CS 163 - Exam 3 Study Guide and Practice Exam

November 6, 2017

Summary

1 Disclaimer

2 Methods and Data
- 2.1 Static vs. Non-Static
  - 2.1.1 Static Example
  - 2.1.2 Non-Static Example
  - 2.1.3 Calling Static and Non-Static Methods
- 2.2 Pass-by-Value vs Pass-by-Reference

3 Objects
- 3.1 General Syntax
- 3.2 Example
- 3.3 Suggestions, Warnings, and Resources

4 ArrayLists
- 4.1 Suggestions, Warnings, and Resources

5 File I/O
- 5.1 Reading a File
- 5.2 Writing to a File
- 5.3 Suggestions, Warnings, and Resources

6 Interfaces

7 Practice Written Exam
- 7.1 Short Answer
- 7.2 Tracing

8 Programming Quiz Practice Exam

9 Suggestions for Studying and Test Taking
- 9.1 Written
- 9.2 Programming Quiz
- 9.3 Common Errors
- 9.4 Challenges

10 Answers to Practice Written and Programming Problems
- 10.1 Written
- 10.2 Tracing
- 10.3 Programming

1 Disclaimer

This is a review of the material so far, but there may be material on the exam not covered in this study guide.

2 Methods and Data

The general format of a method:

public / private static / ∅ returnType MethodName (parameterType parameterName, ...) / (∅) {

}

Note: The slash represents a choice (i.e. a method can be either public or private).
Note: ∅ means you don't include anything.

2.1 Static vs. Non-Static

Static methods belong to the class and only have one copy of the information. For example, a Clock class should be static, because if you change something on a clock you want it change in all other objects too. Static methods are used when you used when you aren't going to use instance variables.

Non-Static methods are instances of the class (belong to the object) and you can use instance variables within the method. For example, a Student class should be non-static because you want to have all of your objects be different (different name, id, major, etc)

Note on calling methods: The only time you need to create an object of the class (for example, `NonStaticExample ex = new NonStaticExample()`) is when you call a non-static method in a static method (a common call is in the main method, which is static). If you were to have two non-static methods or a non-static method calling a static method you wouldn't need to create an object.

2.1.1 Static Example

```
// Example of a static class: Clock

public class Clock {
    private static int hour, minute;

    public Clock (int h, int m){
        hour = h;
        minute = m;
    }
    // default time if no hour and minute are given
    public Clock (){
        hour = 12;
        minute = 0;
    }
    public static void increaseHour(int num){
        hour += num;
    }
    public static void increaseMinute(int num){
        minute += num;
    }
    // Calling above methods to save work
    public static void increaseHour() {
```

```
        increaseHour(1);
    }
    public static void increaseMinute(){
        increaseMinute(1);
    }
    public String toString(){
        return String.format("%02d:%02d", hour, minute); // returns time with 00:00 format.
    }
    public static void main (String [] args){
        Clock c1 = new Clock();
        System.out.println("c1 - Default new clock(should be 12:00): " + c1);
        c1.increaseMinute();
        System.out.println("c1 - adding a minute: " + c1);
        Clock c2 = new Clock (7, 15);
        System.out.println("c2 - created with time 7:15: " + c2);
        System.out.println("c1 - after creating c2: " + c1);
        c2.increaseHour(2);
        System.out.println("c2 - after adding 2 hours: " + c2);
        System.out.println("c1 - after c2 is incremented by 2: " + c1);
    }
}
```

The output from the above code:

```
c1 - Default new clock(should be 12:00): 12:00
c1 - adding a minute: 12:01
c2 - created with time 7:15: 07:15
c1 - after creating c2: 07:15
c2 - after adding 2 hours: 09:15
c1 - after c2 is incremented by 2: 09:15
```

2.1.2 Non-Static Example

```
public class Student {
  private String first;
  private String last;
  private int id;

  public Student (String first, String last, int id) {
    this.first = first;
    this.last = last;
    this.id = id;
  }

  @Override
  public String toString () {
    return "(" + last + ", " + first + ")";
  }
}
```

