

PLEASE HAND IN

UNIVERSITY OF TORONTO
Faculty of Arts and Science
DECEMBER EXAMINATIONS 2000
CSC 108H1 F
St. George Campus
Duration — 3 hours

PLEASE HAND IN

Examination Aids: *The Java API: An Introduction for Students*, July 2000 edition.

Student Number: _____

Last Name: _____

First Name: _____

	L0101	L0201	L0202	L0203
	(J. Clarke)	(G. Baumgartner)	(F. Pitt)	(A. Hunter)
Lecture Section:				
(circle one)	L0301	L0302	L5101	L5102
	(T. Fairgrieve)	(J. Clarke)	(F. Pitt)	(T. Fairgrieve)

Do not turn this page until you have received the signal to start.
(In the meantime, please fill out the identification section above,
and read the instructions below *carefully.*)

This final examination consists of 5 questions on 16 pages (including this one), printed on both sides of the paper. *When you receive the signal to start, please make sure that your copy of the examination is complete.* Answer each question directly on the examination paper, in the space provided.

Be aware that concise, well thought-out answers will be rewarded over long rambling ones. Also, unreadable answers will be given zero (0) so write legibly. In your answers, you may use any class or method mentioned in *The Java API*, unless otherwise indicated for a particular question. You may *not* use classes or methods that are not specifically mentioned in *The Java API*.

General Hint: We were careful to leave ample space on the examination paper to answer each question, so if you find yourself using much more room than what is available, you're probably missing something. Also, remember that hints are just hints: you are not required to follow them if you can think of a different solution.

1: _____/ 60
 # 2: _____/ 15
 # 3: _____/ 25
 # 4: _____/ 15
 # 5: _____/ 20
 TOTAL: _____/135

Good Luck!

Question 1. [60 MARKS]**Part (a)** [4 MARKS]

Complete the identification section at the top of page 1 *including your lecture section*, then write your student number **legibly** at the bottom of the back of every page of this examination (where indicated). (HINT: Also take the time to just *read* every question so you can start working on what you know.)

Part (b) [5 MARKS]

Put a checkmark next to *every* statement below that will compile *without* warning or error.

- | | |
|--|--|
| <input type="checkbox"/> double f = 1; | <input type="checkbox"/> boolean b = null; |
| <input type="checkbox"/> char c = "a"; | <input type="checkbox"/> int j = 10; |
| <input type="checkbox"/> int i = -17; | |

Part (c) [8 MARKS]

Which of the following loops prints out 012345? *Check ALL that apply.*

- | | |
|---|--|
| <input type="checkbox"/> for (int i = 0; i < 5; i++) { System.out.print(i); } | |
| <input type="checkbox"/> for (int i = 0; i <= 5; i++) { System.out.print(i); } | |
| <input type="checkbox"/> for (int i = -1; i < 6; i++) { System.out.print(i); } | |
| <input type="checkbox"/> for (int i = 5; i >= 0; i--) { System.out.print(5 - i); } | |
| <input type="checkbox"/> int i = 0;
while (i < 5) {
System.out.print(i);
i = i + 1;
} | <input type="checkbox"/> int i = -1;
while (i < 5) {
i = i + 1;
System.out.print(i);
} |
| <input type="checkbox"/> int i = 0;
while (i < 5) {
i = i + 1;
System.out.print(i);
} | <input type="checkbox"/> int i = 0;
while (i < 6) {
System.out.print(i);
i = i + 1;
} |

Part (d) [2 MARKS]

If you run the code below, what gets printed out?

```
String s = new String("HeyThere!!Thisisalongstring.");
System.out.println(s.substring(3,7));
```

Check only ONE.

- | | |
|----------------------------------|----------------------------------|
| <input type="checkbox"/> yTh | <input type="checkbox"/> Ther |
| <input type="checkbox"/> HeyT | <input type="checkbox"/> yThere! |
| <input type="checkbox"/> There!! | |

Question 1. (CONTINUED)**Part (e)** [2 MARKS]

What will be the result when you attempt to compile and run the following code?

```
class A {
    public int i;
}

public class T {
    public static void main(String[] args) {
        A a = new A();
        a.i = 10;
        doIt(a);
        System.out.println(a.i);
    }
    public static void doIt(A z) {
        z.i = 3;
    }
} // class T
```

Check only ONE.

- | | |
|--|--|
| <input type="checkbox"/> Compilation and output "10" | <input type="checkbox"/> Compilation and output "3" |
| <input type="checkbox"/> Compilation and output "0" | <input type="checkbox"/> Error during compilation or execution |

Part (f) [2 MARKS]

Given the following code, what will be the output?

```
public class T {
    private static int j = 20;
    public static void main(String[] args) {
        int i = 10;
        T t = new T();
        t.doIt(i);
        System.out.println(i);
        System.out.println(j);
    }
    public void doIt(int x) {
        x *= 2;
        j *= 2;
    }
} // class T
```

Check only ONE.

