CORRIGÉ (LES RÉPONSES APPARAISSENT AVEC CE FOND DE CARACTÈRE)

Contrôle terminal info2 / Semestre 3

M3101 : Principes des systèmes d'exploitation

Nom du responsable :	**François MERCIOL**
Date du contrôle :	**22 octobre 2018**
Durée du contrôle :	**1 heure 30 minutes**
Nombre total de pages :	**10 pages**
Impression :	**Recto – Verso**
Documents autorisés :	**Tous**
Calculatrice autorisée :	**non**
Réponses :	**Sur le sujet : 3 de parties séparées**

Conseils :

- Indiquez **votre nom** sur chaque feuille à rendre **dès qu'elles vous sont données**.
- Afin d'éviter la copie, toute réponse non justifiée sera considérée comme nulle.
- **Les parties sont indépendantes** et corrigées par des enseignants différents (commencez par la plus simple pour vous).
- **Rendez les 3 parties séparément** (dans les 3 réceptacles à la sortie).
- Il est demandé des réponses à la fois claires et concises.
- **Lisez en entier** le contrôle avant de commencer à répondre.
- **Ne restez pas bloqué**, vous pourrez revenir sur une question difficile par la suite.
- **Ne brûlez pas toute votre énergie**, des calculs longs peuvent vous rapporter moins de points que la réponse à des questions de réflexion.
- **Conservez du temps pour chaque partie**
  (90 min / 20 pts => pas plus de 4 minutes 30 par point)

Le devoir se compose de 3 parties :

- [4 pts] questions de cours (QCM : Questions à Choix Multiple / 10 min)
- [8 pts] « Allons au Mont » (problème / 40 min)
- [8 pts] analyse de textes (exercices / 40 min)

Ce corrigé est indicatif. D'autres réponses peuvent être considérées comme justes.

CORRIGÉ (LES RÉPONSES APPARAISSENT AVEC CE FOND DE CARACTÈRE)

NOM : _____ PRENOM : _____ GROUPE TD : ____ Note :

1.) [4 pts] question de cours (QCM : Questions à Choix Multiple / 10 min)

✗ Cochez la case réponse

Question	✾	✱	❀	✲
Pour Unix, un numéro majeur désigne ✾) une version stable ✱) un dispositif matériel ❀) une expression régulière ✲) un développement important		○		
"StringTokenizer" découpe en ✾) lignes ✱) mots ❀) anneaux magiques ✲) paragraphes		○		
En Java, un moniteur ✾) résout une section critique ✱) s'oppose au Mutex ❀) neutralise un sémaphore ✲) prévient de la noyade	○			
L'électricité est un vecteur de transport d'énergie, mais dans le monde, quelle source d'énergie est la moins utilisée par l'Internet ? ✾) Le nucléaire ✱) Les gaz de schiste ❀) Le charbon ✲) Le renouvelable				○
La mémoire flash ✾) sert à étaler sa vie privée ✱) est une forme d'EEPROM ❀) ne dure pas longtemps ✲) permet de modifier un octet		○		
Concernant l'emploi hors État-Unis d'Amérique, l'Internet ✾) détruit des emplois ✱) créer des libraires ❀) repeuple les campagnes ✲) développe le commerce local	○			
Sous Unix, un processus démon ne peut jamais être : ✾) défunt ✱) suspendu ❀) créé par un moldu ✲) tué	○			
Quel est la marque de fin de ligne sous Unix ? ✾) \r\n ✱) \n ❀) \r ✲) \n\r		○		
Le SFINX est ✾) un routeur aléatoire ✱) l'interconnexion des réseaux en France ❀) un animal mythique ✲) le protocole d'échange égyptien		○		
Les premiers systèmes de gestion de fichiers sont apparus dans les années ✾) 60 ✱) 70 ❀) 90 ✲) 80	○			
Pour ne pas céder d'informations aux réseaux commerciaux de collecte de données personnelles (comme FaceBook), il faut ✾) interdire les cookies ✱) éviter les pages avec un "f" ❀) créer un profil restreint ✲) faire un vœu		○		
En informatique, l'acronyme RAID désigne ✾) un regroupement de disques ✱) une unité de police ❀) un algorithme de recherche ✲) un PC en acier renforcé	○			
Les premières mémoires électroniques (sans consommation d'énergie une fois programmées), utilisaient des ✾) petites réserves d'eau ✱) dauphins ❀) rayons de lumière ✲) anneaux magnétiques				○
Sous Unix, un processus terminé dont le père n'a pas encore lu le testament se nomme : ✾) orphelin ✱) accidenté ❀) en attente ✲) zombie				○
Quel est le cycle énergétique le plus court ? ✾) le papier ✱) le vélo de ma grand-mère ❀) le pétrole ✲) le nucléaire	○			
On qualifiera une application de "temps réel" si des actions peuvent y être ✾) abandonnées ✱) réalisé en moins d'1 s ❀) réalisé en moins d'1 ns ✲) réalisé en moins d'1 ms	○			
En informatique, MBR veux dire ✾) Mon Bien Réutilisable ✱) Master Boot Record ❀) Manual Before Repair ✲) Memory Boot Recover		○		
Un programme Java qui ouvre une fenêtre se termine ✾) à la fin du main ✱) par un banquet ❀) dans un temps déterminé ✲) par appel à System.exit				○
L'algorithme de suppression de page LRU : ✾) vide le cache ✱) prive de ressources ❀) prend en file ✲) tient compte du dernier utilisé				○
Sous Unix, le PID désigne : ✾) un processeur ✱) un processus ❀) la Pile Interne des Données ✲) un programme	○			

