TICKET RESERVATION

INVALID CARRIER-

```
public class InvalidCarrierException extends Exception{
      //FILL THE CODE HERE
      public InvalidCarrierException (String message){
          super(message);
      }

}
```

PASSENGER.JAVA-

```
//DO NOT EDIT OR ADD ANY CODE
public class Passenger {

      private String passengerName;
      private long phoneNumber;
      private String emailId;
      private String carrierName;
      private String dateOfJourney;
      private String source;
      private String destination;

      public Passenger() {
            super();
            // TODO Auto-generated constructor stub
      }

      public Passenger(String passengerName, long phoneNumber, String emailId,
String carrierName, String dateOfJourney,
                  String source, String destination) {
            super();
            this.passengerName = passengerName;
            this.phoneNumber = phoneNumber;
            this.emailId = emailId;
            this.carrierName = carrierName;
            this.dateOfJourney = dateOfJourney;
            this.source = source;
            this.destination = destination;
      }

      public String getPassengerName() {
            return passengerName;
      }
      public void setPassengerName(String passengerName) {
            this.passengerName = passengerName;
      }

      public long getPhoneNumber() {
            return phoneNumber;
      }
      public void setPhoneNumber(long phoneNumber) {
            this.phoneNumber = phoneNumber;
      }
      public String getEmailId() {
            return emailId;
      }
      public void setEmailId(String emailId) {
            this.emailId = emailId;
      }
      public String getCarrierName() {
            return carrierName;
      }
      public void setCarrierName(String carrierName) {
            this.carrierName = carrierName;
      }
      public String getDateOfJourney() {
            return dateOfJourney;
      }
      public void setDateOfJourney(String dateOfJourney) {
            this.dateOfJourney = dateOfJourney;
      }
      public String getSource() {
            return source;
      }
      public void setSource(String source) {
            this.source = source;
      }
      public String getDestination() {
            return destination;
      }
      public void setDestination(String destination) {
            this.destination = destination;
      }
}
```

PASSENGER CATERGORY-

```
import java.util.List;
@FunctionalInterface
public interface PassengerCategorization {
      abstract public List<Passenger> retrievePassenger_BySource(List<Passenger>
passengerRecord,String source);


}
```

PASSENGER UTILITY-

```
import java.util.List;
import java.io.*;
import java.util.*;


public class PassengerUtility {

      public List<Passenger> fetchPassenger(String filePath) throws Exception{

            //FILL THE CODE HERE
            List<Passenger> list = new ArrayList<Passenger>();
          String line = "";
          String splitBy = ",";
```

```
              BufferedReader br =  new BufferedReader(new FileReader(filePath));
              while((line=br.readLine())!=null){
                  String[] p = line.split(splitBy);
                 Passenger passenger = new
Passenger(p[0],Long.parseLong(p[1]),p[2],p[3],p[4],p[5],p[6]);
                  if(isValidCarrierName(passenger.getCarrierName())){
                      list.add(passenger);
                  }
              }

            return list;
      }

      public boolean isValidCarrierName (String carrierName)
      {
            //FILL THE CODE HERE
            String temp = carrierName;
          if((temp.toLowerCase()).equals("bella")){
              return true;
          }else{
              try{
              throw new InvalidCarrierException(carrierName+" is an Invalid carrier
name.");
              }
              catch(InvalidCarrierException e){
                  System.out.println(e.getMessage());
              }
          }
            return false;
      }

}
```

SKELETON VALIDATION-

```
import java.lang.reflect.Method;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.util.stream.Stream;

/**
 * @author TJ
 *
 * This class is used to verify if the Code Skeleton is intact and not modified by
participants thereby ensuring smooth auto evaluation
 *
 */
public class SkeletonValidator {

      public SkeletonValidator() {


            validateClassName("PassengerCategorization");
            validateClassName("Passenger");
            validateClassName("InvalidCarrierException");
```

```
		validateClassName("PassengerUtility");

		validateMethodSignature(
			"retrievePassenger_BySource:java.util.List",
				"PassengerCategorization");
		validateMethodSignature(
				"fetchPassenger:java.util.List",
				"PassengerUtility");
		validateMethodSignature(
				"isValidCarrierName:boolean",
				"PassengerUtility");
		validateMethodSignature(
				"searchPassengerRecord:PassengerCategorization",
				"UserInterface");
	}

	private static final Logger LOG = Logger.getLogger("SkeletonValidator");

	protected final boolean validateClassName(String className) {

		boolean iscorrect = false;
		try {
			Class.forName(className);
			iscorrect = true;
			LOG.info("Class Name " + className + " is correct");

		} catch (ClassNotFoundException e) {
			LOG.log(Level.SEVERE, "You have changed either the " + "class 
name/package. Use the correct package "
					+ "and class name as provided in the skeleton");

		} catch (Exception e) {
			LOG.log(Level.SEVERE,
					"There is an error in validating the " + "Class Name. 
Please manually verify that the "
							+ "Class name is same as skeleton before 
uploading");
		}
		return iscorrect;

	}

	protected final void validateMethodSignature(String methodWithExcptn, String 
className) {
		Class cls = null;
		try {

			String[] actualmethods = methodWithExcptn.split(",");
			boolean errorFlag = false;
			String[] methodSignature;
			String methodName = null;
			String returnType = null;

			for (String singleMethod : actualmethods) {
				boolean foundMethod = false;
				methodSignature = singleMethod.split(":");

				methodName = methodSignature[0];
				returnType = methodSignature[1];
```

```
                        cls = Class.forName(className);
                        Method[] methods = cls.getMethods();
                        for (Method findMethod : methods) {
                              if (methodName.equals(findMethod.getName())) {
                                    foundMethod = true;
                                    if (!
(findMethod.getReturnType().getName().equals(returnType))) {
                                          errorFlag = true;
                                          LOG.log(Level.SEVERE, " You have changed
the " + "return type in '" + methodName
                                                      + "' method. Please stick to
the " + "skeleton provided");

                                    } else {
                                          LOG.info("Method signature of " +
methodName + " is valid");
                                    }

                              }
                        }
                        if (!foundMethod) {
                              errorFlag = true;
                              LOG.log(Level.SEVERE, " Unable to find the given
public method " + methodName
                                          + ". Do not change the " + "given public
method name. " + "Verify it with the skeleton");
                        }

                  }
                  if (!errorFlag) {
                        LOG.info("Method signature is valid");
                  }

            } catch (Exception e) {
                  LOG.log(Level.SEVERE,
                              " There is an error in validating the " + "method
structure. Please manually verify that the "
                                          + "Method signature is same as the
skeleton before uploading");
            }
      }

}
```

