Object Oriented Programming

There are four core concept in OOP.
Encapsulation, Inheritance, Polymorphism, Abstraction.

Inheritance:

some elements extands from another class.

```
~~Pub~~
public class person{
   String name; }
   int age;
Void displayInformation1(){
{System.out.Println("Name :"+name);
(System.out.Println("Age : "+ age);

}
}
```

```
public class teacher extends pe
{
String qualification;

 Void displayInformation2(){
 displayInformation1();
System.out.println("Qualific

}
}
```

$$\begin{tikzcd}
\text{Parent class/ Super class/ Base class} & \boxed{\text{Person}} \arrow[l] & \\
& \boxed{\text{Teacher}} \arrow[u] \arrow[r] & \text{child class/ Sub class/ derived class.}
\end{tikzcd}$$

What is inheritance?

⇒ It can be define as the process where one class acquires the properties of another.

Why we need inheritance?

⇒ i) For code Reusability.
ii) For method overriding.
iii) To implement parent-child relationship.

Practical:

*Write a class called Human with variables name & nationality. Create another class called Player & extends it from human. Add the variable position to Player & add a method to print all the variables in Player.

⇒
```
class Human{
 String name;
 String nationality:
  public Human(String name, String nationality){
       this.name = name ;
       this. nationality = nationality;
  }
  public void displayInfo(){
   System.out.println("Name : "+name);
   System.out.println("Nationality :"+nationality);
  }
}
class Player extends Human{
  String position;
   public Player(String name, String nationality String
                                          Position){
    Super(name, nationality);
    this. position = position;
}
```

```
    public void displayInfo(){
      Super.displayInfo();
     System.out.Println("Position :"+ position);
  }
}

public class Main{
    public static void ~~Main~~main(String [] args){
    Player P = new Player("Jehan", "Bangladeshi", "Forward");
     P.displayInfo();
  }
}
```

OOP :

16/10/23

Polymorphism
- method overriding
- method overloading

```
public class Animal{
    void eat(){
        SOP("Eating");
    }
}
class Dog extends Animal{
    Void eat (int x){
        SOP("~~Eating~~ Dog ate" + x " + " times");
    }
}
```

```
public class shape{
    void calculateArea
```

Output.

```
Calculating
```

```
public class shape{
    void calculateArea(){
        SOP("calculating...");
    }
}
class rectangle extends Shape{
```

Kazi Tamim
Kazi-bhai

[illegible]

15.10.23

Eco -
15.10.23

6

☐ Create a football class with method "play". create two sub-classes called "forward" & "goalkepper. override the play method & print "Forward is a attacking" & "Goalkepper is saving" sequentially.

⇒
```
public class football{
        void play(){

        }
    }

class forward extends football{
        void play(){
        SOP("forward is attacking.");
        }
    }
```

GDP =

```
class goalkeeper extends football{
        void play(){
        Sop("Goalkeeper is saving.");

    }
  }
  public class test{
    public static void main(String[] args){
      football f = new football();
      f.play();
      goalkeeper g = new goalkeeper();
      g.play();
  }
}
```

OOP

18/10/23

Encapculation:

→ Default → public → private → protected

↓ Access modifier

```
public class student{
    private int age;
    private string name;

    public void setAge(int age)
    { this.age = age;
    }

    public int getAge(){
        return age;
    }
}

public class test{
  public static void main(String[] args){
     student s = new Student();

     S.setAge(25);
     SOP("Age is:" +s.getAge());
  }
}
```

```
public class Student{
   private int age;
  public void setAge(int age){
     if (age ≥ 18){
        this.age = age;
     }
     else {   SOP("Not accept");
     }
  public in getAge(){
     return age;
   }
}

public class test{
   public static void main(String[] args){
   Student s = new student();
     S.setAge(7);
     SOP("Age is :" + s.getAge());
  }
}
```

Thread:

Thread allows to operate more efficiently by doing multiple things at a same time.

- It works in the background without interrupting the main program.

Example:

```
public class Main extends Thread{
    //code
}
```

```
import java.util.* ;
public test {
        public static void main (String [] args){
        Scanner sc = new Scanner (System.in);
        try {
                int a = sc. nextInt ();
                int d = sc. next Int ();
                int result = a/d ;
              SOP ("Division result:" + result);
            }
            catch ~~{~~ (Exception e) {
              SOP (" Error found" + e.getMessage())
              }
```

OOP 30.10.23

```
public test{
        public static void main (String[] args) {
            try {
                  ~~catch (Exception e){~~
                    expected error zone
            }
                 Scanner sc = new Scanner (System.in)
                 int x = sc.nextInt ();
                }  ~~Exce~~ catch (Exception e) {
                  S.O.P("Error found: + e.getMessage());
                  SOP (" Enter integer value");
                }
              }
```

Exception is used so that if there is any error in any line then it won't effect the other lines and the code will run.

```
public test{
    public static void main (String [] args){
      Scanner sc = new Scanner(System.in) ;
      try{
            int a  = sc. nextInt ( );
            if(a < 0){
             throw ~~a~~ new Exception("Input is ~~valid~~ positive no.");
            }
        } catch (Exception e){
            ~~SOP("Error" + e.getMessage ()~~) ;
                a = sc. next Int();
                 SOP(a);
        }

        try{
              int b = sc. next ~~Int ( )~~ ;
               if (b
     while (true)
```