Pre Assessment

Exercise 1

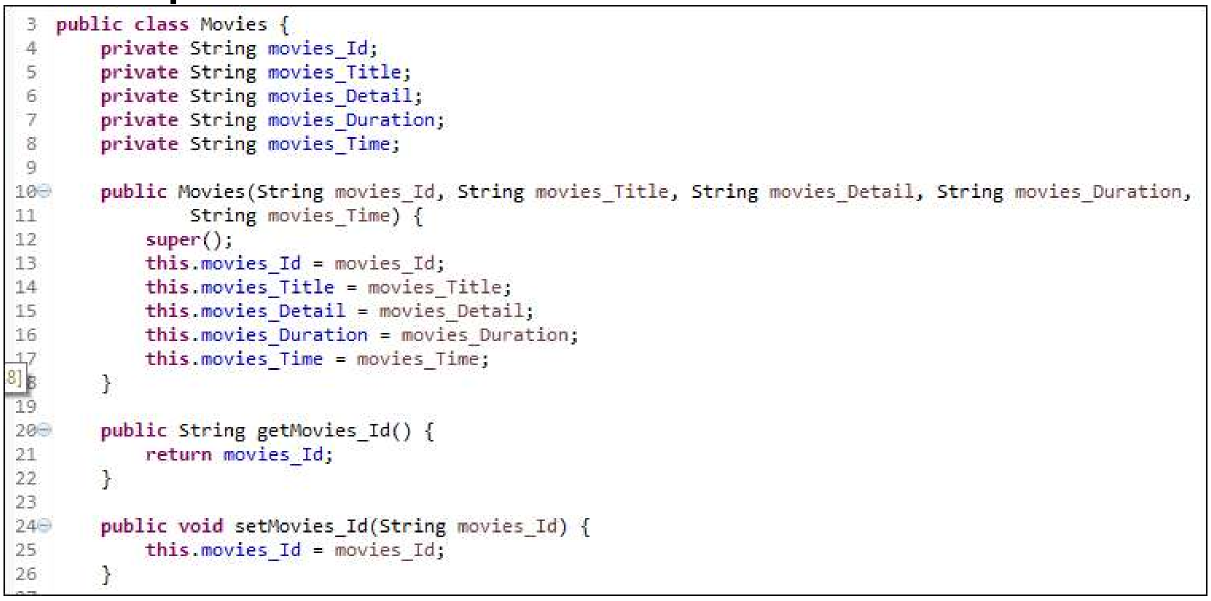
Define the following key Object Orientated Programming (OOP) principles with examples:

Encapsulation:

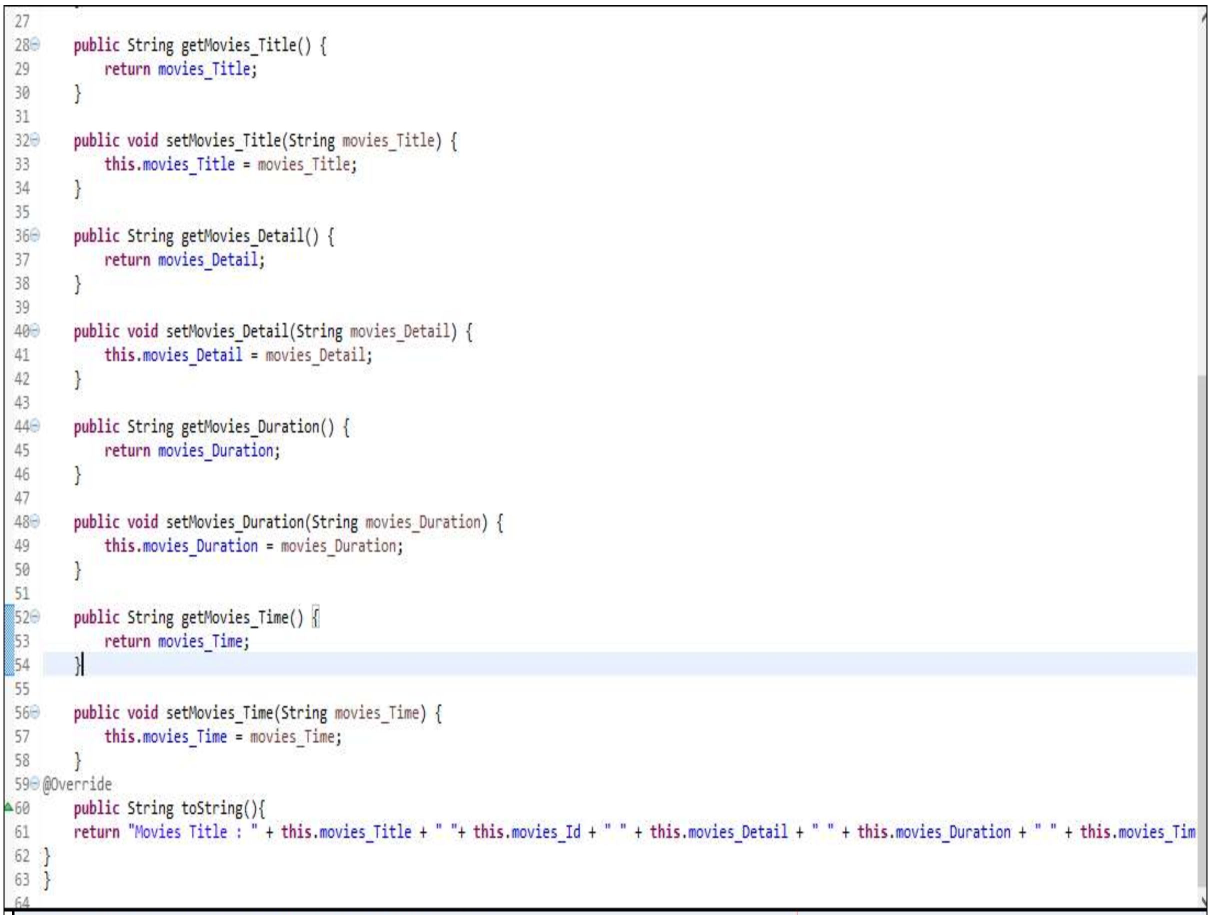
It a process of wrapping code and data together to form into a single unit. It is more than just defining accessor and mutator methods for a class, it is a broader concept of object oriented programming that consist in minimizing the interdependent of classes and it is typically implemented through information hiding.

