CS 163/164 - Exam 2 Study Guide and Practice Exam

October 9, 2017

Summary

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 Loops

2.1 for loops

For loops are generally used when you know when you want to stop. For example if you need to count to 100 or if you need to loop the length of a String.

General syntax:

```java
for (initialize; termination condition; update) {
  // code
}
```

A few examples:

```java
public class ForLoops {
  public static void main (String [] args) {
    for (int i = 0; i < 10; i++)
      System.out.print(i + " ");
    System.out.println(); // used for spacing

    String s = "Hello! How are you?";
    for (int i = 0; i < s.length(); i++)
      System.out.print(s.charAt(i) + ":");
    System.out.println(); // used for spacing

    for (int i = 3; i >= 0; i--)
      System.out.println(i);
    System.out.println("Blastoff!!");

    int count = 0;
    for (int i = 1; i < 50; i+= 2) {
      System.out.print(i + ", ");
      count++;
    }
    System.out.printf("Number of odd numbers < 50: %d\n", count);

    for (char c = 'a'; c <= 'z'; c++)
      System.out.print(c + "*");
    System.out.println(); //used for spacing
  }
}
```

The output from the above code:

```
0 1 2 3 4 5 6 7 8 9
H:e:l:l:o:!: :H:o:w: :a:r:e: :y:o:u:?:
3
2
1
0
Blastoff!!
0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48,
Number of odd numbers < 50: 25
a*b*c*d*e*f*g*h*i*j*k*l*m*n*o*p*q*r*s*t*u*v*w*x*y*z*
```

2.2 while loops

while loops are generally used when you don't know when you are going to end. For example, if you are waiting for a change in the system or for an action from the user (versus knowing you'll end after the 100th run every time like a for loop).

General syntax:

```
initialize
while (termination condition) {
  // code
  update
}
```

A few examples:

```java
public class WhileLoops {
  public static void main (String [] args) {
    int i0 = 0;
    while (i0 < 10) {
      System.out.print(i0 + " ");
      i0++;
    }
    System.out.println(); // used for spacing

    String s = "Hello! How are you?";
    int i1 = 0;
    while (i1 < s.length()) {
      System.out.print(s.charAt(i1) + ":");
      i1++;
    }
    System.out.println(); // used for spacing

    int i2 = 1, count = 0;
    while (i2 < 50) {
      System.out.print(i2 + ", ");
      i2 += 2;
      count++;
    }
    System.out.printf("\nNumber of odd numbers < 50: %d\n", count);
  }
}
```

The output from the above code:

```
0 1 2 3 4 5 6 7 8 9
H:e:l:l:o:!: :H:o:w: :a:r:e: :y:o:u:?:
0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48,
Number of odd numbers < 50: 25
```

2.3 do-while loops

do-while loops are very similar to while loops but do-while loops always execute the code inside the brackets **at least once**.

General syntax:

```
initialize
do {
  // code
  update
} while (termination condition);
```

A few examples:

```java
import java.util.Scanner;

public class DoWhileLoops {
  public static void main (String [] args) {
    int i0 = 0;
    do {
      System.out.print(i0 + " ");
      i0++;
    } while (i0 < 10);
    System.out.println(); // used for spacing

    String s = "Hello! How are you?";
    int i1 = 0;
  do {
      System.out.print(s.charAt(i1) + ":");
       i1++;
    } while (i1 < s.length());
    System.out.println();

    int i2 = 1, count = 0;
    do {
      System.out.print(i2 + ", " );
      i2 += 2;
      count++;
    } while (i2 < 50);
    System.out.printf("\nNumber of odd numbers < 50: %d\n", count);

    Scanner reader = new Scanner (System.in);
    String response = "";
    do {
      System.out.print("Are we there yet? ");
      response = reader.nextLine();
    } while (!response.equalsIgnoreCase("yes"));
    System.out.println("Finally!!!");
    reader.close();
  }
}
```

