9/10/2023

Eg" Same as previous eg program but with little modifications. Only change in Bike.java & Demo.java.

DATE : ....../....../..........

Bike.java

```
public class Bike extends Vehicle{
    public String handle;
    public Bike(){
        super();
        this.handle = "short";
    }
public Bike(String handle, String engine, int wheels,
            int seats, int fueltank,
            String lights){
super(engine, wheels, seats, fueltank,
      lights);
 this.handle = handle;
}
public String getHandle(){
    return handle;
  }
}
```

O/p" Handle: long
Engine Type: diseal
No of Seats: 4
Fuel Tank cap: 40
Head Lamp: LED
No of wheels: 4

Demo.java

```
public class Demo{
    Psvm(S[] args){
  Bike bike = new Bike("long", "diseal", 4, 4, 40, "LED");
Sop("Handle:" +bike.getHandle());
Sop("Engine Type:" +bike.getEngine());
Sop("No of Seats:" +bike.getSeats());
Sop("
       Fuel Tank capacity:" + bike.getFuelTank());
Sop("Head Lamp Type:" + bike.getLights());
}} Sop("No of wheels:" + bike.getWheels());
```

Eg" Modification in Demo.java (toString())

```
Psvm(S[] args){
    Bike bike = new Bike("long",
              "diseal", 4, 4, 40, "LED");
  Sop(bike);
  }
}
```

O/p" org.studyeasy.child.Bike@5e91993f

To Generate toString() → Right click → Source → Generate toString()

Eg" Click Generate ← Check & Uncheck fields
Inherited methods → uncheck getClass(), hashCode(), toString()

Code will be placed in Bike.java

Bike.java

```
@Override
public String toString(){
return "Bike [getHandle()="
   + getHandle() + ", getEngine()=" +
getEngine() + ", getWheels()=" + getWheels()
+ ", getSeats()=" + getSeats() + ",
getFuelTank()=" + getFuelTank() + ",
getLights()=" + getLights() + "]";
}
```

O/p" Bike [getHandle() = long, getEngine() = diseal, getWheels() = 4, getSeats() = 4, ---]

Whenever there is ambiguity or some kind of overriding takes place then the priority of child class element would be always more.

Overriding

Simply implementing custom version of functionality provided by one parent class

Eg" Bike.java
Below generated toString() add

```
public void run(){
    SOP("Running Bike");
    Sop(toString());
}
```

Vehicle.java: add below code

```
public void run(){
    Sop("running vehicle");
}
```

→ Demo.java →

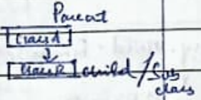
```
Psvm(S[] arg) {
  Bike bike =new Bike("long","dical",4,5,90,
                    "LED");
  bike.run();
}
```

DATE : ....../....../...........

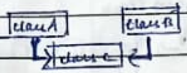
O/p : Running bike
Bike [getHandle()
 =long
 ---- ]

→ Types of Inheritance

→ Single →



→ Multiple →
Not allowed in java

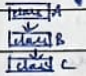


Multiple →
Inherit more than one class

Appears → : → class A & B having same Method
X

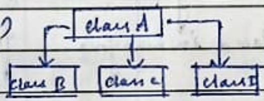
~~Java~~ not allow

→ Multilevel →

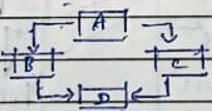


Parent class → Superclass
child class → Subclass

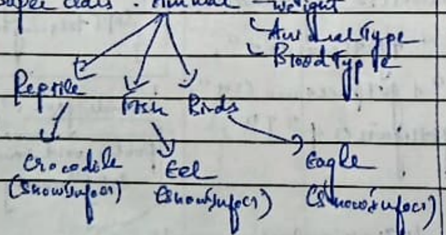
→ Hierarchial →



→ Hybrid →
combination



eg
→ Super class : Animal – height
 – weight
 – AnimalType
 – BloodType



Add ShowInfo()
to every class.