The public setter and getter methods are the access points of the instances variable of the Standard_Ticket class. Any other classes that need to access the variable can go through get and set process as shown below.

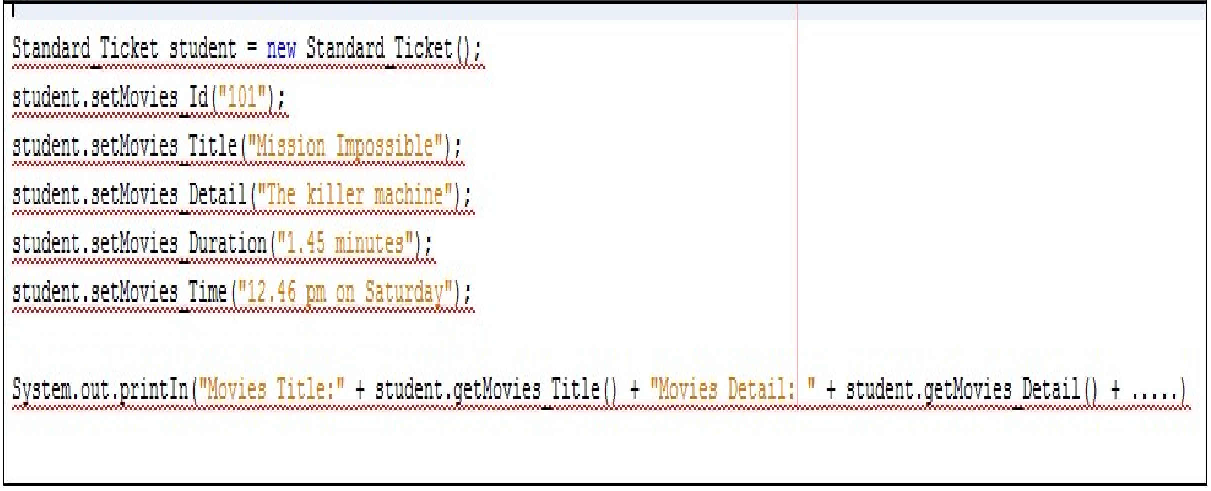
Code sample



```
public class Movies {
    private String movies_Id;
    private String movies_Title;
    private String movies_Detail;
    private String movies_Duration;
    private String movies_Time;

    public Movies(String movies_Id, String movies_Title, String movies_Detail, String movies_Duration,
            String movies_Time) {
        super();
        this.movies_Id = movies_Id;
        this.movies_Title = movies_Title;
        this.movies_Detail = movies_Detail;
        this.movies_Duration = movies_Duration;
        this.movies_Time = movies_Time;
    }

    public String getMovies_Id() {
        return movies_Id;
    }

    public void setMovies_Id(String movies_Id) {
        this.movies_Id = movies_Id;
    }
```



```java

    public String getMovies_Title() {
        return movies_Title;
    }

    public void setMovies_Title(String movies_Title) {
        this.movies_Title = movies_Title;
    }

    public String getMovies_Detail() {
        return movies_Detail;
    }

    public void setMovies_Detail(String movies_Detail) {
        this.movies_Detail = movies_Detail;
    }

    public String getMovies_Duration() {
        return movies_Duration;
    }

    public void setMovies_Duration(String movies_Duration) {
        this.movies_Duration = movies_Duration;
    }

    public String getMovies_Time() {
        return movies_Time;
    }

    public void setMovies_Time(String movies_Time) {
        this.movies_Time = movies_Time;
    }
@Override
    public String toString(){
    return "Movies Title : " + this.movies_Title + " "+ this.movies_Id + " " + this.movies_Detail + " " + this.movies_Duration + " " + this.movies_Tim
 }
 }
```



```java
Standard_Ticket student = new Standard_Ticket();
student.setMovies_Id("101");
student.setMovies_Title("Mission Impossible");
student.setMovies_Detail("The killer machine");
student.setMovies_Duration("1.45 minutes");
student.setMovies_Time("12.46 pm on Saturday");


System.out.printIn("Movies Title:" + student.getMovies_Title() + "Movies Detail: " + student.getMovies_Detail() + .....)
```

Inheritance:

Inheritance is a process whereby one class acquires the properties (methods and fields) of another.

