Section 1 - Introduction and Resources

5A.1.1 Overview

This week you will take a major step in your ability to program. You will be introduced to the important intermediate concept of ***methods***, known in some circles as ***functions***. You will learn about ***parameter passing***, ***local variables***, ***return values*** and other topics related to methods.

An Optional Mathematical Module C

For those of you interested in mathematics, I have created an optional module C that discusses numerical solutions to differential equations. You have enough programming at this point to be able to apply it to such applications.

5A.1.2 Assigned Reading

As usual, read the week's modules and paste every code sample into your IDE so you can test and expand upon it. Do this *before looking at the assignment*. If you want more source material after reading the modules, look up any new terms in your textbook index, and see what it has to say about these things.

Section 2 - Methods (Functions)

5A.2.1 Rationale

We have been using ***methods*** by invoking them (**getline(), isalpha(),** etc.) So far, you could only invoke ***methods*** that I told you existed. This week you get to create your own ***methods***, and you get to decide how they are called.

Methods, also known as ***functions*** in C++, are important for many reasons, but the one-word justification for them is ***modularity***. It would be nearly impossible to write and maintain medium-sized or large programs without breaking them down into palatable and debug-able parts. We will soon see that these parts consist of ***classes*** (which we have briefly touched on), and the most important aspects of the ***classes*** which are the ***methods***. While we won't delve into ***classes*** until next week, we can start learning ***methods*** immediately.

5A.2.2 Introducing Methods

A ***method*** is a sub-program that is dedicated to performing a relatively small task. By ***relatively small,*** I mean anything that is easily describable in a sentence or two, doesn't take too many statements to execute (subjective), and is easily tested.

It may also be a task that is called repeatedly by your program, sometimes with different data being sent to the ***method*** for processing.

We will learn ***methods*** gradually over the next two weeks. The first example will not obey all of the above principles. For one thing one of the ***methods*** will perform a large task and take a lot of statements. This will be fixed in a subsequent version of the program when we learn how to break the task down better.

5A.2.3 A Mortgage Calculator

We're going to write our own mortgage calculator - one that computes monthly payments based on interest, term of loan and principal amount.

Here is a sample of what we are after:

```
 The following program will calculate the
monthly payment  required for a loan of D dollars
over a period of Y years at an annual
interest rate of R%.
---------------


Enter amount of the loan. (only use numbers,
please, no commas or characters like '$')
Your loan amount: 300000

Now enter the interest rate (If the quoted rate is  6.5%,
for example, enter 6.5 without the %.)
Your annual interest rate: 5.75

Enter term of the loan in years: 15


Monthly Payment: 2491.2302610589704

Thanks for using the Foothill Mortgage Calculator.
We hope you'll come back and see us again, soon.
```

We will use the following formula to compute the payment. In this formula

- *L* is the amount of the loan
- *c* is the monthly interest rate (we'll have to convert from annual to monthly)
- *n* is the number of months (we'll have to convert from years to months)
- *P* is the monthly payment we are seeking

Here is the formula. Please don't panic. You don't have to memorize it. You will learn how it gets turned into a Java formula. Ready? Hold your breath:

$$P = L\frac{c(1+c)^n}{(1+c)^n - 1}$$

or, written as a single line:

$$P = L[c(1 + c)^n]/[(1 + c)^n - 1]$$

In the following pages we will write increasingly smarter versions of the program that uses methods to accomplish all of this.

5A.2.4 Simplifying main()

One of the main reasons for writing and using methods is that it makes our programs easier to read and understand. Let's start by seeing how our **main()** ***method*** (the only ***method*** we have written up until now) will look once we have written some other ***methods*** to make it look clean, short and easy to understand.

```
public static void main(String[] args)
{
   stateInstructions();
   getInputAndComputeMonthlyPayment();
   sayGoodbye();
}
```

Before continuing, let's compare this **main()** to the screen shots above.

We see a ***method call*** to **stateInstructions()**. This method is pretty clear, but can you see which of the screen output (in my screen capture image) this method controls? If you answered only the first paragraph, you are correct. It places the overall introduction on the screen.

