Université de Picardie Jules Verne
Faculté de Mathématiques et d'Informatique.

Licence informatique-Miage : Mathématiques financières

Examen 4 janvier 2022

Exercice 1

A. On effectue successivement 24 versements mensuels de 300 euros ayant lieu en fin de mois $1, 2, \cdots, 24$, le taux mensuel étant de 0.5 pourcents.

1. Calculer la valeur *acquise* par cette suite de versement immédiatement après le dernier versement.
2. Calculer la valeur acquise six mois et demi après le dernier versement.
3. Déterminer la valeur *actuelle* de cette suite de versements un mois avant le premier versement.
4. Déterminer la valeur actuelle de cette suite de versements trois mois et demi avant le premier versement.

B. On considère 3 versements mensuels ayant lieu en fin de mois (taux mensuel $t = 1$ pourcents), les versements suivant une progression géométrique de raison 1.2, le premier versement étant égal à 100 euros.

1. Déterminer la valeur acquise par cette suite de versements après le dernier versement.
2. Déterminer la valeur actuelle de cette suite de versements un mois avant le premier versement.

C. On fait un placement unique à intérêts composés au taux annuel de 5.25 pourcents. Quatre ans après ce placement, on retire chaque année 2500 euros. Au cinquième retrait, le compte est épuisé. Quel est le montant de ce placement unique ?

Exercice 2

Un particulier désire emprunter un capital C égal à 15000 euros à un établissement financier. Plusieurs formules de remboursement de cet emprunt sont proposées, toutes au taux annuel de 10 pourcents.

1. Formule 1. Rembourser capital et intérêts en une seule fois trois années après la remise des fonds. Quelle somme devra-t-il payer à la fin de la troisième année ?
2. Formule 2 : Remboursement en trois annuités constantes, première annuité un an après la remise des fonds. Dresser le tableau d'amortissement de cet emprunt indivis (système classique français).
3. Formule 3 : Remboursement en 24 mensualités constantes, première mensualité 13 mois après la remise des fonds, intérêts payés en fin de période. Établir que le montant d'une mensualité notée R satisfait l'équation (justifier soigneusement votre réponse)

$$C = R(1+t)^{-12}\frac{1-(1+t)^{-24}}{t},$$

puis en déduire R (on utilisera un taux mensuel *équivalent* au taux annuel).
4. Formule 4. Remboursement à amortissement constant en trois annuités. Dresser le tableau d'amortissement de cet emprunt.

Exercice 3
Dans cet exercice, on considère que l'emprunt est effectué à annuités constantes, la première annuité étant versé une période après la remise des fonds (système classique français).
A. Déterminer le montant du capital emprunté et le montant de l'annuité constante pour chacun des emprunts dont les éléments connus sont les suivants :
1. Remboursable en 5 ans, taux annuel 7.75 pourcents. Montant du premier amortissement 256.96.
2. Remboursable en 10 ans, taux annuel 10.25 pourcents, montant du dernier amortissement 969.83 euros.
3. Remboursable en 8 ans, montant du premier amortissement 490.86 euros, montant du dernier amortissement 777.96 euros.
B. Un emprunteur a contracté un prêt d'un montant $C = 100000$ euros remboursable en 96 mensualités. Le dernier amortissement est égal à 1237.68 euros
1. Donner une équation satisfaite par le taux mensuel t. Lequel des trois taux suivants est égal au taux mensuel t ? 0.002 ? 0.00375 ? 0.005 ?
2. Donner la valeur de la mensualité ainsi que le capital restant dû au début du 45 ième mois.

Exercice 4
Un particulier souhaite emprunter 30 000 euros au TEG fixe annuel de 8.091 pourcents (taux fixe nominal de 7.388 pourcents). Le remboursement s'effectuera en 96 mensualités de 414.83 euros.
Les frais de dossier s'élève à 450 euros (versé le jour de la remise des fonds).
Coût mensuel de l'assurance facultative de 28.5 euros en plus de l'échéance.
1. a. Calculer les taux mensuels proportionnels et équivalents du taux d'intérêt 7.388 pourcents.
b. Expliquer le calcul du montant des mensualités.
2. a. Déterminer une équation vérifiée par le TEG mensuel de ce prêt (sans prendre en compte l'assurance).
b. Ce TEG est-il égal au TEG annuel de 8.091 pourcents annoncé ?
3. En incluant l'assurance, lequel des trois taux suivants est le TEG de ce prêt ? 8.186 ? 9.675 ? 10.116 ? Justifier soigneusement votre réponse.

