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

Instructeurs: Guy-Vincent Jourdan, Nour El-Kadri et Marcel Turcotte

Février 2016, durée: 2 heures

Identification

Nom de famille : _____ Prénom(s) : _____

Étudiant : _____ Signature : _____

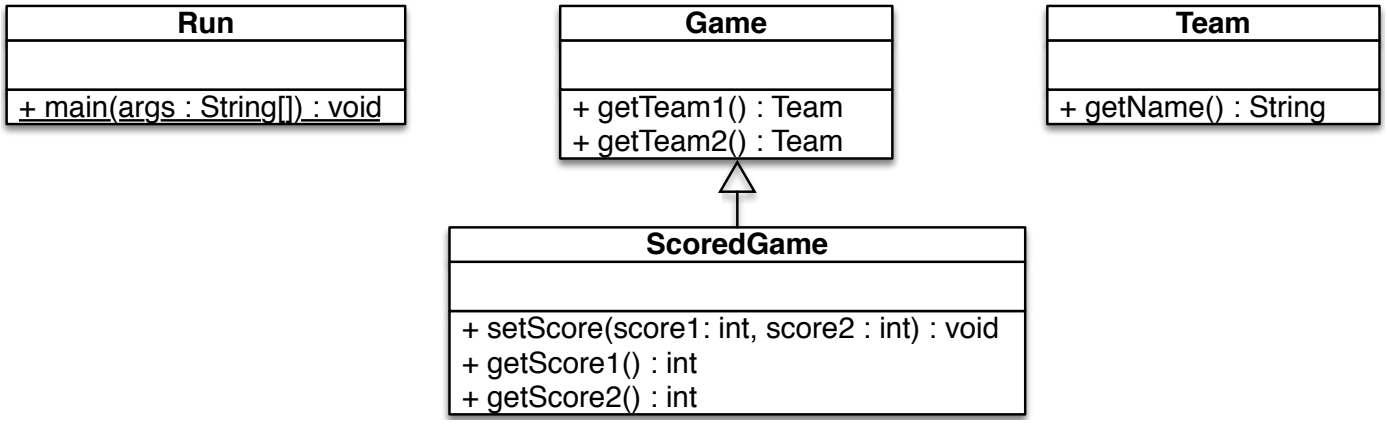
Instructions

1. Examen à livres fermés.
2. L'utilisation de calculatrices, d'appareils électroniques ou tout autre dispositif de communication est interdit.
   (a) Tout appareil doit être éteint et rangé.
   (b) Toute personne qui refuse de se conformer à ces règles pourrait être accusée de fraude scolaire.
3. Répondez sur ce questionnaire.
   (a) Utilisez le verso des pages si nécessaire.
   (b) Aucune page supplémentaire n'est permise.
4. Écrivez vos commentaires et hypothèses afin d'obtenir des points partiels.
5. Écrivez lisiblement, puisque votre note en dépend.
6. Ne retirez pas l'agrafe du livret d'examen.
7. Attendez l'annonce de début de l'examen.

Barème

Question	Maximum	Résultat
1	10	
2	15	
3	20	
4	20	
Total	**65**	

Question 1 (10 points)



Vous devez implémenter les classes **Team**, **Game**, et **ScoredGame**, en fonction du diagramme UML ci-dessus et des instructions qui suivent. Vous devez aussi vous assurer que votre code fonctionne avec la classe **Run** ci-dessous.

A. Un objet de la classe **Team** sauvegarde le nom d'une équipe sportive. Assurez-vous d'y inclure une méthode d'accès en lecture («*getter*») pour cet attribut.

B. La classe **Game** décrit les caractéristiques qui sont communes à tous les sports. Notamment, un sport a deux équipes (objets de la classe **Team**). Vous devez inclure les méthodes d'accès en lecture («*getters*») pour ces attributs.

