Choose 4 of the 1st 5 questions (#'s 1 through 5) to complete. **Use the remaining question as extra credit** (for a max of 5 points). **Specify which question is extra credit by writing EXTRA CREDIT** in caps above the question. If you do not, I will count the last question (#5) as extra credit.

1. (10 points) What is the **output** of the f**ollowing code**?

Code	**Output**

`// assume that a class called`
`// StackofChars exists and that it`
`// supports standard stack operations`
`// such as push, pop, empty, and getSize`
 	
`public static void main(String[] args) {`
`  syso(mystery_method("BA", 'B', 'A'));`
`  syso(mystery_method("AC", 'B', 'A'));`
`  syso(mystery_method("ACBACB", 'B', 'A'));`
`  syso(mystery_method("BABCA", 'B', 'A'));`
`  syso(mystery_method("AACCBB", 'B', 'A'));`
`}`
 	
`public static boolean mystery_method(String s,`
`    char start, char end) {`
`  StackOfCharacters stack = new StackOfChars();`
`  for(int i = 0; i < s.length(); i++) {`
`    char ch = s.charAt(i);`
`    if (ch == start) {`
`      stack.push(ch);`
`    } else if (ch == end) {`
`      if(stack.empty()) {`
`        return false;`
`      }`
`      stack.pop();`
`    }`
`  }`
`  return stack.getSize() == 0;`
`}`
	`true`
	`false`
	`false`
	`true`
	`false`

`// Show work for potential partial credit`
`public static void main(String[] args) {`
`  char[][] arr = {`
`      {'A', 'C'},`
`      {'B', 'D'}`
`  };`
`  char[][] newArr = new char[2][2];`
`  int newRow = 0;`
`  int newCol = 0;`
`  for(int i = arr.length - 1; i >= 0; i--) {`
`    for(int j = 0; j < arr[i].length; j++) {`
`      newArr[newRow][newCol] = arr[i][j];`
`      newRow++;`
`    }`
`    newCol++;`
`    newRow = 0;`
`  }`
`  for(char[] subArr: newArr) {`
`    syso(Arrays.toString(subArr));`
`  }`
`}`
	`[B, A]`
	`[D, C]`

2. (10 points total) **Short Answer. (No really, short – just one or two sentences each)**

   (a) (1 point) Write a **short** code sample that demonstrates a situation where you **have to** use the keyword, **`this`**.

```
public Foo(String name) {
  this.name = name;
}
```

   (b) (3 points) Write the **appropriate visibility modifier** for the method, `foo`, below. The comment describes where the method should be visible. Write "blank" if a visibility modifier is not required:

private – only accessible within class the method or variable is defined in
protected – accessible by subclasses
default (no modifier) – accessible by classes in same package
public – accessible by all other classes

   (c) (2 points) What is the output of the code below? If the result is an error or no output, explain why.

```
public class AddNumbers {
  public static void main(String[] args){
    int res = add(5);
  }
  public static int add(int x) {
    int y;
    return x + y;
  }
}
// OUTPUT: error, uninitialized y
```

```
class MyPoint {
  int x; int y;
}
public class TestMyPoint {
  public static void main(String[] args){
    MyPoint p = new MyPoint();
    System.out.println(p.x);
  }
}
//OUTPUT: 0
```

   (d) (1 point) **When** would you choose to use **static** variables or methods over **instance** variables or methods?

Use instance variables if you want every instance to have its own version of the variable. Use instance methods when you need to work on instance variables. Use static variables and/or methods if you can't guarantee or don't need an instance to be present for your variable or method to be used or if you want a single copy of a variable shared across all instances (generally, though, default to using instance variables and methods)

   (e) (3 points) **What is the output** of the following program... and **why**?

In LotteryTicket.java

```
public class LotteryTicket {
  int[] numbers;

  LotteryTicket(int[] nums) {
    numbers = nums;
  }
}
```

In main method of LotteryTicketTest.java

```
public static void main(String[] args) {
  int[] picks = {4, 2, 3, 2};

  LotteryTicket  t =
    new LotteryTicket(picks);

  syso(Arrays.toString(picks));
  syso(Arrays.toString(t.numbers));
  picks[3] = 42;
  syso(Arrays.toString(picks));
  syso(Arrays.toString(t.numbers));
}
```

```
[4, 2, 3, 2]
[4, 2, 3, 2]
[4, 2, 3, 42]
[4, 2, 3, 42]
```

The member variable, `numbers`, has a references to the same `Array` as the reference in the argument passed in, `picks`.

