

CLUB MEMBER DETAILS

ClubMember.java*

```
public class ClubMember{  
    private int memberId;  
    private String memberName;  
    private String memberType;  
    private double membershipFees;  
  
    public void setMemberId(int memberId){  
        this.memberId = memberId;  
    }  
  
    public int getMemberId(){  
        return memberId;  
    }  
  
    public void setMemberName(String memberName){  
        this.memberName = memberName;  
    }  
  
    public String getMemberName(){  
        return memberName;  
    }  
  
    public void setMemberType(String memberType){  
        this.memberType = memberType;  
    }  
  
    public String getMemberType(){  
        return memberType;  
    }  
  
    public void setMembershipFees(double membershipFees){
```

```

        this.membershipFees = membershipFees;
    }

    public double getMembershipFees(){
        return membershipFees;
    }

    public ClubMember(int memberId, String memberName, String memberType){
        this.memberId = memberId;
        this.memberName = memberName;
        this.memberType = memberType;
    }

    public void calculateMembershipFees (){
        if (memberType.equals("Gold")) membershipFees = 50000.0;
        else if (memberType.equals("Premium")) membershipFees = 75000.0;
    }

}

```

Main.java*

```

import java.util.Scanner;

public class Main{

    public static void main (String[] args) {
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter Member Id");
        int memberId = sc.nextInt();
        sc.nextLine();
        System.out.println("Enter Name");
        String memberName = sc.nextLine();
    }
}

```

```
System.out.println("Enter Member Type");

String memberType = sc.next();


ClubMember clubMemberObj = new ClubMember(memberId,memberName,memberType);
clubMemberObj.calculateMembershipFees();


System.out.println("Member Id is " + clubMemberObj.getMemberId());
System.out.println("Member Name is " + clubMemberObj.getMemberName());
System.out.println("Member Type is " + clubMemberObj.getMemberType());
System.out.println("Membership Fees is " + clubMemberObj.getMembershipFees());
}
}
```

CreditCardValidator

CreditCard.java*

```
package com.cts.entity;

public class CreditCard {
    private String number;

    public CreditCard() {

    }

    public CreditCard(String number) {
        super();
        this.number = number;
    }

    public String getNumber() {
        return number;
    }

    public void setNumber(String number) {
        this.number = number;
    }

}
```

CreditCardService.java*

```
package com.cts.services;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
```

```
import java.util.List;

import java.nio.file.*;
```

```
import com.cts.entity.CreditCard;
```

```
public class CreditCardService {

    //check whether the card is blocklisted and card contains only 16 digits
    public String validate(CreditCard card,String fileName) throws IOException
    {

        String msg=null;
        if(validateAgainstBlocklist(card, fileName))
        {
            msg="Card is blocked";
        }
        else if(validateNumber(card.getNumber()))
        {
            msg="card is not having 16 digits";
        }
        else
        {
            msg="valid card";
        }
        return msg;
    }

    // Validate a credit card against a blocklist.
    public boolean validateAgainstBlocklist(CreditCard card, String fileName) throws IOException {

        //write your code here

        boolean bol = true;

        String str = "";

        str = new String(Files.readAllBytes(Paths.get(fileName)));

        String dig[] = str.split(",");

        String str2 = dig[0];

        String str3 = dig[1];

        if(card.getNumber().equalsIgnoreCase(str2) || card.getNumber().equalsIgnoreCase(str3))
```

```

        {
            bol=true;
        }
        else{
            bol=false;
        }

        return bol;
    }

// Validate the card number length
public boolean validateNumber(String number) {
    int len = number.length();
    boolean bol=true;
    if(len!=16)
    {
        bol=true;
    }
    else{
        bol=false;
    }

    return bol;
}

// Get the blocklisted no's from the file and return list of numbers
public List<String> getBlockListNumbers(String fileName) throws IOException {

    List<String> li = new ArrayList<String>();
    String data = "";
    data = new String(Files.readAllBytes(Paths.get(fileName)));
    String dig1[] = data.split(",");
    for(int i=0;i<dig1.length;i++)
    {
        li.add(dig1[i]);
    }
}

```

```

    }

    return li;
}
}

```

SkeletonValidator.java*

```

package com.cts.skeletonvalidator;

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

public class SkeletonValidator {

    public SkeletonValidator() {
        validateClassName("com.cts.entity.CreditCard");
        validateClassName("com.cts.services.CreditCardService");
        validateMethodSignature(
            "validate:String,validateAgainstBlocklist:boolean,validateNumber:boolean,getBlockListNumbers:List","com.c
ts.services.CreditCardService");
    }

    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 incorrect;
}
```

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

        String[] actualmethods = methodWithExcpn.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().getSimpleName().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");
```

```
}
```

```
}
```

```
}
```

CreditCardValidatorMain.java*

```
package com.cts;
```

```
import java.io.BufferedReader;
```

```
import java.io.IOException;
```

```
import java.io.InputStreamReader;
```

```
import com.cts.entity.CreditCard;

import com.cts.services.CreditCardService;

import com.cts.skeletonvalidator.SkeletonValidator;


public class CreditCardValidatorMain {

    public static void main(String[] args) throws IOException {

        // CODE SKELETON - VALIDATION STARTS

        // DO NOT CHANGE THIS CODE

        BufferedReader b = new BufferedReader(new InputStreamReader(System.in));


        new SkeletonValidator();


        // CODE SKELETON - VALIDATION ENDS


        // Please start your code from here


        String cardNumber = b.readLine();

        CreditCard creditCard = new CreditCard();

        creditCard.setNumber(cardNumber);

        //Write your code here read card numnber and create CreditCard object based on cardnumber

        CreditCardService creditCardService = new CreditCardService();


        String validationMessage=creditCardService.validate(creditCard, "resources/blacklist.csv");

        System.out.println(validationMessage);

    }

}
```

ESHOPPING

Main.java*

```
package com.cts.eshopping.main;

import com.cts.eshopping.orderservice.CartService;
import com.cts.eshopping.skeletonvalidator.SkeletonValidator;
import com.cts.eshopping.vo.OrderLineItem;

public class Main {

    public static void main(String ag[]) {

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

        // Please start your code from here

        OrderLineItem it1 = new OrderLineItem("AM33","Book",200,3);
        OrderLineItem it2 = new OrderLineItem("AM345","Watch",1000,2);
        CartService cs  = new CartService();

        OrderLineItem[] arr = {it1, it2};
        double amt = cs.calculateOrderTotalAmount(arr);
        System.out.println(cs.calculateDiscount(amt));

    }

}
```

CartService.java*/orderService

```
package com.cts.eshopping.orderservice;
```

```
import com.cts.eshopping.vo.OrderLineItem;
```

```
/**
```

```
*
```

```
*/
```

```
public class CartService {
```

```
    /**
```

```
    * Method to calculate total purchase amount for all the order line items
```

```
    *
```

```
    * @param orderLineItems
```

```
    * @return totalOrderAmount
```

```
    */
```

```
    public double calculateOrderTotalAmount(OrderLineItem[] orderLineItems) {
```

```
        double totalOrderAmount = 0;
```

```
        int qt = 0;
```

```
        double cost = 0.0;
```

```
        for(int i=0;i<orderLineItems.length;i++){
```

```
            qt = orderLineItems[i].quantity;
```

```
            cost = orderLineItems[i].itemCostPerQuantity;
```

```
            totalOrderAmount += (qt*cost);
```

```
        }
```

```
        return totalOrderAmount; // TODO change this return value
```

```
    }
```

```
    /**
```

```
    * Method to calculate discount based on order total amount
```

```
    *
```

```
    * @param totalOrderAmount
```

```

    * @return discount
    */

    public double calculateDiscount(double totalOrderAmount) {
double discount = 0.0;

if(totalOrderAmount<1000){
    discount = (totalOrderAmount*10)/100;
}
else if(totalOrderAmount>=1000 && totalOrderAmount<10000){
    discount = (totalOrderAmount*20)/100;
}
else if(totalOrderAmount>=10000){
    discount = (totalOrderAmount*30)/100;
}

        return discount; // TODO change this return value
    }

    /**
     * Method to verify if the order line item is flagged as Bulk Order or not
     *
     * @param lineItem
     * @return boolean
     */
    public boolean isBulkOrder(OrderLineItem lineItem) {
boolean result=false;

if(lineItem.quantity>5){
    result = true;
}
else if(lineItem.quantity<=5 && lineItem.quantity>=1){
    result=false;
}

        return result; // TODO change this return value
    }

```

```

/**
 * Count the number of line items which are ordered in bulk
 *
 * @param orderLineItems
 * @return
 */
public int countOfBulkOrderLineItems(OrderLineItem[] orderLineItems) {
    int count = 0;

    for(int i=0;i<orderLineItems.length;i++){
        if(isBulkOrder(orderLineItems[i])){
            count++;
        }
    }

    return count; // TODO change this return value
}
}

```

SkeletonValidator.java*

```

package com.cts.eshopping.skeletonvalidator;

```

```

import java.lang.reflect.Method;

```

```

import java.util.logging.Level;

```

```

import java.util.logging.Logger;

```

```

/**
 * @author 222805
 *
 * 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("com.cts.eshopping.orderservice.CartService");
        validateClassName("com.cts.eshopping.vo.OrderLineItem");
        validateMethodSignature(
            "calculateOrderTotalAmount:double,calculateDiscount:double,isBulkOrder:boolean,countOfBulkOrderLineItems:int",
            "com.cts.eshopping.orderservice.CartService");
    }

    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 methodWithExcpn, String className) {

    Class cls = null;

    try {

        String[] actualmethods = methodWithExcpn.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");
}

}

}

```

OrderLineItem.java*

```

package com.cts.eshopping.vo;

/**
 * @author Value Object - OrderLineItem
 *
 */
public class OrderLineItem {

    public String itemId;
    public String itemName;
    public double itemCostPerQuantity;
    public int quantity;

    public String getItemId(){
        return itemId;
    }
}

```

```
public void setItemId(String itemId){
    this.itemId = itemId;
}

public String getItemName(){
    return itemName;
}

public void setItemName(String itemName){
    this.itemName = itemName;
}

public double getItemCostPerQuantity(){
    return itemCostPerQuantity;
}

public void setItemCostPerQuantity(double itemCostPerQuantity){
    this.itemCostPerQuantity = itemCostPerQuantity;
}

public int getQuantity(){
    return quantity;
}

public void setItemQuantity(int quantity){
    this.quantity = quantity;
}

public OrderLineItem(String itemId, String itemName, double itemCostPerQuantity, int quantity){
    this.itemId = itemId;
    this.itemName = itemName;
    this.itemCostPerQuantity=itemCostPerQuantity;
    this.quantity = quantity;
}
}
```

Fixed Deposit Details

FDScheme.java*

```
import java.util.*;

class FDScheme{

    private int schemeNo;

    private double depositAmt;

    private int period;

    private float rate;

    public FDScheme(int schemeNo, double depositAmt, int period){

        super();

        this.schemeNo=schemeNo;

        this.depositAmt=depositAmt;

        this.period=period;

        calculateInterestRate();

    }

    public int getSchemeNo(){

        return schemeNo;

    }

    public void setSchemeNo(int schemeNo)

    {

        this.schemeNo=schemeNo;

    }

    public double getDepositAmt(){

        return depositAmt;

    }

    public void setDepositAmt(double depositAmt)

    {

        this.depositAmt=depositAmt;

    }

    public int getPeriod()

    {

        return period;

    }

    public void setPeriod(int period){
```

```

        this.period=period;
    }
    public float getRate(){
        return rate;
    }
    public void setRate(float rate){
        this.rate=rate;
    }
    public void calculateInterestRate()
    {
        if(period>=1 && period<=90)
        {
            this.rate=(float)5.5;
        }
        else if(period>=91 && period<=180)
        {
            this.rate=(float)6.25;
        }
        else if(period>=181 && period<=365)
        {
            this.rate=(float)7.5;
        }
        System.out.println("Interest rate for"+period+"days is"+this.rate);
    }
}

```

Main.java*

```

import java.util.Scanner;

public class Main{

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

```

```
System.out.println("Enter Scheme no");  
int no=sc.nextInt();  
sc.nextLine();  
System.out.println("Enter Deposit amount");  
double amt=sc.nextDouble();  
System.out.println("enter period of deposit");  
int prd=sc.nextInt();  
FDScheme obj=new  
FDScheme(no,amt,prd);  
}  
}
```

GPA CALCULATION

UserInterface.java*

```
package com.ui;
```

```
import com.utility.*;
```

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

```
public class UserInterface {
```

```
public static void main(String []args)
```

 $\{$

```
GPA Calculator gpa = new GPA Calculator();
```

```
gpa.setGradePointList(new ArrayList<Integer>());
```

```
int option=0;
```

```
double gpa1=0;
```

```
Scanner sc = new Scanner(System.in);
```

do

 $\{$

```
System.out.println("1. Add Grade\n2. Calculate GPA\n3. Exit");
```

```
System.out.println("Enter your choice");
```

```
option = Integer.valueOf(sc.nextLine());
```

```
switch(option)
```

 $\{$

```
case 1: System.out.println("Enter the obtained grade");
```

```
char grade = sc.nextLine().charAt(0);
```

```
gpa.addGradePoint(grade);
```

```
break;
```

```
case 2 : gpa1 = gpa.calculateGPAScored();
```

```
if(gpa1 > 0)
```

 $\{$

```
System.out.println("GPA Scored");
```

```
System.out.println(gpa1);
```

}

else

 $\{$

```
System.out.println("No GradePoints available");
```

```

        }

        break;

        case 3 : break;

    }

    }while(option!=3);

    System.out.println("Thank you for using the Application");

}

}

```

GPACalculator.java*

```
package com.utility;
```

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

```
public class GPACalculator {
```

```
    private List<Integer> gradePointList;
```

```
    public List<Integer> getGradePointList() {
        return gradePointList;
    }

```

```
    public void setGradePointList(List<Integer> gradePointList) {
        this.gradePointList = gradePointList;
    }

```

/*This method should add equivalent grade points based on the grade obtained by the student passed as argument into gradePointList

Grade	S	A	B	C	D	E
Grade Point	10	9	8	7	6	5

For example if the grade obtained is A, its equivalent grade points is 9 has to added into the gradePointList*/

```
public void addGradePoint(char gradeObtained) {
```

```
    if(gradeObtained == 'S')
    {
        gradePointList.add(10);
    }
    else if(gradeObtained == 'A')
    {
        gradePointList.add(9);
    }
    else if(gradeObtained == 'B')
    {
        gradePointList.add(8);
    }
    else if(gradeObtained == 'C')
    {
        gradePointList.add(7);
    }
    else if(gradeObtained == 'D')
    {
        gradePointList.add(6);
    }
    else
    {
        gradePointList.add(5);
    }
}
```

/* This method should return the GPA of all grades scored in the semester

GPA can be calculated based on the following formula

$$\text{GPA} = (\text{gradePoint1} + \text{gradePoint2} + \dots + \text{gradePointN}) / (\text{size of List})$$

For Example:

if the list contains the following marks [9,10,8,5]

$$\text{GPA} = (9 + 10 + 8 + 5) / (4) = 8.0 \text{ */}$$


