Object Oriented Programming

Why OOP?

- Allows breakdown large programs to small chunks.
- OOP Systems can easily upgradable.
- Less maintenance cost.
    - *Break the program into small parts **(OBJECTS)** and call in the main file (`method`). Because of this method can easily upgrade and easily remove existing parts as well.*

Disadvantages of OOP

1. Need to have brilliant designing skill and programming skill.
2. The length of the programmes developed using OOP language is much larger than the procedural approach.

Standards

```
//Class Name
Person //--first letter capital

//Variable
person //--first letter simple

//Method
person()  //--first letter simple with last brackets
```

Dot Operator

Always means the “**inside**” meaning from dot operator

```
System.out.println();
```

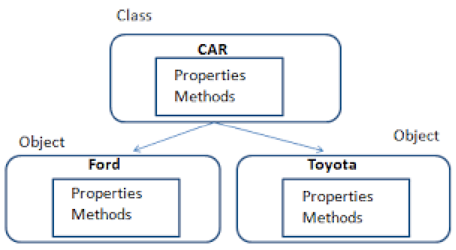
Meaning of above : `System` class —> `out` variable —> `println()` method

```
System.exit();  //exit method inside System class
Person.name;    //name variable value inside System class
```

Concepts

▼ Classes and Objects

Class	Object
Template for creating objects (**Blueprint**)	Object is instance of a class
Logical Entity	Physical Entity
Ex: Car	*Ex: BMW, Audi*
Class generate objects	Object provide life to class
Can't manipulated	Can manipulated



```java
public class Car {
    String color;
    int seats;
    int doors;

    void drive() {
        // do something
    }

    void reverse() {
        // do something
    }

    void park() {
        // do something
    }
}
```

```java
public class Main {
    public static void main(String[] args) {

        //========== Car 01 ==========
        Car car01 = new Car();

        car01.color = "Black";
        car01.seats = 5;
        car01.doors = 4;

        car01.drive();
        car01.reverse();
        car01.park();
        //========== Car 01 ==========


        //========== Car 02 ==========
        Car car02 = new Car();

        car02.color = "Blue";
        car02.seats = 4;
        car02.doors = 2;

        car02.drive();
        car02.reverse();
        car02.park();
        //========== Car 01 ==========
    }
}
```

▼ Constructor

Constructor is a special method.

Properties of Constructor

- Implement in same name as class.
- No return type.
- Can call only, when the object is implementing.

Why Constructor?

If there is something to do when the program is started, can do those things inside the constructor method.

```
public class Run {
    // Constructor
    Run() {
        // Connect to internet
        // Connect to database
        // Create views
    }
}

public class Main {
    public static void main(String[] args) {
        // Call from Run Class
        Run controller = new Run();
    }
}
```

```
public class Run {
    // Constructor
    Run(int doc) {
        // Connect to internet
        // Connect to database
        // Create views
    }
}

public class Main {
    public static void main(String[] args) {
        // Call from Run class
        Run controller = new Run(30);
    }
}
```

Let’s see the standard way of using constructor.

```
public class Car {
    private String color;
    private int seats;
    private int doors;

    // Constructor
    Car(String color, int seats, int doors) {
        this.color = color;
        this.seats = seats;
        this.doors = doors;
    }

    void drive() {
        // do something
    }

    void reverse() {
        // do something
    }

    void park() {
        // do something
    }
}
```

```
public class Main {
    public static void main(String[] args) {

        //========== Car 01 ==========
        Car car01 = new Car("Black",5,4);

        car01.drive();
        car01.reverse();
        car01.park();
        //========== Car 01 ==========


        //========== Car 02 ==========
        Car car02 = new Car("Blue",4,2);

        car02.drive();
        car02.reverse();
        car02.park();
        //========== Car 01 ==========
    }
}
```

▼ Method Overloading

Compile doesn't allow to create two methods with same name and same parameters (***Inside same class***). To overcome this, can create multiple methods with different parameters.

