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 to Computing II (ITI 1121)
Midterm Examination: Solutions

Instructors: Sherif G. Aly, Nathalie Japkowicz, and Marcel Turcotte

February 2015, duration: 2 hours

Identification

Last name: _____ First name: _____

Student number: _____ Section (A or B or C): _____

Instructions

1. This is a closed book examination.
2. No calculators, electronic devices or other aids are permitted.
   (a) Any electronic device or tool must be shut off, stored and out of reach.
   (b) Anyone who fails to comply with these regulations may be charged with academic fraud.
3. Write your answers in the space provided.
   (a) Use the back of pages if necessary.
   (b) You may not hand in additional pages.
4. Write comments and assumptions to get partial marks.
5. Do not remove the staple holding the examination pages together.
6. Beware, poor hand writing can affect grades.
7. Wait for the start of the examination.

Marking scheme

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

Question 1 (35 marks)

Cryptography has been used for thousands of years to protect secrets. Information that needs protection could be put in a form that is more difficult to comprehend through a process called "encryption". The process can be reversed through a process called "decryption".

There are many very effective ways by which this can be achieved to protect the most sensitive secrets, however, two extremely simple ways that have been used since older ages involve:

Substitution : Where a text is replaced by another corresponding cipher text.

For example, if the alphabet of your text is:

`ABCDEFGHIJKLMNOPQRSTUVWXYZ`

And your cipher text is:

`ZEBRASCDFGHIJKLMNOPQTUVWXY`

The sentence "`COMMANDER`" will be encrypted as "`BLJJZKRAO`".
Consequently, "`BLJJZKRAO`" will be decrypted as "`COMMANDER`".

Notice that each character was replaced with the corresponding character in the cipher text.

Caesar : Where each character in your text is replaced by another character, a fixed number of positions (key) down the alphabet.

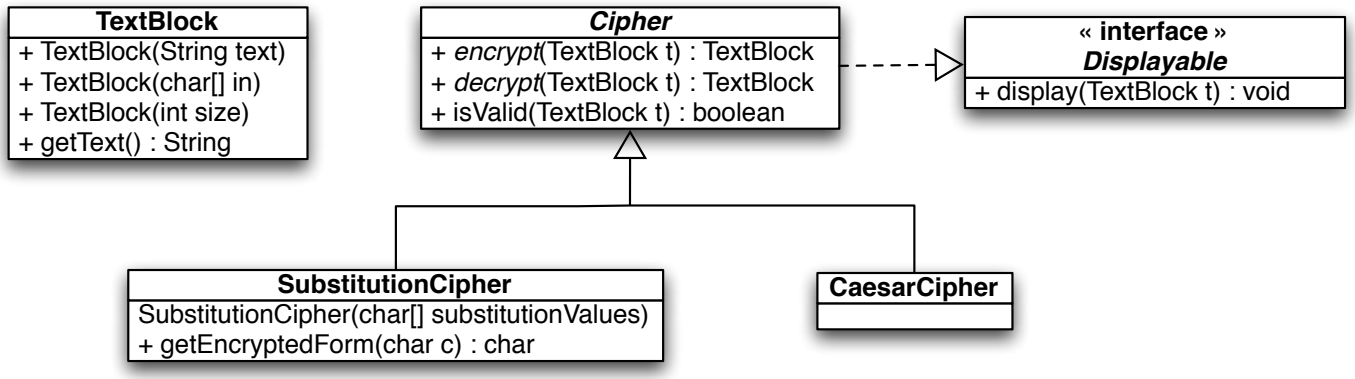
(Do not worry about the implementation details of the Caesar cipher since you will not be asked to implement it.)

On the next page, you will find the UML class diagram for this question, as well the description of each class.