Next we ***invoke a method*** called **getInputAndComputeMonthlyPayment().** Which paragraphs do you think that one controls? It will place on the screen the *four middle paragraphs*: the three inputs and the one result that reports the monthly payment. This is the ***method*** that I said will be too large and do too much in this first incarnation. That's okay, though, because right now we won't have the machinery to break it down. We will do so, in future revisions.

Finally, there is the **sayGoodbye()** ***method call***. This one obviously presents the final paragraph on the screen, thanking the user and signing off.

Our first task will be to learn how to define (write) these three custom-made methods. This will be the first time that we have ever defined our own methods that we can use (i.e., ***invoke***) inside **main().**

Section 3 - Defining Methods

5A.3.1 Defining the First Method

To keep things easy to read, I'll just present the definition of the first method, **stateInstructions()** so we can see where it goes in the ***class***. The ***method definition*** comes right after the **main()** method, and still inside the one and only ***class***, **Foothill**:

```
import java.util.Scanner;

public class Foothill
{
   // main method
   public static void main (String[] args)
   {
      stateInstructions();
      getInputAndComputeMonthlyPayment();
      sayGoodbye();
   }

   // gives an overview to user
   public static void stateInstructions()
   {
      String instructions;
      instructions =
         "The following program will calculate the \n"
         + "monthly payment  required for a loan of D dollars \n"
         + "over a period of Y years at an annual \n"
         + "interest rate of R%.";
      System.out.println(instructions);
      System.out.println("---------------\n");
   }

   // does all the work - gets input, computes and reports answer
   public static void getInputAndComputeMonthlyPayment()
   {
      // Body of this method omitted for now
   }

   // sign off
   public static void sayGoodbye()
   {
      // Body of this method omitted for now
   }
}
```

The ***definition*** of the **stateInstructions()** ***method*** looks a lot like the beginning of the ***definition*** of our older **main()**. Also, the definition of each of the methods (including the ones we did not fully list yet) apparently begins with a ***method header*** that contains the words **public static void** followed by the user-defined ***method name,*** and then empty parentheses, as in:

```
public static void stateInstructions()
```

The word **public** means that this method can be called from anywhere in the program.

The word **void** means that the method does not return a value directly to the **main**().

The word **static** means that this method can be ***called*** from inside **main**() either

- without any identifier in front of it, that is by just stating its name, stateInstructions(); or
- by using the class name to reference the method as in **Foothill.stateInstructions();**

While it's a little early to worry about the exact meaning of **static**, I'll tell you that **static *methods*** cannot be invoked using specific object names in front of them. They can only be invoked using the ***class name*** or no name at all (depending on the context). Sometimes I refer to ***static methods*** by calling them ***class methods*** or even ***static class methods***.

Method definitions can appear in any order inside the class.

A method is ***called*** (or ***invoked*** or ***used***) when its name appears in **main**(). That is, the program begins at the first line in **main()** and only gets *into* a ***method*** if there is a line in **main()** that has the ***method's*** name on it. That means that the method **stateInstructions()** is actually executed *only* if there is a line in **main**() that has the word **stateInstructions()** in it.

When a ***method*** is ***called***, the program execution switches to the top of the ***method definition*** and continues until the ***method*** ends, or a **return** statement is reached. When either of these things happens, the control passes back to **main()**, to the next statement after the ***method call***.

In **main()** you see three ***method*** calls. This breaks **main**() into three modular pieces. We could give different teams responsibility for designing and debugging different methods, but in such a small program, this is not needed.

When you look at **main()** now, it is easy to see where things happen. If there is a problem with the opening remarks, or the arithmetic, we know which method to look into to fix it.

Section 4 - The Mortgage Calculator

5A.4.1 The Program Listing

Now we can look at the full program. You can copy/paste this into your Java IDE and try it out. Make sure your class is called **Foothill**. As I said, the definition of each of the three methods appears right after the **main()**.