Université de Picardie Jules Vernes
Licence 3 Informatique 2021-2022
Sujet 1

Programmation Objet 2 – Examen écrit

Note: La consultation des documents de cours et TD en format papier est autorisée. Veuillez noter sur la copie la numéro de votre sujet.

I. Questions à Choix Multiples.

Entourez les réponses correctes.

1. L'utilisation de l'interface Serializable:

- (A) permet d'indiquer que la sérialisation en flux d'octets est autorisée, car Serializable est une simple interface de marquage,
- (B) permet d'indiquer que la sérialisation en flux d'octets et XML est autorisée, car Serializable est une simple interface de marquage,
- (C) est obligatoire pour pouvoir d'appeler les méthodes writeObject() et readObject() de la classe ObjectOutputStream lorsqu'on veut sérialiser un objet dans un flot.

2. En Java, la classe StringIndexOutOfBoundsException étend la classe IndexOutOfBoundsException. Pour traiter cette exception, comment corriger le code suivant:

```
public class Personne{

    String nom;
    String tel;

    Personne(String le_nom, String le_tel){
       nom=le_nom;
       tel=le_tel;
    }

    static boolean verification(String numero){

       for (int i=0; i<10;i++){
           if (!Character.isDigit(numero.charAt(i)))
             return false;
       }
       return true;

   }

    public static void main(String args[]) {
        if (verification(args[1])){
       Personne contact=new Personne(args[0],args[1]);
       System.out.println("Contact crée avec succes!");
    }
    }
```

}

- (A) Ce code compile correctement, mais on a erreur à l'exécution.
- (B) Ce code ne compile pas si on ne traite pas l'exception.
- (C) Pour pouvoir compiler ce code, on modifie comme suit:

```
public class Personne{

...

static boolean verification(String numero) throws StringIndexOutOfBoundsException {
  for (int i=0; i<10;i++){
      if (!Character.isDigit(numero.charAt(i)))
        return false;

   }
  return true;

}

public static void main(String args[]) {
    if (verification(args[1])){
  Personne contact=new Personne(args[0],args[1]);
  System.out.println("Contact crée avec succes!");
}
}
```

- (D) Pour traiter l'exception, on modifie comme suit:

```
public class Personne{

...

static boolean verification(String numero) throws StringIndexOutOfBoundsException  {

  for (int i=0; i<10;i++){
      if (!Character.isDigit(numero.charAt(i)))
        return false;
   }
  return true;

}

public static void main(String args[]) {

  try{
    if (verification(args[1])){
     Personne contact=new Personne(args[0],args[1]);
    System.out.println("Contact crée avec succes!");
  } catch (IndexOutOfBoundsException ex) {
      System.out.println( "numero invalid" );
    }
}
}
```

3. La classe suivante implemente une structure de données génériques. Corrigez-là si besoin.

```
public class ArbreGen<T extends Comparable<T>>{

    private T valeur;
```

```
    private ArbreGen filsD, filsG;

    public ArbreGen(T val, ArbreGen g, ArbreGen d){
      valeur=val;
      filsG=g;
      filsD=d;
    }

    public T getValeur(){
      return valeur;
    }
  }
```

- (A)

```
public class ArbreGen<T extends Comparable<T>>{

    private T valeur;
    private ArbreGen<T> filsD, filsG;

    public ArbreGen(T val, ArbreGen<T> g, ArbreGen<T> d){
            valeur=val;
            filsG=g;
            filsD=d;
      }

    public T getValeur(){
      return valeur;
    }

}
```

- (B) Le code compile correctement sans warning, il n'y a pas de modification à faire.

- (C)

```
public class ArbreGen<T extends Comparable<T>>{

      private T valeur;
      private ArbreGen<T> filsD, filsG;

     public ArbreGen(T val, ArbreGen g, ArbreGen d){
            valeur=val;
            filsG=g;
            filsD=d;
       }

public T getValeur(){
            return valeur;
      }

}
```

4. On reprend le code de la question précedente et on rajoute une méthode pour l'insértion dans l'arbre. Parmi les réponses suivantes, choissisez les versions correctes.

