APCS Summer Assignment

Ms. Frew and Ms. Gorine Computer Lab 253 or 125

Congratulations you have signed up for an interesting and challenging course – APCS. Students entering APCS are expected to have a working knowledge of Java when they arrive in September. You should have downloaded the software that we use and have completed the Summer Assignment when you return to school in the fall (**see the last page for what to turn in**). You will be much more successful if you complete the required assignment. Do not wait until the last week-end. Get help if you need it. Our e-mails are: ngorine@fcps.edu and lefrew@fcps.edu . We look forward to working with you.

Install Java and jGrasp on your computer use the following Java Installation instructions.

In order to program with Java on your computer, you need two components:

- Java JDK (do **not** use Java 8, JGrasp is not yet compatible with it)
- jGrasp (or other environment)

First you have to download the Java JDK (Java Development Kit). Go to
http://java.sun.com/javase/downloads/widget/jdk6.jsp

And then select your operating system, and click download. If it asks for a log in, click "skip this step". You should be directed to a page with a red table that shows what you want to download, and the JDK should be in it. Below there will be a red box that says "Download selected with Sun Download Manager". You could choose this, but it may be easier to simply click the filename (it's a link on that page) right below the Java SE Development Kit item listed in the table.

Once you do this, the JDK will download, and then you should click through the install process. Once this has finished, you can now download jGrasp.

jGrasp can be downloaded from this page:
http://spider.eng.auburn.edu/user-cgi/grasp/grasp.pl?;dl=download_jgrasp.html

Scroll down just past the survey, and you will find three buttons to download jGrasp for your Operating system. Once again, select the button corresponding to your operating system. The file should download, run, and then just go through the install process. It may give an option about administrator tools. If it does, you don't need to install those.

That should be all you need to start programming with Java on your computer. Run jGrasp (it should have created an icon on your desktop), and it should set itself up, and then open. From here you can start writing programs like usual.

If you need a video describing the process: http://snnetworks.homeip.net/files/tutorials/Installing%20Java.wmv
or http://www.youtube.com/watch?v=YJnK10X9niU&feature=related

There is also a first Java Tutorial to help you get started writing your first program.
http://www.youtube.com/watch?v=BhCU8UYJHYI
Both these tutorials were written by MHS student Kais Sorrels.

Please read through the notes and assignments.
Java Basics – Notes
Discussion Arrays - Notes
Array Exercises - Worksheet
Number Array Lab
String Array Lab
Classes – Notes
Student Lab and StudentArray Lab

You can use this tutorial to help you cover the topics from CS `http://chortle.ccsu.edu/CS151/cs151java.html`

In regular Computer Science we cover the material from parts 3, 4, 6, 8, and 9. We also cover part 5 and 7 but these are not on the AP exam.

Summer Assignment Part 1

APCS Quick Review of Java Basics

Java is an object-oriented language that has revolutionized the software industry. Object-oriented languages use libraries of *classes*. A class may be defined as an abstract data type (essentially an abstraction) that can be used to represent real-world concepts. Examples of classes include Customer, Account, SavingsAccount, Rental, Reservation, String, and the like. The earliest version of Java had about 250 classes, whereas the current version has over 3500 classes! This indicates that Java has grown by leaps and bounds. The language is much faster, easier to use, and far more robust than its early versions.

One of Java's primary strengths is its portability. It is useful to understand what makes Java so portable. The code that you write in Java (referred to as *source code*) is stored in a file that has a .java extension. A sample program (i.e. the source code) is shown below:

```
public class FirstProgram {
      public static void main(String[] args)
      {
            System.out.println("Hi! Welcome to my Java course");
      }
}
```

This program is stored in a file called `FirstProgram.java`. Note that the file name is the same as the class. This is mandatory. Remember that Java is case-sensitive. The next step is to compile the program.

