Polymorphism:

1. Method overloading:

In a class, you can have 2 or more methods with the same name.

1. They should have diff arguments (number/ type/ order)
2. Return type/ access modifier do not matter

Class A{

Public void m1(float a, int b){}

int m1(int a, float b){ return a+ (int) b;}

}

A obj= new A();

Obj.m1(34, 45.6f);

1. Method overriding: You are providing a diff implementation for a method in your child class. So you are overriding the functionality in the child class.
2. Arguments should be exactly same.
3. Return types should be same/ covariant (parent-child relationship)
4. Access modifier cannot be more restrictive.
5. There has to be a parent child relationship

class Car{

void drive(){}

void applyBreaks(){}

Object changeGears(){ }

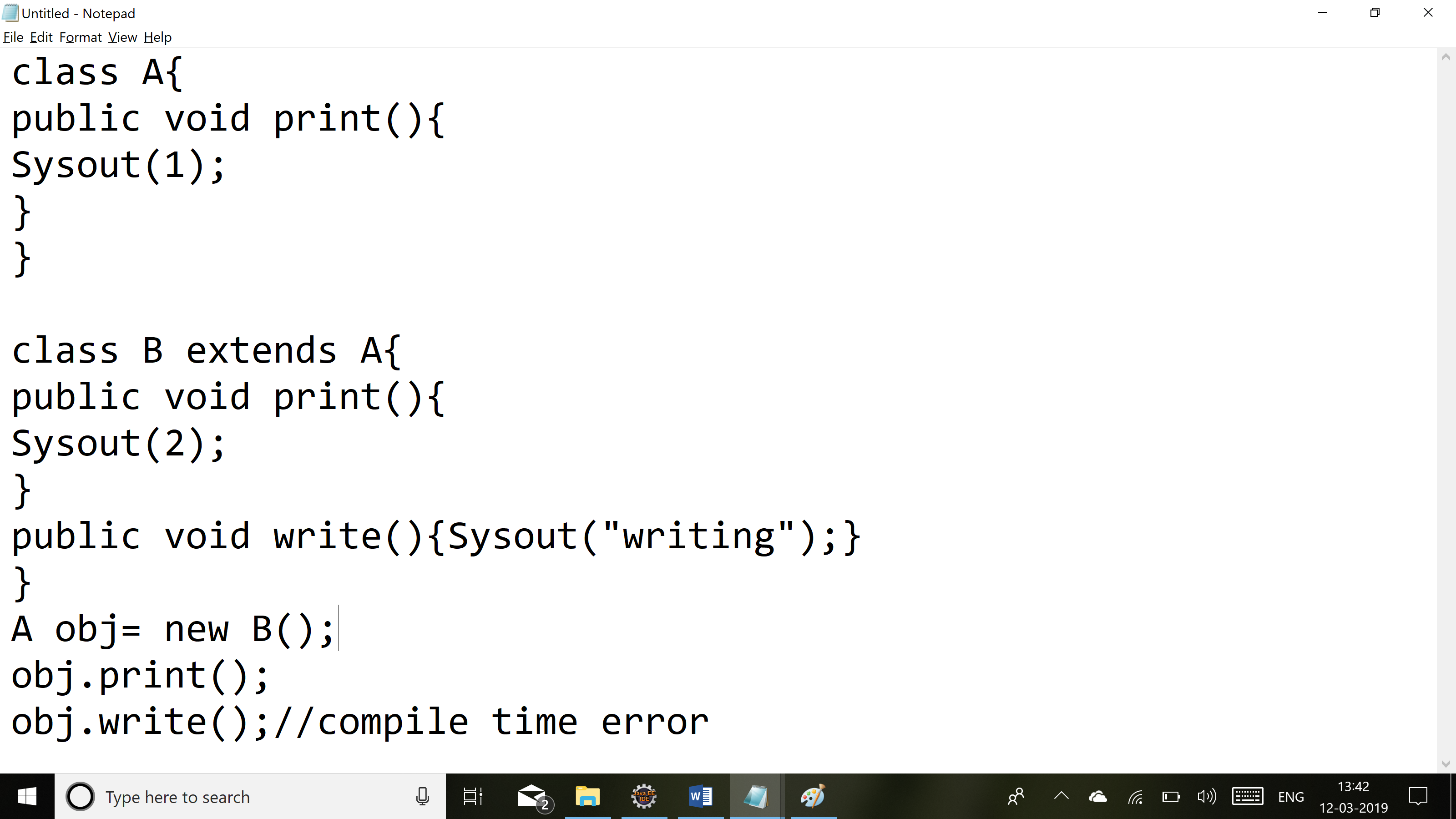
}

Class GearlessCar extends Car{

//@Override// It will check at the compile time whether u have overridden the method properly

Public Object changeGears(int a){ Sysout(“Not applicable”);}

}



**Encapsulation:**

(Capsule: Tightly binding the properties and methods of a class together)

Properties of the class to be private and accessing it using the methods

Class Person{

private String name;

private Float salary;

protected void setName(String name){

this.name= name;

}

Void setSalary(float salary){

If(salary>100000)

this.salary= salary;

}

Public String getName(){

Return name;}

}

**Inheritance: Is-a (Parent-child): All non-private methods/ properties are inherited by child class.**

Class Parent{

Void m1(){}

}

Class Child extends Parent {

}

1. A class cannot extend more than 1 class.(Multiple inheritance is not allowed).

Class Child1 extends Child{}

Parent -> Child -> Child1 (Multi-level inheritance)

**Abstract keyword**

1. Abstract method: A method which is without a body {}.

abstract void m1();

abstract class Car{

abstract void startEngine();

}

If a class has one or more abstract method, you should declare the class to be abstract and the abstract class cannot be instantiated.

Car obj= new Car();

Obj.startEngine();

The child class must implement all the abstract methods and provide definition for them else the child class also has to be declared abstract.

