

CS 107 - Introduction to Programming
Midterm Exam #2 - Prof. Reed
Spring 2009

What is your name?: _____

There are three sections:

- I. True/False60 points; (30 questions, 2 points each)
- II. Multiple Choice40 points; (10 questions, 4 points each)

100 points total

This test is worth **15%** of your final grade. You must put your answers on the bubble form. All code is in Java unless stated otherwise. This test is open book and open notes. For the multiple choice problems, select the *best* answer for each one and select the appropriate letter on your answer sheet. *Be careful* - more than one answer may seem to be correct. Some questions are tricky. You have 50 minutes.

I. True/False: (2 points each)

T F 1. To change your password on the UNIX system, you need to use the *password* command.

T F 2. The following is a valid declaration in Java: `boolean while = true;`

T F 3. The output of the following statements is: 7 Done

```
int x = 3;
int y = 4;
System.out.print( "" + x + y);
System.out.println(" Done");
```

T F 4. The following statements compile and run in Java:

```
String order = "First";
if( order == "First") {
    System.out.println("Are same");
}
```

T F 5. Regardless of the code in the "Driver" section of code, every Java class that is used by other classes must fully implement all the "get" and "set" methods in order for it to compile.

T F 6. *Method overloading* refers to having multiple parameters in a method that are not used within the method.

T F 7. In order for the code shown below to compile and run properly, a *copy constructor* for the Date class must exist.

```
Date d1 = new Date(2,14,2000);
Date d2 = new Date( d1);
```

T F 8. To *chain* constructors you must use *this*.

T F 9. The contents of two objects can be compared using the `==` operator.

T F 10. The *toString* method gets executed only when you call it explicitly, such as in:

```
Date d1 = new Date();
System.out.println("Date is: " + d1.toString() );
```

T F 11. An object *reference* can function as a synonym for an object that already exists.

T F 12. If class *Transaction* contains a static variable called *howMany*, then each instance of the class will get its own copy of variable *howMany*.

T F 13. A *static* variable is just like a class instance variable except it's value cannot be changed.

T F 14. A *static* method can not access non-static instance variables.

T F 15. A non-static method can access static instance variables.

T F 16. Private instance variables can only be accessed through code in the class where they are declared.

T F 17. Two methods in a Java class can have the same method name.

T F 18. A *Constructor* is a special kind of method.

T F 19. If a *Constructor* has a return type at all it must be of type *boolean*.

T F 20. An array can be used on the left-hand-side of a statement, or on the right-hand-side, but not on both sides of the same statement.

T F 21. The index of an array reference can itself be an array element, such as in:

```
int x = theArray[ theArray[3] ];
```

T F 22. Although arrays normally start with 0 as the first index, we can change the default to be the value of our choice.

T F 23. Consider code discussed in class, where we had a *Square* class, a *Board* class, and a *PlayGame* class. Assume the following code would be in the *PlayGame* class. The code shown in the box below would give the same result as the code:

```
String theColor = theBoard.getSquareAtIndex( 5 ).getColor();
```

```
int x = 5;
Square aSquare = theBoard.getSquareAtIndex( x );
String theColor = aSquare.getColor();
```

T F 24. Consider the code discussed in class, where we had a *Square* class, a *Board* class, and a *PlayGame* class. Assume the following code would be in the *Board* class.

```
Square theSquare = new Square(25,25,60,"red",yes,"");
Square anotherSquare = theSquare;
anotherSquare.setColor("green");
```

When run this code will display two different colored Squares on the Canvas.

T F 25. Every set of *if - else - if* statements can be alternatively represented using a *switch - case* statement.

T F 26. Every set of *switch - case* statements can be alternatively represented using *if - else - if* statements.

- T F 27.** The following code in Java stores in variable *sum* the sum of positive even integers less than 10:

```
int sum;
int x;
for(x=0, sum=0; x<=10; x+=2)
    sum+=x;
```

- T F 28.** The `length()` method is used for both strings and arrays. You must be sure to include the parenthesis.

- T F 29.** Assume the code shown below, where method *swapValues2* is called.
Output of this segment of code is: Values are: 7 3

```
int[] numbers = {3,7};
swapValues2( numbers, 0, 1);
System.out.println("Values are: " + numbers[0] + " " +
                    numbers[1]);

// ... other code

public void swapValues2(int[] a, int num1, int num2)
{
    int temp = a[num1];
    a[num1] = a[num2];
    a[num2] = temp;
}
```

- T F 30.** Assume the code shown below, where method *swapValues* is called.
Output of this segment of code is: Values are: 7 3

```
int[] numbers = {3,7};
swapValues1( numbers[0], numbers[1]);
System.out.println("Values are: " + numbers[0] + " " +
                    numbers[1]);

// ... other code

public void swapValues1(int num1, int num2)
{
    int temp = num1;
    num1 = num2;
    num2 = temp;
}
```

II. Multiple Choice (5 pts. each)

31. Consider the program segment given below. Its output is:

```
for( int i=0; i<100; i++) {  
    System.out.printf("%4d", i+1);  
    if( (i+1)%10 == 0) {  
        System.out.println();  
    }  
}
```

- a) All the numbers from 0 to 99, with a line break after the number 10
- b) All the numbers from 0 to 99, with a line break after every 10 numbers
- c) All the numbers from 1 to 100, with a line break after the number 10
- d) All the numbers from 1 to 100 in a grid of 10 rows and 10 columns
- e) None of the above

32. Consider the code shown at right below, that uses the Circle class demonstrated during class. What does the output of this code look like?

