CSC 142 Review:

Select the correct answer between the parentheses:

A class may inherit from (only one/ one or more) class(es) and implement (only one/one or more) interfaces.

e.g.

```
public class A{}
public interface I1{}
Public interface I2{}
public class B extends A implements I1,I2{}
```

Answer briefly the following questions:

What is the Object class?

The root of any class hierarchy

What does instanceof do? Give an example

```
B b = new B();
b instanceof A → true if B is an A
              → false if B is not an A
```

What does getClass() do? Give an example

returns the dynamic type of a variable

```
A
↑
B
```

```
A a = new B();
System.out.print(a.getClass());
→ prints class B
```

Explain the differences and similarities between a concrete class, an abstract class, and an interface by labeling the following statements as true or false.

A concrete class / an abstract class/ an interface

defines a type → true for concrete class, abstract class, interface

can be instantiated

only for a concrete class

contains constructors

only for a concrete class or an abstract class

may contain abstract methods

abstract class / interface

may have instance fields

abstract class / concrete class

may inherit from a class

abstract class / concrete class

may implement an interface

abstract class / concrete class

Inheritance mysteries. **Explain and show your work.**

Assume that the following classes have been defined:

```
public class Denny extends John {
    public void method1() {
        System.out.print("denny 1 ");
    }
    public String toString() {
        return "denny " + super.toString();
    }
}
public class Cass {
    public void method1() {
        System.out.print("cass 1 ");
    }
    public void method2() {
        System.out.print("cass 2 ");
    }
    public String toString() {
        return "cass";
    }
}
public class Michelle extends John {
    public void method1() {
        System.out.print("michelle 1 ");
    }
}
public class John extends Cass {
    public void method2() {
        method1();
        System.out.print("john 2 ");
    }
    public String toString() {
        return "john";
    }
}
```

Given the classes above, what output is produced by the following code?

```
Cass[] elements = {new Cass(), new Denny(), new John(), new Michelle()};
for (int i = 0; i < elements.length; i++) {
    elements[i].method1();
    System.out.println();
    elements[i].method2();
    System.out.println();
    System.out.println(elements[i]);
    System.out.println();
}
```

cass 1
cass 2
cass

denny 1
denny 1 john 2
denny john

cass 1
cass 1 john 2
john

michelle 1
michelle 1 john 2
john

Assume that the following classes have been defined:

```
public class Tulip extends Rose {
    public void method1() {
        System.out.print("Tulip 1
");
    }
}

public class Violet {
    public void method1() {
        System.out.print("Violet 1
");
    }

    public void method2() {
        System.out.print("Violet 2
");
    }

    public String toString() {
        return "Violet";
    }
}
```

```
public class Rose extends Lily {
    public String toString() {
        return "Rose " +
super.toString();
    }
}

public class Lily extends Violet {
    public void method1() {
        super.method1();
        System.out.print("Lily 1 ");
    }

    public void method2() {
        System.out.print("Lily 2 ");
        method1();
    }

    public String toString() {
        return "Lily";
    }
}
```

Given the classes above, what output is produced by the following code?

```
Violet[] pretty = { new Tulip(), new Lily(), new Violet(), new Rose() };
for (int i = 0; i < pretty.length; i++) {
    System.out.println(pretty[i]);
    pretty[i].method1();
    System.out.println();
    pretty[i].method2();
    System.out.println();
    System.out.println();
}
```

Rose Lily
Tulip 1
Lily 2 Tulip 1

Lily
Violet 1 Lily 1
Lily 2 Violet 1 Lily 1

Violet
Violet 1
Violet 2

Rose Lily
Violet 1 Lily 1
Lily 2 Violet 1 Lily 1

Consider the following classes:

```
public class Foo {
    public void method1() {
        System.out.println("foo 1");
        method2();
    }

    public void method2() {
        System.out.println("foo 2");
    }
}

public class Bar extends Foo {
    public void method2() {
        System.out.println("bar 2");
    }

    public void method3() {
        System.out.println("bar 3");
    }
}

public class Baz extends Foo {
    public void method1() {
        System.out.println("baz 1");
    }

    public void method2() {
        System.out.println("baz 2");
        method1();
    }
}

public class Mumble extends Baz {
    public void method1() {
        super.method1();
        System.out.println("mumble 1");
    }

    public void method3() {
        System.out.println("mumble 3");
    }
}
```