It can also be defined as the mechanism in which one object acquires all the properties and behaviour of parent object. The inheritance represent the “**IS-A relationship”** and also known as parent-child relationship. It is used as a method overriding that allows the runtime polymorphism to achieve its function. It also make the code to be reusable.

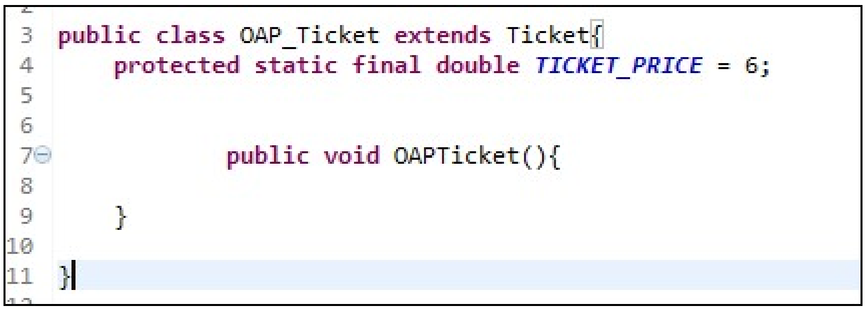
e.g.

class **subclass-name** extends **superclass-name** {

// methods and fields ……………goes here

}

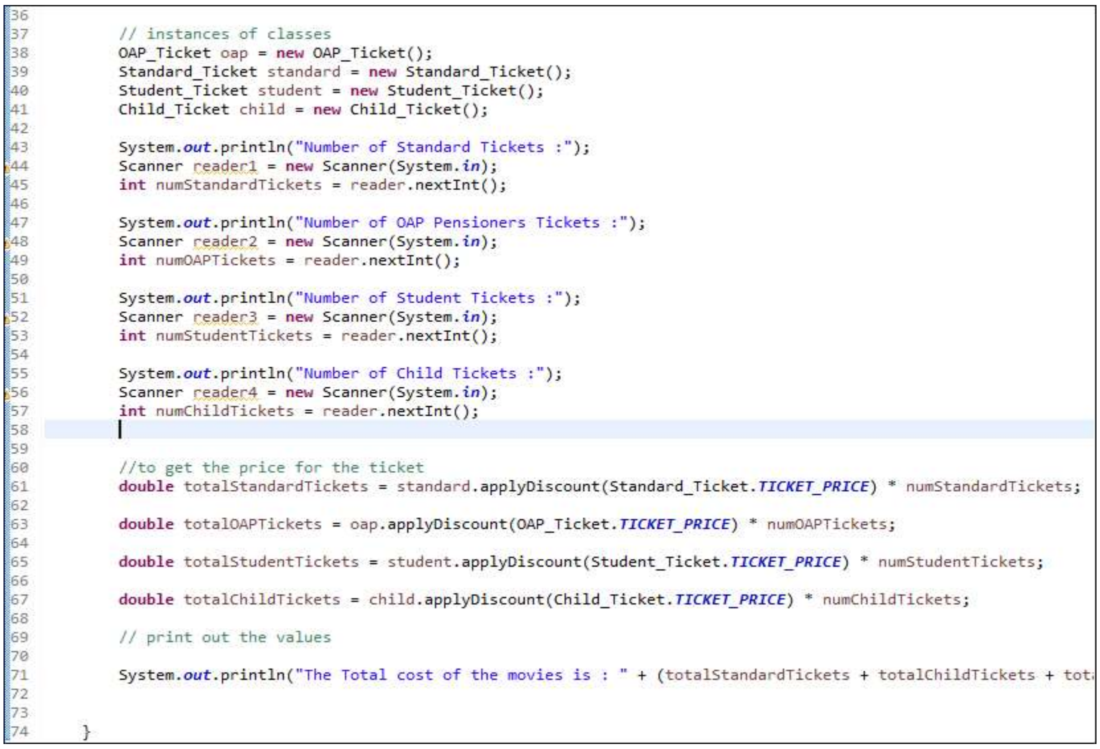
The “E**xtends”** keywords indicates that new class are been derives from the existing class.



```
public class OAP_Ticket extends Ticket{
    protected static final double TICKET_PRICE = 6;


            public void OAPTicket(){

    }

}
```