The output from the above code:

```
0 1 2 3 4 5 6 7 8 9
H:e:l:l:o:!: :H:o:w: :a:r:e: :y:o:u:?:
0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48,
Number of odd numbers < 50: 25
Are we there yet? no
Are we there yet? almost...
Are we there yet? still not there
Are we there yet? yes
Finally!!!
```

2.4 Suggestions, Warnings, and Resources

- Resource: For loops resource
- Resource: While loops resource
- Resource: Do-While loops resource
- Resource: Java Documentation for break and continue statements
- OPTIONAL: For-Each Loop Tutorial
- Warning: Watch the ranges! When a String's length is of size 8, i $\leq$ 9 is the same as i $\leq$ 8. However, i $\leq$ 9 will give you an error.
- Remember: You can always change your update by using the "+=" or "-=" method, especially if you are updating by more than 1.
- Remember: You can always change your starting point, you don't have to start at 0, it is just the most common. If you were asked to reverse a string, you can always make the starting point at stringName.length()-1!
- Summary: Basically you can solve a problem with loops in many different ways whether it's using a different loop, starting at a different place, changing your ending value, or changing how you update.

2.5 FAQs

1. **Q:** When do you use one loop over another?
   A: You can almost always use any loop you choose, but there maybe one that is easier than the other. For example, in the "Are we there yet?" example from above, using a do-while is nice because the loop always runs at least once (you could have also used a while loop but you would have had to ask the "Are we there yet?" question before and in the while loop).

3 Arrays

3.1 1D Arrays

3.1.1 General Syntax

General initialization syntax (there are two ways):

1. When you know the size:
   `typeOfArray [] nameOfArray = new typeOfArray [sizeOfArray];`
2. When you know the values:
   `typeOfArray [] nameOfArray = {values, you, want, in, the, array};`

Ways to print 1-D arrays (there are two ways):

1. Using a loop:

```
for (int i = 0; i < arrayName.length; i++){
    System.out.print(arrayName[i]);
}
```

2. Using `Arrays.toString()`

```
System.out.println(Arrays.toString(arrayName));
```

   Note: To use `Arrays.toString()` you must import the Arrays class (`import java.util.Arrays;`)

You can access an element of a 1D array by using `arrayName[indexOfElement];`.

Below are some examples:

```
import java.util.Arrays;

public class OneDimArrays {
  public static void main (String [] args) {
    // Initializing a 1-D Array:
    int [] iArray = new int [10];
    String [] csClasses = {"CS150", "CS163", "CS164", "CS270", "CS253"};

    // Manipulating 1-D Arrays:

    // Getting the length of the array
    // Note: You could also print length using arrayName.length;
    int arrayLength = csClasses.length;

    // Assigning all indexes to one value
    for (int i = 0; i < iArray.length; i++)
      iArray[i] = 1;

    // Changing a value at a specific index
      iArray[3] = 5;

    // Printing the array using a for loop
    for (int i = 0; i < iArray.length; i++)
```

```
      System.out.print(iArray[i] + ", ");
    System.out.println(); // used for spacing

    // Printing the array using Arrays.toString()
      System.out.println(Arrays.toString(iArray));
  }
}
```

Output from the above code:

```
1, 1, 1, 5, 1, 1, 1, 1, 1, 1,
[1, 1, 1, 5, 1, 1, 1, 1, 1, 1]
```

3.2 2D Arrays

3.2.1 General Syntax

General initialization syntax (there are two ways):

1. When you know the size:
   `typeOfArray [][] nameOfArray = new typeOfArray [numRows][numCols];`

2. When you know the values:

```
typeOfArray [] nameOfArray = {{values, you, want, in, row 1},
                              {values, you, want, in, row 2},
                              {values, you, want, in, row 3}};
```

You can print a 2-D array using a loop:

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

You can access an element of a 2D array by using `arrayName[rowIndex][colIndex];`.