Output from the above code:

```
Total Students in CS160:

Name: Bobby Joe
Major: Mathematics
Minor: Computer Science
Year: Senior
ID Number: 90314

Name: John Doe
Major: Computer Science
Minor: None
Year: Freshman
ID Number: 90213
```

```
Name: Julie Sparkles
Major: English
Minor: None
Year: Junior
ID Number: 91942

Name: Steve Reeves
Major: Physics
Minor: Mathematics
Year: Sophomore
ID Number: 90870

Updated Total Students:

Name: Bobby Joe
Major: Mathematics
Minor: Computer Science
Year: Senior
ID Number: 90314

Name: John Doe
Major: Computer Science
Minor: English
Year: Freshman
ID Number: 90213

Name: Julie Sparkles
Major: Computer Science
Minor: None
Year: Junior
ID Number: 91942

Name: Steve Reeves
Major: Physics
Minor: Mathematics
Year: Junior
ID Number: 90870
```

2.1.3 Calling Static and Non-Static Methods

Calling Static Methods:

```
1  public class StaticExample {
2    public static void printMyArray(int [] array){
3      for (int i = 0; i < array.length; i++){
4          System.out.print(array[i] + " ");
5      }
6      System.out.println(); //used for spacing
7    }
8    public static void doubleMyArray(int [] array) {
9      for (int i = 0; i < array.length; i++)
10       array[i] *= 2;
11   }
12   public static void main (String [] args) {
13     int [] myArray = {1, 2, 3, 4};
14     System.out.println("Initial value of myArray");
15     printMyArray(myArray);
16     doubleMyArray(myArray);
17     System.out.println("Values of myArray after calling doubleMyArray");
18     printMyArray(myArray);
```

```
  }
}
```

Output from the above code:

```
Initial value of myArray
1 2 3 4
Values of myArray after calling doubleMyArray
2 4 6 8
```

Calling Non-Static Methods:

```
public class NonStaticExample {
    private int [] myArray = {1, 2, 3, 4};
    public void printMyArray(){
        for (int i = 0; i < myArray.length; i++){
            System.out.print(myArray[i] + " ");
        }
        System.out.println(); //used for spacing
    }
    public void doubleMyArray() {
      for (int i = 0; i < myArray.length; i++)
        myArray[i] *= 2;
    }
    public static void main (String [] args){
        NonStaticExample ex = new NonStaticExample ();
        System.out.println("Initial value of myArray");
        ex.printMyArray();
        ex.doubleMyArray();
        System.out.println("Values of myArray after calling doubleMyArray");
        ex.printMyArray();
    }
}
```

Output from the above code:

```
Initial value of myArray
1 2 3 4
Values of myArray after calling doubleMyArray
2 4 6 8
```

2.2 Pass-by-Value vs Pass-by-Reference

Pass-by-Value are primitive types that are passed into a method's parameter. **These values are not changed outside the method that initializes them!**. The calling method creates a copy of the values so the original values are never changed. For example:

```
public class PassByValue {
  public static void increment(int n){
    n++;
    System.out.println("Value of n in increment method: " + n);
  }

  public static void main (String [] args) {
    int number = 100;
    System.out.println("Inital value of number: " + number);
    increment(number);
    System.out.println("Value of number after calling increment method: " + number);
  }
}
```

Output from above code:

```
Inital value of number: 100
Value of n in increment method: 101
Value of number after calling increment method: 100
```

Pass-by-Reference are always objects. This is because they have their own specified memory (aka it has memory allocated for the variable), so when a method calls that variable it accesses that place in memory and manipulates that. Therefore, Pass-by-Reference variables are **changed!**. For example:

```
import java.util.Arrays;

public class PassByReference {
  public static void multiplyIndex0(int [] i, int p){
    i[0] *= p;
    System.out.println("Values of i in multiplyIndex0: " + Arrays.toString(i));
  }
  public static void main (String [] args) {
    int [] intArray = {1, 2, 3};
    System.out.println("Inital values of intArray: " + Arrays.toString(intArray));
    multiplyIndex0(intArray, 9);
    System.out.println("Values of intArray after multipleIndex0: " + Arrays.toString(intArray)
    );
  }
}
```

```
Inital values of intArray: [1, 2, 3]
Values of i in multiplyIndex0: [9, 2, 3]
Values of intArray after multipleIndex0: [9, 2, 3]
```

3 Objects

3.1 General Syntax

Creating a constructor:

```
public ClassName (sometimes parameters) {
  // code to initialize instance variables if there parameters
}
```

