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 l'informatique II (ITI1521)
Examen de mi-session

Instructeurs: Sherif G. Aly, Nathalie Japkowicz, et Marcel Turcotte

22 Février 2015, durée: 2 heures

Identification

Nom de Famille: _____ Prénom: _____

Numéro d'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 accusé de fraude scolaire.
3. Répondez aux questions sur ce questionnaire.
   (a) Utilisez le verso des pages si nécessaire.
   (b) Aucune page supplémentaire n'est permise.
4. Ecrivez vos commentaires et hypothèses afin d'obtenir des points partiels.
5. Ne retirez pas l'agrafe du livret d'examen.
6. Écrivez lisiblement, puisque votre note en dépend.
7. Attendez l'annonce de début d'examen.

Barème:

Question	Maximum	Resultat
1	35	
2	15	
3	15	
Total	**65**	

Question 1 (35 points)

La Cryptographie a été utilisée depuis de milliers d'années afin de protéger les secrets. L' information qui requiert la protection peut être transformée en un format qui rend sa comprehension plus difficile en utilisant un processus qui s'appelle le "cryptage" ("encryption" en Anglais). Ce processus peut être inversé par un processus qui s'appelle le "decryptage" ("decryption" en Anglais).

Il existe de nombreux moyens très effectifs pour obtenir la protection des secrets les plus sensibles, cenpendant, deux moyens extrèmement simples, utilisés depuis très longtemps sont:

La Substitution : Où un texte est remplacé par un autre qui s'appelle le texte cipher.

Par exemple, si l'alphabet de votre texte est:

`ABCDEFGHIJKLMNOPQRSTUVWXYZ`

Et votre texte cipher est:

`ZEBRASCDFGHIJKLMNOPQTUVWXY`

La phrase "`COMMANDER`" sera crypté comme cela: "`BLJJZKRAO`".
Par conséquent, "`BLJJZKRAO`" sera décrypé comme cela: "`COMMANDER`".

(Veuillez noter que chaque caractère a été remplacé par le caractère correspondant dans le texte cipher.)

Le Caesar : Où chaque caractère de votre texte est remplacé par un autre caractère, qui se trouve à un nombre de positions fixes de ce caractère dans l'alphabet.

(Note: ne vous inquiétez pas des détails de ce moyens d'encryptage puisque nous ne l'étudierons pas en détail, ici)

Sur la page suivante, vous trouverez la description UML des classes nécessaires pour cette questions, ainsi qu'une description de chaque classe.

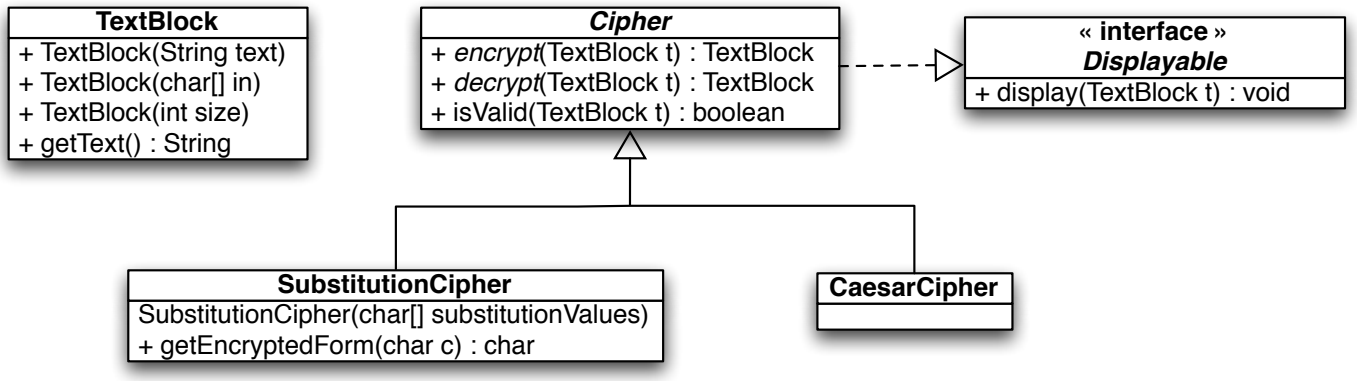
```
Cipher c = new SubstitutionCipher("ZEBRASCDFGHIJKLMNOPQTUVWXY".toCharArray());

TextBlock t = c.encrypt(new TextBlock("COMMANDER"));

c.display(t);
```

L'exécution du programme de Java ci-dessus imprime "`BLJJZKRAO`" à l'écran.

