11 avril 2011, 9:30-12:30, salle 1355

Directives:

1. Toute documentation est **permise**.
2. Les calculatrices électroniques, ordinateurs, ipad, ipod, téléphones cellulaires, sont **interdits**.
3. Inscrivez tout de suite votre **nom**, votre **code permanent** et le **numéro de votre place**.
4. Répondez **sur le questionnaire**, dans l'espace libre qui suit chaque question. Si vous manquez de place, écrivez au verso en l'indiquant très clairement.
5. Dans les questions où ont vous demande de produire des sorties de programme, vous devez tout indiquer, incluant les espace que vous indiquerez par un caractère souligné (_).
6. L'examen est noté sur 80. Une règle de trois vous donnera votre note à cet examen (afin de la ramener à 35). Le barème est donné à titre indicatif seulement.

BONNE CHANCE!!

1.	_____	/20
2.	_____	/16
3.	_____	/20
4.	_____	/24
Total	_____	/ 80

Nom: _____ **Code permanent:** _____

Numéro de votre place: _____

1. (20 points) — Questions de cours

   Considérez la méthode suivante où les numéros de ligne ne font pas partie du code.

```
 1 boolean fonc() {
 2    float d = 0.0;
 3    short j = 2;
 4    final int v;
 5    int i = (int) Math.random() * 100 + 2;
 6    System.out.println(v);
 7    v = 2;
 8
 9    { int i = 3;
10      System.out.println(i);
11    }
12    v /= 2;
13        k = Math.random() * 100 / ((int) Math.random() * 5);
14    if (k) return true;
15    System.err.println((int) (Math.random()*5) / 5);
16  }
```

   (a) (8 points) Entourez le numéro de chaque ligne pour laquelle il y a une erreur de compilation ou d'exécution en indiquant devant s'il s'agit d'une erreur de compilation ou d'exécution. Indiquez ensuite ci-après la raison de chaque erreur ainsi qu'une correction (vous devez préserver autant que faire se peut la sémantique du programme: simplement retirer les lignes fautives n'est — sauf exception — pas la bonne correction).

(b) (2 points) Qu'affiche l'exécution de ce programme:

```
float x = 3 / 2;
int y = 3 / 2;
System.out.println("x=" + x + " y=" + y);
System.out.println("3/2 + 3/2 = " + x + y);
```

(c) (1 point) Qu'affiche l'exécution de ce programme:

```
int i = 2, j = 3;
System.out.println(i+j+" "+i+j*2);
```

(d) (3 points) Réécrivez cette méthode — sans en modifier la sémantique — de manière à ce qu'il n'y ait qu'un seul `return` en dernière ligne de la méthode. Vous ne devez pas utiliser de variable `boolean` ni faire usage d'instructions comme `break` que nous n'avons d'ailleurs pas vu en cours.

```
private int search(int [] t, int v) {
  for (int i=0; i<t.length; ++i)
    if (t[i] == v) return i;
  return -1;
}
```

(e) (3 points) Qu'affiche l'invocation `affiche1()` ?

```
private void affiche1() {
  for (int i=1; i<5; ++i) {
    int j = 0;
    while (j<i)
      System.out.print(j++ + " ");
  }
  System.out.println();
}
```

(f) (3 points) Qu'affiche l'invocation `affiche2()` ?

```
private void affiche2() {
  int i=0;
  while (i<5) {
    for (int j=i; j>1; j--)
      System.out.print(j + " ");
    System.out.println("***");
    ++i;
  }
}
```

2. (16 points) — Récursivité

   Considérez le code suivant:

```
private static int quiz1(int i, int j, int s) {
  if (i > j) return s;
  if (0 == i % 2)
    return quiz1(i+1,j,s+i);
  return quiz1(i+1,j,s);
}
private static int quiz1(int i) { return quiz1(0,i,0); }
```

   (a) (4 points) Qu'affiche l'exécution de:

```
System.out.println(quiz1(10));
```

   (b) (2 points) Qu'affiche l'exécution de:

```
System.out.println(quiz1(0,1,10));
```

   (c) (2 points) Que retourne l'appel: `quiz1(0)` ?

   (d) (2 points) De manière générale, que retourne l'appel: `quiz1(i)` pour un entier `i` strictement positif ?

(e) (2 points) De manière générale, que retourne l'appel: `quiz1(i)` pour un entier `i` strictement négatif ?

(f) (4 points) Écrire une méthode récursive `affiche` qui prend en argument un entier positif `n` et qui affiche une chaîne **aléatoire** de longueur `n` constituée de `1` et de `0` telle que la chaîne soit palindrome. Vous ne devez pas utiliser de boucle. Vous pouvez vous aider d'une seconde méthode si vous en éprouvez le besoin.

