

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

SP.772 FINAL EXAMINATION

WEDNESDAY, MAY 11, 2005

INSTRUCTIONS: You have three hours to complete this exam. There are 230 total possible points. Good luck.

Name:

Java Basics: Expressions and Statements (10 points).....	2
Compiling and Executing Java Code (20 points)	3
Methods (25 points)	5
Writing a Basic Class (35 points)	6
Inheritance, Access Control, and Field Modifiers (35 points).....	8
Implementing an Interface (40 points).....	10
Exceptions (25 points)	12
I/O and Parsing (25 points).....	13
Swing (15 points).....	14
APPENDIX: Specifications of Selected Java Classes	15
public abstract class InputStream extends Object	15
public abstract class OutputStream extends Object	15
public abstract class Reader extends Object	16
public abstract class Writer extends Object	16
public class StringTokenizer extends Object implements Enumeration	17

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` =

`14 % (x / 5 + 1)` =

`--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");
}
```

```
if (grade == 'A') System.out.println("Excellent");
if (grade == 'A' || grade == 'B') System.out.println("Good");
if (grade == 'A' || grade == 'B' || grade == 'C')
    System.out.println("OK");
System.out.println("Let's talk");
```

Other solutions possible. Full credit given for semantically correct solutions.
5 points for correctly translating as if there were `break` statements.

Partial Credit:
+5 - Realize there are no `break` statements

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);
5      }
6
7      static void print(String[] greetings) {
8          for (int i = 0; i <= args.length; i++)
9              System.out.print(greetings[i] + " ");
10         return true;
11     }
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) {
```

```
    for (int i = 0; i <= greetings.length; i++)
        System.out.print(greetings[i] + " ");
    return; // may omit line entirely
```

```
}
```

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)?

```
Echo.java
```

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

```
javac Echo.java
```

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

Echo.class

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

java Echo

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?

9

7. (4 pts) Correctly rewrite this line:

```
for (int i = 0; i < greetings.length; i++)
```

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

java Echo This test is hard (with quotation marks okay)

Methods (25 points)

The questions refer to the program Quiz below.

```
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?

reverses the array args [in constant space]

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.)

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

```
    String[] newArray = new String[args.length];  
  
    for (int i = 0; i < args.length; i++)  
        newArray[args.length - 1 - i] = args[i];  
  
    return newArray;
```

Solution must be different from code provided.
Full credit for any semantically correct solution.

Partial Credit:
+3 - correct return type
NO other partial credit.

```
}
```

Writing a Basic Class (35 points)

1. (5 pts) On the next page begin a class that will represent a football team in a package called `football`. We want to be able to access the class `FootballTeam` from **any** package.
2. (4 pts) Add fields for the following properties, which **cannot** be accessed outside of the class.
 - name of the team
 - number of wins
 - number of losses
3. (4 pts) Write a constructor that accepts the name of the team, the number of wins, and the number of losses as arguments and sets the class properties to those values. This constructor should be accessible from **any** package.
4. (4 pts) Write a second constructor that takes only the name of the team as an argument. This constructor should set the name of the team to the argument and set the number of wins and losses to 0. (Hint: The body of this constructor should be one line and it should call the first constructor.) This constructor should be accessible from **any** package.
5. (4 pts) Write methods that return the name of the team, the number of the wins, and the number of losses. These methods should be accessible from **any** package.
6. (4 pts) Next write a method to increase the numbers of wins by 1 and another method to increase the number of losses by one. These methods should only be accessible from **inside** the `football` package.
7. (4 pts) Write a method that returns true when a team has a “good record,” meaning the team has more wins than losses. This method should be accessible from **any** package.
8. (6 pts) Finally, add a `main` method to the `FootballTeam` class. In the `main` method, construct a `FootballTeam` named "AITI" with 3 wins and 5 losses. Call the method that returns true when the team has a good record and print out the result. Now make three calls to the method that increases the number of wins by 1. Lastly, call the "good record" method again and print out the result.

```

/** Question 1 (5 pts)
 *   -2 for no package statement
 *   -2 for no public keyword
 *   -1 for misspelled or mis-capitalized class or package name.
 **/
package football;

public class FootballTeam {

    /** Question 2 (4 pts)
     *   -2 if any field is not private
     *   -2 if type of any field is wrong
     **/
    private String name;
    private int wins;
    private int losses;

    /** Question 3 (4 pts)
     *   -1 missing public keyword
     *   -1 wrong parameter types or number
     *   -2 for incorrect body
     **/
    public FootballTeam (String name, int wins, int losses) {
        this.name = name;
        this.wins = wins;
        this.losses = losses;
    }

    /** Question 4 (4 pts)
     *   -1 missing public keyword
     *   -1 wrong parameter types or number
     *   -2 for incorrect body
     **/
    public FootballTeam (String name) {
        this(name, 0, 0);
    }

    /** Question 5 (4 pts)
     *   -2 if any method has wrong signature
     *   -1 if any method missing
     *   -1 for wrong body(s)
     **/
    public String getName() { return name; }
    public int getWins() { return wins; }
    public int getLosses() { return losses; }