Veuillez étudier le diagramme de classes sur la page suivante, et suivre les instructions ci-dessous afin d'implémenter les programmes de Java pour cette question. Vous devez considérer seulement l'alphabet "normal" composé des caractères [A-Z].



TextBlock : Contiendra les méthodes suivantes:

- **TextBlock(String s)**: Un constructeur pour initialiser l'objet à partir d'une chaine de caractères donnée.
- **TextBlock(char[] in)**: Un constructeur pour initialiser l'objet à partir d'une séquence (d'un tableau) de caractères donnée.
- **TextBlock(int size)**: Un constructeur pour initialiser l'objet avec une chaine de caractères aléatoires (dans le domaine [A-Z]) de taille "size".
- **String getText()**: qui retourne le texte contenu dans le TextBlock.

Displayable : Contiendra la méthode suivante:

- **void display(TextBlock t)**: Sera utilisé pour afficher le text contenu dans un TextBlock. Ici, utilisez simplement **System.out.println**.

Cipher : Une class abstraite qui contiendra, au moins, les méthodes suivantes:

- **abstract TextBlock encrypt (TextBlock input)**
- **abstract TextBlock decrypt (TextBlock input)**
- **boolean isValid(TextBlock t)**: Cette méthode concrète retournera **true** si l'encryptage d'un **TextBlock**, suivi de son décryptage, vous donne le même **TextBlock** original avec lequel vous avez commencé. Elle retourn **false** si ca n'est pas le cas. En d'autres termes, cette méthode vérifie que "rien ne se gagne et rien ne se perd" dans le processus.

SubstitutionCipher : Hérite de **Cipher** et implémente **Displayable**. Contiendra, au moins, les méthodes suivantes:

- **SubstitutionCipher(char [] substitutionValues)**: qui construira la classe avec les caractères de substitution (donnés sur la page précédente) pour l'alphabet [A-Z].
- **TextBlock encrypt (TextBlock input)**
- **TextBlock decrypt (TextBlock input)**
- **char getEncryptedForm(char c)**: retournera la forme cryptée d'un caractère **c** du tableau substitutionValues.

CaesarCipher : Hérite de Cipher et implémente Displayable. On ne vous demande que de déclarer cette classe. Aucune implémentation n'est requise.

A. Montrez votre implémentation de **TextBlock** dans l'espace ci-dessous.

B. Montrez votre implémentation de **Displayable** dans l'espace ci-dessous.

C. Montrez votre implémentation de **Cipher** dans l'espace ci-dessous.

D. Montrez votre implémentation de **SubstitutionCipher** dans l'espace ci-dessous.

E. Montrez votre déclaration de **CaesarCipher** dans l'espace ci-dessous – aucune implémentation requise.

.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

.
.
.
.
.
.
.

Références

java.util.Random

- Une instance de la classe **java.util.Random** est utilisée pour générer une séquence ("stream") de nombres pseudo-aléatoires.
- La méthode d'instance **int nextInt(int n)** retourne un nombre pseudo-aléatoire, une valeur "int" distribuée uniformément entre 0 (inclue) et la valeur spécifiée (exclue), tirée de la séquence générée par ce générateur de nombres aléatoires.

String

- Le constructeur **String(char[] value)** crée une nouvelle **String** qui contient une copie de la séquence de caractères contenue dans le paramêtre de type tableau de caractère. (Mais cette copie est en format String plutôt qu'en format tableau de caractères).
- **char charAt(int index)** de la classe **String**, retourne la valeur char à l'index spécifié.
- **char [] toCharArray()** de la classe **String**, convertit cette chaine de caractères en un nouveau tableau de caractères.