- | | |
|---|-----------------------------|
| <input type="checkbox"/> Error: Incompatible type for method doIt(int). | <input type="checkbox"/> 10 |
| <input type="checkbox"/> 20 | <input type="checkbox"/> 40 |
| <input type="checkbox"/> 40 | <input type="checkbox"/> 10 |
| | <input type="checkbox"/> 20 |

Question 1. (CONTINUED)**Part (g)** [2 MARKS]

What will be the result when you attempt to compile and run the following code?

```
class Pair {
    private int x, y;
    public Pair(int a, int b) { x = a; y = b; }
    public String print() { return (x + "," + y); }
}
public class T {
    public static void main(String[] args) {
        Pair p1 = new Pair(2,3);
        System.out.print("p1=" + p1);
    }
}
```

Check the best *ONE* answer.

- | | |
|--|--|
| <input type="checkbox"/> Compilation and output "p1=2,3" | <input type="checkbox"/> Compilation and output "" |
| <input type="checkbox"/> Compilation and output "p1=Pair@3d4722" | <input type="checkbox"/> Compile-time error |

Part (h) [5 MARKS]

Consider the following code.

```
public class EqTest {
    private int value;
    public static void main(String[] args) {
        EqTest e = new EqTest(8);
        EqTest f = new EqTest(8);
        if /* place test here */ {
            System.out.println("Equal");
        } else {
            System.out.println("Not equal");
        }
    }
    public EqTest(int value) {
        this.value = value;
    }
    public boolean equals(EqTest other) {
        return (this.value == other.value);
    }
} // class EqTest
```

What test(s) could you put in place of the comment "/* place test here */" to result in an output of "Equal"? *Check ALL that apply.*

- | | |
|--|---|
| <input type="checkbox"/> (e.equals(f)) | <input type="checkbox"/> (equals(e,f)) |
| <input type="checkbox"/> (e == f) | <input type="checkbox"/> (e.value == f.value) |
| <input type="checkbox"/> (f.equals(e)) | |

Question 1. (CONTINUED)**Part (i)** [2 MARKS]

What will be the result when you attempt to compile and run the following code?

```
public class Conv {
    public static void main(String[] args) {
        Conv c = new Conv();
        String s = new String("ello");
        c.aMethod(s);
        System.out.println(s);
    }
    public void aMethod(String s) {
        char c = 'H';
        s += c;
    }
} // class Conv
```

Check only ONE.

- | | |
|---|---|
| <input type="checkbox"/> Compilation and output "Hello" | <input type="checkbox"/> Compilation and output "elloH" |
| <input type="checkbox"/> Compilation and output "ello" | <input type="checkbox"/> Compile-time error |

Part (j) [2 MARKS]

Given the following code, what will be output when the main method from class T is executed?

```
class A {
    public void doIt() { System.out.print("A"); }
}
class B extends A {
    public void doIt() { System.out.print("B"); }
}
class C extends B {
    public void doIt() { System.out.print("C"); }
}
public class T {
    public static void main(String[] args) {
        B b = new B();
        A a = (A) b;
        b.doIt();
        a.doIt();
    }
}
```

Check only ONE.

- | | |
|-----------------------------|---|
| <input type="checkbox"/> AA | <input type="checkbox"/> CC |
| <input type="checkbox"/> BB | <input type="checkbox"/> Exception in thread "main" |
| <input type="checkbox"/> BA | java.lang.ClassCastException: |

Question 1. (CONTINUED)**Part (k)** [2 MARKS]

Which comment best describes variable `m` declared as `int[] m;`? *Check only ONE.*

- `//m` is an array of ints

 `//m` is an array of 0 ints
 `//m` is a reference to an array of ints

 `//m` is a reference to ints

Part (l) [5 MARKS]

In the code below, class `Q` does *not* compile. Cross out all the statements that cause compiler errors.

```

class A {
    public void m1() { System.out.println("a"); }
}
class B extends A {
    public void m1() { System.out.println("b"); }
    public void m2() { System.out.println("B"); }
}
class C extends A {
    public void m1() { System.out.println("c"); }
    public void m2() { System.out.println("C"); }
}
public class Q {
    public static void main(String[] args) {
        A a = new A();
        B b = (B) a;
        a = new B();
        a.m1();
        a.m2();
        C c = new C();
        c.m2();
        B d = (B) c;
        c = (C) (new A());
        a = (A) (new C());
    }
} // class Q
  
```

Part (m) [2 MARKS]

In the code below, how many times does the test `(x < p)` get executed during a single call to `aMethod`?

```

void aMethod(int p) {
    int x = 0;
    while (x < p) {
        x = x + 1;
    }
}
  
```

Check only ONE.