Instantiating an object:

```
ClassName objectName = new ClassName (constructor parameters if any);
```

Note: If there are no parameters in the constructor it would look like:
`ClassName objectName = new ClassName ();`

3.2 Example

```
public class Book {
    //Instance Variables
    private String title;
    private String author;
    private int year;

    //Constructor
    //NOTE: public Book (method name must be the exact same
    //as class name. You are not returning anything so the
    //format is just public name (parameters, if, needed){}
    public Book (String _title, String _author, int _year) {
        title = _title;
        author = _author;
        year = _year;    //NOTE: no return value
    }
    //Getters
    public String getTitle (){
        return title;
    }
    public String getAuthor () {
        return author;
    }
    public int getYear () {
```

```
        return year;
    }
    //Setters
    public void setTitle (String _title){
        title = _title;
    }
    public void setAuthor (String _author) {
        author = _author;
    }
    public void setYear (int _year) {
        year = _year;
    }
    //toString
    public String toString () {
        String s = "";
        s += "Title: " + title + ", " ;
        s += " Author: " + author + ", ";
        s += " Year: " + year;
        return s;
    }
    public static void main (String [] args){
        Book book0 = new Book ("It's Raining from the Clouds",
                               "Oh Knowledgeable One", 1970);
        Book book1 = new Book("Life Without a Cell Phone: The Nightmare of Tweens",
                              "Bored and Un-Social", 2013);
        Book book2 = new Book ("Running out of Clever Names",
                               "Addy Moran", 2016);
        Book [] Library = {book0, book1, book2};

        for (int i = 0; i < Library.length; i++)
            System.out.println(Library[i]);
    }
}
```

Output from the above code:
`Title: It's Raining from the Clouds,  Author: Oh Knowledgeable One,  Year: 1970`
`Title: Life Without a Cell Phone: The Nightmare of Tweens,  Author: Bored and Un-Social,  Year: 2013`
`Title: Running out of Clever Names,  Author: Addy Moran,  Year: 2016`

3.3 Suggestions, Warnings, and Resources

- Resource: Tutorials Point - Method Tutorial

4 ArrayLists

General initialization syntax:
`ArrayList<type> name = new ArrayList<>();` or `ArrayList<type> name = new ArrayList<type>();`

```
import java.util.ArrayList;

public class ArrayListExamples {
  public static void main (String [] args) {
    ArrayList<String> names = new ArrayList<String>();
    names.add("Bob");
    names.add("Bobby");
    names.add("Bobina");

    System.out.println(names);

    for (int i = 0; i < names.size(); i++)
      System.out.print(names.get(i) + " ");
    System.out.println();

    names.remove(0);
    System.out.println(names);
    names.remove("Bobina");
```

```
    System.out.println(names);

    // For primitive types, you MUST use their corresponding wrapper class
    ArrayList<Integer> iList = new ArrayList<>();
    iList.add(1);
    iList.add(0);
    iList.add(1);
    iList.add(0);
    iList.add(1);
    iList.add(1);

    if (iList.contains(0))
      System.out.println("indexOf(0): " + iList.indexOf(0));
    System.out.println("get the int at index 1: " + iList.get(1));
    System.out.println(iList);

    // Creating an ArrayList of a class type
    //   (class below)
    ArrayList <Student> studentList = new ArrayList<>();
    Student s = new Student ("Joe", "Steve", 112);
    studentList.add(s);
    studentList.add(new Student("Arabelle", "Jones", 118));
    studentList.add(new Student("James", "Potter", 200));
    System.out.println(studentList);
  }
}
```

```
public class Student {
  private String first;
  private String last;
  private int id;

  public Student (String first, String last, int id) {
    this.first = first;
    this.last = last;
    this.id = id;
  }

  @Override
  public String toString () {
    return "(" + last + ", " + first + ")";
  }
}
```

Output from the above code:

```
[Bob, Bobby, Bobina]
Bob Bobby Bobina
[Bobby, Bobina]
[Bobby]
indexOf(0): 1
get the int at index 1: 0
[1, 0, 1, 0, 1, 1]
[(Steve, Joe), (Jones, Arabelle), (Potter, James)]
```

