Université Paris Diderot – Paris 7
Introduction à l'informatique et à la programmation (IF1)
Corrigé de l'examen du 7 janvier 2008

Exercice 1. *Conversion Farenheit/Celsius*

```
import fr.jussieu.script.*;
class Conversion {
    public static void main(String[] args){
       double celsius, farenheit; // float est egalement possible
       Deug.print("Temperature en degres Farenheit ? ");
       farenheit = Deug.readDouble();
       // pour obtenir la temperature en degres Celsius, on retranche 32
       // puis on multiplie par 5/9
       // le parenthsage et le type double de la variable farenheit
       // assure que les oprations sont faites sur des double
       celsius = ((farenheit - 32) * 5) / 9;
       Deug.println("La temperature est " + celsius + " degres Celsius");
    }
}
```

Exercice 2. *Calcul (simplifié) de l'impôt sur le revenu en 2008*

```
import fr.jussieu.script.*;
class Impots2008 {
   public static void main(String[] args){
       double revenu, parts, impots, reste;
       int abattement, imposable;
       int qf, resultat;
       Deug.print("Revenu declare ? ");
       revenu = Deug.readFloat();
       Deug.print("Nombre de parts ? ");
       parts = Deug.readFloat();
       abattement =(int) (revenu/10);
       imposable = (int)(revenu - abattement);
       qf = (int)(imposable / parts);
       impots = 0;
       if (qf > 5687)
          if (qf > 11344){
              impots += 5656 * 0.055;
              if (qf > 25195){
                  impots += 13851 * 0.14;
                  if (qf > 67546){
                       impots += 42351 * 0.3;
                       reste = qf - 67646 + 1;
                       impots += reste * 0.4;
                       }
                  else {
                       reste = qf - 25195 +1;
                       impots += reste * 0.3;
                       }
                   }
              else {
                  reste = qf - 11344 + 1;
                  impots += reste * 0.14;
                   }
              }
          else {
              reste = qf - 5688 + 1;
              impots += reste * 0.055;
               }
       resultat = (int) (impots * parts);
       Deug.println("\nMontant de l'impot : " + resultat);
    }
}
```

Exercice 3. *Autour de l'indicateur de chemins de fer*

3.1. *Affichage en clair d'une heure exprimée en secondes écoulées depuis 0h*

```
public static void heure(int date){
   if(date < 0 || date >= 1440) Deug.println("heure incorrecte");
   else {
      int minutes = date%60;
      Deug.print(date/60 + "h");
      if(minutes < 10) Deug.print("0" + minutes);
      else Deug.print(minutes);
   }
}
```

3.2. *Recherche du premier train partant après une heure donnée*

```
public static int prochainTrain(int[] depart, int[] arrivee, int date){
// le parametre arrivee est inutile
   if(date  < 0 || date >= 1440) return -1;
   for(int train  = 0; train < depart.length; train++)
       if (depart[train] >= date) return train;
   return -1;

}
```

3.3. *Recherche de tous les trajets possibles avec une correspondance*

```
public static void tousTrajets(int[] depart1, int[] arrivee1, int[] depart2, int[] arrivee2){
      int train1, train2, correspondance;
      int plusCourt1 = -1;
      int plusCourt2 = -1;
      for(train1 = 0; train1 < depart1.length; train1 ++){
          correspondance = arrivee1[train1] + 15;
          train2 = prochainTrain(depart2, arrivee2, correspondance);
          if(train2 == -1) return;
          heure(depart1[train1]);
          Deug.print(" --> ");
          heure(arrivee1[train1]);
          Deug.print("/ ");
          heure(depart2[train2]);
          Deug.print(" --> ");
          heure(arrivee2[train2]);
          Deug.print(" **** Temps total ");
          heure(arrivee2[train2] - depart1[train1]);
          Deug.print(" **** Temps d'attente ");
          heure(depart2[train2] - arrivee1[train1]);
          Deug.println();
      }
}
```

