

1. Matching [4 pts]

Choose the **best** definition for each of the words below.

1. _____ Superclass
2. _____ Realize (implement) an interface
3. _____ Late (dynamic) binding
4. _____ Overridden Method
 - A. A class which cannot be extended.
 - B. The more general class in an "is-a" relationship.
 - C. When a class has the actual method bodies required by an interface.
 - D. Emits a low level tachyon field when struck by anti-protons.
 - E. The process by which method calls are resolved at runtime.
 - F. Used when classes exhibit an "is-a" relationship.
 - G. The ratio of coupling to cohesion. Important to consider when analyzing how good a design is with regards to the OO paradigm.
 - H. When a class changes the behavior of a method that it would otherwise inherit.
 - I. A class which does not allow its state to be changed by other classes.
 - J. The more specific class in an "is-a" relationship.

2. Misguided Java Programmer [10 pts]

A misguided Java programmer wrote the following code:

```
public class Misguided{
    private String name;
    private int number;
    public Misguided(String name, int number){
        this.name=name;
        this.number=number;
    }

    public static void main(String[] args){
        Misguided mg1=new Misguided("hello",57);
        System.out.println(mg1);
        Misguided mg2=new Misguided("world",3);
        System.out.println(mg2);
    }
} //class Misguided
```

The programmer expected the output to be

hello 57

world 3

however instead it was

Misguided@256a7c

Misguided@720eeb

- 4 (a) Explain why the programmer received this output. Your explanation should include both why the programmer did not get the desired output, as well as why the actual output appeared.
- 6 (b) Without modifying the code in the **main** method, fix the code above so that the program works as it should. (Mark your change clearly in the code above).

3. Object Concepts [11 pts]

- 6 (a) Explain the difference between `==` and `equals()`. Be sure to include:
1. How `==` compares for equality
 2. How `equals()` compares for equality
 3. When to use each
- 5 (b) Compare or contrast interfaces and abstract classes. (You should either explain how they are similar, or how they are different- you do not need to do both).

4. Tracing [18 pts]

Given the following code:

```
class Tracing1Parent{
    public Tracing1Parent(){
        this(2);
        myMethod("default");
    }
    public Tracing1Parent(int n){
        myMethod(n);
    }
    public void myMethod(String s){
        System.out.println("Parent has "+s);
    }
    public void myMethod(int n){
        myMethod("starting loop with "+n);
        for(int i=0;i<n;i++){
            myMethod("something with "+n);
        }
    }
}
} //Tracing1Parent
public class Tracing1 extends Tracing1Parent{
    public Tracing1(){
        super();
        myMethod(-5);
    }
    public void myMethod(String s){
        System.out.println("Child has "+s);
    }
    public static void main(String[] args){
        Tracing1 t=new Tracing1();
        Tracing1Parent tp =t;
        tp.myMethod(1);
        ((Tracing1Parent)t).myMethod("a message");
        t.myMethod(3);
    }
} //class Tracing1
```

What is the output when the class is run?

5. Datatypes and Casting [17 pts]

Assume that you have the following class structure (where appropriate):

- The class `Weapon` is abstract.
- `SharpThing` is an interface.
- The class `Sword` extends `Weapon` and implements `SharpThing`
- The class `Sabre` extends `Sword`

For each of the following code fragments, write "COMPILER ERROR" if the code would result in a compiler error, write "RUNTIME ERROR" if the code would result in a runtime error, and write the output to the screen if the code would execute without error. (You may assume that all classes have a default constructor).

- 1 (a) `Weapon a=null;`
`System.out.println(a);`
- 1 (b) `SharpThing b=null;`
`System.out.println(b);`
- 2 (c) `double c=12;`
`int x=c;`
`System.out.println(x);`
- 3 (d) `Weapon d= new Sabre();`
`System.out.println(d instanceof Sword);`
- 3 (e) `Sabre f= new Sword();`
`System.out.println(f instanceof SharpThing);`
- 3 (f) `SharpThing g= new Sabre();`
`System.out.println(g instanceof Weapon);`
- 4 (g) `SharpThing h= new Sword();`
`Weapon y=(Sabre) h;`
`System.out.println(y instanceof SharpThing);`

6. Iteration I [10 pts]

Write the method `public void createNumberTri(int n)` which takes a number `n` and prints a triangle with `n` rows in the following format:

1

22

333

4444

For this method you must use **for loops only**-if you use any other type of loop, you will receive zero credit.

7. Iteration II [15 pts]

Write the method **public void drawTriangle(int n)** which takes a number and draws a centered triangle of numbers with **n** rows. Blank spaces should be filled in with dots. For example, if 4 were passed in, this method should print:

```
...1...
```

```
..2.2..
```

```
.3.3.3.
```

```
4.4.4.4
```

(Hint: For the k^{th} row (where the first row is $k=1$), you print $n-k$ dots, then alternate between the number and a dot, then print $n-k$ more dots). You may assume $n > 0$. For this method you must use **while loops only** - if you use any other type of loop, you will receive zero credit.

```
public void drawTriangle(int n) {
```

```
}
```

8. Colors and Drawing [15 pts]

Below, you are given a skeleton applet class, with the method `setPixel`, which uses `fillRect` to draw a single pixel. Fill in the `paint` method below so that for each pixel (x,y) , the pixel is colored in with a `Color` such that:

```
red=x+y
```

```
green=x
```

```
blue=y
```

You might wish to remember the following methods from the `Applet` class:

```
public int getWidth()
```

```
public int getHeight()
```

```
import java.awt.*;
import java.applet.*;
public class ColorApplet extends Applet{
    public void paint(Graphics g){
```

```
    }//paint
    public void setPixel(Graphics g, int x, int y, Color c){
        g.setColor(c);
        g.fillRect(x,y,1,1);
    }//setPixel
}//class ColorApplet
```