Below are some examples:

```
public class TwoDimArrays {
  public static void main (String [] args) {
    // creating a 3x3 array
    int [][] board = new int [3][3];

    // assigning all values of the 3x3 array to 0
    for (int row = 0; row < board.length; row++)
      for (int col = 0; col < board[row].length; col++)
        board[row][col] = 0;

    System.out.println("Printing initial values of the 2D array");
    // printing the values of the 2D array
    for (int i = 0; i < board.length; i++) {
      for (int j = 0; j < board[i].length; j++)
        System.out.print(board[i][j] + " ");
      System.out.println(); // used for spacing
    }

    // assigning the third element in the first row to 1
    board[0][2] = 1;
    System.out.println("After changing third element in the first row to 1");
    // printing the values of the 2D array
    for (int i = 0; i < board.length; i++) {
      for (int j = 0; j < board[i].length; j++)
         System.out.print(board[i][j] + " ");
      System.out.println(); // used for spacing
    }

    // assigning the first element in the third row to 1
    board[2][0] = 1;
    System.out.println("After changing first element in the third row to 1");
    // printing the values of the 2D array
    for (int i = 0; i < board.length; i++) {
      for (int j = 0; j < board[i].length; j++)
        System.out.print(board[i][j] + " ");
      System.out.println(); // used for spacing
    }
  }
}
```

Output from the above code:

```
Printing initial values of the 2D array
0 0 0
0 0 0
0 0 0
After changing third element in the first row to 1
0 0 1
0 0 0
0 0 0
After changing first element in the third row to 1
0 0 1
0 0 0
1 0 0
```

Manipulating 2-D Arrays:

```
int [] [] board = new int [3][3];

//assigning all indexes to one value
for (int row = 0; row < board.length; row++)
    for (int col = 0; col < board[row].length; col++)
        board[row][col] = 0;

//changing one value at a specific index
board[0][2] = 1;
```

3.3 Suggestions, Warnings, and Resources

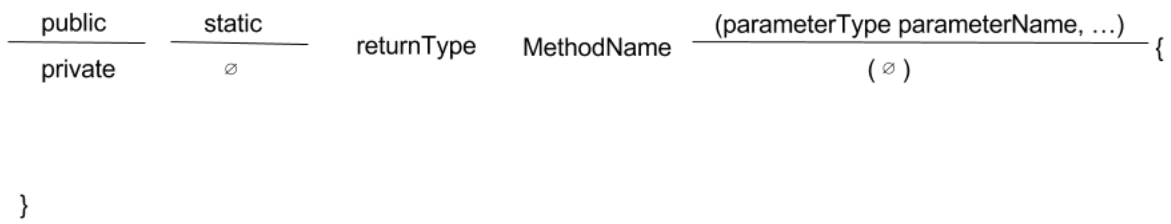
- Warning: Be careful with your indexes. If a 2-D Array has a length of 3 and a height of 3, remember when you print or change the values that you could only use indexes 0 - 2.
- Resource: Tutorials Point - Array Tutorial

3.4 Common Exceptions

1. ArrayIndexOutOfBoundsException: To fix check all of your loop ranges and all the places that you changed a value (i.e. `array[3] = 3;`). Make sure you are never trying to access any index greater than or equal to the array length (same concept for 2-D Arrays).
2. NullPointerException: To fix check to make sure your array has been initialized.

4 Methods and Data

The general format of a method:



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

4.1 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.

