OBJECT-ORIENTED

Brief Introduction to OO Features

- https://www.tutorialspoint.com

ENCAPSULATION

Encapsulation

- A mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit.

- Data hiding - variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class.

- To achieve encapsulation in Java
  - Declare the variables of a class as private.
  - Provide public setter and getter methods to modify and view the variables values.

Encapsulation

```
public class GeometricObject {
        private String color;
        private boolean filled;


        public String getColor() {
                return color;
        }
        public void setColor(String color) {
                this.color = color;
        }
        public boolean isFilled() {
                return filled;
        }
        public void setFilled(boolean filled) {
                this.filled = filled;
        }
}
```

Encapsulation

- Benefit
  - The fields of a class can be made read-only or write-only.
  - A class can have total control over what is stored in its fields.
  - The users of a class do not know how the class stores its data. A class can change the data type of a field and users of the class do not need to change any of their code.

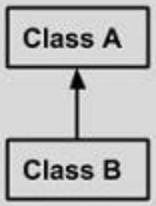
INHERITANCE

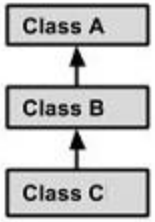
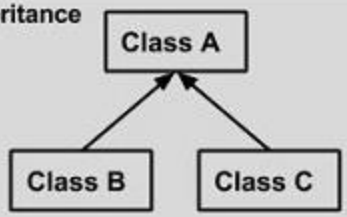
Inheritance

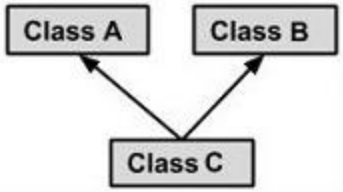
- The process where one class acquires the properties (methods and fields) of another.
- The class whose properties are inherited is known as superclass (base class, parent class).
- The class which inherits the properties of other is known as subclass (derived class, child class).
- `extends` is the keyword used to inherit the properties of a class.

Inheritance

- Types of inheritance

Single Inheritance 	public class A {
	.....
	}
	public class B **extends** A {
	.....
	}

Multi Level Inheritance 	public class A { .....} public class B **extends** A {.....} public class C **extends** B {..... }
Hierarchical Inheritance 	public class A { .....} public class B **extends** A {.....} public class C **extends** A {..... }

Multiple Inheritance 	public class A { .....}
	public class B {.....}
	public class C **extends** A,B {
	.....
	} // Java does not support mutiple Inheritance

Inheritance

```java
public class GeometricObject {
  private String color;
  private boolean filled;
  public GeometricObject() {
        this.color = "blue";
  }
  public GeometricObject(String color, boolean filled) {
        this.color = color;
        this.filled = filled;
  }


  public String getColor() {
        return color;
  }
  public void setColor(String color) {
        this.color = color;
  }
  public boolean isFilled() {
        return filled;
  }
  public void setFilled(boolean filled) {
        this.filled = filled;
  }
  public void printInfo(){
        if(filled){
              System.out.println("The Geometric is
        "+color+" and it is "+"filled.");
        }
        else{
              System.out.println("The Geometric is
        "+color+" but it is not "+"filled.");
        }
  }
}
```

```java
public class Circle extends GeometricObject {
  private double radius;
  private final double PI = Math.PI;

  public Circle() {
        this(1.0);
  }

  public Circle(double radius) {
        this(radius, "white", true);
  }

  public Circle(double radius, String color, boolean filled) {
        super(color, filled);
        this.radius = radius;
  }

  public double getRadius() {
        return radius;
  }
  public void setRadius(double radius) {
        this.radius = radius;
  }
  public double getArea() {
        return radius*radius*Math.PI;
  }
  public double getPerimeter() {
        return 2*radius*Math.PI;
  }
  public void printInfo() {
        super.printInfo();
        System.out.println( "It is a circle with radius of " +
radius);
  }
}
```

POLYMORPHISM

Polymorphism

- The ability of an object to take on many forms.
- The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.
- It is a feature that allows one interface to be used for a general class of actions.

Polymorphism

```
public class PolymorphismDemo {
  public static void main(String[] args) {
    m(new GraduateStudent());
    m(new Student());
    m(new Person());
    m(new Object());
  }

  public static void m(Object x) {
    System.out.println(x.toString());
  }
}

class GraduateStudent extends Student {
}

class Student extends Person {
  public String toString() {
    return "Student";
  }
}

class Person extends Object {
  public String toString() {
    return "Person";
  }
}
```

- An object of a subtype can be used wherever its supertype value is required. This feature is known as *polymorphism*.

ABSTRACTION

Abstraction

- A process of hiding the implementation details from the user, only the functionality will be provided to the user.
- The user will have the information on what the object does instead of how it does it.
- In Java, abstraction is achieved using Abstract classes and interfaces.

Abstraction – Abstract Class

```
public abstract class GeometricObject {
  private String color;
  private boolean filled;

  public GeometricObject() {
            this.color = "blue";
  }
  public GeometricObject(String color, boolean filled) {
            this.color = color;
            this.filled = filled;
  }

  public String getColor() {
            return color;
  }
  public void setColor(String color) {
            this.color = color;
  }
  public boolean isFilled() {
            return filled;
  }
  public void setFilled(boolean filled) {
            this.filled = filled;
  }

  public abstract double getArea();

  public abstract double getPerimeter();
}
```

```
public class Circle extends GeometricObject {
  private double radius;
  private double PI = Math.PI;

  public Circle() {
            this(1.0);
  }
  public Circle(double radius) {
            this(radius, "white", true);
  }
  public Circle(double radius, String color, boolean
filled) {
            super(color, filled);
            this.radius = radius;
  }

  public double getRadius() {
            return radius;
  }
  public void setRadius(double radius) {
            this.radius = radius;
  }
  public double getArea() {
            return radius*radius*PI;

  }
  public double getPerimeter() {
            return 2*radius*PI;
  }
}
```

Abstraction – Interface

```
public interface GeometricObjectInterface {
        public static final String COLOR = "white";
        public abstract double getArea();
        public abstract double getPerimeter();
}
```

```
public class Circle2 implements GeometricObjectInterface {
  private double radius;
  private final double PI = Math.PI;
  public Circle2() {
        this(1.0);
  }
  public Circle2(double radius) {
        this.radius = radius;
  }

  public void setRadius(double radius){
        this.radius = radius;
  }
  public double getRadius() {
        return radius;
  }
  public double getArea(){
        return radius*radius*Math.PI;
  }
  public double getPerimeter(){
        return 2*radius*Math.PI;
  }
}
```