Crocodile.java

```
Public class Crocodile extends
                     Reptile {
  public Crocodile() {
    super();
    egg ="hard shelled";
  }
  Public String ShowInfo() {
   return "Crocodile [skin =" +skin+
        ", egg ---+"]";
  }
}
```

Eel.java

```
Public class Eel extends Fish {
  private String special;
   public Eel() {
     super();
     this.special ="Released
                 Electronic
                    Shock";
   }
   Public String ShowInfo() {
      return "Eel [special
           ---- +"]";
   }
```

Eagle.java

```
public class Eagle extends
                     Bird {
   public String ShowInfo() {
 return "Eagle[feather =" +feather+"
           ---]";
   }
}
```

Reptile.java

```
Public class Reptile extends
                   Animal {
 Protected String skin,egg;
 Protected boolean
                  backbone;
  Public Reptile() {
   heightInFeet =5;
   weightInKilos =20;
   animalType ="Reptile";
   bloodType ="Cold";

  this.skin ="Dry skin";
  this.backbone = true;
  this.egg ="soft shelled";
 }
 public String ShowInfo() {
   return "Reptile [skin=---
           ---"]";
  }
}
```

DATE : ....../....../...........

Fish.java

```
public class Fish extends
                Animal {
Protected boolean waterBone =true;
protected boolean gills =true;
 Public Fish() {
   animalType ="fish";
 this.waterBone ="true";
  this.gills = true;
  }
  public String ShowInfo() {

return "Fish [waterBone
      =" +"]";
  }
}
```

Bird.java

```
Public class Bird extends Animal
                          {
   Protected boolean feather
                    ="true",
   protected boolean canFly="true";
Public Bird() {
```

```
this.feather = true;
this.canFly = true;
}
}
```

Animal.java

```
public class Animal {
    protected float heightInFeet;
    protected float weightInKilos;
    protected String animalType;
    protected String bloodType;
 public Animal() {
     this.heightInFeet = 0;
     this.weightInKilos = 0;
     this.animalType = "unknown";
     this.bloodType = "unknown";
  }
 public String showInfo() {
return "Animal [......"];
 }
}
```

Main.java

```
psvm (String args) {
  Animal animal = new Animal();
  Reptile reptile = new Reptile();
② Sop (reptile.showInfo());
  Crocodile croc = new Crocodile();
① SOP (croc.showInfo());
   Eel eel = new Eel();
③ Sop (eel.showInfo());
  Eagle eagle = new Eagle();
④ Sop (eagle.showInfo());
}
}
```

DATE ....../....../..............

1} Skin = Dry skin
egg = soft shelled
backbone = true
heightIn = 5.0
weight = 20.0
animalType = reptile
blood type = cold

2} Skin = dry
egg = hard
backbone = true
he = 5.0
we = 20.0
an = reptile
blood = cold

3} special = Eel
waterbone = st
gills = t
h = 0
w = 0
animalType = fish
blood = unknown
}
}

Composition

Inheritance: Derive properties & functionalities from our parent class.

Anonymous object = Value of object pass to property

Processor

design technique to implement has a relationship b/w diff types of objects & classes.

Eg: Car, Truck, Bike } Vehicle ; More useful than Inheritance

GraphicsCard.java

```
public class GraphicsCard {
     private String brand;
     private int series;
      private String memory;
   public GraphicsCard() {
     this.brand = "Nvidia";
     this.series = 940;
     this.memory = "2GB";
  }
 public GraphicsCard(String brand,
                     int series,
                     String memory) {
  this.brand = brand;
  this.series = series;
  this.memory = memory;
 }
 @Override
  public String toString() {
  } return "GraphicsCard[brand=...]";
}
```

Processor.java

```
public class Processor {
  private String brand;
  private String series;
  private int generation;
  private int cores;
  private int threads;
   private String cacheMemory;
   private String frequency;
 private String minFrequency;
 private String maxFrequency;
  public Processor() {
   this.brand = "Intel";
   this.series = "i5 7200U";
   this.generation = 7;
   this.cores = 2;
   this.threads = 4;
   this.cacheMemory = "3MB";
   this.frequency = "2.5GHz";
  this.minFrequency = "2.5GHz";
  this.maxFrequency = "3.1GHz";
 }
 public Processor(
    String brand, String series,
   int generation, int cores,
 - int threads
    String cacheMemory,
    String frequency,
 String minFrequency,
   String maxFrequency)
 {
   this.brand = brand;
   this.series = series;
}
@Override
public String toString(){
  return "Processor[ ..... + "]";
 }
}
```

DATE : ……/……/…………

Laptop.java

```
public class Laptop{
    private float screen;
    private Processor processor;
    private String ram;
    private String hardDrive;
    private GraphicsCard graphicsCard;
   private String opticalDrive;
    private String Keyboard;
 public Laptop(){
      this.screen = 15.6f;
      this.processor = new Processor();
      this.ram = "DDR4";
      this.hardDrive = "2TB",
      this.graphicsCard = new GraphicsCard();
      this.opticalDrive = "MLT Layer";
      this.keyboard = "backlit";
  }
 public Laptop(float screen, Processor processor,
              ....){
    super();
   this.screen = screen;
  }
 @Override
  public String toString(){
return "Laptop[..... "]";
  }
}
```