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

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

c.display(t);
```

The execution of the above Java program prints "`BLJJZKRAO`" on the console.

Study the following class diagram, and follow the instructions below to implement the Java programs for this question. Only consider the "usual" alphabet composed of characters in [A-Z]. You can assume that all the input values will be valid.



TextBlock : Will contain the following methods:

- **TextBlock(String s)**: A constructor to initialize the object with the given string.
- **TextBlock(char[] in)**: A constructor to initialize the object with the given sequence of characters.
- **TextBlock(int size)**: A constructor that initializes the object to a random string of size characters in [A-Z].
- **String getText()**: That returns back the stored text.

Displayable : Will contain the following method:

- **void display(TextBlock t)**: Will be used to display the text in a TextBlock. Herein, simply use **System.out.println**.

Cipher : An abstract class that will contain, at the least, the following methods:

- **abstract TextBlock encrypt (TextBlock input)**
- **abstract TextBlock decrypt (TextBlock input)**
- **boolean isValid(TextBlock t)**: this concrete method returns **true** if encrypting a **TextBlock**, and then decrypting it, gives the same original **TextBlock** you started with, and **false** otherwise. In other words, there is no loss of information.

SubstitutionCipher : Inherits from **Cipher** and implements **Displayable**. Will contain, at the least, the following methods:

- **SubstitutionCipher(char [] substitutionValues)**: will initialize the object with the given substitution characters for the alphabet [A-Z].
- **TextBlock encrypt (TextBlock input)**
- **TextBlock decrypt (TextBlock input)**
- **char getEncryptedForm(char c)**: returns the encrypted form of a character **c** from the substitutionValues array.

CaesarCipher : Inherits from Cipher and implements Displayable. You are only required to declare this class, with no implementation.

A. Give your implementation of **TextBlock** in the box below.

```
// MODEL 1

import java.util.Random;

public class TextBlock {

    private static final Random generator = new Random();

    private static final char[] alpha = "ABCDEFGHIJKLMNOPQRSTUVWXYZ".toCharArray();

    private final String text;

    public TextBlock(String text) {
        this.text = text;
    }

    public TextBlock(char[] in) {
        this.text = new String(in);
    }

    public TextBlock(int size) {
        char[] chars = new char[size];
        for (int i = 0; i < size; i++) {
            chars[i] = nextChar();
        }
        this.text = new String(chars);
    }

    public String getText() {
        return text;
    }

    private static char nextChar() {
        return alpha[generator.nextInt(alpha.length)];
    }

}
```

```java
// MODEL 2

import java.util.Random;

public class TextBlock {

    private static final Random generator = new Random();

    private final String text;

    public TextBlock(String text) {
        this.text = text;
    }

    public TextBlock(char[] in) {
        this.text = new String(in);
    }

    public TextBlock(int size) {
        char[] chars = new char[size];
        for (int i = 0; i < size; i++) {
            chars[i] = nextChar();
        }
        this.text = new String(chars);
    }

    public String getText() {
        return text;
    }

    private static char nextChar() {
        return (char) ('A' + generator.nextInt(26));
    }

}
```

```java
// MODEL 3

import java.util.Random;

public class TextBlock {

    private static final Random generator = new Random();

    private static final char[] alpha = "ABCDEFGHIJKLMNOPQRSTUVWXYZ".toCharArray();

    private final String text;

    public TextBlock(String text) {
        this.text = text;
    }

    public TextBlock(char[] in) {
        this.text = new String(in);
    }

    public TextBlock(int size) {
        char[] chars = new char[size];
        for (int i = 0; i < size; i++) {
            chars[i] = nextChar();
        }
        this.text = new String(chars);
    }

    public String getText() {
        return text;
    }

    private static char nextChar() {
        return alpha[generator.nextInt(alpha.length)];
    }

    private static boolean equals(String a, String b) {
        if (a==null || b==null) {
            return a == b;
        } else {
            return a.equals(b);
        }
    }

    // If validate uses the equals method for TextBlock
    // objects, then TextBlock needs to override equals

    public boolean equals(Object other) {
        if (! (other instanceof TextBlock)) {
            return false;
        }
        return equals(getText(), ((TextBlock) other).getText());
    }
}
```

B. Give your implementation of **Displayable** in the box below.