If all goes well, a file called `FirstProgram.class` is created. While compilers of most languages translate source code to a format that is specific to the architecture of the machine on which the compilation is done, Java's compiler transforms the source code to what is known as ***bytecode***. Bytecode is independent of a machine's architecture and may be executed on any Java Virtual Machine (JVM). JVM, as the name implies, is not a real machine, but a virtual one that runs in software and is independent of technology/hardware/operating-system. Bytecode provides the virtual machine instructions for the JVM. The program may then be executed. The output of our sample code will be:
Hi! Welcome to my Java course

This output appears on the command/DOS window.
Note the following:

1. The word `public` means that everyone can access the class/method
2. Every application should have a `main` method. A method consists of one or more statements that fulfill a well-defined function. The word *void* implies that the method does not return anything. The argument `String[] args` means that `main` takes an *array* of `Strings` as an argument. Arguments are in parentheses immediately following the name of the method. Details of methods will be discussed later.
3. System.out refers to standard output (the command window in our example). `println` is a method associated with the standard output. This method takes a `String` as its argument and displays the string on the standard output.

VARIABLES AND IDENTIFIERS

A variable is a location in memory capable of storing a value. The *Head First Java* book likens variables to cups that come in different sizes (e.g. 12, 16, 20 oz) and have different names (e.g. small, medium, large or if you are a Starbucks fan, tall, grande, etc.) to identify them. That is, a variable has a size and an identifier. Variables are place holders for data. In reality, we deal with different types of data. Sometimes we use whole numbers (they have no fractional part!). For example, the number of students in my Java class, the number of items in stock, the number of books I own, the number of copies of *The Monk and the Philosopher* in your library, and the like are all represented by whole numbers. Such numbers are called integers. Floating point numbers such as the tax rate, the interest earned, amount of money in an account, etc. have a fractional part. Item descriptions, names of people, etc. may be represented as Strings. In summary, a variable must have a name, should represent a type, which in turn indicates a size.

Java is a strongly typed language. This means that every variable you use in your program must have a type.

Primitives represent very fundamental types of data. These are typically available in programming languages. Some of the primitives in Java are:

Integers

Name	Size	Range of values
byte	8 bits	-128 to 127
short	16 bits	-32768 to 32767
int	32 bits	-2147483648 to 2147483647
long	64 bits	……..

Floating Point

float	32 bits
double	64 bits

The AP subset uses the highlighted types . In addition to the above, you can represent true/false using **boolean**. The data type **char** is used to store characters. The size of **char** in Java is 16 bits (0 to 65535).

IDENTIFIERS

Names of variables, classes and methods **must** start with a letter (a-z, A-Z), dollar sign (**$**) or an underscore (_). Names may not start with a number. However, numbers may follow a letter, $ or an underscore. Also, make sure that the name is not one of Java's keywords (i.e. reserved words). Examples of keywords in Java are: **class, interface, int, private, try, throw, etc.** Follow the conventions listed below:

a. Class names: Start with an uppercase letter. Do not use names that already exist in the Java libraries. For example, String is a Java class. Do not use String as the name of your class.
b. Method and variable names: Start with a lowercase letter and capitalize the beginning of each new word. Some examples are: **`getCustomerName()`**, **`taxRate`**, **`balance`**, **`minimumBalance`**, **`getMinimumBalance()`**, **`computePay()`**
c. Names of constants: Constants are usually in uppercase. Example: **PI**

DECLARING VARIABLES

Examples:

```
int classSize; //declares a variable called classSzie that can store an integer value
classSize = 30; //here 30 is assigned to classSize
```

Note that the variable is to the left of the "=" and the literal value (30 in this example) is to the right of the "equal to" sign.

You could combine the two statements into one single statement as shown below:

```
int classSize = 30;
```

Examples of variable declarations:

```
double taxRate = 5.25;
int number = 23;
float interestRate = 5.75f; //note the suffix f. Without the suffix Java assumes a double
```

Note: You need the suffix f (or F)

```
boolean isAvailable = false;
boolean isPrime = true;
```

Mixing types

You can assign a smaller sized type to a larger one. For example, an int can be assigned to a double, a byte to an int, a short to an int and so on.

```
short s = 10;
int i = s; //this is ok because s has fewer bits than what int uses
```

However, larger types cannot be ***directly*** assigned to smaller type. Consider the following:

```
int i = 25.35; //this will not work
```