```
public double calculateGPAScore() {  
  
    double gpa=-1;  
    double total=0,value=0,size=0;  
  
    size = gradePointList.size();  
    if(size < 1)  
    {  
        return 0;  
    }  
    // fill the code  
    Iterator i = gradePointList.iterator();  
    while(i.hasNext())  
    {  
        value = (Integer)i.next();  
        total += value;  
    }  
    gpa = total/size;  
  
    return gpa;  
}  
}
```

HUNGER EATS

FoodProduct.java*

```
package com.bean;
```

```
public class FoodProduct {

    private int foodId;
    private String foodName;
    private double costPerUnit;
    private int quantity;

    public int getFoodId() {
        return foodId;
    }
    public void setFoodId(int foodId) {
        this.foodId = foodId;
    }
    public String getFoodName() {
        return foodName;
    }
    public void setFoodName(String foodName) {
        this.foodName = foodName;
    }
    public double getCostPerUnit() {
        return costPerUnit;
    }
    public void setCostPerUnit(double costPerUnit) {
        this.costPerUnit = costPerUnit;
    }
    public int getQuantity() {
        return quantity;
    }
    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }
}
```

```
}
```

```
}
```

UserInterface.java*

```
package com.ui;

import java.util.Scanner;
import com.utility.Order;
import com.bean.FoodProduct;

public class UserInterface {

    public static void main(String[] args) {

        // fill the code

        Scanner sc=new Scanner(System.in);
        int itemno;
        String bank;
        System.out.println("Enter the number of items");
        itemno=sc.nextInt();
        System.out.println("Enter the item details");
        Order z=new Order();
        for(int i=0;i<itemno;i++){
            FoodProduct fd=new FoodProduct();
            System.out.println("Enter the item id");
            fd.setFoodId(sc.nextInt());
            System.out.println("Enter the item name");
            fd.setFoodName(sc.next());
            System.out.println("Enter the cost per unit");
            fd.setCostPerUnit(sc.nextDouble());
            System.out.println("Enter the quantity");
            fd.setQuantity(sc.nextInt());
            z.addToCart(fd);
        }
    }
}
```

```

        System.out.println("Enter the bank name to avail offer");

        bank=sc.next();

        z.findDiscount(bank);

        System.out.println("Calculated Bill Amount:"+z.calculateTotalBill());

    }

}

```

Order.java*

```

package com.utility;

import java.util.*;
import com.bean.FoodProduct;

public class Order {

    private double discountPercentage;

    private List<FoodProduct> foodList=new ArrayList<FoodProduct>();

    public double getDiscountPercentage() {

        return discountPercentage;

    }

    public void setDiscountPercentage(double discountPercentage) {

        this.discountPercentage = discountPercentage;

    }

    public List<FoodProduct> getFoodList() {

        return foodList;

    }

    public void setFoodList(List<FoodProduct> foodList) {

        this.foodList = foodList;

    }

}

```

//This method should set the discount percentage based on bank passed as argument

```
public void findDiscount(String bankName) {
```

```
    // fill the code
```

```
    if(bankName.equals("HDFC")){
```

```
        discountPercentage=15.0;
```

```
    }
```

```
    else if(bankName.equals("ICICI")){
```

```
        discountPercentage=25.0;
```

```
    }
```

```
    else if(bankName.equals("CUB")){
```

```
        discountPercentage=30.0;
```

```
    }
```

```
    else if(bankName.equals("SBI")){
```

```
        discountPercentage=50.0;
```

```
    }
```

```
    else if(bankName.equals("OTHERS")){
```

```
        discountPercentage=0.0;
```

```
    }
```

```
}
```

//This method should add the FoodProduct Object into Food List

```
public void addToCart(FoodProduct foodProductObject) {
```

```
    // fill the code
```

```
    List<FoodProduct> f=getFoodList();
```

```
    f.add(foodProductObject);
```

```
    setFoodList(f);
```

```
}
```

//method should return the total bill amount after discount

```
// based on the bank name
```

```
public double calculateTotalBill() {
```

```
    // fill the code
```

```
    double bill=0;
```

```
    List<FoodProduct> f=getFoodList();
```

```
    for(int i=0;i<f.size();i++){
```

```
        //
```

```
        // System.out.println(f.get(i).getCostPerUnit());
```

```
        //
```

```
        // System.out.println(f.get(i).getQuantity());
```

```
        bill+=f.get(i).getQuantity()*f.get(i).getCostPerUnit()*1.0;
```

```
    }
```

```
    bill=bill-((bill*discountPercentage)/100);
```

```
    return bill;
```

```
}
```

```
}
```

INSURANCE PREMIUM GENERATOR

PropertyDetails.java*

```
package com.cts.insurance.entity;

public class PropertyDetails {

    private Integer builtUpArea;
    private Integer builtYear;
    private Integer reconstructionCost;
    private Integer householdValuation;
    private String burglaryCoverReqd;
    private String politicalUnrestCoverReqd;
    private Integer sumAssured;

    public PropertyDetails() {

    }

    public Integer getBuiltUpArea() {
        return builtUpArea;
    }

    public void setBuiltUpArea(Integer builtUpArea) {
        this.builtUpArea = builtUpArea;
    }

    public Integer getBuiltYear() {
        return builtYear;
    }

    public void setBuiltYear(Integer builtYear) {
        this.builtYear = builtYear;
    }

    public Integer getReconstructionCost() {
```

```
        return reconstructionCost;
    }

    public void setReconstructionCost(Integer reconstructionCost) {
        this.reconstructionCost = reconstructionCost;
    }

    public Integer getHouseholdValuation() {
        return householdValuation;
    }

    public void setHouseholdValuation(Integer householdValuation) {
        this.householdValuation = householdValuation;
    }

    public String getBurglaryCoverReqd() {
        return burglaryCoverReqd;
    }

    public void setBurglaryCoverReqd(String burglaryCoverReqd) {
        this.burglaryCoverReqd = burglaryCoverReqd;
    }

    public String getPoliticalUnrestCoverReqd() {
        return politicalUnrestCoverReqd;
    }

    public void setPoliticalUnrestCoverReqd(String politicalUnrestCoverReqd) {
        this.politicalUnrestCoverReqd = politicalUnrestCoverReqd;
    }

    public Integer getSumAssured() {
        return sumAssured;
    }
}
```



```

public void setSumAssured(Integer sumAssured) {
    this.sumAssured = sumAssured;
}

```

```

    public PropertyDetails(Integer builtUpArea,Integer builtYear, Integer reconstructionCost, Integer
householdValuation,
        String burglaryCoverReqd, String politicalUnrestCoverReqd) {
        super();
        this.builtUpArea = builtUpArea;
        this.builtYear=builtYear;
        this.reconstructionCost = reconstructionCost;
        this.householdValuation = householdValuation;
        this.burglaryCoverReqd = burglaryCoverReqd;
        this.politicalUnrestCoverReqd = politicalUnrestCoverReqd;
    }
}

```

Constants.java*

```

package com.cts.insurance.misc;

```

```

public class Constants {
    public final static String YES = "Yes";
    public final static String NO = "No";
    public final static double MIN_PREMIUM_AMOUNT = 5000;
    public final static int MIN_HOUSEHOLD_VALUATION=0;
}

```

CalculatePremiumService.java*

```

package com.cts.insurance.services;

```

```
import com.cts.insurance.entity.PropertyDetails;
```

```
import com.cts.insurance.misc.Constants;
```

```
import java.time.LocalDate;
```

```
public class CalculatePremiumService {
```

```
    public boolean checkOwnerDetails(String name,String mobile) {
```

```
        //name cannot have numbers or special characters; minimum length of name=2
```

```
        //mobile number begins with any digit between 6 and 9; length=10
```

```
        return name.matches("[a-zA-Z ]{2,}$") && mobile.matches("[6-9][0-9]{9}$");
```

```
    }
```

```
    public double getPremiumAmount(PropertyDetails propertyDetails) {
```

```
        double amountToBePaid = 0;
```

```
        double additionalAmount1=0;
```

```
        double additionalAmount2=0;
```

```
        /*invoke validatePropertyParameters(propertDetails) and check the response
```

```
        * if true ,calculate premium amount to be paid by calling
```

```
        * the methods calculatePremiumByPropertyAge(propertyDetails),
```

```
        * calculatePremiumForBurglaryCoverage(propertyDetails, amountToBePaid) and
```

```
        * calculatePremiumForPoliticalUnrestCoverage(propertyDetails, amountToBePaid)
```

```
        *
```

```
        * return the premium amount rounded off to zero decimal places
```

```
        * else return 0;
```

```
        */
```

```
        if(!validatePropertyParameters(propertyDetails)) {
```

```
            return 0;
```

```
        }
```

```
        amountToBePaid=calculatePremiumByPropertyAge(propertyDetails);
```

```
        additionalAmount1=calculatePremiumForBurglaryCoverage(propertyDetails, amountToBePaid);
```

```
        additionalAmount2=calculatePremiumForPoliticalUnrestCoverage(propertyDetails,  
amountToBePaid);
```

```
        return Math.round(amountToBePaid+additionalAmount1+additionalAmount2);
```

```

}

public boolean validatePropertyParams(PropertyDetails propertyDetails) {

    /*
     * conditions to be checked
     * builtUpArea between 400 and 15,000 sq. ft.
     * reconstructionCost between Rs.1,000 and Rs.10,000
     * householdValuation either same as Constants.MIN_HOUSEHOLD_VALUATION
     * between Rs.1,00,000 and Rs.15,00,000
     * builtYear between 2000 and current year
     */

    int builtUpArea = propertyDetails.getBuiltUpArea();
    if(!(builtUpArea>=400 && builtUpArea<=15000)) return false;
    int reconstructionCost = propertyDetails.getReconstructionCost();
    if(!(reconstructionCost>=1000 && reconstructionCost<=10000)) return false;
    int householdValuation = propertyDetails.getHouseholdValuation();
    if(!((householdValuation==Constants.MIN_HOUSEHOLD_VALUATION) || (householdValuation >=
100000 && householdValuation <= 1500000))) {
        return false;
    }
    int builtYear = propertyDetails.getBuiltYear();
    if(!(builtYear>=2000 && builtYear<=LocalDate.now().getYear())) {
        return false;
    }
    return true;
}

public double calculatePremiumByPropertyAge(PropertyDetails propertyDetails) {

    //Write your code here based on business rules

    //Use Constants.MIN_PREMIUM_AMOUNT

    int sumAssured =
propertyDetails.getBuiltUpArea()*propertyDetails.getReconstructionCost()+propertyDetails.getHouseholdValuation(
);

    int propertyAge = LocalDate.now().getYear()-propertyDetails.getBuiltYear();
    propertyDetails.setSumAssured(sumAssured);

    double premium = 0;

    if(propertyAge>15) {

```

```

        premium = Constants.MIN_PREMIUM_AMOUNT+(propertyDetails.getSumAssured()*0.35);
    }
    else if(propertyAge>=6) {
        premium = Constants.MIN_PREMIUM_AMOUNT+(propertyDetails.getSumAssured()*0.2);
    }
    else {
        premium = Constants.MIN_PREMIUM_AMOUNT+(propertyDetails.getSumAssured()*0.1);
    }
    return premium;
}

```

```

public double calculatePremiumForBurglaryCoverage(PropertyDetails propertyDetails, double amount) {
    //write your code here based on business rules
    if(propertyDetails.getBurglaryCoverReqd().equalsIgnoreCase(Constants.YES)) {
        return amount*.01;
    }
    return 0;
}

```

```

public double calculatePremiumForPoliticalUnrestCoverage(PropertyDetails propertyDetails, double amount) {
    //Write your code here based on business rules
    //Ex:-propertyDetails.getPoliticalUnrestCoverReqd().equalsIgnoreCase(Constants.YES) to check
condition
    if(propertyDetails.getPoliticalUnrestCoverReqd().equalsIgnoreCase(Constants.YES)) {
        return amount*.01;
    }
    return 0;
}
}

```

SkeletonValidator.java*

```
package com.cts.insurance.skeleton;
```

```
import java.lang.reflect.Method;
```

```
import java.util.logging.Level;
```

```
import java.util.logging.Logger;
```

```
/**
```

```
 * @author
```

```
 *
```

```
 * 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("com.cts.insurance.entity.PropertyDetails");
```

```
        validateClassName("com.cts.insurance.misc.Constants");
```

```
        validateClassName("com.cts.insurance.services.CalculatePremiumService");
```

```
        validateClassName("com.cts.insurance.InsurancePremiumGeneratorApp");
```

```
        validateMethodSignature(
```

```
            "checkOwnerDetails:boolean,getPremiumAmount:double,validatePropertyParameters:boolean,calculatePremiumByPropertyAge:double,calculatePremiumForBurglaryCoverage:double,calculatePremiumForPoliticalUnrestCoverage:double","com.cts.insurance.services.CalculatePremiumService");
```

```
    }
```

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

```
    protected final boolean validateClassName(String className) {
```

```
        boolean incorrect = false;
```

```
        try {
```

```
            Class.forName(className);
```

```
            incorrect = 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 incorrect;

}

```

```

protected final void validateMethodSignature(String methodWithExcpn, String className) {

    Class cls = null;

    try {

        String[] actualmethods = methodWithExcpn.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().getSimpleName().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");

}

}

}

```

InsurancePremiumGeneratorApp.java*

```
package com.cts.insurance;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

import com.cts.insurance.entity.PropertyDetails;
import com.cts.insurance.misc.Constants;
import com.cts.insurance.services.CalculatePremiumService;
import com.cts.insurance.skeleton.SkeletonValidator;

public class InsurancePremiumGeneratorApp {

    public static void main(String[] args)throws IOException {

        // CODE SKELETON - VALIDATION STARTS
        // DO NOT CHANGE THIS CODE

        SkeletonValidator validator = new SkeletonValidator();

        // CODE SKELETON - VALIDATION ENDS

        // Please start your code from here
        String name = "";
        String mobile = "";
        Integer builtUpArea = 0;
        Integer builtYear=0;
        Integer reconstructionCost = 0;
        Integer householdValuation = Constants.MIN_HOUSEHOLD_VALUATION;
        String burglaryCoverReqd = "";
        String politicalUnrestCoverReqd = "";

        //writer the code for creating BufferedReader object here
```



```
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
```

```
CalculatePremiumService premiumService = new CalculatePremiumService();
```

```
System.out.println("Enter the name");
```

```
//read name
```

```
name = br.readLine();
```

```
System.out.println("Enter the mobile");
```

```
//read mobile
```

```
mobile = br.readLine();
```

```
//validate name and mobile format; continue if true
```

```
if(premiumService.checkOwnerDetails(name, mobile)) {
```

```
System.out.println("Enter the Built-Up Area(In sq.ft.)between 400 and 15,000");
```

```
//read builtUpArea
```

```
builtUpArea = Integer.parseInt(br.readLine());
```

```
System.out.println("Enter the year the house was built");
```

```
//read builtYear
```

```
builtYear = Integer.parseInt(br.readLine());
```

```
System.out.println("Enter the Re-construction cost(per sq.ft.)between 1,000 and 10,000");
```

```
//read reconstructionCost
```

```
reconstructionCost = Integer.parseInt(br.readLine());
```

```
System.out.println(
```

```
    "Do you want to include valuation of HouseHold Articles? Please provide yes/no");
```