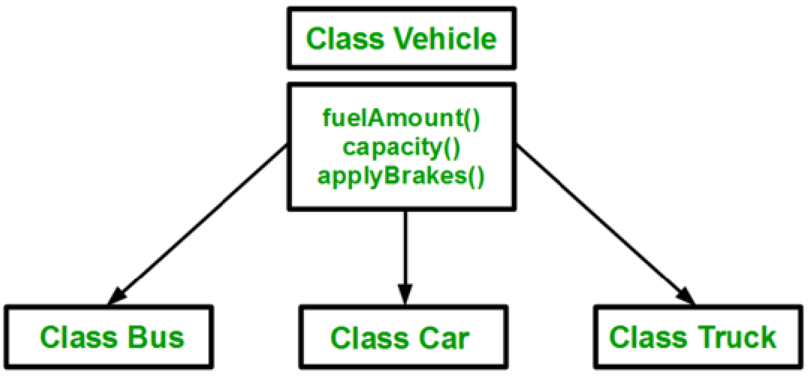
```
    public static void summation() {
        // method 01
        // do something
    }
    public static void summation(int x) {
        // method 02
        // do something
    }

    public static void main(String[] args) {
        summation();      // call to method 01
        summation(12);    // call to method 02
    }
```

▼ Inheritance

Inheritance: පරම්පරාවෙන් ලැබෙන උරුමය

Giving the methods of one class to another class and doing something improved from it.



Here, `Vehicle` **class is super class** and `Bus` **,** `Car` **,** `Truck` **classes are sub classes**.

Super Class	Sub Classes
`Vehicle`	`Bus`
	`Car`
	`Truck`

Sub classes also have all methods in `Vehicle` super class.

Also can add new methods and attributes to sub classes as well.

```
class Car {
    String ABSBreakSystem;

    gpsSystem(){}
    reverseCamera(){}
    autoDriving(){}
}
```

Standard Inheritance in Coding

```
// SUPER CLASS : Having generic elements
public class Phone {
    private String macAddress;  //--------> This private varible is not visible for extended sub class (SmartPhone)
    String brand;
    String name;

    void call() {
        // do something
    }

    void sms() {
        // do something
    }
}

// SUB CLASS : Having specific elements
public class SmartPhone extends Phone{
    String wifi;

    void internet() {
        // do something
    }

    void camera() {
        // do something
    }
}
```

▼ Access Modifiers

Modifier	Class	Package	Sub Class	Global
Public	Yes	Yes	Yes	No
Protected	Yes	Yes	Yes	No
Default	Yes	Yes	No	No
Private	Yes	No	No	No

▼ Method Overriding

```
public class Cat {
    public void sound() {
        System.out.println ("Meow");
    }
}

public class Lion extends Cat {
    public void sound() {
        System.out.println ("Roar");
    }
}
```

Same `sound()` method in `Cat` class is override in `Lion` class

▼ Super Keyword

2 Works can be done using `super` keyword

- Call the constructor of super class
- Access methods and variable in super class inside sub class

```
// SUPER CLASS
public class Cat {
    String hair;

    // Default Constructor
    Cat() {}

    public void sound() {
        System.out.println ("Meow");
    }
}

// SUB CLASS
public class Lion extends Cat {
    // Call to super class variable
    String hair = super.hair;

    // Default Constructor
    Lion() {}

    public void sound() {
        // Call to super class method
        super.sound();
        System.out.println ("Roar");
    }
}
```

▼ Upcasting & Downcasting

Upcasting

Process of applying subclass variable to superclass variable ***(Bigger ← Smaller)***

```
int x = 5;
long y = 55;

x = y; // Error: Because long is bigger type than int
y = x; // possible
```

```
class A{}
class B extends A{}

class Main{
  B objectB = new B();
  A objectA = new A();

  objectB = ObjectA; // Error: B class is smaller than A, because A is super class of B
  objectA = objectB; // possible
}
```