Anonymous object

Hello.java

```
public class Hello{
   Psvm(String[] args){
   Laptop lappy = new Laptop();
   SOP(lappy);
  }
}
```

o/p

```
Laptop[screen=5.6,
Processor=Processor[brand="Intel",
Series=i5 6200u, generation=7,
 cores=2, threads=4,
  cacheMemory=3MB, frequency=2.5gHz,
minFrequency=2.5gHz, maxFrequency=3.1gHz],
 ram=DDR4, hardDrive=2TB,
  graphicsCard=GraphicsCard
      [brand=Nvidia, series=940,
     memory=2GB],
  opticalDrive=MLT Layer,
 Keyboard=backlit]
```

→ Generate getters for Processor class → for all fields/properties

→ Order → access method → Processor class

? → way → Access object → need to
   lots like diff      inside object

→ In Laptop.java → all getters for all properties

→ In Hello.java

```
Psvm(String[] args){
  Laptop lappy = new Laptop();
 SOP(lappy.getProcessor().
                 getBrand());
 }
}
```

o/p: Intel

→ Adding functionality

```
Laptop gamingLaptop
    = new Laptop(17f,
      new Processor("Intel",
                   ~~~~~ ""),
        "DDR4",
        "2TB",
        new GraphicsCard
         ("Nvidia", 1050,
          "4GB"),
       null,
        "backlit"
   );
```

Go to processor.java → Generator setter for frequency property.

```
public void setFrequency
        (String frequency){
 this.frequency = frequency;
 }
```

~~Add method in~~ Laptop.java

eg: Graphics Card.java

10/10/2023

DATE : ......./......./..........

```
public Class GraphicsCard {
    private String brand;
    private int Series;
    private String memory;

  public GraphicsCard() {
      this.brand = "Nvidia";
      this.Series = 940;
      this.memory = "2gb";
  }
  public GraphicsCard(String brand, int series,
                        String memory) {
 this.brand = brand;
 this.series = series;
  this.memory = memory;
 }
 @Override
  public String toString() {
        return "GraphicsCard
[brand=" + brand + ", series=" + series + ",
  memory=" + memory + "]";
  }
}
```

Processor.java

```
public class Processor {
    private String brand;
     private String series;
     private int generation;
    private int cores;
     private int threads;
    private String cacheMemory;
    private String frequency;
    private String minFrequency;
    private String maxFrequency;
```

```
public Processor() {
     this.brand = "Intel";
     this.series = "i5 7500u";
     this.generation = 8;
     this.cores = 2;
      this.threads = 4;
      this.cacheMemory = "3MB";
      this.frequency = "2.5GHz";
      this.minFrequency = "2.5GHz";
      this.maxFrequency = "3.1GHz";
   }

Public Processor(String brand, String series, int generation,
                        int cores, int threads, String cacheMemory,
                 String frequency, String minFrequency,
            String maxFrequency) {
  this.brand = brand;
  this.series = series;
  this.generation = generation;
  this.cores = cores;
  this.threads = threads;
  }
}
@Override
  public String toString() {
      return "Processor[ — ]";
  }
  public String getBrand() {
    }  return brand;
  public String getSeries() {
      return Series;
   }
  public int getGeneration() {
  }  return generation;
```

DATE : ......./......./..........

OD = null

key = board

RW

Screen = 60

Processor
= Processor
brand=Intel
series=i5 7500u
gen=8
cores=4
threads=4
cacheMemory=3MB
f=2.5 GHz
minf=2.5 GHz
maxf=3.5 GHz
ram = DDR4
hardDrive = 1TB
graphicscard
= brand=nvidia
series=1050
memory=4GB

```
public int getcores(){
    return cores;
}
public int getThreads(){
    return threads;
}
public String getCacheMemory(){
    return cacheMemory;
}
public String getFrequency(){
    return frequency;
}
public String getMinFrequency(){
    return minFrequency;
}
public String getMaxFrequency(){
    return maxFrequency;
}
public void setFrequency(String frequency){
    this.frequency = frequency;
}
}
```