Par exemple, l'appel `affiche(6)` pourrait afficher `100001` ou `011110` tandis que l'appel `appel(7)` pourrait afficher `1010101` ou `0010100`.

3. (20 points) — Algorithmique

   Considérez la grille suivante (vous noterez que toutes les lignes n'ont pas le même nombre de colonnes):

```
F O R D Z
T J M I A I F G
R N A O E
I F I V C
X Y N T A
```

   (a) (3 points) Écrire un code Java qui crée cette grille sous la forme d'un tableau de `char` à 2 dimensions (vous ne devez pas utiliser de `String`).

   (b) (4 points) Écrire une méthode:
   `droite(char [][] grille, int i, int j, String mot)`
   qui retourne `true` si `mot` apparaît depuis la case `(i,j)` en allant vers la droite, et `false` sinon. Par exemple `droite(grille,0,0,"FOR")` devrait retourner `true` pour la grille représentée ci-dessus, ainsi que `droite(grille,1,5,"IF")`.

(c) (6 points) Écrire une méthode:
`isPresent(char [][] Grille, int i, int j, String mot)`
qui généralise l'idée précédente en retournant `true` si `mot` est présent depuis la case `(i,j)` dans l'une des 8 directions (droite, gauche, haut, bas, ainsi que les 4 diagonales) et `false` sinon.

Par exemple `isPresent(grille,1,2,"MAIN")` retournera `true` car `MAIN` débute en case `(1,2)` en direction du bas. De même `isPresent(grille,1,1,"JAVA")` retournera `true` car `JAVA` apparaît en case `(1,1)` en diagonale vers la gauche et vers le bas.

Vous ne devez pas écrire 8 méthodes équivalentes conceptuellement à la méthode `droite`, mais généraliser le concept sous-jacent. Vous pouvez vous aider d'au plus une méthode intermédiaire.

(d) (3 points) Écrire une méthode:
`isPresent(char [][] grille, String mot)`
qui retourne `true` si `mot` est présent dans `grille` et `false` sinon.

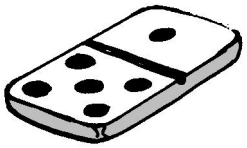
Par exemple `isPresent(grille,"JAVA")` retournera `true` tandis que `isPresent(grille,"VACANCES")` retournera `false`.

(e) (4 points) Écrire une méthode:
`String [] trouve(char [][] grille, String [] mots)`
qui cherche dans `grille` les mots de `mots` et qui retourne un tableau contenant l'ensemble des mots de `mots` trouvés dans la grille `grille`.

Par exemple, le code suivant affichera `[JAVA, MAIN, VOID, INT, TRI, IF, FOR]`:

```
String [] mots= {"JAVA", "MAIN", "ALEA",
                 "VOID", "INT", "TRI", "IF", "FOR"};
java.util.Arrays.toString(trouve(grille, mots));
```

4. (24 points) — POO – Dominos



Vous allez dans ce problème simuler une version simplifiée du jeu de domino où deux joueurs posent à tour de rôle un domino sur le tapis. Le premier joueur à se débarrasser de tous ses dominos gagne la partie. Un domino est une paire de valeurs entières entre `1` et `domaine`[1] (inclus) que l'on représentera sous la forme `(i,j)` où `i` et `j` sont les deux valeurs du domino. Par exemple `(5,1)` et `(1,5)` représentent le domino de la figure ci-dessus.

Le jeu commence par une distribution aléatoire d'un nombre égal de dominos à chaque joueur. Un premier domino (également tiré aléatoirement) est de plus déposé sur le tapis. Un joueur peut poser un de ses dominos sur le tapis si et seulement si il possède un domino dont l'une des valeurs est égale à la valeur droite du domino le plus à droite du tapis[2]. Ainsi, si sur le tapis le dernier domino posé est `(5,1)` alors le joueur pourra y accoler un domino dont l'une des valeurs est `1`. Si un joueur pose un *double*, c'est-à-dire un domino dont les deux valeurs sont égales, alors ce joueur rejoue. Si un joueur ne peut joueur, il passe son tour. Le premier joueur qui dépose l'ensemble de ses dominos gagne la partie.

[1] Habituellement, le domaine est `6`.

[2] Simplification des règles habituelles du jeu.

