Université d'Ottawa
Faculté de génie

École d'ingénierie et de
technologie de l'information

uOttawa

L'Université canadienne
Canada's university

University of Ottawa
Faculty of Engineering

School of Information
Technology and Engineering

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

Instructeur: Miguel Garzón

Mars 2013, durée: 2 heures

Identification

Nom, prénom : _____

Numéro d'étudiant : _____ Signature : _____

Instructions

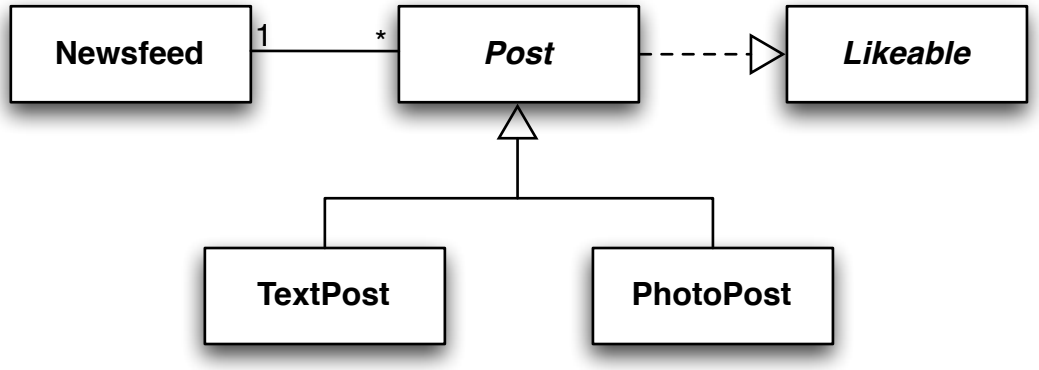
1. Examen à livres fermés ;
2. Sans calculatrice ou toute autre forme d'aide. L'utilisation des appareils électroniques ou tout autre dispositif de communication n'est pas autorisé. Tout appareil doit être éteint, rangé et hors de portée ;
3. Répondez sur ce questionnaire. Utilisez le verso des pages 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	35	
2	15	
3	15	
Total	**65**	

Question 1 (35 marks)

La compagnie **Gazouillis** veut implémenter un logiciel pour échanger des messages texte et des photos. On vous demande de créer l'implémentation initiale qui sera montrée aux investisseurs de la compagnie.



Le diagramme de classes UML ci-dessous donne une vue d'ensemble de l'application. Suivez les instructions suivantes :

A. Concevez l'interface **Likeable**. L'interface déclare deux méthodes : **like()** et **int getLikes()**. (5 points)

B. Écrivez l'implémentation de la classe abstraite **Post**. Elle implémente toutes les caractéristiques qui sont communes à ses sous-classes, **TextPost** et **PhotoPost**. (10 points)
- **Post** réalise l'interface **Likeable**.
- Tous les **Post** messages ont un nom d'utilisateur, une étiquette de temps (time stamp) (de type **java.util.Date**), en plus d'un compteur pour garder le nombre de 'likes'.
- La valeur de l'étiquette de temps est automatiquement assignée lorsqu'un objet est crée. Utilisez **java.util.Calendar.getInstance().getTime()** pour obtenir un objet **Date** représentant le temps actuel. L'objet **Date** a une méthode **toString()** qui convertit la date en une chaîne de caractères **String**.

```
Date rightNow = Calendar.getInstance().getTime();
System.out.println(rightNow);
```

- Chaque appel à la méthode **like()** incrémente le nombre de 'likes' pour ce message.

C. Donnez l'implémentation de la classe **PhotoPost**. La classe **PhotoPost** est dérivée de la classe **Post**. Cette classe utilise deux variables d'instance afin de sauvegarder le nom du fichier et la légende. (5 points)

D. Donnez l'implémentation de la classe **TextPost**. Cette dernière est dérivée de la classe **Post**. Elle sauvegarde un message texte. (5 points)

E. Donnez l'implémentation de la classe **NewsFeed**. Un objet **NewsFeed** sauvegarde un ensemble d'objets de type **Post**. (10 points)
- Elle utilise un tableau dynamique (tableau adapté automatiquement selon la taille des données) pour sauvegarder les messages **Post**.
- Le constructeur prend deux paramètres, la capacité initiale du tableau et l'incrément de la capacité.
- Chaque fois que le tableau est plein, l'implémentation doit créer un nouveau tableau dont sa taille sera déterminée par l'incrément.

- Cette classe a une méthode pour ajouter un message **Post**. Le message doit être ajouté à la suite du dernier message ajouté.
- Elle a une méthode qui retourne le message trouvé à l'index donné, **Post get(int index)**.
- Elle a une méthode **size** qui retourne le nombre de messages actuellement sauvegardés.