Here is the listing. See how much you can understand before reading on.

```
import java.util.Scanner;

public class Foothill
{
   // main method
   public static void main (String[] args)
   {
      stateInstructions();
      getInputAndComputeMonthlyPayment();
      sayGoodbye();
   }

   // gives an overview to user
   public static void stateInstructions()
   {
      String instructions;
      instructions =
         "The following program will calculate the \n"
         + "monthly payment  required for a loan of D dollars \n"
         + "over a period of Y years at an annual \n"
         + "interest rate of R%.";
      System.out.println(instructions);
      System.out.println("---------------\n");
   }

   // does all the work - gets input, computes and reports answer
   public static void getInputAndComputeMonthlyPayment()
   {
      Scanner input = new Scanner(System.in);

      String prompt, strUserResponse;
      double dblPrincipal, dblRate, dblYears, dblMoRt, dblMonths;
      double dblTemp, dblPmt;

      // get principal
      prompt = "\nEnter amount of the loan. (only use numbers, \n"
         + "please, no commas or characters like '$')\n"
         + "Your loan amount: ";
      System.out.print(prompt);
      strUserResponse = input.nextLine();
      dblPrincipal = Double.parseDouble(strUserResponse);

      // get interest
      prompt = "\nNow enter the interest rate (If the quoted rate is "
         + " 6.5%, \nfor example, enter 6.5 without the %.)\n"
         + "Your annual interest rate: ";
      System.out.print(prompt);
      strUserResponse = input.nextLine();
      dblRate = Double.parseDouble(strUserResponse);

      // get length of loan
      prompt = "\nEnter term of the loan in years: ";
      System.out.print(prompt);
      strUserResponse = input.nextLine();
      dblYears = Double.parseDouble(strUserResponse);
```

```
      // convert years to months
      dblMonths = dblYears * 12;

      // convert rate to decimal and months
      dblMoRt = dblRate / (100 * 12);

      // use formula to get result
      dblTemp = Math.pow(1 + dblMoRt, dblMonths);
      dblPmt = dblPrincipal * dblMoRt * dblTemp
         / ( dblTemp - 1 );

      System.out.println("Monthly Payment: " + dblPmt);
  }

   // sign off
   public static void sayGoodbye()
   {
      String signoff;
      signoff =
         "\nThanks for using the Foothill Mortgage Calculator. \n"
         + "We hope you'll come back and see us again, soon.";
      System.out.println(signoff);
   }
}
```

5A.4.2 Discussion of the Program

As we have been saying, everything in this program, as in our previous programs, appears inside our one and only ***class***. We continue to call that class **Foothill**. By doing all the work inside our non-main methods, we have made ***main()*** very simple.

We already discussed what **stateInstructions()** does. You can see that **sayGoodbye()** works exactly the same way, but with a different phrase to send to the screen. That leaves the ***method*** **getInputAndComputeMonthlyPayment()** to discuss.