- (A)

```
public void insertion(T val) {
      if (val == getValeur())
          return;  // la valeur est deja dans l'arbre
      if (val < getValeur()) {
```

```
            if (filsG!= null)
              filsG.insertion(val);
            else
              filsG = new ArbreGen(val,null,null);
        }
        if (val> getValeur()) {
            if (filsD!= null)
              filsD.insertion(val);
            else
              filsD = new ArbreGen(val,null,null);
        }
    }
```

- (B)

```
    public void insertion(T val) {
        if (val.compareTo(getValeur())==0)
            return;  // la valeur est deja dans l'arbre
        if (val.compareTo(getValeur())>0) {
            if (filsG!= null)
              filsG.insertion(val);
            else
              filsG = new ArbreGen<>(val,null,null);
        }
        if (val.compareTo(getValeur())<0) {
            if (filsD!= null)
              filsD.insertion(val);
            else
              filsD = new ArbreGen<>(val,null,null);
    }
}
```

- (C)

```
    public ArbreGen<T> insertion(T val) {

        if (val< getValeur()) {
            if (filsG!= null)
              filsG.insertion(val);
            else {
              filsG = new ArbreGen<T>(val,null,null);
              return filsG;
            }
        }
        if (val > getValeur()) {
            if (filsD!= null)
              filsD.insertion(val);
            else {
              filsD = new ArbreGen<T>(val,null,null);
              return filsD;
          }
        }
        return this;
}
```

II. 1. Donner le code obtenu à la compilation après effacement:

```
public class TestListe {

    public static void main(String[] args) {
      Double sum=0.0;
      ArrayList<Integer>  liste=new ArrayList<>();

       Iterator<Integer> it=liste.iterator();
```

```
        while (it.hasNext()) {

            sum=sum+it.next();
        }
        System.out.println("la somme est"+sum);

    }
}
```

2. On modifie le code de l'exercice 1. comme suit:

```
public class TestListe {

    public static void main(String[] args) throws InterruptedException{
        Double sum=0.0;
        ArrayList<Integer>  liste=new ArrayList<>();
        Thread tache = new Thread(new ThreadRemplissage(liste));
        tache.start();

          Iterator<Integer> it=liste.iterator();
          while (it.hasNext()) {
              Integer l=it.next();
            sum=sum+l;

        }
        System.out.println("la somme est"+sum);

    }

    static class ThreadRemplissage implements Runnable {
      private ArrayList<Integer> maListe;
      public ThreadRemplissage(ArrayList<Integer> l) {
          maListe = l;
      }

      public void run() {
          for (int i=0; i<200; ++i)
            maListe.add(i);
      }
    }

}
```

Est-ce que ce code compile et s'exécute correctement? Justifiez votre réponse. En cas de besoin, corrigez-le.

III. 1. Dans le programme suivant, on effectue en parallèle plusieurs calculs et on souhaite écrire les résultats dans un fichier commun. Corrigez le programme, afin de s'assurer qu'un seul thread écrira dans le fichier à la fois.

```
public class EcritureParallele {

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

       FileWriter flot = new FileWriter("resultats.txt");


      Thread [] taches=new Thread[1000];

      for (int i=0; i<1000; i++) {
        taches[i] = new Thread(new ThreadEcriture(flot, i));
        taches[i].start();
```

```
      }

      for (int i=0; i<1000; i++)
        taches[i].join();
      flot.close();

  }
}

public class ThreadEcriture implements Runnable {

    int n;
    FileWriter flot;

    public ThreadEcriture(FileWriter f, int m){
      flot=f;
      n=m;
    }

    public void run() {
      int sum=0;
      for (int i=0; i <n;i++)
          sum=sum+i*i;

      try{
          flot.write(sum+"\n");
         } catch (IOException ex){}

    }

}
```

2. Les opérations IO sont coûteuses et l'approche prise ci-dessous risque de ralentir la performance du programme. Une meilleure approche serait de suivre le pattern Producer-Consumer, avec un seul thread dédié à l'écriture. On modifie la classe du thread qui effectue le calcul comme suit. Modifiez le code de la classe EcritureParallele en conséquence.

```
public class ThreadEcriture implements Runnable {

    int n;
    ArrayBlockingQueue<Integer> fileDeJob;

    public ThreadEcriture(ArrayBlockingQueue<Integer> f, int m){
      fileDeJob=f;
      n=m;

    }

    public void run() {
        try {
        int sum=0;
      for (int i=0; i <n;i++)
          sum=sum+i*i;

        fileDeJob.put(sum);
      } catch (InterruptedException ie) {}


    }

}
```