Chapter 2

Methods and method calls

Lesson page 2-1. Methods

Activity 2-1-1 Methods are recipes

Question 1. The sequence of instructions for wrapping a present given below is correct, but it leaves out a lot of details. It is possible to follow these instructions exactly and end up with a present that looks nice or looks as if it was wrapped by a two year old.

1. Cut paper to the size of the present. (Do you leave extra to overlap? If so, how much?)
2. Wrap the present in the paper. (Just crumple the paper around the present?)
3. Use the tape to attach the paper. (Wind the tape around and around and around?)
4. Write the name of the person who is to receive the present on the label using the pen.
5. Wrap the string around the present. (How many times?)
6. Attach the label to the present. (Using the string? Tape? Chewing gum?)

Question 2. It is possible to generalize cooking pasta and potatoes, if you don't add oil to the water when boiling pasta. However, rice requires a fixed ratio of water to rice, and salt usually isn't added to rice when cooking.

1. Fill medium pot 2/3 of the way full.
2. Add a dash of salt.
3. Add one serving of pasta/potatoes.
4. Boil until done.

Activity 2-1-2 The blackbox view of a method

Question 3. A method is a parameterized sequence of instructions. In Java there are three types of methods: function, procedure, and constructor.

Question 4. A procedure is a Java method that has keyword `void` in its header.

Question 5. A specification (of a method) is a clear and precise description of what the method does, in terms of its parameters.

Question 6. A method header has the form:

<*modifiers*><*return type*><*name*> (<*parameters*>)

For example, the method header

public static int `addOne(int x)`

has modifiers **public** and **static**, return type `int`, name `addOne`, and one parameter `x` (which is of type `int`).

(The < > are not necessary, the answer "modifiers, return type, name, and parameters" is also correct.)

Question 7. In Java, the body of a method is the sequence of instructions that occur within the braces of a method (after the header).

Question 8. A parameter is a variable that is declared within the parentheses of the header of a method.

Question 9. The form of a parameter declaration is: <*type*><*identifier*>

Question 10. Yes, `int` is a type.

Question 11. Yes, a parameter is a variable.

Question 12. An entity (a class, method or field) that is declared `public` can be accessed by any part of the program.

Question 13. **public void** `myFirstHeader(int i, int j)`

Activity 2-1-3 Check your understanding of terminology

Activity 2-1-4 Understanding procedure calls

Question 14. To figure out what execution of a procedure call does, make a copy of the specification of the procedure and replace occurrences of the parameters by the corresponding arguments of the call. The resulting statement will tell you what the procedure call does.)

Question 15. The method call `drawRect(22, 30, 40, 10);` draws a rectangle with top-left corner at pixel (22, 30), height 10, and width 40.

Activity 2-1-5 The general form of a procedure call

Question 16. Every procedure call consists of:

1. An identifier, which is the procedure name.
2. Zero or more arguments, separated by commas and enclosed in parentheses.
3. A semicolon.

Question 17. An argument is an expression that appears within the parentheses of a method call.

Question 18. The rules for the number and types of arguments are:

1. A method call has one argument for each parameter of the method.
2. If a method call has no arguments, parentheses are still mecessary (but only whitespace can appear between them).
3. The type of an argument must match the type of the corresponding parameter. For example, method

```
public void bob(int x) { };
```

   can be called using `bob(smells);` iff variable `smells` is of type `int`.

Question 19. True; this is a valid procedure call:

```
drawLine(0, 10, (33)/11, 4);
```

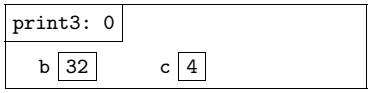
Lesson page 2-2. Method bodies and method calls

Activity 2-2-1 The method body

Question 1. To execute a procedure call:

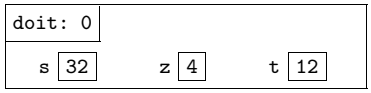
1. Evaluate the arguments of the call.
2. Draw a frame for the call.
3. Draw the parameters and local variables in the frame.
4. Store the value of each argument in the corresponding parameter.
5. Execute the method body.
6. Erase the frame for the call.

