Solution to Method Development

1. Tracing Programs

For each program below, show what is displayed on the screen when the code executes.

```
import java.util.Arrays;
public class ReferenceSemantics
{
  public static int mystery(int a, int [] b)
  {
    a++;
    b[0] = 0;
    b = new int[4];
    return a / 2;
  }

  public static void main(String [] args)
  {
    int x = 10;
    int [] y = {3, -3, 3};

    x = x + mystery(x, y);

    System.out.println(x);
    System.out.println(Arrays.toString(y));
  }
}
```

screen

```
15
[0, -3, 3]
```

```
public class NestedCalls
{
  public static int mystery1(int a)
  {
    System.out.println("m 1");
    return a / 2;
  }

  public static String mystery2(int b)
  {
    String s = "m 2";
    System.out.println(s);
    for(int i=0; i<b ; i++) {
      s = s + " " + i;
    }
    return s;
  }
  public static void main(String [] args)
  {
    System.out.println("main");
    System.out.println(mystery2(mystery1(4)));
    System.out.println("end main");
  }
}
```

```
screen
main
m 1
m 2
m 2 0 1
end main
```

2. Method development walkthrough

a. Start with a method signature: what's the return type, how many arguments does it take, and what is the data type of each argument?

```
// I'm going to skip the "public class <classname>" for this
// and following problems

// the method returns the number of i's, so it returns an int
// you have to give it a String in order for it to work, so it
// should have a String argument
public static int getNumLetterI(String s)
{
}
```

b. Next, decide on an *algorithm*, or recipe, for taking the inputs to this command (the arguments) and computing the outputs (the return value). Write down a step-by-step procedure, in English, for computing the number of 'i's in a String.

```
variable count starts at 0
variable index starts at 0
while index is less than the length of the String s
      check the letter at position index of s
      if the letter at position index is an 'i':
            increase count by 1
      increase the index by 1
```

c. Next, translate your algorithm into code. Don't forget a return value at the end of the method (and make sure the data type of the return value matches the return type in your method signature).

```
public static int getNumLetterI(String s)
{
      int count = 0;
      int index = 0;
      while(index < s.length() ) {
            if(s.charAt(index) == 'i') {
                  count++;
            }
            index++;
```

```
	}
	return count;
}
```

d. Create a program whose main() method repeatedly reads in Strings from the user and prints out how many ‘i’s are in that String, until the user enters a word with no ‘i’.

```
public static void main(String [] args) {
	Scanner kb = new Scanner(System.in);
	String s = kb.next();
	int numI = getNumLetterI(s);
	while(numI>0) {
		System.out.println(numI);
		s = kb.next();
		numI = getNumLetterI(s);
	}
}
```

e. Modify the method from (c) so that it accepts an argument of type char as well as of type String, and counts how many of that character appear in the String.

```
public static int getNumLetter(String s, char letter)
{
	int count = 0;
	int index = 0;
	while(index < s.length() ) {
		if(s.charAt(index) == letter) {
			count++;
		}
		index++;
	}
	return count;
}
```

f. Have the main() method read in a String and a letter from the user, and print out how many of the letter appear in the String.

```
public static void main(String [] args) {
	Scanner kb = new Scanner(System.in);
	String s = kb.next();
	char letter = kb.next().charAt(0);
	int numLetter = getNumLetter(s, letter);
	while(numLetter>0) {
		System.out.println(numLetter);
		s = kb.next();
		letter = kb.next().charAt(0);
		numLetter = getNumLetter(s, letter);
	}
}
```

3. Another method development

a. Write a method to compute whether an integer is prime or not. Start with the method signature: what is the return type, how many arguments does it need, and what type are the arguments?

```
// this method returns either a true or a false (prime or not).
// so it must have return type Boolean.
// you have to give it a number for it to check for primality,
// so it needs an int argument.
public static boolean isPrime(int num)
{

}
```

b. Next, decide on an *algorithm*, or recipe, for checking whether a number is prime. Without thinking about the code, write down a simple step-by-step procedure for deciding whether or not a number is prime.

```
input:  num, an int
output:  whether num is prime or not (a boolean)
if num<2 return false
variable potentialFactor starts at 2
while potentialFactor < num
      if potentialFactor divides num evenly
            return false
      otherwise
            increase potentialFactor by 1
at the end, return true if we made it through all potentialFactors, and none
of them divided num evenly
```

