MASSACHUSETTS INSTITUTE OF TECHNOLOGY

SP.772 FINAL EXAMINATION

WEDNESDAY, MAY 11, 2005

Name:

INSTRUCTIONS: You have three hours to complete this exam. There are 180 total possible points. Good luck.

Java Basics: Expressions and Statements (10 points)

1. (4 pts) Given the declaration `int x = 9`, what is the value of the following expressions?

```
x / 3.0 / 6 * 2     =
```

```
x % (x / 7) * 7     =
```

```
--x + ++x           =
```

```
--x == x++          =
```

2. (6 pts) Rewrite the following code segment using `if-else` statements. Assume that `grade` has been declared as a type `char`.

```
switch (grade) {
 case 'A':
     System.out.println("Excellent");
 case 'B':
     System.out.println("Good");
 case 'C':
     System.out.println("OK");
 default:
     System.out.println("Let's talk");
}
```

Compiling and Executing Java Code (20 points)

The questions in this section refer to the *incorrect* Echo program below. The Echo program accepts arguments from the command line and prints them (echoes them) to the screen. Line numbers are included for your reference and are not part of the program code. You may find it helpful to read all of the questions before beginning this section.

```
1    public class Echo {
2
3       public static void main(String[] args) {
4           print(args);
6       }
7
8       static void print(String[] greetings) {
9           for (int i = 0; i <= greetings.length(); i++)
10              System.out.print(greetings[i] + " ");
11          return true;
12      }
13   }
```

1. (4 pts) The Echo program will not compile as written; there are two *compile-time* errors in the *body* of the `print` method. Correct these errors by rewriting the body below.

    ```
    static void print(String[] greetings) {




    }
    ```

Assume you corrected the errors and want to run the program.

2. (2 pts) What do you name the file (what do you save the file as)?

3. (2 pts) What command do you type to compile `class Echo` (from the command prompt)?

4. (2 pts) What is the name of the file created when you compile `class Echo`?

3

5. (2 pts) Now that it is compiled, what command can you type to run the program?

When you run it, the Java interpreter gives the following *run-time* error:

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0
    at Echo.print(Echo.java:10)
    at Echo.main(Echo.java:4)
```

6. (2 pts) Which line (number) is actually responsible for the error?

7. (4 pts) Correctly rewrite this line:

Assume you save the changes, recompile, and run your program successfully.

8. (2 pts) What command do you type to run the program and produce the following output:

```
This test is hard
```

Methods (25 points)

The questions refer to the program `Quiz` below.

```java
public class Quiz {

    public static String[] mystery(String[] args) {
        int i = 0;
        while (i < (args.length + 1)/2) {
            String hold = args[i];
            args[i] = args[args.length - (++i)];
            args[args.length - i] = hold;
        }
        return args;
    }

    public static void main(String[] args) {
        String[] postMystery = mystery(args);
        for (int i = 0; i < postMystery.length; i++) {
            System.out.print(postMystery[i] + " ");
        }
    }
}
```

1. (10 pts) What does the `mystery` method do?

2. (15 pts) Write a side-effect free implementation of `mystery` with the same signature and behavior, as started below. (Hint: your implementation should not mutate its argument.)

```java
public static String[] mystery(String[] args) {




















}
```

Inheritance, Access Control, and Field Modifiers (35 points)

The first four questions reference three error-free Java files, which are printed on the next page: `Plant.java`, `Fruit.java`, and `Strawberry.java`. Each is in a separate package.

1. (5 pts) Add `import` statements to the files so that they compile.

2. (8 pts) Add a field to the `Strawberry` class to count the total number of `Strawberries` created in the program. The field should be accessible within `strawberries` package and within subclasses of `Strawberry`.

3. (10 pts) Implement the `Strawberry` constructor, remembering to update your count variable from question 2. A strawberry contains 3.2 kcal per gram.

4. (12 pts) Referencing your completed classes, write the output of the following code segments or write "error" if the code would generate a compiling or execution error:

Code Segment	Output
`Strawberry s = new Strawberry(10);` `System.out.println(s);`	
`Fruit f = new Strawberry(20);` `System.out.println((Plant) f);`	
`Plant p = (Strawberry) new Fruit(1, 3);` `System.out.println(p.nutrition());`	
`Plant p = new Strawberry(100);` `System.out.println(p.nutrition());`	

Plant.java

```
package plants;

//to do: Add import statement, if necessary.


public interface Plant {
    public double nutrition();
}
```

Fruit.java

```
package fruits;

//to do: Add import statement, if necessary.