- a) A circle that moves in an inward shrinking spiral.
- b) A circle that itself moves in a clockwise circle.
- c) A circle that itself moves in a counter-clockwise circle.
- d) A circle that moves in an outward growing spiral.
- e) None of the above.

```
public class picture  
{  
    public static void main()  
    {  
        Circle firstCircle = new Circle();  
        firstCircle.makeVisible();  
  
        for (int i=31;i>0;i--) {  
            firstCircle.slowMoveHorizontal(i*2);  
            firstCircle.slowMoveVertical(i*2);  
            firstCircle.slowMoveHorizontal((-i)*2);  
            firstCircle.slowMoveVertical((-i)*2);  
        }//end for( int i...  
    }//end main...  
}  
//end class picture
```

33. What is the output of the program segment shown at right below?

a) *

b) *****

**
*

c) *

**
*

d) *

**
*

e) None of the above

```

for( int i=1; i<=11; i++) {
    if( i < (11+1)/2) {
        for( int j=1; j<i*2; j++) {
            System.out.print('*');
        }
    }
    else {
        for( int j=1; j<((11+1)-i)*2; j++) {
            System.out.print('*');
        }
    }
    System.out.println();
} //end for( int i...

```

34. Consider method `first` shown at right. For positive numbers, how would you best describe its return value?

- a) $x + y$
- b) $x * x$
- c) $x * y$
- d) x^y
- e) None of the above

```

public int first(int x, int y)
{
    int z=0;
    for (int i=0; i<y; i++) {
        z += x;
    }
    return z;
}

public int second(int x, int y)
{
    int z=1;
    for (int i=0; i<y; i++) {
        z = first( z, x);
    }
    return z;
}

```

35. Consider method `second` shown at right. For positive numbers, how would you best describe its return value?

- a) $x + y$
- b) $x * x$
- c) $x * y$
- d) x^y
- e) None of the above

36. Consider the class given below, along with the driver class for it.

<pre>class ClassB { private int value; public ClassB() { value = 0; } public void setValue(int value) { value = value + 1; } } //end ClassB</pre>	<pre>class ClassBDriver { public void doIt() { int number = 3; ClassB instance1 = new ClassB(); instance1.setValue(number); number = instance1.value; System.out.println("value is: "+number); } } //end ClassBDriver</pre>
---	--

When running method doIt() in the ClassBDriver class, the output will be:

- a) value is: 3
- b) value is: 0
- c) value is: 4
- d) doesn't compile
- e) None of the above

37. Assume that you create class *Employee* that includes an instance of class *Date* to store the startDate for each employee. Assume that you have written some test code in class *EmployeeDriver* shown below, where you change the startDate for e1. To your surprise when you run this code the startDate for e2 has changed as well. What is the *most likely* explanation for this?

```
class EmployeeDriver
{
    public static void main(String[] args)
    {
        Employee e1 = new Employee();
        Employee e2 = new Employee( e1);

        e1.changeDate( 9, 30, 1923);

        System.out.println(e1);
        System.out.println(e2);
    }
}
```

- a) The Date class fields are declared as *static*
- b) The Employee copy constructor does not create a new Date
- c) The Date class copy constructor does not chain to the fully qualified constructor
- d) The new value happens to be the same as the default value
- e) None of the above

38. What is the output of the code given in the two columns below when an object of type *Confuse* is created and used to call method *startUp()*?

<pre>class Confuse { private int x; private int y; public Confuse() { x = 2; y = 7; } private void first(int z) { y = z; y++; } private void second(int s, int t) { setXY((y-s), (x+t)); s = 1; t = 3; } }</pre>	<pre>private void setXY(int s, int y) { x = s; this.y = y; first(y); } private void display() { System.out.println(x + y); } public void startUp() { first(y); second(y, x); display(); } } //end class Confuse</pre>
--	--

- a) 1
- b) 3
- c) 5
- d) 8
- e) None of the above

39. What is the output of the code given below when method *useArray()* is called?

<pre>public void useArray() { char[] theArray= {'E','n','t','r','o','p','y'}; processArray(theArray); for(int i=0; i<theArray.length; i++) System.out.print(theArray[i]); }</pre>	<pre>public void processArray(char[] w) { int x = w.length; char c; for (int i=0; i<x; i++) { c = w[i]; w[i] = w[x-i-1]; w[x-i-1] = c; } }</pre>
--	---

- a) The contents of the original array in reverse order
- b) The contents of the original array in the original order
- c) The contents of the original array with half of the characters reversed
- d) The original array with characters rearranged so they are neither in the original nor reversed order
- e) None of the above

40. What is the output of the code given below when called with:

ClassE classInstance = new ClassE(3);

```
class ClassD
{
    private int x;

    public ClassD( int value)
    {
        x = value;
        method1( x);
    }

    private void method1( int x)
    {
        x+=2;
        ClassE instance1 =
            new ClassE( x);
    }
} // end of classD
```

```
class ClassE
{
    private int x=2;

    public ClassE(int x)
    {
        if ( x == 5) {
            ClassD instance1 =
                new ClassD( x);
        }
        x++;
        System.out.print(" " + x);
    }

    public void method1( int x)
    {
        x = x + 1;
    }
} // end of classE
```

- a) 2
- b) 4
- c) 2 4
- d) 2 3 4
- e) None of the above