- `x` times

 `(x - p) + 1` times if `x >= 0`
 `p + 1` times if `p >= 0`

 `p + 2` times
 `p` times

Question 1. (CONTINUED)

Part (n) [2 MARKS]

Given the following code, what will be output when the main method from class T is executed?

```
public class T {
    public static void main(String[] args) {
        T t = new T();
        int i = 7;
        t.doIt(i, "Got here");
    }
    public void doIt(double d, String s) { System.out.print(s + " " + d); }
    public void doIt(String s, int i) { System.out.print(i + " " + s); }
    public void doIt(int i) { System.out.print("i is " + i); }
    public void doIt(String s) { System.out.print(s); }
} // class T
```

Check only ONE.

- | | |
|---------------------------------------|-------------------------------------|
| <input type="checkbox"/> Got here 7.0 | <input type="checkbox"/> Got here |
| <input type="checkbox"/> i is 7 | <input type="checkbox"/> 7 Got here |

Part (o) [5 MARKS]

Write the output of the following program in the “box” to the right of the code.

```
public class Q {
    public static void main(String[] args) {
        int[] a = new int[2];
        a[0] = 0;
        m(a);
        System.out.println(a[0]);
    }
    public static void m(int[] a) {
        a[0] = 1;
        int[] b = a;
        b[0] = 2;
        System.out.println(a[0]);
        int[] c = new int[2];
        c[0] = 3;
        b = c;
        System.out.println(a[0]);
        c[0] = 4;
        System.out.println(a[0]);
        int[] d = new int[2];
        d[0] = 5;
        a = d;
        d[0] = 6;
        a = new int[2];
        a[0] = 7;
    }
} // class Q
```

Question 1. (CONTINUED)**Part (p)** [10 MARKS]

In the code below, fill in each blank (indicated by “_____”) with the most appropriate word *or words* from the following list (you may use each word more than once): void, char, int, double, public, private, static, final, Trouble, Difficulty, String, Integer, BufferedReader, IOException.

```

class Holder {
    public static void main(String[] args) {
        Trouble tr = new Trouble();
        tr.setBobble("1234".trim(), "1234".length());
        Difficulty d3 = new Difficulty("blech");
        System.out.println(Trouble.doGood(d3));
        tr.keepWorking(tr.shine("carefully")).doNothing();
    }
} // class Holder

class Difficulty { /* body not shown */ }

class Trouble {

    public void setBobble(_____ foo, _____ bar) {

        _____ bug = Integer.parseInt(foo);

        slug = shine();

    }

    public _____ doGood(_____ deed) {

        Trouble t2 = deed.makeTrouble();

        _____ worms = t2.shine("Eat 'em fried.");

        return t2;

    }

    public _____ shine() { /* body not shown */ }

    public String shine(String s) { /* body not shown */ }

    public _____ keepWorking(String s) { /* body not shown */ }

    public _____ doNothing() { /* body not shown */ }

    private int slug;

} // class Trouble

```

Question 2. [15 MARKS]

In the space below, write a static method `longestRun` that has one parameter of type `BufferedReader`. This method should read lines of input from the given `BufferedReader` until it reaches an empty line (*i.e.*, a line containing *no* character), and then return the length of the longest sequence of *consecutive* repeated lines. For example, if the input is

```
hello
hello
bye
bye
bye
hello
hello
```

(with an empty line at the end), then `longestRun` should return 3.

(HINT: This can be solved *without* using Arrays or Vectors. You may want to do this question last. If the input consists of a single empty line, make sure you return a reasonable value.)

[Use page 16 if you need more space...]

Question 3. [25 MARKS]

Suppose we already have classes that define things you can buy (**Purchases**) and human beings (**Persons**), part of which are given below. Assume that neither class **Purchase** nor class **Person** overrides the `toString()` method.

```
class Purchase {
    private double price; // The price is in dollars.
    public Purchase(double price) { ... }
    public double getPrice() { ... }
}
class Person {
    private String name;
    public Person(String name) { ... }
    public String getName() { ... }
}
```

In this question, you must define two new classes.

Part (a) [10 MARKS]

A **LumpOfCoal** is a subclass of **Purchase**. In addition to a price, it also has a weight in kilograms (which may have a fractional part). The **LumpOfCoal** class also has these methods:

- a constructor, taking the price and the weight as parameters (the price is not necessarily related to the weight);
- `toString()`—the usual method (it should return "a lump of coal" concatenated with the weight and price of the **LumpOfCoal**).

In the space below, write Java code to define the class **LumpOfCoal**.