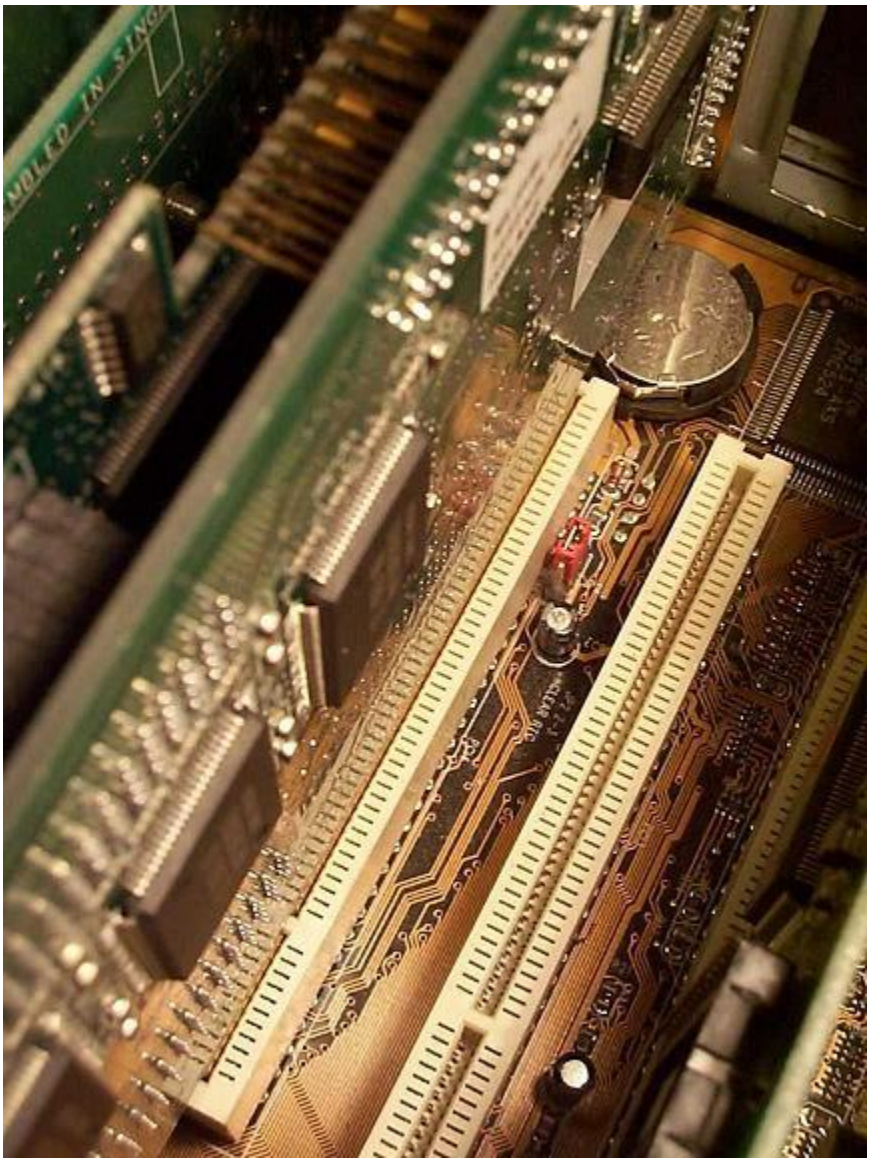
- First, notice that there is nothing major inside this ***method*** that is new (not counting **Math.pow()** which we'll get to in a moment). These are all statements that we could have placed directly in **main()** and we would have (or should have) been able to understand them even last week.
- Second, notice that this ***method*** declares ***variables*** (both **String** objects and ***primitive*** **doubles**). In fact, all of our ***methods*** declare variables inside their definitions.
- Third, because we don't know (yet!) how to transfer a variable or its value from one ***method*** to another, we had to do a lot inside **getInputAndComputeMonthlyPayment().** It would have been better to have one ***method*** get the input, a second compute the payment and a third report the result. We'll get there soon.
- Fourth, there is a method call inside the definition of **getInputAndComputeMonthlyPayment()** -- a call to a method **Math.pow()**. This is a function that raises one number, base, to an exponent, power. **Math.pow(A, B)** will raise the number A to the power B. This is useful if B is a large value or a fractional number.

However, if B is 2 or 3, it is easier to use ordinary multiplication rather than invoking this function.

- Fifth, because the formula was a substantially complicated one, I broke up the computation into a few lines. If you study these lines and compare them to the formula I gave earlier, you should be able to see that this gives the right answer.

That's all I can think to say about this program. You should meditate on these comments as you run the program and examine its source. You might try changing the wording around but you'll have to keep the math as is if you want it to be accurate. While you're at it, you can see if you can afford that home you want.

Section 5 - Locals and Functional Return



5A.5.1 Local Variables in Methods

The words **prompt** and **dblRate** in the ***method*** **getInputAndComputeMonthlyPayment()** are ***variable*** (or ***object***) ***identifiers***. (**prompt** is actually a **String** ***reference***, as we learned last week, but we informally call it a ***variable*** sometimes.) Up until now, we have had only one method, **main**(), so the ***scope*** of these variables, that is, the place where they can be used, was not an issue: we declared the ***variable objects*** at the top of the program and used them whenever and wherever we wanted. Now that we have multiple method definitions, we have to be more careful.