+	-	.	/20

Corrigé (Les réponses apparaissent avec ce fond de caractère)

NOM : _____ PRENOM : _____ GROUPE TD : ____ Note :

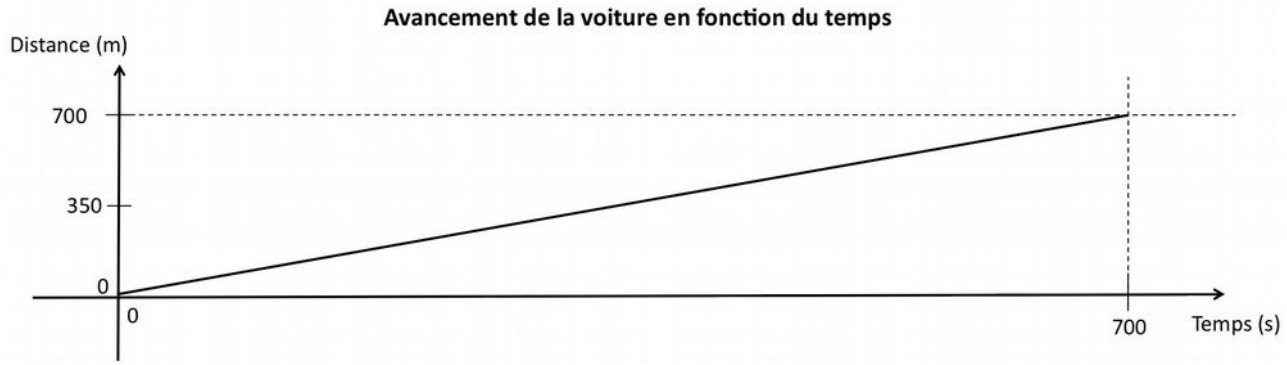
2.) [8 pts] « Allons au Mont » (problème / 40 min)

Pour se rendre au Mont-Saint-Michel, avant l'ajout du pont-passerelle qui est insubmersible, il était nécessaire d'emprunter un chemin sableux, qui se dévoilait au gré des marées. Chaque jour, la mer ouvrait donc ce passage une heure avant la basse mer, et une heure après. Le reste du temps, il était nécessaire de passer par le pont-passerelle qui est insubmersible.