Laptop.java

```
public class Laptop {
    private float screen;
    private Processor processor;
    private String vram;
    private String hardDrive;
    private GraphicsCard graphicsCard;
    private String opticalDrive;
    private String keyboard;
    public Laptop(){
        this.screen = 15.6f;
        this.processor = new Processor();
        this.vram = "DDR4";
        this.hardDrive = "2TB";
        this.graphicsCard = new GraphicsCard();
        this.opticalDrive = "MLS Layer";
        this.keyboard = "backlit";
    } ← Add line 50
@Override
public String toString(){
return "Laptop[— — ]";
}
public Processor getProcessor(){
    return processor;
}
public float getScreen(){
    return screen;
}
public String getRam(){
    return vram;
}
public String getHardDrive(){
    return hardDrive;
}
public GraphicsCard getGraphicsCard(){
    return graphicsCard;
}
public String getOpticalDrive(){
    return opticalDrive;
}

public String getKeyboard(){
    return keyboard;
}

public String gamingMode(){
    processor.setFrequency
        (processor.getMaxFrequency());
    return "Success";
} }
```

Line 50

```
public Laptop(—){
    super();
    this.screen = screens;
    ——
    ——
}
```

Hello.java

```
public class Hello {
    psvm(String[] args){
        Processor processor =
            new Processor
        ("Intel", "i7200U", 7, 4, 4, "6MB",
         "2.5GHz", "2.1GHz", "2.5GHz");

        GraphicsCard graphicsCard
            = new GraphicsCard
                ("Nvidia", 1050, "4GB");

        Laptop gamingLaptop
            = new Laptop(19f, processor,
            "DDR4", "2TB", graphicsCard,
            null, "backlit");
        Sop(gamingLaptop);
        gamingLaptop.
                gamingMode();
        Sop("Gaming mode on");
        Sop("Current frequency :" +

        gamingLaptop.getProcessor().
                getFrequency());
    }
}
```

O/p - Laptop[ - ]
Gaming Mode on
Current frequency : 2.5 GHz

## c) Encapsulation

- Unrestricted access to properties (get of values) of properties
- No specific Restriction.
- Pillar :- To give a shell to data which is inside your object (encapsulate/hide it) outside world. and give access whenever required.

DATE : ......./......./...............

- Control access to the outside world.

Eg:- Person.java

```
public class Person{
    public String name ="John",
    public int age =21;
    public String gender ="Male",
  @Override
  public String toString(){
return "Person[name=" +name +", age=" +age +",
       gender=" +gender + "]" ;
   }
 public String getName(){
    return name;
   }
 public int getAge(){
      return age;
   }
 public String getGender(){
    return gender;
   }
  }
```

make it private

No direct access or Modify properties

Hello.java

```
public class Hello {
    psvm(String[] args){
    Person john = new Person();
   SOP(john);
     john.age =-10;
      SOP(john);
  }
}
```

O/p:- Person[name=John, age=21, gender=Male]

Person[name=John, age=-10, gender=Male]

- Not only Security for not only for data properties would be initialized when we are making use of specific constructors.

Eg:- Person.java

```
public class Person{
  private String name = "John",
  private int age=25;
   private String gender;
 public Person(){
   this.name ="JohnDoe";
   this.age=25;
   this.gender="Male",
  }
 public Person (String name, int age,
                        String gender){
 this.name = name;
 this.age =age;
} this.gender =gender;
```

```
@Override
 public String toString(){
 return "Person[name=" + name
}
DATE : ......./......./............... + "]" ;
public String getName(){
   return name;
}
public int getAge(){
  return age;
}
  public String getGender(){
     return gender;
}
* public boolean ---- x
       setAge(int age){
 if(age >=10 && age<=100){
     this.age =age;
     return true;
  }
  return false;
}
}------------------------x
```

Hello.java

```
psvm(String[] args){
Person john =new Person();
 SOP(john);
Person pooja = new
        Person("Pooja",20,"Female");
  SOP(pooja);
   pooja.setAge(-10);
 SOP(pooja);
}
```

O/p :: Person [name = Joun , age = 26 , gender = Male]

Person [name : Pooja ; age : 26, gender = Female]

DATE : ......./......./...............

Person [name = Pooja, age : 26 gender : Female]

Try Setting age to 30 → you get

Person [name = Pooja; age = 30, gender = Female]

The age is -30 So age still 26.

Polymorphism (Poly & morphism → Multiple forms.)

Late Binding

Concept :: Dynamically we can bind some specific entities based on situation.

Eg :: SamsungNotes.java

```
public class SamsungNotes
        extends Phone {
   public SamsungNotes (String model){
     super(model);
  }
 public void features(){
      Sop ("Android flagship");
  }
}
```

Sequence of getter method doesn't matter Anywhere in class.