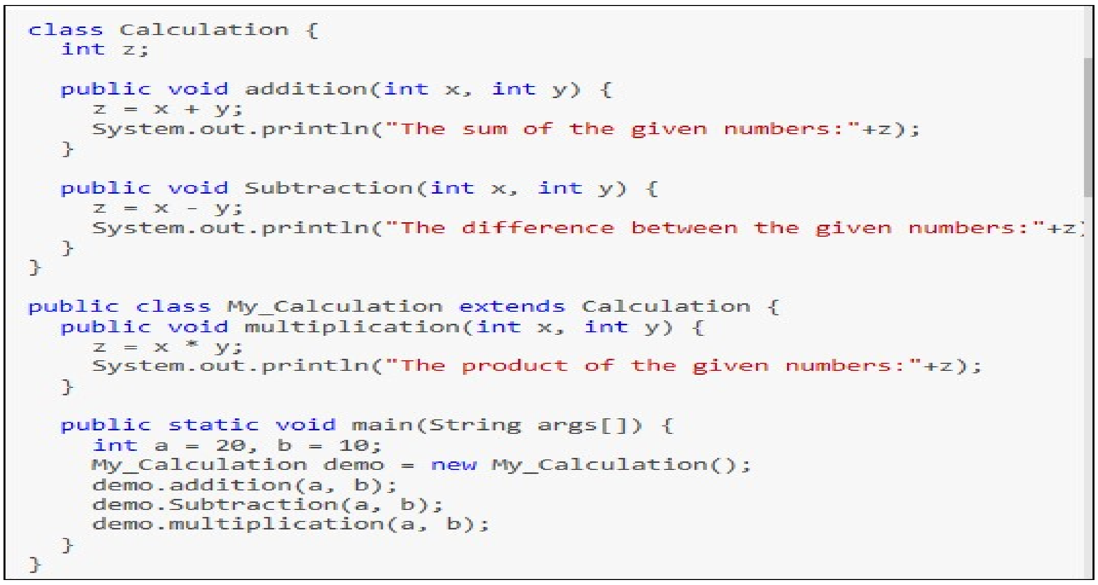
The inheritance is used mostly in the java language because it is everywhere. It is impossible to write even the tiniest java program without using inheritance.

Code Sample:



```
        // instances of classes
        OAP_Ticket oap = new OAP_Ticket();
        Standard_Ticket standard = new Standard_Ticket();
        Student_Ticket student = new Student_Ticket();
        Child_Ticket child = new Child_Ticket();

        System.out.println("Number of Standard Tickets :");
        Scanner reader1 = new Scanner(System.in);
        int numStandardTickets = reader.nextInt();

        System.out.println("Number of OAP Pensioners Tickets :");
        Scanner reader2 = new Scanner(System.in);
        int numOAPTickets = reader.nextInt();

        System.out.println("Number of Student Tickets :");
        Scanner reader3 = new Scanner(System.in);
        int numStudentTickets = reader.nextInt();

        System.out.println("Number of Child Tickets :");
        Scanner reader4 = new Scanner(System.in);
        int numChildTickets = reader.nextInt();


        //to get the price for the ticket
        double totalStandardTickets = standard.applyDiscount(Standard_Ticket.TICKET_PRICE) * numStandardTickets;

        double totalOAPTickets = oap.applyDiscount(OAP_Ticket.TICKET_PRICE) * numOAPTickets;

        double totalStudentTickets = student.applyDiscount(Student_Ticket.TICKET_PRICE) * numStudentTickets;

        double totalChildTickets = child.applyDiscount(Child_Ticket.TICKET_PRICE) * numChildTickets;

        // print out the values

        System.out.println("The Total cost of the movies is : " + (totalStandardTickets + totalChildTickets + tot


    }
```

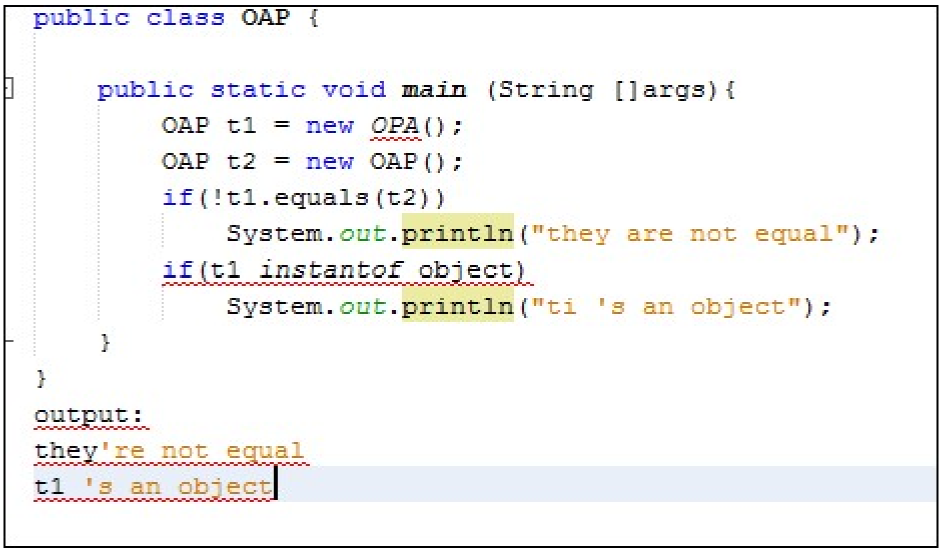
Another Example:



```
class Calculation {
   int z;

   public void addition(int x, int y) {
      z = x + y;
      System.out.println("The sum of the given numbers:"+z);
   }

   public void Subtraction(int x, int y) {
      z = x - y;
      System.out.println("The difference between the given numbers:"+z
   }
}

public class My_Calculation extends Calculation {
   public void multiplication(int x, int y) {
      z = x * y;
      System.out.println("The product of the given numbers:"+z);
   }

   public static void main(String args[]) {
      int a = 20, b = 10;
      My_Calculation demo = new My_Calculation();
      demo.addition(a, b);
      demo.Subtraction(a, b);
      demo.multiplication(a, b);
   }
}
```

https://www.bing.com/search?q=inheritance+in+java&go=Search&qs=bs&form=QBRE

Another example:



