

1. Matching [5 pts]

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

1. _____ Recursion
 2. _____ Public method
 3. _____ Private method
 4. _____ Array
 5. _____ Polymorphism
- A.** The process by which method calls are resolved at runtime based on the actual type of the object on which the method is invoked.
 - B.** A fixed length data structure that has elements of a specific type
 - C.** A programming technique in which loops are used to allow an action to be taken multiple times.
 - D.** A method that returns a boolean value.
 - E.** A method that is accessible by all classes
 - F.** A programming technique whereby a method calls itself.
 - G.** The ability of an instance of a subclass to be treated as if it were an instance of its super class
 - H.** A dynamic data structure that stores a collection of objects.
 - I.** The case in a recursive method in which no recursion is required, as the answer is immediately known.
 - J.** A method that is accesible only by a class and its inner classes.
 - K.** The process by which method calls are resolved at compile time based on the declared type of the object on which the method is invoked.

2. Tracing [15 pts]

What is the output when the main method in the code below is run?

```
class TraceSuper {
    public String name;
    TraceSuper() {
        this("Super");
        showName(17);
    }
    TraceSuper(String s){
        name = s;
        showName(3);
    }
    void showName(int n) {
        System.out.println(name + ": " + n);
    }
}
//-----
public class TraceB extends TraceSuper {
    public TraceB(String n) {
        showName(89);
    }
    void showName(int n) {
        System.out.println(name + "(child): " + n);
    }
    public static void main(String[] args) {
        TraceSuper ar[] = new TraceSuper[2];

        ar[0] = new TraceB("Sally");
        ar[1] = new TraceSuper("George");
        for(int i = 0; i < 2; i++ ) {
            ar[i].showName(2 - i);
        }
    }
} // end of main(String[] args)

} // end of class TraceB
```

3. Polymorphism [12 pts]

Given the following class hierarchy:

- Number is abstract.
- Comparable is an interface
- Float extends Number and implements Comparable

Consider code inside a method with the header

```
public void someMethod(Object o, Number a, Comparable b, Float c)
```

For each of the following array declarations, circle **all** assignments that are legal (as written, with no extra casts). Be sure to circle **all** that are legal.

3 (a) `Float[] arr4=new Float[5];`

<code>arr4[0]=o;</code>	<code>arr4[0]=a;</code>	<code>arr4[0]=b;</code>	<code>arr4[0]=c;</code>
<code>o=arr4[0];</code>	<code>a=arr4[0];</code>	<code>b=arr4[0];</code>	<code>c=arr4[0];</code>

3 (b) `Comparable[] arr3=new Comparable[5];`

<code>arr3[0]=o;</code>	<code>arr3[0]=a;</code>	<code>arr3[0]=b;</code>	<code>arr3[0]=c;</code>
<code>o=arr3[0];</code>	<code>a=arr3[0];</code>	<code>b=arr3[0];</code>	<code>c=arr3[0];</code>

3 (c) `Number[] arr2=new Number[5];`

<code>arr2[0]=o;</code>	<code>arr2[0]=a;</code>	<code>arr2[0]=b;</code>	<code>arr2[0]=c;</code>
<code>o=arr2[0];</code>	<code>a=arr2[0];</code>	<code>b=arr2[0];</code>	<code>c=arr2[0];</code>

3 (d) `Object[] arr1=new Object [5];`

<code>arr1[0]=o;</code>	<code>arr1[0]=a;</code>	<code>arr1[0]=b;</code>	<code>arr1[0]=c;</code>
<code>o=arr1[0];</code>	<code>a=arr1[0];</code>	<code>b=arr1[0];</code>	<code>c=arr1[0];</code>

4. **Parameter Passing [12 pts]**

Given the following code:

```
public class Test4 {  
  
    public static void product( int x, int result ) {  
        result = result * x;  
    }  
  
    public static void main(String[] args) {  
        int result = 1;  
        product(5, result);  
        product(-3, result);  
        product(2, result);  
        System.out.println("The product is " + result);  
    } // end of main(String[] args)  
} // end of class Test4
```

The programmer expected the output to say **The product is -30**.

- 4 (a) Which of the following actually occurs with the code above:
- The code compiles without error and produces the expected output.
 - The code compiles without error, but produces some other output.
 - The code causes a compiler error.
- 4 (b) Explain your answer from (a). If the program works correctly, explain why it does so, with specific reference to the concept of parameter passing. If the program produces the incorrect output, include the actual output in your explanation. If the program will not compile, include why in your explanation.
- 4 (c) If the program does not work properly, fix the code so that it works properly by clearly marking your changes above. Your fix may not be superficial, i.e. you may not simply cross out all of the code and simply print the expected output. Your fix should address the issues explained in (b).

5. Iteration [11 pts]

Write the method `public int countLettersMatching(String s, char c)` which returns the number of letters in `s` which match the letter `c`. You **must use iteration only** for this method. If you do not use iteration, or if you use any recursion, you will receive no credit.

```
public int countLettersMatching(String s, char c) {
```

```
}
```

6. Recursion [13 pts]

Write the method **public int power(int x, int y)** which returns x^y . Remember that any number raised to the 0th power is 1. You **must use recursion only** for this method. If you do not use recursion, or if you use any iteration, you will receive no credit.

```
public int power(int x, int y) {
```

```
}
```

7. Arrays [16 pts]

Write the method **public boolean isAll(double[] data, double x)** which returns *true* if all elements in **data** are **x**, *false* otherwise. You may use iteration and/or recursion, as you see fit for this problem. You may assume that **data** is non-null. It may be useful to remember that for a 0 element array, this method should always return *true*.

```
public boolean isAll(double[] data, double x) {
```

```
}
```

8. **Vectors [16 pts]**

Write the method **public boolean contains(Vector data, int x)** which takes a Vector containing Integers and returns *true* if **data** contains an element representing the value **x**, *false* otherwise. You may use iteration and/or recursion, as you see fit for this problem. You may assume that **data** is non-null. You **may not** use the builtin **contains** method in Vector for this problem, or you will receive no credit. It may be useful to remember that a Vector of 0 elements does not contain anything.

```
public boolean contains(Vector data, int x) {
```

```
}
```