3.4. *Recherche de trains permettant d'arriver le plus tard possible avant une heure donnée*

```
public static void trajetAdapte(int[] depart1, int[] arrivee1, int[] depart2, int[] arrivee2, int h){
      int train1, train2, correspondance;
      // recherche du train sur la seconde partie du trajet
      for(train2 = arrivee2.length - 1; train2 >= 0; train2 --)
          if(arrivee2[train2] <= h) break;
      if(train2 == -1) {
              Deug.println("Voyage impossible (pas de train seconde partie du trajet)");
              return;}
      // recherche du train sur la premiere partie du trajet
      // heure d'arrivee du premier train : depart du second - 15
      int h2 = depart2[train2]-15;
      for(train1 = arrivee1.length - 1; train1 >= 0; train1 --)
          if(arrivee1[train1] <= h2) break;
      if(train1 == -1) {
              Deug.println("Voyage impossible (pas de train premiere partie du trajet)");
              return;}
      Deug.println("Premier train : " + train1);
      Deug.println("Second train : " + train2);
}
```

Exercice 4. *Création d'une pyramide*

```
import fr.jussieu.script.*;
class Pyramide {
    public static void espace(){ Deug.print(" ");}
    public static void entier(int n){ Deug.print(n);}
    public static void ligne(){ Deug.println();}
    public static void main(String[] args){
       int lig = 1;
       int nbEspaces = 10;
       while(lig < 11) {
               for(int i=0; i <nbEspaces; i++) espace();
               for(int i=0; i<lig; i++) entier((lig+i)%10);
               for(int i=lig-1; i>0; i--) entier((lig+i-1)%10);
               ligne();
               lig++; nbEspaces--;
       }
    }
}
```

Exercice 5. *Autour du jeu de la vie*

5.1. *Construction du tableau des voisins d'un point donné*

On travaille modulo la dimension concernée et pour éviter les valeurs négatives on ajoute $n-1$ plutôt que -1.

```
    public static int[][] voisins(boolean[][] t, int i, int j){
        int[][] v = new int[8][2];
        v[0][0] = (t.length + i - 1) % t.length; v[0][1] = (t[0].length + j - 1) % t[0].length;
        v[1][0] = (t.length + i - 1) % t.length; v[1][1] = j;
        v[2][0] = (t.length + i - 1) % t.length; v[2][1] = (j + 1) % t[0].length;
        v[3][0] = i; v[3][1] = (t[0].length + j - 1) % t[0].length;
        v[4][0] = i; v[4][1] = (t[0].length + j + 1) % t[0].length;
        v[5][0] = (i + 1) % t.length; v[5][1] = (t[0].length + j - 1) % t[0].length;
        v[6][0] = (i + 1) % t.length; v[6][1] = j;
        v[7][0] = (i + 1) % t.length; v[7][1] = (j + 1) % t[0].length;
        return v;
    }
```

5.2. *Calcul de l'état suivant d'une case donnée*

```
    public static boolean etatSuivant(boolean[][] t, int i, int j){
        int[][] v = voisins(t, i, j);
        int n = 0; // compteur du nombre de voisins contenant une cellule
        for(int k = 0; k < 8; k++)
           if(t[v[k][0]][v[k][1]]) n++;
        if(n == 3) return true;
        if(n == 2 && t[i][j]) return true;
        return false;
    }
```

5.3. *Calcul de la génération suivante*

```
    public static void generationSuivante(boolean[][] t){
        boolean[][] t2 = new boolean[t.length][t[0].length];
        for(int i = 0; i < t.length; i++)
           for(int j = 0; j < t[0].length; j++)
               t2[i][j] = etatSuivant(t, i, j);
        for(int i = 0; i < t.length; i++)
           for(int j = 0; j < t[0].length; j++)
               t[i][j] = t2[i][j];
    }
```