The literal (i.e. what is being assigned) is 25.35 which is a double (occupying 8 bytes or 64 bits). The variable on the left hand side of '=' is an int that takes up 4 bytes or 32 bits. Java recognizes that there is a potential loss of information when a larger data type is assigned to a smaller one. Hence, the compiler flags an error. You may force Java to go ahead with the assignment by ***casting*** or converting the double (i.e. 25.35) to an integer before assigning it. This is done as follows:

```
int i = (int) 25.35; //i will be equal to 25
```

Using char

A character is enclosed in single quotes. Character assignments are done as follows:
`char ch = 'A'; //assigns the letter A to ch`

Internally the character is mapped on to an integer value. You may display the integer value as follows:
`System.out.println((int) ch); //note that ch is first cast to an integer and then displayed`

What does the following display?

```
char ch = 'A' + 1;
System.out.println(ch);
```

This displays the character 'B'.

DECLARING A CONSTANT

Suppose we wish to declare a constant called PI (remember it is customary to use all caps for constants):
`final double PI = 3.14;` `// PI` cannot be changed, because the assignment is ***final***.

ESCAPE SEQUENCES
There are some special characters referred to as "escape sequences". Some of them are:
\b for backspace, \t for tab, \n for linefeed (newline), \r for carriage return, \" for double quote, \' for single quote, \\ for backslash

Examples:
```
System.out.println("Hello!\nNext Line\nLast line");
The display will be:
Hello!
Next Line
Last Line
```

The character `'\n'` stands for newline character and has the effect of inserting a newline in the string. `'\t'` inserts a tab. Suppose we wish to display "The Jungle Book" (with the quotes).

The following WILL NOT work:
`System.out.println(""The Jungle Book""); //gives an error`

The right way to do it is:
`System.out.println("\"The Jungle Book\"");`

Note that preceding " with a backslash forces `System.out.println` to display the double quote.

The forward slashes (//) are used to start comments. The compiler ignores comments (i.e. anything after //). Multiple line comments be shown between /* and */. An example is shown below:

```
/* This is a long comment……..
      More comments…..
      A few more….
      And we are done …..*/
```

There are also JavaDoc comments using //* and */. Read about them on the JavaDoc Info Sheet.

READING IN DATA FROM THE KEYBOARD

A. COMMAND WINDOW

For applications that use the DOS window, data may be read in from the keyboard using a class called ***Scanner***. The Scanner class is in the package java.util. Therefore, this package has to be imported in applications that use Scanner. The following program reads in a name and displays a greeting to the user.

```
import java.util.Scanner; //Scanner may be replaced by an asterisk if more than one class is needed
       // from the java.util package

import java.util.Scanner;
 public class HelloFriend {
       public static void main(String[] args)
       {
           String name;
           Scanner input = new Scanner(System.in);
            System.out.print("Enter name first and last: "); // always include a prompt
            name = input.nextLine();
           System.out.println(name + "! That is a nice name");
       }
}
//-----------------------------------------------------------------------------------------------------------------------------------------------
//Reading in a combinations of Strings (input.next() ) and integers (input.nextInt() )

import java.util.*;
public class TemperatureDOS {
       public static void main(String[] args)
       {
            double centigrade, fahrenheit;
            Scanner input = new Scanner(System.in);
            System.out.print("Enter temperature in centigrade:");
            centigrade = input.nextDouble();
            fahrenheit = 1.8 * centigrade + 32.0;
            System.out.println(centigrade + " degrees C = " + fahrenheit + " degrees F");
       }
}
//-----------------------------------------------------------------------------------------------------------------------------------------------
//Read in a centigrade temperature, convert it to
//fahrenheit and display the results
 import java.util.*;
 public class FindName {
       public static void main(String[] args)
       {
            String name;
             int age;
            Scanner input = new Scanner(System.in);
            System.out.print("What is your first name:");
            first = input.next (); // enters a String up to white space
           System.out.print("What is your age:");
            age = input.nextInt (); // enters a String up to white space
            System.out.println("Hello " + firstName + "You are "+ age +" years old);
       }
}
```

