```
public class CreditCardPayment {

	public Map<Integer,Map> getcreditCardFineDetails(String filePath) throws
creditCardCalculatorException{

		Map<String, CreditCardVO> subMap1=new HashMap<String, CreditCardVO>();

		Map<String, CreditCardVO> subMap2=new HashMap<String, CreditCardVO>();

		Map<Integer, Map> mainMap=new HashMap<Integer, Map>();

		InputStream in=null;

		String data=null;

		long daysDiff=0;

		CreditCardVO creditCardVO=null;

		List<String> dataInFile=new ArrayList<String>();

		List<CreditCardVO> cardDetails=new ArrayList<CreditCardVO>();

		List<CreditCardVO> sortedCardDetails=new ArrayList<CreditCardVO>();

		Set<String> visaCreditCards=new TreeSet<String>();

		Set<String> amexCreditCards=new TreeSet<String>();

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

		sdf.setLenient(false);

		@SuppressWarnings("unused")

		Date formatChecker1=null;

		Date formatChecker2=null;

		File file=new File(filePath);

		if(file.isFile() && file.exists()){

			//Path path=Paths.get(filePath);

			try {

				//in=Files.newInputStream(path);

				in=new FileInputStream(file);

				BufferedReader bufReader=new BufferedReader(new
InputStreamReader(in));
```

```
while ((data = bufReader.readLine()) != null) {
	dataInFile.add(data);
}
for(String s:dataInFile){
	if(!s.isEmpty() && null!=s){
		String[] indCardDetails=s.split("\\|");

		creditCardVO=new CreditCardVO();
		creditCardVO.setCustomerName(indCardDetails[1]);
		if(null!=indCardDetails[0] &&
!indCardDetails[0].isEmpty() && indCardDetails[0].length()==9 && indCardDetails[0].startsWith("4")){
			visaCreditCards.add(indCardDetails[0]);

	creditCardVO.setCardNumber(indCardDetails[0]);

		}else if(null!=indCardDetails[0] &&
!indCardDetails[0].isEmpty() &&  indCardDetails[0].length()==10 &&
(indCardDetails[0].startsWith("34")||indCardDetails[0].startsWith("37"))){
			amexCreditCards.add(indCardDetails[0]);

	creditCardVO.setCardNumber(indCardDetails[0]);
		} else {
			throw new
creditCardCalculatorException("Invalid Card Number");
		}
		if(!indCardDetails[2].isEmpty()){

	creditCardVO.setBillAmount(Integer.valueOf(indCardDetails[2]));}
		if(indCardDetails[3].length()==10 &&
indCardDetails[4].length()==10){
			formatChecker1=sdf.parse(indCardDetails[3]);
			formatChecker2=sdf.parse(indCardDetails[4]);
```

```
                                                    if((null!=indCardDetails[3] &&
!indCardDetails[3].isEmpty() && (sdf.format(formatChecker1).equals(indCardDetails[3]))||
                                                                ((null!=indCardDetails[4] &&
!indCardDetails[4].isEmpty() && (sdf.format(formatChecker2).equals(indCardDetails[4])))){

        creditCardVO.setDueDate(sdf.parse(indCardDetails[3]));

        creditCardVO.setPaymentDate(sdf.parse(indCardDetails[4]));
                                                    }else{
                                                            throw new
creditCardCalculatorException("");
                                                    }
                                                }else{
                                                    throw new creditCardCalculatorException("");
                                                }
                                                //
        System.out.println(sdf.format(sdf.parse(indCardDetails[3])));

                                                    cardDetails.add(creditCardVO);
                                            }
                                        }

                                        for(CreditCardVO credVo1:cardDetails){
                                            for(CreditCardVO credVo2:cardDetails){

        if(credVo1.getCardNumber().equals(credVo2.getCardNumber()) &&
credVo2.getPaymentDate().compareTo(credVo1.getPaymentDate())==1){
                                                    credVo1=credVo2;
                                                }
                                            }
                                            sortedCardDetails.add(credVo1);
                                        }
```

```
                        //System.out.println("Sorted Set::"+sortedCardDetails);

                        for(String cardNo:visaCreditCards){
                                for(CreditCardVO creditCardVo:sortedCardDetails){
                                        if(cardNo.equals(creditCardVo.getCardNumber())){
                                                Date d1=creditCardVo.getDueDate();
                                                Date d2=creditCardVo.getPaymentDate();
                                                if(d1.after(d2) ||d1.equals(d2)){
                                                        creditCardVo.setFine(0);
                                                        creditCardVo.setCreditGrade('A');
                                                }
                                                else if(d1.before(d2)){
                                                        daysDiff=(d2.getTime()-
d1.getTime())/(24*60*60*1000);
                                                        if(daysDiff<=5 && daysDiff>0){

        creditCardVo.setFine((creditCardVo.getBillAmount()*10)/100);
                                                        }else if(daysDiff>5){

        creditCardVo.setFine((creditCardVo.getBillAmount()*20)/100);
                                                        }
                                                        creditCardVo.setCreditGrade('B');
