Université d'Ottawa
Faculté de génie

École de science informatique
et de génie électrique

uOttawa

University of Ottawa
Faculty of Engineering

School of Electrical Engineering
and Computer Science

Introduction à l'informatique II (ITI1521)
Examen mi-session

Instructeur: Marcel Turcotte

Mars 2012, durée : 2 heures

Identification

Nom, prénom : _____

Numéro d'étudiant : _____ Signature : _____

Consignes

1. Examen à livres fermés ;
2. Sans calculatrice ou toute autre forme d'aide. L'utilisation des appareils électroniques ou tout autre dispositif de communication n'est pas autorisé. Tout appareil doit être éteint, rangé et hors de portée ;
3. Répondez sur ce questionnaire. Utilisez le verso des pages si nécessaire, mais vous ne pouvez remettre aucune page additionnelle ;
4. Écrivez lisiblement, puisque votre note en dépend ;
5. Commentez vos réponses ;
6. **Ne retirez pas l'agrafe**.

Barème

Question	Maximum	Résultat
1	15	
2	30	
3	25	
Total	**70**	

Question 1 (15 points)

Un jeu de cartes comprends 52 cartes. Chaque carte possède une couleur (**suit**) et un rang (**rank**). Il y a quatre couleurs : pique (**spades**), coeur (**heart**), carreau (**diamond**) et trèfle (**club**), ainsi que 13 rangs : as (**ace**), 2, 3, 4, 5, 6, 7, 8, 9, 10, valet (**jack**), reine (**queen**) et roi (**king**). Les tableaux ci-dessous donnent les valeurs entières utilisées afin de représenter les couleurs et les rangs des cartes pour cette question.

```
int[] suits = { 4, 3, 2, 1 };
int[] ranks = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13 };
```

Où 4 désigne le pique, 3 désigne le coeur, 2 désigne le carreau et 1 désigne le trèfle. Pour les rangs, 1 désigne l'as, $2 \ldots 10$, désignent les rangs deux à dix, 11 désigne le valet, 12 désigne la reine et 13 désigne le roi.

Étant donné la représentation des couleurs et des rangs ci-dessus, vous devez concevoir en Java une méthode de classe (**static**) nommée **getDifference**. Celle-ci devra calculer la différence de pointage entre la couleur la plus fréquente et la couleur la moins fréquente d'une main de n cartes. Le pointage pour une couleur donnée est la somme des pointages des cartes de la main ayant cette couleur. Le pointage pour une carte est le produit de sa couleur par son rang.

L'exemple qui suit représente une main de cinq cartes : as de coeur, 10 de pique, valet de carreau, 3 de trèfle, et la reine de coeur, à l'aide de deux tableaux de valeurs entières :

```
int[] handRanks = { 1, 10, 11, 3, 12 };
int[] handSuits = { 3,  4,  2, 1,  3 };
```

Lorsque deux couleurs ont la même fréquence, nous choisirons la première occurrence. Dans cet exemple, le coeur est la couleur la plus fréquente puisque la valeur 3 apparait deux fois. Le pique est la couleur la moins fréquente puisque la valeur 4 n'apparait qu'une fois.

En supposant que la méthode **getDifference** est déclarée dans la classe **Q1**, alors l'exécution des énoncés ci-dessous affichera la valeur 53.

```
int[] handRanks = { 1, 10, 11, 3, 12, 2, 6, 10, 5, 12 };
int[] handSuits = { 2,  3,  1, 4,  2, 2, 4,  3, 2,  2 };
System.out.println( Q1.getDifference( handRanks, handSuits, 10 ) );
```