Discussion - Arrays

Arrays allow programmers to store multiple values under a common name. Of course we can always just create different variables using different names.

```
double number1;
double number2;
  :
double number50;
```

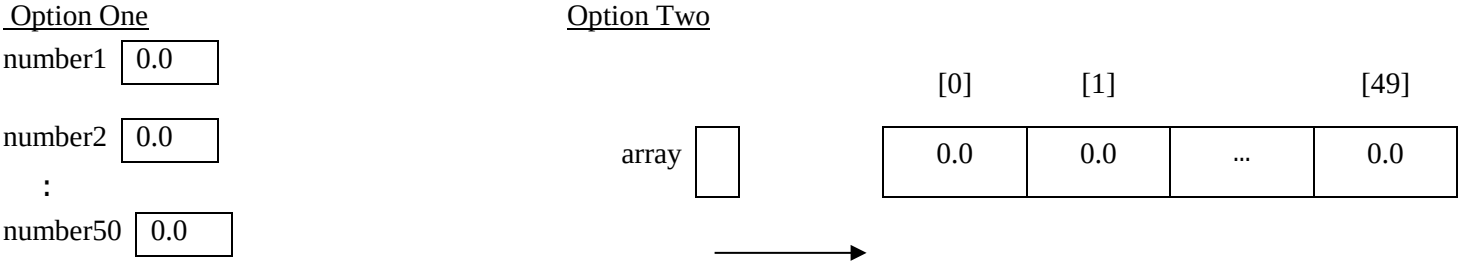
But this is terribly inconvenient. Fifty commands are required to set each of our variables to zero.

```
number1 = 0.0;
number2 = 0.0;
  :
number50 = 0.0;
```

A situation such as this begs for an array with a loop. Our fifty declarations are replaced with one declaration and our fifty initialization commands are replaced with one for-loop containing only one command.

```
double[] array = new double[50];
for(int k = 0; k < 50; k++)
   array[k] = 0.0;
```

Three lines, that's it, instead of a hundred. The beauty of an array is that it distinguishes different variables by an integer value only. Since an integer value can be controlled by a for-loop, large tasks require little code. Box diagrams for the two options shown above should clarify the distinction.



An array is made up of a certain number of *cells*. In our example there are fifty cells. These cells exist together as one logical unit where the fifty variables in option one exist as fifty separate entities. In Java, arrays are objects requiring the keyword `new` and accessed through references (arrow). The integer corresponding to each cell is that cell's *index*. Indices are always integers for all arrays.

Be careful! Arrays are zero-indexed so the first cell is cell number zero and the fiftieth cell is cell number forty-nine. You cannot change this fact so you must either adjust your thinking or accept frustration.

The name of an array does not have to be array, the type of data stored in an array does not have to be double, and the size of an array does not have to be fifty. Some examples:

```
int[] scores = new int[18];
char[] alphabet = new char[26];
String[] words = new String[numitems]; //numitems must be initialized
```

Discussion Min and Max

The Math class defines methods min and max for both the primitive types int and double.

```
a = Math.min(-2, -8);
b = Math.max(37., 1.0475);
c = Math.max(a, b);
d = Math.min(a, Math.min(b, c));
```

The min and max methods are *overloaded* because the same name, min or max, is used more than once.

```
public static int min(int arg1, int arg2)
{
   . . .
}
public static double min(double arg1, double arg2)
{
   . . .
}
```

The machine decides which of these two methods to use based on what type of arguments you pass. In the examples shown above the value of a will be an integer because –2 and –8, the two arguments to min, are both integers. The machine will call the integer version of min because of these two integer arguments and that min method will return an integer value.

If either of the two arguments is a double then the min method for doubles is called and a double value is returned. Thus b, c, and d are all doubles. Note carefully the code for finding the min of three values.

Be careful! You can store an integer value in a double but you can't store a double value in an integer. In the examples shown above it is necessary that b, c, and d are all doubles because they store double values. But a could be declared either as an `int` or a `double`. If a were declared as a double then it would store the "integer" value –8.0.