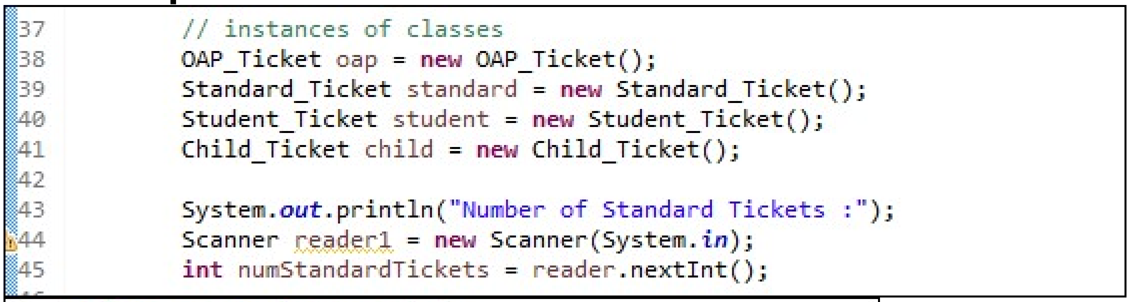
```
public class OAP {

    public static void main (String []args){
        OAP t1 = new OPA();
        OAP t2 = new OAP();
        if(!t1.equals(t2))
            System.out.println("they are not equal");
        if(t1 instantof object)
            System.out.println("ti 's an object");
    }
}
output:
they're not equal
t1 's an object
```

(Source: oracle textbook)

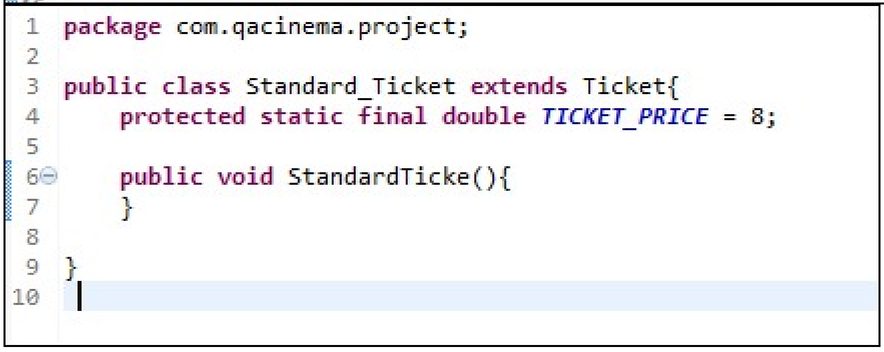
Polymorphism:

Polymorphism is the ability of two different objects to respond to the same request message in their own unique way. The dog could be train to respond to the command bark and Bird may respond to the command Chirp. While both could be trained to respond to the command speak.

- **Code sample**

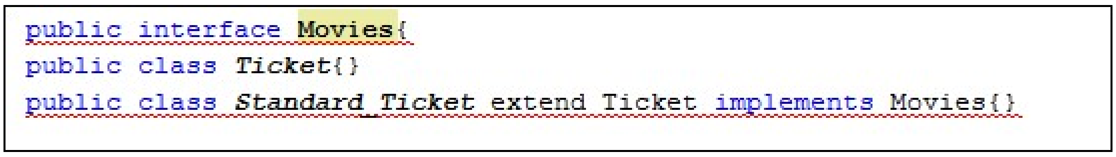


```
        // instances of classes
        OAP_Ticket oap = new OAP_Ticket();
        Standard_Ticket standard = new Standard_Ticket();
        Student_Ticket student = new Student_Ticket();
        Child_Ticket child = new Child_Ticket();

        System.out.println("Number of Standard Tickets :");
        Scanner reader1 = new Scanner(System.in);
        int numStandardTickets = reader.nextInt();
```