Les questions se trouvent sur les pages suivantes. Vous devrez répondre sous chaque question. En cas de manque de place vous pourrez utiliser la zone ci-dessous en extension.

2.A) [2 pt] Passer

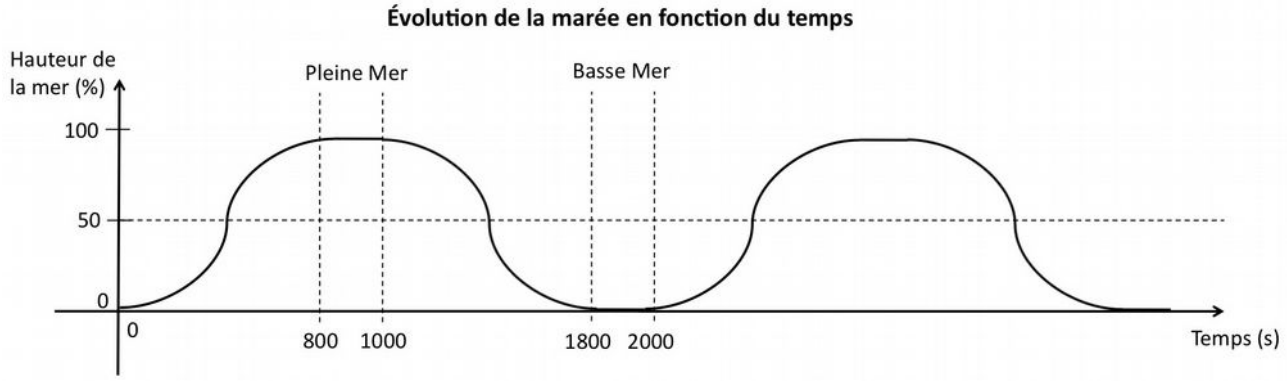
On souhaite écrire un algorithme comportant un Thread qui permet de modéliser un visiteur en voiture qui parcoure ce passage d'autrefois. On considère que le passage fait 700 mètres. L'avancement de la voiture, mètre par mètre, peut ainsi être simulé par une boucle comportant une pause de 1s à chaque itération. Une fois le passage parcouru par la voiture, le Thread est terminé. Écrivez cet algorithme, ainsi que le « Main » pour le tester.




```
class Voiture extends Thread {
   private String nomVoiture = "";
   private int distanceParcourue;

   public Voiture (String nomVoiture) {
      this.nomVoiture = nomVoiture;
      distanceParcourue = 0;
   }

   public void run () {
      try {
         for (int i = 0; i < 700; i++) {
            distanceParcourue++;
            System.out.println ("La voiture ("+ nomVoiture +
                              ") a parcourue "+
                              distanceParcourue + " mètres");
            sleep (1000); //(En millisecondes !)
         }
      } catch (InterruptedException e) {
        //imposé par le sleep !
      }
   }

   public static void main (String[] args) {
  // On initialise les objes voitures
      Voiture v1 = new Voiture ("Kangoo");
      Voiture v2 = new Voiture ("Picasso");
      Voiture v3 = new Voiture ("Logan");

   //On lance les threads
      v1.start();
      v2.start();
      v3.start();
   }
}
```

2.B) [2 pt] Simuler la marée

On souhaite également simuler le mouvement des marées dans un algorithme comportant un Thread. Comme le montre le schéma ci-dessous, les marées montent, et descendent indéfiniment, du moins tant que notre Lune sera présente. Ce mouvement pourra donc être modélisé par l'itération d'un compteur variant de 0 à 100, puis inversement de 100 à 0, par incrément de 1, où la valeur 0 indiquera la basse mer, et la valeur 100 indiquant la pleine mer. Chaque itération comportera une attente de 8s et les phases de pleine mer et basse mer comporteront elles une attente de 200s. Écrivez cet algorithme, ainsi que le « Main » pour le tester.