```
//read response
```

```
String response = br.readLine();
```

```
//if (response is "yes" case insensitive)
```

```
if(response.equalsIgnoreCase("yes")) {
```

```
    System.out.println("Enter the Household valuation between Rs.1,00,000 and Rs.15,00,000");
```

```
    //read householdValuation
```

```
    householdValuation = Integer.parseInt(br.readLine());
```

```
}
```

```
System.out.println("Do you want to include Burglary cover? Please provide yes/no");
```

```

        //read burglaryCoverReqd
        burglaryCoverReqd = br.readLine();

        System.out.println("Do you want to include Political unrest cover? Please provide yes/no");

        //read politicalUnrestCoverReqd
        politicalUnrestCoverReqd = br.readLine();

        //create PropertyDetails Object
        PropertyDetails propertyDetails = new PropertyDetails(builtUpArea, builtYear, reconstructionCost,
        householdValuation, burglaryCoverReqd, politicalUnrestCoverReqd);

        double premiumAmount = premiumService.getPremiumAmount(propertyDetails);
        if(premiumAmount==0.0) {
            System.out.println("Incorrect figures provided");
        }else {
            System.out.println("Sum Insured: Rs."+propertyDetails.getSumAssured()+"\nInsurance
        Premium for the property of " + name + ": Rs." + premiumAmount);
        }
    }
    else {System.out.println("Invalid Details");}

}

}

```

NUMEROLOGY NUMBER

Main.java*

```
import java.util.Scanner;

public class Main {

    private static int getSum(long num) {
        char[] chars = Long.toString(num).toCharArray();
        int sum = 0;

        for (char ch : chars) {
            sum += Character.digit(ch, 10);
        }

        return sum;
    }

    private static int getNumerology(long num) {
        String string = String.valueOf(num);

        while (string.length() != 1) {
            string = String.valueOf(getSum(Long.parseLong(string)));
        }

        return Integer.parseInt(string);
    }

    private static int getOddCount(long num) {
        int oddCount = 0;

        for (char ch : Long.toString(num).toCharArray()) {
            if (Character.digit(ch, 10) % 2 != 0) {
                ++oddCount;
            }
        }
    }
}
```

```
        return oddCount;
    }

    private static int getEvenCount(long num) {
        int evenCount = 0;

        for (char ch : Long.toString(num).toCharArray()) {
            if (Character.digit(ch, 10) % 2 == 0) {
                ++evenCount;
            }
        }

        return evenCount;
    }

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

        System.out.println("Enter the number");
        long num = scanner.nextLong();

        System.out.println("Sum of digits");
        System.out.println(getSum(num));

        System.out.println("Numerology number");
        System.out.println(getNumerology(num));

        System.out.println("Number of odd numbers");
        System.out.println(getOddCount(num));

        System.out.println("Number of even numbers");
        System.out.println(getEvenCount(num));
    }
}
```

OIL STORES

Oil.java*

```
import java.util.Scanner;

public class Oil{

    //Fill the code here

    private String name;
    private int pack;
    private char category;
    private float cost;
    public Oil(String name,int pack,char category,float cost){
        this.name=name;
        this.pack=pack;
        this.category=category;
        this.cost=cost;
    }
    public void setName(String name){
        this.name=name;
    }
    public String getName(){
        return name;
    }
    public void setPack(int pack){
        this.pack=pack;
    }
    public int getPack(){
        return pack;
    }
    public void setCategory(char category){
        this.category=category;
    }
    public char getCategory(){
        return category;
    }
}
```

```

    }

    public void setCost(float cost){

        this.cost=cost;
    }

    public float getCost(){

        return cost;
    }

    public float calculateTotalCost(float qty){

        float price=((qty*1000)/pack)*cost;

        return price;
    }
}

```

Main.java*

```

import java.util.Scanner;

public class Main{

    public static void main (String[] args) {

        Scanner sc=new Scanner(System.in);

        //Fill the code

        System.out.println("Enter oil name");
        String n=sc.next();

        System.out.println("Enter pack capacity");
        int pc=sc.nextInt();

        System.out.println("Enter category");
        char cat=sc.next().charAt(0);

        System.out.println("Enter cost");
        float c=sc.nextFloat();

        Oil obj=new Oil(n,pc,cat,c);

        obj.setName(n);

        obj.setPack(pc);

        obj.setCategory(cat);
    }
}

```

```
obj.setCost(c);  
System.out.println("Enter Quantity to purchase");  
float qty=sc.nextFloat();  
System.out.println("Oil cost rs."+obj.calculateTotalCost(qty));  
}  
}
```

PAYMENT-INHERITENCE

Bill.java*

```
public class Bill {

    public String processPayment(Payment payObj) {
        String result="Payment not done and your due amount is "+payObj.getDueAmount();
        // Fill your code
        if(payObj instanceof Cheque)
        {
            Cheque cheque=(Cheque)payObj;
            if(cheque.payAmount())
                result="Payment done successfully via cheque";

        }
        else if(payObj instanceof Cash)
        {
            Cash cash=(Cash)payObj;
            if(cash.payAmount())
                result="Payment done successfully via cash";

        }
        else if(payObj instanceof Credit)
        {
            Credit credit=(Credit)payObj;
            if(credit.payAmount())
                result="Payment done successfully via credit card. Remaining amount in your "+credit.getCardType()+" card
is "+credit.getCreditCardAmount();

        }

        return result;
    }
}
```


Cash.java*

```
public class Cash extends Payment
{
    private int cashAmount;

    public int getCashAmount() {
        return cashAmount;
    }

    public void setCashAmount(int cashAmount) {
        this.cashAmount = cashAmount;
    }

    public boolean payAmount()
    {
        return getCashAmount() >= getDueAmount();
    }
}
```

Cheque.java

```
import java.util.*;
import java.util.GregorianCalendar;
import java.text.ParseException;
import java.util.Calendar;
import java.util.Date;
import java.text.SimpleDateFormat;

public class Cheque extends Payment
{
    private String chequeNo;
    private int chequeAmount;
    private Date dateOfIssue;

    public String getChequeNo() {
```

```

        return chequeNo;
    }

    public void setChequeNo(String chequeNo) {
        this.chequeNo = chequeNo;
    }

    public int getChequeAmount() {
        return chequeAmount;
    }

    public void setChequeAmount(int chequeAmount) {
        this.chequeAmount = chequeAmount;
    }

    public Date getDateOfIssue() {
        return dateOfIssue;
    }

    public void setDateOfIssue(Date dateOfIssue) {
        this.dateOfIssue = dateOfIssue;
    }

    private int findDifference(Date date){
        Calendar myDate=new GregorianCalendar();
        myDate.setTime(date);
        return (2020-myDate.get(Calendar.YEAR))*12+(0-myDate.get(Calendar.MONTH));
    }

    @Override
    public boolean payAmount()
    {
        int months=findDifference(getDateOfIssue());
        return (getChequeAmount()>=getDueAmount()&months<=6);
    }

    // Fill your code
    public void generateDate(String date)
    {
        try{
            Date issueDate=new SimpleDateFormat("dd-MM-yyyy").parse(date);

```

```

        setDateOfIssue(issueDate);
    }
    catch(ParseException e)
    {
        e.printStackTrace();
    }
}

}

```

Credit.java*

```

public class Credit extends Payment
{
    private int creditCardNo;

    private String cardType; //(silver,gold,platinum) String,
    private int creditCardAmount;

    public int getCreditCardNo() {
        return creditCardNo;
    }

    public void setCreditCardNo(int creditCardNo) {
        this.creditCardNo = creditCardNo;
    }

    public String getCardType() {
        return cardType;
    }

    public void setCardType(String cardType) {
        this.cardType = cardType;
    }

    public int getCreditCardAmount() {
        return creditCardAmount;
    }

    public void setCreditCardAmount(int creditCardAmount) {

```

```
        this.creditCardAmount = creditCardAmount;
    }
}
```

```
public Credit(int creditCardNo, String cardType) {
    super();
}
```

```
// Fill your code
```

```
    }
    public Credit()
    {
    }
}
```

```
// Fill your code
```

```
@Override
```

```
public boolean payAmount()
{
    int tax=0;
    boolean isDeducted=false;
    switch(cardType)
    {
        case "silver": setCreditCardAmount(10000);
            tax=(int)(0.02*getDueAmount()+getDueAmount());
            if(tax<=getCreditCardAmount())
            {
                setCreditCardAmount(getCreditCardAmount()-tax);
                isDeducted=true;
            }
            break;
        case "gold": setCreditCardAmount(50000);
            tax=(int)(0.05*getDueAmount()+getDueAmount());
            if(tax<=getCreditCardAmount())
```

```

        {
            setCreditCardAmount(getCreditCardAmount()-tax);
            isDeducted=true;
        }
        break;
    case "platinum": setCreditCardAmount(100000);
        tax=(int)(0.1*getDueAmount()+getDueAmount());
        if(tax<=getCreditCardAmount())
        {
            setCreditCardAmount(getCreditCardAmount()-tax);
            isDeducted=true;
        }
        break;
    }
    return isDeducted;
}
}

```

Payment.java*

```

public class Payment {
    private int dueAmount;
    public int getDueAmount() {
        return dueAmount;
    }
    public void setDueAmount(int dueamount) {
        this.dueAmount = dueamount;
    }
    public boolean payAmount() {

        return false;
    }
}

```

Main.java*

```
import java.util.*;

public class Main {

    public static void main(String[] args) {

        // Fill your code

        Bill bill=new Bill();

        Scanner sc=new Scanner(System.in);

        System.out.println("Enter the due amount;");

        int dueAmount=sc.nextInt();

        System.out.println("Enter the mode of payment(cheque/cash/credit:");

        String mode=sc.next();

        switch(mode)

        {

            case "cash" : System.out.println("Enter the cash amount:");

                int cashAmount=sc.nextInt();

                Cash cash=new Cash();

                cash.setCashAmount(cashAmount);

                cash.setDueAmount(dueAmount);

                System.out.println(bill.processPayment(cash));

                break;

            case "cheque" : System.out.println("Enter the cheque number:");

                String number=sc.next();

                System.out.println("Enter the cheque amount:");

                int chequeAmount=sc.nextInt();

                System.out.println("Enter the date of issue:");

                String date=sc.next();

                Cheque cheque=new Cheque();

                cheque.setChequeAmount(chequeAmount);

                cheque.setChequeNo(number);

                cheque.generateDate(date);

                cheque.setDueAmount(dueAmount);

                System.out.println(bill.processPayment(cheque));
```

```
        break;
    case "credit" : System.out.println("Enter the credit card number:");

        int creditNumber = sc.nextInt();

        System.out.println("Enter the card type(silver,gold,platinum):");

        String cardType=sc.next();

        Credit credit=new Credit();

        credit.setCardType(cardType);

        credit.setCreditCardNo(creditNumber);

        credit.setDueAmount(dueAmount);

        System.out.println(bill.processPayment(credit));

        break;

    default:

        break;

}

        sc.close();

}

}
```

POWER PROGRESS

Main.java*

```
import java.util.*;

public class Main
{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);
        //Fill the code
        int m=sc.nextInt();
        if(m<=0){
            System.out.println(""+m+" is an invalid");
            return;
        }
        int n=sc.nextInt();
        if(n<=0){
            System.out.println(""+n+" is an invalid");
            return;
        }
        if(m>=n){
            System.out.println(""+m+" is not less than "+n);
            return;
        }

        for(int i=1;i<=n;i++){
            System.out.print((int)Math.pow(m,i)+" ");
        }

    }
}
```


PRIME NUMBERS ENDING WITH 1

Main.java*

```
import java.util.Scanner;

public class Main {

    public static void main(String args[]) {

        Scanner sc = new Scanner(System.in);

        // fill the code

        int low, high;

        int last=0;

        int flag = 0;

        System.out.println("Enter the first number");

        low = sc.nextInt();

        System.out.println("Enter the last number");

        high = sc.nextInt();

        if (low > high || low < 0 || high < 0 || low == high)

            ;

        else {

            int i = low;

            high = high + 30;

            while (i <= high-1) {

                int x = i % 10;

                for (int j = 2; j <= i / 2; j++) {

                    if (i % j != 0 && x == 1) {

                        flag = 1;

                    } else {

                        flag = 0;

                        break;

                    }

                }

                if (flag == 1 )

                    System.out.println(i);

                i++;

            }

        }

    }

}
```

SINGAPORE TOURISM

Main.java*

```
import java.util.*;

public class Main
{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);
        Map<String,Integer> map=new HashMap<>();
        map.put("BEACH",270);
        map.put("PILGRIMAGE",350);
        map.put("HERITAGE",430);
        map.put("HILLS",780);
        map.put("FALLS",1200);
        map.put("ADVENTURES",4500);
        System.out.println("Enter the Passenger Name");
        String pname=sc.next();
        System.out.println("Enter the Place");
        String name=sc.next();
        if(!map.containsKey(name.toUpperCase()))
        {
            System.out.println(name+" is an invalid place");
        }
        else
        {
            System.out.println("Enter the no of Days");
            int nod=sc.nextInt();
            if(nod<=0)
            {
                System.out.println(nod+" is an invalid days");
            }
            else
            {
                System.out.println("Enter the no of Tickets");
```

```
int not=sc.nextInt();
if(not<=0)
{
    System.out.println(not+" is an invalid tickets");
}
else
{
    double d=(double)map.get(name.toUpperCase());
    double totalcost=d*(double)not*(double)nod;
    if(totalcost>=1000)
    {
        totalcost=totalcost-((totalcost*15)/100);
    }
    System.out.printf("Bill Amount is %.2f", totalcost);
}

}

//Fill the code
}

}
```

SUBSTITUTION CYPHER TECHNIQUE

Main.java*

```
import java.util.Scanner;

public class Main {

    public static void main (String[] args) {

        Scanner sc=new Scanner(System.in);

        System.out.println("Enter the Encrypted text: ");

        String code= sc.nextLine();

        int shift = 7;

        int f=0;

        String decryptMessage = " ";

        for(int i = 0; i < code.length(); i++){

            char alpha = code.charAt(i);

            if(alpha >='a' && alpha <= 'z'){

                f=1;

                alpha=(char)(alpha - shift);

                if(alpha < 'a'){

                    alpha = (char)(alpha - 'a'+ 'z'+1);

                }

                decryptMessage=decryptMessage+alpha;

            }

            else if (alpha >='A' && alpha <= 'Z'){

                f=1;

                alpha=(char)(alpha - shift);

                if(alpha < 'A'){

                    alpha = (char)(alpha - 'A'+ 'Z'+1);

                }

                decryptMessage=decryptMessage+alpha;

            }

            else if(alpha == ' '){

                decryptMessage=decryptMessage+alpha;

            }

        }

    }

}
```

```
    }  
}  
if(decryptMessage.length() == 0 || f == 0){  
    System.out.println("No hidden Message");  
    System.exit(0);  
}  
System.out.println("Decrypted Text:\n"+decryptMessage);  
}  
}
```

Account.java*