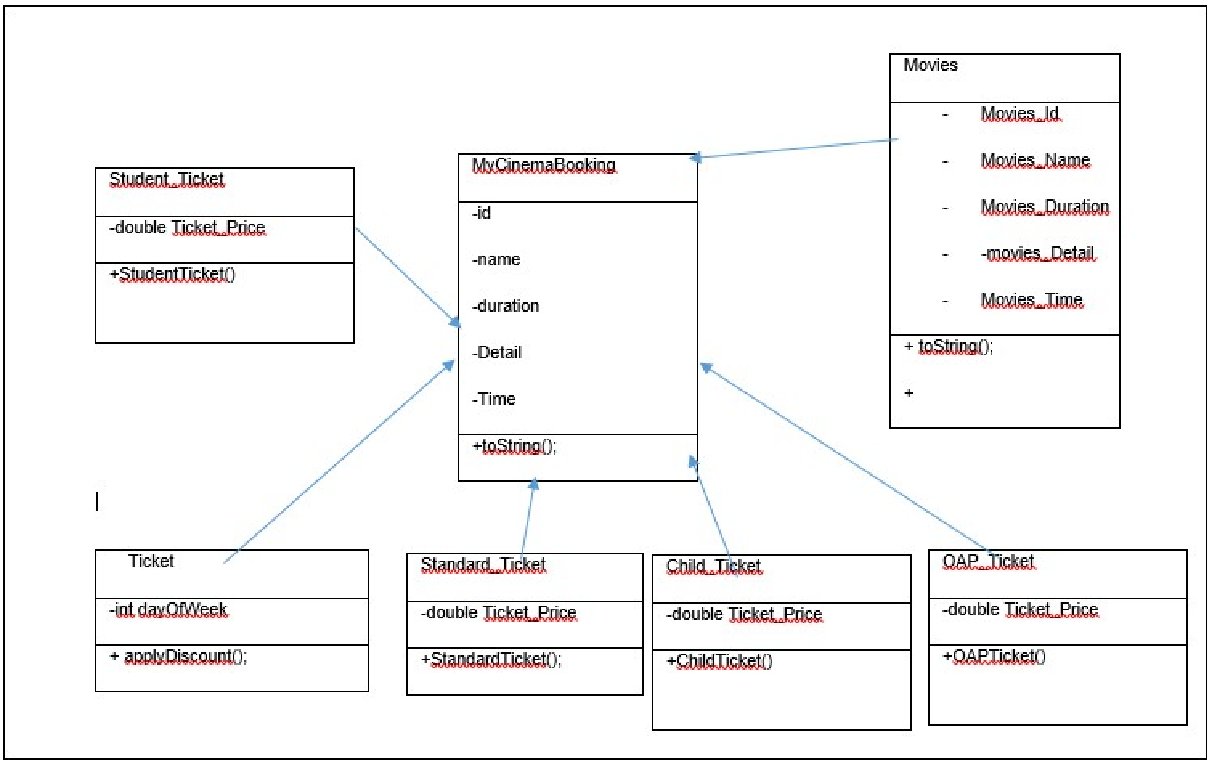


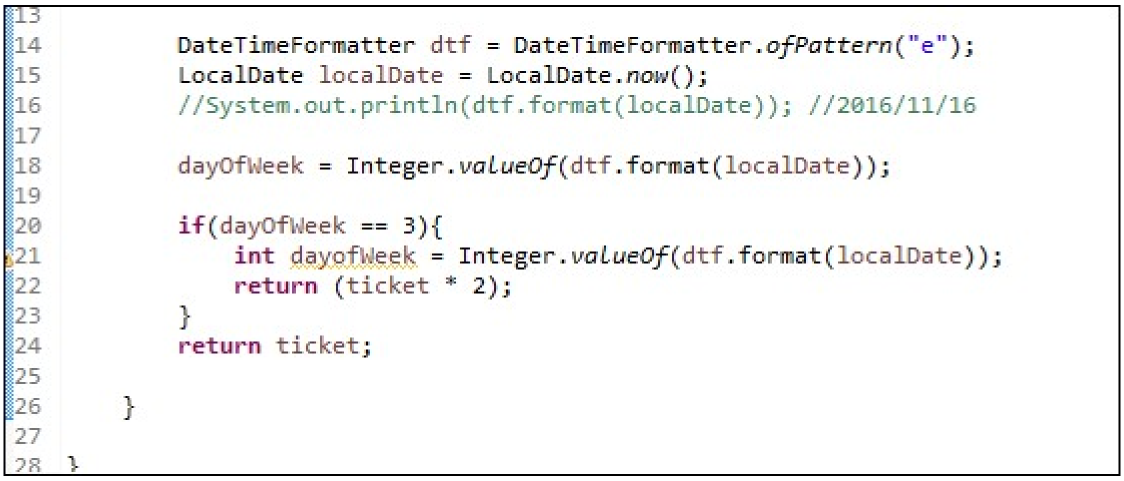
```
package com.qacinema.project;

public class Standard_Ticket extends Ticket{
    protected static final double TICKET_PRICE = 8;

    public void StandardTicke(){
    }

}
```

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent reference is used to refer to child class object.



```
public interface Movies{
public class Ticket{}
public class Standard_Ticket extend Ticket implements Movies{}
```

