

CST8132: Object-Oriented Programming (in Java)

Midterm Test: Part A

Thursday March 6, 2014
Course Professor: Rex Woollard

1. This midterm test will be conducted during the regular 2-hour lecture.
2. You will have no aids in completing this test.
3. The midterm is split into two parts: A and B.
4. Part A is a mix of multiple choice and short answer. All answers are to be entered on a the test paper. Part A will be worth 21 marks.
5. Part B involves writing Java program code to solve a problem. Part B will be worth 24 marks: program code 20; and the memory map 4. Write your answer for Part B on separate blank sheets of paper. Your solution will be stapled to this test paper when complete.
6. You will begin with Part A. When you have completed Part A, submit it to me. At that point, you can choose to take an unsupervised break before receiving Part B.
7. You may separate the pages in the test. They will be re-stapled when you submit the test.

Name: _____

Multiple Choice Questions

Answer the following multiple choice questions on the attached sheet.

1. Immediately after this statement executes, what best describes the result:

```
Actor[] x = new Actor[1000];
```

- a) **x** is an array of 1000 **Actor** objects.
 - b) **x** is an index into an array of **Actor** objects.
 - c) **x** is allocated on the *heap* and the resulting array is allocated on the *stack*.
 - d) **x** refers to an array of references to objects which could be of any subclass type (such as **Hobbit** or **Wizard**, etc.)
 - e) generates a syntax error because **Actor** is an **abstract** class.
2. Immediately after the last statement executes, what best describes the result:

```
Actor x = new Wizard();  
// ... some additional code  
x = new Hobbit();
```

- a) The **Wizard** object will be eligible for garbage collection.
 - b) **Wizard** is a subclass of **x**.
 - c) **x** becomes a **Hobbit** object.
 - d) The statement generates a warning about overwriting.
 - e) None of the above.
3. The term *signature* can be used when describing a method. In this context, a *signature* is:
- a) Defined by the **import** statement for that class.
 - b) The number of arguments and the data type of each argument.
 - c) The return type of method (for example, **double** or **String**).
 - d) The variable names in the argument list.
 - e) All of the above.
4. Which of the following is an application of the principle of *inheritance*:
- a) An object of class **A** has a reference to a class **B** object.
 - b) Several methods have the same name, but have different *signatures*.
 - c) Objects created with **new** are allocated on the *heap*.
 - d) Fields are usually declared **private**.
 - e) All classes are ultimately derived from the *super class* called **Object**.
5. Consider the following statement which is defined in a class (let's call it class **Test**). The keyword **static**:

```
private static int serial = 0;
```

- a) means that **serial** is a constant.
 - b) ensures that only one instance of **serial** exists and it will be shared by all objects of type **Test**.
 - c) means that **serial** should be capitalized (e.g. **SERIAL**) to comply with Java naming conventions.
 - d) results in a syntax error because it is missing the keyword **final**.
 - e) None of the above.
6. A *virtual method* requires which of the following to be in effect:
- a) The method signatures must be identical in the *super class* and *subclass*.
 - b) The *subclass* overrides all methods in the ultimate *super class* (called **Object**).
 - c) The keyword *virtual* is applied to the method name.
 - d) The *super class* must be declared to be **abstract**.
 - e) All of the above.
7. Which of the following is true about a *primitive* variable:
- a) It holds the raw machine-code address of a variable.
 - b) During program execution, accessing a primitive is slower than accessing a reference-based object.
 - c) Primitives are close to the machine-code level, thus have different byte sizes on different hardware platforms.
 - d) The value stored in the variable is an actual value (as opposed to a reference to something).
 - e) None of the above.
8. Which of the following is true about a *reference* variable:
- a) It holds the raw machine-code address of a variable.
 - b) When created, it will always contain location information for some object.
 - c) It can hold location information for objects of the named class, or objects of any subclass.
 - d) Once the location information in the *reference* variable is established, it cannot be changed.
 - e) None of the above.
9. Which of the following statements about *constructors* is correct:
- a) A *constructor* has the same name as the class name.
 - b) A *constructor* is responsible for the initialization of an object's instance fields.
 - c) *Constructor* methods have no return type.
 - d) A class can have several *constructors*.
 - e) All of the above.