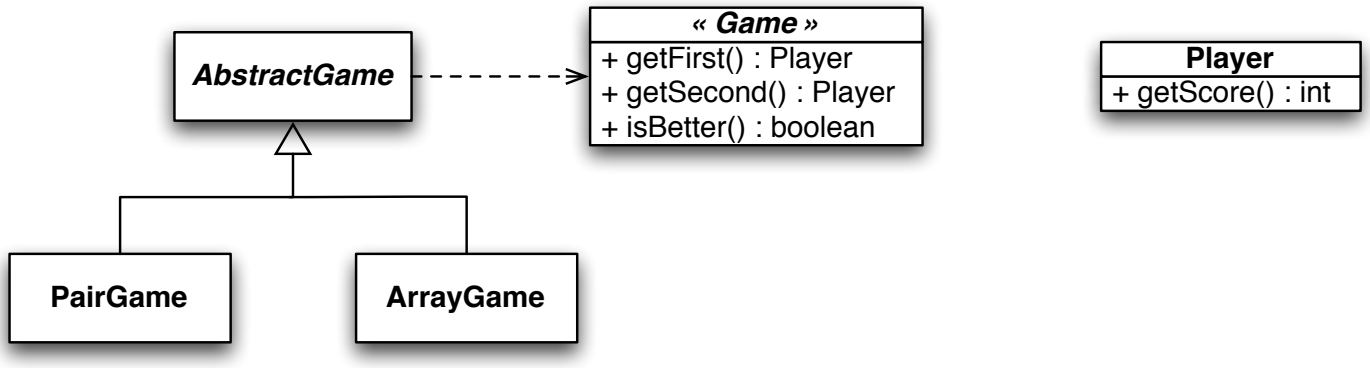
Cette valeur est obtenue comme suit : $53 = 2 \times (1+12+2+5+12) - 1 \times 11$. Supposez que les références **handRanks** et **handSuits** sont non-null, et que le contenu des deux tableaux désignés est valide.

```
public static int getDifference( int[] handRanks, int[] handSuits, int n ) {









}
```

Question 1 (suite)

Question 2 (30 points)

Pour assister la conception de jeux à deux participants, vous décidez de créer les classes et l'interface représentées par les diagrammes UML ci-dessous.



Il y a une interface nommée **Game**, une classe abstraite nommée **AbstractGame**, ainsi que deux implémentations concrètes, **PairGame** et **ArrayGame**. Un objet de la classe **Player** possède une méthode **getScore** qui retourne un entier représentant le pointage courant de ce joueur. Vous trouverez des informations complémentaires dans les pages qui suivent.

Pour un jeu à deux participants, il vous faudra sauvegarder des informations au sujet des deux participants (des objets de la classe **Player**). Ainsi, il y aura des méthodes **getFirst** et **getSecond** qui retourneront une référence vers le premier et second participant, respectivement. Il y aura aussi une méthode **isBetter** qui doit retourner **true** si le pointage du premier participant est supérieur au pointage du second, et **false** sinon.

L'exécution des énoncés Java ci-dessous produira le résultat qui suit en sortie : **36 is a lower score than 40**.

```
Player p1, p2, p3, p4;

p1 = new Player( 36 );
p2 = new Player( 25 );
p3 = new Player( 40 );
p4 = new Player( 28 );

Game g1, g2;

g1 = new PairGame( p1, p3 );
g2 = new ArrayGame( p2, p4 );

if ( ! g1.isBetter() ) {
    System.out.print( g1.getFirst().getScore() );
    System.out.println( " is a lower score than " + g1.getSecond().getScore() );
}

if ( g2.isBetter() ) {
    System.out.print( g2.getFirst().getScore() );
    System.out.println( " is a higher score than " + g2.getSecond().getScore() );
}
```

Assurez-vous d'inclure dans vos solutions tous les constructeurs et les méthodes d'accès nécessaires à l'exécution des énoncés ci-dessus.

A. Vous devez concevoir l'interface **Game**. L'interface déclare trois méthodes. Il y a deux méthodes d'accès, nommées **getFirst** et **getSecond**, elles retournent toutes deux une référence de type **Player**. Finalement, l'interface déclare aussi une méthode appelée **isBetter** qui retourne une valeur booléenne. (5 points)

B. Vous devez implémenter la classe abstraite **AbstractGame**. Cette dernière réalise l'interface **Game**. Elle possède une méthode concrète **isBetter**, qui retourne **true** si le pointage du premier participant est supérieur au pointage du second, et **false** sinon. (7 points).

