Methods

- Method Basics
- Parameters
- Void vs. Non-void Methods
- Recursion

Reading:
- Section 2.1
- Sections 1.5 and 1.6, at your leisure (not required, per se)
As discussed previously, a typical non-trivial Java program consists of a collection of classes, each of which consists of a collection of methods (and other stuff).

In general, the purpose of methods is to encapsulate relatively small, cohesive chucks of code, that are typically called repeatedly from various parts of a program.

- Encapsulate – bring together and enclose as a unit
- Relatively small – no specific “size,” per se, but rather a judgment call
- Cohesive – everything in the method contributes to a single specific task.

The use of classes and methods in a program is not unlike the use of drawers and files in an filing cabinet.
public class methodTest {
    public static void hello() {
        int i; // Normally we would declare variable i
        System.out.println("About to print 10 lines"); // in the for-loop but for the purpose of
        for (i=1; i<=10; i++) // discussion we will declare it outside
            System.out.println("hello world"); // the loop in these examples
        System.out.println("About to leave method hello");
    }

    public static void goodbye() {
        int j;
        System.out.println("About to print 20 lines");
        for (j=1; j<=20; j++)
            System.out.println("goodbye world");
        System.out.println("About to leave method goodbye");
    }

    public static void main(String[] args) {
        System.out.println("Here we go!");
        hello();
        System.out.println("Almost done!");
        goodbye();
        System.out.println("All done!");
    }
}

A method has a **header** (a.k.a. **prototype** or **signature**), which is the first line of the method.

A method has a **body**, which is the executable part of the method.

Every method has a name, which is an identifier that you come up with.

Of all the methods in a program, exactly one must be named “main.”
Method Terminology

- When a program runs, execution begins in the main method.

- All other methods must be explicitly *called* or *invoked* to execute.

- A method can be called by main, or any other method (subject to a few rules).

- After executing, the flow of control will *return* back to the point where the method was invoked.
A method may have *local variables* declared in the body of the method.

Local variables are only accessible in the body of the method where they are declared.

More formally, the *scope* of a variable extends from the point of its’ declaration to the end of the inner-most enclosing block, which is defined by curly-braces; a variable can only be accessed when it is “in scope.”

(Note that some people use the phrase “frame of reference” rather than “scope,” but those people are strange and misguided…)

A local variable only “exists” when its method is executing.
public class methodTest {
    public static void hello(int k) {
        int i;
        System.out.println("About to print " + k + " lines");
        for (i=1; i<=k; i++)
            System.out.println("hello world");
        System.out.println("About to leave method hello");
    }
    public static void goodby(int m) {
        int j;
        System.out.println("About to print " + m + " lines");
        for (j=1; j<=m; j++)
            System.out.println("goodby world");
        System.out.println("About to leave method goodby");
    }
    public static void main(String[] args) {
        Scanner kb = new Scanner(System.in);
        System.out.println("Here we go!");
        hello(3);
        System.out.print("Enter n:");
        int n = kb.nextInt();
        System.out.println("Here we go!");
        hello(n);
        System.out.println("Almost done!");
        goodby(n*2);
        System.out.println("All done!");
    }
}
The variables \( k \) and \( m \) in the methods \textit{hello} and \textit{goodby}, respectively, are called \textit{parameters} or, more specifically, \textit{formal parameters}.

The 3, \( n \) and \( n*2 \) in the call to \textit{hello} and \textit{goodby}, respectively, are called \textit{actual parameters}; sometimes also called \textit{arguments}.