Corrigé (Les réponses apparaissent avec ce fond de caractère)

```
public class Mer extends Thread {
   private int niveauMer;
   private Boolean merMontante;
   private Boolean merDescendante;

   public Mer () {
      //On début en basse mer montante.
      niveauMer = 0;
      merMontante = true;
      merDescendante = false;
   }

   public void run () {
      try {
         while (true) {
          // boucle à l'infini (à remplacer par
          // un incrément fini pour vos tests...)
            if (merMontante) {
             // S'en suit une incrémentation pour
             // la mer montante
               niveauMer++;
               System.out.println ("Niveau de la mer : "+
                                   niveauMer);
               if (niveauMer == 100) {
                  System.out.println ("La mer est Haute / niveau -> "+
                                      niveauMer);
                  sleep (200000);
                  merMontante = false;
                  merDescendante = true;
                  continue;
               }
            } else if (merDescendante) {
             // S'en suit une décrémentation pour
             // la mer descendante
               niveauMer--;
               System.out.println ("Niveau mer : "+ niveauMer);
               if (niveauMer == 0) {
                  System.out.println ("La mer est basse / niveau -> "+
                                      niveauMer);
                  sleep (200000);
                  merMontante = true;
                  merDescendante = false;
                  continue;
               }
            }
            sleep (8000);
          //Attente de 8 secondes à chaque itération
          }
      } catch (InterruptedException e) {
      //imposé par le sleep !
   }
   }

   public static void main (String[] args) {
      Mer m = new Mer(); //On créer une mer..
      m.start (); // On lance le thread
   }
}
```

2.C) [4 pt] Passer tant qu'il est encore temps

Avec les algorithmes ci-dessus, de nombreux visiteurs verraient leurs voitures sombrer sous l'eau, ce qui est peu souhaitable. Il faut donc modifier vos algorithmes pour que les voitures ne puissent passer sur le passage que lorsque la mer est en phase descendante à partir du niveau de moitié (indice du niveau de la mer = 50) et ce jusqu'au quart de la phase montante (soit l'indice du niveau de la mer = 25) étant donné qu'il faut compter le temps de chemin. Écrivez la modification des algorithmes permettant de modéliser cela ainsi que le « Main » pour le tester.