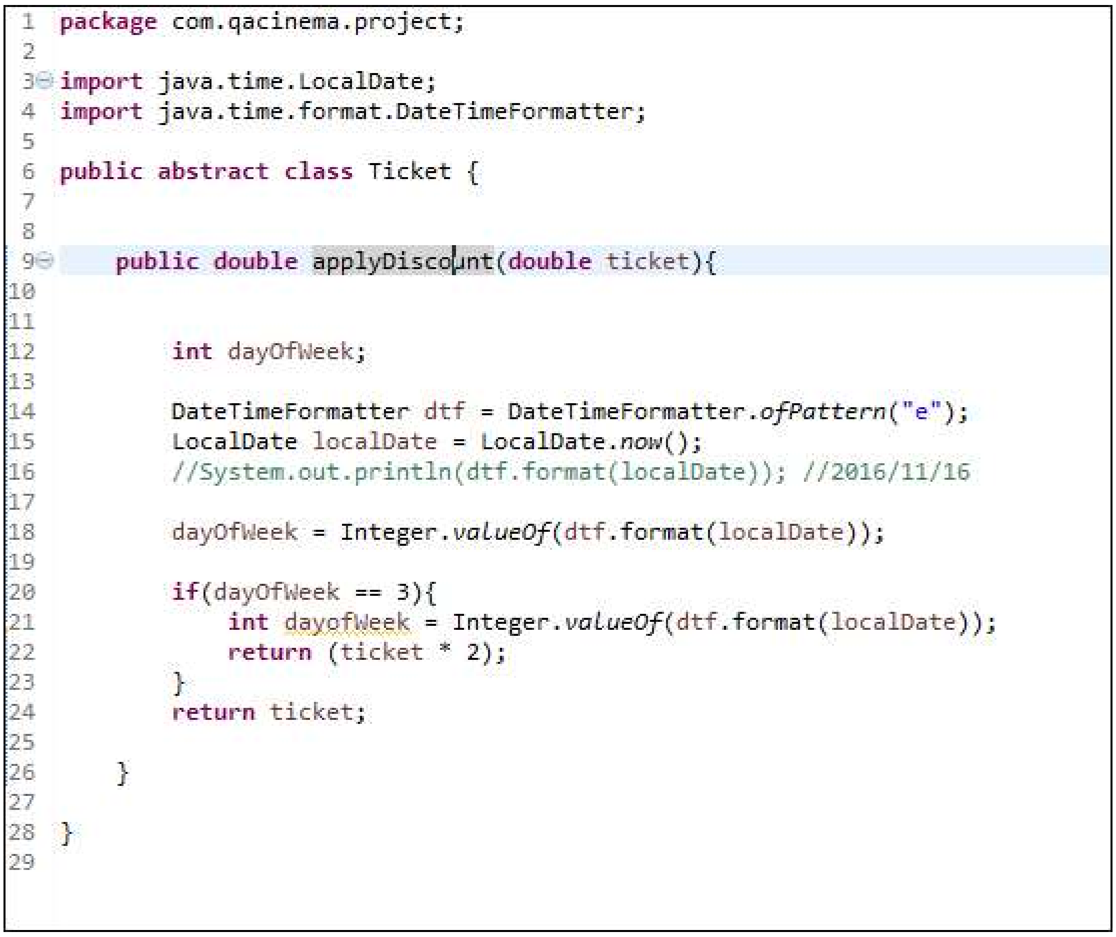




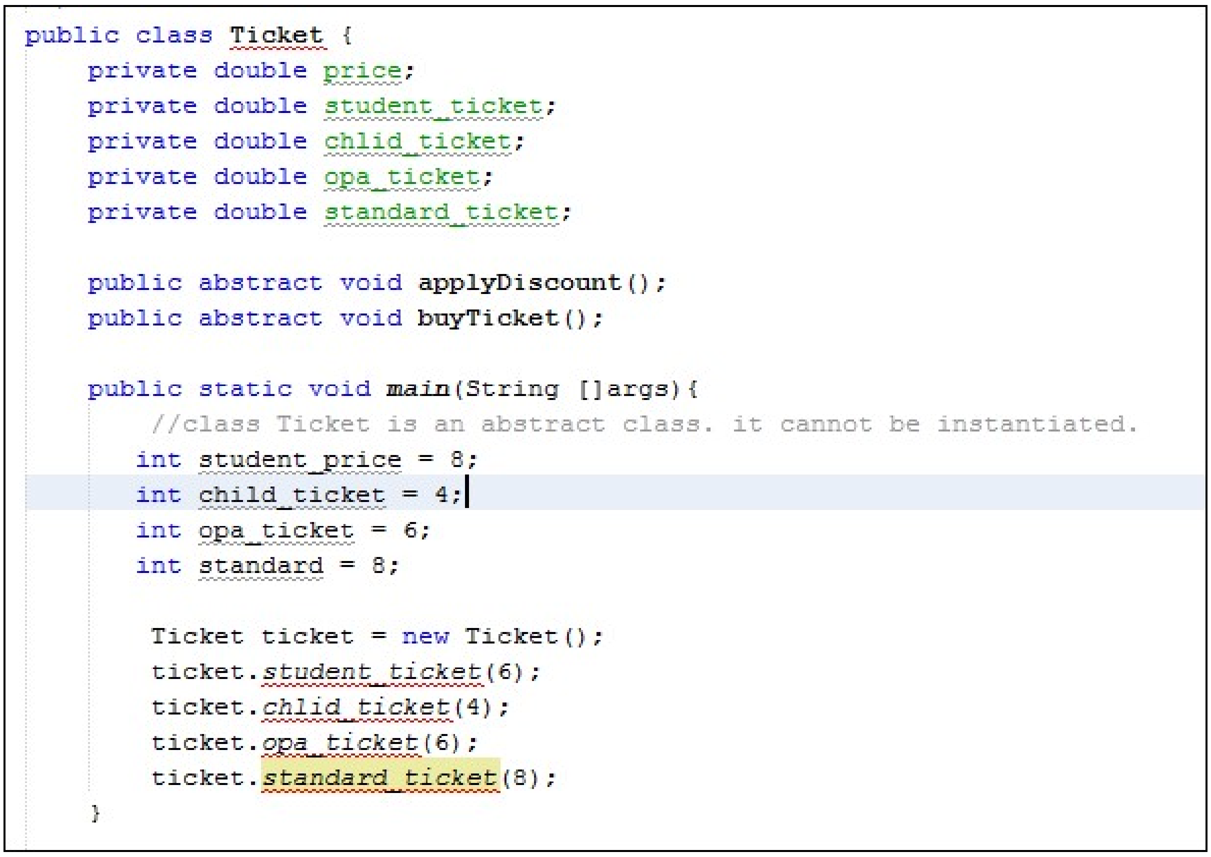
```
        DateTimeFormatter dtf = DateTimeFormatter.ofPattern("e");
        LocalDate localDate = LocalDate.now();
        //System.out.println(dtf.format(localDate)); //2016/11/16

        dayOfWeek = Integer.valueOf(dtf.format(localDate));

        if(dayOfWeek == 3){
            int dayofWeek = Integer.valueOf(dtf.format(localDate));
            return (ticket * 2);
        }
        return ticket;

    }

}
```