The machine will automatically convert integer values to doubles but it will not convert double values to integers unless you explicitly say to do so. For instance, it would be an error to declare b as an int. The machine thinks that the result of `Math.max(37., 1.0475)` will be a double value. If you try to store a double value in an int then the decimal part will be lost generating a possible loss of precision error.

Be careful! The error is for possible loss of precision. In our example the max value that b gets is 37.0 and there would be no harm, no loss of precision, storing 37.0 in an int. The machine doesn't know this.

To explicitly store a double value in an int you must *cast*. For instance:

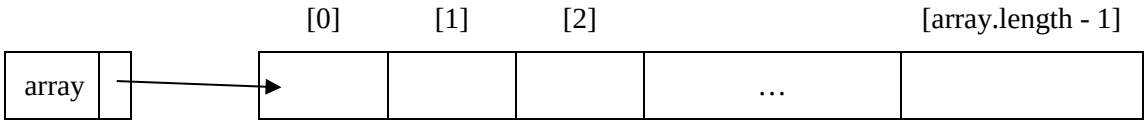
```
int n = (int)Math.max(0.25, 0.2);
```

Here n will get the value zero. Since one fourth is bigger than one fifth the max method return 0.25, which cannot be stored in an int. But the cast to `int` truncates this decimal, dropping the .25 and leaving us with only 0.

Array Exercises (5 points)

Answer these questions. **If an array index goes out of bounds or a variable is out of scope, note it** in your answer and show the output as it existed at the time of the error.

These code fragments all manipulate the array's **values** through the array's **index** numbers. The only way to figure out what's happening to the values and the indexes is to draw a picture of each array:



1) Write the contents of `circle` after this code has run:

```
double [] circle = {1.0, 10.0, 1.0, 0.0};
for (int index=0; index < circle.length; index++)
  circle[index]=circle[index]*circle[index]*Math.PI;
```

[0]	[1]	[2]	[3]
1.0	10.0	1.0	0.0

2) Write the contents of `array` after this code has run:

```
double [] array = {1.0, 5.0, 2.0, 3.0, 5.0};
for (int pos = array.length - 1; pos > 0; pos--)
    array[pos] = array[pos - 1];
```

[0]	[1]	[2]	[3]	[4]
1.0	5.0	2.0	3.0	5.0

3) Write the contents of `myArray` after this (useless) code has run:

```
int [] myArray = {4, 9, 3, 5, 6, 2, 1};
for (int i=0; i<= myArray.length - 1; i++)
    if (myArray[i] > myArray[i + 1])
        {
            int temp = myArray[i];
            myArray[i] = myArray[i + 1];
            myArray[i + 1] = temp;
        }
```

[0]	[1]	[2]	[3]	[4]	[5]	[6]
4	9	3	5	6	2	1

temp | |

4) What is the output of this (useless) code?

```
int [] myList = {1, 2, 3, 4, 5};
changeArray(myList);
System.out.print("The changed list is ");
for (int i=0; i< myList.length; i++)
    System.out.println(myList[i] + " ");

public static void changeArray(int[] tempList)
{
   for (int i=0; i < tempList.length; i++)
        tempList[i] = tempList[i] + tempList[2];
}
```

[0]	[1]	[2]	[3]	[4]
1	2	3	4	5

Number Array Lab
Sum, Average, Min, Max (10 points)

Objective
Calculate using data in an array.

Background
The main method must be tagged static because the java command for running programs only knows to call a method with precisely the signature:

```
public static void main(String[] args)
```

The length of an array cannot be changed once the array is created. Use the public variable `length` to find the length of an array. For instance:

```
int n = (int)(Math.random() * 51 + 25);
double[] array = new double[n];
System.out.println("The length is " + array.length + ".");
```

```
// this creates integers [25, 75]
// the general equation for random integers
// between [low, high] is
// (int)(Math.random() * (high-low+1) + low);
```

Specification
Create a class called `ArrayCalculations`. Create an array of 20 random integers between [1, 500]. Find the sum and average of the data. Find the smallest value and the largest value. Display these four results correctly labeled using System.out.