    /** Question 6 (4 pts)
     *   -2 if any method has wrong signature
     *   -2 for wrong body(s)
     **/
    void win() { wins++; }
    void lose() { losses++; }

    /** Question 7 (4 pts)
     *   -2 for wrong signature
     *   -2 for wrong body
     **/
    public boolean hasGoodRecord() {
        return (wins > losses);
    }

    /** Question 8 (6 pts)
     *   -2 for wrong signature
     *   -4 maximum for failure to follow specifications
     **/
    public static void main(String[] args) {

        FootballTeam aiti = new FootballTeam("AITI", 3, 5);

        System.out.println(aiti.hasGoodRecord());

        for (int i = 0; i < 3; i++)
            aiti.win();

        System.out.println(aiti.hasGoodRecord());
    }
}

```

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: `Shape.java`, `Rectangle.java`, and `Square.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 `Square` class to count the number of `Squares` created in the program.
3. (10 pts) Implement the `Square` constructor, remembering to update your count variable from question 14.
4. (12 pts) Referencing your completed classes, circle the output of the following code segments or circle "error" if the code would generate a compiling or execution error.

<pre>Square s = new Square(3); System.out.println(s.toString());</pre> <p style="text-align: center;"><u>square</u> rectangle error</p>	<pre>Rectangle r = new Square(3); System.out.println(r.toString());</pre> <p style="text-align: center;"><u>square</u> rectangle error</p>
<pre>Shape p = new Square(3); System.out.println(p.toString());</pre> <p style="text-align: center;"><u>square</u> rectangle error</p>	<pre>Square s = new Square(3); System.out.println(s.area());</pre> <p style="text-align: center;">3 <u>9</u> error</p>
<pre>Rectangle r = new Rectangle(3, 3); System.out.println(r.area());</pre> <p style="text-align: center;">3 9 <u>error</u></p>	<pre>Shape p = new Shape(3, 3); System.out.println(p.area());</pre> <p style="text-align: center;">3 9 <u>error</u></p>

5. Shape.java

```
package shapes;

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

/** -1 for putting an import statement here */

public interface Shape {
    public int area();
}
```

Rectangle.java

```
package rectangles;

//to do: Add import statement, if necessary.
/** -2 for failing to import package shapes */
import shapes;

public abstract class Rectangle implements Shape {

    private int width, height;

    public Rectangle(int width, int height) {
        this.width = width;
        this.height = height;
    }

    public int area() { return width * height; }

    public toString() { return "rectangle"; }
}
```

Square.java

```
package squares;

//to do: Add import statement, if necessary.
/** -1 for failing to import package rectangles */
import rectangles;
/** -1 for failing to import package shapes */
import shapes;

public class Square extends Rectangle implements Shape {

    //to do: Declare field to count number of squares created
    /** Question 14 (8 pts)
     * -2 if count not private
     * -4 if keyword static is missing
     * -2 if type is not int
     */
    private static int count = 0;

    public Square(int size) {

        //to do: Implement constructor; remember to update count field
        /** Question 15 (10 pts)
         * -5 for wrong call to superclass constructor
         * -5 for not updating the global counter
         */
        super(size, size);
        count++;
    }

    public String toString() { return "square"; }
}
```

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.

A queue may be used as follows:

<i>Sample main method</i>	<i>Output</i>
<pre>public static void main(String[] args) { Queue line = new AITQueue(); line.enqueue("Hello"); line.enqueue("World"); System.out.println(line.dequeue()); System.out.println(line.dequeue()); }</pre>	<pre>Hello World</pre>

```

//3 pts for "implements Queue"
//3 pts for keyword final
public final class QueueClass implements Queue {

    //3 pts for private field
    //3 pts for correct initialization
    //either in declaration or in a no-argument constructor.
    private List myQueue = new ArrayList();

    //2 pts for a public constructor
    public QueueClass() {}

    //2 pts for correct signature
    //3 pts for semantically correct body
    public int size() {
        return myQueue.size();
    }

    //2 pts for correct signature
    //3 pts for semantically correct body
    public boolean isEmpty() {
        return (myQueue.size()==0);
    }

    //2 pts for correct signature
    //6 pts for semantically correct body
    public void enqueue(Object o) {
        myQueue.add(o);
    }

    //2 pts for correct signature
    //6 pts for semantically correct body
    public Object dequeue() {
        if (!isEmpty())
            return myQueue.remove(0);
    }
}

```

Exceptions (25 points)

1. (8 pts) Describe two differences between checked and unchecked exceptions.

- Checked Exceptions (2 pts each)
 - subclass of `Exception`
 - must be handled by programmer (either with `try-catch` block or declared in method signature as thrown)
- Unchecked Exceptions (2 pts each)
 - subclass of `RuntimeException`
 - program will compile without explicit handling of unchecked exceptions

Award 1 pt partial credit for each other correct difference (no more than 2 differences per exception type)

2. (8 pts) Change the `divide` method so that it throws an `IllegalArgumentException` with an appropriate message if `b` is zero.