Question 2 (15 points)

Pour l'implémentation partielle de la classe **IntArrayList** sur la page suivante, veuillez implémenter la méthode **removeDuplicates**.

- Après un appel à la méthode **removeDuplicates**, le tableau désigné par la variable d'instance **elems** ne contient aucune valeur dupliquée.
- La méthode retient la valeur la plus à gauche pour chaque pair dupliquée.
- La taille du tableau appelé **elems** doit être exactement celle du nombre de valeurs uniques.

L'exécution de la méthode **IntArrayList.test** imprimera le résultat suivant:

```
[2, 1, 1, 5, 3, 1, 2, 4]
[2, 1, 5, 3, 4]
```

(Voir la page suivante)

```
public class IntArrayList {

    private int[] elems;

    public IntArrayList(int[] elems) {
        // precondition: elems is not null
        this.elems = new int[elems.length];
        System.arraycopy(elems, 0, this.elems, 0, elems.length);
    }

    public void removeDuplicates() {









                                    }

    public String toString() {
        return java.util.Arrays.toString(elems);
    }

    public static void main(String[] args) {

        IntArrayList xs;
        xs = new IntArrayList(new int[]{2, 1, 1, 5, 3, 1, 2, 4});

        System.out.println(xs);
        xs.removeDuplicates();
        System.out.println(xs);

    }

}
```

Question 3 (15 points)

A. **Vrai** ou **Faux**. Le programme de Java ci-dessous imprime 2000.

```
public class Savings {

    private int value;

    public Savings(int value) {
        this.value = value;
    }

    public static int getValue() {
        return value;
    }

    public static void addValue(int amount) {
        value = value + amount;
    }

    public static void spendAll() {
        value = 0;
    }
}

public class Test {
    public static void main(String[] args) {
        Savings savings = new Savings(1000);
        savings.addValue(3000);
        savings.spendAll();
        savings.addValue(2000);
        System.out.println(savings.getValue());
    }
}
```

B. **Vrai** ou **Faux**. Si **hasFeathers()** est une méthode de la classe **Animal** retournant un booléen; et **d** est une référence à un objet de la classe **Dog**, sous-classe de la classe **Mammal**, elle-même sous-classe de la classe **Animal**; alors la commande **boolean t = d.hasFeathers();** est valide.

C. **Vrai** ou **Faux**. Le code de Java ci-dessous ne compilera pas.

```
public abstract class Replaceable {
    public abstract void replace(String part);
}

public class Worker implements Replaceable {

    private String name;

    public Worker(String name) {
        this.name = name;
    }

    public void replace(String name) {
        this.name = name;
    }
}
```

D. La méthode **Test.main** ci-dessous imprimera:

(a) animal

(b) mammal

(c) dog

(d) aucune de ces réponses

```
public class Animal {

    public String getKind() {
        return "animal";
    }
}

public class Mammal extends Animal {

    public String getKind() {
        return "mammal";
    }
}

public class Dog extends Mammal {

    public String getKind() {
        return "dog";
    }
}

public class Test {

    public static void main(String[] args) {
        Animal creature;
        creature = new Dog();
        System.out.println(creature.getKind());
    }
}
```

E. Laquelle(Lesquelles) des trois déclarations suivantes est(sont) correcte(s)?

(a) En Java, une classe donnée peut implémenter de nombreux interfaces.

(b) En Java, une classe donnée peut avoir de nombreuses sous-classes et de nombreuses super-classes.

(c) En Java, chaque sous-classe **Y** de la classe **X** doit contenir un appel explicite à **super()** dans son constructeur.

i. Seulement (a)

ii. (a) et (b)

iii. (a) et (c)

iv. Seulement (c)

v. (b) et (c)

F. Quel mot clé, ou paire de mots clés garantit qu'une méthode définie dans une super-classe 1) ne sera pas visible à partir d'autres paquets (packages), 2) ne sera visible que par ses sous-classes, et 3) ne peut pas être remplacée (overridden) dans ses sous-classes?