USER INTERFACE-

```
import java.util.*;
import java.io.*;

public class UserInterface{
      public static PassengerCategorization searchPassengerRecord(){

            //FILL THE CODE HERE
            return (list,source)->{
                List<Passenger> result = new ArrayList<Passenger>();
                for(Passenger pass : list){

if((pass.getSource().toLowerCase()).equals(source.toLowerCase())){
```

```
                        result.add(pass);
                    }
                }
                return result;
            };
        }

        public static void main(String [] args)
        {
              //VALIDATION STARTS
               new SkeletonValidator();
              //DO NOT DELETE THIS CODE
               //VALIDATION ENDS

               PassengerCategorization pc = searchPassengerRecord();
               //FILL THE CODE HERE
               System.out.println("Invalid Carrier Records are:");
               PassengerUtility pu = new PassengerUtility();
               List<Passenger> list = null;
               try{
               list = pu.fetchPassenger(new String("PassengerRecord.txt"));
               }
               catch(FileNotFoundException e){
                   e.printStackTrace();
               }
               catch(IOException e){
                   e.printStackTrace();
               }
               catch(Exception e){
                   e.printStackTrace();
               }
               System.out.println("Enter the source to search");
               Scanner sc = new Scanner(System.in);
               String inp = sc.next();

               List<Passenger> result = pc.retrievePassenger_BySource(list,inp);
               if(result.size()==0){
                   System.out.println("No Passenger Record");
               }
               else{
                   for(Passenger passenger: result){
                       System.out.println(passenger.getPassengerName()+"
"+passenger.getPhoneNumber()+" "+passenger.getDateOfJourney()+" "+
passenger.getDestination());
                   }
               }
        }
}
```

ZEE LAPTOP AGENCY

INVALID LAPTOP-

```
package com.cts.zeelaptopagency.exception;
```

```
public class InvalidLaptopIdException extends Exception{
      public InvalidLaptopIdException() {

      }

      public InvalidLaptopIdException(String string) {
            super(string);
      }


}
```

MAIN.JAVA-

```
package com.cts.zeelaptopagency.main;
import com.cts.zeelaptopagency.service.LaptopService;
import java.util.*;
import com.cts.zeelaptopagency.skeletonvalidator.SkeletonValidator;
import java.io.*;
import com.cts.zeelaptopagency.vo.Laptop;
import com.cts.zeelaptopagency.exception.*;

public class Main {
      public static void main(String args[]) {

            // CODE SKELETON - VALIDATION STARTS
                        // DO NOT CHANGE THIS CODE
                        new SkeletonValidator();
                        // CODE SKELETON - VALIDATION ENDS

                        //Add your code here to retreive file object from Service
class
                        //Add Code here to print valid LaptopDetails returned by
Service Method
                        LaptopService l=new LaptopService();
                        File f=l.accessFile();
                        List<Laptop> lap=l.readData(f);
                        System.out.println("The Valid Laptop Details are:-");
                        for(Laptop la:lap)
                        {
                            try{
                            if(l.validate(la.getLaptopId())==true){
                            System.out.println(la.toString());
                            }
                            }
                            catch(InvalidLaptopIdException e)
                            {
                                e.printStackTrace();
                            }
                        }

}
}
```