Class Mercedes extends Car{

Void startEngine(){}

}

abstract class A{

abstract private void m1();

}

You can have an abstract class with no abstract methods.

abstract class Person{

String name;

Int age;

}

Class Manager extends Person{

}

Class Developer extends Person{

}

**Interface:**

100% abstract class.

Interface I1{

Int i=23; // Variables are public, static, final (constant)

void m1(); //By-default methods in an interface are public, abstract

void m2(){}

}

We can also have default methods in an interface and provide default implementation for methods. (java 8)

default void m2(){ }

**Contract:**

Interface Car{

void drive();

void changeGears();

void makePayment(){// -500 }

}

abstract Class WagonR implements Car{

}

A class can implement more than 1 interface.

Class Car implements I1, I2…{}

**interface** I1{

**void** drive();

}

**interface** I2 **extends** I1, I3{

// void drive();

**void** play();

}

interface I3{}

Variables:

1) Local

2) Instance: Object of the class.

3) static: Belongs to a class.

a) variable

b) method

public class Employee {

String name;

int age;

static int noOfEmployees;

//

Employee(String name, int age){

noOfEmployees++;

this.name= name;

this.age= age;

}

//If it is a static method, u cannot access non-static members inside it

static void printCount() {

System.out.println(noOfEmployees);

}

}

Instance block : {}

class A{

int a;

{

Sysout(1);

}

A(){

Sysout(2);

}

A(int a){ //A obj= new A(5); A obj1= new A();

this();

this.a= a;

Sysout(3);

}

}

**Static block:**

**static** {

//At time when the class is loaded into the JVM. So this can be used to

//initialize your static variables.

}

**class A{**

**static{**

**Sysout(1);**

**}**

**{**

**Sysout(2);**

**}**

**A(){Sysout(3);}**

**public static void main(String args[]){**

**A obj1= new A(); A obj2= new A();**

**}**

**}**

Type of Variables:

1. Local
2. Instance:
3. Static

Access modifiers:

1. Public
2. Default
3. Protected
4. Private

Class: public , default

Polymorphism: Method overloading, Method overriding

Constructor, super, this

Abstraction: abstract, interfaces: default

Encapsulation

Inheritance