The type of the actual parameter must "match" the type of the formal parameter (as with assignment statements).

\begin{quote}
\texttt{hello(“dog”);} \textit{doesn’t make sense.}
\end{quote}

Formal parameters and local variables can be named using any identifier.
Parameter Passing

A formal parameter is much like a local variable, in that it only exists while the method is executing; memory for each formal parameter is:

- Allocated when the method is invoked
- De-allocated when the method terminates

An actual parameter can be:

- A value (i.e., a literal or constant)
- A variable
- More generally, an expression

When a method is invoked each actual parameter is:

- Evaluated
- Copied into the corresponding formal parameter
public class methodTest {
    public static void hello(int k) {
        int i;
        System.out.println("About to print " + k + " lines");
        for (i=1; i<=k; i++)
            System.out.println("hello world");
        System.out.println("About to leave method hello");
    }

    public static void goodby(int k) {
        int i; // Notice the name of the local variable
        System.out.println("About to print " + k + " lines"); // is the same as in the above method.
        for (i=1; i<=k; i++) // Same for the formal parameter.
            System.out.println("goodby world");
        System.out.println("About to leave method goodby");
    }

    public static void main(String[] args) {
        Scanner kb = new Scanner(System.in);

        System.out.print("Enter n:");
        int n = kb.nextInt();
        System.out.println("Here we go!");
        hello(n);
        System.out.println("Almost done!");
        goodby(n*2);
        System.out.println("All done");
    }
}
Local variables and formal parameters in different methods can be named the same, since they have non-overlapping scope.

From a “software-standards” point of view, shouldn’t they be named differently?

Although such a standard is not entirely unheard of, most would consider it unnecessary and unworkable in practice.

In short, happens all the time…
import java.util.*;  // This line was left off of all previous programs for the sake of brevity
public class mparTest {

    public static void helloGoodbye(int k, int m) {  // Notice the use of multiple formal parameters
        System.out.println("About to print " + k + " lines");
        for (int i=1; i<=k; i++)
            System.out.println("hello world");

        System.out.println("About to print " + m + " lines");
        for (int i=1; i<=m; i++)
            System.out.println("goodby world");
        System.out.println("About to leave method helloGoodbye");
    }

    public static void main(String[] args) {
        Scanner kb = new Scanner(System.in);

        System.out.print("Enter n:");
        int n = kb.nextInt();
        System.out.println("Here we go!");
        helloGoodbye(n, n*2);  // Note that multiple actual parameters must be provided
        System.out.println("All done");
    }
}
Methods with Multiple Parameters

- Actual parameters can be arbitrary expressions, so long as the type of each actual parameter matches the corresponding formal parameter.

- All actual parameters are evaluated before invoking the method.

```java
public static void main(String[] args) {
    Scanner kb = new Scanner(System.in);
    System.out.print("Enter n:");
    int n = kb.nextInt();
    System.out.println("Here we go!");
    helloGoodbye(5, 20);
    helloGoodbye(n, n*2);
    helloGoodbye(n*30-17, n*(n+1)/2);
    System.out.println("All done!");
}
```
import java.util.*;

public class mparTest {

    public static void helloGoodbye(int k, String s1, int m, String s2) {
        // Notice the String parameters
        System.out.println("About to print "+ k + " lines");
        for (int i=1; i<=k; i++)
            System.out.println(s1);

        System.out.println("About to print "+ m + " lines");
        for (int i=1; i<=m; i++)
            System.out.println(s2);
        System.out.println("About to leave method helloGoodbye");
    }

    public static void main(String[] args) {
        Scanner kb = new Scanner(System.in);

        System.out.print("Enter n:");
        int n = kb.nextInt();
        System.out.println("Here we go!");
        helloGoodbye(n, "hello world", n*2, "goodby world");
        System.out.println("All done!");
    }
}
A method may have zero or more formal parameters.
- They do not have to all be of the same type

The number of actual parameters must be the same as the number of formal parameters.

The type of each actual parameter must “match” with the type of the corresponding formal parameter (as with assignment statements).

When a method is invoked:
- Each of the actual parameters is evaluated
- The resulting values are copied to the corresponding formal parameters
In the previous program, should one combined method be used, or two separate methods?

This question is somewhat analogous to the question concerning whether two files in a filing cabinet should be combined.