```
// MODEL

public interface Displayable {
    public abstract void display(TextBlock text);
}
```

C. Give your implementation of **Cipher** in the box below.

```
// MODEL 1

public abstract class Cipher implements Displayable {

    public abstract TextBlock encrypt(TextBlock input);

    public abstract TextBlock decrypt(TextBlock input);

    public boolean validate(TextBlock input) {
        return decrypt(encrypt(input)).getText().equals(input.getText());
    }

    public void display(TextBlock text) {
        System.out.println(text.getText());
    }

}
```

```
// MODEL 2

public abstract class Cipher implements Displayable {

    public abstract TextBlock encrypt(TextBlock input);

    public abstract TextBlock decrypt(TextBlock input);

    public boolean validate(TextBlock input) { // If TextBlock overrides equals!
        return decrypt(encrypt(input)).equals(input);
    }

    public void display(TextBlock text) {
        System.out.println(text.getText());
    }

}
```

D. Give your implementation of **SubstitutionCipher** in the box below.

```
// MODEL 1

public class SubstitutionCipher extends Cipher {

    private static final char[] alpha = "ABCDEFGHIJKLMNOPQRSTUVWXYZ".toCharArray();
    private final char[] substitutionValues;

    public SubstitutionCipher(char[] vs) {
        this.substitutionValues = new char[vs.length];
        System.arraycopy(vs, 0, this.substitutionValues, 0, vs.length);
    }

    public TextBlock encrypt(TextBlock input) {
        String textIn = input.getText();
        char[] out = new char[textIn.length()];

        for (int i=0; i<textIn.length(); i++) {
            out[i] = getEncryptedForm(textIn.charAt(i));
        }
        return new TextBlock(out);
    }

    public TextBlock decrypt(TextBlock input) {
        String textIn = input.getText();
        char[] out = new char[textIn.length()];

        for (int i=0; i<textIn.length(); i++) {
            out[i] = getDecryptedForm(textIn.charAt(i));
        }
        return new TextBlock(out);
    }

    private static int indexOf(char[] in, char c) {
        int pos=in.length-1;
        while (pos>=0 && in[pos] != c) {
            pos--;
        }
        return pos; // will be -1 if c was not found
    }

    private char getDecryptedForm(char c) {
        // precondition: c is in the range 'A' to 'Z'

        int pos=indexOf(substitutionValues, c);
        return alpha[pos];
    }

    public char getEncryptedForm(char c) {
        // precondition: c is in the range 'A' to 'Z'

        int pos=indexOf(alpha, c);
        return substitutionValues[pos];
    }
}
```

```java
// MODEL 2

public class SubstitutionCipher extends Cipher {

    private final char[] substitutionValues;

    public SubstitutionCipher(char[] vs) {
        this.substitutionValues = new char[vs.length];
        System.arraycopy(vs, 0, this.substitutionValues, 0, vs.length);
    }

    public TextBlock encrypt(TextBlock input) {
        String textIn = input.getText();
        char[] out = new char[textIn.length()];

        for (int i=0; i<textIn.length(); i++) {
            out[i] = getEncryptedForm(textIn.charAt(i));
        }

        return new TextBlock(out);
    }

    public TextBlock decrypt(TextBlock input) {
        String textIn = input.getText();
        char[] out = new char[textIn.length()];

        for (int i=0; i<textIn.length(); i++) {
            out[i] = getDecryptedForm(textIn.charAt(i));
        }

        return new TextBlock(out);
    }

    private static int indexOf(char[] in, char c) {
        int pos=in.length-1;
        while (pos>=0 && in[pos] != c) {
            pos--;
        }
        return pos; // will be -1 if c was not found
    }

    private char getDecryptedForm(char c) {
        // precondition: c is in the range 'A' to 'Z'

        int pos=indexOf(substitutionValues, c);
        return (char) ('A'+pos);
    }

    public char getEncryptedForm(char c) {
        // precondition: c is in the range 'A' to 'Z'

        int pos = c - 'A';
        return substitutionValues[pos];
    }
}
```

E. Give the declaration of **CaesarCipher** in the box below – no implementation needed.

