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 to Computing II (ITI 1121)
MIDTERM EXAMINATION

Instructors: Guy-Vincent Jourdan and Marcel Turcotte

March 2019, duration: 2 hours

Identification

Last name: _____ First name: _____

Student #: _____ Seat #: _____ Signature: _____ 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. Beware, poor hand-writing can affect grades.
6. Do not remove pages or the staple holding the examination pages together.
7. Wait for the start of the examination.

Marking scheme

Question	Maximum	Result
1	10	
2	5	
3	20	
4	15	
Total	**50**	

Directives

- For all the questions of this examination, with the exception of the classes **Math** and **System**, you cannot use the Java libraries. Specifically, do not use **Arrays** and **ArrayList**. There should be no import statements.

Question 1 (10 marks)

You must implement the class **Product** with the following characteristics.

- Has a class variable **taxRate** of type **double**. Its initial value is 0.13.
- Each **Product** has a **description** (of type **String**) and a **price** (of type **double**). Accordingly, the constructor has two parameters corresponding to these variables. Assume that the value of the parameter **price** is positive.
- A class method **setTaxRate**, which sets **taxRate** to a new value. Assume that the value of the parameter is between 0.0 and 1.0.
- An instance method **getPriceWithTax** returning the price of the product with the tax included.
- **Product** overrides the method **equals** from the class **Object**. Make sure that your method is as robust as possible.

Implement the class **Product** in the space provided on the next page.

Implement the class **Product** in the space below.

Question 2 (5 marks)

For this question, assume that you have been provided with valid implementations of the interfaces **Queue** and **Stack**. On the next page, we refer to these implementations as **QueueImplementation** and **StackImplementation**, respectively. You will find the interfaces **Queue** and **Stack** below.

```
public interface Queue<E> {

    /**
     * Returns true if the queue is currently empty.
     * @return true if the queue is empty
     */

    boolean isEmpty();

    /**
     * Adds the reference elem at the rear of the queue.
     * @param elem the reference of the new element
     */

    void enqueue(E elem);

    /**
     * Removes and returns the front element of the queue.
     * @return the reference of the removed element
     */

    E dequeue();

}
```

```
public interface Stack<E> {

    /**
     * Returns true if the stack is currently empty.
     * @return true if the stack is empty
     */

    boolean isEmpty();

    /**
     * Adds the reference elem onto the top of this stack.
     * @param elem the reference of the new element
     */

    void push(E elem);

    /**
     * Removes and returns the top element of the stack.
     * @return the reference of the removed element
     */

    E pop();

}
```

Carefully analyze the source code below and give the output that will be printed.

```
public class Test {

    public static void testQueue() {

        Queue<String> q;
        q = new QueueImplementation<String>();

        q.enqueue("");

        for (int i=0; i<7; i++) {
            String elem;
            elem = q.dequeue();
            System.out.println("["+elem+"]");
            q.enqueue(elem+"0");
            q.enqueue(elem+"1");
        }

    }

    public static void testStack() {

        Stack<String> s;
        s = new StackImplementation<String>();

        s.push("");

        for (int i=0; i<7; i++) {
            String elem;
            elem = s.pop();
            System.out.println("["+elem+"]");
            s.push(elem+"0");
            s.push(elem+"1");
        }

    }

    public static void main(String[] args) {
        System.out.println("Calling testQueue() ::");
        testQueue();
        System.out.println("Calling testStack() ::");
        testStack();
    }

}
```

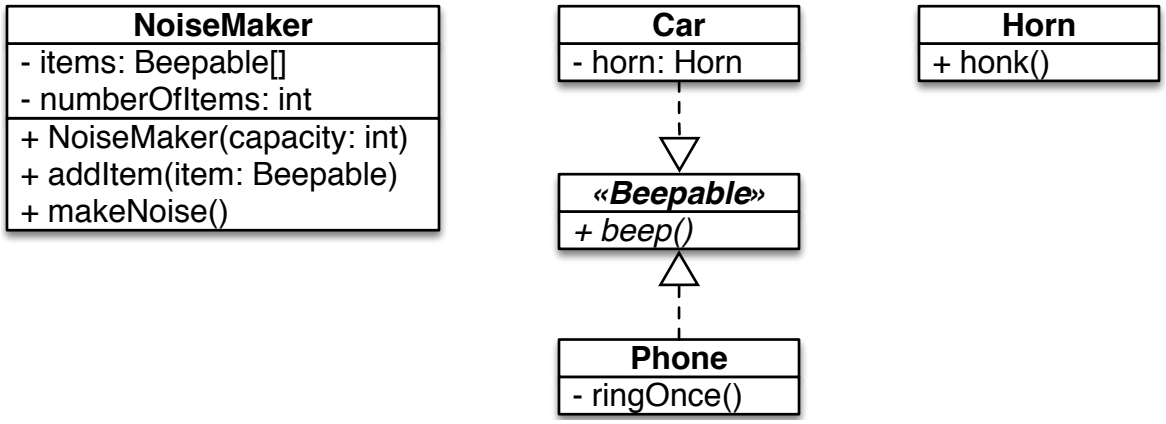
Give your answer in the space provided on the next page.