Suppose the following variables are defined:

```
Foo var1 = new Bar();
Foo var2 = new Mumble();
Bar var3 = new Bar();
Baz var4 = new Baz();
Baz var5 = new Mumble();
Object var6 = new Baz();
```

Indicate on each line below the output produced by each statement shown. If the statement produces more than one line of output indicate the line breaks with slashes as in `a/b/c` to indicate three lines of output with `a` followed by `b` followed by `c`. If the statement causes an error, write the word `error` to indicate this.

`var1.method1();`	`foo 1/bar 2`
`var2.method1();`	`baz 1/mumble 1`
`var3.method1();`	`foo 1/bar 2`
`var4.method1();`	`baz 1`
`var5.method1();`	`baz 1/mumble 1`
`var6.method1();`	`error`
`var1.method2();`	`bar 2`
`var2.method2();`	`baz 2/baz 1/mumble 1`
`var3.method2();`	`bar 2`
`var4.method2();`	`baz 2/baz 1`
`var5.method2();`	`baz 2/baz 1/mumble 1`
`var6.method2();`	`error`
`var3.method3();`	`bar 3`
`var5.method3();`	`error`
`((Bar) var1).method3();`	`bar 3`
`((Mumble) var4).method3();`	`error`
`((Mumble) var5).method3();`	`mumble 3`
`((Bar) var2).method3();`	`error`
`((Baz) var2).method2();`	`baz 2/baz 1/mumble 1`
`((Mumble) var6).method2();`	`error`

1D arrays.

Write a method weave that takes two arrays of ints, a and b, and that returns an array that contains the elements of a and b in the order

a[0], b[0], a[1], b[1], etc.

If one of the arrays a or b is longer than the other, just add the extra elements at the end of the array.

For example, consider the arrays
 a: [10, 20, 30], and b : [1, 2, 3, 4, 5, 6]
The call weave(a, b) would return the array
[10, 1, 20, 2, 30, 3, 4, 5, 6]
As you can see, the array returned by the method contains the elements of a and b in the order a[0], b[0], a[1], b[1], etc. Since b is longer the extra elements of b are added at the end of the array.

In your solution, you can use only 3 arrays, namely the two arrays a, and b passed to the method and the array returned by the method.

You can’t use any ArrayList, String, or Collection object.

You may assume that none of the arrays passed to the method are null, though one of them or both of them may be empty. If both arrays passed to the method are empty, just return an empty array.

Write your solution on the next page.

```java
public static int[] weave(int[] a, int[] b) {
    // weave a and b in c
    int[] c = new int[a.length + b.length];

    // loop over the length of the shorter array
    int len = Math.min(a.length, b.length);
    int k = 0; // current index in c
    for (int i = 0; i < len; i++) {
        // take an element in a
        c[k++] = a[i];
        // take an element in b
        c[k++] = b[i];
    }

    // add what is left in the longer array (at most one of the two loops
    // is executed)
    // add what is left in a?
    for (int i = len; i < a.length; i++) {
        c[k++] = a[i];
    }
    // or in b?
    for (int i = len; i < b.length; i++) {
        c[k++] = b[i];
    }

    return c;
}
```

Write a method named `swapAll` that accepts two arrays of integers as parameters and swaps their entire contents. You may assume that the arrays passed are not null and are the same length.

For example, if the following arrays are passed:

```
int[] a1 = {11, 42, -5, 27, 0, 89};
int[] a2 = {10, 20, 30, 40, 50, 60};
swapAll(a1, a2);
```

After the call, the arrays should store the following elements:

```
a1: {10, 20, 30, 40, 50, 60}
a2: {11, 42, -5, 27, 0, 89}
```

```java
public static void swapAll(int[] a1, int[] a2) {
    // Note that
    // int[] temp = a1;
    // a1 = a2;
    // a2 = temp;
    // doesn't work
    // Only the references to the arrays are swapped
    // and not their elements
    for (int i = 0; i < a1.length; i ++) {
        int temp = a1[i];
        a1[i] = a2[i];
        a2[i] = temp;
    }
}
```

