**Name:**_____

# CST:8132: OOP (in Java): Midterm: Part 1: Solution

**You have 1 hour 50 minutes for both parts of this midterm. After completing *Part 1*, you must submit it before receiving *Part 2*. You may take an unsupervised break between *Part 1* and *Part 2*.**

## Part 1: 24 marks  Part 2: 29 marks

### *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 a reference variable that now stores the location information of an array of 1000 references to objects of type *Actor*.◄
   c) *x* is allocated on the *heap* and the resulting array is allocated on the *stack*.
   d) *x* is an array of 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) *x* is garbage collected before the new *Hobbit()* object is created..
   b) *Wizard* is a subclass of *x*.
   c) *x* is a reference variable holding location information for a *Hobbit* object. ◄
   d) The statement generates a warning about overwriting.
   e) All of the above.

3. The term *signature* can be used when describing a method. In this context, a *signature* is:
   a) The number of arguments sent to the method.
   b) The precise name of the method (including upper/lower case).
   c) The data type of arguments sent to the method.
   d) 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) Fields are usually declared *private*.
   d) All classes are ultimately derived from the *super class* called *Object*. ◄
   e) Objects created with *new* are allocated on the *heap*.

5. Consider the following statement. The keyword **final** means:

```
private final int MAX = 1000;
```

   a) **MAX** is a constant.◄
   b) **MAX** will never be garbage collected.
   c) A single instance of **MAX** will be shared by all objects created from the class where **MAX** is defined.
   d) Capitalizing **MAX** is part of the Java language syntax specification.
   e) All 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 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 an object is created using a *new* operator. That memory is released when:

a) The *delete* operator is called using the reference variable to define the location of the object.

b) Program execution enters a *catch* block.

c) The garbage collector discovers that there are no outstanding references to the object. ◄

d) The class loader detects an exception.

e) None of the above.

11. In the example code fragment shown below, the keyword *abstract*:

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

a) Implies that no object of type *Test* can ever be created. ◄

b) Makes class *Test* independent of all other classes, in particular, it is not a subclass of the class *Object*.

c) Is needed so that the class *Test* can implement virtual methods.

d) Ensures that only one object of type *Test* is ever created.

e) All of the above.

12. The following statements compare and contrast *array* and *ArraryList*. Which is true?

a) Both *ArrayList* and *array* objects automatically keep track of their capacity and the number of elements actually in use.

b) *ArrayList* can expand its storage space as needed; an *array* cannot change the initial size of the array. ◄

c) The *array* type extends from *Object*. The *ArrayList* type extends from *array.*

d) The elements in an *ArrayList* can be primitives (*int*, *float*, *double* etc.) or reference-to values. An *array* can store only reference-to values.

e) All of the above.

---

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

```
class SuperClass {
    private int x;
    private int y;
    public SuperClass ( ) { x = 10;  y = 20; }
    public SuperClass (int x, int y) {  this.x = x;     this.y = y;  }
    public String toString ( )
       { return "Numbers are: " + x  + " and " + y;   }
    public  int  returnProduct ( ) { return (x*y); }
}
class SubClass extends SuperClass {
    private int z;
    public SubClass( )  { super();  z = 30; }
    public int returnProduct( )  { return (super.returnProduct() *
z); }
};
```

and the declarations:

```
SuperClass obj1 = new SuperClass(4, 5);
SuperClass obj2 = new SubClass();
```

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

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

a) Nothing

b) Numbers are: 10 and 20

c) Numbers are: 10 and 20 and 30

d) None of the above. Numbers are: 4 and 5 ◄

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

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

a) Nothing

b) Numbers are: 10 and 20 ◄

c) Numbers are: 10 and 20 and 30

d) None of the above

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

```
System.out.print("Product is:" + obj2.returnProduct());
```

a) Product is: 20

b) Product is: 200

c) Product is: 6000 ◄

d) Invalid statement – won't compile

e) None of the above.

16. Which statement will create an array of 100 references to SuperClass objects:

a) **SuperClass [100] numbers;**

b) **SuperClass numbers [100];**

c) **SuperClass numbers [99];**

d) **SuperClass numbers [101];**

e) None of the above. ◄

For the following 2 questions, assume that the following class specification is part of the program:

```
public class Actor {
    private String name;
    private int strength;
    private static int count;

    public Actor(String name, int strength) {
        this.name = name;
        this.strength = strength + ++count;
    }

    public String toString() {
        return "Name:"+name+" Strength:"+strength;
    }

  // many public methods
} // end class Actor
```

```
public class TestActor {
    public static void main(String[ ] args) {
        Actor a1 = new Actor("Bob", 33);
        Actor a2 = new Actor("Bill", 22);
        Actor a3 = new Actor("Boris", 11);
        System.out.println(a3);

        // more code
```

17. What will be output by **System.out.println(a3);**
    a) *Name:BobBillBoris Strength:66*
    b) *Name:Boris Strength:11*
    c) *Name:Boris Strength:11.0*
    d) **strength + ++count** is a syntax error. Won't execute.
    e) None of the above. ◄ Name:Boris Strength:14

18. Assume that you do now have a correctly built array of 100 references to *Actor* objects managed through the variable *aaActors*. Identify the code needed to create a *Hobbit* object and capture it as the second element in the array.
    a) *aaActors[1] = new Hobbit( );* ◄
    b) *Actor[2] = new Hobbit( );*
    c) *new Actor[1] = Hobbit( );*
    d) *aaActors[2] = new Hobbit( );*
    e) 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("testing", 0.5);
        System.out.println(s);
        double doubleValue = 2.5;
        s.doSomething(doubleValue);
        System.out.println(doubleValue);
        s = new Stuff("again", 1.5);
        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 double d;

   public Stuff(String s, double d) {
        this.d = d;
        string = s;
   }

   public void doSomething(double d) {
        d = d * 2;
        System.out.println(this);
   }

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

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

**Your Answer:**

**testing has 0.5**
**testing has 0.5**
**2.5**
**3.0**
**again has 1.5**
**word**

## *Identify Syntax Errors (3 Marks)*

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

```
public class Actor extends Hobbit {
 public static final 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() {
   super.displayStatus();
   System.out.printf(" Stealth: %d", stealth);
   }

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

Hobbit extends Actor

Use “this.” to assign to the instance variable

Calls itself, need *super.toString()*