

Final Exam Practice

Will be discussed Monday, May 9 in lab section – answers will be distributed at that time

- 1) Implement the method **upsideDownBackwards**, with the prototype listed below. This method returns a **deep copy** of the source array, but with the rows in upside-down order and the columns are in reverse order. (So the first row becomes the last row; the second row becomes the next-to-last row, etc. For the elements of an individual row, the last becomes the first and the first becomes the last, etc.) The source array might be a ragged array and any array could be size 0 but none will be null. (Remember, *ragged* means that the number of columns may not be the same for every row.) You may assume that there is a Cat copy constructor available.

```
public static Cat[][] upsideDownBackwards(Cat[][] source){
```

- 2) Determine the output for the following program – assuming the line in the middle is replaced with each of the three options – one of these options is used in the run, then replace with another and run again, etc.:

```
private static void f(String a) {
    try {
        int x = 7 / a.length();
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}

public static void main(String[] args) {
    try {
        /*OPTIONS(Select one of these statements for this line): */
        // f("Hello");      f("");      f(null);
        System.out.println("D");
    } catch (NullPointerException e) {
        System.out.println("E");
    } catch (ArithmeticException e) {
        System.out.println("F");
    }
    System.out.println("G");
}
```

- 3) Consider the following code fragment. Assume that the variable named *value* is of type *int* and *str* of type *String*.

Define a *switch* statement so that will have a result that is equivalent in final result to the code fragment above. You may not write any statements before or after the switch statement. Your switch statement should not have any unnecessary redundancy. **You may not have any if/else statements or any ternary ?: operators in your switch statement.**

```
if (value > -2) {
    str = "A";
} else if (value < 0){
    str = "B";
} else{
    str = "C";
}
if (value > 4) {
    str = "B";
} else if (value == 3){
    str = "x";
} else if (value == 6){
    str = "m";
} else if (value == -6){
    str = "w";
}
```

- 4) Assume that there is a class called Mammal which has been extended to a subclass called Dog, and that the following statements have been executed. For each write "works fine" or "will not compile" or "compiles but throws and exception"

```
Mammal a = new Mammal();
```

```
Dog c = new Dog();
```

```
Mammal x = new Dog();
```

