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

Instructor: Marcel Turcotte

February 2011, duration: 2 hours

Identification

Student name: ____

Student number: ____________________ Signature: ______________________________

Instructions

1. This is a closed book examination;
2. No calculators or other aids are permitted;
3. Write comments and assumptions to get partial marks;
4. Beware, poor hand writing can affect grades;
5. **Do not remove the staple holding the examination pages together**
6. Write your answers in the space provided. Use the back of pages if necessary. You may **not** hand in additional pages.

Marking scheme

Question	Maximum	Result
1	15	
2	32	
3	23	
Total	**70**	

Question 1: (15 marks)

For this question, n items are stored into k boxes. Given the weight of the items (**weights**) and their box assignment (**map**), in Java, write a static method that calculates the weight difference between the heaviest and the lightest box.

- There are n items, numbered $0, 1 \ldots n-1$.
- There are k boxes, numbered $0, 1 \ldots k-1$.
- The formal parameter **weights** designates an array of size n of integers where **weights**[**i**] is the weight of the **i**th item. Hence, **weights**[**0**] is the weight of item 0, **weights**[**1**] is the weight of item 1, etc.
- The formal parameter **map** designates an array of size n of integers such that **map**[**i**] is the number of the box where item **i** is stored. Hence, **map**[**0**] indicates the box number of item 0, **map**[**1**] indicates the box number of item 1, etc.

```
public static int getWeightDifference( int[] weights, int[] map, int k ) {











}
```

If the above method is declared in the class **Q1**, the program below would display 70.

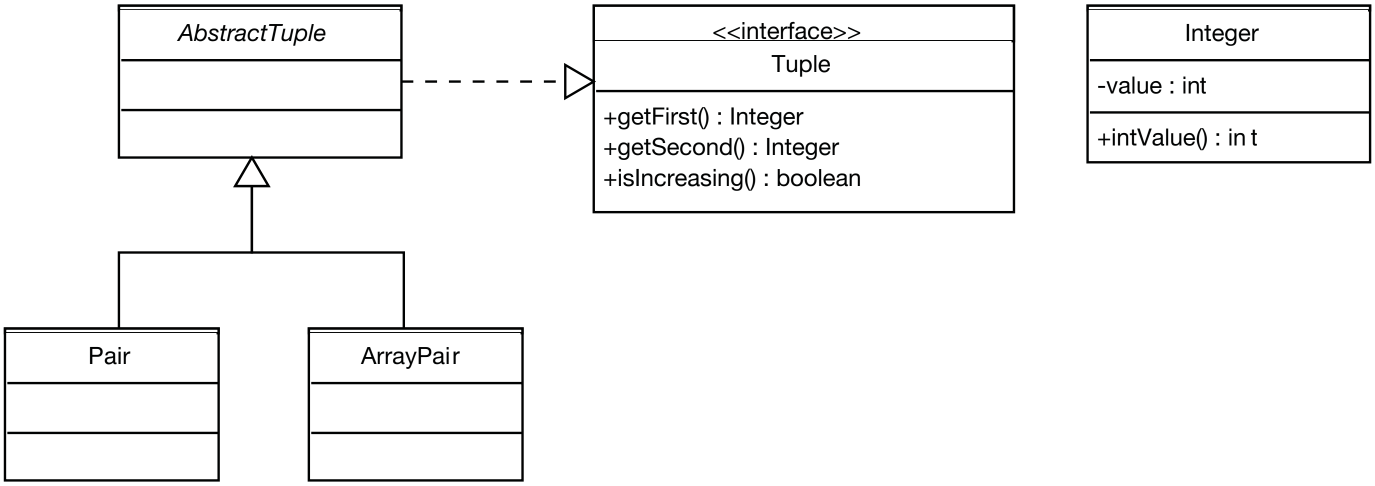
```
int[] weights = { 10, 50, 30, 10, 40, 20, 10, 10 };
int[] map     = {  2,  0,  1,  0,  0,  1,  2,  2 };
System.out.println( Q1.getWeightDifference( weights, map, 3 ) );
```

You can assume that **weights** and **map** are not **null**, and the content of both arrays is valid.