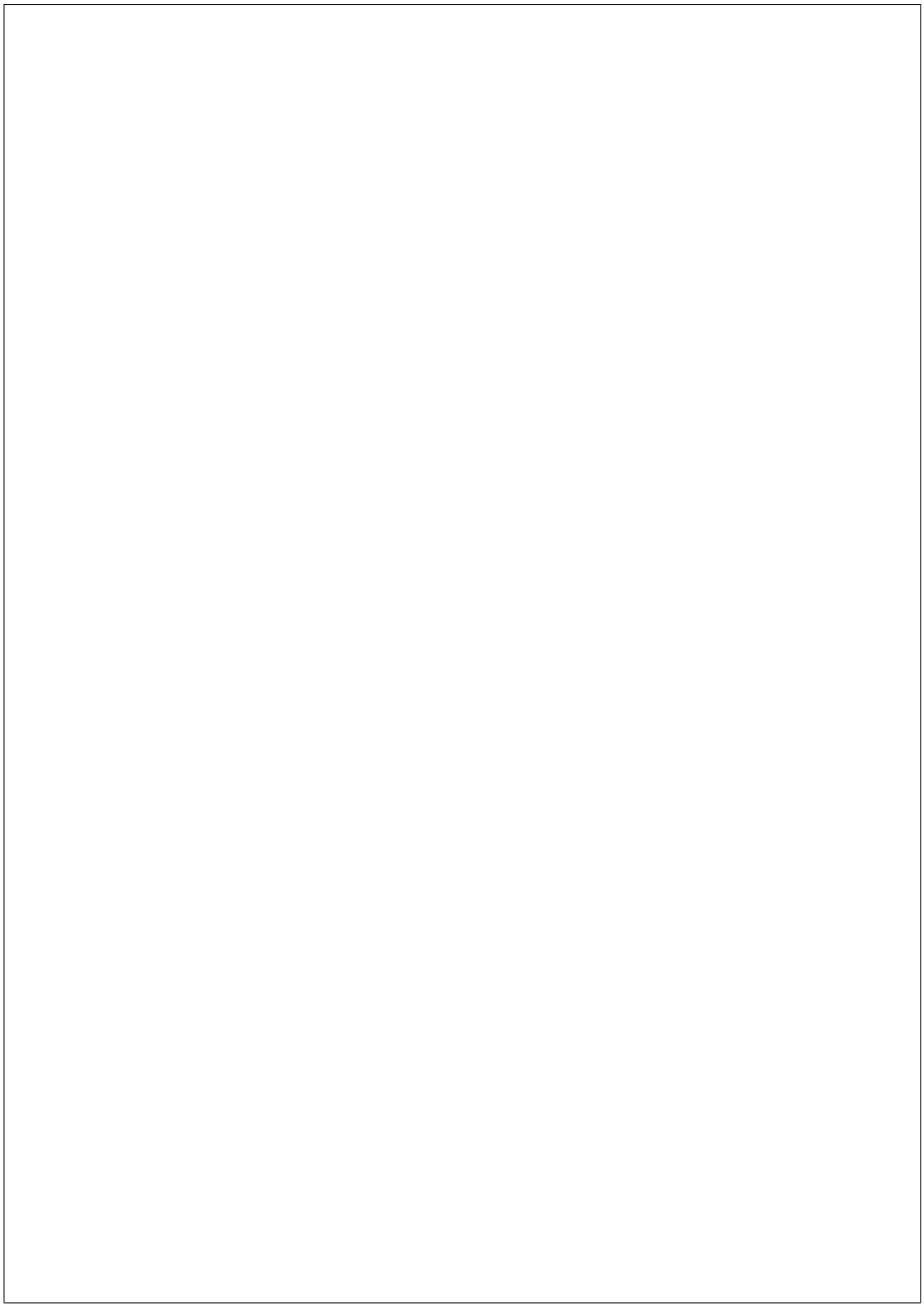
Conseil: lisez tout l'énoncé du problème avant de commencez à répondre aux questions posées. Posez vous des questions sur l'organisation du problème en classes. En particulier, rien dans le sujet ne vous indique où l'intelligence du jeu (quel domino choisit un joueur, etc.) doit être programmée. Les contraintes des classes qui sont exprimées dans la suite ne sont que partielles et vous aurez à ajouter à certaines classes des méthodes de votre cru. N'hésitez surtout pas à commenter certains éléments de votre code. Dans certaines situations, un commentaire bien pensé vaut autant de point qu'un code mal écrit.