```
public class Account {  
    long accountNumber;  
    double balanceAmount;  
  
    public Account(long accno, double bal){  
        super();  
        this.accountNumber=accno;  
        this.balanceAmount=bal;  
    }  
    public long getAccountNumber(){  
        return accountNumber;  
    }  
    public void setAccountNumber(long accno){  
        this.accountNumber=accno;  
    }  
    public double getBalanceAmount(){  
        return balanceAmount;  
    }  
    public void setBalanceAmount(double bal) {  
        this.balanceAmount=bal;  
    }  
    public void deposit(double depositAmt){  
        float total=(float)(balanceAmount+depositAmt);  
        balanceAmount=total;  
    }  
    public boolean withdraw(double withdrawAmt){  
        float total;  
        if(withdrawAmt>balanceAmount){  
            System.out.println("Insufficient balance");  
  
            return false;  
        }  
    }  
}
```

```

    }else{
        total=(float)(balanceAmount-withdrawAmt);
        setBalanceAmount(total);
        return true;
    }
}
}
}

```

Main.java*

```

import java.util.Scanner;

public class Main{
    static Account ac=new Account(0, 0);
    public static void main (String[] args) {
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter the account number:");
        ac.setAccountNumber(sc.nextLong());
        System.out.println("Enter the available amount in the account:");
        ac.setBalanceAmount(sc.nextDouble());
        System.out.println("Enter the amount to be deposited:");
        ac.deposit(sc.nextDouble());
        System.out.printf("Available balance is:%.2f",ac.getBalanceAmount());
        System.out.println();
        System.out.println("Enter the amount to be withdrawn:");
        ac.withdraw(sc.nextDouble());
        System.out.printf("Available balance is:%.2f",ac.getBalanceAmount());
        //Fill the code
    }
}

```

THE NEXT RECHARGE DATE

Main.java*

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class Main {

    public static void main(String [] args)throws Exception {

        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Recharged date");
        String date=br.readLine();
        String currentDate="29/10/2019";
        if(Main.isValidFormat(date)&&(Main.dateCompare(date,currentDate))){
            System.out.println("Validity days");
            int days=Integer.parseInt(br.readLine());
            if(days>0)
                System.out.println(Main.futureDate(date,days));
            else
                System.out.println(days+ "is not a valid days");
        }
        else
            System.out.println(date+ "is not a valid date");
    }

    public static boolean isValidFormat(String date){
        String regex="^(3[01]|[12][0-9]|0[1-9])/(1[0-2]|0[1-9])/[0-9]{4}$";
        Pattern pattern=Pattern.compile(regex);
        Matcher matcher=pattern.matcher((CharSequence)date);
        return matcher.matches();
    }
}
```



```

}

public static boolean dateCompare(String date1,String date2)throws ParseException{

    SimpleDateFormat sdformat=new SimpleDateFormat("dd/MM/yyyy");

    Date d1=sdformat.parse(date1);

    Date d2=sdformat.parse(date2);

    if((d1.compareTo(d2)<0)|| (d1.compareTo(d2)==0))

        return true;

    else

        return false;

}

public static String futureDate(String date,int days){

    Calendar c=Calendar.getInstance();

    SimpleDateFormat sdformat=new SimpleDateFormat("dd/MM/yyyy");

    try{

        Date mydate=sdformat.parse(date);

        c.setTime(mydate);

        c.add(Calendar.DATE, days);

    }catch(ParseException e){

        e.printStackTrace();

    }

    String toDate=sdformat.format(c.getTime());

    return toDate;

}

}

```

A New You Spa

DiamondMembers.java*

```
public class DiamondMembers extends Members{

    // Fill the code

    public DiamondMembers(String customerId,String  customerName,long mobileNumber,String
memberType,String emailId){

        super(customerId,customerName,mobileNumber,memberType,emailId);

        /*this.customerId = customerId;

            this.customerName = customerName;

            this.mobileNumber = mobileNumber;

            this.memberType = memberType;

            this.emailId = emailId;*/

    }
```

```
    public boolean validateCusomerId(){

        // Fill the code

        boolean b=true;

        String s1 = this.customerId.toUpperCase();

        String regex="[DIAMOND]{7}[0-9]{3}";

        if(s1.matches(regex)){

            b=true;

        }

        else{

            b=false;

        }

        return b;

    }

    public double calculateDiscount(double purchaseAmount){

        // Fill the code
```

```

        double discount=purchaseAmount*0.45;

        double updateamount=purchaseAmount-discount;

        return updateamount;

    }

}

```

GoldMembers.java*

```

public class GoldMembers extends Members {

    public GoldMembers(String customerId,String  customerName,long mobileNumber,String memberType,String
    emailId){

        super(customerId,customerName,mobileNumber,memberType,emailId);

    }

    // Fill the code

    public boolean validateCusomerId(){

        boolean b=true;

        String s1 = this.customerId.toUpperCase();

        String regex="[GOLD]{4}[0-9]{3}";

        if(s1.matches(regex)){

            b=true;

        }

        else{

            b=false;

        }

        return b;

        // Fill the code

    }

    public double calculateDiscount(double purchaseAmount){

        // Fill the code

        double discount=purchaseAmount*0.15;

        double updateamount=purchaseAmount-discount;

        return updateamount;

    }

}

```

Members.java*

```
abstract public class Members {  
    protected String customerId;  
    protected String customerName;  
    protected long mobileNumber;  
    protected String memberType;  
    protected String emailId;  
  
    abstract public double calculateDiscount(double purchaseAmount);  
  
    public String getCustomerId() {  
        return customerId;  
    }  
    public void setCustomerId(String customerId) {  
        this.customerId = customerId;  
    }  
    public String getCustomerName() {  
        return customerName;  
    }  
    public void setCustomerName(String customerName) {  
        this.customerName = customerName;  
    }  
    public long getMobileNumber() {  
        return mobileNumber;  
    }  
    public void setMobileNumber(long mobileNumber) {  
        this.mobileNumber = mobileNumber;  
    }  
  
    public String getMemberType() {  
        return memberType;  
    }  
    public void setMemberType(String memberType) {
```

```

        this.memberType = memberType;
    }

    public String getEmailId() {
        return emailId;
    }

    public void setEmailId(String emailId) {
        this.emailId = emailId;
    }

    public Members(String customerId, String customerName, long mobileNumber, String memberType, String
emailId) {
        this.customerId = customerId;
        this.customerName = customerName;
        this.mobileNumber = mobileNumber;
        this.memberType = memberType;
        this.emailId = emailId;
    }

}

```

PlatinumMembers.java*

```

public class PlatinumMembers extends Members {

    // Fill the code

    public PlatinumMembers(String customerId,String customerName,long mobileNumber,String
memberType,String emailId){
        super(customerId,customerName,mobileNumber,memberType,emailId);
        /*customerId = customerId;

            customerName = customerName;

            mobileNumber = mobileNumber;

            memberType = memberType;

            emailId = emailId;

        */
    }
}

```

```

public boolean validateCusomerId(){
    // Fill the code

    boolean b=true;

    String s1 = this.customerId.toUpperCase();
    String regex="[PLATINUM]{8}[0-9]{3}";
    if(s1.matches(regex)){
        b=true;
    }
    else{
        b=false;
    }

    return b;
}

public double calculateDiscount(double purchaseAmount){
    // Fill the code

    double discount=purchaseAmount*0.3;
    double updateamount=purchaseAmount-discount;
    return updateamount;
}

}

```

UserInterface.java*

```

import java.util.Scanner;

public class UserInterface {

    public static void main(String[] args){

        Scanner sc=new Scanner(System.in);

        System.out.println("Enter Customer Id");
    }
}

```

```

String cid=sc.nextLine();

System.out.println("Enter Customer name");

String cname=sc.nextLine();

System.out.println("Enter mobile number");

long mob=sc.nextLong();

sc.nextLine();

System.out.println("Enter Member type");

String mem=sc.nextLine();

System.out.println("Enter Email Id");

String email=sc.nextLine();

System.out.println("Enter amount Purchased");

double amount=sc.nextDouble();

DiamondMembers d=new DiamondMembers(cid,cname,mob,mem,email);

GoldMembers g=new GoldMembers(cid,cname,mob,mem,email);

PlatinumMembers p=new PlatinumMembers(cid,cname,mob,mem,email);


double res=0.0;

if(d.validateCusomerId()){

    res= d.calculateDiscount(amount);

    System.out.println("Name :"+d.getCustomerName());

    System.out.println("Id :"+d.getCustomerId());

    System.out.println("Email Id :"+d.getEmailId());

    System.out.println("Amount to be paid :"+res);

} else if(g.validateCusomerId()){

    res= g.calculateDiscount(amount);

    System.out.println("Name :"+g.getCustomerName());

    System.out.println("Id :"+g.getCustomerId());

    System.out.println("Email Id :"+g.getEmailId());

    System.out.println("Amount to be paid :"+res);

} else if(p.validateCusomerId()){

    res= p.calculateDiscount(amount);

```

```
System.out.println("Name :"+p.getCustomerName());
```

```
System.out.println("Id :"+p.getCustomerId());
```

```
System.out.println("Email Id :"+p.getEmailId());
```

```
System.out.println("Amount to be paid :"+res);
```

```
} else{
```

```
    System.out.println("Provide a valid Customer Id");
```

```
}
```

```
    // Fill the code
```

```
}
```

```
}
```


BATTING AVERAGE

UserInterface.java*

```
package com.ui;

import com.utility.Player;
import java.util.ArrayList;
import java.util.Scanner;

public class UserInterface {

    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);
        Player player=new Player();
        player.setScoreList(new ArrayList<>());
        boolean flag=true;
        while(flag)
        {
            System.out.println("1. Add Runs Scored");
            System.out.println("2. Calculate average runs scored");
            System.out.println("3. Exit");
            System.out.println("Enter your choice");
            int choice=sc.nextInt();
            switch(choice)
            {
                case 1: {
                    System.out.println("Enter the runs scored");
                    int runScored=sc.nextInt();
                    player.addScoreDetails(runScored);
                    break;
                }
                case 2: {
                    System.out.println("Average runs secured");
                    System.out.println(player.getAverageRunScored());
                    break;
                }
            }
        }
    }
}
```

```

        case 3: {
            System.out.println("Thank you for using the Application");
            flag=false;
            break;
        }
    }
}

}

}

```

Player.java*

```
package com.utility;
```

```
import java.util.List;
```

```
public class Player {
```

```
    private List<Integer> scoreList;
```

```
    public List<Integer> getScoreList() {
        return scoreList;
    }

```

```
    public void setScoreList(List<Integer> scoreList) {
        this.scoreList = scoreList;
    }

```

```

//This method should add the runScored passed as the argument into the scoreList
    public void addScoreDetails(int runScored) {

```

```

        // fill the code

        scoreList.add(runScored);

    }

```

/* This method should return the average runs scored by the player

Average runs can be calculated based on the sum of all runScored available in the scoreList divided by the number of elements in the scoreList.

For Example:

List contains[150,50,50]

average runs secured= $(150+50+50)/3=83.33333333333333$

so this method should return 83.33333333333333

If list is empty return 0

*/

```

public double getAverageRunScored() {

```

```

    // fill the code

    if(scoreList.isEmpty()) {
        return 0.0;
    }

    int size=scoreList.size();
    int totalScore=0;
    for(int score : scoreList)
    {
        totalScore+=score;
    }

    return (double) totalScore / (double) size;
}

```

```

}

```

Change The Cash

Main.java*

```
import java.util.*;

public class Main{

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String a = sc.next();
        if(a.length() < 3) {
            System.out.println("String length of " + a + " is too short");
            return;
        }
        else if(a.length() > 10) {
            System.out.println("String length of " + a + " is too long");
            return;
        }

        char[] arr = a.toCharArray();
        char[] arr1 = new char[arr.length];
        int j = 0;
        for(int i = 0; i < a.length(); i++) {
            if((arr[i]<65 || ((arr[i]>90) && (arr[i]<97)) || arr[i]>122)) {
                arr1[j++] = arr[i];
            }
        }
        if(j!=0) {
            System.out.print("String should not contain ");
            for(int i = 0; i<=j; i++) {
                System.out.print(arr1[i]);
            }
            return;
        }
        char b = sc.next().charAt(0);
```

```
int present = 0;

for(int i = 0; i<a.length(); i++) {

    if(arr[i] == Character.toUpperCase(b)) {

        arr[i] = Character.toLowerCase(b);

        present = 1;

    }

    else if(arr[i] == Character.toLowerCase(b)) {

        arr[i] = Character.toUpperCase(b);

        present = 1;

    }

}

if(present == 0) {

    System.out.println("Character " + b + " is not found");

}

else {

    for(int i = 0; i <a.length(); i++) {

        System.out.print(arr[i]);

    }

}

}
```

Check Number Type

NumberType.java*

```
public interface NumberType
{
    public boolean checkNumberType(int n);
}
```

NumberTypeUtility.java*

```
import java.util.Scanner;

public class NumberTypeUtility
{
    public static NumberType isOdd()
    {
        return n -> n%2 != 0;
    }

    public static void main (String[] args)
    {

        Scanner sc=new Scanner(System.in);
        int n=sc.nextInt();
        if(isOdd().checkNumberType(n))
        {
            System.out.println(n+" is odd");
        }
        else
        {
            System.out.println(n+" is not odd");
        }
    }
}
```

Cheque Payment Process

PaymentDao.java*

```
package com.cts.paymentProcess.dao;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

import com.cts.paymentProcess.model.Payment;
import com.cts.paymentProcess.util.DatabaseUtil;

public class PaymentDao {

    private Connection connection;

    public List<Payment> getAllRecord(){

        connection=DatabaseUtil.getConnection();
        PreparedStatement statement=null;
        ResultSet resultSet=null;
        List<Payment> paymentList=new ArrayList<Payment>();
        try {
            statement=connection.prepareStatement("select * from cheque_payments");
            resultSet=statement.executeQuery();
            while(resultSet.next()){
                Payment payment =new Payment();
                payment.setCustomerNumber(resultSet.getInt("customerNumber"));
                payment.setChequeNumber(resultSet.getString("chequeNumber"));
                payment.setPaymentDate(resultSet.getDate("paymentDate"));
                payment.setAmount(resultSet.getInt("amount"));
                paymentList.add(payment);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return paymentList;
    }
}
```

```

        }

        } catch (SQLException e) {

            e.printStackTrace();
        } finally {
            try {
                resultSet.close();
                statement.close();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }

        return paymentList;
    }
}

```

Payment.java*

```
package com.cts.paymentProcess.model;
```

```
import java.util.Date;
```

```
public class Payment {
```

```
    private int customerNumber;
```

```
    private String chequeNumber;
```

```
    private Date paymentDate;
```

```
    private int amount;
```

```
    public int getCustomerNumber() {
```

```
        return customerNumber;
```

```
    }
```



```

public void setCustomerNumber(int customerNumber) {
    this.customerNumber = customerNumber;
}

public String getChequeNumber() {
    return chequeNumber;
}

public void setChequeNumber(String chequeNumber) {
    this.chequeNumber = chequeNumber;
}

public Date getPaymentDate() {
    return paymentDate;
}

public void setPaymentDate(Date paymentDate) {
    this.paymentDate = paymentDate;
}

public int getAmount() {
    return amount;
}

public void setAmount(int amount) {
    this.amount = amount;
}

@Override
public String toString() {

    return String.format("%15s%15s%15s%15s", customerNumber, chequeNumber, paymentDate,
amount);

}

}