The question is one of organization, or rather, software design.
Consider the following program:

```java
public class sickMethod {
    static int x, y, z;

    public static void main(String[] args) {
        System.out.println("**********************************************");
        System.out.println(" Welcome to the random calculation program ");
        System.out.println("**********************************************");
        x = 25;
        y = 300;
        z = x * y;
        System.out.println(z);
        System.out.println("**********************************************");
        System.out.println(" We hope you enjoyed the experience ");
        System.out.println(" Please come again! ");
        System.out.println("**********************************************");
    }
}
```
Now consider the following two versions of the previous program, each making use of methods.

http://cs.fit.edu/~pbernhar/teaching/cse1001/sickMethod1.txt

http://cs.fit.edu/~pbernhar/teaching/cse1001/sickMethod2.txt

Which would you prefer?
public class intTest {

    public static void exam(double m) {
        int x = 0;

        System.out.println(m);
        System.out.println(i);
        System.out.println(x);
    }

    public static void main(String[] args) {
        int i;
        double m1;

        exam(3.0);
        i = 10;
        m1 = i;
        exam(i);
        x = 0;
        m1 = 0.0;
        for (int x=1; x<=10; x++)
            System.out.println("hello");
        exam("dog");
        for (int x=1; x<=10; x++) {
            System.out.println("hello");
        exam(10.5);
        }
    }
}
All of the methods we have seen so far are called *void* methods.

Among other things, this is indicated by the word *void* in the first line.

*non-void* methods are another type of method.

Used when a method is needed that computes or constructs something that is to be used by another part of a program:

- Value of a function
- Address of a location in memory containing important data
- An object that is to be displayed by another part of the program
- *In a sense, a non-void method is a parameter “in reverse”*
Non-Void Methods

- Often times the programmer has the option of writing a method as either void or non-void method; in such a case the choice is a judgment call.

- What is syntactically different about the following program?

  http://my.fit.edu/~pbernhar/Teaching/SoftwareDevelopment1/functions.txt

- There are 3 main things to know about non-void methods…
Non-Void Methods

First - Every non-void method will have:

- A non-void type in the header, call the return type of the method
- At least one return statement

```java
public static int parabola(int x) {
    int y;
    
    y = x * x + 3 * x - 10;
    
    return y;
}
```
Second - Every return statement in a non-void method must specify an object to be returned that is “compatible” the return type.

```java
public static int parabola(int x) {
    int y;
    
    y = x * x + 3 * x - 10;
    
    return y;
}
```
Non-Void Methods

- More generally, any expression of the return type will work (value, variable, or more complicated expression).

```java
public static int parabola(int x) {
    int y;

    y = x * x + 3 * x - 10;
    if (y > 10)
        return 25; // Returning a literal
    else if (y > 5)
        return y; // Returning the contents of a variable
    else
        return (y*x+37); // Returning the result of an expression
}
```
Some things that won’t work:

```java
public static int parabola(int x) {
    double y;

    y = x * x + 3 * x - 10;
    if (y > 10)
        return 25.0;
    else if (y > 5)
        return "dog";
    else
        return y;
}
```
Third - Every non-void method will be invoked in a context that expects a returned object of the return type.

```java
int n, m;
int answer;
double anotherAnswer;

System.out.print("Enter a value:");
n = kb.nextInt();

System.out.print("Enter another value:");
m = kb.nextInt();

answer = parabola(n);
anotherAnswer = funny(n,m);
System.out.println(parabola(3));
answer = 3 * parabola(m) + n;
System.out.println("Forth result:" + (parabola(n) * funny(n,m)));
anotherAnswer = funny(n+5, parabola(m));
answer = parabola(funny(3,5));          // Why doesn’t this one work?
```
Remember this program?

```java
public class sickMethod {
    static int x, y, z;

    public static void main(String[] args) {
        System.out.println("**********************************************");
        System.out.println(" Welcome to the random calculation program ");
        System.out.println("**********************************************");
        x = 25;
        y = 300;
        z = x * y;
        System.out.println(z);
        System.out.println("**********************************************");
        System.out.println(" We hope you enjoyed the experience ");
        System.out.println(" Please come again! ");
        System.out.println("**********************************************");
    }
}
```
A version using a non-void method:

http://cs.fit.edu/~pbernhar/teaching/cse1001/sickMethod3.txt
A method can return just about any type of object:

```java
public static String bigCat(String s1, String s2) {
    String s, newS1, newS2;

    newS1 = s1.toUpperCase();
    newS2 = s2.toUpperCase();
    s = newS1 + newS2;

    return s;
}

public static void main(String[] args) {
    String a = "sponge ";
    String b = "bob";
    String c,d;

    c = bigCat(a,b);
    System.out.println(c);

    d = bigCat("man ", "cow");
    System.out.println(d);
}
```
Converting an integer to an equivalent string of words is called *anglicizing*.

37,452

“Thirty seven thousand, four hundred, fifty two.”

Technically, anglicizing an arbitrary integer to words is impossible…why?

To make the problem solvable, the range of integers will be restricted to values in the range $0 < n < 1,000,000$
First lets consider a vary large number:

358,294,341,964,781
First let's consider a very large number:

358,294,341,964,781

trillion  billion  million  thousand
First let's consider a very large number:

358,294,341,964,781
First let's consider a very large number:

358,294,341,964,781
First let's consider a very large number:

358,294,341,964,781
First let's consider a vary large number:

358,294,341,964,781
First lets consider a vary large number:

358,294,341,964,781
The anglicizing program will make use of the following methods.

- Returns the English equivalent of n, for \(0 < n < 10\); returns "", if \(n = 0\)

```java
public static String digitName(int n) {
    if (n == 0) return "";
    else if (n == 1) return "one ";
    else if (n == 2) return "two ";
    else if (n == 3) return "three ";
    else if (n == 4) return "four ";
    else if (n == 5) return "five ";
    else if (n == 6) return "six ";
    else if (n == 7) return "seven ";
    else if (n == 8) return "eight ";
    else return "nine ";
}
```
Returns the English equivalent of \( n \), for \( 10 \leq n \leq 19 \)

```java
public static String teenName(int n) {
    if (n == 10) return "ten ";
    else if (n == 11) return "eleven ";
    else if (n == 12) return "twelve ";
    else if (n == 13) return "thirteen ";
    else if (n == 14) return "fourteen ";
    else if (n == 15) return "fifteen ";
    else if (n == 16) return "sixteen ";
    else if (n == 17) return "seventeen ";
    else if (n == 18) return "eighteen ";
    else return "nineteen ";
}
```
Returns the English equivalent of $10n$, for $2 \leq n \leq 9$

```java
public static String tensName(int n) {
    if (n == 2) return "twenty ";
    else if (n == 3) return "thirty ";
    else if (n == 4) return "forty ";
    else if (n == 5) return "fifty ";
    else if (n == 6) return "sixty ";
    else if (n == 7) return "seventy ";
    else if (n == 8) return "eighty ";
    else return "ninety ";
}
```
Returns the English equivalent of n, for 0 < n < 100

```java
public static String anglicize3(int n) {
    String s;

    if (n < 10)
        s = digitName(n);
    else if (n < 20)
        s = teenName(n);
    else // n >= 20
        s = tensName(n/10) + digitName(n%10);

    return s;
}
```
Non-Void Methods

- Returns the English equivalent of \( n \), for \( 0 < n < 1000 \)

```java
public static String anglicize2(int n) {
    String s;

    if (n < 100)
        s = anglicize3(n);
    else
        s = digitName(n/100) + "hundred " + anglicize3(n%100);

    return s;
}
```
Returns the English equivalent of n, for 0 < n < 1000000

```java
public static String anglicize1(int n) {
    String s;

    if (n < 1000)
        s = anglicize2(n);
    else
        s = anglicize2(n/1000) + "thousand " + anglicize2(n%1000);

    return s;
}
```
Non-Void Methods