String Array Lab
Harry Potter (10 points)

Objective
Manipulate a String Array

Create a class called **`HarryPotter`**. In a `HarryPotter` class declare an array of **`names`** suitable for holding 5 names. **Make sure you label the output** (eg. `The last element of the array is Snape.`)

What does the statement **`System.out.println(names.length);`** do?
(It returns how many elements are in the names array)

What does the statement **`System.out.println(myName.length());`** do?
(It returns how many characters are in the String myName)

a. Use five assignment statements or an initializer list to assign the names "Hermione", "Harry", "Ron", " Dumbledore", and "Snape" to the **`names`** array.

b. Print out **only** the last element of the **`names`** array.

c. Use a for-loop to print the **`names`** array out in reverse order, one name per line followed by the length of the name String.

d. Add a **`Scanner input`** to your lab so we can read data from the user.

e. Create a new array **names2** that can hold ten names. Copy the first 5 names from the **names** array to the **names2** array and then add 5 more wizards (good, bad, or Hogwarts professors or students) from Harry Potter to the **names2** array. (It is a good idea to use a loop to copy the names and another loop to read in the 5 new names, The **`Scanner`** method **`nextLine()`** reads a string from the keyboard (**If you can't think of any more Harry Potter wizards, Google**) . **When you are finished there should be two arrays – names and names2.**

f. Print one of the characters from the **names2** array at random. (Hint: Pick a random subscript (0-9) and print out the name at that subscript.)

g. Use a for-loop to print a numbered list of the names in the **names2** array to the console. For example, the output might look like:

```
1) Harry
2) Ron
3) Hermione
4)
5) ...
6)
7)
8)
9)
10) Snape
```

h. Write a new for-loop that computes the average name **length** of a list of names and prints it out. (For example `The average length of a name in the first list is 6.2.`) There is a `length()` method in the String class.

Creating your own Classes and Objects Notes -

When you write your own classes, you can include the following methods to accomplish the tasks explained:

- **Constructor(s**) : to initialize the instance variables to the given input argument values
- **Mutator method(s):** one method for each instance variable to change its value to the input argument's value
- **Accessor method**(s): one method for each instance variable to return its value to the client class
- **equals method**: to compare the current object with the object in argument list and return true for equality (define what equals means for you in the equals method's body) and false otherwise
- **toString method:** to return a String representation of the object (possibly contains the necessary instance variable values) to be used along with String concatenation and print statements

Sun's Java style guidelines suggest putting fields at the top of the class, followed by constructors, followed by methods.

```
public class <class name> {
// fields
private <type> <name>;
private <type> <name>;
...
// constructor
public <class name>(<type> <name>, ..., <type> <name>) {
<statement>;
<statement>;
...
<statement>;
}
...
// methods
public <type> <name>(<type> <name>, ..., <type> <name>) {
<statement>;
<statement>;
...
<statement>;
}
...
}
```

For example:

```
// A Participant object represents data about one person in a
// Body Mass Index (BMI) study program.

  public class Participant {
    private String name;
    private double height;
    private double weight;

// Constructs a participant with the given state.
   public Participant(String name, int height, int weight) {
     this.name = name;
     this.height = height;
     this.weight = weight;
  }
  // Returns the participant's body mass index (BMI).
  public double getBMI() {
     return weight / (height * height) * 703;
 }
 // Returns the participant's height.
   public double getHeight() {
```

The "this" is optional if the parameter names are different from the instance variables

accessor methods – returning the instance variables

```
    return height;
 }

// Returns the participant's name.
   public String getName() {
     return name;
   }

// Returns the participant's weight.
    public double getWeight() {
      return weight;
    }

// Returns the String will all information you want printed in a print statment.
    public String toString() {
      return "Name: "+name+"\nHeight: "+height+ "\nWeight: "+weight;
    }

// Returns true if two objects are equal.  All fields must match.
    public boolean equals(Participant p) {
      return (name.equals(p.name) && height==p.height && weight==p.weight);
    }
}
```

You can also have mutator methods (or set methods that reassign the values of your instance variables.