```

PaymentService.java*

```
package com.cts.paymentProcess.service;
```

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

```
import java.util.Calendar;
```

```
import java.util.List;
```

```
import java.util.stream.Collectors;
```

```
import com.cts.paymentProcess.dao.PaymentDao;
```

```
import com.cts.paymentProcess.model.Payment;
```

```
public class PaymentService {
```

```
    private PaymentDao paymentDao=new PaymentDao();
```

```
    public List<Payment> findCustomerByNumber(int customerNumber){
```

```
        List<Payment> list=paymentDao.getAllRecord();
```

```
        List<Payment> list2 = new ArrayList<>();
```

```
        list2 = list.stream().filter(x->x.getCustomerNumber()==customerNumber).collect(Collectors.toList());
```

```
        return list2;
```

```
    }
```

```
    public List<Payment> findCustomerByYear(int year){
```

```
        List<Payment> list=paymentDao.getAllRecord();
```

```
        List<Payment> list2 = new ArrayList<>();
```

```
        list2 = list.stream().filter(x->x.getPaymentDate().getYear()==(year-1900)).sorted(Comparator.comparingInt(Payment::getAmount)).collect(Collectors.toList());
```

```
        return list2;
```

```
    }
```

```
}
```

SkeletonValidator.java*

```
package com.cts.paymentProcess.skeletonValidator;

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

public class SkeletonValidator {

    public SkeletonValidator(){

        validateClassName("com.cts.paymentProcess.dao.PaymentDao");
        validateMethodSignature("getAllRecord:java.util.List","com.cts.paymentProcess.dao.PaymentDao");

        validateClassName("com.cts.paymentProcess.model.Payment");
        validateMethodSignature("toString:java.lang.String","com.cts.paymentProcess.model.Payment");

        validateClassName("com.cts.paymentProcess.service.PaymentService");

        validateMethodSignature("findCustomerByNumber:java.util.List,findCustomerByYear:java.util.List","com.cts.paymentProcess.service.PaymentService");

        validateClassName("com.cts.paymentProcess.util.DatabaseUtil");

        validateMethodSignature("getConnection:java.sql.Connection","com.cts.paymentProcess.util.DatabaseUtil")
;

    }

    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 methodWithExcpn, String className) {

    Class cls = null;

    try {

        String[] actualmethods = methodWithExcpn.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");  
    }  
}
```

```
}
```

DatabaseUtil.java*

```
package com.cts.paymentProcess.util;
```

```
import java.io.FileInputStream;
```

```
import java.io.IOException;
```

```
import java.io.InputStream;
```

```
import java.sql.Connection;
```

```
import java.sql.DriverManager;
```

```
import java.sql.SQLException;
```

```
import java.util.Properties;
```

```
import java.util.ResourceBundle;
```

```
public class DatabaseUtil {
```

```
    private DatabaseUtil() {
```

```
    }
```

```
    private static Connection con = null;
```

```
    private static Properties props = new Properties();
```

```
//ENSURE YOU DON'T CHANGE THE BELOW CODE WHEN YOU SUBMIT
```

```
    public static Connection getConnection() {
```

```
        try{
```

```

        FileInputStream fis = null;

        fis = new FileInputStream("resource/database.properties");

        props.load(fis);


        // load the Driver Class
        try {

            Class.forName(props.getProperty("DB_DRIVER_CLASS"));

        } catch (ClassNotFoundException e) {

            // TODO Auto-generated catch block

            e.printStackTrace();

        }


        // create the connection now

        try {

            con =
DriverManager.getConnection(props.getProperty("DB_URL"),props.getProperty("DB_USERNAME"),props.getPropert
y("DB_PASSWORD"));

            } catch (SQLException e) {

                // TODO Auto-generated catch block

                e.printStackTrace();

            }

        }

        catch(IOException e){

            e.printStackTrace();

        }

        return con;

    }

}

```

App.java(Main)*

```
package com.cts.paymentProcess;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.List;
import java.util.Scanner;

import com.cts.paymentProcess.model.Payment;
import com.cts.paymentProcess.service.PaymentService;
import com.cts.paymentProcess.skeletonValidator.SkeletonValidator;

public class App {

    public static void main(String[] args) throws ParseException {

        new SkeletonValidator();

        Payment payment=null;
        Scanner scanner=new Scanner(System.in);

        do{

            System.out.println("Select Option:");
            System.out.println("1.Customer list\n2.Yearly Customer List\n3.Exit");
            int choice=scanner.nextInt();
            switch(choice){

                case 1:System.out.println("Enter customer number");
                int number=scanner.nextInt();
                List<Payment> numberList=new PaymentService().findCustomerByNumber(number);
                if(numberList.size()==0){
                    System.out.println("\nNo Records Found\n");
                }else{
```



```

        System.out.format("%15s%15s%15s%15s\n","Customer Number","Cheque Number","Payment
Date","Amount");

        numberList.stream()

        .forEach(System.out::println);

    }

        break;

        case 2: System.out.println("Enter year");

        int year=scanner.nextInt();

        List<Payment> yearList=new PaymentService().findCustomerByYear(year);

        if(yearList.size()==0){

            System.out.println("\nNo Records Found\n");

        }else{

            System.out.format("%15s%15s%15s%15s\n","Customer Number","Cheque Number","Payment
Date","Amount");

            yearList.stream()

            .forEach(System.out::println);

        }

        break;

        case 3: System.exit(0);

        default: System.out.println("\nWrong Choice\n");

    }

    }while(true);

}

}

```

Employee Eligibility for Promotion

Main.java*

```
import java.time.format.DateTimeFormatter;

import java.time.temporal.ChronoUnit;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.time.*;

import java.util.TreeMap;
import java.util.Date;
import java.util.Map;

import java.util.Map.Entry;

import java.util.Scanner;

public class Main {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        //System.out.println("In-time");

        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd/MM/yyyy");

        LocalDate currTime = LocalDate.of(2019, 01, 01);

        String fdt=currTime.format(formatter);

        // System.out.println(fdt);

        int no = sc.nextInt();

        Map<String, String> m = new TreeMap<>();

        for (int i = 1; i <= no; i++) {

            String id = sc.next();

            String date = sc.next();

            m.put(id, date);

        }

    }

}
```

```

int count = 0;

int val = 0;

for (Entry<String, String> entry : m.entrySet()) {
    if (entry.getValue().matches("(0[1-9] | [1-2][0-9] | 3[0-1])/(0[1-9] | 1[0-2])/[0-9]{4}"))
    {
        val++;

        LocalDate InTime = LocalDate.parse(entry.getValue(), formatter);
        Period in = Period.between(InTime, currTime);
        long lin = in.get(ChronoUnit.YEARS);
        if (lin >= 5)
        {
            count++;

            System.out.println(entry.getKey());
        }
    }
    else
    {
        System.out.println("Invalid date format");
        break;
    }
}

if (count == 0 && val == no)
    System.out.println("No one is eligible");
}
}

```

Exam Scheduler

AssessmentDao.java*

```
package com.cts.cc.dao;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.time.Duration;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.Period;
import java.util.List;
import java.sql.*;

import com.cts.cc.model.Assessment;
import com.cts.cc.util.DatabaseUtil;

public class AssessmentDAO {

    public int uploadAssessments(List<Assessment> assessments) throws Exception {

        if(assessments==null || assessments.isEmpty()) {

            throw new Exception("List is Empty");

        }

        int rowCount = 0;

        //Write your logic here...

        try{

            Connection con = DatabaseUtil.getConnection();

            for(Assessment a:assessments)

            {

                PreparedStatement st = con.prepareStatement("insert into assessment values(?,?,?, ?,?,?)");

                st.setString(1,a.getExamCode());

                st.setString(2,a.getExamTitle());

                st.setString(3,a.getExamDate().toString());

                st.setString(4,a.getExamTime().toString());

            }

        }

    }

}
```

```

        st.setString(5,a.getExamDuration().toString());

        st.setString(6,a.getEvalDays().toString());

        int rs=st.executeUpdate();

        if(rs!=-1)

            rowsCount=rowsCount+1;
    }
    } catch(SQLException e){
    }

    return rowsCount;
}

public Assessment findAssessment(String code) throws Exception {
    Assessment assessment = null;

    Connection conn = DatabaseUtil.getConnection();

    String sql = "SELECT * FROM assessment where code=?";

    PreparedStatement ps = conn.prepareStatement(sql);

    ps.setString(1, code);

    ResultSet rs = ps.executeQuery();

    if(rs.next()) {
        assessment = new Assessment();

        assessment.setExamCode(rs.getString(1));

        assessment.setExamTitle(rs.getString(2));

        assessment.setExamDate(LocalDate.parse(rs.getString(3)));

        assessment.setExamTime(LocalTime.parse(rs.getString(4)));

        assessment.setExamDuration(Duration.parse(rs.getString(5)));

        assessment.setEvalDays(Period.parse(rs.getString(6)));

    }

    return assessment;
}
}

```

GenerateAssessmentFunction.java*

```
package com.cts.cc.functions;

import java.time.Duration;
import java.time.LocalDate;
import java.time.LocalTime;
import java.time.Period;
import java.util.function.Function;

import com.cts.cc.model.Assessment;

public class GenerateAssessmentFunction implements Function<String, Assessment>{

    @Override
    public Assessment apply(String t) {
        //Write your code here...

        String temp[]=t.split(",");

        Assessment a = new
Assessment(temp[0],temp[1],LocalDate.parse(temp[2]),LocalTime.parse(temp[3]),Duration.parse(temp[4]),Period.p
arse(temp[5]));

        return a;
    }

}
```

Assessment.java*

```
package com.cts.cc.model;

import java.time.Duration;
import java.time.LocalDate;
import java.time.LocalTime;
import java.time.Period;
import java.time.format.DateTimeFormatter;

public class Assessment {
```

```
private String examCode;

private String examTitle;

private LocalDate examDate;

private LocalTime examTime;

private Duration examDuration;

private Period evalDays;
```

```
public Assessment(String examCode, String examTitle, LocalDate examDate, LocalTime examTime, Duration
examDuration,
```

```
        Period evalDays) {

    super();

    this.examCode = examCode;

    this.examTitle = examTitle;

    this.examDate = examDate;

    this.examTime = examTime;

    this.examDuration = examDuration;

    this.evalDays = evalDays;

}
```

```
public Assessment() {

}
```

```
public String getExamCode() {

    return examCode;

}
```

```
public void setExamCode(String examCode) {

    this.examCode = examCode;

}
```

```
public String getExamTitle() {

    return examTitle;

}
```

```
public void setExamTitle(String examTitle) {  
    this.examTitle = examTitle;  
}
```

```
public LocalDate getExamDate() {  
    return examDate;  
}
```

```
public void setExamDate(LocalDate examDate) {  
    this.examDate = examDate;  
}
```

```
public LocalTime getExamTime() {  
    return examTime;  
}
```

```
public void setExamTime(LocalTime examTime) {  
    this.examTime = examTime;  
}
```

```
public Duration getExamDuration() {  
    return examDuration;  
}
```

```
public void setExamDuration(Duration examDuration) {  
    this.examDuration = examDuration;  
}
```

```
public Period getEvalDays() {  
    return evalDays;  
}
```

```
public void setEvalDays(Period evalDays) {  
    this.evalDays = evalDays;  
}
```



```
}
```

```
public void printDetails() {  
    DateTimeFormatter date1=DateTimeFormatter.ofPattern("dd-MMM-y");  
    DateTimeFormatter date2=DateTimeFormatter.ofPattern("HH:mm");  
    LocalTime t=examTime.plus(examDuration);  
    String d=DateTimeFormatter.ofPattern("HH:mm").format(t);  
    LocalDate t1=examDate.plus(evalDays);  
    String d1=DateTimeFormatter.ofPattern("dd-MMM-y").format(t1);  
    System.out.println("Assessment Code: "+examCode);  
    System.out.println("Title: "+examTitle);  
    System.out.println("Assessment Date: "+examDate.format(date1));  
    System.out.println("Start Time: "+examTime.format(date2));  
    System.out.println("End Time: "+d);  
    System.out.println("Result Date: "+d1);  
    //Write your code here...  
}
```

```
}
```

DatabaseUtil.java*

```
package com.cts.cc.util;  
  
import java.io.FileInputStream;  
import java.io.IOException;  
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.SQLException;  
import java.util.Properties;  
  
public class DatabaseUtil {  
  

```

```
    private static Connection con = null;
```

```
private static Properties props = new Properties();
```

```
//ENSURE YOU DON'T CHANGE THE BELOW CODE WHEN YOU SUBMIT
```

```
public static Connection getConnection() throws ClassNotFoundException, SQLException {
```

```
    try{
```

```
        FileInputStream fis = null;
```

```
        fis = new FileInputStream("resource/connection.properties");
```

```
        props.load(fis);
```

```
        // load the Driver Class
```

```
        Class.forName(props.getProperty("DB_DRIVER_CLASS"));
```

```
        // create the connection now
```

```
        con =
```

```
        DriverManager.getConnection(props.getProperty("DB_URL"),props.getProperty("DB_USERNAME"),props.getProperty("DB_PASSWORD"));
```

```
    }
```

```
    catch(IOException e){
```

```
        e.printStackTrace();
```

```
    }
```

```
        return con;
```

```
    }
```

```
}
```

FileUtil.java*

```
package com.cts.cc.util;
```

```
import java.io.IOException;
```

```
import java.nio.file.Files;
```

```
import java.nio.file.Path;
```

```
import java.nio.file.Paths;
```

```
import java.util.List;
```

```
import java.util.function.Function;
```

```

import java.util.stream.Collectors;

import java.util.stream.Stream;

import java.io.*;

import java.util.*;

import com.cts.cc.functions.GenerateAssessmentFunction;

import com.cts.cc.model.Assessment;

public class FileUtil {

    public static List<Assessment> loadData(String fileName) throws IOException {

        List<Assessment> list = null;

        Function<String, Assessment> function = new GenerateAssessmentFunction();

        BufferedReader br=new BufferedReader(new FileReader(fileName));

        String line="";

        list=new ArrayList<Assessment>();

        while((line=br.readLine())!=null)

        {

            list.add(function.apply(line));

        }

        //Write your code here...

        return list;

    }

}

```

SkeletonValidator.java*

```

package com.cts.cc;

import java.lang.reflect.Constructor;

import java.lang.reflect.Method;

import java.sql.Connection;

import java.time.Duration;

import java.time.LocalDate;

import java.time.LocalDateTime;