10. You know that memory is allocated when a *stack-oriented* variable is created in a method. That memory is released when:
- Program execution enters a **catch** block.
 - The garbage collector discovers that there are no outstanding references to the variable.
 - The program execution leaves the block of code where the variable was defined.
 - The class loader detects an exception.
 - None of the above.
11. In the example code fragment shown below, the keyword **abstract**:

```
public abstract class Test { // . . . more class code
```

- Implies that no object of type **Test** can ever be created.
 - Makes class **Test** independent of all other classes, in particular, it is not a subclass of the class **Object**.
 - Requires the implementation of the **toString()** method.
 - Ensures that only one object of type **Test** is ever created.
 - All of the above.
12. The following statements compare and contrast *array* and **ArrayList**. Which one is true (or are all true)?
- Both **ArrayList** and *array* objects automatically keep track of their capacity and the number of elements actually in use.
 - The *array* type **extends** from **Object**. The **ArrayList** type **extends** from *array*.
 - The elements in an **ArrayList** can be primitives (**int**, **float**, **double** etc.) or reference-to values. An *array* can store only reference-to values.
 - ArrayList** can expand its storage space as needed; an *array* cannot change the initial size of the array.
 - All of the above.

The following declaration will be used in the next three questions:

```
class SuperClass {
    private int x;
    private int y;
    public SuperClass ( )
        { x = 2; y = 3; }
    public SuperClass (int x, int y)
        { this.x = x; this.y = y; }
    public String toString ( )
        { return "Numbers are: " + x + " and " + y; }
    public int returnSum ( )
        { return (x+y); }
}

class SubClass extends SuperClass {
    private int z;
    public SubClass ( )
        { super ( ); }
    public SubClass(int x, int y)
        { super(x, y); z = 4; }
    public int returnSum ( )
        { return (super.returnSum() + z); }
}
```

and the declarations:

```
SuperClass obj1 = new SuperClass();
SuperClass obj2 = new SubClass(1, 2);
```

13. What will the following statement display to the screen?

```
System.out.println (obj1);
```

- Something like: **SuperClass@4e3380fa** (the **Object** implementation of **toString()**)
- Numbers are: 1 and 2**
- Numbers are: 2 and 3**
- Numbers are: 2 and 3 and 9**
- None of the above.

14. What will the following statement display to the screen?

```
System.out.println(obj2);
```

- Something like: **SubClass@4e7770bb** (the **Object** implementation of **toString()**)
- Numbers are: 1 and 2**
- Numbers are: 2 and 3**
- Numbers are: 1 and 2 and 3**
- None of the above

15. Given the following statement, what will display to the screen?

```
System.out.print("Sum is:" + obj2.returnSum());
```

- Sum is: 20**
- Sum is: 200**
- Sum is: 4**
- Invalid statement – won't compile
- None of the above.

Trace Program Execution (3 Marks)

Show the output that will result from the following code:

```
public class TestStuff {
    public static void main(String[] args) {
        Stuff s = new Stuff("in", 5);
        System.out.println(s);
        double doubleValue = 2.5;
        s.doSomething(doubleValue);
        System.out.println(doubleValue);
        s = new Stuff("more", 3);
        String str = "word";
        System.out.println(s.changeSomething(str));
        System.out.println(s);
        System.out.println(str);
    }
}
```

```
public class Stuff {
    private static final int n = 2;
    private String string;
    private int num;

    public Stuff(String s, int num) {
        this.num = num;
        string = s;
    }

    public void doSomething(double d) {
        d = d * num;
        System.out.println(this);
    }

    public double changeSomething(String s) {
        s = string;
        return n * num;
    }

    public String toString() {
        return string + " has " + num;
    }
}
```

Program Output

Identify Syntax Errors (3 Marks)

Find 3 syntax and / or logic errors. Circle each error and suggest a correction.

```
public class Hobbit extends Actor {
    public static constant double MAX_STEALTH = 100.0;
    private double stealth;

    public void Hobbit() {
        stealth = MAX_STEALTH / 2.0;
    }

    public void setStealth(double stealth) {
        stealth = stealth;
    }

    public void displayStatus() {
        displayStatus();
        System.out.printf(" Stealth: %d", stealth);
    }

    public toString() {
        return String.format("%s Stealth:%4.1f",
            super.toString(), stealth);
    }
} // end class Hobbit
```