4.1.1 Static Example

```java
// 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
```

4.1.2 Non-Static Example

```java
import java.util.Arrays;

/* Example of a Non-Static Class: Student
 * If you are trying to use instance variables the methods either need to be non-static or you
     need to create an object inside the static method (for example, making an R9 object inside
     the main just to test).

 * When you create multiple objects (like Student s1, s2, ...) in a non-static environment you
     are creating SEPERATE objects (the information stored in the instance variables are
    different for each object). This is good because Bobby Joe should be able to have a
    different name, major, minor, year, and id number compared to Julie Sparkles. With non-
    static if we change John Doe's information it wouldn't change Steve Reeve's information.
    However, if the instance variables and methods were static it WOULD change Steve Reeve's
    information if we changed John Doe's information (because when creating multiple objects in
     a static environment you are using ONE "version" of the instance variables). This could be
     good if you only want one copy (for example pi (Math.PI), there should only be one copy of
     pi), however that wouldn't be appropriate for this class.
 */

public class Student {
    // instance variables
    private String name, year, major, minor;
    private int id;

    // constructor
    public Student (String _name, String _major, String _year, int _id){
        this.name = _name;
        major = _major;
        minor = "None";
        year = _year;
        id = _id;
    }
    // Overloading previous constructor
    public Student (String _name, String _major, String _minor, String _year, int _id){
        name = _name;
        major = _major;
        minor = _minor;
        year = _year;
        id = _id;
    }

    public void increaseYear (){
        switch (year){
            case "Freshman": year = "Sophomore"; break;
            case "Sophomore": year = "Junior"; break;
            case "Junior": year = "Senior"; break;
            case "Senior": year = "Super Senior"; break;
            default: year = "Unknown"; break;
        }
    }

    public void changeMajor (String new_major){
        major = new_major;
    }
    public void addMinor (String _minor){
        minor = _minor;
    }
    //toString
    public String toString (){
        return String.format("Name: %s%nMajor: %s%nMinor: %s%nYear: %s%nID Number: %d", name,
    major, minor, year, id);
    }
    public static void main (String [] args){
        Student bob = new Student ("Bobby Joe", "Mathematics", "Computer Science", "Senior",
    90314);
        Student john = new Student ("John Doe", "Computer Science", "Freshman", 90213);
        Student julie = new Student ("Julie Sparkles", "English", "Junior", 91942);
        Student steve = new Student ("Steve Reeves", "Physics", "Mathematics", "Sophomore",
    90870);
        //System.out.println(bob);
        Student [] cs160 = {bob, john, julie, steve};
        System.out.println("Total Students in CS160:\n");
        for (int i = 0; i < cs160.length; i++){
            System.out.println(cs160[i] + "\n");
        }
        julie.changeMajor("Computer Science"); // because it's awesome
        john.addMinor("English");
        steve.increaseYear();
        System.out.println("Updated Total Students:\n");
        for (int i = 0; i < cs160.length; i++){
            System.out.println(cs160[i] + "\n");
        }
    }
}
```

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
```

4.1.3 Calling Static and Non-Static Methods

Calling Static Methods:

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

    // Static method call to another class
    double min = Math.min(3, 6);
  }
}
```

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:

```java
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
```

4.2 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:

```java
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:

```java
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]
```

5 Objects

5.1 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 ();`

5.2 Example

```java
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
```

5.3 Suggestions, Warnings, and Resources

- Resource: Tutorials Point - Method Tutorial

6 Bitwise Operators

- AND (&)
- OR (|)
- NOT or Compliment (∼)
- XOR (∧)
- Left shift (<<)
- Right shift (>>)

6.1 Suggestions, Warnings, and Resources

- Resource: Tutorials Point - Bitwise Tutorial
- Resource: Tutorials Point - Bitwise Example

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, 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. 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:

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

17. 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.
18. Create an array of type `Student` called `overAchievers` and insert student0 and student1 (from questions 5 and 6) into the array.

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).

```java
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++){
                System.out.println(carsCharacters[i].getNickName());
            }
        }
}
```

8 Programming Quiz Practice Exam

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

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));
```

Note: For answers 4- 7, your implementation may be different (¡ vs ¡=, while, do-while, or for, etc). These answers are just for guidance and there are many ways to correctly implement these questions.

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

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

18.
```
Student [] overAchievers = {student0, student1};
```

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
   ```