Give the output of the program **Test**.

```
> java Test
```

Question 3 (20 marks)

This question is about several classes all related to the interface **Beepable**. The UML diagram below shows their relationships and characteristics.



- All **Beepable** objects have a method **beep**.
- Objects of the classes **Car** and **Phone** can be seen as **Beepable**.
- A **Phone** can beep by calling its method **ringOnce()**, which simply prints "ring!".
- A **Car** can beep by calling the method **honk()** of its **horn**, which simply prints "honk!".
- A **NoiseMaker** stores a maximum of **n Beepable** objects, where the value of **n** is passed as a parameter to its constructor. Assume that the value of **n** will always be positive.
- The method **addItem** can be used to add a **Beepable** object to **NoiseMaker**. It displays a message "This NoiseMaker is full" if the array is full and ignores that item. Furthermore, it displays the message, "null is not a valid value" and ignores that item, if the value of the parameter is **null**.
- When the method **makeNoise** is called, **NoiseMaker** must ask all the **Beepable** objects to beep.

In particular, executing the following statements:

```
NoiseMaker m;
m = new NoiseMaker(5);

m.addItem(new Phone());
m.addItem(new Car());
m.addItem(new Car());
m.addItem(new Phone());
m.addItem(new Phone());
m.addItem(new Car());

m.makeNoise();
```

produces the following output:

```
This NoiseMaker is full
ring!
honk!
honk!
ring!
ring!
```

A. Implement the interface **Beepable**.

B. Implement the class **Car**.

Here is the class **Horn**.

```
public class Horn {

    public void honk() {
        System.out.println("honk!");
    }

}
```

C. Implement the class **Phone**.

D. Implement the class **NoiseMaker**.

Question 4 (15 marks)

For this question, you must provide an implementation of a class to represent a polynomial. A polynomial is a formula of the form $f(t) = 2.0 + 4.0 \times t^2 - t^3$. This particular polynomial is of degree 3 and its coefficients are 2.0, 0.0, 4.0, and -1.0.

- Specifically, you must store the coefficients of the polynomial into an array.
- There are two constructors. One of them receives the reference of an array that contains the coefficients to be used to initialize this polynomial. Assume this reference is not **null**. The second constructor receives the degree of the polynomial only. Assume this degree is positive. With the second constructor, all the coefficients are initially zero.
- There is a method **set(int index, double value)** that changes the value of the coefficient at the specified index of the polynomial. Assume that the value of **index** is valid for this polynomial.
- The method **get(int index)** returns the coefficient at the specified index. Assume that the value of **index** is valid for this polynomial.
- The method **eval(double t)** calculates the value of the polynomial for the value **t**. Evaluating the above polynomial for the value 2.0 returns the value 10.0, which is $2.0 + 4.0 \times 2.0^2 - 2.0^3$. Hint: you can use **Math.pow(base,exponent)** to help you with this calculation.
- The method **toString** returns a **String** representation of this polynomial with the format presented in the example on the next page.

Make sure that running the program below using your implementation produces the expected output.

```
public class TestPolynomial {

    public static void main(String[] args) {

        Polynomial f,g,h;

        double[] coefficients;
        coefficients = new double[]{2.0, 0.0, 4.0, -1.0};

        f = new Polynomial(coefficients);

        coefficients[1] = 3.0;

        g = new Polynomial(coefficients);

        h = new Polynomial(12);

        h.set(0,  7.0);
        h.set(2,  4.0);
        h.set(6, -2.0);
        h.set(12,  5.0);

        System.out.println(f);
        System.out.println(g);
        System.out.println(h);

        System.out.println(f.eval(2));

    }

}
```

Expected output:

```
2.0 + 4.0 * t^2 + (-1.0) * t^3
2.0 + 3.0 * t^1 + 4.0 * t^2 + (-1.0) * t^3
7.0 + 4.0 * t^2 + (-2.0) * t^6 + 5.0 * t^12
10.0
```

```
public class Polynomial {

   // Instance variable(s)




    // Constructors
    public Polynomial(double[] coefficients) {









    }
    public Polynomial(int degree) {









    }
    // Setter
    public void set(int index, double value) {


    }
    // Getter
    public double get(int index) {


    }
```

```
    // Instance methods

    public double eval(double t) {









    }

    public String toString() {

```