Q1.

**Encapsulation** – Encapsulation is a java device in which provides wrapping of the variables and the methods that operate on the variables within a single class, hiding these variables from other classes, unless the class containing them provides methods for accessing.

public class EncapExample

{

private int age;

public int getAge()

{

return age;

}

public void setAge(int newAge)

{

age = newAge;

}

**Inheritance** – This class structure and hierarchy defines one superclass or parent class, acquiring the specification of properties, methods and variables of the child classes and holding these through its constructor for access/outputting. The child classes inherit methods, variables from the parent class.

public abstract class Pointinspace

{

protected double angle;

protected String colour;

protected double position;

protected double size;

protected SolarSystem solar;

public Pointinspace(double angle, String colour, double position)

{

this.angle = angle;

this.colour = colour;

this.position = position;

}

public double getAngle(){return angle;}

public abstract void move();

public class Planet extends Pointinspace

{

private double speed;

public Planet(String colour, double position, double size, double speed, SolarSystem solar, double angle)

{

super(angle, colour, position, size, solar);

this.speed = speed;

}

public void move()

{

solar.drawSolarObject(position, angle, size, colour);

angle += speed;

}

**Polymorphism** – This provides the ability for an object to have multiple forms and accessed through object references from other classes in OOP. Most commonly used in inheritance hierarchies in which the parent classes object reference denotes the child object class.

Child Class object:

public StandardTicket(String viewDay, int ticketAmount, int price, String ticketType, int endPrice)

{

super(viewDay, ticketAmount, price, ticketType, endPrice);

}

Overriding data in the Child Class from the Main class into the super Constructor:

orderSpec[0] = new StandardTicket(viewDay, adultTicketAmount, 0, "", endPrice);

orderSpec[1] = new StudentTicket(viewDay, studentTicketAmount, 0, "", endPrice);

orderSpec[2] = new ChildTicket(viewDay, childTicketAmount, 0, "", endPrice);

orderSpec[3] = new OAPTicket(viewDay, OAPTicketAmount, 0, "", endPrice);

**Abstraction** – The process of hiding the complexity of the underlying implementation of classes, essentially providing information defining what the object does rather than how It does it. An Abstract class and method hierarchy is to be instantiated throughout all child classes.

public abstract class Pointinspace

{

protected double angle;

protected String colour;

protected double position;

protected double size;

protected SolarSystem solar;

public Pointinspace(double angle, String colour, double position)

{

this.angle = angle;

this.colour = colour;

this.position = position;

}

Abstract method print() instantiated in parent class and implemented in every child class.

public abstract void print();

Child Class:

public void print()

{

if(viewDay.equals("Wednesday"))

{

int r = price - 2;

System.out.print("\n view day: " + viewDay + "\n Amount of tickets: " + ticketAmount + "\n Price per ticket: \u00a3" + price + "\n Type: " + ticketType + "\n Total to pay: \u00a3" + ticketAmount \* r);

}

else

{

System.out.print("\n view day: " + viewDay + "\n Amount of tickets: " + ticketAmount + "\n Price per ticket: \u00a3" + price + "\n Type: " + ticketType + "\n Total to pay: \u00a3" + ticketAmount \* price);

}

}