public abstract class Fruit implements Plant {
    private double weight;
    private double kcalPerGram;

    public Fruit(double weight, double kcalPerGram) {
        this.weight = weight;
        this.kcalPerGram = kcalPerGram;
    }

    public double nutrition() { return weight * kcalPerGram; }

    public String toString() {
        return (nutrition() < 50 ? "delicious" : "nutritious");
    }
}
```

Strawberry.java

```
package strawberries;

//to do: Add import statement, if necessary.



public class Strawberry extends Fruit {

    //to do: Declare field to count number of strawberries created



    public Strawberry(double weight) {

        //to do: Implement constructor; remember to update count field




    }


    public String toString() { return super.toString()+" strawberry"; }
}
```

7

Implementing an Interface (40 points)

1. Write a class that implements the `Queue` interface, as shown below. A queue is a data structure that accepts data and then returns it in the order in which it was received (first-in, first-out order). Items are added to the tail of the queue and removed from the head.

```
public interface Queue {

    public int size(); //Returns number of objects in queue
    public boolean isEmpty(); //Returns true if queue is empty

    //Adds an item to the tail of the queue
    public void enqueue(Object o);

    //Removes and returns the item from the head of the queue
    public Object dequeue();

}
```

Your queue implementation must be accessible and usable from **any** package. However, any attempt to extend your class should produce a compile-time error. Your methods must not throw any exceptions.

A queue may be used as follows:

Sample main method	Output
```public static void main(String[] args) { Queue line = new AITIQueue(); line.enqueue("Hello"); line.enqueue("World"); System.out.println(line.dequeue()); System.out.println(line.dequeue()); }```	Hello World

Exceptions (25 points)

The first two questions refer to the program `Exceptional` below.

```java
public class Exceptional {

    public static void divide(String x, String y) {
        String ans = "";
        try {
            int dx = Integer.parseInt(x);
            int dy = Integer.parseInt(y);
            ans = String.valueOf(dx/dy);
        } catch (NumberFormatException nfe) {
            ans = "bad input";
        } catch (NullPointerException npe) {
            ans = "null input";
        } finally {
            System.out.print("answer: ");
        }
        System.out.println(ans);
    }

    public static void main(String[] args) {
        divide(args[0], args[1]);
    }
}
```

1. (12 pts) Write the output of the following commands or, if the program terminates abruptly, write the name of the exception that is printed to the console:

Command	Output
java Exceptional 4 2	
java Exceptional 3 null	
java Exceptional 1 0	
java Exceptional 2	

2. (3 pts) Write a command that will cause a `NullPointerException` to be thrown inside `divide` or write "none" if a `NullPointerException` can never be thrown.

10

3. (10 pts) Circle True or False

True / False   `RuntimeException` is a subclass of `Exception`.

True / False   A method cannot throw both checked and unchecked exceptions.

True / False   `Exception` is a subclass of `Error`.

True / False   An error-free class and all of its clients will continue to compile if the clause "`throws RuntimeException`" is added to the signature of its constructor.

True / False   An error-free class and all of its clients will continue to compile if the clause "`throws Exception`" is added to the signature of its constructor.

I/O and Parsing (25 points)

1. (4 pts) In order to read and write files in Java, you should import which packages (choose all that apply)?

   a) `javax.swing`   (+1 pt for *not* choosing)
   b) `java.util`   (+1 pt for *not* choosing)
   c) `java.awt`   (+1 pt for *not* choosing)
   d) `java.io`   (+1 pt for choosing)

2. (4 pts) Which classes could you instantiate to read from a character stream (choose all that apply)?

   a) `InputStream`   (+1 pt for *not* choosing)
   b) `InputStreamReader` (+1 pt for *not* choosing)
   c) `FileReader`   (+1 pt for choosing)
   d) `BufferedReader`   (+1 pt for choosing)

3. (4 pts) To write to a byte stream, you could use a class that extends which of the following classes (choose all that apply)?

   a) `OutputStreamWriter` (+1 pt for *not* choosing)
   b) `OutputStream`   (+1 pt for choosing)
   c) `FileWriter`   (+1 pt for *not* choosing)
   d) `Writer`   (+1 pt for *not* choosing)

4. (13 pts) Complete the method `squishInts` below. It expects a `String` argument containing integers separated by the @ character and should return the integer represented by the argument when all the occurrences of @ removed. For example, when passed the `String "34@1@100@5"`, it should return the integer 3411005. You can assume `java.util.*` has been imported. (Hint: use a StringTokenizer.)

```
public static int squishInts(String intString) {




















}
```

