Day 2

01-03-2022

Object declaration

In Java We can take the value through keyboards lot of ways

1. Scanner class.

Scanner is a pre-defined class part of util package. Package is a collection classes and interfaces.

Syntax to create the Scanner class object.

Scanner obj = new Scanner(System.in);

Day 3

02-03-2022

OOPS Concept

object : any real word entity.

Properties or state ---🡪 have

Person

Behaviour ---🡪do/does

Place

Bank

Animal

Car

Employee

Customer

class : blue print of object or template of object or user-defined data type which help describe the object.

method or function : it is use to write set of instruction to perform a specific task.

Syntax

returnType methodName(parameterList) {

}

void info() { void means to return type.

}

int getNumber() {

coding……

return 100;

}

int add(int a, int b) {

}

Syntax to create the class object.

ClassName refereceName = new ClassName();

Types of variable or fields.

1. Instance variable:
2. The variable which declared inside a class but outside a method is known as instance variable.
3. It hold default value with respective their data types : int family 🡪0, float 🡪 0.0, char 🡪 space, String null, and boolean false.
4. Instance variable we can use directly inside a all methods but method must be part of same class and it must be non static.
5. Local variable
   1. The variable which declared inside a method is known as local variable.
   2. It hold doesn’t hold default value.
   3. Scope within that method where it declared.
6. Static variable

Constructor :

Constructor is special method which help to create the object.

Points

1. Constructor have same name as class itself.
2. Constructor no return type not even void also.
3. Constructor no need to call it will call automatically when we create the object.

If we write any constructor by default empty constructor is present.

If we want we can write more than one constructor.

Encapsulation : Binding or wrapping data and code in a single unit is known as Encapsulation.

Class

**JavaBean class or POJO class(Plain Old Java Object)**

For all variable must be private and for each variable we have to provide setter and getter methods.

Setter to set the value and getter to get the value.

When we display the reference in println internally it will call toString() method of object class.

That method it return the output as [packageName.className@coede](mailto:packageName.className@coede)

So if you want meaningful output we have to override toString() method in user defined class.

Inheritance : Inheritance is use to inherits or acquire properties and behaviour of old class to new class.

class A {

}

class B extends A {

}

Types of inheritance

1. Single inheritance

Class A { }

Class B extends A{}

1. Multilevel inheritance

Class A { }

Class B extends A { }

Class C extends B{ }

Class D extends D{ }

1. Hierarchical inheritance

Class A { }

Class B extends A{ }

Class C extends A{ }

1. Multiple inheritance

Class A { }

Class B { }

Class C extends A,B{ } Wrong in Java. Java doesn’t support this type of inheritance.

It support using interface.

OOPs relationship

1. Manager/Developer Is a relationship Employee
2. Has a relationship

abstract class Employee {

id,name,salary

}

class Manager extends Employee {

numberOfEmp;

Address add = new Address();

}

class Developer extends Employee{

techName;

}

class ProjectManager extends Manager{

}

class Address {

city,state etc.

}

Has a relationship

Association

class A {

B obj = new B(); 0, 1 or many

}

class B {

A obj1 = new A(); 0, 1 or many

}

Aggregation : but is known as weak association.

class Manager {

Address ladd = new Address();

Address padd = new Address();

}

class Address {

}

Composition it is known as strong association : composition.

class Student {

StudentHistory sh = new StudentHistory();

}

class StudentHistory {

}

Polymorphism

One name many forms.

Compile time polymorphism or static binding or early binding

Method overloading

Run time polymorphism or late binding or dynamic binding

Method overriding

abstract keyword

abstract method

abstract class

Inheritance Example

**package** com;

**class** Bike {

**void** speed() {

System.***out***.println("60km/hr");

}

}

**class** Honda **extends** Bike {

**void** color() {

System.***out***.println("Gray");

}

}

**class** Pulsar **extends** Bike {

**void** color() {

System.***out***.println("Black");

}

}

**public** **class** InheritanceDemoWithAbstrat {

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

Honda hh = **new** Honda(); hh.color(); hh.speed();

Pulsar pu = **new** Pulsar(); pu.color(); pu.speed();

}

}

Method overriding

**package** com;

**class** Bike {

**void** speed() {

System.***out***.println("60km/hr");

}

}

**class** Honda **extends** Bike {

**void** color() {

System.***out***.println("Gray");

}

}

**class** Pulsar **extends** Bike {

**void** speed() {

System.***out***.println("90km/hr");

}

**void** color() {

System.***out***.println("Black");

}

}

**public** **class** InheritanceDemoWithAbstrat {

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

Honda hh = **new** Honda(); hh.color(); hh.speed();

Pulsar pu = **new** Pulsar(); pu.color(); pu.speed();

}

}

**abstract class example**

**package** com;

**abstract** **class** Bike {

**abstract** **void** speed();

**void** mailage() {

System.***out***.println("50km/lt");

}

}

**abstract** **class** Honda **extends** Bike {

**void** color() {

System.***out***.println("Gray");

}

}

**class** Pulsar **extends** Bike {

**void** speed() {

System.***out***.println("90km/hr");

}

**void** color() {

System.***out***.println("Black");

}

}

**public** **class** InheritanceDemoWithAbstrat {

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

//Honda hh = new Honda(); hh.color(); hh.speed();

Pulsar pu = **new** Pulsar(); pu.color(); pu.speed();

}

}

Final keyword

final variable : if variable is final we can’t change the value of the variable.

final int A=10;

A=20; Error

final method : if method is final we can’t override that method.

final class : if class is final we can’t extends that class.

static keyword.

interface

class extends only one class

interface extends more than one interface

class implements more than one interface

interface can’t extends or implements class

interface is known as 100% pure abstract class.

interface Abc {

fields ; public static final int A=10;

methods public abstract void dis1();

}

interface Mno {

int B=20;

void dis2();

}

interface Xyz extends Abc,Mno{

int C=30;

void dis3();

}

class Demo implements Abc,Mno {

public void dis1() { }

public void dis2() { }

}