C. Donnez l'implémentation de la classe (concrète) **PairGame**. Cette dernière est dérivée de la classe **AbstractGame**. Pour cette implémentation, vous devez utiliser deux variables d'instance afin de sauvegarder la référence du premier et du second participant (objets de la classe **Player**). Ajoutez tous les constructeurs et toutes les méthodes d'accès nécessaires. (8 points)

D. Donnez l'implémentation de la classe (concrète) **ArrayGame**. Cette dernière est dérivée de la classe **AbstractGame**. Pour cette implémentation, vous devez utiliser un tableau de taille 2 afin de sauvegarder la référence du premier et du second participant (objets de la classe **Player**). Ajoutez tous les constructeurs et toutes les méthodes d'accès nécessaires. (10 points)

Question 3 (25 points)

A. (5 points) En respectant les consignes présentées en classe, et décrites dans les notes de cours, dessinez les diagrammes de mémoire pour tous les objets et toutes les variables locales de la méthode **Q3.test** à la suite de l'exécution de l'énoncé « **second = null;** ».

```
public class Player {
    private String name;
    private int score = 0;
    public Player( String name, int score ) {
        this.name = name;
        this.score = score;
    }
}
public class Elem {
    private Player player;
    private Elem next;
    public Elem( Player player, Elem next ) {
        this.player = player;
        this.next = next;
    }
}
public class Q3 {
    public static void test() {
        String name;
        int score;
        Player player;
        Elem first, second;
        name = "Diddy";
        score = 45;
        player = new Player( name, score );
        second = new Elem( player, null );
        first = new Elem( new Player( "Lanky", 32 ), second );
        second = null;
    }
}
```

Réponse :

B. (2 points) On peut assigner à une variable de type **T** la référence d'un objet de la classe **S** si **S** est une sous-classe de **T**. **Vrai** ou **faux**.

Réponse :

C. (2 points) Java définit automatiquement un constructeur

(a) si le programmeur ne définit pas de constructeur pour une classe

(b) si le programmeur ne définit pas de constructeur par défaut pour une classe

(c) si le programme fait référence au constructeur d'arité 0

(d) pour toutes les classes

Réponse :

D. (3 points) _____ permet la création de nouvelles classes à partir de classes existantes.

(a) Le polymorphisme

(b) L'héritage

(c) La surcharge de nom

(d) Un constructeur

(e) Aucune de ces réponses

Réponse :

E. (3 points) Quels énoncés sont vrai ?

(I) Une interface ne contient que des méthodes abstraites et des constantes.

(II) Un classe qui réalise une interface, mais qui n'implémente pas une ou plusieurs méthodes de l'interface, doit être déclarée abstraite.

(III) Bien qu'une classe ne puisse implémenter qu'une interface, elle peut posséder plus d'un parent.

a. I seulement

b. I et II seulement

c. I et III seulement

d. II et III seulement

e. I, II, et III

Réponse :

F. (2 points) Cette déclaration de classe produira une erreur de compilation. Pourquoi ?

```
public class Vehicle {
    public abstract int getVIN();
    // ...
}
```

Réponse :

G. (3 points) Donnez le résultat affiché sur la sortie lorsqu'on exécute la méthode **main**.

```
public class Information {
  private int a;
  private static int b;

  public Information() {
      b++;
  }

  public void revealInfo() {
      System.out.println( "Value of a = " + a );
      System.out.println( "Value of b = " + b );
  }

  public static void main( String[] args ){
      Information i1 = new Information();
      i1.revealInfo();
      Information i2 = new Information();
      i2.revealInfo();
  }
}
```

Réponse :

H. (5 points) Identifiez quatre (4) erreurs de compilation ainsi qu'une erreur de logique dans le programme Java ci-dessous.

```
public class LinkedStack {

    private static class Node<E> {

        private E value;
        private Node<E> next;

        private void Node( E value, Node<E> next ) {
            this.value = value;
            next = next;
        }

    }

    private Node<T> top;

    public String toString() {

        String result = "";

        private Node<T> p;

        p = top;

        while ( p != 0 ) {
             if (p > 0) {
                result = result + ", ";
             }
             result = result + p.value;
             p++;
        }

        return result;
    }

}
```