LAPTOP SERVICE.JAVA-

```
package com.cts.zeelaptopagency.service;
```

```
import com.cts.zeelaptopagency.vo.Laptop;
import java.io.File;
import java.io.*;
import java.util.List;
import java.util.*;

import com.cts.zeelaptopagency.exception.InvalidLaptopIdException;
import com.cts.zeelaptopagency.vo.Laptop;

public class LaptopService {


      /**
       * Method to access file
       *
       * @return File
       */
      public File accessFile()
      {

            //Type Code to open text file here
            //File f=new File("LaptopDetails.txt");


            return new File("LaptopDetails.txt");  //TODO change this return value
            }


      /**
       * Method to validate LaptopId and, for invalid laptopId throw
InvalidLaptopIdException with laptopId as argument
       *
       * @param laptopid
       * @return status
       */

      public boolean validate(String laptopId)throws InvalidLaptopIdException {

            if(laptopId.toUpperCase().startsWith("ZEE"))
            {

            }else{
                throw new InvalidLaptopIdException(laptopId);
            }
            return true;
         //TODO change this return value
      }


      /**
       * Method to read file ,Do necessary operations , writes validated data to
List and prints invalid laptopID in its catch block
       *
       * @param file
       * @return List
       */

      public List<Laptop> readData(File file)
      { String s1="";
```

```
            int c;
            FileInputStream file1;
            List<Laptop> lap=new LinkedList<>(); ;
            try{
              file1=new FileInputStream(file);


            while((c=file1.read())!=-1)
            {
                s1+=(char)c;
            }
            }catch(FileNotFoundException e)
            {
                e.printStackTrace();
            }catch(IOException e)
            {
                 e.printStackTrace();
            }
            String[] arr=s1.split("\n");
            String[] laptopids=new String[4];
            Laptop l;
            for(String s:arr)
            {


                l=new Laptop();
                laptopids=s.split(",");
                l.setLaptopId(laptopids[0]);
                l.setCustomerName(laptopids[1]);
                l.setBasicCost(Double.parseDouble((laptopids[2])));
                l.setNoOfDays(Integer.parseInt(laptopids[3]));
                this.calculateFinalAmount(l);
                l.setTotalAmount(l.getBasicCost()*l.getNoOfDays());
                lap.add(l);

                }


                    return lap;  //TODO change this return value

        }




        /**
         * Method to find and set totalAmount based on basicCost and noOfdays
         *
         *
         *
         */

        public void calculateFinalAmount(Laptop l)

        {
              //Type code here to calculate totalAmount based on no of days and basic
cost
              double d=l.getBasicCost()*l.getNoOfDays();
              l.setTotalAmount(d);
```

```
      }

}
```

SKELETON VALIDATOR-

```
package com.cts.zeelaptopagency.skeletonvalidator;


       import java.lang.reflect.Method;
          import java.util.logging.Level;
           import java.util.logging.Logger;


          public class SkeletonValidator {

            public SkeletonValidator() {

      validateClassName("com.cts.zeelaptopagency.service.LaptopService");
                  validateClassName("com.cts.zeelaptopagency.vo.Laptop");

                  validateMethodSignature(

      "accessFile:java.io.File,validate:boolean,readData:java.util.List",
                              "com.cts.zeelaptopagency.service.LaptopService");

            }

            private static final Logger LOG =
Logger.getLogger("SkeletonValidator");
            protected final boolean validateClassName(String className) {

                  boolean iscorrect = false;
                  try {
                        Class.forName(className);
                        iscorrect = true;
                        LOG.info("Class Name " + className + " is correct");

                  } catch (ClassNotFoundException e) {
                        LOG.log(Level.SEVERE, "You have changed either the " +
"class name/package. Use the correct package "
                                    + "and class name as provided in the
skeleton");

                  } catch (Exception e) {
                        LOG.log(Level.SEVERE,
                                    "There is an error in validating the " + "Class
Name. Please manually verify that the "
                                                + "Class name is same as skeleton
before uploading");
                  }

                  return iscorrect;
            }

            protected final void validateMethodSignature(String methodWithExcptn,
String className) {
```

```
                  Class cls = null;
                  try {

                        String[] actualmethods = methodWithExcptn.split(",");
                        boolean errorFlag = false;
                        String[] methodSignature;
                        String methodName = null;
                        String returnType = null;

                        for (String singleMethod : actualmethods) {
                              boolean foundMethod = false;
                              methodSignature = singleMethod.split(":");

                              methodName = methodSignature[0];
                              returnType = methodSignature[1];
                              cls = Class.forName(className);
                              Method[] methods = cls.getMethods();
                              for (Method findMethod : methods) {
                                    if (methodName.equals(findMethod.getName())) {
                                          foundMethod = true;
                                          if (!
(findMethod.getReturnType().getName().equals(returnType))) {
                                                errorFlag = true;
                                                LOG.log(Level.SEVERE, " You have
changed the " + "return type in '" + methodName
                                                            + "' method. Please
stick to the " + "skeleton provided");

                                          } else {
                                                LOG.info("Method signature of " +
methodName + " is valid");
                                          }

                                    }
                              }
                              if (!foundMethod) {

                                    errorFlag = true;
                                    LOG.log(Level.SEVERE, " Unable to find the
given public method " + methodName
                                                + ". Do not change the " + "given
public method name. " + "Verify it with the skeleton");
                              }

                        }
                        if (!errorFlag) {
                              LOG.info("Method signature is valid");
                        }

                  } catch (Exception e) {
                        LOG.log(Level.SEVERE,
                                    " There is an error in validating the " +
"method structure. Please manually verify that the "
                                                + "Method signature is same as the
skeleton before uploading");
                  }
            }

          }
```