```

```

import java.time.Period;

import java.util.List;

import java.util.logging.Level;

import java.util.logging.Logger;


import com.cts.cc.model.Assessment;


public class SkeletonValidator {

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


    public SkeletonValidator() {

        String assessmentClass = "com.cts.cc.model.Assessment";

        String assessmentDAOClass = "com.cts.cc.dao.AssessmentDAO";

        String funtionalClass = "com.cts.cc.functions.GenerateAssessmentFunction";

        String databaseUtilClass = "com.cts.cc.util.DatabaseUtil";

        String fileUtilClass = "com.cts.cc.util.FileUtil";


        Class[] assessmentParams = { String.class, String.class, LocalDate.class, LocalTime.class,
Duration.class,
                Period.class };

        String[] assessmentFields = { "examCode", "examTitle", "examDate", "examTime", "examDuration",
"evalDays" };


        testClass(assessmentClass, assessmentParams);

        testClass(assessmentDAOClass, null);

        testClass(funtionalClass, null);

        testClass(databaseUtilClass, null);

        testClass(fileUtilClass, null);


        testFields(assessmentClass, assessmentFields);

        testMethods(assessmentClass, "printDetails", null, null);

        testMethods(assessmentDAOClass, "uploadAssessments", new Class[] { List.class }, boolean.class);

        testMethods(assessmentDAOClass, "findAssessment", new Class[] { String.class }, Assessment.class);

        testMethods(funtionalClass, "apply", new Class[] { String.class }, Assessment.class);
    }
}

```

```

        testMethods(databaseUtilClass, "getConnection", null, Connection.class);
        testMethods(fileUtilClass, "loadData", new Class[] { String.class }, List.class);
    }

```

```

public void testClass(String className, Class[] paramTypes) {
    try {
        Class classUnderTest = Class.forName(className);
        LOG.info("Class Name " + className + " is correct");
        Constructor<?> constructor = classUnderTest.getConstructor(paramTypes);
        constructor.equals(constructor);
        LOG.info(className + " Constructor is valid");
    } 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 (NoSuchMethodException e) {
        LOG.log(Level.SEVERE, " Unable to find the given constructor. "
            + "Do not change the given public constructor. " + "Verify it with the
skeleton");
    } catch (SecurityException e) {
        LOG.log(Level.SEVERE,
            "There is an error in validating the " + className + ". " + "Please verify the
skeleton manually");
    }
}

```

```

public void testFields(String className, String[] fields) {
    try {
        Class classUnderTest = Class.forName(className);
        for (String field : fields) {
            classUnderTest.getDeclaredField(field);
        }
        LOG.info("Fields in " + className + " are 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 (NoSuchFieldException e) {
            LOG.log(Level.SEVERE,
                "You have changed one/more field(s). " + "Use the field name(s) as provided
in the skeleton");
        } catch (SecurityException e) {
            LOG.log(Level.SEVERE,
                "There is an error in validating the " + className + ". " + "Please verify the
skeleton manually");
        }
    }

    public void testMethods(String className, String methodName, Class[] paramTypes, Class returnType) {
        try {
            Class classUnderTest = Class.forName(className);
            Method method = classUnderTest.getDeclaredMethod(methodName, paramTypes);
            Class retType = method.getReturnType();
            if (returnType != null && !retType.equals(returnType)) {
                LOG.log(Level.SEVERE, " You have changed the " + "return type in '" + methodName
                    + "' method. Please stick to the " + "skeleton provided");
                throw new NoSuchMethodException();
            }
            LOG.info(methodName + " signature is valid.");
        } 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 (NoSuchMethodException e) {
            LOG.log(Level.SEVERE, "You have changed/removed method " + methodName + ". "
                + "Use the method signature as provided in the skeleton");
        } catch (SecurityException e) {
            LOG.log(Level.SEVERE,
                "There is an error in validating the " + className + ". " + "Please verify the
skeleton manually");
        }
    }
}

```

Main.java*

```
package com.cts.cc;

import java.util.List;

import com.cts.cc.dao.AssessmentDAO;
import com.cts.cc.model.Assessment;
import com.cts.cc.util.FileUtil;

public class Main {

    public static void main(String[] args) {

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

        try {

            List<Assessment> assessments = FileUtil.loadData("resource/data.txt");

            AssessmentDAO dao = new AssessmentDAO();

            dao.uploadAssessments(assessments);

            Assessment assessment = dao.findAssessment("ASEJE025");

            assessment.printDetails();

        } catch (Exception e) {

            System.out.println(e);

        }

    }

}
```

Book Details

Main.java*

```
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        //fill the code

        String string = sc.nextLine();

        if(string.length()==18)

        {

            String substr1 = string.substring(0,3);

            int i1 = Integer.parseInt(substr1);

            if(i1>100 && i1<104)

            {

                String substr2 = string.substring(3,7);

                int i2 = Integer.parseInt(substr2);

                if(i2>=1900 && i2<=2020)

                {

                    String substr3 = string.substring(7,12);

                    //System.out.println(substr3);

                    int i3 = Integer.parseInt(substr3);

                    if(i3>=10)

                    {

                        String substr4 = string.substring(12,13);

                        String substr5 = string.substring(13,18);

                        if((substr4.charAt(0)>='A'&&substr4.charAt(0)<='Z') || (substr4.charAt(0)>='a'&&substr4.charAt(0)<='z'))

                        {

                            if((substr5.charAt(0)>='0'&&substr5.charAt(0)<='9')&&(substr5.charAt(1)>='0'&&substr5.charAt(1)<='9')&&

                                (substr5.charAt(2)>='0'&&substr5.charAt(2)<='9')&&(substr5.charAt(3)>='0'&&substr5.charAt(3)<='9')&&

                                    (substr5.charAt(4)>='0'&&substr5.charAt(4)<='9'))

                            {
```



```

        String substr6 = string.substring(12,18);
        System.out.println("Department Code: "+substr1);
        if(i1==101)
            System.out.println("Department Name: "+"Accounting");
        else if(i1==102)
            System.out.println("Department Name: "+"Economics");
        else if(i1==103)
            System.out.println("Department Name: "+"Engineering");
        System.out.println("Year of Publication: "+substr2);
        System.out.println("Number of Pages: "+substr3);
        System.out.println("Book Id: "+substr6);

    }
    else
    {
        String substr6 = string.substring(12,18);
        System.out.printf("%s is invalid book id",substr6);
        System.out.printf("\n");
    }
}
else
{
    String substr6 = string.substring(12,18);
    System.out.printf("%s is invalid book id",substr6);
    System.out.printf("\n");
}
}
else
{
    System.out.printf("%s are invalid pages",substr3);
    System.out.printf("\n");
}
}

```

```
    }
    else
    {
        System.out.printf("%d is invalid year",i2);
        System.out.printf("\n");
    }
}
else
{
    System.out.printf("%d is invalid department code",i1);
    System.out.printf("\n");
}
}
else
{
    System.out.printf("%s is invalid input",string);
    System.out.printf("\n");
}
}
}
```

Find Membership Category

Member.java*

```
public class Member {

    private String memberId;

    private String memberName;

    private String category;

    public String getMemberId() {

        return memberId;

    }

    public void setMemberId(String memberId) {

        this.memberId = memberId;

    }

    public String getMemberName() {

        return memberName;

    }

    public void setMemberName(String memberName) {

        this.memberName = memberName;

    }

    public String getCategory() {

        return category;

    }

    public void setCategory(String category) {

        this.category = category;

    }

    public Member(String memberId, String memberName, String category) {

        super();

        this.memberId = memberId;

        this.memberName = memberName;

        this.category = category;

    }

}
```

```
}
```

ZeeShop.java*

```
import java.util.List;
```

```
public class ZEEShop extends Thread
```

```
{
```

```
    private String memberCategory;
```

```
    private int count;
```

```
    private List<Member> memberList;
```

```
    public ZEEShop(String memberCategory, List<Member> memberList) {
```

```
        super();
```

```
        this.memberCategory = memberCategory;
```

```
        this.memberList = memberList;
```

```
    }
```

```
    public String getMemberCategory() {
```

```
        return memberCategory;
```

```
    }
```

```
    public void setMemberCategory(String memberCategory) {
```

```
        this.memberCategory = memberCategory;
```

```
    }
```

```
public int getCount() {  
    return count;  
}
```

```
public void setCount(int count) {  
    this.count = count;  
}
```

```
public List<Member> getMemberList() {  
    return memberList;  
}
```

```
public void setMemberList(List<Member> memberList) {  
    this.memberList = memberList;  
}
```

```
public void run()  
{  
  
    synchronized(this)  
    {  
        for(Member m:memberList)  
        {  
            if(m.getCategory().equals(memberCategory))  
                count++;  
        }  
    }  
}
```

```
}
```

```
}
```

```
}
```

Main.java*

```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;
```

```
public class Main {
```

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

```
        // TODO Auto-generated method stub
```

```
        List<Member> mList=new ArrayList<Member>();
```

```
        System.out.println("Enter the number of Members:");
```

```
        Scanner sc=new Scanner(System.in);
```

```
        int tot=sc.nextInt();
```

```
        String[] str=new String[tot];
```

```
        for(int i=0;i<str.length;i++)
```

```
        {
```

```
            System.out.println("Enter the Member Details:");
```

```
            str[i]=sc.next();
```

```
        }
```

```
        Member m[]=new Member[tot];
```

```
        for(int i=0;i<m.length;i++)
```

```
        {
```

```
            String s[]=str[i].split(":");
```

```

        m[i]=new Member(s[0],s[1],s[2]);
        mList.add(m[i]);

    }

    System.out.println("Enter the number of times Membership category needs to be searched:");
    int tot1=sc.nextInt();
    ZEEShop t1[]=new ZEEShop[tot1];
    for(int i=0;i<tot1;i++)
    {
        System.out.println("Enter the Category");
        String s1=sc.next();
        t1[i]=new ZEEShop(s1,mList);
        t1[i].start();

        //System.out.println(s1+" "+t1.getCount());
    }

    try {
        Thread.currentThread().sleep(2000);
    } catch (InterruptedException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    for(ZEEShop s:t1)
    {
        System.out.println(s.getMemberCategory()+":"+s.getCount());
    }

}

}

```

Go Hospitals

InPatient.java*

```
public class InPatient extends Patient{

    private double roomRent;

    public InPatient(String patientId, String patientName, long mobileNumber, String gender, double roomRent) {
        super(patientId,patientName,mobileNumber,gender);
        this.roomRent=roomRent;
    }

    public double getRoomRent() {
        return roomRent;
    }

    public void setRoomRent(double roomRent) {
        this.roomRent = roomRent;
    }

    // Fill the code

    public double calculateTotalBill(int noOfDays,double medicinalBill){
        // Fill the code
        double bill_amount;
        bill_amount=((this.roomRent*noOfDays)+medicinalBill);
        return bill_amount;
    }

}
```

OutPatient.java*

```
public class OutPatient extends Patient{
```



```
private double consultingFee;
```

```
public OutPatient(String patientId, String patientName, long mobileNumber, String gender, double consultingFee)
{
    super(patientId,patientName,mobileNumber,gender);
    this.consultingFee=consultingFee;
}
```

```
public double getConsultingFee() {
    return consultingFee;
}
```

```
public void setConsultingFee(double consultingFee) {
    this.consultingFee = consultingFee;
}
```

```
// Fill the code
```

```
public double calculateTotalBill(double scanPay,double medicinalBill){
    // Fill the code
    double bill_amount;
    bill_amount=this.consultingFee+scanPay+medicinalBill;
    return bill_amount;
}
}
```

Patient.java*

```
public class Patient {

    protected String patientId;
    protected String patientName;
```

```
protected long mobileNumber;
```

```
protected String gender;
```

```
public Patient(String patientId, String patientName, long mobileNumber, String gender) {
```

```
    this.patientId = patientId;
```

```
    this.patientName = patientName;
```

```
    this.mobileNumber = mobileNumber;
```

```
    this.gender = gender;
```

```
}
```

```
public String getPatientId() {
```

```
    return patientId;
```

```
}
```

```
public void setPatientId(String patientId) {
```

```
    this.patientId = patientId;
```

```
}
```

```
public String getPatientName() {
```

```
    return patientName;
```

```
}
```

```
public void setPatientName(String patientName) {
```

```
    this.patientName = patientName;
```

```
}
```

```
public long getMobileNumber() {
```

```
    return mobileNumber;
```

```
}
```

```
public void setMobileNumber(long mobileNumber) {
```

```
    this.mobileNumber = mobileNumber;
```

```
}
```

```
public String getGender() {
```

```
    return gender;
```

```
}
```

```
public void setGender(String gender) {
```

```
    this.gender = gender;
```

```
}
```

```
}
```

UserInterface.java*

```
import java.util.Scanner;
```

```
public class UserInterface {
```

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

```
        Scanner read=new Scanner(System.in);
```

```
        System.out.println("1.In Patient");
```

```
        System.out.println("1.Out Patient");
```

```
        System.out.println("Enter the choice");
```

```
        int ch=read.nextInt();
```

```
        System.out.println("Enter the details");
```

```
        System.out.println("Patient Id");
```

```
        String id=read.nextLine();
```

```
        System.out.println("Patient Name");
```

```
        String name=read.nextLine();
```

```
        read.nextLine();
```

```
        System.out.println("Phone Number");
```

```
        long num=read.nextLong();
```

```
        System.out.println("Gender");
```

```
        String gen=read.next();
```

```
        if(ch==1){
```

```
            System.out.println("Room Rent");
```

```
            double rent=read.nextDouble();
```

```
            InPatient in=new InPatient(id,name,num,gen,rent);
```

```
            System.out.println("Medicinal Bill");
```

```

        double bill=read.nextDouble();

        System.out.println("Number of Days of Stay");

        int days=read.nextInt();

        System.out.println("Amount to be paid "+in.calculateTotalBill(days,bill));
    }

    else{

        System.out.println("Consultancy Fee");

        double fee=read.nextDouble();

        OutPatient out=new OutPatient(id,name,num,gen,fee);

        System.out.println("Medicinal Bill");

        double medbill=read.nextDouble();

        System.out.println("Scan Pay");

        double pay=read.nextDouble();

        System.out.println("Amount to be paid "+out.calculateTotalBill(pay,medbill));

    }

    // Fill the code

}

}

```

Grade Calculation

GradeCalculator.java*

```
public class GradeCalculator extends Thread {

    private String studName;
    private char result;
    private int[] marks;

    public String getStudName(){
        return studName;
    }

    public void setStudName(String studName){
        this.studName = studName;
    }

    public char getResult(){
        return result;
    }

    public void setResult(char result){
        this.result = result;
    }

    public int[] getMarks(){
        return marks;
    }

    public void setMarks(int[] marks){
        this.marks = marks;
    }
}
```

```

public GradeCalculator(String studName, int[] marks){

    this.studName = studName;

    this.marks = marks;

}

public void run(){

    int sum = 0;

    int[] score = getMarks();

    for(int i = 0;i<score.length;i++)

        sum = sum+score[i];

    if((400<=sum)&&(sum<=500))

        System.out.println(getStudName()+":'+A');

    if((300<=sum)&&(sum<=399))

        System.out.println(getStudName()+":'+B');

    if((200<=sum)&&(sum<=299))

        System.out.println(getStudName()+":'+C');

    if(sum<200)

        System.out.println(getStudName()+":'+E');

    }

}

```

Main.java*

```

import java.util.Scanner;

import java.io.BufferedReader;

import java.io.InputStreamReader;

public class Main {

    public static void main(String[] args) throws Exception {

        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

        System.out.println("Enter the number of Threads:");

        int th = Integer.parseInt(br.readLine());

        GradeCalculator obj = null;

        String str = "";
    }
}