```
public static double divide(double a, double b) {  
    if (b==0)  
        throw new IllegalArgumentException("cannot divide by 0");  
  
    return a / b;  
}
```

- 2 pts for no message, i.e. missing "cannot divide by 0"
- 4 pts for incorrect syntax (missing `new`, etc.)
- 2 pts discretionary (really useless message, etc.)

3. (9 pts) Change this method so that it catches the `IllegalArgumentException` thrown by `divide` and prints the message "the divisor is zero" if it is thrown.

```
public static void printQuotient(double a, double b) {  
    try {  
        System.out.println(divide(a, b));  
    } catch (IllegalArgumentException e) {  
        System.out.println("the divisor is zero");  
    }  
}
```

- +3 pts – used (or attempted to use) `try-catch` block
- +2 pts – correct syntax of `try-catch` block
- +2 pts – catches `IllegalArgumentException`
- +2 pts – prints correct message (-1 if prints an incorrect msg)

I/O and Parsing (25 points)

- (4 pts) In order to read and write files in Java, you should import which packages (choose all that apply)?
 - javax.swing (+1 pt for *not* choosing)
 - java.util (+1 pt for *not* choosing)
 - java.awt (+1 pt for *not* choosing)
 - java.io** (+1 pt for choosing)
- (4 pts) To read from a character stream, you should use a class that extends which of the following abstract classes (choose all that apply)?
 - InputStream (+1 pt for *not* choosing)
 - OutputStream (+1 pt for *not* choosing)
 - Reader** (+1 pt for choosing)
 - Writer (+1 pt for *not* choosing)
- (4 pts) To write to a byte stream, you should use a class that extends which of the following abstract classes (choose all that apply)?
 - InputStream (+1 pt for *not* choosing)
 - OutputStream** (+1 pt for choosing)
 - Reader (+1 pt for *not* choosing)
 - Writer (+1 pt for *not* choosing)
- (13 pts) Complete the method `sumInts` below. It expects a `String` argument containing integers separated by the `@` character and should return the sum of those integers. For example, when passed the `String "34@1@100@5"`, it should return 140. You can assume `java.util.*` has been imported. (Hint: use a `StringTokenizer`.)

```
public static int sumInts(String intString) {  
  
    //4 pts for correct Tokenizer construction  
    StringTokenizer st= new StringTokenizer(intString,"@");  
  
    //2 pts for initializing sum to 0;  
    int sum=0;  
  
    //5 pts for semantically correct loop  
    while (st.hasMoreTokens()) {  
        sum += Integer.parseInt(st.nextToken())  
    }  
  
    //2 pts for correct return type (must be int)  
    return sum;  
  
}
```

Swing (15 points)

1. (4 pts) To create graphical interfaces in Java, you should import which packages (choose all that apply)?

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

2. (2 pts) Which corner of the screen has the pixel coordinates (0, 0)?

- a) Top-left** (+2 pts, no partial credit)
- b) Top-right
- c) Bottom-left
- d) Bottom-right

3. (3 pts) Name one Layout Manager and briefly describe how it lays out its components.

Either describe `FlowLayout` or `BorderLayout`. Other answers may be valid and should be checked in API.

- `FlowLayout`
 - Lays components out from left to right, top to bottom. (*full credit for this explanation or a logically equivalent one*)
- `BorderLayout`
 - Lays components out in five zones: North South Center East West

4. (6 pts, 1 pt for each correct match) Match the following Swing component with what it is used for:

Component

Use

- | | | |
|-----------------------------|----------|---|
| 1. <code>JFrame</code> | B | A) Generated when a <code>JButton</code> is clicked |
| 2. <code>JComponent</code> | D | B) An application window |
| 3. <code>JLabel</code> | F | C) Provides space for the user to type into |
| 4. <code>JPanel</code> | E | D) All the Swing components inherit from this class |
| 5. <code>TextField</code> | C | E) An intermediate container |
| 6. <code>ActionEvent</code> | A | F) Displays text that the user cannot edit |

APPENDIX: Specifications of Selected Java Classes

public abstract class InputStream extends Object	
int	<u>available()</u> 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	<u>close()</u> Closes this input stream and releases any system resources associated with the stream.
void	<u>mark(int readlimit)</u> Marks the current position in this input stream.
boolean	<u>markSupported()</u> Tests if this input stream supports the <code>mark</code> and <code>reset</code> methods.
abstract int	<u>read()</u> Reads the next byte of data from the input stream.
int	<u>read(byte[] b)</u> Reads some number of bytes from the input stream and stores them into the buffer array <code>b</code> .
int	<u>read(byte[] b, int off, int len)</u> Reads up to <code>len</code> bytes of data from the input stream into an array of bytes.
void	<u>reset()</u> Repositions this stream to the position at the time the <code>mark</code> method was last called on this input stream.
long	<u>skip(long n)</u> Skips over and discards <code>n</code> bytes of data from this input stream.

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

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

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