```
public class CaesarCipher extends Cipher {

}
```

Reference

java.util.Random

- An instance of the class **java.util.Random** is used to generate a stream of pseudorandom numbers.
- The instance method **int nextInt(int n)**, returns a pseudorandom number, uniformly distributed int value between 0 (inclusive) and the specified value (exclusive), drawn from this random number generator's sequence.

String

- The constructor **String(char[] value)** allocates a new **String** so that it represents the sequence of characters currently contained in the character array argument.
- **char charAt(int index)** of the class **String**, returns the char value at the specified index.
- **char [] toCharArray()** of the class **String**, converts this string to a new character array.

Question 2 (15 marks)

For the partial implementation of the class **IntArrayList** below, implement the method **removeDuplicates**.

- After a call to the method **removeDuplicates**, the array designated by the instance variable **elems** contains no duplicated values.
- The method keeps the leftmost value for each duplicated pairs.
- Finally, the size of the array designated by **elems** must be exactly that of the number of unique values.

The execution of the method **IntArrayList.test** prints the following:

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

```
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() {

        // MODEL

        int copy[], last;
        copy = new int[elems.length];

        last = 0;
        for (int i = 0; i < elems.length; i++) {
            boolean found = false;
            for (int j = 0; j < last && !found; j++) {
                if (elems[i] == copy[j]) {
                    found = true;
                }
            }
            if (!found) {
                copy[last++] = elems[i];
            }
        }

        elems = new int[last];

        System.arraycopy(copy, 0, this.elems, 0, last);

    }

    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 marks)

A. **True** or **False**. The Java program below prints 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. **True** or **False**. **hasFeathers()** is a method from the class **Animal** that returns a boolean. **d** is a reference to an object of the class **Dog**, which is a subclass of the class **Mammal**, itself a subclass of the class **Animal**. "**boolean t = d.hasFeathers();** is a valid statement.

C. **True** or **False**. The following Java code below will not compile.

```
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. The method **Test.main** below will print:

(a) animal

(b) mammal

(c) dog

(d) none of the above

```
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. Which of the following three statements is or are correct?

(a) In Java, a given class is allowed to implement many different interfaces.

(b) In Java, a given class is allowed to have many different subclasses, and many different direct superclasses.

(c) In Java, every subclass **Y** of class **X** must contain an explicit call to **super()** in its constructor.

i. Only (a)

ii. (a) and (b)

iii. (a) and (c)

iv. Only (c)

v. (b) and (c)

F. Which keyword, or pair of keywords, ensures that a method defined in a superclass 1) will not be visible from other packages, 2) will be visible only by its subclasses, and 3) cannot be overridden in its subclasses?

(a) private, static

(b) static

(c) final

(d) protected, static

(e) protected, final

G. Consider the partial (incomplete) implementation below.

```
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;
    }
}
```

Which of the following statements is true?

(a) Variable **c** cannot be declared as **Comparable**.

(b) "**c = new Comparable();**" causes a compile-error.

(c) The value of **f** cannot be assigned to **c**.

i. only (a)

ii. only (b)

iii. only (c)

iv. (a) and (b)

v. (b) and (c)