Inheritance

Suppose a class `GroceryBill` keeps track of a list of items being purchased at a market:

Method / Constructor	Description
`public GroceryBill(Employee clerk)`	constructs a `GroceryBill` object for the given *`clerk`*
`public void add(Item i)`	adds *`i`* to this bill's total
`public double getTotal()`	returns the cost of these items
`public void printReceipt()`	prints a list of items

`GroceryBill` objects interact with `Item` objects. An `Item` has the following public methods:

Method / Constructor	Description
`public double getPrice()`	returns the price for this item
`public double getDiscount()`	returns the discount for this item

For example, a candy bar item might cost `1.35` with a discount of `0.25` for preferred customers, meaning that preferred customers get it for `1.10`. (Some items will have no discount, `0.0`.) Currently the above classes do not consider discounts. Every item in a bill is charged full price, and item discounts are ignored.

Define a class `DiscountBill` that extends `GroceryBill` to compute discounts for preferred customers. The constructor for `DiscountBill` accepts a parameter for whether the customer should get the discount.

Your class should adjust the amount reported by `getTotal` for preferred customers. For example, if the total would have been `$80` but a preferred customer is getting `$20` in discounts, then `getTotal` should report the total as `$60` for that customer. You should also keep track of how many items a customer is getting a non-zero discount for and the overall discount, both as a total amount and as a percentage of the original bill. Include the extra methods below that allow a client to ask about the discount:

Method / Constructor	Description
`public DiscountBill(Employee clerk, boolean preferred)`	constructs discount bill for given *`clerk`*
`public int getDiscountCount()`	returns the number of items that were discounted, if any

`public double getDiscountAmount()`	returns the total discount for this list of items, if any
`public double getDiscountPercent()`	returns the percent of the total discount as a percent of what the total would have been otherwise

If the customer is not a preferred customer the `DiscountBill` behaves at all times as if there is a total discount of 0.0 and no items have been discounted.

```java
public class DiscountBill extends GroceryBill {

    private boolean preferred;
    private int discountCount;
    private double discountAmount;

    public DiscountBill(Employee clerk, boolean preferred) {
        super(clerk);
        this.preferred = preferred;
    }

    public void add(Item i) {
        super.add(i);
        if (preferred && i.getDiscount() > 0) {
            discountCount ++;
            discountAmount += i.getDiscount();
        }
    }

    public int getDiscountCount() {
        return discountCount;
    }

    public double getDiscountAmount() {
        return discountAmount;
    }

    public double getTotal() {
        return super.getTotal() - discountAmount;
    }

    public double getDiscountPercent() {
        return discountAmount / super.getTotal() * 100;
    }
}
```

1D and 2D arrays

Write a method that takes an array of ints a and a positive int n and that returns a 2D rectangular array that has n rows and that contains the elements of a.

The 2D array is filled with the elements of a taken in the order of increasing index (a[0], a[1], a[2], etc.) such that the columns are alternately filled top down and bottom up. That is the first column is filled from top to bottom, the second column is filled from bottom to top, the third column is filled from to top to bottom, and so on. Any unused element in the 2D array is set to 0.

For example if a contains the elements
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15}
and n is 4
the method should return the 2D array

1	8	9	0
2	7	10	15
3	6	11	14
4	5	12	13

```
public static int[][] oneDtoTwoD(int[] a) {
    // if the 2d array is nxn then nxn >= a.length
    // that is n = Math.ceil(Math.sqrt(a.length));
    int n = (int) Math.ceil(Math.sqrt(a.length));
    int[][] b = new int[n][n];

    // loop over the columns of b
    // and add the elements of a
    // as long as there are elements of a to add
    int col = 0; // column index in b
    int row = 0; // row index in b
    int rowUpdate = 1; // +/-1 added to row (initially +1)
    for (int i = 0; i < a.length; i ++) {
        b[row][col] = a[i];
        row += rowUpdate;
        if (row == n) {
            // reach the bottom of a column
            // go up the next one
            col ++;
            row --;
            rowUpdate = -1;
        } else if (row == -1) {
            // reach the top of a column
            // go down the next one
            col ++;
            row ++;
            rowUpdate = 1;
        }

    }
    return b;
}
```