LAPTOP.JAVA-

```
package com.cts.zeelaptopagency.vo;
/**
 * Value Object - Laptop
 *
 */

public class Laptop {
      private String laptopId;
      private String customerName;
      private double basicCost;
      private int noOfDays;
      private double totalAmount;



      public Laptop()
      {

      }
      public String toString()
      {
          return "Laptop [laptopId="+this.getLaptopId()+",
customerName="+this.getCustomerName()+", basicCost="+this.getBasicCost()+",
noOfDays="+this.getNoOfDays()+", totalAmount="+this.getTotalAmount()+"]";
      }
      public String getLaptopId() {
            return laptopId;

      }
      public void setLaptopId(String laptopId) {
            this.laptopId = laptopId;
      }
      public String getCustomerName() {
            return customerName;
      }
      public void setCustomerName(String customerName) {
            this.customerName = customerName;
      }
      public double getBasicCost() {
            return basicCost;
      }
      public void setBasicCost(double basicCost) {
            this.basicCost = basicCost;
      }
      public int getNoOfDays() {
            return noOfDays;
      }
      public void setNoOfDays(int noOfDays) {
            this.noOfDays = noOfDays;
      }
      public double getTotalAmount() {
            return totalAmount;
      }
      public void setTotalAmount(double totalAmount) {
            this.totalAmount = totalAmount;
      }





}
```

LAPTOP DETAILS-

```
Laptop Details:
ZEE01,Jack,2000.50,4
ZEE02,Dev,4000.00,3
EEZ03,John,4500.00,5
ZAE04,Milan,3500.00,4
ZEE05,Surya,2500.50,7
ZEE06,Milan,5000.00,6
```

DOLLAR CITY THEME PARK

USER INTERFACE-

```
package com.ui;

import java.util.Scanner;

import com.utility.ThemeParkBO;

public class UserInterface {
      public static void main(String[] args) {
            Scanner sc = new Scanner(System.in);

            // Fill the UI code
            boolean flag = true;
            int choice = 0;
            ThemeParkBO park = new ThemeParkBO();
            while(flag){

                System.out.println("1.Add booking details");
                System.out.println("2.Average customer booked");
                System.out.println("3.Exit");
                System.out.println("Enter your choice");
                choice = sc.nextInt();

                switch(choice){
                    case 1:
                    System.out.println("Enter the day");
                    String day = sc.next();
                    System.out.println("Enter the customer count");
                    int cc = sc.nextInt();
                    park.addBookingDetails(cc);

                    break;
               case 2:
                   double res = park.findAverageCustomerBooked();
                   if(res==0){
                       System.out.println("No records found");
                       //break;
                   }
                   else{
                       System.out.println(res);
                       //break;
                   }
                   break;
               case 3:
                   System.out.println("Thank you for using the application");
                   flag = false;
                   break;
                }
            }

      }
}
```

THEMEPARKBO.JAVA-

```
package com.utility;

import com.ui.UserInterface;
import java.util.*;
import java.util.List;

public class ThemeParkBO {

      private List<Integer> bookingList = new ArrayList<>();

      public List<Integer> getBookingList() {
            return bookingList;
      }

      public void setBookingList(List<Integer> bookingList) {
            this.bookingList = bookingList;
      }

      // This Method should add the customerCount passed as argument into the
      // bookingList

      public void addBookingDetails(int customerCount) {

            // Fill the Code here
            bookingList.add(customerCount);

      }

      /*
       * This method should return the average customer booked based on the
       * customerCount values available in the bookingList.
       */

      public double findAverageCustomerBooked() {
            double avg;

            // Fill the Code here
            double count = 0;
            double counter = 0;
            for(int i=0;i<bookingList.size();++i){
                count+=bookingList.get(i);
                counter++;
            }

        if(counter==0) return 0;
        avg = count/counter;
            return avg;
      }
}
```

PASSENGER

PASSENGER UTILITY-

```
import java.util.List;
import java.io.*;
import java.util.*;


public class PassengerUtility {

      public List<Passenger> fetchPassenger(String filePath) throws Exception{

            //FILL THE CODE HERE
            List<Passenger> list = new ArrayList<Passenger>();
          String line = "";
          String splitBy = ",";

              BufferedReader br =  new BufferedReader(new FileReader(filePath));
              while((line=br.readLine())!=null){
                  String[] p = line.split(splitBy);
                 Passenger passenger = new
Passenger(p[0],Long.parseLong(p[1]),p[2],p[3],p[4],p[5],p[6]);
                  if(isValidCarrierName(passenger.getCarrierName())){
                      list.add(passenger);
                  }
              }

            return list;
      }

      public boolean isValidCarrierName (String carrierName)
      {
            //FILL THE CODE HERE
            String temp = carrierName;
          if((temp.toLowerCase()).equals("bella")){
              return true;
          }else{
              try{
              throw new InvalidCarrierException(carrierName+" is an Invalid carrier
name.");
              }
              catch(InvalidCarrierException e){
                  System.out.println(e.getMessage());
              }
          }
            return false;
      }

}
```