Variables declared inside a ***method*** are ***local*** to that method, which means their names are known *only within that one method*. Other *methods* can declare objects with the same names, but they will have nothing to do with the **prompt** and **dblRate** in the method **getInputAndComputeMonthlyPayment().**

This is one of the most important simple facts about methods.

If the following two methods appear in the same program:

```
void dog()
{
   int bird;
   // method statements ...
}

void cat()
{
   double bird;
   // method statements ...
}
```

then the two **bird** objects are distinct and unrelated.

This motivates a question:

If ***objects*** in different methods are distinct, how do you design programs so that different methods operate on the *same* data?

It seems reasonable that we would want a method to calculate a number, and return that number back to **main**() which would, in turn, pass it on to a second method for further processing. (Excellent question -- what's your name?)

5A.5.2 Returning Values from Methods

There are three ways that information can be transferred between **main**() and the ***methods*** that it calls. This section is devoted to the first of these ways: ***functional return***. Remember - ***methods***

are also called ***functions***, so that's where the term ***functional return*** arose. ***Methodal return***, just didn't sound right.

You can use ***functions*** to return values to the **main()** through the ***functional return*** by taking the following steps:

1. **Declare the data type of the return object in the method header.** If we want the method to ***return*** an **int** then the header would be written:

```
2.    int abalone()
3.    {
4.    }
```

   If we want it to ***return*** a **double** then it might look like this:

```
static double abalone()
{
}
```

   Notice that we can have other modifying words like **static** before the method name in addition to the return type.

5. **Supply a "return" statement** in the method, which describes the actual value that is to be sent back, or ***returned*** to the **main**().

```
6.    double abalone()
7.    {
8.       // method statements ...
9.
10.        return 1.020203;  //  use parens if you want
11.
12.        // possibly other method statements ...
13.    }
```

   The **return** statement can appear anywhere in the method. For instance, if it appears inside a ***while loop*** or ***if statement,*** then there would probably be other statements after it

```
if (pistol == loaded)
   return 1.020203;
// otherwise we continue with statements
```

   There can be more than one ***return*** statement; the first one that is encountered on a given run causes the method to return. The ***return value*** can come from a constant, object or even expression:

```
if (pistol == loaded)
   return 1.020203;
else if (pistol == empty)
   return x;
else if (pistol == broken)
   return (1.020203 + x)/pistol;
```

However, complex expressions in **return** statements are not recommended because they make debugging difficult.

You may be wondering how **main**() receives and uses this ***functional return*** value. When the method is called, it begins executing. When it returns, control passes back to the **main**(). At that point any ***functional return*** value *replaces* the ***method call.*** If the ***method call*** is inside a larger expression, then this causes the returned value to be used in place of the method call inside that expression

Examples:

```
y = cat();
z = (1.5 + cat()) / y;
p = (cat() + 1) / (cat() + 2);
cat();
```

In the first example ...

... **cat**() is called and the value it returns replaces the call ***cat()***. Let's say it returned **1.020203**. Then after the method was complete and returned to the **main**(), the calling statement would be

```
y = 1.020203;
```

the value replacing the method call.

In the second example ...

... the same thing happens, but the ***functional return*** replaces the **cat()** which is inside a more complex expression.

In the third example ...

... **cat**() is called twice in a single statement in the third example, but they are separate calls, so the method is entered and exits twice, each time replacing the **cat()** with the resultant value (which might be different each time it is called!).

In the last example ...

... **cat**() is called, the ***functional return*** value replaces it, and nothing is done with the value. It evaporates. It is not an error to throw away a ***functional return*** value, however if you do it always for a particular method, then perhaps you should change its return type to **void** and not bother supplying a value in its return statement.

If a ***void*** method, i.e. a method whose return type is void:

```
void foo();
```

has a ***return*** statement, then it cannot have a value after the return. Conversely, a ***non- void*** method must have a ***return*** statement, and it must contain a ***value***.

