

APS105S Winter 1999 Lecture 8: Methods

Suppose you want to make a large rainbow jello dessert for a party, but you're too busy to make it yourself. Instead, you want to leave instructions for your kid brother to follow...

1. boil water
 2. dissolve **RED** package in 1 cup of hot water
 3. mix in 1 cup of cold water
 4. pour into jello mold
 5. re-fridge until set
-
6. boil water
 7. dissolve **ORANGE** package in 1 cup of hot water
 8. mix in 1 cup of cold water
 9. pour into jello mold
 10. re-fridge until set

....etc..... YELLOW, GREEN, BLUE (15 more steps)

As you can see, the instructions are very similar and very repetitive. It would be better just to give generic instructions once, and describe how to customize it at each layer:

Method: MakeJelloLayer(*COLOUR*)

1. boil water
2. dissolve *COLOUR* package in 1 cup of hot water
3. mix in 1 cup of cold water
4. pour into jello mold
5. re-fridge until set

Main:

```
MakeJelloLayer( RED )
MakeJelloLayer( ORANGE )
MakeJelloLayer( YELLOW )
MakeJelloLayer( GREEN )
MakeJelloLayer( BLUE )
```

Example

```
class Factorial
{
    static int factorial(int n)
    {
        int i, fact = 1;
        for( i=1; i<=n; i++ )
            fact *= i;
        return fact;
    }

    public static void main(String[] args)
    {
        int five = 5, five_fact;
        five_fact = factorial( five );
        System.out.println( five_fact );
        System.out.println(
            factorial( 3*Stdin.getInt() ) );
    }
}
```

“`static int factorial(int n)`” defines a new method
(NOTE: NO SEMICOLON)

Q: is “`public static void main(String[] args)`” a method?

A: YES! but main() is a special one... it is where your program starts!

Alternatively, suppose you are preparing a 12-course meal. It would be very awkward to have the whole meal written up as one 8-page recipe, with the steps for the courses mixed together. Instead, it would be better to separate each course into its own recipe, preferably each on its own page. This keeps things orderly, and makes it easier for you to mix & match with recipes from other meals.

Methods

- groups of instructions in our program
 - you give each method a name
 - organizes a program: each method has different purpose/task
 - can be a generic set of instructions, made more specific by *PARAMETERS* (example: colours of jello)
 - may compute some value and return the *RESULT*
 - re-usable!!!
-
- in other languages, called:
 - functions, procedures, subroutines
 - functions are....
 - procedures are...

methods you've used already:

```
Stdin.getDouble();
Math.sqrt( x );
System.out.println( "Hello world" );
```

the underlined word is the method name

- in the factorial() method, “n” is called the *PARAMETER* or *ARGUMENT*
 - factorial can safely modify the value of “n” *without* affecting the rest of the program.... “n” is a “local” variable
- in main(), we can *CALL* or *INVOKE* the factorial() method...
 - must provide a value for the integer parameter “n” by placing an integer expression inside the ()
 - can call any method from any other method (almost)
- likewise, “args” is an argument to main() !!!!
 - learn more about main() args later
- remember scope?????????
 - each method has *its own* scope!!!
 - inside factorial() method, we can't “see” the variable **five_fact** located inside main()
 - inside main(), we can't “see” variables **i**, **fact**, or **n** located inside factorial()
 - think of:
 - { as “begin new scope”
 - still knows about vars from previous {
 - } as “forget all variables since most recent scope began”

Scope Example

```
{
  int i = 1;

  if( true ) {
    int j = 2;
    System.out.println(i+j);    // ok
    {
      System.out.println(i+j+k); // NO k
      int k = 3;
      System.out.println(i+j+k); // ok
    }
    System.out.println(i+j+k);  // NO k
  }

  System.out.println(i+j+k);    // NO j,k
  System.out.println(i);        // NO i
}
```

```
class Power
{
  static double power( double x, int n )
  {
    int i;
    double pow = 1.0;
    if( n < 0 )
      return 0.0; // reasonable????
    for( i=0; i<n; i++ )
      pow *= x;
    return pow; // ok
  }

  public static void main(String[] args)
  {
    double x_n;
    x_n = power( 3.0, 2 );
    System.out.println( x_n );
    do {
      x_n = power( Stdin.getDouble(),
                  Stdin.getInt() );
      System.out.println( x_n );
    } while( x_n != 0.0 );
  }
}
```

- here, power() has 2 parameters: x and n
- when we call power() from inside main
 - we must give values to x and n
- values are assigned by matching the POSITION of the params
 - 1st to 1st, 2nd to 2nd, etc
- the TYPE of each parameter must MATCH
 - power(3, 2.0) is incorrect!!!

General Form

```
static return-value-type method-name(
  parameter-list )
{
  ... body of method...
}
```

return-value-type: one of
int
double
boolean
void (for no return value)
(there are others....)

method-name:
your choice, start with a lower case
letter

parameter-list:
type variable-name
type variable-name, type variable-
name, ...

- inside the body, you can place one or more return statements:
return expression;
- this immediately **stops** the method and returns the value of expression to the caller
 - (note: type of expression must match return-value-type)
- if “**void**” used as return type, can just say
return;
- to return nothing (right away),
- OR can wait for program to reach the last } of the method.

- to call a method from our program (from inside main, or from inside another method), type
method-name(parameters)
- if method has no parameters, you still type the ()
- we end with “;” if it is the end of a statement
- if the method returns a value, you must “use” that value inside an expression, eg:
x_n = power(3.0,2); //ok
System.out.println(power(3.0,2)); //ok
power(3.0,2); //incorrect!!!!

Reading

- Methods — in Chapter 5 of textbook
 - 5.1, 5.2, 5.4, 5.5, 5.7, 5.9