```

```
String details[] = new String[th];  
for(int i=0; i<th; i++){  
    System.out.println("Enter the String:");  
    str = br.readLine();  
    details[i]=str;  
}
```

```
for(int i=0; i<th; i++){  
    String sp[] = details[i].split(":");  
    int k = 0;  
    int arr[] = new int[sp.length];  
    for(int j = 1; j<sp.length; j++)  
        arr[k++] = Integer.parseInt(sp[j]);  
    obj = new GradeCalculator(sp[0],arr);  
    obj.start();  
    try{  
        Thread.sleep(1000);  
    }  
    catch(Exception e)  
    {  
        System.out.println(e);  
    }  
}
```

```
}
```

```
}
```

Passanger Amenity

Main.java*

```
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {

        int num,n,i,count1=0,count2=0,y;

        char alpha,ch;

        String n1,n2;

        Scanner sc = new Scanner(System.in);

        //fill the code

        System.out.println("Enter the number of passengers");

        n=sc.nextInt();

        if(n<=0){

            System.out.println(n+" is invalid input");

            System.exit(0);

        }

        String[] arr1=new String[n];

        String[] arr2=new String[n];

        for(i=0;i<n;i++,count1=0,count2=0){

            System.out.println("Enter the name of the passenger "+(i+1));

            arr1[i] =sc.next();

            System.out.println("Enter the seat details of the passenger "+(i+1));

            arr2[i]= sc.next();

            num =Integer.parseInt(arr2[i].substring(1,(arr2[i].length())));

            alpha= arr2[i].charAt(0);

            if(num>=10 && num<=99){

                count2++;

            }

            for(ch=65;ch<84;ch++){

                if(ch==alpha){

                    count1++;

                }

            }

            if(count1==0){
```



```

        System.out.println(""+alpha+" is invalid coach");
        System.exit(0);
    }
    if(count2==0){
        System.out.println(num+" is invalid seat number");
        System.exit(0);
    }

}

for(i=0;i<n;i++){
    for(int j=i+1;j<n;j++){
        if(arr2[i].charAt(0)==arr2[j].charAt(0)){

if((Integer.parseInt(arr2[i].substring(1,(arr2[i].length())))<(Integer.parseInt(arr2[j].substring(1,arr2[j].length())))){

            n1=arr1[i];
            n2=arr2[i];
            arr1[i]=arr1[j];
            arr2[i]=arr2[j];
            arr1[j]=n1;
            arr2[j]=n2;
        }
    }
    else
        if(arr2[i].charAt(0)<arr2[j].charAt(0))
        {
            n1=arr1[i];
            n2=arr2[i];
            arr1[i]=arr1[j];
            arr2[i]=arr2[j];
            arr1[j]=n1;
            arr2[j]=n2;
        }
    }
}
}

```

```
for(i=0;i<n;i++){  
    String a=arr1[i].toUpperCase();  
    String b=arr2[i];  
    System.out.print(a+" "+b);  
    System.out.println("");  
}  
}  
}
```

Perform Calculation

Calculate.java*

```
public interface Calculate {  
  
    float performCalculation(int a,int b);  
}
```

Calculator.java*

```
import java.util.Scanner;  
  
public class Calculator {  
  
  
  
  
  
  
    public static void main (String[] args) {  
  
  
  
  
        Scanner sc=new Scanner(System.in);  
  
        int a = sc.nextInt();  
  
        int b= sc.nextInt();  
  
        Calculate Perform_addition = performAddition();  
  
        Calculate Perform_subtraction = performSubtraction();  
  
        Calculate Perform_product = performProduct();  
  
        Calculate Perform_division = performDivision();  
    }  
}
```

```
System.out.println("The sum is "+Perform_addition.performCalculation(a,b));
```

```
System.out.println("The difference is "+Perform_subtraction.performCalculation(a,b));
```

```
System.out.println("The product is "+Perform_product.performCalculation(a,b));
```

```
System.out.println("The division value is "+Perform_division.performCalculation(a,b));
```

```
}
```

```
public static Calculate performAddition(){
```

```
    Calculate Perform_calculation = (int a,int b)->a+b;
```

```
    return Perform_calculation;
```

```
}
```

```
public static Calculate performSubtraction(){
```

```
    Calculate Perform_calculation = (int a,int b)->a-b;
```

```
    return Perform_calculation;
```

```
}
```

```
public static Calculate performProduct(){
```

```
    Calculate Perform_calculation = (int a,int b)->a*b;
```

```
    return Perform_calculation;
```

```
}
```

```
public static Calculate performDivision(){
```

```
    Calculate Perform_calculation = (int a,int b)->{
```

```
        float c = (float)a;
```

```
        float d = (float)b;
```

```
        return (c/d);
```

```
    };
```

```
    return Perform_calculation;
```

```
}
```

```
}
```

Query DataSet

Query.java*

//Write the required business logic as expected in the question description

```
public class Query {  
    private String queryId;  
    private String queryCategory;  
    private DataSet primaryDataSet;  
    private DataSet secondaryDataSet;  
  
    @Override  
    public String toString()  
    {  
        String g="";  
        g+="Primary data set"+"\\n";  
        g+="Theatre id : "+primaryDataSet.getTheatreId()+"\\n";  
        g+="Theatre name : "+primaryDataSet.getTheatreName()+"\\n";  
        g+="Location : "+primaryDataSet.getLocation()+"\\n";  
        g+="No of Screen : "+primaryDataSet.getNoOfScreen()+"\\n";  
        g+="Ticket Cost : "+primaryDataSet.getTicketCost()+"\\n";  
  
        g+="Secondary data set"+"\\n";  
        g+="Theatre id : "+secondaryDataSet.getTheatreId()+"\\n";  
        g+="Theatre name : "+secondaryDataSet.getTheatreName()+"\\n";  
        g+="Location : "+secondaryDataSet.getLocation()+"\\n";  
        g+="No of Screen : "+secondaryDataSet.getNoOfScreen()+"\\n";  
        g+="Ticket Cost : "+secondaryDataSet.getTicketCost()+"\\n";  
  
        g+="Query id : "+queryId+"\\n";  
        g+="Query category : "+queryCategory+"\\n";  
  
        return g;  
    }  
}
```

```
public class DataSet{

    private String theatreId;

    private String theatreName;

    private String location;

    private int noOfScreen;

    private double ticketCost;


    public double getTicketCost()

    {

        return ticketCost;

    }


    public void setTicketCost(double a)

    {

        ticketCost=a;

    }


    public int getNoOfScreen()

    {

        return noOfScreen;

    }


    public void setNoOfScreen(int a)

    {

        noOfScreen=a;

    }

    public String getLocation ()

    {

        return location;

    }

    public void setLocation(String a)

    {

        location=a;

    }

}
```

```
}
```

```
public String getTheatreName ()
```

```
{
```

```
    return theatreName;
```

```
}
```

```
public void setTheatreName(String a)
```

```
{
```

```
    theatreName=a;
```

```
}
```

```
public String getTheatreId ()
```

```
{
```

```
    return theatreId;
```

```
}
```

```
public void setTheatreId(String a)
```

```
{
```

```
    theatreId=a;
```

```
}
```

```
}
```

```
public void setSecondaryDataSet(DataSet pD)
```

```
{
```

```
    this.secondaryDataSet=pD;
```

```
}
```

```
public DataSet getSecondaryDataSet()
```

```
{
```

```
    return this.secondaryDataSet;
```

```
}
```

```
public void setPrimaryDataSet(DataSet pD)
```

```
{
```

```
    this.primaryDataSet=pD;
```

```
}
```



```

public DataSet getPrimaryDataSet()
{
    return this.primaryDataSet;
}

public void setQueryId (String queryId)
{
    this.queryId=queryId;
}

public void setQueryCategory(String queryCategory)
{
    this.queryCategory=queryCategory;
}

public String getQueryId()
{
    return this.queryId;
}

public String getQueryCategory()
{
    return this.queryCategory;
}
}

```

TestApplication.java*

```

import java.util.*;

public class TestApplication {

    //Write the required business logic as expected in the question description

    public static void main (String[] args) {

        Scanner sc= new Scanner (System.in);

        Query q= new Query();

        Query.DataSet pd= q.new DataSet();

        Query.DataSet sd= q.new DataSet();

        System.out.println("Enter the Details for primary data set");

        System.out.println("Enter the theatre id");
    }
}

```

```
pd.setTheatreId(sc.nextLine());
System.out.println("Enter the theatre name");
pd.setTheatreName(sc.nextLine());
System.out.println("Enter the location");
pd.setLocation(sc.nextLine());
System.out.println("Enter the no of screens");
pd.setNoOfScreen(sc.nextInt());
System.out.println("Enter the ticket cost");
pd.setTicketCost(sc.nextDouble());

System.out.println("Enter the Details for secondary data set");
System.out.println("Enter the theatre id");
String id2=sc.next();
// System.out.println(id2);
sd.setTheatreId(id2);
System.out.println("Enter the theatre name");
sc.nextLine();
sd.setTheatreName(sc.nextLine());
System.out.println("Enter the location");
String gll=sc.nextLine();
sd.setLocation(gll);
System.out.println("Enter the no of screens");

// System.out.println(gll);
// String pp=sc.nextLine();
// System.out.println(pp);
sd.setNoOfScreen(sc.nextInt());
System.out.println("Enter the ticket cost");
sd.setTicketCost(sc.nextDouble());
sc.nextLine();
System.out.println("Enter the query id");
q.setQueryId(sc.nextLine());
System.out.println("Enter the query category");
q.setQueryCategory(sc.nextLine());
```

```
q.setSecondaryDataSet(sd);  
q.setPrimaryDataSet(pd);  
System.out.println(q.toString());
```

```
}
```

```
}
```

Retrive Flight Based On Source And Destination

Flight.java*

```
public class Flight {  
  
    private int flightId;  
    private String source;  
    private String destination;  
    private int noOfSeats;  
    private double flightFare;  
    public int getFlightId() {  
        return flightId;  
    }  
    public void setFlightId(int flightId) {  
        this.flightId = flightId;  
    }  
    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;  
    }  
    public int getNoOfSeats() {  
        return noOfSeats;  
    }  
    public void setNoOfSeats(int noOfSeats) {  
        this.noOfSeats = noOfSeats;  
    }  
}
```

```

    public double getFlightFare() {
        return flightFare;
    }

    public void setFlightFare(double flightFare) {
        this.flightFare = flightFare;
    }

    public Flight(int flightId, String source, String destination,
        int noOfSeats, double flightFare) {
        super();
        this.flightId = flightId;
        this.source = source;
        this.destination = destination;
        this.noOfSeats = noOfSeats;
        this.flightFare = flightFare;
    }

}

```

FlightManagement.java*

```

import java.util.ArrayList;
import java.sql.*;

```

```

public class FlightManagementSystem {

```

```

    public ArrayList<Flight> viewFlightBySourceDestination(String source, String destination){

```

```

        ArrayList<Flight> flightList = new ArrayList<Flight>();

```

```

        try{

```

```

            Connection con = DB.getConnection();

```

```

            String query="SELECT * FROM flight WHERE source= '" + source + "' AND destination= '" + destination + "' ";

```

```
Statement st=con.createStatement();
```

```
ResultSet rst= st.executeQuery(query);
```

```
while(rst.next()){
```

```
    int flightId= rst.getInt(1);
```

```
    String src=rst.getString(2);
```

```
    String dst=rst.getString(3);
```

```
    int noofseats=rst.getInt(4);
```

```
    double flightfare=rst.getDouble(5);
```

```
        flightList.add(new Flight(flightId, src, dst, noofseats, flightfare));
```

```
    }
```

```
}catch(ClassNotFoundException | SQLException e){
```

```
    e.printStackTrace();
```

```
}
```

```
return flightList;
```

```
}
```

```
}
```

DB.java*

```
import java.io.FileInputStream;
```

```
import java.io.IOException;
```

```
import java.sql.Connection;
```

```
import java.sql.DriverManager;
```

```
import java.sql.SQLException;
```

```
import java.util.Properties;
```

```
public class DB {
```

```
    private static Connection con = null;
```

```
    private static Properties props = new Properties();
```

//ENSURE YOU DON'T CHANGE THE BELOW CODE WHEN YOU SUBMIT

```
    public static Connection getConnection() throws ClassNotFoundException, SQLException {
        try{

            FileInputStream fis = null;

            fis = new FileInputStream("database.properties");
            props.load(fis);

            // load the Driver Class

            Class.forName(props.getProperty("DB_DRIVER_CLASS"));

            // create the connection now

            con =
            DriverManager.getConnection(props.getProperty("DB_URL"),props.getProperty("DB_USERNAME"),props.getProperty("DB_PASSWORD"));
        }
        catch(IOException e){
            e.printStackTrace();
        }

        return con;
    }
}
```

Main.java*

```
import java.util.Scanner;

import java.util.ArrayList;

public class Main{

    public static void main(String[] args){

        Scanner sc=new Scanner(System.in);

        System.out.println("Enter the source");

        String source=sc.next();

        System.out.println("Enter the destination");

        String destination=sc.next();
```

```
FlightManagementSystem fms= new FlightManagementSystem();
ArrayList<Flight> flightList=fms.viewFlightBySourceDestination(source,destination);
if(flightList.isEmpty()){
    System.out.println("No flights available for the given source and destination");
    return;
}
System.out.println("Flightid Noofseats Flightfare");
for(Flight flight : flightList){
    System.out.println(flight.getFlightId()+" "+flight.getNoOfSeats()+" "+flight.getFlightFare());
}

}

}
```


Silver Health Plan Insurance

FamilyInsurancePolicy.java

```
public class FamilyInsurancePolicy extends InsurancePolicies{

    public FamilyInsurancePolicy(String clientName, String policyId, int age, long mobileNumber, String emailId) {
        super(clientName, policyId, age, mobileNumber, emailId);
    }

    public boolean validatePolicyId()
    {
        int count=0;
        if(policyId.contains("FAMILY"));
        count++;
        char ch[]=policyId.toCharArray();
        for(int i=6;i<9;i++)
        {
            if(ch[i]>='0' && ch[i]<='9')
                count++;
        }
        if(count==4)
            return true;
        else
            return false;
    }

    public double calculateInsuranceAmount(int months, int no_of_members)
    {
        double amount=0;
        if(age>=5 && age<=25)
            amount=2500*months*no_of_members;
        else if (age>25 && age<60)
            amount=5000*months*no_of_members;
        else if (age>=60)
            amount=10000*months*no_of_members;
```

```

        return amount;
    }

}

```

IndividualInsurancePolicy.java*

```

public class IndividualInsurancePolicy extends InsurancePolicies{

    public IndividualInsurancePolicy(String clientName, String policyId, int age, long mobileNumber, String
    emailId) {

        super(clientName, policyId, age, mobileNumber, emailId);

    }

    public boolean validatePolicyId()
    {

        int count=0;
        if(policyId.contains("SINGLE"));
        count++;
        char ch[]=policyId.toCharArray();
        for(int i=6;i<9;i++)
        {

            if(ch[i]>='0' && ch[i]<='9')
                count++;

        }
        if(count==4)
            return true;
        else
            return false;
    }

    public double calculateInsuranceAmount(int months)
    {

        double amount=0;

```

```

        if(age>=5 && age<=25)
            amount=2500*months;
        else if (age>25 && age<60)
            amount=5000*months;
        else if (age>=60)
            amount=10000*months;
        return amount;
    }

}