Downcasting

Applied subclass variable to superclass convert again to subclass.

```
class A{}
class B extends A{}

class Main{
  B objectB = new B();
  A objectA = new A();

  objectA = objectB; // possible
  objectB = ObjectA; // Error: B class is smaller than A, because A is super class of B (Compiler Error)

  // To avoid above error:
  objectB = (B) ObjectA; // possible
}
```

▼ Polymorphism

Polymorphism means "**many forms**", and it occurs when we have many classes that are related to each other by inheritance.

```
class Animal {
  public void animalSound() {
    System.out.println("The animal makes a sound");
  }
}

class Pig extends Animal {
  public void animalSound() {
    System.out.println("The pig says: wee wee");
  }
}

class Dog extends Animal {
  public void animalSound() {
    System.out.println("The dog says: bow wow");
  }
}
```

```
class Main {
  public static void main(String[] args) {
    Animal myAnimal = new Animal();  // Create a Animal object
    Animal myPig = new Pig();  // Create a Pig object
    Animal myDog = new Dog();  // Create a Dog object

    myAnimal.animalSound(); // The animal makes a sound
    myPig.animalSound();    // The pig says: wee wee
    myDog.animalSound();    // The dog says: bow wow
  }
}
```

Another Example

```
class ClassA{
  void print(){
    Syste.out.println("A");
  }
}

class ClassB{
  void print(){
    Syste.out.println("B");
  }
}

class Main{
  public static void main (String[] args){
    ClassB b = new ClassB();
    b.print(); //print B

    ClassA a = new ClassA();
    a.print(); //print A

    //upcasting
    ClassA a1 = new ClassB();
    a1.print(); //print B
  }
}
```

▼ Abstraction

Abstract Method → Method without having body

Need **Abstract Class** to save abstract method

Can’t create object from abstract classes

No constructor for abstract class: Because can’t create objects in abstract class

```
abstract class Vehicle{
  // Abstract method
  abstract void park();

  // Non-Abstract method
  void print(){
    System.out.println("Hello");
  }
}
```

Example

```
abstract class Vehicle{
  // Abstract method
  abstract void park();
}
```

```
class Car extends Vehicle{

}

class Main{
  Vehicle v1 = new Vehicle(); // Error: Can't
  Vehicle v2 = new Car();     // Possible
}
```

Example 02:

```
// Abstract class
abstract class Animal {
  // Abstract method (does not have a body)
  public abstract void animalSound();

  // Regular method
  public void sleep() {
    System.out.println("Zzz");
  }
}

// Subclass (inherit from Animal)
class Pig extends Animal {
  public void animalSound() {
    // The body of animalSound() is provided here
    System.out.println("The pig says: wee wee");
  }
}

class Main {
  public static void main(String[] args) {
    Pig myPig = new Pig(); // Create a Pig object
    myPig.animalSound();
    myPig.sleep();
  }
}

================================================================
OUTPUT:
The pig says: wee wee
Zzz
```

▼ Encapsulation

Protect the variables in the class. Using `Getters & Setters`

```
class Student {
    // Private variables are not allowed outside the class
    private int id;
    private String name;

    // ID Getter
    public void getId(int id){
        // this.id = id inside student class
        this.id = id;
    }

    // ID Setter
    public void setID(){
        // return the id inside student class
        return id;
    }

    // Name Getter
    public void getName(int name){
        // this.name = name inside student class
        this.name = name;
    }
    // Name Setter
    public void setID(){
        // return the name inside student class
        return name;
    }
}

class Main{
    public static void main(String args[]){
        Student s1 = new Student();
        s1.id = 1;
        s1.name = "Root"
    }
}
```

▼ Interfaces

In java, ***multiple inheritance is not working***. The solution for that is interfaces.

```
class Animal{}

interface AnimalUI{}

interface AnimalMedUI{}

// Sub class
class ZooAnimal extends Animal implements AnimalUI, AnimalMedUI{}
```

Specificatios of Interfaces

- Can’t create objects
- Can’t create constructors
- Can use sub class methods
- No Normal methods, ***public abstract methods only***
- **Variable** should be ***public static final*** only.