Note: when you compare Strings (or objects) you use .equals and when you use primitives (int, double, boolean, char) you use = =

To create a class that uses the Participant method make a ParticipantTester with a main method. Remember the `Participant` and the `ParticipantTester` classes are stored in **separate files** in the **same directory**.

```
public class ParticipantTester {
   public static void main(String[] args)
    {
        Participant john = new Participant("John Smith", 70, 180);
        Participant jane = new Participant("Jane Doe", 66, 130);
        Participant jim = new Participant("John Smith", 60, 180);
        System.out.println(john.getName()+"'s BMI is "+john.getBMI());
        System.out.println(jane.getName()+"'s BMI is "+jane.getBMI());

        System.out.println(john);
        System.out.println(jane);
        System.out.println(jim);
        if(jim.equals(john))
               System.out.println(john.getName()+" is the same as "+ jim.getName());
        else System.out.println(john.getName()+" is not the same as "+ jim.getName());

        Participant[] people = new Participant(5);
        people[0] = new Participant("Sam", 62, 160);
        people[1] = new Participant("Brad", 62, 200);
        people[2] = new Participant("Jan", 62, 120);
        people[3] = new Participant("Tom", 62, 150);
        people[4] = new Participant("Fred", 62, 210);

       double sumHeight=0; sumWeight=0;
        for (int j = 0; j < people.length; j++){
          System.out.println(people[j]);
          sumHeight += people[j].getHeight();
          sumWeight += people[j].getWeight();
        }
        System.out.println("The average height is "+ sumHeight/people.length);
```

```
        System.out.println("The average weight is "+ sumWeight/people.length);
    }
}
```

Summary

- Object-oriented programming is a different philosophy of writing programs that focuses on nouns or entities in a program, rather than verbs or actions of a program. In object-oriented programming, state and behavior are grouped into objects that communicate with each other.

- A class serves as the blueprint for a new type of object, specifying their data and behavior. The class can be asked to construct many objects (also called "instances") of its type.

- The data for each object is specified using special variables called fields.

- The behavior of each object is specified by writing instance methods in the class. Instance methods exist inside an object and can access and act on that object's internal state.

- A class can define a special method called a constructor that creates and returns a new object and initializes its state. The constructor is the method that will be called when external client code creates a new object of your type using the new keyword.

- Usually one object can communicate with others despite not knowing all details about how the other objects work, a principle known as abstraction. Most objects protect their internal data from unwanted external modification, a principle known as encapsulation. Encapsulation is provided by declaring fields with the private modifier, which prevents other classes from directly modifying their values.

- Two common categories of object methods are accessors and mutators. Accessors provide us with information about the object, such as the length method of a String object or the getX method of a Point object. Mutators allow us to modify the state of the object, such as the translate method of a Point object.

- Objects can be made easily printable by writing a toString method. Objects can be made testable for equality by writing an equals method.

- The keyword this can be used when an object wishes to refer to itself. It is also used when a class has multiple constructor and one constructor wishes to call another.

- An array of objects can be created by making an array of your class type `Participant[] people = new Participant(5);` You must also instantiate each Participant in the array

Summer Assignment Part 2

Student Lab and StudentArrayLab **(10 points)**

Objective

Calculate using data in an array.
Write a program that contains an array of Students.

a. First create a Student class

Make a `Student` class that has

- the student ID number,
- name,
- the students' grade point average.
- Write a constructor with parameters to instantiate an object.
- Write a toString() method to display one student object.
- Write a get method for each instance variable (The set is optional)

Test your class by making a `StudentTester` class

b. Now make a new class `StudentTester` class which creates 5 `Student` objects. You may create the objects individually. For example s1 = new Student(1234, "Mary Smith", 3.8);

- Print the first object with all the information
- Print the second object's ID
- Print the third object's Name
- Print the fourth object's GPA
- Print all the objects

What to turn in on the <u>first day</u> of class. Partial Credit will be given.
Put everything in a Word document and print it out 2 pages per sheet

1. Array Exercises with the answers
2. A listing of Number Array Lab java code and the output of 3 runs
3. A listing of the String Array Lab java codes and the output of a run (other runs would be identical)
4. A listing of the Student class java code (it does not run)
5. A listing of Student Tester class java code and the output of a sample run.