OBJECT ORIENTED PROGRAMING CONCEPTS in JAVA

OOP

- An object-based application in Java is based on
  1. Declaring classes, (like template)
  2. Creating objects from them and
  3. Interacting between these objects.

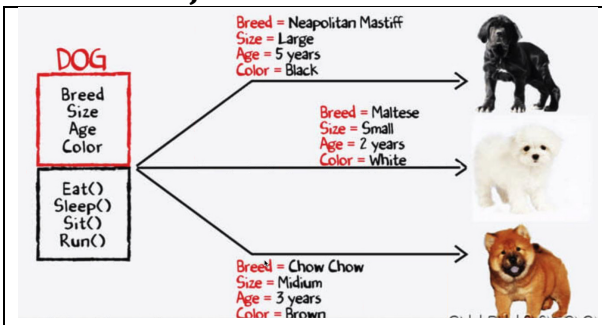
CLASS

- A class is a blueprint or prototype (it is an idea, definition, description) that defines the variables and the methods (functions) common to all objects of a certain kind.

```java
public class Employee {
  public String empName; //Employee name
  public double hourly_rate, hours_worked;
  public double getWeeklyHours() {
   return (hourly_rate * hours_worked);
  }
} //end of class
```

Attributes → **PROPERTIES**	Behaviors → **METHODS**
name	walk
weight	run
height	jump
gender	speak
age	sleep
Person has a name, but **it doesn't say what the name is.** Because the class is **just a description** of something	

CLASS & OBJECT



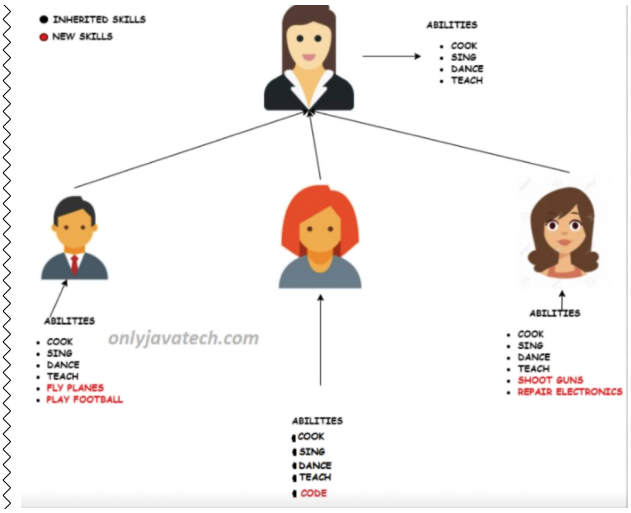

```java
class Dog{

    //variables
    String bread;
    String size;
    int age;
    String color;

    //method
    public String getInfo(){
        return("Breed is: " + breed + "Size is: " + size);
    }
}

public class Execute{
    public static void main(String[] arg){
        Dog maltese = new Dog();
        maltese.breed = "Maltese";
        maltese.size = "Small";
        maltese.age = 2;
        maltese.color = "white";
        System.out.println(maltese.getInfo());
    }
}
```

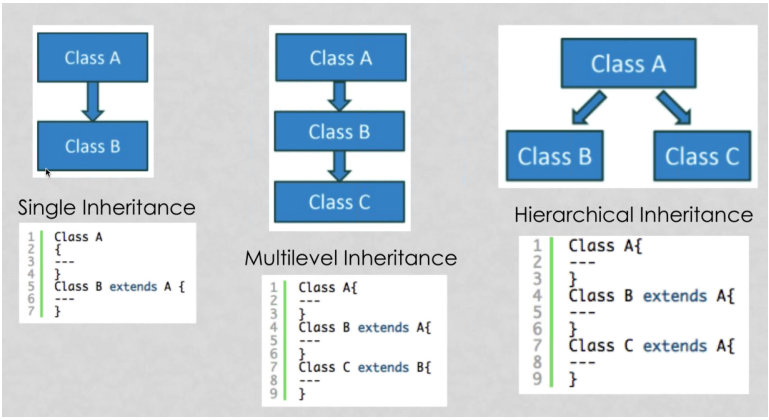
OOP CONCEPTS IN JAVA

1- Inheritance (JS)

