

1) Implement method

```
public static Cat[][] upsideDown(Cat[][] source){
    int numRows = source.length;
    Cat[][] temp = new Cat[numRows][];
    int currRow = numRows-1;
    for (int row = 0; row < numRows; row++, currRow--){
        int numCols = source[row].length;
        temp[currRow] = new Cat[numCols];
        int currCol = numCols-1;
        for (int col = 0; col < numCols; col++, currCol--){
            temp[currRow][currCol] = new Cat(source[row][col]);
        }
    }
    return temp;
}
```

2) Determine the output

a) `f("Hello");`

A

C

D

G

b) `f("");`

B

C

D

G

c) `f(null);`

C

E

G

3) Define a *switch*

```
switch (value){
    case 3:
        str = "x";
        break;
    case 0 :
    case 1:
    case 2:
    case 4:
        str = "A";
        break;
    default:
        str = "B";
}
```

break;

}

4) Give results:

- a. Mammal p = (Mammal) c; Works Fine
- b. Dog q = (Dog) a; Compiles, but Throws an Exception
- c. Mammal r = c; Works Fine
- d. Dog s = a; Will NOT compile
- e. Dog t = (Dog) x; Works Fine
- f. Dog u = x; Will NOT compile
- g. For this part, assume that the Dog class has an instance method called "bark", which is not implemented in the Mammal class. x.bark(); Will NOT compile
- h. Suppose you add an instance method to the Mammal class called grow() and that you *override* that method in the Dog class. Consider the code below. **Which grow method will be called?**
x.grow(); CIRCLE ONE: ~~Mammal~~ / (Dog)

5) Indicate what is on the stack and what is on the heap after each of the following statements (draw a picture):

```
Integer x = new Integer(5);
// x is on the stack pointing at an integer object that contains a 5
int a = 7;
// a is put on the stack with the value 7 on the stack
Integer[] b = {new Integer(1), new Integer(5)};
// b is put on the stack with a reference to a 2 element array on the heap
// the first element of that array is a reference to another space on the heap
// that is an Integer that contains a 1
// the second element of that array is a reference to another space on the heap
// that is an Integer that contains a 5
Integer[] c = b;
// c is put on the stack - it is a reference pointing to the same space on the heap
// as where b is pointing to
Integer[] d = new Integer[b.length];
// d is put on the stack pointing to a 2 element array (distinct from b's)
// the elements of the array are null
for (int i = 0; i < b.length; i++) {
    d[i] = b[i];
}
// the elements of the array pointed to by d now point to the elements of the
// array pointed to by b
// they are distinct arrays but share the values
int[] e = {8, 9};
// e comes on to the stack pointing to a two element array on the heap
// the 8 and 9 are directly in that array (not pointed to by it)
```

6) Assume you have the class MyClass as defined below.

```
public class MyClass{
    private Obj1[][] list1;
    private ArrayList<Obj2> list2;
```

a) Write a copy constructor for the MyClass.

```
public MyClass(MyClass other){
    list1 = new Obj1[other.list1.length][];
    for (int row = 0; row < other.list1.length; row++){
        list1[row] = new Obj1[other.list1[row].length];
        for (int col = 0; col < other.list1[row].length; col++){
            list1[row][col] = new Obj1(other.list1[row][col]);
        }
    }
}
```

```

        list2 = new ArrayList<Obj2>();
        for (Obj2 temp: other.list2){
            list2.add(new Obj2(temp));
        }
    }
}

b) Write a equals for the MyClass (two MyClass objects are considered equal if the contents of the array is
exactly the same and in the order and the objects of the arrayLists are exactly the same and in the same
order).
public boolean equals(Object other){
    if (other == null ){
        return false;
    } else if (other.getClass() != getClass()){
        return false;
    } else{
        MyClass local = (MyClass) other;
        if (list1.length != other.list1.length || list2.size() != other.list2.size()){
            return false;
        } else {
            for (int row = 0; row < list1.length; row++){
                if (list1[row].length != other.list1[row].length){
                    return false;
                }
                for (int col = 0; col < list1[row].length; col++){
                    if (!(list1[row][col].equals(other.list1[row][col]))){
                        return false;
                    }
                }
            }
            for (int cur = 0; cur < list2.size(); cur++){
                if (!(list2.get(cur).equals(other.list2.get(cur)))){
                    return false;
                }
            }
        }
        return true;
    }
}

c) Write a method which returns the largest int (based on the individual getValues in the array (list1) (call it
largestInArray). Return a -1 if there is nothing in the array.
public int largestInArray(){
    if (list1.length == 0){
        return -1;
    }
    int largestSoFar = -1;
    for (int row =0; row < list1.length; row++){
        for (int col = 0; col < list1[row].length; col++){
            if (list1[row][col].getValue() > largestSoFar){
                largestSoFar = list1[row][col].getValue();
            }
        }
    }
    return largestSoFar;
}