Ajoutez tous les constructeurs nécessaires. Chaque attribut doit avoir un accesseur (getter). Voici un programme test pour illustrer l'usage des classes de cette application. Vous pouvez supposer que tous les paramètres sont valides. Écrivez vos réponses dans les boîtes appropriées.

```
public class Test {

    public static void main(String[] args) {

        NewsFeed messages;
        Post msg1, msg2;

        messages = new NewsFeed(100, 10);

        msg1 = new PhotoPost("Ronaldo", "funny.png", "Birthday party");
        msg1.like();
        messages.add(msg1);

        msg2 = new TextPost("Ronaldo", "Dinner at your place with Pepe");
        msg2.like();
        msg2.like();

        messages.add(msg2);
        messages.add(new TextPost("Falcao", "Okay"));

        for (int i=0; i<messages.size(); i++) {
            System.out.println(messages.get(i));
        }

    }
}
```

L'exécution du programme Java ci-dessus produira le résultat qui suit en sortie :

```
PhotoPost: Sun Feb 24 11:31:08 2013, Ronaldo, likes = 1, funny.png, Birthday party
TextPost: Sun Feb 24 11:31:08 2013, Ronaldo, likes = 2, Dinner at your place with Pepe
TextPost: Sun Feb 24 11:31:08 2013, Falcao, likes = 0, Okay
```

Likeable

```
// End of Likeable
```

Post

// End of Post

TextPost

```
// End of TextPost
```

PhotoPost

// End of PhotoPost

NewsFeed

// End of NewsFeed

Question 2 (15 points)

Pour la classe **Q2**, vous devez implémenter la méthode de classe **boolean isReverse(Stack s1, Stack s2)**.

- La méthode **isReverse** retourne vrai (**true**) si et seulement si la pile désignée par **s1** contient les mêmes éléments de la pile désignée par **s2** ET les éléments dans la pile désignée par **s1** sont dans l'ordre inverse des éléments de la pile désignée par **s2**.
- À la suite d'un appel à la méthode **isReverse**, la pile désignée par **s1** doit contenir les mêmes éléments et dans le même ordre, comme c'était le cas avant l'appel de la méthode **isReverse**. Également, **s2** doit contenir les mêmes éléments et dans le même ordre, comme c'était le cas avant l'appel de la méthode **isReverse**.
- Vous pouvez assumer que **s1** et **s2** ne seront pas **null**.
- Une pile vide est considérée comme l'inverse d'une autre pile vide.
- Les paramètres de la méthode **isReverse** sont de type **Stack**, qui est une interface.

Pour cette question, considérez l'interface **Stack** suivante :

```
public interface Stack {
    public abstract void push(int item);
    public abstract int pop();
    public abstract boolean isEmpty();
}
```

- Remarquez que le paramètre de la méthode **push** et la valeur de retour de la méthode **pop** sont de type **int**.
- Vous ne pouvez pas utiliser des tableaux pour stocker les données temporaires.
- Assumez l'existence de la classe **DynamicStack**, qui réalise l'interface **Stack**. Cette classe possède un constructeur et sa signature est **DynamicStack()**.
- Vous devez utiliser des piles (objets de la classe **DynamicStack**) pour sauvegarder les données temporaires.
- Vous ne savez rien à propos de l'implémentation de la classe **DynamicStack**. En particulier, vous ne savez pas si cette classe utilise un tableau ou pas.
- Vous pouvez supposer que **DynamicStack** peut sauvegarder un grand nombre d'éléments.

Écrivez votre réponse dans la page suivante.

```
public class Q2 {
    public static boolean isReverse(Stack s1, Stack s2) {









    } // End of isReverse
} // End of Q2
```

(Question 2 suite)

Question 3 (15 points)

A. Questions Vrai ou Faux (5 points)

(a) Le programme Java ci-dessous affiche la valeur **true**.
Vrai ou **Faux**.

```
public class Test {
    public static void main(String[] args) {
        int[] a, b;
        a = new int [100];
        a[0] = 5;
        b = a;
        b[0] = 6;
        System.out.println( a == b );
    }
}
```

(b) Une variable locale est confinée à la méthode contenant sa déclaration.
Vrai ou **Faux**.

(c) Dans le code de la classe **Counter** ci-dessous, le mot clé **this** peut être omis sans affecter la compilation ou l'exécution du programme.
True ou **Faux**

```
public class Counter {
    private int value = 0;
    public Counter( int initialValue ) {
        this.value = initialValue;
    }
}
```

(d) Une variable de type référence **T**, peut référer à un objet de classe **S**, si **S** est une superclasse de **T**.
Vrai ou **Faux**

(e) Une classe peut réaliser plusieurs interfaces.
Vrai ou **Faux**

B. Questions à choix multiple (6 points)

(a) Les membres _____ d'une super classe ne sont jamais accessibles dans la classe dérivée.

i. public
ii. private
iii. protected
iv. a, b, et c
v. aucune des réponses ci-dessus

Réponse :

(b) Le(a) _____ nous permet de créer de nouvelles classes basées sur une classe déjà existante.

i. polymorphisme
ii. héritage
iii. surcharge d'une méthode
iv. le constructeur de copie
v. aucune des réponses ci-dessus

Réponse :

(c) Laquelle des expressions suivantes est la représentation postfixe de l'expression infixe :

$$((((9 + 7)/8) * 6) - 2)$$

i. 9 7 + 8 / 6 * 2 −
ii. 9 8 7 + / 6 * 2 −
iii. 9 7 8 6 2 + / * −
iv. 9 / 7 8 + 6 * 2 −

C. 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 ainsi que le paramètre de la méthode **Counts.main** à la suite de l'exécution de l'énoncé "**cs = new Counts();**" (4 points).

```
public class Word {

    private String word = null;
    private int count = 0;

    public Word(String word, int count) {
        this.word = word;
        count = count;
    }

}

public class Counts {

    private Word[] words;

    public Counts() {
        words = new Word[2];
        words[0] = new Word("ITI1121", 200);
        words[1] = new Word("ITI1521", 55);
    }

    public static void main(String[] args) {
        Counts cs;
        cs = new Counts();
        // here
    }

}
```

```
// Diagramme de memoire
```

(espace en blanc)