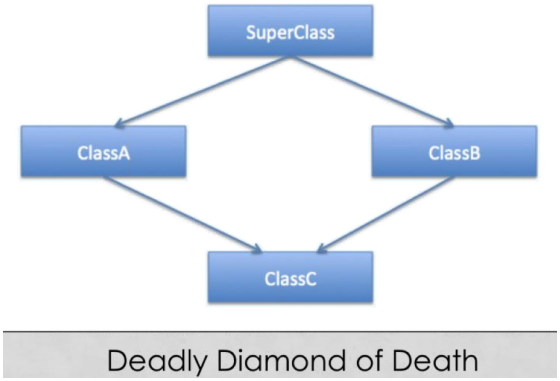
- Computer programs are designed in such a way where everything is an object that interacts with one another. (*It is code reusability*)
- Inheritance is one such concept where the properties of one class can be inherited by the other. It helps to reuse the code and establish a relationship between different classes.

- Parent class (Super or Base Class)
- Child Class (Subclass)





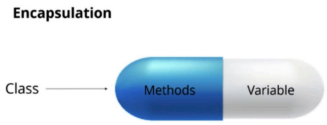
Class B has everything class A that have.



Multiple inheritance is **NOT** supported in Java

2- Encapsulation (JS)

```
public class Person{

    int age;

}

public class Test{

    public static void main(String[] args){

        Person p1 = new Person();
        p1.age = -20;
    }
}
```



It is a mechanism where you bind your data and code together as a single unit. It also means to hide your data in order to make it safe from any modification.

We can achieve encapsulation in Java by;
- Declaring the variables of a class as private.
- Providing public setter and getter methods to modify and view the variables values.

```
public class Person{

    private int age;

    public int getAge(){

        return age;
    }

    public void setAge(int age){

        if(age<18 || age>60){

            //error

        }else{

            this.age = age;

        }
    }
}
public class Test{

    public static void main(String[] args){

        Person p1 = new Person();
        p1.setAge(-20);

    }
}
```

void – your method doesn't return anything

3- Abstraction

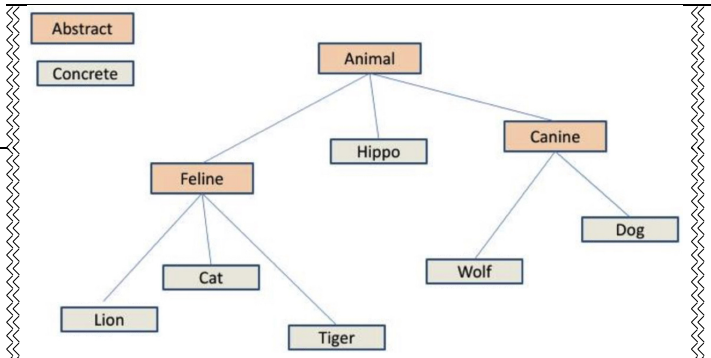
Which is unknown

- It deals with hiding the details and showing the essentials things to the user.
- You can achieve the abstraction in two ways:

I. ABSTRACT CLASSES

CLASSES

-we **can not** create new Animal
-we **can** create new Dog

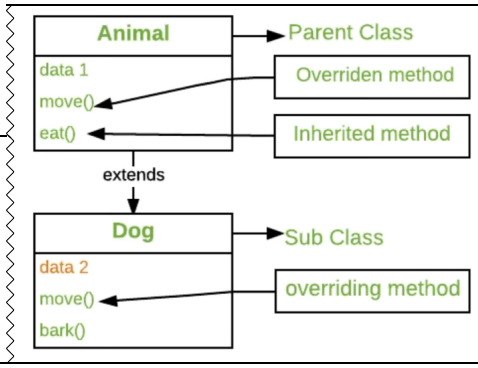



```
abstract class Canine extends Animal {

    //Code
}
```

- **Abstract classes** can not be instantiated
- An abstract class has virtually no use, no value, no purpose in life, unless it is **extended**
- With an abstract class, the guys doing the work at runtime are **instances of a subclass** of your abstract class.

