#How to approach a problem

1. Determine inputs/Output
2. Breakdown the problem into smaller parts
3. Describe each task in pseudo code
4. Test pseudo code
5. Write the code and test

# Compile-time Error

When there is something wrong with the rules of the programming language . it detects the issue while compiling the code to a Java virtual machine and the code will throw the error as a result the code will not be executed and won't run . It is also called Syntax error

# Run-time Error

It is often called logical error . Mostly the compiler does not recognise the error and runs the program but the output of the program is not as accepted . Sometimes it shows some exceptions in java line 3/0 is an exception .

# Class

Class is like a template with a group of variables and Methods/functions for specific objects . It defines the structure and the behavior of an object .

# Objects

Objects are the instances of class containing the behavior of the class.

# Variable

Variables are the location storage of any data type .All variables must be assigned before using them .

#Name

There are some naming convention of variable . The name of the variable must start with underscore(_) or latters , special symbols are not allowed in variable names . Names are case sensitive (case ,Case,cAse,caSe,casE ..all are different). Can't use reserved words in names . In general we use camel case (each word starts with uppercase letters with no space – HabiburRahman,StudentName) and names of variables and methods must start with lowercase letters (methodName,varaiable Name) . Class name must start with uppercase letters (ClassName,NewClassName)

#Encapsulation

Encapsulation is the process of hiding implementation details and providing the methods to access the end result . By using a private access modifier .

#Methods

Methods are the set of code that works for a specific field . same as we learnet the "C" Functions .

#Constructors

Constructors set the initial value for the object everytime we create an object .The constructor name must be the same as the Class name .

#Class implementation

1. Specify the Methods
2. Specify the variables
3. Specify Constructor

#Unit Testing

Unit testing verifies that a class is working correctly in a isolated situation without the complete program

#Conditions

```
        if(condition == true)
                {This will execute}
        Else
                {This will execute }
(condition)?{if true this will execute  }:{false will execute this };
#Loops (same meaning)
        for(int i=0; i<arrary.length();i++){
                dataTypeOfTheArry value = array[i];
                sout(value)
}
Enhanced for loop:
        For ( dataTypeOfTheArry arry : value){
                sout(values)
}
```

#Designing good methods

1. Cohesive Public Interface (All the methods and elements must be related to the class topic . Class Name must be something that fully represents the works of the class)
2. Minimizing dependencies (try to make the methods as simple as you can so that it is specific to a simple micro work . for example if we create a method to print average we should create different methods one for calculating average and one for printing the average that way the print and and average are different and they are not dependent on each other. We can use elsewhere we need them separately)
3. Minimizing Side Effects

# INHERITANCE

Classes that have common behavior that gets the behavior from another class called supper class . the classes that takes the behavior is called subclasses . The process is called inheritance . for example if you take Human ,Boy,Girl . in this classes Boy and and Girl have some common characteristics that come from a class called Human .

In code : We use “extends” to inherit the behaviors and “supper” to access the super class instead of the ClassName(Human)

```
/* Superclass
*/
class Human {
   private String name;
```

```
    private int age;

    public Human(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public void introduce() {
        System.out.println("Hello, I am " + name + " and I am " + age + " years old.");
    }
}

// Subclass 1
class Boy extends Human {
    public Boy(String name, int age) {
        super(name, age); // Call the superclass constructor
    }

    public void playFootball() {
        System.out.println("I'm a boy, and I love playing football!");
    }
}

// Subclass 2
class Girl extends Human {
    public Girl(String name, int age) {
        super(name, age); // Call the superclass constructor
    }

    public void playDolls() {
        System.out.println("I'm a girl, and I love playing with dolls!");
    }
}

public class InheritanceExample {
    public static void main(String[] args) {
        Boy boy = new Boy("John", 10);
        Girl girl = new Girl("Alice", 8);

        boy.introduce();
        boy.playFootball();

        girl.introduce();
        girl.playDolls();
```

```
  }
}
```