Phone.java

```
public class Phone {
    Private String model;
    public Phone (String model){
       this.model = model;
    }
```

```
public void features(){
    Sop("Feature Phone");
 }
public String getModel(){
    return model;
 }
}
```

Hello.java

```
public class Hello {
    Psvm (String args){
  Phone phone = new Phone
                ("Nokia 3310");
  Sop(phone.getModel());
  phone.features();
 SamsungNotes notes = new
                    SamsungNotes
                    ("Note 8");
  Sop(notes.getModel());
  notes.features();
 }
}
```

O/p :: Nokia 3310
Feature Phone
Note 8
Android Flagship

SamsungNotes ← Child class

Phone — Parent class

Whenever there is (is-A) relationship Below Scenario is Possible.

```
Phone notes = new SamsungNotes
                   ("Notes");
```

(Child) Parent ← Using class Constructor

Q: Object nokia3310 contains multiple forms of features() Based on req → features select & call.

DATE : ......./......./...............

Q: No features() in Nokia3310 class No ambiguity features() from parent class (Phone) called auto.

Nokia3310.java

```
Public class Nokia3310
           extends Phone {
   public Nokia3310
           (String model){
     super(model);
   }
 }
```

SamsungNotes.java

```
public class SamsungNotes
        extends Phone {
public SamsungNotes
           (String model){
    Super(model);
 }
public void features(){
  Sop("Android flagship");
 }
}
```

Phone.java

```
Public class Phone {
 Private String model;
 public Phone
   (String model){
```

```
 this model =model;
}
public void features(){
    Sop("Feature phone");
 }
```

DATE : ......../......../............

```
public String getModel(){
    return model;
  }
}
```

Anlet to ↓ Parent ext.

Hello.java

```
public class Hello {
    Psvm(S[] args){
  Phone note8 = new SamsungNote8("note8");
  Sop(note8.getModel());
 note8.features();

  Phone nokia3310 = new Nokia3310("Nokia 3310");
  Sop(nokia3310.getModel());
 nokia3310.features();
 }
}
```

O/p :- note8
Android flagship
Nokia 3310
Features Phone

→ Change Nokia3310 to Nokia
SamsungNote8 to Samsung

Eg: Hello.java

Phone() → Non Static method → Create obj. of Hello class & then only access Phone()

```
Psvm(S[] args){
 Phone nokia3310 = new Hello().
                       Phone(1);
 SoP(nokia3310.getModel());
 nokia3310.features();
}
public Phone phone(int dailyDriver){
   Switch(dailyDriver){
 Case1: return new Nokia("3310");
 Case2: return new Samsung("Note8");
  }
 return null;
 }
}
```

→ last page

(Lines to be added → Next Page)

O/p - 3310
Feature Phone
Note8
Android flagship

List of → Every Animal object accomodated → In Array.

Benefits

```
Psvm(S[] args){
 List<Animal> animals = new ArrayList<>();
Animal animal = new Animal();
Animal reptile = new Reptile();
Animal croc = new Crocodile();
Animal eel = new Eel();
Animal eagle = new Eagle();

animals.add(animal);
animals.add(reptile);
animals.add(croc);
animals.add(eel);
animals.add(eagle);
 listAnimals(animals);
}
```

DATE : ......../......../............

```
Public static void
  listAnimals
     (List<Animal>
          animals){
 for(Animal
      animal : animals){
 SoP(animal.showInfo());
 }
}
```

List<Reptile> — ; cannot work

error. ← You get

Inside Reptile class no showInfo() → we still get o/p

We get showInfo() from parent class
↓
So this is

Smart Switching / Dynamic Binding

→ Animal object can contain any Subclass object.

→ Sub class reference inside Super class object.

→ Class → User defined datatype.

→ In Java Variable of Class are set as private to achieve encapsulation.

→ Setter
getter } Methods ——→ obtain encapsulation.

DATE : ....../....../..............

→ Eg :-
```
class Bank {
        public void print() {
            SOP("Printing pass book for bank");
        }
    }
class SbiBank extends Bank {
    public void print() {
        SOP("Printing passbook for SBI Bank");
    }
}
public class Testpolymorphism {
    PSVM (S[] args) {
        Bank Bankprint = new SbiBank();
        Bankprint.print();
    }
}
```

O/P : Printing passbook for SBI Bank

```
Phone nokia3310 = new Hello().phone(1);
```

```
Phone nokia3310 = phone(dailydriver);
```

```
Phone notes8 = new Hello().phone(2)
SOP(notes8.getModel());
notes8.features();
```

x