                                                }
                                                subMap1.put(cardNo, creditCardVo);
                                        }
                                }
                        }
                        for(String cardNo:amexCreditCards){
                                for(CreditCardVO creditCardVo:sortedCardDetails){
```

```
                                        if(cardNo.equals(creditCardVo.getCardNumber())){

                                                Date d1=creditCardVo.getDueDate();
                                                Date d2=creditCardVo.getPaymentDate();
                                                if(d1.after(d2) ||d1.equals(d2)){
                                                        creditCardVo.setFine(0);
                                                        creditCardVo.setCreditGrade('A');
                                                }
                                                else if(d1.before(d2)){
                                                        daysDiff=(d2.getTime()-
d1.getTime())/(24*60*60*1000);
                                                        if(daysDiff<=5 && daysDiff>0){

        creditCardVo.setFine((creditCardVo.getBillAmount()*10)/100);
                                                        }else if(daysDiff>5){

        creditCardVo.setFine((creditCardVo.getBillAmount()*20)/100);
                                                        }
                                                        creditCardVo.setCreditGrade('B');
                                                }
                                                subMap2.put(cardNo, creditCardVo);
                                        }
                                }
                        }
                        mainMap.put(1, subMap1);
                        mainMap.put(2, subMap2);
                } catch (IOException e) {
                        throw new creditCardCalculatorException("Invalid File or path");
                } catch (ParseException e) {
                        throw new creditCardCalculatorException("Invalid Date Format");
```

```
                }
            }else{
                throw new creditCardCalculatorException("");
            }
            return mainMap;
        }

        public static void main(String[] args) {
            CreditCardPayment cp = new CreditCardPayment();
            try {

        System.out.println(cp.getcreditCardFineDetails("D:\\workspace\\SVN_Restructure_06-
16\\CreditCardPayment\\src\\input.txt"));
            } catch (creditCardCalculatorException e) {
                // TODO Auto-generated catch block
                e.getMessage();
                e.printStackTrace();
            }
        }
}

class creditCardCalculatorException extends Exception {
        String message;

        public creditCardCalculatorException(String message) {
            this.message = message;
        }
}
```

```
class CreditCardVO implements Comparable<CreditCardVO> {
	String customerName;
	String cardNumber;
	Integer billAmount;
	Date dueDate;
	Date paymentDate;
	int fine;
	char creditGrade;
	/**
	 * @return the creditCardType
	 */
	public char getCreditGrade() {
		return creditGrade;
	}
	/**
	 * @param creditCardType the creditCardType to set
	 */
	public void setCreditGrade(char creditCardType) {
		this.creditGrade = creditCardType;
	}
	/**
	 * @return the fine
	 */
	public int getFine() {
		return fine;
	}
	/**
	 * @param fine the fine to set
	 */
```

```
public void setFine(int fine) {
		this.fine = fine;
}
/**
 * @return the customerName
 */
public String getCustomerName() {
		return customerName;
}
/**
 * @param customerName the customerName to set
 */
public void setCustomerName(String customerName) {
		this.customerName = customerName;
}
/**
 * @return the cardNumber
 */
public String getCardNumber() {
		return cardNumber;
}
/**
 * @param cardNumber the cardNumber to set
 */
public void setCardNumber(String cardNumber) {
		this.cardNumber = cardNumber;
}
/**
 * @return the billAmount
```

```
 */
public Integer getBillAmount() {
	return billAmount;
}
/**
 * @param billAmount the billAmount to set
 */
public void setBillAmount(Integer billAmount) {
	this.billAmount = billAmount;
}
/**
 * @return the dueDate
 */
public Date getDueDate() {
	return dueDate;
}
/**
 * @param dueDate the dueDate to set
 */
public void setDueDate(Date dueDate) {
	this.dueDate = dueDate;
}
/**
 * @return the paymentDate
 */
public Date getPaymentDate() {
	return paymentDate;
}
/**
```

```
 * @param paymentDate the paymentDate to set
 */
public void setPaymentDate(Date paymentDate) {
	this.paymentDate = paymentDate;
}
/* (non-Javadoc)
 * @see java.lang.Object#toString()
 */
@Override
public String toString() {
	StringBuilder builder = new StringBuilder();
	builder.append("CreditCardVO [customerName=");
	builder.append(customerName);
	builder.append(", cardNumber=");
	builder.append(cardNumber);
	builder.append(", billAmount=");
	builder.append(billAmount);
	builder.append(", dueDate=");
	builder.append(dueDate);
	builder.append(", paymentDate=");
	builder.append(paymentDate);
	builder.append(", fine=");
	builder.append(fine);
	builder.append(", creditGrade=");
	builder.append(creditGrade);
	builder.append("]");
	return builder.toString();
}
@Override
```

```
	public int compareTo(CreditCardVO vo) {
		return this.paymentDate.compareTo(vo.getPaymentDate());
	}
}
```