ABSTRACT METHODS

-An abstract class means the class must be **extended**;
-An abstract class method means the method must be **overridden.**




```
public abstract void eat();
```

An abstract method has **NO** body!
-You already decided there is not any code that would make sense in the abstract, you won't put in a method body. So no curly braces, just end the declaration with a semicolon.
-If you declare an abstract method, you **MUST** mark the class abstract as well. You can not have an abstract method in a non-abstract class.

Summary;

- An abstract class is the one whose instances can not be created.
- Any class that has at least one abstract method has to be compulsory declared as an abstract class.
- Abstract class can contain both abstract and non-abstract methods
- An abstract method has no body
- Child class needs to override the all abstract methods.

II. INTERFACE

Interface in Java is a blueprint of a class or you can say it is a collection of abstract methods. In an interface, each method is public and abstract.

Interface also helps to achieve multiple inheritances in Java.

Example: We create Iphone and developer put "cook" or "dance" option on it and if it is not compulsory to create an iphone, developer put them implementing part and they are interface.

```
//To DEFINE an interface:
    public interface Pet{...}

//To IMPLEMENT an interface
    public class Dog extends Canine implements Pet{...}
```

Dog has access to Canine, Pet is interface

```
//All interface methods are abstract,so they MUST end in semicolon.
//They have no BODY
public interface Pet{

    public abstract void beFriendly();
    public abstract void Play();

}


public class Dog extends Canine implements Pet{

    public void beFriendly(){...} //You said you are a Pet, you must implement the Pet methods.It is your contract.

    public void play(){...} //You said you are a Pet, you must implement the Pet methods.It is your contract.

    public void roam(){...} //These are just normal overriding methods

    public void eat(){...} //These are just normal overriding
}
```

4- Polymorphism

Polymorphism means taking many forms. It is the ability of a variable, function, or object to take on multiple forms. Polymorphism in Java is 2 types:

I. Run time polymorphism (Dynamic Binding)

Method overriding is an example of run time polymorphism.

Rules for Java Method Overriding

- ✓ Method must have **same name** as in the parent class
- ✓ Method must have **same parameter** as in the parent class
- ✓ There must be IS-A relationship (**inheritance**)

```
class over{

    public void travel(){

        System.out.println("This method specifies the different forms of travel");
    }
}

public class override extends over{

    //Method is overriden here
    public void travel(){
        System.out.println("This method specifies car travel");
    }

    public static void main(String[] args){
        override r=new override();
        r.travel();
    }
}
```

II. Compile time polymorphism (Static Binding)

Method overloading is an example of compile time polymorphism. Method overloading is a feature that allows a class to have two or more methods having the same name but the arguments passed to the methods are different. Unlike method overriding, arguments can differ in:

1. Number of parameters passed to a method
2. Data type of parameters
3. Sequence of data types when passed to a method

```
class Adder{

    static int add(int a , int b){
        return a+b;
    }

    static double add(double a, double b){
        return a+b;
    }
}


public static void main(String args[]){

    System.out.println(Adder.add(11,11));
    System.out.println(Adder.add(12.3,12.6));
}
```

	Overloading	**Overriding**
Definition	Methods having same name but each must have different number of parameters or parameters having different types & order	Sub class have method with same name and exactly the same number and type of parameters and same return type as super class method
Meaning	More than one method shares the same name in the class but having different signature	Method of base class is re- defined in the derived class having same signature
Behavior	To add/extend more to method's behavior	To change existing behavior of method
Polymorphism	Compile Time	Run Time
Inheritance	Not Required	Always Required
Method Signature	Must have different signature	Must have same signature