```

- d) Write a method which returns the largest int (based on the individual `getValues`) in the ArrayList (`list2`) (call it `largestInList`). Return a -1 if there is nothing in the ArrayList.

```
public int largestInList(){
    if (list2.size() == 0){
        return -1;
    }
    int largestSoFar = -1;
    for (Obj2 temp: list2){
        if (temp.getValue() > largestSoFar){
            largestSoFar = temp.getValue();
        }
    }
    return largestSoFar;
}
```

- e) Write a method that uses the two previous methods to return a single integer which is the largest of the whole class. Returns a -1 if and only if they are both empty.

```
public int largestWhole(){
    return (largestInArray() > largestInList() ? largestInArray():largestInList());
}
```

- f) Write a sum method that returns the sum of all `getValues` (in both data members)

```
public int sumAll(){
    int sum = 0;
    for (int row = 0; row < list1.length; row++){
        for (int col = 0; col < list1[row].length; col++){
            sum += list1[row][col].getValue();
        }
    }
    for (Obj2 temp: list2){
        sum += temp.getValue();
    }
    return sum;
}
```

- g) Write a `toString` method that returns the first value (the `[0][0]` element) of the array followed by the first item in the ArrayList. If either is empty (there are no objects inside) return the string "empty".

```
public String toString(){
    if (list2.size() == 0){
        return "empty";
    }
    if (list1.length == 0){
        return "empty";
    }
    boolean allEmpty = true;
    for (Obj1[] row: list1){
        if (row.length != 0){
            allEmpty = false;
        }
    }
    if (allEmpty){
        return "empty";
    }
    if (list1[0].length == 0){
```

```

        return "empty";
    }
    return (list1[0][0] + " & " + list2.get(0));
}

```

- h) Write a method that returns a Stack of all objects on the array (order does not matter) but all items present in the array must be on a Stack that is returned as the return value of the method.

```

public Stack<Obj1> stackIt(){
    Stack<Obj1> temp = new Stack<Obj1>();
    for (int row = 0; row < other.list1.length; row++){
        for (int col = 0; col < other.list1[row].length; col++){
            temp.push(new Obj1(list1[row][col]));
        }
    }
    return temp;
}

```

- i) Write a method that uses a for loop to increaseValue of all of the objects in list1. Rewrite using for-each.

```

public void increaseArrFirstWay(){
    for (int row = 0; row < other.list1.length; row++){
        for (int col = 0; col < other.list1[row].length; col++){
            list1[row][col].increaseValue(5);
        }
    }
}
public void increaseArrSecondWay(){
    for (Obj1[] row : list1){
        for (Obj1 temp: row){
            temp.increaseValue(5);
        }
    }
}

```

- j) Write a method that uses a for loop to increaseValue of all of the objects in list2. Rewrite with for/each.

```

public void increaseListFirstWay(){
    for (int i = 0; i < list2.size(); i++){
        list2.get(i).increaseValue(5);
    }
}
public void increaseListSecondWay(){
    for (Obj2 temp: list2){
        temp.increaseValue(5);
    }
}

```

- 7) Answer each of the following:

- 2 raised to the 6th
- CPU = Central Processing Unit, RAM = Random Access Memory, API = Application Programmers Interface, GUI = Graphical User Interface, IDE = Integrated Development Environment, JVM = Java Virtual Machine
- Explain the purpose of each of the following: junit testing, the debugger, eclipse, JVM, memory: stack&heap – look these up or come see me – too much to type here
- Explain the purpose of each of the following key words: public, private, final, finally, void, static, this, super – look these up or come see me – too much to type here

- e) alias
- f) polymorphic
- g) 24Temp (_9_2) break (While) dog@99
- h) ordered highest to lowest
 - (decrement)
 - / (division)
 - (subtraction)
 - > (greater than)
 - && (logical AND)
 - = (assignment)
- i) In Java, a class can directly extend more than one class simultaneously. TRUE / (FALSE)
- j) In Java, a class can directly implement more than one interface simultaneously. (TRUE) / FALSE
- k) ArrayList<String> temp = new ArrayList<String>();
- l) temp.add("Jan"); temp.add("Mike"); temp.add("Angel"); temp.add("Jon");
- m) Object
- n) (TRUE)/FALSE
- o) TRUE/(FALSE)
- p) any instance method has a this – static methods do not
- q) Answer the following about selecting methods
 - a. early (static) binding
 - b. late (dynamic) binding
 - c. late (dynamic) binding
- r) StringBuffer

8) Implement the method below. Assume you are in the Cat class (and the Cat has a String as its name, a double as the weight).

- a. Write a compareTo method which returns a negative value if the current object is less in weight than the parameter, a positive value if the current object is heavier than the parameter, and zero if they are close enough in weight to be equal. You may assume there is a EPSILON constant already defined that tells you what is "close enough" to the same to be called equal – they should be considered equal if they are within that EPSILON of each other in weight.

```
public int compareTo(Cat other){
    double weightDiff = Math.abs(weight – other.weight);
    if (weightDiff < EPSILON){
        return 0;
    } else if (weight < other.weight){
        return -1;
    } else {
        return 1;
    }
}
```

- b. Write a toString method for the Cat class which gives the name followed by the weight (with the weight in parentheses). Make sure the weight is printed with no more than two places after the decimal point.

```
public String toString(){
    int weightInt = (int)(weight * 100 + 0.5);
    //multiplying by 100 to get to 2 digits beyond decimal before truncating
    // adding the .5 after the multiply to round the 3rd digit after the decimal
```

```

        String temp = name + "(";
        temp += weightInt / 100;
        temp += ".";
        temp += weightInt % 100;
        temp += ")";
        return temp;
    }

```

9) Implement the method

```

public static void specialPrint() {
    for (int i = 1; i <= 100; i++){
        if (i %2 == 1 && i > 50){
            System.out.println("oddbig");
        } else if (i %2 == 1){
            System.out.println("odd");
        } else if (i > 50){
            System.out.println("big");
        } else{
            System.out.println(i);
        }
    }
}

```

10) Consider the class **Fruit** and interface **hasSeeds**, below.

```

public class Fruit {
    private String fruitType;
    public Fruit(String fruitType) {
        this.fruitType = fruitType;
    }
    public Fruit(Fruit x) {
        fruitType = x.fruitType;
    }
    public String toString(){
        return ("*" + fruitType + "*");
    }
}

public interface hasSeeds {
    public int getNumSeeds();
}

class Apple extends Fruit implements hasSeeds {
    private int numSeeds;
    public Apple(int s) {
        super("Apple");
        numSeeds = s;
    }
    public Apple() {
        this(13);
    }
    public Apple(Apple a) {
        super(a);
        numSeeds = a.numSeeds;
    }
}

```

```

    }
    public boolean equals(Object x) {
        if (x == null) {
            return false;
        }
        if (x.getClass() != getClass())
            return false;
        Apple a = (Apple)x;
        return numSeeds == a.numSeeds;
    }
    public String toString() {
        return super.toString() + " with " + numseeds + " seeds";
    }
    public int getNumSeeds() {
        return numSeeds;
    }
}

```