Question 3. (CONTINUED)**Part (b)** [10 MARKS]

The other class you have to write is **Customer**, a subclass of **Person**. A **Customer** is a **Person** with an amount of money, and with a list of purchases that he/she has bought. The **Customer** class also needs these methods:

- a constructor, taking the name and the initial amount of money as parameters;
- **buy**, taking a **Purchase** as parameter. The customer will buy the purchase if the customer has enough money to do so; otherwise, the customer does nothing. Buying the purchase involves adding it to the customer's list of purchases, and reducing the customer's amount of money by the price of the purchase.

In the space below, write Java code to define the class **Customer**.

(HINT: Store the amount of money as a **double** and use a **Vector** to store the list of purchases.)

Question 3. (CONTINUED)**Part (c)** [5 MARKS]

Write a fragment of code (for a `main()` method, possibly) that creates a `Customer` and two `LumpOfCoals` and that causes the customer to purchase the lumps of coal. The customer's name is "Konrad Adenauer" and he starts out with \$25.00; the lumps of coal have weights of 1.3kg and 2.5kg and prices of \$12.37 and \$8.45, respectively.

Question 4. [15 MARKS]

The "Writer's Workbench" is a set of programs used to analyze documents. These documents are stored as lines of text in a file. One useful program counts the number of times a given word is used in a document. Suppose that you have been provided with a method with prototype

```
public static int occurrencesOfInString(String word, String line)
```

that counts the number of times `word` occurs as a *separate* word in `line`, ignoring capitalization. For example, if the `String lineOfText` is declared and initialized using

```
String lineOfText = "The rain in Spain falls mainly on the plain.";
```

then the method call `occurrencesOfInString("the", lineOfText)` returns 2, while the method call `occurrencesOfInString("in", lineOfText)` returns 1. (In the first case, the initial "The" was counted, and in the second case, the sequence "in" inside "rain", "Spain", "mainly", and "plain" were not counted.)

Note that if `word` is the empty `String` (""), the method call `occurrencesOfInString(word, line)` always returns 0, regardless of the value of `line`.

Part (a) [5 MARKS]

Assume that the document being analyzed has been read in by a Java program and stored in a `Vector` object. Each element in the `Vector` object is a `String` representing one line of text from the document.

Complete method `occurrencesOf` on the next page so that it counts and returns the number of occurrences of a given word in the entire given document. (This method could be used by the Writer's Workbench main method to count the frequency of word usage.) Your method should make use of the `occurrencesOfInString()` method. (Assume that both methods are in the same class.)

Question 4. (CONTINUED)**Part (a)** (CONTINUED)

```
public int occurrencesOf(String word, Vector document) {  
    // Complete this method according to the description on the previous page.
```

```
} // occurrencesOf()
```

Part (b) [10 MARKS]

In order to use method `occurrencesOfInString()` with confidence that it works properly, you need to test it thoroughly. List the test cases that you would use to completely test the `occurrencesOfInString()` method. For each test case, clearly indicate the purpose of the case and give an example `word` String and `line` String that could be used. You should provide *at least* 5 test cases.

(HINT: Start with 5 or 6 test cases and come back to provide more only if you have time at the end. Note that you *can* get full marks with 5 good test cases.)

Question 5. [20 MARKS]

Suppose that you are given the following Java class that stores information about a country.

```
class Country {
    private String name;    // country's name
    private int population; // number of people
    private double area;    // area in square kilometers

    public Country(String n, int p, double a) {
        name = n;
        population = p;
        area = a;
    }
    public String getName() { return name; }
    public int getPop() { return population; }
    public double getArea() { return area; }
} // class Country
```

Complete the class below by adding appropriate code as indicated in the comments. You may *not* change the `Country` class in any way, and you may assume that no country will have an area of zero. Also, assume that the user knows the input format: therefore, there will be *no* error in the input, and the user does *not* need any prompting.

```
import java.io.*;
public class HighDensity {
    public static void main(String[] args) throws IOException {
        BufferedReader in = new BufferedReader(new InputStreamReader(System.in));

        // Read the number of countries to create, then declare and instantiate an
        // array that is able to hold that number of Country objects.

        // Read information for each country and use this information to create the
        // Country objects in the array.
```

[More space on the next page...]

Question 5. (CONTINUED)

```
// Write code to find and print the name of the country that has the highest
// population density. The population density of a country is the number of
// people per square kilometer. You can assume that there will be at least
// one country in the array (so you do NOT have to check for this case). If
// two countries have exactly the same population density, print the name of
// the country that occurs earliest in the array.
```

```
} // main()
} // class HighDensity
```

There is NO question on this page!

*[If you need extra space to answer a question, use the space below and indicate **clearly** the question number.]*

Total Marks = 135

Student #: _____

Page 16 of 16

END OF EXAMINATION