#override

Overriding means changing the work of a Method that is declared in the super class . her to override the introduce in Boy class code is given below :

```
class Boy extends Human {
     public Boy(String name, int age) {
          super(name, age);
     }
     @Override
     public void introduce() {
         System.out.println("I'm a boy, and I am "
     super.name + " and I am " + super.age +          “years old.");
     }
 }
```

#overLoading

In java overloading means using the same named Methods with different variables . For example Here in the Geometry class is one method named “area” but it has two functions one is with single arguments other with double arguments . if we call with a single parameter the square one will be called with two parameter rectangle area will be called . This is the overloading .

```
class Geometry {
    // Calculate the area of a square
    public double area(double side) {
        return side * side;
    }

    // Calculate the area of a rectangle
    public double area(double length, double width) {
        return length * width;
    }
}

public class OverloadingExample {
    public static void main(String[] args) {
        Geometry geometry = new Geometry();

        // Calculate and print the area of a square
        double squareSide = 5.0;
        double squareArea = geometry.area(squareSide);
        System.out.println("Area of the square: " + squareArea);
        // Calculate and print the area of a rectangle
        double rectangleLength = 6.0;
        double rectangleWidth = 4.0;
        double rectangleArea = geometry.area(rectangleLength,
rectangleWidth);
        System.out.println("Area of the rectangle: " +
rectangleArea);
    }
}
```

#polymorphism

In java polymorphism means using same named methods or classes for different uses by using different return types and parameters . something similar to overloading implementation .

#Abstract classes

Abstract Classes are the super classes in the inheritance where methods and variables are only declared in the super class . Each subclass must Override methods and give the value of the variables . If we implement the abstraction in the above inheritance :

```
// Abstract superclass
abstract class Human {
    private String name;
    private int age;

    public Human(String name, int age) {
        this.name = name;
        this.age = age;
    }

    // Abstract method for introduction
    public abstract void introduce();
}

// Subclass 1
class Boy extends Human {
    public Boy(String name, int age) {
        super(name, age);
    }

    @Override
    public void introduce() {
        System.out.println("I'm a boy, and I am " + super.name + "
and I am " + super.age + " years old.");
    }
    public void playFootball() {
        System.out.println("I'm a boy, and I love playing
football!");
    }
}

// Subclass 2
class Girl extends Human {
    public Girl(String name, int age) {
        super(name, age);
    }

    @Override
    public void introduce() {
        System.out.println("I'm a girl, and I am " + super.name + "
and I am " + super.age + " years old.");
    }

    public void playDolls() {
        System.out.println("I'm a girl, and I love playing with
dolls!");
    }
}

public class InheritanceExample {
    public static void main(String[] args) {
        Human boy = new Boy("John", 10);
        Human girl = new Girl("Alice", 8);

        boy.introduce();
        ((Boy) boy).playFootball(); // Casting to call
subclass-specific method

        girl.introduce();
        ((Girl) girl).playDolls(); // Casting to call
subclass-specific method
    }
}
```

#some keyWords

1.Final : the value can be assigned only once and methods can’t be overridden .

2.Protected: variable and the method will only be accessed within the class it is created .
3.Public : Things can be accessed from anywhere within the same package
4.Static: static methods and variables can be used without any object because they are object independent . common for all the objects .

#Interface

Interface is a class that contains the methods to be followed by the children classes or subclass but it doesn't implement the methods . all the subclasses must implement the method declared in the Interface class . all the methods in the interface are abstract methods .
One single class can inherit from more than one interface . for example :

```
// First interface
interface Swim {
    void swim();
}

// Second interface
interface Fly {
    void fly();
}

// Class implementing both Swim and Fly interfaces
class Bird implements Swim, Fly {
    @Override
    public void swim() {
        System.out.println("Bird can swim in the water.");
    }

    @Override
    public void fly() {
        System.out.println("Bird can fly in the sky.");
    }
}

public class MultipleInheritanceExample {
    public static void main(String[] args) {
        Bird bird = new Bird();
        bird.swim();
        bird.fly();
    }
}
```