Question 2. Evaluate the arguments: the first argument evaluates to 32; the second, to 4. Now draw the frame, draw in the parameters, and assign the value of each argument to the corresponding parameter:



Executing the method body results in 32, 4, and 36 being printed on separate lines. Then, the above frame is erased.

Question 3. The first argument evaluates to 32, the second to 4, and the third to 12. Drawing the frame, putting in the parameters, and assigning argument values to parameters results in:



Question 4. Here are two versions of `strangeAdd`:

```
public static void strangeAdd(int first, int second) {
    int storage= (2 * first) + (3 * second);
    System.out.println(storage);
}
```

or

```
public static void strangeAdd(int first, int second)
    {  System.out.println((2 * first) + (3 * second));  }
```

Activity 2-2-2 Example of method-call execution

Question 5. A frame is a "box" that holds information for a method call.

Question 6. How to execute a procedure call:

1. Evaluate the arguments of the call.
2. Draw a frame for the call.
3. Draw the parameters in the frame.
4. Store the value of each argument in the corresponding parameter.
5. Execute the method body.
6. Erase the frame for the call.

Activity 2-2-3 Executing inner calls

Question 7. How to execute a procedure call:

1. Evaluate the arguments of the call.
2. Draw a frame for the call.
3. Draw the parameters in the frame.

4. Store the value of each argument in the corresponding parameter.
5. Execute the method body.
6. Erase the frame for the call.

Question 8. Nothing happened to the frame for method `paint` while `drawTriangle` was being executed.

Question 9. False. When `drawTriangle` finished executing, we continued execution of `paint` at the code following the call on `drawTriangle`.

Activity 2-2-4 The stack of frames for calls

Question 10. A stack is a list of elements that can be changed by only two operations:

1. Push: add an element to the top (or front) of a list.
2. Pop: remove an element from the top (or front) of a list.

Question 11. (a) When a procedure is called, a frame for the procedure is created and "pushed" onto the stack of frames. (b) When execution of the procedure finishes, the frame for that procedure is "popped" from the stack and discarded.

Activity 2-2-5 Review of methods

Activity 2-2-6 Writing procedure calls

Question 12. Debugging is the fine art of detecting and eliminating errors in a program without introducing new ones. (The answer "debugging is finding and eliminating errors in a program." is also correct.)

Question 13. Call to draw a circle with center `(40,50)` and radius `20`:

```
drawCircle(20, 30, 40);
```

Call to draw a circle with top left corner `(20,30)` and side length 40:

```
drawCircle(20, 30, 40);
```

Lesson page 2-3. Two components of method bodies

Activity 2-3-1 Local variables

Question 1. A local variable is a variable that is declared within the body of a method.

Question 2. Form of a local variable declaration:

<*type*> <*variable name*> ;

Question 3. To initialize a variable means to store a first (or initial) value in it.

Question 4. A local variable must be initalized before you use it.

Question 5. The scope of something is its general range or extent. In a programming language, the scope of a variable is the part of the program in which it can be referenced by its name.

Question 6. In Java, the scope of a *local variable* is the sequence of statements that follow its declaration, up to the end of the block in which it is declared.

Question 7. The box below indicates the scope of variable `aVar`.

```
public void foo() {
    System.out.println();
    int bVar;
    bVar= 3;
    int aVar;
    System.out.println();
    aVar= bVar * 2;
}
```

Activity 2-3-2 Combining declaration and initialization

Question 8. The form of an initialized variable declaration is:

<*type*> <*variable name*> = <*expression*>;

Question 9. The two-line declaration and assignment can be combined as:

```
int timeToGetUp= 0530;
```

Question 10. The one-line declaration and initalization can be rewritten as:

```
boolean youWin;
youWin= morePoints && gameOver;
```

Question 11. When you declare a local variable, initalize it.

Activity 2-3-3 Check your understanding of local variables

Activity 2-3-4 The return statement

Question 12. Execution of a **return** statement within a procedure body terminates execution of the procedure body and thus of the procedure call that caused it to execute.

Question 13. False. A procedure body need not have a **return** statement.