PASSENGER CATEGORIZATION-

```
//DO NOT ADD OR EDIT ANY CODE HERE
import java.util.List;
@FunctionalInterface
public interface PassengerCategorization {
      abstract public List<Passenger> retrievePassenger_BySource(List<Passenger>
passengerRecord,String source);


}
```

PASSENGER SKELETION-

```
import java.lang.reflect.Method;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.util.stream.Stream;

/**
 * @author TJ
 *
 * This class is used to verify if the Code Skeleton is intact and not modified by
participants thereby ensuring smooth auto evaluation
 *
 */
public class SkeletonValidator {

      public SkeletonValidator() {


            validateClassName("PassengerCategorization");
            validateClassName("Passenger");
            validateClassName("InvalidCarrierException");
            validateClassName("PassengerUtility");

            validateMethodSignature(
                  "retrievePassenger_BySource:java.util.List",
                        "PassengerCategorization");
            validateMethodSignature(
                        "fetchPassenger:java.util.List",
                        "PassengerUtility");
            validateMethodSignature(
                        "isValidCarrierName:boolean",
                        "PassengerUtility");
            validateMethodSignature(
                        "searchPassengerRecord:PassengerCategorization",
                        "UserInterface");
      }

      private static final Logger LOG = Logger.getLogger("SkeletonValidator");
      protected final boolean validateClassName(String className) {

            boolean iscorrect = false;
            try {
                  Class.forName(className);
                  iscorrect = true;
                  LOG.info("Class Name " + className + " is correct");

            } catch (ClassNotFoundException e) {
                  LOG.log(Level.SEVERE, "You have changed either the " + "class 
name/package. Use the correct package "
                              + "and class name as provided in the skeleton");

            } catch (Exception e) {
                  LOG.log(Level.SEVERE,
                              "There is an error in validating the " + "Class Name. 
Please manually verify that the "
                                          + "Class name is same as skeleton before 
uploading");
            }
            return iscorrect;

      }

      protected final void validateMethodSignature(String methodWithExcptn, String 
className) {
            Class cls = null;
            try {

                  String[] actualmethods = methodWithExcptn.split(",");
                  boolean errorFlag = false;
                  String[] methodSignature;
                  String methodName = null;
                  String returnType = null;

                  for (String singleMethod : actualmethods) {
                        boolean foundMethod = false;
                        methodSignature = singleMethod.split(":");

                        methodName = methodSignature[0];
                        returnType = methodSignature[1];
                        cls = Class.forName(className);
                        Method[] methods = cls.getMethods();
                        for (Method findMethod : methods) {
                              if (methodName.equals(findMethod.getName())) {
                                    foundMethod = true;
                                    if (!
(findMethod.getReturnType().getName().equals(returnType))) {
                                          errorFlag = true;
                                          LOG.log(Level.SEVERE, " You have changed 
the " + "return type in '" + methodName
                                                      + "' method. Please stick to 
the " + "skeleton provided");

                                    } else {
                                          LOG.info("Method signature of " + 
methodName + " is valid");
                                    }
                              }
                        }
                        if (!foundMethod) {
                              errorFlag = true;
                              LOG.log(Level.SEVERE, " Unable to find the given
public method " + methodName
                                          + ". Do not change the " + "given public
method name. " + "Verify it with the skeleton");
                        }

                  }
                  if (!errorFlag) {
                        LOG.info("Method signature is valid");
                  }

            } catch (Exception e) {
                  LOG.log(Level.SEVERE,
                              " There is an error in validating the " + "method
structure. Please manually verify that the "
                                          + "Method signature is same as the
skeleton before uploading");
            }
      }

}
```

PASSENGER USER INTERFACE-

```
import java.util.*;
import java.io.*;

public class UserInterface{
      public static PassengerCategorization searchPassengerRecord(){

            //FILL THE CODE HERE
            return (list,source)->{
                List<Passenger> result = new ArrayList<Passenger>();
                for(Passenger pass : list){

if((pass.getSource().toLowerCase()).equals(source.toLowerCase())){
                        result.add(pass);
                    }
                }
                return result;
            };
      }

      public static void main(String [] args)
      {
            //VALIDATION STARTS
             new SkeletonValidator();
            //DO NOT DELETE THIS CODE
             //VALIDATION ENDS

             PassengerCategorization pc = searchPassengerRecord();
             //FILL THE CODE HERE
             System.out.println("Invalid Carrier Records are:");
             PassengerUtility pu = new PassengerUtility();
             List<Passenger> list = null;
             try{
             list = pu.fetchPassenger(new String("PassengerRecord.txt"));
             }
             catch(FileNotFoundException e){
                 e.printStackTrace();
             }
             catch(IOException e){
                 e.printStackTrace();
             }
             catch(Exception e){
                 e.printStackTrace();
             }
             System.out.println("Enter the source to search");
             Scanner sc = new Scanner(System.in);
             String inp = sc.next();

             List<Passenger> result = pc.retrievePassenger_BySource(list,inp);
             if(result.size()==0){
                 System.out.println("No Passenger Record");
             }
             else{
                 for(Passenger passenger: result){
                     System.out.println(passenger.getPassengerName()+"
"+passenger.getPhoneNumber()+" "+passenger.getDateOfJourney()+" "+
passenger.getDestination());
                 }
             }
      }
}
```