Corrigé (Les réponses apparaissent avec ce fond de caractère)

```
public class MerPassage extends Thread {
   public int niveauMer;
   private Boolean merMontante;
   private Boolean merDescendante;
   private Boolean etatPassage;

   public MerPassage () {
      //On débute en basse mer, montante.
      niveauMer = 0;
      merMontante = true;
      merDescendante = false;
   }

   /** Bloque le passage si la mer est trop haute Libère le passage et
notify les voitures qui attendent quand le niveau de la mer le permet. */
   public synchronized void gerePassage ()
  throws InterruptedException {
      // Si la mer monte ou qu'elle n'est pas encore assez descendu,
  // on bloque le passage
      if ((merDescendante && niveauMer > 50) ||
      (merMontante && niveauMer>25)){
         System.out.println ("Le passage est fermé,"+
                             " attendez la marée basse -"+
                             " Niveau de la mer : " + niveauMer);
      } else if ((merDescendante && niveauMer < 50) ||
            ( merMontante && niveauMer<25)){
      // On ouvre le passage qui est maintenant accessible
      // on prévient les voitures qui attendent
         System.out.println ("Ouverture du passage -"+
                             " Bienvenue au Gois -"+
                             " Niveau de la mer : "+ niveauMer);
         notifyAll ();
      }
   }

   /** Permet à une voiture de passer lorsque la marée le permet  Met
la voiture en attente si la marée ne le permet pas. Dans notre cas la ges-
tion du passage (l'autre méthode) est très rapide, mais si elle prenait plus
de temps l'emploi des mots-clé synchronized sur ces deux méthodes ne
permettrait pas à une voiture d'utiliser le passage tant qu'il y aurait une
gestion du passage. */
   public synchronized void utiliserPassage ()
  throws InterruptedException {
      if ((merMontante && niveauMer > 25) ||
      (merDescendante && niveauMer > 50)) {
      // Si la mer ne permet pas de passer sur le passage,
      // on précise au voiture d'attendre et de se détendre
         // Les threads seront en attente passive !
         wait();
      }
      //Une fois débloqué par le notify -> On passe !
      System.out.println ("On passe sur le passage ! -"+
                          " Niveau mer = "+ niveauMer +
                          " Mer montante ? = "+ merMontante +
                          " Mer descendante ?"+ merDescendante);
   }

   public void run () {
      try {
         while (true) {
          // boucle à l'infini (remplacer par
          // une itération fini lors de vos essais)
            if (merMontante) {
             //Si la mer monte
               niveauMer++;
               gerePassage ();
             // Dès que la mer bouge,
             // on gère le passage (Méthode synchronisée !)
               System.out.println ("Mer montante -"+
                                   " Niveau de la mer : "+
                                   niveauMer);
             if (niveauMer == 100) {
                System.out.println ("La mer est haute -"+
                                    " Niveau de la mer -> "+
                                    niveauMer);

                sleep (200000); //En millisecondes !
                // On inverse la phase de la marée,
               // on passe en mer descendante
                merMontante = false;
                merDescendante = true;
                continue;
             }
            } else if (merDescendante) {
             //Si la mer descend
               niveauMer--;
               gerePassage ();
            // Dès que la mer bouge, on gère le passage
            // (Méthode synchronisée !)
              System.out.println ("Mer descendante -"+
                                  " Niveau mer : "+niveauMer);

              if (niveauMer == 0) {
                 System.out.println ("La mer est basse -"+
                                     " Niveau de la mer -> "+
                                     niveauMer);
                 sleep (200000); // En millisecondes !
                 // On inverse la phase de la marée,
                // on passe en mer montante
                 merMontante = true;
                 merDescendante = false;
                 continue;
              }
            }
            sleep (8000);
          // On dors un peu, la mer ça fatigue !
          // Mais surtout parce que c'est demandé
          // et pour afficher les traces correctement
          }
       } catch (InterruptedException e) {
      //imposé par le sleep !
   }
   }
}
public class VoiturePassage extends Thread {
   private String nomVoiture = "";
   private int distanceParcourue;

   MerPassage mer;

   public VoiturePassage (String nomVoiture, MerPassage m) {
      this.nomVoiture = nomVoiture;
      distanceParcourue = 0;
      mer = m;
   }

   public void run () {
      try {
         for (int i = 0; i < 700; i++) {
          // On dors un peu avant de prendre la route pour
          // que les traces ne défilent pas trop vite
            sleep (1000);
          // On utilise le passage (Méthode synchronisée !)
            mer.utiliserPassage ();
            distanceParcourue++;
            System.out.println ("La voiture ("+ nomVoiture +
                                ") a parcourue "+
                                distanceParcourue + " mètres.");
          // Pour simuler l'avancement de la voiture comme
          // demandé dans l'énoncé
            sleep (1000);

         }
       } catch (InterruptedException e) {
      //imposé par le sleep !
   }
   }
}
public class Passage {

   public static void main (String args[]) {
      //On initialise une mer et des voitures
      MerPassage mer = new MerPassage ();
      VoiturePassage voit = new VoiturePassage  ("Kangoo", mer);
      VoiturePassage voit2 = new VoiturePassage ("Mercedes", mer);
      VoiturePassage voit3 = new VoiturePassage ("Ford", mer);
      VoiturePassage voit4 = new VoiturePassage ("Citroen", mer);
      VoiturePassage voit5 = new VoiturePassage ("Opel", mer);

      //On lance les threads -> Autrement dit on précise au système
      //qu'il peut appeler la méthode run quand il le souhaite !
      mer.start ();
      voit.start ();
      voit2.start ();
      voit3.start ();
      voit4.start ();
      voit5.start ();
   }
}
```

NOM : _____ PRENOM : _____ GROUPE TD : ____ Note :

3.) [8 pts] Analyse de texte (exercices / 40 min)

3.A) [2 pt] Ensemble de techniques

Le texte suivant est a analyser en Java.