LoantrustInsurance:

return map --> Map<Integer,Map<String,List<LoanTrustVo>>>

POLICYNUMBER pannumber startdate period LoanRequestAmount Policyamt

FST-1234-200000;FS123; 26/07/2017;48;400000;40000

Policynumber = plocyType-policynumber-assuredsum

validation

all fileds should be mandatory

policycode should be FSI/NRS

policynuber should be non zero

policynumber should be a number

policyType should be FST or NRS

policyamt should not be greater than assuredamt

start date should be dd/MM/yyyy

pan number should be start with FS and followd by three digit number

key1

//Not exactly from the exam requirement

if policy Type FST and request loan amt > 40% assuredsum ELG

if policy Type FST and request loan amt > 60% assuredsum NELG

if policy Type NRS and request loan amt > 60% assuredsum ELG

if policy Type NRS and request loan amt > 60% assuredsum and < 70%policyamt ELG

if policy Type NRS and request loan amt > 60% assuredsum and > 70%policyamt NELG

Map1("ELG",ELGlist)

Map1("NELG",NELGlist)

if NELG then netpayment should be requestloan amt

if ELG then netpayment should be calculate as follows

Netpayment= caluculateassuredamt+interestamt

caluculateassuredamt=assuredamt/period

interestamt=(policyamt*1.2*loanperiod)/period

loanperiod(have to calculate in months)=enddate-sanctiondate

enddate=startdate+(period ----> given in months)

Duplicate

if policynumber is repeat then all the records should added in list

Invalid

if the loanrequest amt is greater than sumassured that that is invalid record

Map2("DUP",dupilcatelist)

Map2("INV",invalidlist)

Output Map

Map(1,Map1)

Map(2,Map2)