Abstraction:

Abstract class can never be instantiated but they can be subclassed. It means that you cannot create new instances of an abstract class.
Abstract class can be compile and execute an abstract class, as long as you don’t try to make an inheritance of it.

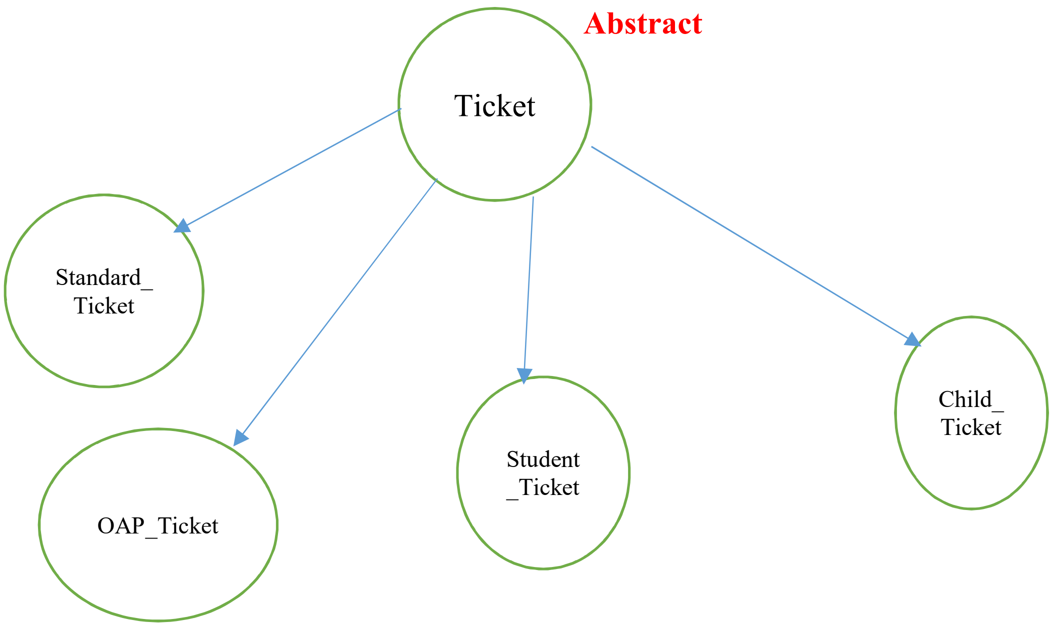


```java
package com.qacinema.project;

import java.time.LocalDate;
import java.time.format.DateTimeFormatter;

public abstract class Ticket {


    public double applyDiscount(double ticket){


        int dayOfWeek;

        DateTimeFormatter dtf = DateTimeFormatter.ofPattern("e");
        LocalDate localDate = LocalDate.now();
        //System.out.println(dtf.format(localDate)); //2016/11/16

        dayOfWeek = Integer.valueOf(dtf.format(localDate));

        if(dayOfWeek == 3){
            int dayofWeek = Integer.valueOf(dtf.format(localDate));
            return (ticket * 2);
        }
        return ticket;

    }

}
```



```
public class Ticket {
    private double price;
    private double student_ticket;
    private double chlid_ticket;
    private double opa_ticket;
    private double standard_ticket;

    public abstract void applyDiscount();
    public abstract void buyTicket();

    public static void main(String []args){
       //class Ticket is an abstract class. it cannot be instantiated.
      int student_price = 8;
      int child_ticket = 4;
      int opa_ticket = 6;
      int standard = 8;

       Ticket ticket = new Ticket();
       ticket.student_ticket(6);
       ticket.chlid_ticket(4);
       ticket.opa_ticket(6);
       ticket.standard_ticket(8);
    }
```

The information above shows that if the program is compile, there might be an error. In addition, to fix it a code sample that includes a method ending in a semicolon, but without an abstract modifier on the class or method. The method and claas abstract could be marked or change the semicolon to code (i.e change the semicolon at the end of the method declaration into a curly brace pair.)


Abstract
Ticket
Standard_ Ticket
OAP_Ticket
Student _Ticket
Child_ Ticket