4.1 Suggestions, Warnings, and Resources

- Warning: For primitive types you must use their corresponding wrapper classes!
- Resource: Java Documentation ArrayList
- Resource: Tutorial's Point ArrayList Tutorial

5 File I/O

5.1 Reading a File

Think of reading a file just as reading input from the keyboard (the only differences are when setting up the Scanner instead of having `System.in` you are going to use a File object and you need to catch the exception).

```
import java.io.File;
import java.util.Scanner;
import java.io.IOException;
import java.io.FileNotFoundException;

public class ReadFile {
  public static void main (String [] args) {
    // Below are three different ways of
    // creating a Scanner that reads a file
    // and different ways to write an try-catch block
    System.out.println("Reading the file the first time:");
    String filename = "example.txt";
    try {
      Scanner reader = new Scanner (new File (filename));
      String line;
      while (reader.hasNextLine()){
        line = reader.nextLine();
        System.out.println(line);
      }
      reader.close();
    } catch (IOException e) {
      System.out.println(e);
    }

    System.out.println("\nReading the file the second time:");
    try {
      Scanner reader = new Scanner (new File ("example.txt"));
      String word;
      while (reader.hasNext()) {
        word = reader.next();
        System.out.println(word);
      }
      reader.close();
    } catch (Exception e) {
      System.out.println("Could not read file");
      System.exit(-1);
    }

    System.out.println("\nReading the first word of the file (again):");
    try {
      File f = new File ("example.txt");
      Scanner reader = new Scanner (f);
      String firstWord = reader.next();
      System.out.println("First word is: " + firstWord);
      reader.close();
    } catch (FileNotFoundException e) {
      System.out.println(e);
    }
  }
}
```

Contents of `example.txt`:

```
Hey
how's it going??
Okay,
Bye
```

The output to the code above:

```
Reading the file the first time:
```

```
Hey
how's it going??
Okay,
Bye

Reading the file the second time:
Hey
how's
it
going??
Okay,
Bye

Reading the first word of the file (again):
First word is: Hey
```

5.2 Writing to a File

Think of PrintWriters as regular `System.out.print`/`System.out.println`/`System.out.printf` statements except instead of printing to the console you're writing to a file (just remember to use the PrintWriter name instead of `System.out` and you have to catch an exception).

```
1  import java.io.PrintWriter;
2  import java.io.File;
3  import java.io.IOException;
4
5  public class WriteFile {
6    public static void main (String [] args) {
7      try {
8        /*
9        Could also:
10       File f = new File ("output.txt");
11       PrintWriter writer = new PrintWriter (f);
12       */
13       PrintWriter writer = new PrintWriter (new File ("output.txt"));
14       System.out.println("Writing to the file...");
15       writer.print("Hey! ");
16       String fileContents = "write this to the file...";
17       writer.println(fileContents);
18       writer.printf("Purpose: %s\nDate: %d-%d-%d", "CS150 Study Guide", 11, 2, 2017);
19       // this must happen, otherwise the file won't get written!
20       writer.close();
21       System.out.println("Finished writing to the file");
22     // Could also use Exception e
23     } catch (IOException e) {
24       // Could also write your own error message then exit
25       System.out.println(e);
26     }
27   }
28 }
```

The output from the above code:

```
Writing to the file...
Finished writing to the file
```

The resulting file `output.txt` contains:

```
Hey! write this to the file...
Purpose: CS150 Study Guide
Date: 11-2-2017
```

5.3 Suggestions, Warnings, and Resources

- Warning: You **MUST** close your PrintWriter!
- Warning: You must pass a File object to read/write to a file (you can't just put the path to the file as a String)
- Resource: Tutorials Point - File I/O Tutorial
- Resource: Geeks For Geeks - Exceptions in Java
- Resource: Oracle's Exceptions Tutorial

6 Interfaces

Interfaces only have method headings, the class that implements the interface completes the methods from the method heading. If you'd like to implement an interface your class must have all of the methods from the interface or else your code won't compile. A few examples:

1. `public class P9 implements Interface`
2. `public class Q1 implements QuizInterface`
3. `public class TicTacToe implements Game`