c. Finally, translate your algorithm into code. Remember to return a value (whose type matches the return type in your method signature) at the end.

```
public static boolean isPrime(int num)
{
      if(num<2) {
            return false;
      }
      int potentialFactor = 2;
      while(potentialFactor<num) {
            if(num % potentialFactor == 0) {
                  return false;     // this immediately exits the method
            }
            potentialFactor++;
      }
      // If we get to this statement, it means we've checked all potential
      // factors, and none of them divided num evenly (or else we would
      // already have returned false).
      // So num has no factors between 2 and (num-1), so it's prime.
      return true;
}
```

d. Write a program that prints out all the prime numbers between 1 and 100. Make calls to the method you defined above.

```
public class PrintPrimes
{
      // this is just copied from above
      public static boolean isPrime(int num)
      {
            int potentialFactor = 2;
            while(potentialFactor<num) {
                  if(num % potentialFactor == 0) {
                        return false;
                  }
                  potentialFactor++;
            }
            return true;
      }

      public static void main(String [] args)
      {
            for(int i=1; i<100; i++) {
                  System.out.print("" + i + " is ");
                  if(isPrime(i)) {
                        System.out.println("prime");
                  } else {
                        System.out.println("not prime");
                  }
            }
      }
}
```

4. Taste of programming with Methods

a. Write methods to display a small triangle, a small square, and a small horizontal line in with ASCII art. Make calls to these methods from the main() method so that you create the following figure:

```
XX
XX
=================
  ^
 ^^^
^^^^^
XX
XX
```

```
public class drawFigure {
      public static void drawTriangle() {
            System.out.println("  ^");
            System.out.println(" ^^^");
            System.out.println("^^^^^");
      }

      public static void drawSquare() {
            System.out.println(“XX”);
            System.out.println(“XX”);
      }

      public static void drawLine() {
            System.out.println(“================”);
      }

      public static void main(String [] args) {
            drawSquare();
            drawLine();
            drawTriangle();
            drawSquare();
      }
}
```

b. Modify your methods so that they accept a char argument, and the characters that make up each figure match the value of the argument. Then add arguments to your method calls in main() so that the program now displays:

```
XX
XX
----------------
  u
 uuu
uuuuu
OO
OO
```

```
public class drawFigure {
      public static void drawTriangle(char C) {
            System.out.println("  " + C);
            System.out.println(" " + C + C + C);
            System.out.println("" + C + C + C + C + C);
      }

      public static void drawSquare(char C) {
            System.out.println("" + C + C);
            System.out.println("" + C + C);
      }

      public static void drawLine(char C) {
            for(int i=0; i<16; i++) {
                  System.out.print(C);
            }
            System.out.println();
      }

      public static void main(String [] args) {
            drawSquare('X');
            drawLine('-');
            drawTriangle('u');
            drawSquare('O');
      }
}
```

5. Test of method development.

For the main() method below, come up with a way to break it into multiple methods (also known as "decomposing"), so that the resulting program is better organized. The goal is to print out the entire stairwell with any given number of step, length of run, and height of rise.

```
public class PrintSteps
{
  public static void main(String [] args)
  {
    System.out.println("-----");
    System.out.println("     |");
    System.out.println("     |");
    System.out.println("      -----");
    System.out.println("           |");
    System.out.println("           |");
  }
}
```

```
public class PrintSteps{
public static void main(String [] args){
int width = 5, height = 2, number = 2;

for(int i = 0; i<number; i++){
   drawStep(width, height, i);
}
}
public static void drawStep(int w, int h, int n){
int start = n*(w+1);
drawTop(start, w);
for(int i=0; i<h; i++)
   drawVLine(start+w);
}
public static void drawTop(int s, int w){
int i = 0;
for(;i<s;i++)
   System.out.print(" ");
for(;i<s+w;i++)
   System.out.print("-");
System.out.println();
}
public static void drawVLine(int s){
for(int i=0;i<s;i++)
   System.out.print(" ");
System.out.println("|");
}
}
```