3. (10 points) **Multiple choice**. **There can be more than one correct answer** in some cases. Circle **all correct answers**. Here's an example question: circle all valid logical operators (there are 2!)...
i. && ii. || iii. ** iv. %%

(a) Which of the following types are **not** primitives (that is, which are reference types)?

i. `int` ii. `String` iii. `char` iv. *any type of* `Array`

(b) What **method(s)** could be used in the blank space below to **create an `Array` from the string, `dashed`**, using the separator, `"-"`?

```
String dashed = "lots-of-dashes";
String[] words = dashed._____; // --> {"lots", "of", "dashes"}
```

i. `toArray("-")` ii. `explode()` iii. `split("-")` iv. `split()`

(c) Based on the variables defined below, circle all of the expressions on the right that evaluate to true:

```
String s1 = new String("cheese");
String s2 = new String("cheese");
String s3 = "cheese";
String s4 = "cheese";
```

i. `s1 == s2`
ii. `s3 == s4`
iii. `s1 == s3`
iv. `s1.equals(s2)`
v. `s3.equals(s4)`
vi. `s1.compareTo(s3) > 0`

(d) A **constructor** is **different** from a **regular method** because:

i. constructors must be named the same name as the class they are in
ii. constructors do not have a specified return type
iii. at least one explicitly defined constructor must be present per class
iv. constructors cannot be overloaded

(e) Which of the following is true about binary search:

i. the Array being searched does not have to be ordered for binary search to work
ii. binary search may not always outperform linear search
iii. if the element being searched for is less than the element at the midpoint, then the new start index is the midpoint index - 1

(f) The default value for a member variable of type _____ is _____:

i. `boolean: true` ii. `int: -1` iii. `Array: {}` iv. `reference: null`

(g) A static method can access and/or invoke:

i. instance variables ii. instance methods iii. static variables iv. static methods

(h) The scope of static and instance variables (that is, where are these variables can be accessed) is:

i. only a class's methods ii. the entire class iii only a class's constructors

(i) Given the declaration `Circle[] x = new Circle[10]`, which of the following statements is **most accurate**? The variable, **`x`**, contains...
i. an array of ten int values.
ii. a reference to an array and each element in the array can hold a Circle object.
iii. an array of ten objects of the Circle type.
iv. a reference to an array and each element in the array can hold a reference to a Circle object.

4. (10 points) **Hunting for errors.** Circle 5 compile-time and/or run-time errors in the code below. Label them and write out why there is an error in the space underneath the code.

Foo.java

```
public class Foo {

  char label;
  private int id;
  static int magicNumber = 100;

  public Foo(int x) {
    id = x;
    label = 'z';
  }
  (a)
  public void setLabel(char label) {
    label = label;
  }

  public int getLabel1() {
    return label;
  }
  (b)
  public static int getLabel2() {
    return label;
  }

  public int getNumber() {
    return magicNumber;
  }
}
```

FooTest.java

```
public class FooTest {
  public static void main(
      String[] args) {
(c)
    Foo f = new Foo();

    syso(f.label);
(d) syso(f.id);
    syso(Foo.getLabel2());
(ef) syso(Foo.getNumber);
  }
}
```

(a) Must qualify with this (member variable, label, is hidden by the parameter with the same name)

(b) static methods can't access instance variables

(c) There's no constructor that matches the signature (and default constructor is not present)

(d) id is private and cannot be accessed

(e) instance method cannot be called on class

(f) (Also missing parentheses)

5. (10 points total) **Mash up.**

(a) (3 points) Circle True or False

i. True / False - A method that changes the value of a private variable is called an **mutator**

ii. True / False - Static variables are accessible from all methods of the same class.

iii. True / False - A local variable in a method can have the same name as a data field in the same class.

iv. True / False - the words "**object**" and "**instance**" can be used interchangeably

(b) (4 points) Coding Questions

i. Explain each part of `public static void main(String[] args)`

It's a method called main with a single argument, an Array of Strings. The method can be accessed by any other class, and it does not need an instance to be called. It does not return anything.

ii. Write a short class (~6 lines long!) called Tracker that keeps track of how many instances of the Tracker class were made. Here's how the class may be used:

```
Tracker t1 = new Tracker();
System.out.println(Tracker.numInstances);
Tracker t2 = new Tracker();
System.out.println(t1.numInstances);
// prints out 1... then 2
```

```
public class Tracker {
      static int numInstances = 0;
      Tracker() {
            numInstances++;
      }
}
```

(c) (3 points) Short Answer

i. Why does Java intern Strings?