H. The execution of the Java program **Test.main** below:

(a) Produces a compile-time error

(b) Produces `$20.0USD is equal to $25.0CDN`

(c) Produces `$20.0USD is equal to $16.0CDN`

(d) Produces `USD@677327b6 is equal to CDN@14ae5a5`

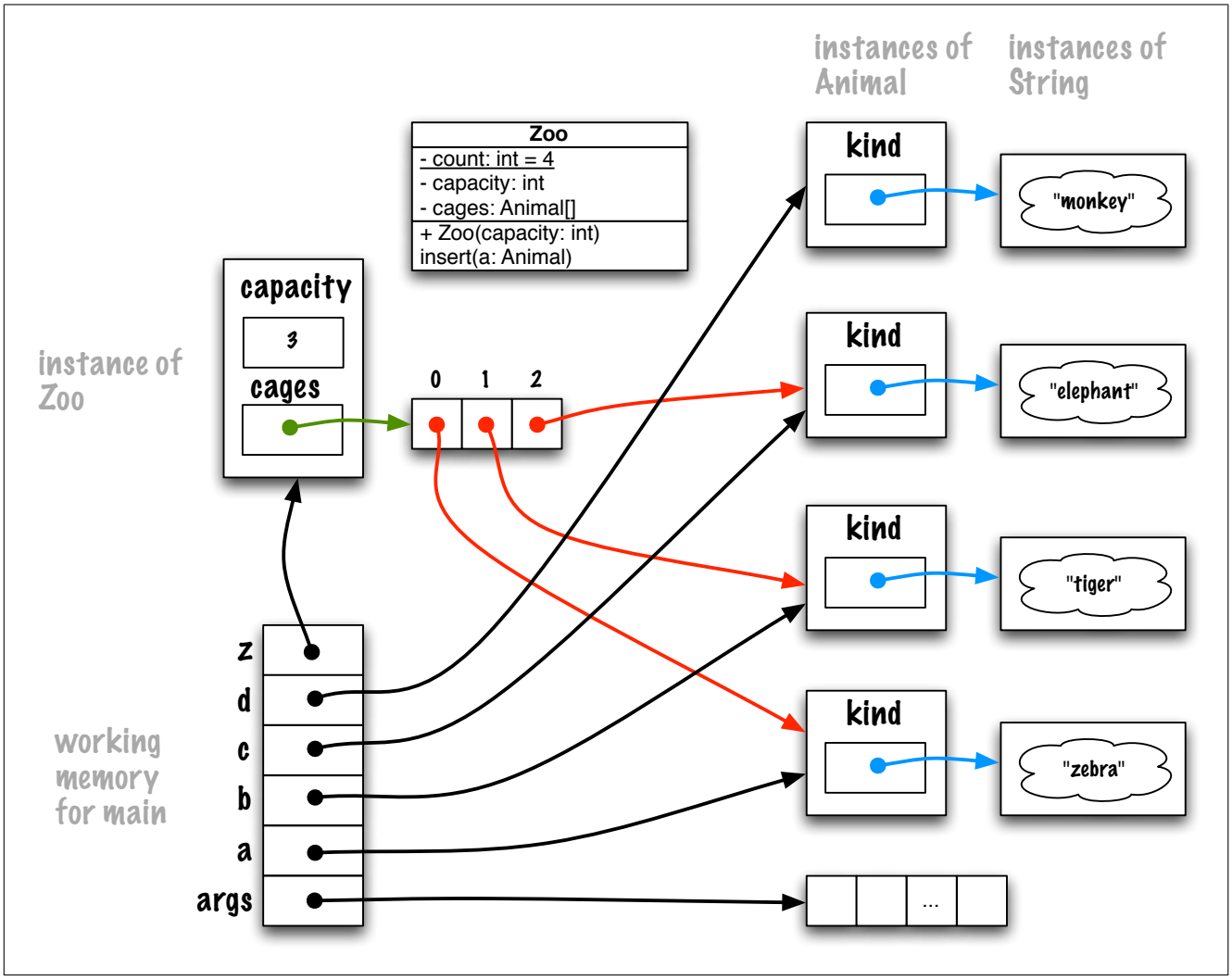
(e) Produces a run-time error

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

The declarations for the classes **Currency**, **USD**, and **CDN** can be found ont the next page.

```
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. Following the guidelines presented in class, as well as the lecture notes, draw the memory diagrams for all the objects, the local variables, and parameter of the method **Test.main** following the execution of the statement "**z.insertAnimal(d);**". The implementation of the classes **Zoo** and **Animal** can be found on the next page.



```
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.");
        }

    }
}
```

(blank space)