- Outputs the English equivalent of n, for 0 < n < 1000000

```java
public static void main(String[] args) {
    int x;
    String s;

    System.out.print("Enter a positive integer:");
    x = kb.nextInt();
    if (x <= 0)
        System.out.println("Number is too small");
    else if (x >= 1000000)
        System.out.println("Number is too large");
    else {
        s = anglicize1(x);
        System.out.println(s);
    }
}
```

- The complete program:
  [http://cs.fit.edu/~pbernhar/teaching/cse1001/anglicize2.txt](http://cs.fit.edu/~pbernhar/teaching/cse1001/anglicize2.txt)
Different Forms of a Method

Recall the following, somewhat poorly written program:

```java
public static double funny1(int a, int b) {
    double x, y, z;

    if (a < b) {
        x = Math.sqrt(a);
        y = Math.sqrt(b);
        z = x + y;
        return z;
    }
    else {
        x = Math.sqrt(a);
        y = Math.sqrt(b);
        z = x - y;
        return z;
    }
}

public static void main(String[] pars) {
    double f;

    f = funny1(3,4);
}
```
Common code is frequently “factored out”:

```java
public static double funny2(int a, int b) {
    double x, y, z;

    x = Math.sqrt(a);
    y = Math.sqrt(b);

    if (a < b)
        z = x + y;
    else
        z = x - y;

    return z;
}
```
Different Forms of a Method

- This version eliminates the variable z; also note the expressions in the return statements.

```java
public static double funny3(int a, int b) {
    double x, y;
    x = Math.sqrt(a);
    y = Math.sqrt(b);
    if (a < b)
        return x+y;
    else
        return x-y;
}
```

- Note that expressions appear in the return statements.

- Also note the introduction of an additional return statement; the method no longer has a “single point of exit.”
This version eliminates the \texttt{else} clause:

```java
public static double funny4(int a, int b) {
    double x, y;

    x = Math.sqrt(a);
    y = Math.sqrt(b);

    if (a < b)
        return x+y;

    return x-y;
}
```
Different Forms of a Method

- This version eliminates the variables $y$ and $z$; note the expressions in the return statements:

```java
public static double funny5(int a, int b) {
    if (a < b)
        return Math.sqrt(a)+Math.sqrt(b);
    return Math.sqrt(a)-Math.sqrt(b);
}
```

- A return statement may return an expression that is compatible with the return-type of the method.
  - literal
  - variable
  - expression

- Which version do you prefer?
Every method has a "signature" that uniquely identifies and distinguishes that method; up until now, the signature was simply the method name.

More generally, the signature consists of the following components:
- method name
- number of formal parameters
- types of the formal parameters
- order of the formal parameters

Thus, two methods could have the same name, so long as they have a different number of formal parameters, the parameters are of different types, or the types of the parameters appear in a different order.
- The names of the formal parameters do not matter, i.e., they are not part of the signature.
- The return-type of a method also does not matter.

This works somewhat like the name of a person – first name, middle, and last.
Overloaded Methods

- When a program contains more than one method with the same name then the name is said to be "overloaded."

- Let's look at the code:
  
  http://cs.fit.edu/~pbernhar/teaching/cse1001/overloadTest.txt
  http://cs.fit.edu/~pbernhar/teaching/cse1001/overloadTest1.txt
Method Vocabulary

Make sure you know each of the following terms:

- void method
- non-void method
- method header (also called the "declaration," "signature" or "prototype" of the method)
- method body
- formal parameter
- actual parameter (also called an "argument")
- calling, or invoking a method
- returning from a method
- return statement
- return type
- local variable
- overloading, or overloaded method
- recursive method (haven’t done this one yet)

Try methodizing this one:

http://cs.fit.edu/~pbernhar/teaching/cse1001/nested.txt