To save space / memory usage (single copy rather than multiple!). Done with String literals – even when concatenated.

ii. In what conditions is linear search faster than binary search... and what conditions is binary search faster than linear search?

Linear search is faster if it's a small Array and / or the element is at the beginning of the Array. Binary search is faster for larger Arrays, and it assumes that the Array is already sorted.

iii. Why use the StringBuilder class over string concatenation?

String concatenation creates an entirely new string, StringBuilder just changes the current instance... so it uses up less memory / less space.

6. (20 points) **Finish the implementation of `StackOfCats`** below. (A `Cat` is defined by the class on the right). Implement `push`, `pop`, `empty`, and `peek`. The stack's capacity should be dynamic – it can grow to accommodate more `Cats`.

```
public class StackOfCats {

  private Cat[] cats;
  private int size = 0;

  StackOfCats(int capacity) {
    this.cats = new Cat[capacity];
  }
  // 1. implement push

  // 2. implement pop

  // 3. implement empty

  // 4. implement peek
  }
}
```

```
public class Cat {

  public String name;
  int lives = 9;

  Cat(String catName) {
    this.name = catName;
  }

}
```

```
public void push(Cat c) {
      if (size == elements.length) {
            // have to add for this since size starts out at 0
            Cat[] newCats = new Cat[cats.length + 5];
            for(int i = 0; i < cats.length; i++) {
                  cats[i] = cats[i];
            }
            elements = newElements;
      }
      elements[size] = c;
      size++;
}


public Cat pop() {
      size--;
      return cats[size];
}

public boolean empty() {
      return size == 0;
}

public Cat peek() {
      return cats[size - 1];
}
```

7. (20 points) **Robots! Fill in the blanks** in the left column and **implement the sort method** used in the right column.

Robot.java

```
public class Robot {

  public String name;

  // Add a variable called villain and
  // initialize it to false. Make it so
  // that it cannot be accessed directly,
  // and all instances share the same
  // copy (if it is changed, it changes
  // for all instances).

  _____________________________________

  Robot(String s) {this.name = s;}

  public String toString() {return name;}

  // Add a method called toggleVillain
  // that changes the variable above so
  // that it is the opposite of
  // whatever it currently is.

  _____________________________________

  _____________________________________

  _____________________________________
}
```

In RobotTest class of TestRobot.java

```
public static void main(
    String[] args) {
  Robot[] robots = new Robot[7];
  robots[0] = new Robot("Eve");
  robots[1] = new Robot("Alice");
  robots[2] = new Robot("Bob");
  robots[3] = new Robot("Chuck");
  robots[4] = new Robot("Mal");
  robots[5] = new Robot("Frank");
  robots[6] = new Robot("Carol");

  // Write the sort method used
  // below so that it sorts the
  // robots in alphabetical order
  // by name. You must use
  // selection sort.

  ___// define sort below_______

  ___// define sort below______

  sort(robots);
  syso(Arrays.toString(robots));

  // prints out [Alice, Bob,
  // Carol, Chuck, Eve, Frank,
  // Mallory]
}
```

(a) Fill in the blanks above, Define `sort` method below using selection sort

```
private static boolean villainMode = false;


public void toggleVillainMode() {
      villainMode = !villainMode;
}




public static void sort(Robot[] arr) {
      for(int i = 0; i < arr.length - 1; i++) {
            int mIndex = i;
            for(int j = i + 1; j < arr.length; j++) {
                  if(arr[mIndex].name.compareTo(arr[j].name) > 0) {
                        mIndex = j;
                  }
            }
            if (mIndex != i) {
                  Robot temp = arr[i];
                  arr[i] = arr[mIndex];
                  arr[mIndex] = temp;
            }
      }
}
```

8. (20 points) **Create two methods that process 2 dimensional Arrays:**

(a) `public static int sumDiagonals(int[][] arr)`

Calculate the sum of all digits in both diagonals in a 2-dimensional Array. Assume that the Array passed in will have an equal number of rows and columns (but that number can vary: 3x3, 4x4, etc).