(blank space)

Question 2: (32 marks)

A tuple holds two **Integer** numbers (objects of the class **Integer**). All the tuples have a method **getFirst**, as well as a method **getSecond**, returning a reference to the first, and second, number of the tuple, respectively. A tuple has a method **isIncreasing** that returns **true** if the first element is smaller than the second element, and **false** otherwise.



For this question there is an interface named **Tuple**, an abstract class named **AbstractTuple**, and two concrete implementations, called **Pair** and **ArrayPair**. Their complete description can be found on the next pages. An object of the class **Integer** has a method **intValue** that returns the value of the object as an **int**. The execution of the statements below produces the following output: `7 is less than 17`.

```
Integer n1, n2, n3;

n1 = new Integer( 7 );
n2 = new Integer( 17 );
n3 = new Integer( 2 );

Tuple t1, t2;

t1 = new Pair( n1, n2 );
t2 = new ArrayPair( n2, n3 );

if ( t1.isIncreasing() ) {
    System.out.println( t1.getFirst() + " is less than " + t1.getSecond() );
}

if ( t2.isIncreasing() ) {
    System.out.println( t2.getFirst() + " is less than " + t2.getSecond() );
}
```

Make sure to include the constructors and access methods that are necessary for the execution of the above statements.

A. Implement the interface **Tuple**. The interface declares 3 methods. There are two access methods, named **getFirst** and **getSecond**, and both return a reference of type **Integer**. Finally, the interface also declares a method called **isIncreasing** that returns a **boolean** value. (6 marks)

B. Write the abstract class named **AbstractTuple**, which implements the interface **Tuple**. The class **AbstractTuple** has a concrete implementation of the method **isIncreasing**, which returns **true** if the first element of the tuple is less than the second element, and **false** otherwise. (8 marks)

C. Write a concrete implementation of the class **AbstractTuple** called **Pair**. The implementation has two instance variables that are references to the first and second element of this tuple. Add all the necessary constructors and access methods. (8 marks)

D. Write a concrete implementation of the class **AbstractTuple** called **ArrayPair**. The implementation uses an array of size 2 to store references to the first and second element of this tuple. Add all the necessary constructors and access methods. (10 marks)

Question 3: (23 marks)

A. Following the guidelines presented in class, as well as the lecture notes, draw the memory diagrams for all the objects and all the local variables of the method **Q3.test** following the execution of the statement “**line = new Line(origin, new Point(11, 21)**)”.

```
public class Point {
    private int x = 0;
    private int y = 0;
    public Point( int x, int y ) {
        this.x = x;
        y = y;
    }
}

public class Line {
    private Point a;
    private Point b;
    public Line( Point a, Point b ) {
        this.a = a;
        this.b = b;
    }
}

public class Q3 {
    public static void test() {
        int zero;
        Point origin;
        Line line;
        zero = 0;
        origin = new Point( zero, 0 );
        line = new Line( origin, new Point( 11, 21 ) );
        // Here!
    }
}
```

Answer:

B. The class **DynamicArrayStack** implements the interface **Stack** using the dynamic array technique presented in class. This allows a data structure (here a stack) to increase its physical size according to the needs of the application.

For the partial implementation of the class **DynamicArrayStack** below, implement the method void **trimToSize()** that reduces the physical size of the stack to the maximum of minCapacity and its logical size.

```java
public class DynamicArrayStack<E> implements Stack<E> {

        // The minimum physical size of the stack

        private final int minCapacity;

        // A default value for the increment that is used to increase
        // the capacity of the stack

        private static final int DEFAULT_INCREMENT = 25;

        // Each time the stack becomes full, a larger array is created,
        // increment is the number of cells to add to the size of the
        // previous array in order to obtain the size of the new array.

        private final int increment;

        // A reference to an array that holds the elements of the stack

        private E[] elems;

        // An instance variable that keeps track of the position of the
        // top element of the stack within the array. For an empty
        // stack the value is -1.

        private int top = -1;

        // Implement the method void trimToSize() that reduces the
        // physical size of the stack to the maximum of minCapacity and its
        // logical size.

        public void trimToSize() {






        }
}
```