(a) (5 points) Écrire une classe `Domino` capable de représenter un domino. Voici du code qui devrait fonctionner avec votre classe; vous êtes libres d'étendre cette classe comme bon vous semble.

```
// crée un domino aléatoirement avec des valeurs entre 1 et 6:
Domino d1 = new Domino();
// crée un domino aléatoirement avec des valeurs entre 1 et 10:
Domino d2 = new Domino(10);

System.out.println(d1); // affiche le domino sous forme d'une paire (ex: (4,2))
System.out.println(d2); // affiche le domino sous forme d'une paire (ex: (4,8))

int v1 = d1.getFace(1); // retourne la valeur de l'un des côtés du domino d1
int v2 = d1.getFace(2); // retourne la valeur de l'autre côté du domino d1
```



(b) (8 points) Écrire une classe `Joueur` capable de représenter un joueur. Votre classe devra être compatible avec ce code:

```
// crée un joueur Bob avec 20 dominos tirés aléatoirement
Joueur j1 = new Joueur("Bob",20);
// crée un joueur Jean avec 10 dominos tirés aléatoirement
Joueur j2 = new Joueur("Jean");
// affichera Jean possède 10 dominos
System.out.printf("%s possède %d dominos\n", j1.getName(), j1.size());
```

(c) (5 points) Écrire une classe `Tapis` capable de représenter le tapis du jeu.Votre classe devra être compatible avec ce code:

```
// crée un tapis de jeu avec un domino tiré aléatoirement (valeurs entre 1 et 6)
Tapis tapis = new Tapis(new Domino());
// affiche le tapis (la suite de dominos)
System.out.println(tapis);
```

(d) (6 points) Écrire une fonction `main` qui crée deux joueurs `Rolf` et `Gisela` possédant chacun `N` dominos, où `N` sera lu depuis la ligne de commande. Votre fonction a pour but de faire jouer les joueurs à tour de rôle (rappel: un double permet au joueur qui le pose de rejouer) jusqu'à ce qu'un joueur gagne, c'est-à-dire qu'il se débarrasse de tous ses dominos, ou qu'aucun joueur ne puisse plus continuer, auquel cas il y a match nul. Vous choisirez au hasard le joueur qui commencera. Voici une trace d'exécution que vous devrez suivre: vous devez indiquer la même information que celle donnée ici, à savoir les dominos de chaque joueur, les dominos posés sur le tapis, le nom du joueur qui joue à un moment donné et le coup qu'il joue, ainsi que le gagnant ou les ex-aequo en cas de match nul.

Aucune contrainte ne vous est imposée quant au choix du domino qu'un joueur va poser. Vous pouvez décider de poser le premier domino du joueur qui peut s'ajouter au tapis si vous le voulez. Rappelez vous qu'un domino ne peut être posé qu'à la suite (droite) du dernier domino déjà posé sur le tapis.

```
Rolf a 10 dominos: (2,2)(2,5)(1,2)(1,6)(3,3)(2,3)(3,4)(2,2)(4,4)(2,2)
Gisela a 10 dominos: (6,4)(3,4)(6,5)(1,5)(4,1)(4,6)(3,2)(6,6)(5,5)(2,6)
round 1: tapis = (2,1)
Rolf joue le domino (1,2)
Gisela joue le domino (3,2)
round 1 : Rolf: 9 -- Gisela: 9
round 2: tapis = (2,1)(1,2)(2,3)
Rolf joue le domino (3,3)
Rolf joue le domino (2,3)
Gisela joue le domino (2,6)
round 2 : Rolf: 7 -- Gisela: 8
round 3: tapis = (2,1)(1,2)(2,3)(3,3)(3,2)(2,6)
Rolf joue le domino (1,6)
Gisela joue le domino (1,5)
round 3 : Rolf: 6 -- Gisela: 7
round 4: tapis = (2,1)(1,2)(2,3)(3,3)(3,2)(2,6)(6,1)(1,5)
Rolf joue le domino (2,5)
round 4 : Rolf: 5 -- Gisela: 7
round 5: tapis = (2,1)(1,2)(2,3)(3,3)(3,2)(2,6)(6,1)(1,5)(5,2)
Rolf joue le domino (2,2)
Rolf joue le domino (2,2)
Rolf joue le domino (2,2)
round 5 : Rolf: 2 -- Gisela: 7
round 6: tapis = (2,1)(1,2)(2,3)(3,3)(3,2)(2,6)(6,1)(1,5)(5,2)(2,2)(2,2)(2,2)
round 6 : Rolf: 2 -- Gisela: 7
match null: il reste 2 dominos a Rolf et 7 a Gisela
tapis:(2,1)(1,2)(2,3)(3,3)(3,2)(2,6)(6,1)(1,5)(5,2)(2,2)(2,2)(2,2)
Rolf a 2 dominos: (3,4)(4,4)
Gisela a 7 dominos: (6,4)(3,4)(6,5)(4,1)(4,6)(6,6)(5,5)
```