Input Array	Method Call, Processing	Return Value

`int[][] arr = {{1, 2, 3},`
`{4, 5, 6},`
`{8, 8, 9}};`
	`sumDiagonals(int[][] arr)`
	
	`(1 + 5 + 9) + (3 + 5 + 8)`
		`31`

(b) `public static int[] unravel(int[][] arr)`

Take an incoming 2-dimensional Array and give back a new 1-dimensional Array. Do this by taking every column in the original 2-dimensional Array, starting with the left most column, and going from top to bottom in the column, add the elements to the new 1-dimensional Array.

Input Array	Method Call, Processing	Result
`int[][] arr = {{1, 3, 5},` `{2, 4, 6}};`	`unravel(int[][] arr)`	`// arr is...` `{1, 2, 3, 4, 5, 6}`

```
public static int sumDiagonals(int[][] arr) {
      int sumDiag1 = 0;
      int sumDiag2 = 0;
      for(int i = 0; i < arr.length; i++) {
            sumDiag1 += arr[i][i];
            sumDiag2 += arr[i][arr[i].length - 1 - i];
      }
      return sumDiag1 + sumDiag2;
}
// or nested loop and check if == and sum is == to length - 1
```

```
public static int[] unravel(int[][] arr) {
   int[] newArr = new int[arr[0].length * arr.length];
   int i = 0;
   for(int col = 0; col < arr[0].length; col++) {
         for(int row = 0; row < arr.length; row++) {
               newArr[i] = arr[row][col];
               i++;
         }
   }
   return newArr;
}
```

9. (20 points) Write a class, **`TicTacToe`**, that represents a 3 x 3 Tic Tac Toe board. X and O "marks" can be placed on the board, and the board can be printed to show all marks placed. **Implement the entire class**, and make sure it adheres to the following requirements and example usage.

(a) **Create a method that allows a mark to be placed on the board**. The method should be called **`place`**, and it should take a character as an argument, as well as a row and column that specifies which row and column to place the mark in. **If the row and column are not within the boundaries of the board, do not add a new mark to the board.**
 	
(b) **When a TicTacToe board object is printed out it should show all of the marks placed.** If no piece is in a row and column, then leave that position blank by printing out a space. Each row is bordered by vertical pipes: "|". **You must use a `StringBuilder` to as part of you implementation.**
	`// Example Usage`
	`TicTacToe t = new TicTacToe();`
	`t.place('X', 1, 1);`
	`t.place('O', 2, 2);`
	`t.place('X', 2, 1);`
	`t.place('O', 1, 0);`
	`t.place('X', 0, 1);`
	`t.place('O', 9, 9); // no mark`
	`syso(t);`
	
	`// Output of code above`
	`| X |`
	`|OX |`
	`| XO|`

```
public class TicTacToe {

	private char[][] board = new char[3][3];

	public void place(char piece, int row, int col) {
		if((piece == 'X' || piece =='O')
				&& row >= 0 && row <= this.board.length
				&& col >= 0 && col <= this.board[row].length) {
			this.board[row][col] = piece;
		}
	}

	public String toString() {
		StringBuilder sb = new StringBuilder();
		for(char[] row: this.board) {
			sb.append('|');
			for(char col: row) {
				if(col != 0) {
					sb.append(col);
				} else {
					sb.append(' ');
				}
			}
			sb.append("|\n");
		}
		return sb.toString();
	}
}
```

```
public static void reverseRowElements(int[][] arr) {
                    for(int i = 0; i < arr.length; i++) {
                          for(int j = 0; j < arr[i].length / 2; j++) {
                                int temp = arr[i][j];
                                arr[i][j] = arr[i][arr[i].length - 1 - j];
                                arr[i][arr[i].length - 1 - j] = temp;
                          }
                    }
      }
      // or counter from other end
```