C. Give the result that will be printed on the standard output when the following **main** method is executed.

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

    Stack<Integer> s, t;

    s = new DynamicArrayStack<Integer>( 100 );

    for (int i=1; i<5; i++) {
        s.push( new Integer( i ) );
    }

    Integer x = null;

    if ( ! s.isEmpty() ) {
        x = s.pop();
    }

    t = new DynamicArrayStack<Integer>( 100 );

    while ( ! s.isEmpty() ) {
        t.push( s.pop() );
    }

    t.push( x );

    while ( ! t.isEmpty() ) {
        System.out.print( t.pop() );
        if ( ! t.isEmpty() ) {
        System.out.print( "," );
        }
    }
    System.out.println();

}
```

Answer:

D. If a subclass's constructor does not make an explicit call to a superclass's constructor:

(a) a run-time error will result

(b) a compile-time error will result

(c) the constructor will be called anyway

(d) the class will be implicitly declared as abstract

(e) none of the above

Answer:

E. All Java classes are subclasses of the ________________ class.

(a) String

(b) java.lang

(c) Java

(d) Class

(e) Object

Answer:

F. Consider the following line of code.

```
Comparable s = new String();
```

Which of the following statements is true about this line?

(a) It will result in a compile-time error.

(b) It will result in a run-time error.

(c) It will create a String object pointed to by a Comparable reference.

(d) Although it is perfectly valid Java, it should be avoided due to confusion.

(e) none of the above are true

Answer:

G. Give the result that will be printed on the standard output when the following **main** method is executed.

```
public class Ticket {
    private int nextSerialNumber = 100;
    private int serialNumber;
    public Ticket() {
        serialNumber = nextSerialNumber;
        nextSerialNumber = nextSerialNumber + 1;
    }
    public int getSerialNumber() {
        return serialNumber;
    }
    public static void main( String[] args ) {
        Ticket t1, t2, t3;
        t1 = new Ticket();
        t2 = new Ticket();
        t3 = new Ticket();
        System.out.print( t1.getSerialNumber() + "," );
        System.out.print( t2.getSerialNumber() + "," );
        System.out.println( t3.getSerialNumber() );
    }
}
```

```
Answer:
```

H. Give the result that will be printed on the standard output when the following **main** method is executed.

```
public class Ticket {
    private static int nextSerialNumber = 100;
    private static int serialNumber;
    public Ticket() {
        serialNumber = nextSerialNumber;
        nextSerialNumber = nextSerialNumber + 1;
    }
    public int getSerialNumber() {
        return serialNumber;
    }
    public static void main( String[] args ) {
        Ticket t1, t2, t3;
        t1 = new Ticket();
        t2 = new Ticket();
        t3 = new Ticket();
        System.out.print( t1.getSerialNumber() + "," );
        System.out.print( t2.getSerialNumber() + "," );
        System.out.println( t3.getSerialNumber() );
    }
}
```

```
Answer:
```

I. Give the result that will be printed on the standard output when the following **main** method is executed.

```
public class Ticket {
    private static int nextSerialNumber = 100;
    private int serialNumber;
    public Ticket() {
        serialNumber = nextSerialNumber;
        nextSerialNumber = nextSerialNumber + 1;
    }
    public int getSerialNumber() {
        return serialNumber;
    }
    public static void main( String[] args ) {
        Ticket t1, t2, t3;
        t1 = new Ticket();
        t2 = new Ticket();
        t3 = new Ticket();
        System.out.print( t1.getSerialNumber() + "," );
        System.out.print( t2.getSerialNumber() + "," );
        System.out.println( t3.getSerialNumber() );
    }
}
```

```
Answer:
```

(blank space)