(a) private, static

(b) static

(c) final

(d) protected, static

(e) protected, final

G. Veuillez considérer l'implémentation partielle (incomplète) ci-dessous.

```
public interface Comparable {

}

public class Fraction implements Comparable {

}

public class Test {

    public static void main(String[] args) {
        Comparable c;
        Fraction f;
        c = new Comparable();
        f = new Fraction();
        c = f;
    }
}
```

Laquelle (Lesquelles) des allégations suivantes est (sont) vraie(s)?

(a) La variable **c** ne peut pas être déclarée comme appartenant à la classe **Comparable**.

(b) "**c = new Comparable();**" cause une erreur de compilation.

(c) La valeur de **f** ne peut pas être assignée à **c**.

i. Seulement (a)

ii. Seulement (b)

iii. Seulement (c)

iv. (a) et (b)

v. (b) et (c)

H. L'exécution du programme de Java **Test.main** ci-dessous:

(a) Produit une erreur de compilation

(b) Produit `$20.0USD is equals to $25.0CDN`

(c) Produit `$20.0USD is equals to $16.0CDN`

(d) Produit `USD@677327b6 is equals to CDN@14ae5a5`

(e) Produit une erreur d'exécution (run-time error)

```
public class Test {
    public static void main(String[] args) {
        Currency v;
        v = new USD(20.0);
        System.out.println(v + " is equals to " + v.toCDN());
    }
}
```

Les déclarations des classes **Currency**, **USD**, et **CDN** se trouvent sur la page suivante.

```
public abstract class Currency {

    public static final double EXCHANGE_RATE_USD_TO_CDN = 1.25;
    private final String title;
    private final double amount;

    public Currency(String title, double amount) {
        this.title = title;
        this.amount = amount;
    }

    public String getTitle() {
        return title;
    }

    public double getAmount() {
        return amount;
    }

    public abstract Currency toUSD();
    public abstract Currency toCDN();

    public String toString() {
        return "$"+getAmount()+getTitle();
    }
}

public class USD extends Currency {

    public USD(double amount) {
        super("USD", amount);
    }

    public Currency toUSD() {
        return new USD(getAmount());
    }

    public Currency toCDN() {
        return new CDN(EXCHANGE_RATE_USD_TO_CDN * getAmount());
    }
}

public class CDN extends Currency {

    public CDN(double amount) {
        super("CDN", amount);
    }

    public Currency toUSD() {
        return new USD(getAmount() / EXCHANGE_RATE_USD_TO_CDN);
    }

    public Currency toCDN() {
        return new CDN(getAmount());
    }
}
```

I. Suivez les instructions présentées en classe, ainsi que dans les notes de cours, pour dessiner le diagramme de mémoire pour tous les objets, variables locales et paramètres de la méthode **Test.main** suivant l'exécution de la commande "**z.insertAnimal(d);**". L'implémentation des classes **Zoo** et **Animal** se trouvent sur la page suivante.

```
public class Test {
    public static void main(String[] args) {

        Animal a, b, c, d;
        a = new Animal("zebra");
        b = new Animal("tiger");
        c = new Animal("elephant");
        d = new Animal("monkey");

        Zoo z;
        z = new Zoo(3);

        z.insertAnimal(a);
        z.insertAnimal(b);
        z.insertAnimal(c);
        z.insertAnimal(d);

    }
}
```

```
public class Animal {

    private String kind;

    public Animal(String kind) {
        this.kind = kind;
    }

    public String getKind() {
        return this.kind;
    }

}

public class Zoo {

    private static int count = 0;
    private int capacity;
    private Animal[] cages;

    public Zoo(int capacity) {
        this.capacity = capacity;
        cages = new Animal[capacity];
    }

    void insertAnimal(Animal animal) {
        if (count < capacity) {
            cages[count++] = animal;
        } else {
            System.err.println("Sorry, too many animals.");
        }

    }
}
```

(page de brouillon)