Question 14. During execution of a procedure body, only one **return** statement can be executed, and only once.

Question 15. A method name is overloaded if two (or more) methods defined in the same class have the same name. This is called overloading. Overloading is allowed, as long as the number and/or type of their parameters are different, so that one can unambiguously determine which one a method call refers to.

Question 16. In the appendix, there are ten (10) different procedures with name `println`.

Lesson page 2-4. Functions

Activity 2-4-1 Form of a function

Question 1. A procedure doesn't return a value, but a function does. Thus, the return type of a procedure is always `void` and the return type of a function is the type of the value it returns.

Question 2. A procedure has the keyword **void** in its header; a function has the type of value that it returns.

Question 3. The return statement in a function body has the form:

return <*expression*> ;

where the expression has the same type as the return type of the function.

Question 4. The type that replaces keyword **void** in a function header is the type of the value returned by the function.

Question 5. False. A function body must have a return statement.

Activity 2-4-2 The function call

Question 6. Procedures and functions are abstraction mechanisms.

Question 7. A procedure call is a statement.

Question 8. A function call is an expression.

Question 9. The purpose of a function specification is to explain WHAT the method does. The body of a function does it.

Question 10. There are two cases:

1. `a` $\geq$ `0`. Then `a-a` is the smallest, and the result is 0.
2. `a` $\leq$ `0`. Then `a` is the smallest, and the result is `a`.

Note the overlap: if `a` = 0, then either `a` or `a-a` is the smallest (but they are the same).

Activity 2-4-3 Evaluating a function call

Question 11. The steps in evaluating a function call are:

1. Evaluate the arguments of the call.
2. Draw a frame for the call (push it on the stack).
3. Draw in the parameters and local variables.
4. Assign the values of the arugments to the parameters.
5. Execute the method body, until it executes a statement **`return e;`** – and then evaluate `e`.
6. Erase the frame (pop it from the stack).
7. Push the value of `e` onto the stack.

Question 12. Executing a procedure call and evaluating a function call are the same except that evaluating a function call has the additional step of evaluating the operand `e` of a return statement **`return e;`** and pushing its value on the stack.

Activity 2-4-4 Check your understanding of functions

Lesson page 2-5. Top-down programming

Activity 2-5-1 Top-down design

Question 1. Top-down programming is a progamming development strategy in which, at each step, some abstract statement (written in English, for example) is implemented. The implementation could involve introducing some variables and/or writing the abstract statment as a series of other statements, which could be in Java or English.

Question 2. Top-down programming is also called *stepwise refinement*.

Question 3. A statement-comment is a comment in a program that is a statement written in English. Our convention is to indent the statements that implement it.

Activity 2-5-2 Edgar Allen Poe used top-down design

Question 4. Poe decided *The Raven* should be 100 lines long.

Question 5. The topic was beauty.

Question 6. The tone was melancholic.

Question 7. The one-word refrain would have to be sonorous and susceptible to protracted emphasis. Poe chose the word “Nevermore”.

Activity 2-5-3 Anglicizing integers —functionally

Question 8. The number `1` is treated differently because, in English, the numbers in the range `10..19` are treated differently from the numbers in the range `20..99`. For example, `20` is "twenty", and the numbers in the range `21..29` all begin with "twenty". But the numbers in the range `11.19` all have special names and don't being with "ten".

Activity 2-5-4 Anglicizing integers —using assignments

Question 9.

```
/** = the integer corresponding to name, for
 *    name one of "zero", "one", "two", ..., "nine".
 * = -1 for any other value in name
 */
public static int onesNames(String name) {
    if (name.equals("zero"))  { return 0; }
    if (name.equals("one"))   { return 1; }
    if (name.equals("two"))   { return 2; }
    if (name.equals("three")) { return 3; }
    if (name.equals("four"))  { return 4; }
    if (name.equals("five"))  { return 5; }
    if (name.equals("six"))   { return 6; }
    if (name.equals("seven")) { return 7; }
    if (name.equals("eight")) { return 8; }
    if (name.equals("nine"))  { return 9; }
    return -1;
}
```