*/

```
public class LoanTrustInsurance {

	public static void main(String[] args) {
		LoanTrustInsurance loanTrustInsurance = new LoanTrustInsurance();
		String filePath = "D:\\loaninput.txt";
		try {
			System.out.println(loanTrustInsurance.loanTrustDetails(filePath,
"02/07/2017"));
		} catch (LoanTrustException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
	}
```

```
public Map<Integer, Map<String, List<LoanTrustVo>>> loanTrustDetails(String filePath, String sanctionDate)
                throws LoanTrustException {
        Map<Integer, Map<String, List<LoanTrustVo>>> retMap = new HashMap<>();
        Map<String, List<LoanTrustVo>> innermap1 = new HashMap<>();
        Map<String, List<LoanTrustVo>> innermap2 = new HashMap<>();
        List<LoanTrustVo> loanTrustList = new ArrayList<>();
        File file = new File(filePath);
        if (!file.exists()) {
                throw new LoanTrustException("File should not be empty");
        }
        try {
                BufferedReader br = new BufferedReader(new FileReader(file));
                String line = null;
                while ((line = br.readLine()) != null) {
                        LoanTrustVo loanTrustVo = new LoanTrustVo();

                        String[] loanArray = line.split(";");

                        if (loanArray.length != 6) {
                                throw new LoanTrustException("All fields are Mandatory");
                        }

                        for (String str : loanArray) {
                                if (str.isEmpty() || str.equals(null)) {
                                        throw new LoanTrustException("input field should not be a null or empty");
                                }
                        }
```

```
String[] plcysplit = loanArray[0].split("-");
Float assuredamt = Float.parseFloat(plcysplit[2]);

if (loanArray[0].matches("((FST)|(NRS))[-]{1}[1-9]+[-]{1}[0-9]+")) {
	loanTrustVo.setPolicyNumber(loanArray[0]);
} else {
	throw new LoanTrustException("Policy number is not in the
specified Format");
}

if (loanArray[1].matches("(FS)[0-9]{3}")) {
	loanTrustVo.setPanNumber(loanArray[1]);
} else {
	throw new LoanTrustException("PAN number is not in the
specified Format");
}

SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
sdf.setLenient(false);
Date d = sdf.parse(loanArray[2]);
loanTrustVo.setStartDate(d);

loanTrustVo.setPeriod(Integer.parseInt(loanArray[3]));

loanTrustVo.setLoanRequestAmount(Float.parseFloat(loanArray[4]));

if (assuredamt > Float.parseFloat(loanArray[5])) {
	loanTrustVo.setPolicyamt(Float.parseFloat(loanArray[5]));
```

```
                } else {
                        throw new LoanTrustException("Policyamt should not be
greater than assuredamt");
                }

                loanTrustList.add(loanTrustVo);
        }

        List<LoanTrustVo> elgList = new ArrayList<>();
        List<LoanTrustVo> nonElgList = new ArrayList<>();
        List<LoanTrustVo> dupList = new ArrayList<>();
        List<LoanTrustVo> invList = new ArrayList<>();

        // To Find Duplicate Elements
        Map<Integer, LoanTrustVo> dupMap = new HashMap<Integer, LoanTrustVo>();

        for (LoanTrustVo loanTrustVo : loanTrustList) {

                Calendar c = Calendar.getInstance();
                c.setTime(loanTrustVo.getStartDate());
                c.add(Calendar.MONTH, loanTrustVo.getPeriod());
                Date endDate = c.getTime();
                loanTrustVo.setEndDate(endDate);

                String[] plcysplit = (loanTrustVo.getPolicyNumber()).split("-");
                String plcyTyp = plcysplit[0];
                Integer plcycode = Integer.parseInt(plcysplit[1]);
                Float assuredamt = Float.parseFloat(plcysplit[2]);
```

```
                        if (dupMap.containsKey(plcycode)) {
                                dupList.add(dupMap.get(plcycode));
                        } else {
                                dupMap.put(plcycode, loanTrustVo);
                        }

                        Float reqLnAmnt = loanTrustVo.getLoanRequestAmount();
                        Float plcyAmnt = loanTrustVo.getPolicyamt();

                        if (plcyTyp.equals("FST") && reqLnAmnt > (0.4 * assuredamt)) {
                                calculateNetPaymentEligible(loanTrustVo, sanctionDate,
assuredamt);
                                elgList.add(loanTrustVo);
                        }