INVALID CARRIER EXEMPTION -

```
public class InvalidCarrierException extends Exception{
      //FILL THE CODE HERE
      public InvalidCarrierException (String message){
          super(message);
      }

}
```

EMPLOYEE SALARY

EMPLOYEE-

```
public class Employee {

// Fill the code
private String employeeName;
private int employeeId;
private int incrementPercentage;
private double salary;

public void setEmployeeId(int employeeId){
this.employeeId=employeeId;
}
```

```
public int getEmployeeId(){
return employeeId;
}
public void setEmployeeName(String employeeName){
this.employeeName=employeeName;
}
public String getEmployeeName(){
return employeeName;
}
public void setSalary(double salary){
this.salary=salary;
}
public double getSalary(){
return salary;
}
public void setIncrementPercentage(int incrementPercentage){
this.incrementPercentage=incrementPercentage;
}
public int getIncrementPercentage(){
return incrementPercentage;
}
public Employee(int employeeId,String employeeName,double salary){
this.employeeId=employeeId;
this.employeeName=employeeName;
this.salary=salary;
}
public void findIncrementPercentage(int yearsOfExperience){
//Calculate the incremented salay of the employee
if(yearsOfExperience>=1&&yearsOfExperience<=5){
incrementPercentage=15;
}
else if(yearsOfExperience>=6&&yearsOfExperience<=10){
incrementPercentage=30;
}
else if(yearsOfExperience>=11&&yearsOfExperience<=15){
incrementPercentage=45;
}
}
public double calculateIncrementSalary(){
double incrementedSalary=salary+((salary*(double)incrementPercentage)/100);
return incrementedSalary;
}
```

MAIN .JAVA-

```
import java.util.*;
public class Main {

public static void main(String[] args)
{
Scanner read=new Scanner(System.in);

//Fill the code
try
{
System.out.println("Enter the Employee Id");
int id=Integer.parseInt(read.nextLine());
System.out.println("Enter the Employee Name");
String name=read.nextLine();
System.out.println("Enter the salary");
double salary=Double.parseDouble(read.nextLine());
System.out.println("Enter the Number of Years in Experience");
int exp_year=Integer.parseInt(read.nextLine());
Employee e=new Employee(id,name,salary);
e.findIncrementPercentage(exp_year);

double incrementedSalary=e.calculateIncrementSalary();
System.out.printf("Incremented Salary %.2f", incrementedSalary);
}
catch(Exception e)
{
System.out.println(e);
}
}

}
```

HOME APPLIANCES

USER INTERFACE-

```
import java.util.*;
public class HomeAppliances {
              public static void main(String[] args) {
                             Scanner sc = new Scanner(System.in);
                             System.out.println("Enter Product Id");
                             String id = sc.nextLine();
                             System.out.println("Enter Product Name");
                             String name = sc.nextLine();
                             switch (name)
                             {
                             case "AirConditioner":
                             {
                                           System.out.println("Enter Batch Id");
                                           String batch = sc.next();
                                           System.out.println("Enter Dispatch
date");
                                           String date = sc.next();
                                           System.out.println("Enter Warranty
Years");
                                           int years = sc.nextInt();
                                           System.out.println("Enter type of Air
Conditioner");
                                           String type = sc.nextLine();
                                           System.out.println("Enter quantity");
                                           double capac = sc.nextDouble();
                                           AirConditioner ob1 = new
AirConditioner(id, name, batch, date, years, type,
capac);
                                           double price =
ob1.calculateProductPrice();
                                           System.out.printf("Price of the Product
is %.2f ", price);
                             }
                             case "LEDTV":
                             {
                                           System.out.println("Enter Batch Id");
                                           String batch = sc.nextLine();
                                           System.out.println("Enter Dispatch
date");
                                           String date = sc.nextLine();
                                           System.out.println("Enter Warranty
Years");
                                           int years = sc.nextInt();
                                           System.out.println(name);
                                           System.out.println("Enter size in
inches");
                                           int size = sc.nextInt();
                                           System.out.println("Enter quality");
                                           String quality = sc.nextLine();
                                           LEDTV ob2 = new LEDTV(id, name, batch,
date, years, size, quality);
                                           double price =
ob2.calculateProductPrice();
                                           System.out.printf("Price of the Product
is %.2f ", price);
                             }
                             case "MicrowaveOven":
                               {
                                           System.out.println("Enter Batch Id");
                                           String batch = sc.nextLine();
                                           System.out.println("Enter Dispatch
date");
                                           String date = sc.nextLine();
                                           System.out.println("Enter Warranty
Years");
                                           int years = sc.nextInt();
                                           System.out.println("Enter quantity");
                                           int quantity = sc.nextInt();
                                           System.out.println("Enter quality");
                                           String quality = sc.nextLine();
                                           MicrowaveOven ob3 = new
MicrowaveOven(id, name, batch, date, years,
quantity, quality);
                                           double price =
ob3.calculateProductPrice();
                                           System.out.printf("Price of the Product
is %.2f ", price);
                             }
                               default: {
                                           System.out.println("Provide a valid
Product name");
                             }
                             }

              }
}
```