Cette phrase est composée de 7 mots.

Indiquez 3 manières (avec le code Java correspondant) de récupérer les chaînes de caractères qui composent la phrase précédente.

On suppose la déclaration :

```
String words = "Cette phrase est composée de 7 mots.";
```

On peut retrouver les mots :

*** soit avec la classe String**

```
String tab [] = words.split ("\\s");
```

***soit avec la classe StringTokenizer**

```
StringTokenizer st = new StringTokenizer (words, " ");
while (st.hasMoreTokens ()) {
   String word = st.nextToken ();
   System.out.print (" - "+word);
}
System.out.println ();
```

*** soit avec les classes Pattern et Matcher**

```
Pattern p =
   Pattern.compile ("((\\p{IsLatin}|\\p{Digit})+)\\W*");
Matcher m = p.matcher (words);
for (int pos = 0; m.find (pos); pos += m.group (0).length ())
    System.err.println ("find group: "+m.group (1));
```

3.B) [3 pt] Reconnaissance de modèles

On veut analyser les lignes suivantes

Corrigé (Les réponses apparaissent avec ce fond de caractère)

2018-10-22 10:30 Lucien progression 3

2018-10-22 Jacques progression 18

Bob progression 42

2018-10-22 10:30 Fabienne réussi

Elles sont composées d'une suite d'éléments séparée par un ou des espaces quelconques :

- date (optionnelle)
- heure (optionnelle)
- prénom commençant par une majuscule
- évènement en minuscule
- argument numérique (optionnel)

Cherchez un modèle générique pour ces différentes lignes

```
public String format =
   "((\\d{4}-\\d{2}-\\d{2})\\s+)?"+
   "((\\d{2}:\\d{2})\\s+)?"+
   "(\\p{Upper}\\w*\\s+)"+
   "(\\p{Lower}+)"+
   "(\\s+(\\d+))?";
```

Corrigé (Les réponses apparaissent avec ce fond de caractère)

Ecrivez une méthode d'analyse qui affiche les différents groupes d'éléments si une ligne correspond au modèle que vous avez écrit.

```
public void parse (String line) {
   System.err.println ("expr: "+format);
   System.err.println ("line:"+line);
   Pattern p = Pattern.compile (format);
   Matcher m = p.matcher (line);
   boolean b = m.matches ();
   System.err.println ("matches: "+b);
   if (b) {
       System.err.println ("date:"+m.group (2));
       System.err.println ("heure:"+m.group (4));
       System.err.println ("firstname:"+m.group (5));
       System.err.println ("event:"+m.group (6));
       System.err.println ("arg:"+m.group (8));
   }
}
```

3.C) [3 pt] Reconnaissance de date

Le texte suivant est à analyser en Java.

Il est 8h 42 minutes et 30 secondes le 22-10-2018

Quelle classe est la plus adaptée pour formater une date ?

La classe la plus appropriée est java.text.SimpleDateFormat

Ecrivez la chaîne de caractères de format (comme elle doit se trouver dans un programme Java (il y a des « \ »)).

```
String format = "'Il est 'H'h 'm' minutes et 's' secondes le 'dd-MM-yyyy";
```

Ecrivez une méthode qui prend une date et retourne une chaîne de caractères au format précédant :

```
public String date2string (Date date) {
   SimpleDateFormat sdf = new SimpleDateFormat (dateFormat);
   return sdf.format (date);
}
```

Ecrivez me méthode analysant une chaîne suivant le format précédent et retournant une date :

```
public Date string2date (String string) {
   SimpleDateFormat sdf = new SimpleDateFormat (dateFormat);
   return sdf.parse (string, new ParsePosition (0));
}
```