**=>Inheritance**

Java allow multilevel inheritance but not multiple inheritance.

Parent class can be extended by more than one child class, but child class can only extend one parent class.

If we want to extend more than one parent class then you have to use interface not inheritance.

**=>Upcasting and Down casting**

=> When you make object of sub class and create a reference variable of parent class then you can only access the properties and method of parent class not child class.

NOTE: which properties and methods from which class you access depends on the reference variable type not the object, until unless they are not private.

=> You can have as many constructors as you want with different parameters passing’s, it is also called method overloading.

=> Child class can access the properties and methods of parent class and also its own, but the parent class cannot access the P&M of child class

**=>super keyword**

It mostly refers to the directly above class constructors whether with parameters or not.

In java every class is the child class of the object class by default.

You can access parent class property using \*super.name\* like this.

Here super is keyword and name is the property name you want to access.

If we do not call parent class constructor using super keyword then it will automatically call the default parent class constructor (the constructor with no parameter)

**=>Polymorphism**

Act of representing the same thing in different ways is called \*polymorphism\*

***Types of polymorphism***

1. Compile time / static polymorphism

This polymorphism capability can be achieved via method overloading and also operator overloading but Java does not support operator overloading so it cannot be achieved using operator overloading.

Now what is method overloading?

in method overloading a method has same name but different types, return types and order of the parameter types different given in the methods.

1. Run time/Dynamic polymorphism

***Method overriding?***

In method overriding the parent and the child has the methods with the same name types and everything except the body. If we make the child class object and call the method then it will first give the priority to the method of the object class in this case it is child. If you write @override on the method and it does not give error it means it is overriding the parent class method if giving error it means there is no method to override.

**E.g:**

**Scenerio:**

Shapes circle=new Circle();

circle.area();

Here the Shapes is the parent class and the Circle is the child class both are having the method with the same name area. Now what will print circle.area ?

Explanation:

What it will be able to access is defined by the reference variable type and which one will be able to access will be defined by the type of the object and how it will work?

It will only work if the reference variable is of type parent and the object is of child class then the what you will be able to access depends on the object not reference variable but that method should also be present in the parent class. This whole concept is called overriding and accessing the method in this way is called upcasting as explained above.

We know all of this theory but how does java know all of this?

Using Dynamic method dispatch

It determines at the run time which method to call.

**=>Final keyword**

It is used to create a constant.

It can also be used to prevent overriding and inheritance.

You can not override a method which is final.

Overridden is also known as late binding because it determines which method to call at the run time.

Early binding: since using the final keyword resolve the issue to determine which method need to call. Final make it clear that there is only one method to call so it happen at compile time, that is why it is called early binding.

Note:

If you write a final with a class then implicitly all the methods of that class will also become final.

**=>Static keyword**

Using static keyword with the final keyword

If the class is final and then it has a method which is static, so all the methods will be set to final but not this static method because it is object independent, it does not depend on the class object. So, it will also cannot be overridden.

The thing we discuss earlier {

**Scenerio:**

Shapes circle=new Circle();

circle.area();

Here the Shapes is the parent class and the Circle is the child class both are having the method with the same name area. Now what will print circle.area ?

Explanation:

What it will be able to access is defined by the reference variable type and which one will be able to access will be defined by the type of the object and how it will works ?

It will only works if the reference variable is of type parent and the object is of child class then the what you will be able to access depends on the object not reference variable but that method should also be present in the parent class. This whole concept is called overring and accessing the method in this way is called upcasting as explained above.

}

Now if we try to access the area method it will call the method of the parent not child because the method is static so it can not be overridden that is why the parent method will be called.

Note:

Static method can be inherited but cannot be overridden.

Overriding depends on objects, static does not depend on object that’s why the static methods can not be overridden.

In Polymorphism both overriding and overloading does not apply to instance variables.

Instance variables //properties/variables of the class

**=>Encapsulation**

Wrapping up the implementation of the data members and the methods in a class.

**=>Abstraction**

Hiding unnecessary information and showing only valuable information.

For example:

When we start the car, we use the key to rotate a little, we just need to know where to put the key and how much we have to rotate. We do not need to know the internal working of how the engine and the all the staff working.

ArrayList list=new ArrayList();

We use these classes but we do not care how they are working at the background that is why it is called abstract classes. However we can access the methods and properties of abstract classes.

Encapsulation vs Abstraction

* Abstraction solving a design level issue and encapsulation is solving a implementation level issue.
* Abstraction works on external issues and encapsulation works on internal issues.
* Abstraction is the process of gaining information and encapsulation is the process of containing information.

Data hiding

* It focuses on hiding the data just for the security purpose.

Data encapsulation

* It focuses on hiding the complexity of the system.
* Encapsulation is the sub-process of the data hiding.

**Summary**

* final prevents inheritance and method overriding; it resolves method calls at compile time.
* static methods are class-level methods that cannot be overridden and do not depend on object instances.
* Method overriding and upcasting allow child class methods to be called through parent references, except for static methods, which always refer to the parent class method.
* Polymorphism involves method overriding and overloading, but it does not apply to instance variables.