For each code fragment below, answer each with: Works Fine, Compiles but Throws an Exception, Won't compile

- ```
Mammal p = (Mammal) c;
```
- ```
Dog q = (Dog) a;
```
- ```
Mammal r = c;
```
- ```
Dog s = a;
```
- ```
Dog t = (Dog) x;
```
- ```
Dog u = x;
```
- For this part, assume that the Dog class has an instance method called "bark", which is not implemented in the Mammal class.

```
x.bark();
```
- 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);  
int a = 7;  
Integer[] b = {new Integer(1), new Integer(5)};  
Integer[] c = b;  
Integer[] d = new Integer[b.length];  
for (int i = 0; i < b.length; i++) {  
    d[i] = b[i];  
}  
int[] e = {8, 9};
```

- 6) Assume you have the class MyClass as defined below. Assume every type of class mentioned here is mutable. Write all methods mentioned. You may assume Obj1 and Obj2 classes have good copy constructors, good toString methods, good equals methods, and each has a getValue method that returns a positive int, and each has an increaseValue method that increases the value of the item by the int amount passed as the parameter.

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

- Write a copy constructor for the MyClass.
- Write an equals method 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).
- 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.
- 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.
- 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.
- Write a sum method that returns the sum of all getValues (in both data members)
- 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".
- 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.
- Write a method that uses a for loop to increaseValue of all of the objects in list1. Rewrite with for/each.

- j) Write a method that uses a for loop to increaseValue of all of the objects in list2. Rewrite with for/each.
- 7) Answer each of the following:
- How many distinct sequences of 0's and 1's could be represented using all of a series of 6 bits?
 - What do the following acronyms stand for: CPU, RAM, API, GUI, IDE, JVM
 - Explain the purpose of each of the following: junit testing, the debugger, eclipse, JVM, memory: stack&heap
 - Explain the purpose of each of the following key words: public, private, final, finally, void, static, this, super
 - What do you call the situation where two different variables refer to the same object?
 - What do you call the type of variable that can at different moments in time reference different types of objects?
 - Which of the following are valid **Java identifiers** that could be used to name methods or variables? (Circle all the valid ones.)
 24Temp _9_2 break While dog@99
 - List the following operators in order of precedence. (Put the operator with highest precedence at the top; lowest at the bottom.)

> (greater than)	List the operators here: _____
/ (division)	_____
-- (decrement)	_____
- (subtraction)	_____
= (assignment)	_____
&& (logical AND)	_____
 - In Java, a class can directly extend more than one class simultaneously. TRUE / FALSE
 - In Java, a class can directly implement more than one interface simultaneously. TRUE / FALSE
 - Write a statement that declares a variable that could be used to refer to an ArrayList that will contain String references.
 - Write statements to put all of the names of your immediate family into that arrayList.
 - What class is at the top of every inheritance diagram in Java? (The one class that all other classes extend either directly or indirectly.)
 - You can access a static variable in a class even if no objects of that class have been instantiated. TRUE/FALSE
 - You can access an instance variable in a class even if no objects of that class have been instantiated. T/F
 - In which kind of methods can you use the Java keyword "this"? (and where can't you use it?)
 - Answer the following about selecting methods
 - In some languages, when you override a method the *compiler* decides whether the base-class version or the derived-class version will run. What kind of binding is this?
 - In other languages, the decision of which version of the overridden method to call is made at run-time, depending on the class of the object on the heap. What kind of binding is this?
 - What kind of binding does Java use?
 - If you need to continuously re-size a string, what class should you use instead of the String class?
- 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).
- 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.
 - 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.

- 9) Implement the method below. The method will print out all the numbers from 1 to 100 with one value on each line, *but*: for numbers that are **odd** it will print “odd” instead of the number; for numbers that are **greater than 50** it will print “big” instead of the number; and for numbers that are **both odd and greater than 50** it will print “oddbig” instead of the number.

```
public static void specialPrint() {
```

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

```
public class Fruit {
    private String fruitType;
    public Fruit(String fruitType) {
        this.type = type;
    }
    public Fruit(Fruit x) {
        type = x.fruitType;
    }
    public String toString(){
        return ("*" + fruitType + "*");
    }
}
public interface hasSeeds {
    public int getNumSeeds();
}
```

Implement the complete class called **Apple** that extends the class `Fruit` and implements the `hasSeeds` interface. It has a single private instance variable of type `int`, called `numSeeds`, which represents the number of seeds in the Apple. Your class should contain the following public methods.

- Apple Constructor with one int parameter:** This constructor takes one `int` argument, which represents the number of seeds in the Apple. It will initialize the field “`numSeeds`” to be equal to the parameter, and the field `fruitType` will be set equal to the String “Apple”.
- Apple Constructor with no parameters:** This constructor will initialize an `Apple` with 13 seeds and `fruitType` equal to “Apple”. **You must call the constructor from part a in your implementation of this constructor.**
- Apple Copy Constructor.** **You must call the copy constructor of the `Fruit` class while implementing this one.**
- Apple equals:** You must follow the example of a “well-designed equals method” given in the lecture. As we are defining it, to be equal means that both objects must be of the same class (recall the method `getClass`) and both objects must have the same number of seeds.
- Apple toString:** Write a `toString` method for this class which returns a String containing both the name and the number of seeds in some reasonable format.
- Possibly other method(s):** Implement any method(s) that are required because of the fact that this class extends `Fruit` and/or implements `hasSeeds`. **Note: You will lose points if you implement methods that are not necessary.** If you do implement some method(s) here, use simple and reasonable implementation(s).