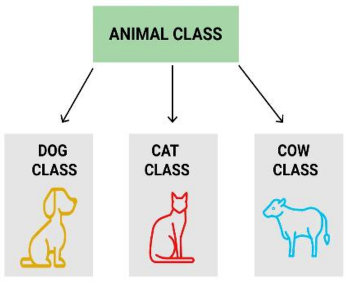
Inheritance in Object-Oriented Programming:

- *Inheritance* is a mechanism where ***a class*** (child or derived class) ***inherits attributes*** and ***methods*** from ***another class*** (parent or base class).
- *Inheritance* allows ***child classes*** to ***reuse*** the ***code*** (attributes and methods) defined in their ***parent class***, ***reducing code duplication***.
- By ***inheriting*** from a ***parent class***, ***child classes*** can ***focus*** on their ***specific functionalities***, making the code more ***modular*** and ***easier to maintain***.
- *Inheritance* creates ***a hierarchy*** of ***classes***, where a ***parent class*** can have ***multiple child classes***, each inheriting from it.
- *Inheritance* often represents an "***is-a***" relationship between classes (e.g., a car ***is a*** vehicle).
- The class being ***inherited from*** is called the ***parent*** or ***base class***, while the class ***inheriting*** is called the ***child*** or ***derived class***.



Sample code:

```java
class Shape{
    protected void area(){
        System.out.println("Display Area..");
    }
}
```

```java
class Triangle extends Shape {
    public void area(int l, int h) {
        System.out.println((float) (l * h) / 2);
    }
}
```

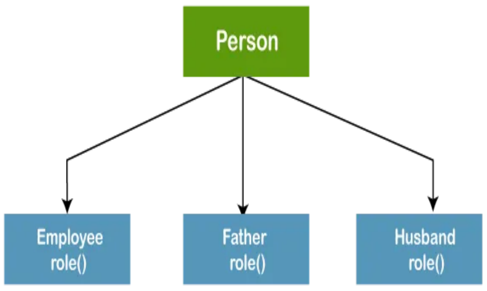
```java
public class Single_Level{
    public static void main(String[] args){
        Triangle sh = new Triangle();
        sh.area();// parent Class area method
        sh.area(5, 3); // own area method
    }
}
```

Output:

```
Display Area..
7.5
```

Polymorphism in Object-Oriented Programming:

- *Polymorphism* is one of the core concepts in OOP that ***allows objects*** to ***behave differently*** based on their ***specific class type***.
- The word ***polymorphism*** means having ***many forms***, and it comes from the ***Greek*** words ***poly*** (***many***) and ***morph*** (***forms***), this means ***one entity*** can take ***many forms***.
- *Polymorphism* allows the ***same method*** or ***object*** to ***behave differently*** based on the ***context***, especially on the ***project's actual runtime class***.
- it's achieved through ***two*** main types:
    1. *Compile-time polymorphism* (static) $\longrightarrow$ *Method overloading*
    2. *Runtime polymorphism* (dynamic) $\longrightarrow$ *Method overriding*



Runtime polymorphism (dynamic):

- *Dynamic polymorphism*, also known as *late binding polymorphism*, is a concept ***where the method to be executed*** is ***determined*** at ***runtime***, not at ***compile time***.
- This is achieved through mechanisms like ***method overriding***, allowing ***subclasses*** to provide their ***own implementations*** for ***methods*** defined in the parent class.

Sample code:

```java
interface Car{
    void accelerate();
}
```

```java
class Manual implements Car{
    @Override
    public void accelerate(){
        System.out.println("More acceleration due to patrol");
    }
}
```

```java
class Electric implements Car{
    @Override
    public void accelerate(){
        System.out.println("Less acceleration due to battery");
    }
}
```

```java
public class Dynamic_polymorphism{
    public static void main(String[] args){
        Car wagonR = new Manual();
        Car tesla = new Electric();

        wagonR.accelerate();
        tesla.accelerate();
    }
}
```

Output:

```
More acceleration due to patrol
Less acceleration due to battery
```

Compile-time polymorphism (static):

- *Static polymorphism*, also known ***early binding where the method to be executed*** is ***determined*** at ***compile time***.
- This is achieved through ***method overloading***, where a class can have ***multiple methods*** with the ***same name*** but ***different parameters***.
- The ***compiler*** resolves ***which method to call*** based on the ***arguments*** provided ***during compilation***.

Sample code:

```java
class Students {
  String name;
  int age;

  void printInfo(String name){
     System.out.println("name:  "+ name);
  }

  void printInfo(int age){
     System.out.println("age:  "+ age);
  }

  void printInfo(String name, int age){
     System.out.println("name:  "+name+"\nage:  "+ age);
  }
}
```

```java
public class Static_polymorphism {
    // Polymorphism -> many forms
    public static void main(String[]args){
        Students s1 = new Students();

        s1.name = "Siba";
        s1.age = 20;

        s1.printInfo(s1.age);
        s1.printInfo(s1.name);
        s1.printInfo(s1.name, s1.age);
    }
}
```

Output:

```
age : 20
name: Siba
name: Siba
age : 20
```

@Sibasundarj8