                        if (plcyTyp.equals("FST") && reqLnAmnt > (0.6 * assuredamt)) {
                                calculateNetPaymentNonEligible(loanTrustVo, reqLnAmnt);
                                nonElgList.add(loanTrustVo);
                        }

                        if (plcyTyp.equals("NRS") && reqLnAmnt > (0.6 * assuredamt)) {
                                calculateNetPaymentEligible(loanTrustVo, sanctionDate,
assuredamt);
                                elgList.add(loanTrustVo);
                        }

                        if (plcyTyp.equals("NRS") && reqLnAmnt > (0.6 * assuredamt) &&
reqLnAmnt < (0.7 * plcyAmnt)) {
                                calculateNetPaymentEligible(loanTrustVo, sanctionDate,
assuredamt);
```

```
                                elgList.add(loanTrustVo);
                        }

                        if (plcyTyp.equals("NRS") && reqLnAmnt > (0.6 * assuredamt) &&
reqLnAmnt > (0.7 * plcyAmnt)) {
                                calculateNetPaymentNonEligible(loanTrustVo, reqLnAmnt);
                                nonElgList.add(loanTrustVo);
                        }

                        if (reqLnAmnt > assuredamt) {
                                invList.add(loanTrustVo);
                        }
                }

                innermap1.put("ELG", elgList);
                innermap1.put("NELG", nonElgList);
                innermap2.put("DUP", dupList);
                innermap2.put("INV", invList);

                retMap.put(1, innermap1);
                retMap.put(2, innermap2);

        } catch (FileNotFoundException e) {
                // TODO Auto-generated catch block
                throw new LoanTrustException(e);
        } catch (IOException e) {
                // TODO Auto-generated catch block
                throw new LoanTrustException(e);
        } catch (ParseException e) {
```

```
                // TODO Auto-generated catch block
                throw new LoanTrustException(e);
            }

            return retMap;

        }

        private void calculateNetPaymentEligible(LoanTrustVo loanTrustVo, String sanctionDate, Float
assuredamt) {
            // TODO Auto-generated method stub

            int loanPeriod = 0;
            SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
            Date d = null;
            try {
                d = sdf.parse(sanctionDate);
            } catch (ParseException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            Calendar c1 = Calendar.getInstance();
            c1.setTime(d);
            Calendar c2 = Calendar.getInstance();
            c2.setTime(loanTrustVo.getEndDate());
            int years = c2.get(Calendar.YEAR) - c1.get(Calendar.YEAR);
            int months = c2.get(Calendar.MONTH) - c1.get(Calendar.MONTH);
            int datediff = c2.get(Calendar.DATE) - c1.get(Calendar.DATE);
            if (datediff < 0) {
```

```
                loanPeriod = ((years * 12) + (months))- 1;
            } else {
                loanPeriod = (years * 12) + months;
            }

            Float interestAmount = (float) ((loanTrustVo.getPolicyamt() * 1.2 * loanPeriod) /
(loanTrustVo.getPeriod()));
            Float caluculateassuredamt = (assuredamt / loanTrustVo.getPeriod());
            Float netPaymentValue = interestAmount + caluculateassuredamt;
            loanTrustVo.setNetPaymentValue(netPaymentValue);
        }

        private void calculateNetPaymentNonEligible(LoanTrustVo loanTrustVo, Float reqLnAmnt) {
            // TODO Auto-generated method stub
            loanTrustVo.setNetPaymentValue(reqLnAmnt);
        }
}

class LoanTrustVo {
        private String policyNumber;
        private String panNumber;
        private Date startDate;
        private Integer period;
        private float loanRequestAmount;
        private float Policyamt;
        private Date endDate;
        private float netPaymentValue;

        public String getPolicyNumber() {
```

```
        return policyNumber;
}

public void setPolicyNumber(String policyNumber) {
        this.policyNumber = policyNumber;
}

public String getPanNumber() {
        return panNumber;
}

public void setPanNumber(String panNumber) {
        this.panNumber = panNumber;
}

public Date getStartDate() {
        return startDate;
}

public void setStartDate(Date startDate) {
        this.startDate = startDate;
}

public Integer getPeriod() {
        return period;
}

public void setPeriod(Integer period) {