APPENDIX: Specifications of Selected Java Classes

	public abstract class `InputStream` extends `Object`
int	**available**() Returns the number of bytes that can be read (or skipped over) from this input stream without blocking by the next caller of a method for this input stream.
void	**close**() Closes this input stream and releases any system resources associated with the stream.
void	**mark**(int readlimit) Marks the current position in this input stream.
boolean	**markSupported**() Tests if this input stream supports the `mark` and `reset` methods.
abstract int	**read**() Reads the next byte of data from the input stream.
int	**read**(byte[] b) Reads some number of bytes from the input stream and stores them into the buffer array `b`.
int	**read**(byte[] b, int off, int len) Reads up to `len` bytes of data from the input stream into an array of bytes.
void	**reset**() Repositions this stream to the position at the time the `mark` method was last called on this input stream.
long	**skip**(long n) Skips over and discards `n` bytes of data from this input stream.

	public abstract class `OutputStream` extends `Object`
void	**close**() Closes this output stream and releases any system resources associated with this stream.
void	**flush**() Flushes this output stream and forces any buffered output bytes to be written out.
void	**write**(byte[] b) Writes `b.length` bytes from the specified byte array to this output stream.
void	**write**(byte[] b, int off, int len) Writes `len` bytes from the specified byte array starting at offset `off` to this output stream.
abstract void	**write**(int b) Writes the specified byte to this output stream.

public abstract class `Reader` extends `Object`	
abstract void	**close**() Close the stream.
void	**mark**(int readAheadLimit) Mark the present position in the stream.
boolean	**markSupported**() Tell whether this stream supports the mark() operation.
int	**read**() Read a single character.
int	**read**(char[] cbuf) Read characters into an array.
abstract int	**read**(char[] cbuf, int off, int len) Read characters into a portion of an array.
boolean	**ready**() Tell whether this stream is ready to be read.
void	**reset**() Reset the stream.
long	**skip**(long n) Skip characters.

public class `BufferedReader` extends `Reader`	
void	**close**() Close the stream.
void	**mark**(int readAheadLimit) Mark the present position in the stream.
boolean	**markSupported**() Tell whether this stream supports the mark() operation, which it does.
int	**read**() Read a single character.
int	**read**(char[] cbuf, int off, int len) Read characters into a portion of an array.
String	**readLine**() Read a line of text.
boolean	**ready**() Tell whether this stream is ready to be read.
void	**reset**() Reset the stream to the most recent mark.
long	**skip**(long n) Skip characters.

public class `InputStreamReader` extends `Reader`	
public class `FileReader` extends `InputStreamReader`	
void	**close**() Close the stream.
String	**getEncoding**() Return the name of the character encoding being used by this stream.
int	**read**() Read a single character.
int	**read**(char[] cbuf, int offset, int length) Read characters into a portion of an array.
boolean	**ready**() Tell whether this stream is ready to be read.

public abstract class `Writer` extends `Object`	
abstract void	**close**() Close the stream, flushing it first.
abstract void	**flush**() Flush the stream.
void	**write**(char[] cbuf) Write an array of characters.
abstract void	**write**(char[] cbuf, int off, int len) Write a portion of an array of characters.
void	**write**(int c) Write a single character.
void	**write**(String str) Write a string.
void	**write**(String str, int off, int len) Write a portion of a string.

public class `OutputStreamWriter` extends `Writer`	
public class `FileWriter` extends `OutputStreamWriter`	
void	**close**() Close the stream.
void	**flush**() Flush the stream.
String	**getEncoding**() Return the name of the character encoding being used by this stream.
void	**write**(char[] cbuf, int off, int len) Write a portion of an array of characters.
void	**write**(int c) Write a single character.
void	**write**(String str, int off, int len) Write a portion of a string.

public class `StringTokenizer` extends `Object` implements `Enumeration`	
StringTokenizer(String str) Constructs a string tokenizer for the specified string.	
StringTokenizer(String str, String delim) Constructs a string tokenizer for the specified string.	
StringTokenizer(String str, String delim, boolean returnDelims) Constructs a string tokenizer for the specified string.	
int	**countTokens**() Calculates the number of times that this tokenizer's `nextToken` method can be called before it generates an exception.
boolean	**hasMoreElements**() Returns the same value as the `hasMoreTokens` method.
boolean	**hasMoreTokens**() Tests if there are more tokens available from this tokenizer's string.
Object	**nextElement**() Returns the same value as the `nextToken` method, except that its declared return value is `Object` rather than `String`.
String	**nextToken**() Returns the next token from this string tokenizer.
String	**nextToken**(String delim) Returns the next token in this string tokenizer's string.

EC.S01 Internet Technology in Local and Global Communities
Spring 2005-Summer 2005