5A.5.3 Methods Calling Methods

There is no law that says all methods must be called from **main**(). Up to this point in the lecture, I spoke as if methods were, in fact, called from **main**(), but they may be called from other ***methods***:

```
void main()
{

   spaceControl();

}

void spaceControl()
{
   float y;

   y = desk();

}

float desk()
{
   float x, y, a;

   // statements using x and y ...

   a =  x + y;
   return a;
}
```

Terminology

We don't always know, by looking at a method definition, from where it will be called. Therefore, rather than assuming that it is main(), we often say that the method is *called from* and *returns to* ***"the client"***, or ***"the caller."*** That client or caller could be main(), but it could also be some other method.

Section 6 - Mortgage Version 2

5A.6.1 Using the Functional Return in Mortgage Calculator

Let's modify the earlier program to use a functional return value. We change the large function:

```
import java.util.Scanner;

public class Foothill
{
   // main method
   public static void main (String[] args)
   {
      double answer;

      stateInstructions();
      answer = getInputAndComputeMonthlyPayment();
      System.out.println("\n\nMonthly Payment: "
            + answer);
      sayGoodbye();
   }

   // gives an overview to user
   public static void stateInstructions()
   {
      String instructions;

      instructions =
         "The following program will calculate the \n"
         + "monthly payment  required for a loan of D dollars \n"
         + "over a period of Y years at an annual \n"
         + "interest rate of R%.";
      System.out.println(instructions);
      System.out.println("---------------\n");
   }

   // does all the work - gets input, computes and returns answer
   public static double getInputAndComputeMonthlyPayment()
   {
      Scanner input = new Scanner(System.in);
      String prompt, strUserResponse;
      double dblPrincipal, dblRate, dblYears, dblMoRt, dblMonths;
      double dblTemp, dblPmt;

      // get principal
      prompt = "\nEnter amount of the loan. (only use numbers, \n"
         + "please, no commas or characters like '$')\n"
         + "Your loan amount: ";
      System.out.print(prompt);
      strUserResponse = input.nextLine();
      dblPrincipal = Double.parseDouble(strUserResponse);
```

```
      // get interest
      prompt = "\nNow enter the interest rate (If the quoted rate is "
         + " 6.5%, \nfor example, enter 6.5 without the %.)\n"
         + "Your annual interest rate: ";
      System.out.print(prompt);
      strUserResponse = input.nextLine();
      dblRate = Double.parseDouble(strUserResponse);

      // get length of loan
      prompt = "\nEnter term of the loan in years: ";
      System.out.print(prompt);
      strUserResponse = input.nextLine();
      dblYears = Double.parseDouble(strUserResponse);

      // convert years to months
      dblMonths = dblYears * 12;

      // convert rate to decimal and months
      dblMoRt = dblRate / (100 * 12);

      // use formula to get result
      dblTemp = Math.pow(1 + dblMoRt, dblMonths);
      dblPmt = dblPrincipal * dblMoRt * dblTemp
         / ( dblTemp - 1 );

      // now that we have computed the payment, return it
      return dblPmt;
  }

   // sign off
   public static void sayGoodbye()
   {
      String signoff;
      signoff =
         "\nThanks for using the Foothill Mortgage Calculator. \n"
         + "We hope you'll come back and see us again, soon.";
      System.out.println(signoff);
   }
}
```

Notice that **main**() now has its own ***local variable***, **answer** which is charged with the responsibility of catching the ***return*** value from the **getInputAndComputeMonthlyPayment()** ***method***. It is then used for the output.

We are also doing something that is going to be very important in all future assignments: ***separating the calculation operations from the I/O (Input/Output) operations***. I am very much an advocate of not doing both **I/O** and calculations in the same method. When you see me do so, it is only because we have not (or had not) the tools to accomplish a clean separation. But once you know how to pass parameters and values between one method and another, I will expect you to never put too many different tasks in a single method, especially not calculation and I/O.

For the time being, we don't know how to pass two pieces of information to a method so we couldn't separate the input function from the number crunching function. But we will get there next time.