        this.period = period;
```

```
}

public float getLoanRequestAmount() {
	return loanRequestAmount;
}

public void setLoanRequestAmount(float loanRequestAmount) {
	this.loanRequestAmount = loanRequestAmount;
}

public float getPolicyamt() {
	return Policyamt;
}

public void setPolicyamt(float policyamt) {
	Policyamt = policyamt;
}

public Date getEndDate() {
	return endDate;
}

public void setEndDate(Date endDate) {
	this.endDate = endDate;
}

public float getNetPaymentValue() {
	return netPaymentValue;
}
```

```
	public void setNetPaymentValue(float netPaymentValue) {
		this.netPaymentValue = netPaymentValue;
	}

	@Override
	public String toString() {
		return "LoanTrustVo [policyNumber=" + policyNumber + ", panNumber=" + panNumber
+ ", startDate=" + startDate
				+ ", period=" + period + ", loanRequestAmount=" + loanRequestAmount
+ ", Policyamt=" + Policyamt
				+ ", endDate=" + endDate + ", netPaymentValue=" + netPaymentValue +
"]";
	}

	@Override
	public int hashCode() {
		final int prime = 31;
		int result = 1;
		result = prime * result + Float.floatToIntBits(Policyamt);
		result = prime * result + ((endDate == null) ? 0 : endDate.hashCode());
		result = prime * result + Float.floatToIntBits(loanRequestAmount);
		result = prime * result + Float.floatToIntBits(netPaymentValue);
		result = prime * result + ((panNumber == null) ? 0 : panNumber.hashCode());
		result = prime * result + ((period == null) ? 0 : period.hashCode());
		result = prime * result + ((policyNumber == null) ? 0 : policyNumber.hashCode());
		result = prime * result + ((startDate == null) ? 0 : startDate.hashCode());
		return result;
	}
```

```
	@Override
	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (obj == null)
			return false;
		if (getClass() != obj.getClass())
			return false;
		LoanTrustVo other = (LoanTrustVo) obj;
		if (Float.floatToIntBits(Policyamt) != Float.floatToIntBits(other.Policyamt))
			return false;
		if (endDate == null) {
			if (other.endDate != null)
				return false;
		} else if (!endDate.equals(other.endDate))
			return false;
		if (Float.floatToIntBits(loanRequestAmount) !=
Float.floatToIntBits(other.loanRequestAmount))
			return false;
		if (Float.floatToIntBits(netPaymentValue) !=
Float.floatToIntBits(other.netPaymentValue))
			return false;
		if (panNumber == null) {
			if (other.panNumber != null)
				return false;
		} else if (!panNumber.equals(other.panNumber))
			return false;
		if (period == null) {
			if (other.period != null)
```

```
                        return false;
                } else if (!period.equals(other.period))
                        return false;
                if (policyNumber == null) {
                        if (other.policyNumber != null)
                                return false;
                } else if (!policyNumber.equals(other.policyNumber))
                        return false;
                if (startDate == null) {
                        if (other.startDate != null)
                                return false;
                } else if (!startDate.equals(other.startDate))
                        return false;
                return true;
        }
}

class LoanTrustException extends Exception {
        /**
         *
         */
        private static final long serialVersionUID = 1L;

        public LoanTrustException(String msg) {
                super(msg);
        }

        public LoanTrustException(Throwable throwable) {
                super(throwable);
```

```
        }

        public LoanTrustException(String msg, Throwable throwable) {
                super(msg, throwable);
        }
}
```

```
/*
 * Input.txt (cardNumber|CustomerName|BillAmt|DueDate|PaymentDate)
 4000000000011|nizampatnam|1000|01012017|05012017
 4000000000012|Anusha|2000|01022017|31012017
 340000000001|swetha|3000|01032017|05032017
 350000000002|Sravani|4000|01042017|05042017

 VISA card  -  total 16 digit,  number starts with 4
 AMEX card – total 15 digit, number starts with 34 / 37

 Validations – throw custom exception:
 1.      If input text not available, not able to read (ie. For FileNotFound/IO exceptions)
 2.      If card number invalid (not VISA/AMEX)
 3. Date validation(dd/MM/yyyy)
 4.

 Business logic:
 1.      If customer paid bill amt before due date, then he is categorized as 'A' GRADE otherwise 'B'
 2.      For fine calculation
 a.      VISA CARD
 i.      If payment date – due date <= 5 then  fine = billAmt * 10%
```

