters

This is also an example of method overloading

**Also here same method is represented in different ways, so polymorphism is achieved**

While making objects java is determining itself which constructor to call during the compile time

So called compile time polymorphism

**Do not be confused here the actual memory allocation for the objects happen at the run time, but which constructor is to be called is determined at the run time**

Here java does type checking of the arguments passed, and the data type of the parameters of the constructors, and it determines which constructor would be called later on during the run time

**Runtime polymorphism / Dynamic polymorphism**

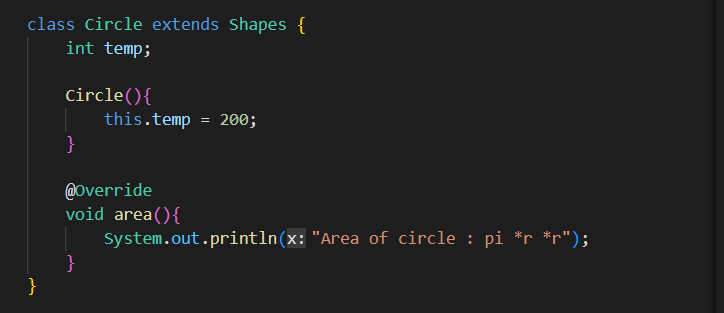
This is achieved by method overriding

Method overriding occurs when there is same method in the child class and parent class, and the method of the parent class is overridden by the method in the child class

**DO NOT BE CONFUSED**

Method overloading occurs when there is same function name but with different parameters number or parameters data type or return type in the SAME CLASS

Here JVM determine which method to call during the run time not the compile time so called run time through some thing called **dynamic object dispatch** polymorphism, this makes sense because the object themselves are not made at compile time but at run time, so the method overriding also occur at the run time

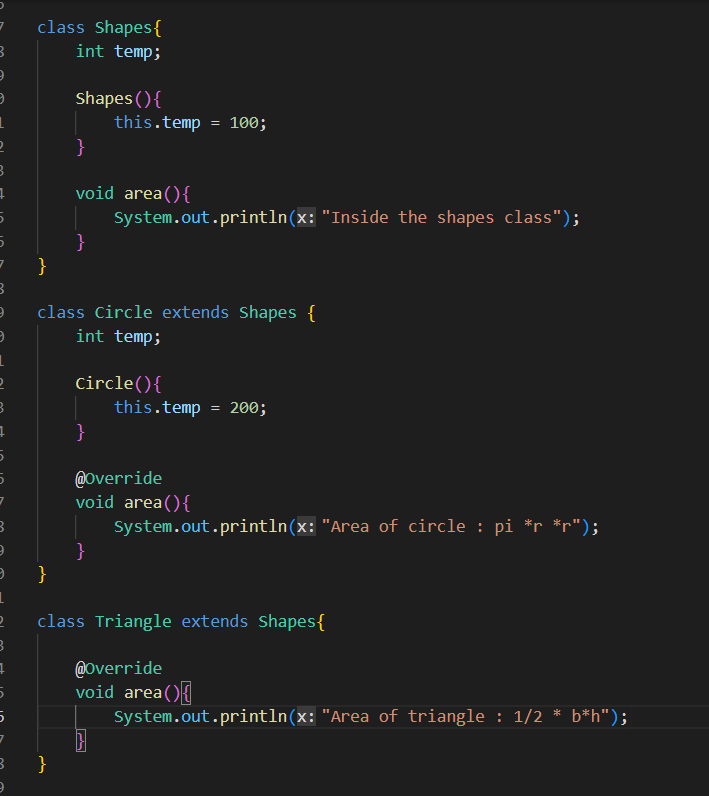


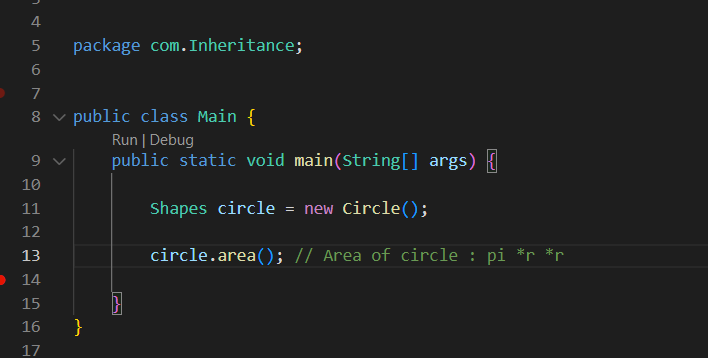
@override

This is called annotation

Used to check whether the method is being overridden or not

It gives the error if the method is not being overridden





Here the data type of the reference variable circle is Shapes class

So the area () method of the Shapes class should be accessed

But here the area( ) of Circle class is accessed

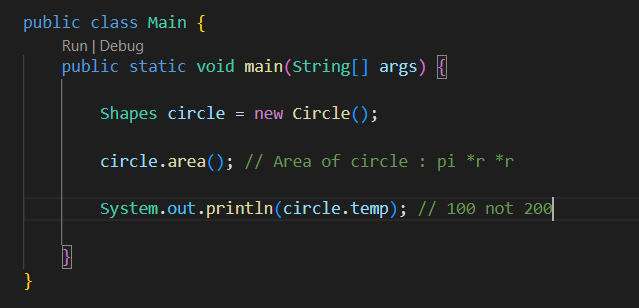
How ?

It’s an absolute rule that only the methods and properties of the data type of the reference variable are accessed by the reference variable

But if the method is overridden in the child class then the method of the child class will be called

Here the area( ) method of the parent class Shapes is being overridden by the area( ) method of the child class Circle

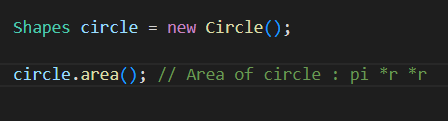
Also this run time polymorphism does not occur for data members or data variables



Here on doing circle.temp we got 100 not 200

Hence the run time polymorphism does not occur for the data members or data variables

So as per the rule the data member of the data type of the reference variable is accessed



*Reference evariable circle is of Shapes type, and object pointed by it is child class*

*As per rules suru maa Shapes class maa area( ) method lai herxaa, and also child class maa pani herxaa. Yedi child class maa pani same method xaa vannae chai, child class ko method lae parent class ko method lai override garxaa and child class ko method call hunxaa*

Real life example of polymorphism

Let say a parent have an skill of making stone scriptures, and the son inherits that function from parent. So after long time people will by default look at the son’s skill because the son might have added new functionality, developed new techniques and so on. It’s natural in real world to go after the latest thing at first