C. Finalement, la classe **ScoredGame** est une spécialisation de la classe **Game**. Un objet de la classe **ScoredGame** sauvegarde le pointage («*score*») de chacune des deux équipes. Vous devez fournir toutes les méthodes d'accès nécessaires. Consultez le diagramme UML et le code source de la classe **Run** ci-dessous pour connaitre les signatures exactes de ces méthodes.

```
public class Run {

    public static void main(String[] args) {

        Team senators, canadiens;

        senators = new Team("Ottawa Senators");
        canadiens = new Team("Montreal Canadiens");

        ScoredGame s;

        s = new ScoredGame(senators, canadiens);
        s.setScore(5, 0);

    }

}
```

A. Donnez votre implémentation de la classe **Team** dans la boîte ci-dessous.

B. Donnez votre implémentation de la classe **Game** dans la boîte ci-dessous.

C. Donnez votre implémentation de la classe **ScoredGame** dans la boîte ci-dessous.

Question 2 (15 points)

A. Réparez 5 erreurs du programme Java ci-dessous, de sorte que la compilation et l'exécution du code ne causent aucune erreur. Cette méthode calcule et retourne la factorielle de la valeur du paramètre. Supposez que la valeur du paramètre sera toujours plus grande ou égale à 0. Juxtaposez les étiquettes **a**, **b**, **c**, **d**, et **e** sur le code source, là ou se trouve l'erreur, et donnez la version modifiée du code sur la ligne correspondante. Il pourrait y avoir plus d'une erreur par ligne de code.

```
public class Utils {

    public static factorial(n) {

        result = 1;

        for (i=1; i<=n; i++) {
            result = i * result;
        }

        result;
    }
}
```

(a) _____

(b) _____

(c) _____

(d) _____

(e) _____

B. Dans la boîte ci-dessous, indiquez pourquoi la compilation des classes **Person** et **Child** sera un échec. Encerclez le ou les énoncés qui sont la source du problème.

```
public class Person {
    private String name;
    private int age;
}
```

```
public class Child extends Person {
    private int grade;
    public Child(String name, int age, int grade) {
        this.name = name;
        this.age = age;
        this.grade = grade;
    }
}
```

C. Étant donné la déclaration de la classe **Cell** ci-dessous.

```
public class Cell<E> {

    private E value;

    public Cell(E value) {
        this.value = value;
    }

    public boolean isEqual(Cell<E> other) {

        if (other == null) {
            return false;
        }

        if (this.value == null || other.value == null) {
            return this.value == null && other.value == null;
        }

        return this.value.equals(other.value);
    }

}
```

(a) Déclarez deux variables références, que vous nommerez respectivement **s** et **t**, et qui serviront à désigner des objets de la classe **Cell** pour sauvegarder des valeurs de type **String**.

(b) Donnez les énoncés Java pour créer deux objets de la classe **Cell** pour sauvegarder des valeurs de type **String**, sauvegardez les références de ces objets dans les variables références **s** et **t** déclarée ci-dessus. Le premier objet doit être initialisé avec la chaîne de caractères `"Suki"` alors que le second doit être initialisé avec la chaîne `"Juliette"`.

(c) Donnez l'expression Java pour appeler la méthode **isEqual** de l'objet désigné par **s** avec comme paramètre actuel la référence de l'objet désigné par **t**.

(d) Déclarez une interface nommée **Taxable**. Un objet **Taxable** possède une méthode **getTaxes** qui retourne une valeur de type **double**.

Question 3 (20 points)

L'interface **ExtendedStack** ci-dessous déclare les méthodes habituelles pour une pile, mais aussi les méthodes **count**, qui retourne le nombre d'éléments sauvegardés dans la pile, et la méthode **reverse**, qui renverse l'ordre des éléments de la pile.

```
public interface ExtendedStack {
    boolean isEmpty();
    void push(String element);
    String pop();
    String peek();
    int count();
    void reverse();
}
```

Pour la classe **ExtendedStackImplementation** ci-dessous, vous devez compléter l'implémentation des méthodes **peek**, **count**, et **reverse**.

- La classe **ExtendedStackImplementation** réalise l'interface **ExtendedStack**.
- La classe **ExtendedStackImplementation** possède un seul constructeur et ce dernier n'a aucun paramètre.
- Supposez que le constructeur et les méthodes **isEmpty**, **push**, et **pop** de la classe **ExtendedStackImplementation** ont déjà été implémentées (sans erreurs) et que vous pouvez donc les utiliser.
- Cette implémentation permet de sauvegarder un nombre arbitrairement grand d'éléments.
- Cependant, ne faites aucune supposition sur l'implémentation de la classe **ExtendedStackImplementation**. En particulier, ne supposez pas qu'elle utilise un tableau.