i. If payment date – due date > 5 then fine = billAmt * 20%

b. AMEX CARD

i. If payment date – due date <= 5 then fine = billAmt * 10%

ii. If payment date – due date > 5 and billamount <15k then fine = billAmt * 20%

ii. If payment date – due date > 5 and billamount>15k then fine = billAmt * 30%

NOTE: remove duplicates if same record exists, by overriding it based on the latest payment date.

Expected OUTPUT (cardNumber|CustomerName|BillAmt|DueDate|PaymentDate|fine|Grade)

4000000000011|nizampatnam|1000|01-01-2017|05-01-2017|100|A

4000000000012|Anusha|2000|01022017|31012017|200|B

340000000001|swetha|3000|01032017|05032017|300|B

350000000002|Sravani|4000|01042017|05042017|400|B

Map<Integer, Map<String, CardVO>>

1 - 4000000000011 - card detail object

4000000000012- card detail object

2 - 340000000001- card detail object

350000000002- card detail object

*

*

*

* */

```
public class CardDetail {
	public static void main(String[] args) {
		CardDetail cardDetail = new CardDetail();
```

```
		try {
			System.out.println(cardDetail.calculateCardFineDetails("input.txt"));
		} catch (Exception e) {
			e.printStackTrace();
		}
	}

	public Map calculateCardFineDetails(String fileName) throws Exception {
		Map<Integer, Map<String, CreditCardVO>> map = new HashMap<Integer, Map<String,
CreditCardVO>>();
		File file = new File(fileName);
		if (!file.exists()) {
			throw new CreditCardValidationException("Invalid file path");
		}
		try {
			BufferedReader br = new BufferedReader(new FileReader(file));
			Map<String, CreditCardVO> visaMap = new HashMap<String, CreditCardVO>();
			Map<String, CreditCardVO> amexMap = new HashMap<String, CreditCardVO>();
			List<CreditCardVO> visaList = new ArrayList<CreditCardVO>();
			List<CreditCardVO> amexList = new ArrayList<CreditCardVO>();
			String line = null;
			while ((line = br.readLine()) != null) {
				CreditCardVO cardVO = new CreditCardVO();
				String[] strArr = line.split("\\|");
				cardVO.setCustomerName(strArr[1]);
				cardVO.setBillAmount(Integer.parseInt(strArr[2]));
				String creditCardNo = strArr[0];
				SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
				sdf.setLenient(false);
```

```
            try {
                        Date dueDate = sdf.parse(strArr[3]);
                        cardVO.setDueDate(dueDate);
                        Date paymentDate = sdf.parse(strArr[4]);
                        cardVO.setPaymentDate(paymentDate);

            } catch (ParseException e) {
                        throw new CreditCardValidationException(
                                    "invalid date format");
            }
            if (creditCardNo.matches("[4]{1}[0-9]{15}")) {
                        cardVO.setCreditCardNumber(creditCardNo);
                        visaList.add(cardVO);
            } else if (creditCardNo.matches("(34|37)[0-9]{13}")) {
                        cardVO.setCreditCardNumber(creditCardNo);
                        amexList.add(cardVO);
            } else {
                        throw new CreditCardValidationException(
                                    "invalid credit card number");
            }
}

for (CreditCardVO cardVO : visaList) {
            if (cardVO.getPaymentDate().getTime() < cardVO.getDueDate()
                                    .getTime()) {
                        cardVO.setGrade("A");
            } else {
                        cardVO.setGrade("B");
                        long diff = (cardVO.getPaymentDate().getTime() - cardVO
```

```
                                .getDueDate().getTime());
                int days = (int) (diff / (1000 * 60 * 60 * 24));
                if (days <= 5) {
                        cardVO.setFine(cardVO.getBillAmount() * 0.1);
                } else {
                        cardVO.setFine(cardVO.getBillAmount() * 0.2);
                }

        }
        removeDuplicates(visaMap, cardVO);
}
for (CreditCardVO cardVO : amexList) {
        if (cardVO.getPaymentDate().getTime() < cardVO.getDueDate()
                        .getTime()) {
                cardVO.setGrade("A");
        } else {
                cardVO.setGrade("B");
                long diff = (cardVO.getPaymentDate().getTime() - cardVO
                                .getDueDate().getTime());
                int days = (int) (diff / (1000 * 60 * 60 * 24));
                if (days <= 5) {
                        cardVO.setFine(cardVO.getBillAmount() * 0.1);
                } else {
                        if (cardVO.getBillAmount() < 15000)
                                cardVO.setFine(cardVO.getBillAmount() * 0.2);
                        else
                                cardVO.setFine(cardVO.getBillAmount() * 0.3);
                }
```

```
                        }
                        removeDuplicates(amexMap, cardVO);
                    }
                    map.put(1, visaMap);
                    map.put(2, amexMap);

                } catch (Exception e) {
                    throw new CreditCardValidationException(e.getMessage());
                }
                return map;
            }

            private void removeDuplicates(Map<String, CreditCardVO> map,
                    CreditCardVO cardVO) {
                if (map.containsKey(cardVO.getCreditCardNumber())) {
                    if (map.get(cardVO.getCreditCardNumber()).getPaymentDate()
                            .getTime() < cardVO.getPaymentDate().getTime()) {
                        map.put(cardVO.getCreditCardNumber(), cardVO);
                    }
                } else {