7 Practice Written Exam

7.1 Short Answer

1. Write a for loop that prints the numbers 3 to 8 separated by a comma. It is okay to have a trailing comma at the end.
2. Write a while loop that prints the numbers 3 to 8 separated by a comma. It is okay to have a trailing comma at the end.
3. Write a do-while loop that prints the numbers 3 to 8 separated by a comma. It is okay to have a trailing comma at the end.
4. Using a loop (of any kind) print all numbers that are a multiple of three and that is between 1 and 50 (inclusive) separated by semicolons.
5. Using a loop (of any kind) print each character of the pre-defined String `s` separated by a new line.
6. Using a loop (of any kind) print the pre-defined String `s` backwards (characters all on the same line).
7. Using a loop (of any kind) print every other letter of the pre-defined String variable `s` (characters all on the same line).
8. Initialize a 1-D String array called `names` with the values: Bob, Bobina, and Joe.
9. Create a 1-D double array called `averages` with a capacity of 10.
10. Using a loop (of any kind) print the contents of `names` all on new lines.
11. Using `Arrays.toString()` print the values in `averages`.
12. Instantiate a 4x4 2-D int array called `matrix`.
13. Initialize every value of `matrix` to 1.
14. Change the value on the first row, second column to 2.
15. Print `matrix` using a for loop.
16. Initialize an int ArrayList called `iList`.
17. Add 3 into `iList`.
18. Add 4 into `iList`.
19. Add 99 into `iList`.
20. Print `iList` using the default `toString` method.
21. Print every element in `iList` separated by a comma, all on the same line using a for-each loop
22. Inside the predefined class `Student` create a Student object called `student0`, who's name is "James Bond" with a student id of 007. Use the following code as guidance:

```
public class Student {
    String id;
    String name = "";
    public Student (String _id, String _name){
        id = _id;
        name = _name;
    }
}
```

23. Using the same class (`Student`) and the code from above. Create an Student object called `student1`, who's name is "Jr Bond" with a student id of 008.

24. Create an array of type `Student` called `overAchievers` and insert student0 and student1 (from questions 5 and 6) into the array.

25. Initialize a Scanner that reads from the console.

26. Initialize a Scanner that reads from a file stored in the String variable `inputFile`. Include a try-catch.

27. Initialize a Scanner that reads and stores the following from a file called `input.txt`: a line, a word, an integer, and a double. Include a try-catch.
    Assume the file is in the correct format Include a try-catch.

28. Initialize a PrintWriter that writes to a file called `ouput.txt`. Include a try-catch.

29. Initialize a PrintWriter that writes to a file stored in the String variable `outputFile` and write the pre-defined double `d` to 4 decimal places and a pre-defined String `s` both on new lines. Include a try-catch.

7.2 Tracing

Instructions: For each question (unless specified differently) write what would be printed (even if there are errors earlier in the code that would cause the program not to compile).

```
import java.util.Arrays;

public class Car {
    private String make;
    private String model;
    private int year;
    private String nickName;
    private double miles;
    public static Car [] carArray;

    public Car (String make, String model, int year, String nickName, double miles){
        setMake(make);
        setModel(model);
        setYear(year);
        setNickName(nickName);
        setMiles(miles);
    }
    public String getMakeAndModel (){
        return make + " " + model;
    }
    public void setMake (String s) {
        make = s;
    }
     public void setModel (String s) {
        model = s;
    }
     public void setYear (int i) {
        year = i;
    }
     public void setNickName (String s) {
        nickName = s;
    }
     public void setMiles (double d) {
        miles = d;
    }
    public int getYear () {
        return year;
    }
    public String getNickName() {
        return nickName;
    }
    public double getMiles () {
        return miles;
    }
    public String toString (){
        String s = "Make: " +  make;
        s += " Model: " + model;
        s += " Year: " + year;
        s += " Nickname: " + nickName;
        s += " Mileage: " + miles;
        return s;
    }
    public static void main (String [] args){
        Car c0 = new Car ("Chevy", "Camaro", 2013,
                          "Lightning McQueen", 15000);
        Car c1 = new Car ("Ford", "F150", 1950, "Tow Mater", 200000);
        Car c2 = new Car ("Ford", "Coupe", 1936, "Doc Hudson", 150000);
        Car c3 = new Car ("Mack", "Flintstone", 1980, "Mack", 100000);
        Car [] carsCharacters = {c0, c1, c2, c3};
        //Question 1:
        System.out.println(carsCharacters[2]);
        //Question 2:
        System.out.println(c1.getYear());
        //Question 3:
        for (int i = 0; i < carsCharacters.length; i++){
```

```
66              System.out.println(carsCharacters[i].getNickName());
67          }
68      }
69  }
```