MICROWAVE.JAVA

```
public class MicrowaveOven extends ElectronicProducts {
              private int quantity;
              private String quality;
              public int getQuantity() {
                             return quantity;
              }
              public void setQuantity(int quantity) {
                             this.quantity = quantity;
              }
              public String getQuality() {
                             return quality;
              }
              public void setQuality(String quality) {
                             this.quality = quality;
              }
              // Include Constructor
              public MicrowaveOven(String productId, String productName, String
batchId, String
dispatchDate, int warrantyYears,
                                           int quantity, String quality) {
                             super(productId, productName, batchId, dispatchDate,
warrantyYears);
                             this.quantity = quantity;
                             this.quality = quality;
              }
              public double calculateProductPrice() {
                             // Fill Code
                             double price = 0;
                             if (quality == "Low") {
                                           price = quantity * 1250;
                             } else if (quality == "Medium") {
                                           price = quantity * 1750;
                             } else if (quality == "High") {
                                           price = quantity * 2000;
                             }
                             return price;
              }
}
```

                 ELECTRONIC PRODUCT.JAVA-

```
public class ElectronicProducts {
              protected String productId;
              protected String productname;
              protected String batchId;
              protected String dispatchDate;
              protected int warrantyYears;
              public String getProductId() {
                             return productId;
              }
              public void setProductId(String productId) {
                             this.productId = productId;
              }
              public String getProductname() {
                             return productname;
              }
              public void setProductname(String productname) {
                             this.productname = productname;
              }
              public String getBatchId() {
```

```
                        return batchId;
            }
            public void setBatchId(String batchId) {
                        this.batchId = batchId;
            }
            public String getDispatchDate() {
                        return dispatchDate;
            }
            public void setDispatchDate(String dispatchDate) {
                        this.dispatchDate = dispatchDate;
            }
            public int getWarrantyYears() {
                        return warrantyYears;
            }
            public void setWarrantyYears(int warrantyYears) {
                        this.warrantyYears = warrantyYears;
            }
            public ElectronicProducts(String productId, String productname,
String batchId, String
dispatchDate,
                                    int warrantyYears) {
                        this.productId = productId;
                        this.productname = productname;
                        this.batchId = batchId;
                        this.dispatchDate = dispatchDate;
                        this.warrantyYears = warrantyYears;
            }
}
```

AIR CONDITIONER .JAVA-

```
public class AirConditioner extends ElectronicProducts {
            private String airConditionerType;
            private double capacity;
            public String getAirConditionerType() {
                        return airConditionerType;
            }
            public void setAirConditionerType(String airConditionerType) {
                        this.airConditionerType = airConditionerType;
            }
            public double getCapacity() {
                        return capacity;
            }
            public void setCapacity(double capacity) {
                        this.capacity = capacity;
            }
            // Include Constructor
            public AirConditioner(String productId, String productName, String
batchId, String
dispatchDate, int warrantyYears,
                                    String airConditionerType, double
capacity) {
                        super(productId, productName, batchId, dispatchDate,
warrantyYears);
                        this.airConditionerType = airConditionerType;
                        this.capacity = capacity;
            }
            public double calculateProductPrice() {
```

```
                              // Fill Code
                              double cost = 0;
                              if (airConditionerType == "Residential") {
                                            if (capacity == 2.5) {
                                                          cost = 32000;
                                            } else if (capacity == 4) {
                                                          cost = 40000;
                                            } else if (capacity == 5.5) {
                                                          cost = 47000;
                                            }
                              } else if (airConditionerType == "Commercial") {
                                            if (capacity == 2.5) {
                                                          cost = 40000;
                                            } else if (capacity == 4) {
                                                          cost = 55000;
                                            } else if (capacity == 5.5) {
                                                          cost = 67000;
                                            }
                              } else if (airConditionerType == "Industrial") {
                                            if (capacity == 2.5) {
                                                          cost = 47000;
                                            } else if (capacity == 4) {
                                                          cost = 60000;
                                            } else if (capacity == 5.5) {
                                                          cost = 70000;
                                            }
                              }
                              return cost;
              }
}
```

LED TV.JAVA

```
public class LEDTV extends ElectronicProducts {
              private int size;
              private String quality;
              public int getSize() {
                              return size;
              }
              public void setSize(int size) {
                              this.size = size;
              }
              public String getQuality() {
                              return quality;
              }
              public void setQuality(String quality) {
                              this.quality = quality;
              }
              // Include Constructor
              public LEDTV(String productId, String productName, String batchId,
String dispatchDate, int
warrantyYears, int size,
                                            String quality) {
                              super(productId, productName, batchId, dispatchDate,
warrantyYears);
                              this.size = size;
                              this.quality = quality;
              }
              public double calculateProductPrice() {
                              // Fill Code
                              double price = 0;
                              if (quality == "Low") {
                                            price = size * 850;
                              } else if (quality == "Medium") {
                                            price = size * 1250;
                              } else if (quality == "High") {
                                            price = size * 1550;
                              }
                              return price;
              }
}
```