N'utilisant que les méthodes de l'interface **ExtendedStack** qui ont déjà été implémentées (**isEmpty**, **push**, et **pop**), donnez l'implémentation des méthodes **peek**, **count**, et **reverse**. Vous n'avez pas à traiter les situations d'erreur, en particulier, supposez qu'il n'y aura pas d'appels à méthode **peek** lorsque la pile est vide.

La méthode **peek** retourne l'élément du dessus de la pile, sans pour autant le retirer. Plus pécisément, le contenu de pile doit être le même avant et après un appel à la méthode **peek**.

```
public class ExtendedStackImplementation implements ExtendedStack {

    // ...

    public String peek() {









   }

}
```

En n'utilisant que les méthodes qui ont déjà été implémentées, donnez l'implémentation de la méthode **count**. La méthode **count** retourne l'élément qui se trouvent dans la pile. Le contenu de pile doit être le même avant et après un appel à la méthode **count**.

```
public class ExtendedStackImplementation implements ExtendedStack {

    // ...

    public int count(){









    }
}
```

Finalement, toujours à l'aide des méthodes déjà implémentées, donnez l'implémentation de la méthode **reverse**, qui renverse l'ordre des éléments de la pile (c'est-à-dire, qui conserve les mêmes éléments, mais inverse l'ordre de ceux-ci dans la pile.)

```
public class ExtendedStackImplementation implements ExtendedStack {
...
    public void reverse(){









    }
}
```

Question 4 (20 points)

Vous devez concevoir une classe afin de représenter une permutation. Ici, une permutation est un arrangement (donc sans répétition) des entiers 0 à $n-1$, où n est la taille («*size*») de la permutation. On nomme **permutation-identité**, la permutation constituée des entiers 0 à $n-1$ en ordre. Par exemple, $0, 1, 2, 3, 4$ est la permutation-identité de taille 5.

A. Donnez l'implémentation de la classe **Permutation**. Ajoutez la ou les variables d'instance nécessaires.

B. Implémentez le constructeur **Permutation(int size)**. Ce constructeur initialise cette permutation afin qu'elle soit une permutation-identité de taille **size**.

C. Implémentez la méthode d'accès en lecture **int getSize()**, qui retourne la taille de cette permutation.

D. Implémentez la méthode **int get(int pos)**, qui retourne l'élément à la position **pos** de cette permutation. Supposez que la valeur du paramètre sera toujours un entier plus grand ou égal à zéro, mais plus petit que la taille de la permutation.

E. Donnez l'implémentation de la méthode **rotate(int n)**. Cette méthode transforme la permutation en décalant ses éléments de **n** positions vers la droite. Les **n** éléments de la partie droite deviennent les **n** éléments de la partie gauche de la permutation. Vous trouverez un exemple d'appel et de sortie ci-dessous.

F. Implémentez la méthode **shuffle** qui réarrange les éléments de la permutation de façon aléatoire. Utilisez votre temps intelligemment. Vous souhaitez peut-être revenir sur cette question une fois que vous aurez complété le reste de l'examen.

Voici un exemple de l'utilisation de la classe **Permutation** :

```
Permutation p;
p = new Permutation(5);

for(int i = 0; i < p.getSize(); i++) {
    System.out.print(" p[" + i + "] = "+ p.get(i));
}

p.rotate(2);

System.out.println("\n after rotation of 2:");

for(int i = 0; i < p.getSize(); i++) {
    System.out.print(" p[" + i + "] = "+ p.get(i));
}
```

Le programme ci-dessus produira la sortie suivante sur la console.

```
 p[0] = 0 p[1] = 1 p[2] = 2 p[3] = 3 p[4] = 4

 after rotation of 2:

 p[0] = 3 p[1] = 4 p[2] = 0 p[3] = 1 p[4] = 2
```

```
import java.util.Random;
```