8 Programming Quiz Practice Exam

1. Create a class called `SuperHero`.
2. Create three instance variables: a String for the super hero's name, an int for their age, and an ArrayList of Strings of their super powers.
3. Create a constructor for the SuperHero class that takes parameters to fill the instance variables (i.e. a String, an int, and an ArrayList of Strings).
4. Create a toString method that prints the super hero's name, age, and superpowers a ll on new lines. Example:

   ```
   Superman
   28
   Flying, Strong, Comes included with cape,
   ```

5. Write a void method called `addPower` that takes in a String. The method adds a superpower to the superpower ArrayList.
6. Create an ArrayList of Strings that has the values "strong", "brave", "tights"
7. Create a `SuperHero` object, his name is "Spiderman", his age is 26, and his powers are included in the ArrayList from the previous question.
8. Using the `toString` method, print the contents of the `SuperHero` object created in the previous questions
9. Add "climbing" to Superman's superpowers
10. Create a `SuperHero` object based on the following information:
    name = Hulk, age = 24, powers = strong, camo
11. Create a `SuperHero` object based on the following information:
    name = Thor, age = 38, powers = enchanted hammer, thunder
12. Create an ArrayList of `SuperHero` objects filled with the SpiderMan, Hulk, and Thor `SuperHero` objects.
13. Create a `PrintWriter` that writes the ArrayList of super heroes to a file called `super.txt`.

9 Suggestions for Studying and Test Taking

9.1 Written

When reading through code and writing the output: Write your variables on the side and as your variables change in the program, you change your variables on the side.

Practice writing code in Eclipse and before you run your program guess what the output would be. This is good practice for testing your own programs and also for the code tracing part of the exam.

If you need more tracing examples (or more coding examples in general), there is a "Programs" tab on the CS150 homepage. There are also examples on the Progress page.

9.2 Programming Quiz

Redo past recitations and assignments until you no longer need to use the internet, friends, or past code.

Practice writing code in Eclipse. Make up projects and problems or ask a TA and they can give you some challenges.

Look at code, the more exposure you get to code (whether it's your own code or not) the easier it is to understand. Some sample code is under the "Programs" tab and the Progress Page.

9.3 Common Errors

- Incorrect brackets around conditional statements
- Semicolons right after loops and if statements

9.4 Challenges

CodingBat and Hackerrank offer good extra coding practice.

ANSWERS ON THE NEXT PAGE

10 Answers to Practice Written and Programming Problems

10.1 Written

Note: Your implementations may be different (different variable names, $<$ vs $\leq$, while, do-while, or for, etc). These answers are just for guidance and there are many ways to correctly implement these questions.

1.
```
// could also have (i < 9)
for (int i = 3; i <= 8; i++)
  System.out.print(i + ",");
```

2.
```
// could also have (i1 <= 8)
// could also increment outside of print
int i1 = 3;
while (i1 < 9)
  System.out.print(i1++ + ",");
```

3.
```
// could use pre or post increment or increment like in the previous question
int i2 = 3;
do {
  System.out.print(i2 + ",");
  i2++;
} while (i2 <= 8);
```

4.
```
for (int i = 0; i <= 50; i++)
  if (i % 3 == 0)
     System.out.print(i + ";");
```

5.
```
for (int i = 0; i < s.length(); i++)
  System.out.println(s.charAt(i));
```

6.
```
for (int i = s.length() - 1; i >= 0; i--)
  System.out.print(s.charAt(i));
```

7.
```
for (int i = 0; i < s.length(); i+=2)
  System.out.print(s.charAt(i));
```

8. `String [] names = {"Bob", "Bobina", "Joe"};`

9. `double [] averages = new double [10];`

10.
```
for (int i = 0; i < names.length; i++)
  System.out.println(names[i]);
```

11. `System.out.println(Arrays.toString(averages));`

12. `int matrix [] [] = new int [4][4];`

13.
```
for (int row = 0; row < matrix.length; row++)
    for (int col = 0; col < matrix[row].length; col++)
        matrix[row][col] = 1;

```

14. board[0][1] = 2;