                    map.put(cardVO.getCreditCardNumber(), cardVO);
                }
            }
        }

        class CreditCardVO {
            private String creditCardNumber;
            private String customerName;
            private int billAmount;
```

```
private Date dueDate;
private Date paymentDate;
private double fine;
private String grade;

public String getCreditCardNumber() {
	return creditCardNumber;
}

public void setCreditCardNumber(String creditCardNumber) {
	this.creditCardNumber = creditCardNumber;
}

public String getCustomerName() {
	return customerName;
}

public void setCustomerName(String customerName) {
	this.customerName = customerName;
}

public int getBillAmount() {
	return billAmount;
}

public void setBillAmount(int billAmount) {
	this.billAmount = billAmount;
}
```

```
public Date getDueDate() {
	return dueDate;
}

public void setDueDate(Date dueDate) {
	this.dueDate = dueDate;
}

public Date getPaymentDate() {
	return paymentDate;
}

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

public double getFine() {
	return fine;
}

public void setFine(double fine) {
	this.fine = fine;
}

public String getGrade() {
	return grade;
}

public void setGrade(String grade) {
```

```
		this.grade = grade;
	}

	@Override
	public String toString() {
		return "CreditCardVO [creditCardNumber=" + creditCardNumber
				+ ", customerName=" + customerName + ", billAmount="
				+ billAmount + ", dueDate=" + dueDate + ", paymentDate="
				+ paymentDate + ", fine=" + fine + ", grade=" + grade + "]";
	}
}

class CreditCardValidationException extends Exception {
	CreditCardValidationException(String str) {
		super(str);
	}
}
```

```
import java.util.Date;

public class Passenger {
String id,name;
int ticket;
Date date;

public Passenger(String id, String name, int ticket, Date date) {
	super();
	this.id = id;
	this.name = name;
	this.ticket = ticket;
```

```
        this.date = date;
}

public String getId() {
        return id;
}

public void setId(String id) {
        this.id = id;
}

public String getName() {
        return name;
}

public void setName(String name) {
        this.name = name;
}

public int getTicket() {
        return ticket;
}

public void setTicket(int ticket) {
        this.ticket = ticket;
}

public Date getDate() {
        return date;
```

```
}

public void setDate(Date date) {
	this.date = date;
}

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;
import java.util.Scanner;

public class Main {

public static void main(String args[]) throws ParseException {
	String id,name,dates;
	int ticket,i,n;
	Date date;
	@SuppressWarnings("resource")
	Scanner sc=new Scanner(System.in);
	System.out.println("Enter the number of passenger");
	n=sc.nextInt();
	sc.nextLine();
	List<Passenger> list=new ArrayList<>();
	for(i=1;i<=n;i++)
```

```
	{

		System.out.println("Enter the details of passenger "+i);
		id=sc.nextLine();
		name=sc.nextLine();
		ticket=sc.nextInt();
		sc.nextLine();
		dates=sc.nextLine();
		SimpleDateFormat sdf=new SimpleDateFormat("dd/mm/yyyy");
		date=sdf.parse(dates);
		Passenger e=new Passenger(id, name, ticket, date);
		list.add(e);
	}
	Map<Integer,Map<Integer,List<Passenger>>> res=new HashMap<>();
	res=createMap(list);
	for(Entry<Integer, Map<Integer, List<Passenger>>> pat: res.entrySet())
	{
		for(Entry<Integer,List<Passenger>> chi: pat.getValue().entrySet())
		{
			System.out.print(chi.getKey()+"	");
			for(Passenger a:chi.getValue())

	System.out.println(a.getId()+","+a.getName()+","+a.getTicket()+","+a.getDate());
		}
	}
	}

public static Map<Integer,Map<Integer,List<Passenger>>> createMap(List<Passenger> list) {
	Map<Integer,List<Passenger>> child=new HashMap<>();
```

```
		for(Passenger pass:list)
		{
			if(child.containsKey(pass.getTicket()))
				child.get(pass.ticket).add(pass);
			else
			{
				List<Passenger> newlist=new ArrayList<>();
				newlist.add(pass);
				child.put(pass.ticket,newlist);
			}
		}
		Map<Integer,Map<Integer,List<Passenger>>> parent=new HashMap<>();
		parent.put(1, child);
		return parent;
}

}

import java.util.Scanner;


public class Managerclass {

		/**
		 * @param args
		 * @throws AutoManagerException
		 */

		private  String getdetails(String id, String per, String acc) throws AutoManagerException {
```

```
        String ans="";
if(id.matches("[0-9]{6}"))
{
        if(per.equals("A") || per.equals("B"))
        {
                if(acc.equals("YES"))
                        {
                                ans="SUCCESS";
                        }
                else
                        {
                         throw new AutoManagerException("acc is imcorect");
                        }
        }
        else
        {
                throw new AutoManagerException("per is imcorect");
        }

}
else
{
        throw new AutoManagerException("id is imcorect");
}
        // TODO Auto-generated method stub
        return ans;
}
public static void main(String[] args) {
        // TODO Auto-generated method stub
```

```
    Scanner sc=new Scanner(System.in);
    String id=sc.nextLine();
    String per=sc.nextLine();
    String acc=sc.nextLine();
    String username=sc.nextLine();
    Managerclass manager=new Managerclass();
    try{
    String ans=manager.getdetails(id,per,acc);
    System.out.println(ans);
    }
    catch(AutoManagerException e)
    {
        //System.out.println("hi");
    }
        }

}
        class AutoManagerException extends Exception{

                public AutoManagerException(String s)
                {
                        System.out.println(s);
                }



        }
```

}