```

InsurancePolicies.java*

```

public class InsurancePolicies {
    protected String clientName;
    protected String policyId;
    protected int age;
    protected long mobileNumber;
    protected String emailId;
    public String getClientName() {
        return clientName;
    }
    public void setClientName(String clientName) {
        this.clientName = clientName;
    }
    public String getPolicyId() {
        return policyId;
    }
    public void setPolicyId(String policyId) {
        this.policyId = policyId;
    }
}

    public int getAge() {
        return age;
    }

    public void setAge(int age) {

```

```

        this.age = age;
    }

    public long getMobileNumber() {
        return mobileNumber;
    }

    public void setMobileNumber(long mobileNumber) {
        this.mobileNumber = mobileNumber;
    }

    public String getEmailId() {
        return emailId;
    }

    public void setEmailId(String emailId) {
        this.emailId = emailId;
    }

    public InsurancePolicies(String clientName, String policyId, int age, long mobileNumber, String emailId) {
        super();
        this.clientName = clientName;
        this.policyId = policyId;
        this.age = age;
        this.mobileNumber = mobileNumber;
        this.emailId = emailId;
    }
}

```

SeniorCitizenPolicy.java*

```

public class SeniorCitizenPolicy extends InsurancePolicies{

    public SeniorCitizenPolicy(String clientName, String policyId, int age, long mobileNumber, String emailId) {
        super(clientName, policyId, age, mobileNumber, emailId);
    }

    public boolean validatePolicyId()
    {
        int count=0;
        if(policyId.contains("SENIOR"));
    }
}

```

```

        count++;
        char ch[]=policyId.toCharArray();
        for(int i=6;i<9;i++)
        {
            if(ch[i]>='0' && ch[i]<='9')
                count++;
        }
        if(count==4)
            return true;
        else
            return false;
    }

    public double calculateInsuranceAmount(int months, int no_of_members)
    {
        double amount=0;
        if(age>=5 && age<60)
            amount=0;
        else if (age>=60)
            amount=10000*months*no_of_members;
        return amount;
    }
}

```

UserInterface.java*

```

import java.util.Scanner;

public class UserInterface {

    public static void main(String[] args)
    {

```

```

Scanner sc=new Scanner(System.in);

System.out.println("Enter Client name");

String name=sc.next();

System.out.println("Enter Policy Id");

String id=sc.next();

System.out.println("Enter Client age");

int age=sc.nextInt();

System.out.println("Enter mobile number");

long mnum=sc.nextLong();

System.out.println("Enter Email Id");

String email=sc.next();


InsurancePolicies policy=new InsurancePolicies(name,id,age,mnum,email);

System.out.println("Enter the months");

int month=sc.nextInt();


double amount=0;

if(id.contains("SINGLE"))
{
    IndividualInsurancePolicy g=new IndividualInsurancePolicy(name,id,age,mnum,email);

    if(g.validatePolicyId())
    {
        //System.out.println(g.validatePolicyId());

        amount=g.calculateInsuranceAmount(month);

        System.out.println("Name :"+name);

        System.out.println("Email Id :"+email);

        System.out.println("Amount to be paid :"+amount);

    }
    else
    {
        System.out.println("Provide valid Policy Id");

    }
}
}

```

```

else if(id.contains("FAMILY"))
{
    FamilyInsurancePolicy g=new FamilyInsurancePolicy(name,id,age,mnum,email);
    if(g.validatePolicyId())
    {
        System.out.println("Enter number of members");
        int num=sc.nextInt();
        amount=g.calculateInsuranceAmount(month,num);
        System.out.println("Name :"+name);
        System.out.println("Email Id :"+email);
        System.out.println("Amount to be paid :"+amount);
    }
    else
    {
        System.out.println("Provide valid Policy Id");
    }
}

else if(id.contains("SENIOR"))
{
    SeniorCitizenPolicy g=new SeniorCitizenPolicy(name,id,age,mnum,email);
    if(g.validatePolicyId())
    {
        System.out.println("Enter number of members");
        int num=sc.nextInt();
        amount=g.calculateInsuranceAmount(month,num);
        System.out.println("Name :"+name);
        System.out.println("Email Id :"+email);
        System.out.println("Amount to be paid :"+amount);
    }
    else
    {
        System.out.println("Provide valid Policy Id");
    }
}

```

else

System.out.println("Provide valid Policy Id");

}

}

Travel Request System

TraveRequestDao.java*

```
package com.cts.travelrequest.dao;
```

```
import java.io.FileInputStream;
```

```
import java.io.IOException;
```

```
import java.sql.Connection;
```

```
import java.sql.DriverManager;
```

```
import java.sql.*;
```

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

```
//import java.util.Properties;
```

```
import com.cts.travelrequest.vo.TravelRequest;
```

```
class DB{
```

```
    private static Connection con = null;
```

```
    private static Properties props = new Properties();
```

```
    //ENSURE YOU DON'T CHANGE THE BELOW CODE WHEN YOU SUBMIT
```

```
    public static Connection getConnection() throws ClassNotFoundException, SQLException {
```

```
        try{
```

```
            FileInputStream fis = null;
```

```
            fis = new FileInputStream("resource/database.properties");
```

```
            props.load(fis);
```

```
            // load the Driver Class
```

```
            Class.forName(props.getProperty("DB_DRIVER_CLASS"));
```

```
            // create the connection now
```

```

        con =
DriverManager.getConnection(props.getProperty("DB_URL"),props.getProperty("DB_USERNAME"),props.getPropert
y("DB_PASSWORD"));
    }
    catch(IOException e){
        e.printStackTrace();
    }

    return con;
}
}

```

```

/**
 * Method to get travel details based on source and destination city      *
 * @return list
 */
public class TravelRequestDao{

    // public PreparedStatement prepareStatement(String query) throws SQLException{}

    public static List<TravelRequest> getTravelDetails(String sourceCity, String destinationCity) {

        List<TravelRequest> travel=new ArrayList<>();

        try{

            Connection con=DB.getConnection();

            String query="Select * from t_travelrequest where sourceCity=? and destinationCity=?;";

            PreparedStatement ps=con.prepareStatement(query);

            ps.setString(1,sourceCity);

            ps.setString(2,destinationCity);

            ResultSet rs=ps.executeQuery();

            while(rs.next()){

                String tid=rs.getString("travelReqId");

                java.sql.Date date=rs.getDate("travelDate");

                String apstat=rs.getString("approvalStatus");

                String sour=rs.getString("sourceCity");

                String des=rs.getString("destinationCity");

                double cost=rs.getDouble("travelCost");

                TravelRequest tr=new TravelRequest(tid,date,apstat,sour,des,cost);

```

```

        travel.add(tr);
    }
}
catch(ClassNotFoundException e){
    e.printStackTrace();
}
catch(SQLException e ){
    e.printStackTrace();
}

    return travel; //TODO change this return value

}

/**
 * Method to calculate total travel cost based approvalStatus
 * @return list
 */
public static double calculateTotalTravelCost(String approvalStatus) {
    double amount=0;
    try{
        Connection con=DB.getConnection();
        String query="select travelCost from t_travelrequest where approvalStatus=?";
        PreparedStatement ps1=con.prepareStatement(query);
        ps1.setString(1,approvalStatus);
        ResultSet rs1=ps1.executeQuery();
        while(rs1.next()){
            amount+=rs1.getDouble("travelCost");
        }
    }
    catch(ClassNotFoundException e){
        e.printStackTrace();
    }
}

```

```

        catch(SQLException e){
            e.printStackTrace();
        }

        return amount; //TODO change this return value
    }
}

```

TravelRequestService.java*

```
package com.cts.travelrequest.service;
```

```
import java.util.List;
```

```
import com.cts.travelrequest.dao.TravelRequestDao;
```

```
import com.cts.travelrequest.vo.TravelRequest;
```

```
public class TravelRequestService {
```

```
    /**
```

```
    * Method to validate approval status
```

```
    *
```

```
    * @return status
```

```
    */
```

```
    public String validateApprovalStatus(String approvalStatus) {
```

```
        if(approvalStatus.equalsIgnoreCase("Approved") || approvalStatus.equalsIgnoreCase("Pending")){
```

```
            return "valid";
```

```
        }
```

```
        return "invalid"; //TODO change this return value
```

```
    }
```

```
    /**
```

```
    * Method to validate source and destination city
```

```
    *
```

```
    * @return status
```

```
*/
```

```
public String validateSourceAndDestination(String sourceCity, String destinationCity) {
```

```
    if(!sourceCity.equalsIgnoreCase(destinationCity)){
```

```
        if(sourceCity.equalsIgnoreCase("Pune") || sourceCity.equalsIgnoreCase("Mumbai") ||  
sourceCity.equalsIgnoreCase("Chennai") || sourceCity.equalsIgnoreCase("Bangalore") ||  
sourceCity.equalsIgnoreCase("Hydrabad")){
```

```
            if(destinationCity.equalsIgnoreCase("Pune") ||  
destinationCity.equalsIgnoreCase("Mumbai") || destinationCity.equalsIgnoreCase("Chennai") ||  
destinationCity.equalsIgnoreCase("Bangalore") || destinationCity.equalsIgnoreCase("Hydrabad")){
```

```
                return "valid";
```

```
            }
```

```
        else{
```

```
            return "invalid";
```

```
        }
```

```
    }
```

```
    else{
```

```
        return "invalid";
```

```
    }
```

```
}
```

```
else{
```

```
    return "invalid";
```

```
}
```

```
}
```

```
/**
```

```
 * Method to invoke getTravelDetails method of TravelRequestDao class
```

```
 *
```

```
 * @return listOfTravelRequest
```

```
 */
```

```
public List<TravelRequest> getTravelDetails(String sourceCity, String destinationCity) {
```

```
    if(this.validateSourceAndDestination(sourceCity,destinationCity).contentEquals("valid")){
```

```
        return TravelRequestDao.getTravelDetails(sourceCity,destinationCity);
```

```
    }
```

```
    else{
```

```
        return null;
```

```
    }
```

```

    }

    /**
     * Method to invoke calculateTotalTravelCost method of TravelRequestDao class
     *
     * @return totalCost
     */
    public double calculateTotalTravelCost(String approvalStatus) {
        if(this.validateApprovalStatus(approvalStatus).equals("valid")){
            return TravelRequestDao.calculateTotalTravelCost(approvalStatus);
        }
        else{
            return -1;
        }
    }
}

```

SkeletonValidator.java*

```

package com.cts.travelrequest.skeletonvalidator;

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

/**
 * @author t-aarti3
 *
 * 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("com.cts.travelrequest.service.TravelRequestService");
        validateClassName("com.cts.travelrequest.vo.TravelRequest");
    }
}

```

```

        validateMethodSignature(

            "validateApprovalStatus:java.lang.String,validateSourceAndDestination:java.lang.String,getTravelDetails:java
            .util.List,calculateTotalTravelCost:double",

                "com.cts.travelrequest.service.TravelRequestService");

    }

    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");
}
}
}
}

```

TravelRequest.java*

```

package com.cts.travelrequest.vo;

import java.util.Date;

public class TravelRequest {
    // member variables
    private String travelReqId;
    private Date travelDate;
    private String approvalStatus;
    private String sourceCity;
    private String destinationCity;
    private double travelCost;

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

```

```

}

// parameterized constructor

public TravelRequest(String travelReqId, Date travelDate, String approvalStatus, String sourceCity,
                    String destinationCity, double travelCost) {

    super();

    this.travelReqId = travelReqId;

    this.travelDate = travelDate;

    this.approvalStatus = approvalStatus;

    this.sourceCity = sourceCity;

    this.destinationCity = destinationCity;

    this.travelCost = travelCost;

}

// setter, getter

/**
 * @return the travelReqId
 */
public String getTravelReqId() {

    return travelReqId;

}

/**
 * @param travelReqId
 *      the travelReqId to set
 */
public void setTravelReqId(String travelReqId) {

    this.travelReqId = travelReqId;

}

/**
 * @return the travelDate
 */
public Date getTravelDate() {

    return travelDate;

}

/**

```

```
* @param travelDate
*     the travelDate to set
*/
public void setTravelDate(Date travelDate) {
    this.travelDate = travelDate;
}
/**
* @return the approvalStatus
*/
public String getApprovalStatus() {
    return approvalStatus;
}
/**
* @param approvalStatus
*     the approvalStatus to set
*/
public void setApprovalStatus(String approvalStatus) {
    this.approvalStatus = approvalStatus;
}
/**
* @return the sourceCity
*/
public String getSourceCity() {
    return sourceCity;
}
/**
* @param sourceCity
*     the sourceCity to set
*/
public void setSourceCity(String sourceCity) {
    this.sourceCity = sourceCity;
}
/**
* @return the sourceCity
```

```

    */
    public String getDestinationCity() {
        return destinationCity;
    }
    /**
     * @param destinationCity
     *      the destinationCity to set
     */
    public void setDestinationCity(String destinationCity) {
        this.destinationCity = destinationCity;
    }
    /**
     * @return the travelCost
     */
    public double getTravelCost() {
        return travelCost;
    }
    /**
     * @param travelCost
     *      the travelCost to set
     */
    public void setTravelCost(double travelCost) {
        this.travelCost = travelCost;
    }
}

```

Main.java*

```

package com.cts.travelrequest.main;

import java.sql.*;
import java.util.*;
import java.text.SimpleDateFormat;
import com.cts.travelrequest.service.TravelRequestService;

```

```

import com.cts.travelrequest.skeletonvalidator.SkeletonValidator;

import com.cts.travelrequest.vo.TravelRequest;

public class Main {

    public static void main(String[] args) throws SQLException {

        // CODE SKELETON - VALIDATION STARTS

        // DO NOT CHANGE THIS CODE

        new SkeletonValidator();

        // CODE SKELETON - VALIDATION ENDS

        //TravelRequest tr=new TravelRequest();

        //List<TravelRequest> ltr=new ArrayList<>();

        TravelRequestService service = new TravelRequestService();

        Scanner sc=new Scanner(System.in);

        System.out.println("Enter source city:");

        String sourceCity=sc.next();

        System.out.println("Enter destination city:");

        String destinationCity=sc.next();

        System.out.println("Enter approval status to find total travel cost:");

        String status=sc.next();


        if(service.validateSourceAndDestination(sourceCity,destinationCity).equals("valid")){

            List<TravelRequest> ltr=service.getTravelDetails(sourceCity, destinationCity);

            if(ltr.isEmpty()){

                System.out.println("No travel request raised for given source and destination cities");

            }

            else{

                for(TravelRequest t:ltr){

                    SimpleDateFormat sd= new SimpleDateFormat("dd-MMM-YYYY");

                    String d=sd.format(t.getTravelDate());

                    System.out.println(t.getTravelReqId()+"\t| "+d+"\t| "+t.getApprovalStatus()+"\t| "+t.getSourceCity()+"\t| "+t.getDestinationCity()+"\t| "+t.getTravelCost());

                }

            }

        }

    }

}

```

```
else{  
    System.out.println("Provide correct source and destination city");  
}  
if(service.validateApprovalStatus(status).contentEquals("valid")){  
    System.out.println(service.calculateTotalTravelCost(status));  
}  
else{  
    System.out.println("Provide valid approval status");  
}
```

```
}
```

```
}
```