15.
```
for (int row = 0; row < matrix.length; row++){
    for (int col = 0; col < matrix[row].length; col++)
        System.out.print(matrix[row][col]);
     System.out.println();  //added for spacing
}
```

16. `ArrayList<Integer> iList = new ArrayList<Integer>();`
    NOTE: This is also correct: `ArrayList<Integer> iList = new ArrayList<>();`

17. `iList.add(3);`

18. `iList.add(4);`

19. `iList.add(99);`

20. `System.out.println(iList);`

21.
```
for (int i : iList)
  System.out.print(i + ",");
```

22. `Student student0 = new Student ("007", "James Bond");`

23. `Student student1 = new Student ("008", "Jr Bond");`

24.
```
Student [] overAchievers = {student0, student1};
```

25. `Scanner keys = new Scanner (System.in);`

26. **NOTE**: for questions 14 - 17, you're exceptions may be different, look at the reading/writing to a file sections above to find what is acceptable.

```
try {
  Scanner reader = new Scanner (new File(inputFile));
} catch (IOException e) {
  System.out.println(e);
}
```

27.
```
String word = "";
int i = 0;
double d = 0.0;
String line = "";
try {
  Scanner reader = new Scanner (new File("input.txt"));
  line = reader.nextLine();
  word = reader.next();
  i = reader.nextInt();
  d = reader.nextDouble();
  reader.close();
} catch (Exception e) {
  System.out.println(e);
}
```

28.
```
try {
  PrintWriter pw = new PrintWriter (new File("output.txt"));
} catch (Exception e) {
  System.out.println("Could not print to file");
  System.exit(-1);
}
```

29.
```
try {
  PrintWriter pw = new PrintWriter (new File(outputFile));
  pw.printf("%.4f\n", d);
  pw.println(s);
  pw.close();
} catch (IOException e) {
  System.out.println(e);
}
```

10.2 Tracing

1. Make: Ford Model: Coupe Year: 1936 Nickname: Doc Hudson Mileage: 150000.0
2. 1950
3. `Lightning McQueen`
   `Tow Mater`
   `Doc Hudson`
   `Mack`

10.3 Programming

NOTE: Your variable names/try-catch statements may look different.

```
import java.util.ArrayList;
import java.io.PrintWriter;
import java.io.File;
import java.io.IOException;

public class SuperHero {
  private String name;
  private int age;
  private ArrayList<String> superpowers;

  public SuperHero (String name, int age, ArrayList<String> superpowers) {
    this.name = name;
    this.age = age;
    this.superpowers = superpowers;
  }

  @Override
  public String toString () {
     String ret = name + "\n" + age + "\n";
     for (String power : superpowers)
       ret += power + ",";
     return ret;
  }

  public void addPower (String power) {
    superpowers.add(power);
  }

  public static void main (String [] args) {
    ArrayList<String> spiderman_powers = new ArrayList<>();
    spiderman_powers.add("strong");
    spiderman_powers.add("brave");
    spiderman_powers.add("tights");

    SuperHero spiderman = new SuperHero ("Spiderman", 26, spiderman_powers);

    System.out.println(spiderman);

    spiderman.addPower("climbing");

    ArrayList<String> hulk_powers = new ArrayList<String>();
    hulk_powers.add("strong");
    hulk_powers.add("camo");
    SuperHero hulk = new SuperHero("Hulk", 24, hulk_powers);

    ArrayList<String> thor_powers = new ArrayList<String>();
    thor_powers.add("enchanted hammer");
    thor_powers.add("thunder");
    SuperHero thor = new SuperHero ("Thor", 38, thor_powers);

    ArrayList<SuperHero> superheroes = new ArrayList<SuperHero>();
    superheroes.add(spiderman);
    superheroes.add(hulk);
```

```
54      superheroes.add(thor);
55
56      try {
57        PrintWriter printer = new PrintWriter (new File ("super.txt"));
58        for (SuperHero hero : superheroes)
59          printer.println(hero);
60        printer.close();
61      } catch (IOException e) {
62        System.out.println(e);
63      }
64    }
65  }
```

`super.txt` should have the following contents:

```
Spiderman
26
strong,brave,tights,climbing,
Hulk
24
strong,camo,
Thor
38
enchanted hammer,thunder,
```