CINEMA

BOOK A MOVIE TICKET-

```
public class BookAMovieTicket {
 protected String ticketId;
 protected String customerName;
 protected long mobileNumber;
 protected String emailId;
 protected String movieName;
 public void setticketId( String ticketId){
 this.ticketId=ticketId;
 }
 public void setcustomerName( String customerName){
 this.customerName=customerName;
 }
 public void setmobileNumber( long mobileNumber){
 this.mobileNumber=mobileNumber;
 }
 public void setemailId( String emailId){
 this.emailId=emailId;
 }
 public void setmovieName( String movieName){
 this.movieName=movieName;
 }
 public String getticketId(){
 return ticketId;
 }
 public String getcustomerName(){
 return customerName;
 }
 public String getemailId(){
 return emailId;
 }
 public String getmovieName(){
 return movieName;
 }
 public long getmobileNumber(){
 return mobileNumber;
 }
 public BookAMovieTicket(String ticketId,String customerName,long
mobileNumber,String emailId,String movieName){
```

```
this.ticketId=ticketId;
this.customerName=customerName;
this.mobileNumber=mobileNumber;
this.emailId=emailId;
this.movieName=movieName;
}

}
```

PLATINUM TICKET-

```
public class PlatinumTicket extends BookAMovieTicket {
public PlatinumTicket(String ticketId, String customerName, long mobileNumber,
String emailId, String movieName) {
super(ticketId, customerName, mobileNumber, emailId, movieName);
}
public boolean validateTicketId(){
int count=0;
if(ticketId.contains("PLATINUM"));
count++;
char[] cha=ticketId.toCharArray();
for(int i=8;i<11;i++){
if(cha[i]>='1'&& cha[i]<='9')
count++;
}
if(count==4)
return true;
else
return false;
}
public double caculateTicketCost(int numberOfTickets,String ACFacility){
double amount;
if(ACFacility.equals("yes")){
amount=750*numberOfTickets;
}
else{
amount=600*numberOfTickets;
}
return amount;
}
}
```

GOLD TICKET-

```
public class GoldTicket extends BookAMovieTicket {
public GoldTicket(String ticketId, String customerName, long mobileNumber,
String emailId, String movieName) {
super(ticketId, customerName, mobileNumber, emailId, movieName);
}
public boolean validateTicketId(){
int count=0;
if(ticketId.contains("GOLD"));
count++;
char[] cha=ticketId.toCharArray();
for(int i=4;i<7;i++){
if(cha[i]>='1'&& cha[i]<='9')
count++;
}
if(count==4)
return true;
else
return false;
}
public double caculateTicketCost(int numberOfTickets,String ACFacility){
double amount;
if(ACFacility.equals("yes")){
amount=500*numberOfTickets;
}
else{
amount=350*numberOfTickets;
}
return amount;
}
}
```

SILVER TICKET-

```
public class SilverTicket extends BookAMovieTicket{
public SilverTicket(String ticketId, String customerName, long mobileNumber,
String emailId, String movieName) {
super(ticketId, customerName, mobileNumber, emailId, movieName);
}
public boolean validateTicketId(){
int count=0;
if(ticketId.contains("SILVER"));
count++;
char[] cha=ticketId.toCharArray();
for(int i=6;i<9;i++){
if(cha[i]>='1'&& cha[i]<='9')
count++;
}
if(count==4)
return true;
else
return false;
}
public double caculateTicketCost(int numberOfTickets,String ACFacility){
double amount;
if(ACFacility.equals("yes")){
amount=250*numberOfTickets;
}
else{
amount=100*numberOfTickets;
}
return amount;
}
}
```

USER INTERFACE-

```
import java.util.*;
public class UserInterface {
public static void main(String[] args) {
Scanner sc=new Scanner(System.in);
System.out.println("Enter Ticket Id");
String tid=sc.next();
System.out.println("Enter Customer Name");
String cnm=sc.next();
System.out.println("Enter Mobile Number");
long mno=sc.nextLong();
System.out.println("Enter Email id");
String email=sc.next();
System.out.println("Enter Movie Name");
String mnm=sc.next();
System.out.println("Enter number of tickets");
int tno=sc.nextInt();
System.out.println("Do you want AC or not");
String choice =sc.next();
if(tid.contains("PLATINUM")){
PlatinumTicket PT=new PlatinumTicket(tid,cnm,mno,email,mnm);
boolean b1=PT.validateTicketId();
if(b1==true){
double cost =PT.caculateTicketCost(tno, choice);
System.out.println("Ticket cost is "+ cost);
}
else if(b1==false){
System.out.println("Provide valid Ticket Id");
System.exit(0);
}
}
else if(tid.contains("GOLD")){
GoldTicket GT=new GoldTicket(tid,cnm,mno,email,mnm);
boolean b2=GT.validateTicketId();
if(b2==true){
double cost=GT.caculateTicketCost(tno, choice);
System.out.println("Ticket cost is "+cost);
}
else if (b2==false){
System.out.println("Provide valid Ticket Id");
System.exit(0);
}
}
else if(tid.contains("SILVER")){
SilverTicket ST=new SilverTicket(tid,cnm,mno,email,mnm);
boolean b3=ST.validateTicketId();
if(b3==true){
double cost=ST.caculateTicketCost(tno, choice);
System.out.println("Ticket cost is "+cost);
}
else if(b3==false){
System.out.println("Provide valid Ticket Id");
System.exit(0);
}
}
}
}
```