

Syntax error, run-time error and logical error

Revision

- We usually want a loop to iterate through each value in a range.
- Such loops are significantly simpler than other loops.
- A `for`-statement should be used for such loops in place of `while` statements.
- A `continue`-statement usually makes sense in a `for` statement, since the increment expression will still be evaluated even after a `continue`-statement is executed.

0911B L13-1

When programs become complicated, we usually get various errors:

- Syntax error: the compiler don't understand what you want to do. A human language example: "Box, Go assignment get!". It doesn't follow the rule of the language.
- Run-time error: there is something unexpected during the time when the program runs. It's like saying "Find the door after getting out of the lift. Go into it.", but then there is no door when you actually go there.
- Logical error: your instructions can actually be done, but it is not what you really want. E.g. saying "After going into the door, turn left to find the assignment boxes" when you really want turning right.

0911B L13-2

Example program

In the following program, there is a syntax error and some logical error. One of the logical errors causes a run-time error.

```
// Print a table of integer division
import chapman.io.*;
public class Errs {
    public static void main(String[] args) {
        int i, j;
        for (i = 0, i <= 9, i++) {
            for (j = 0; j <= 9; j++);
                Fmt.printf("%4d", i/j);
            System.out.println();
        }
    }
}
```

0911B L13-3

The need of methods

- We can write more complicated programs with top-down design.
- However, with just this technique, programs are easy to write, but difficult to read.
- Once the program is completed with details filled, nowhere shows the high level structure of the program.
- Ideally, we want the main program to contain just the top-level view, which calls routines to do the actual job.
- Also, if your program need to do a task more than once, you don't have to write it several times. You just call it several times.
- Such routines are called Methods in Java.

0911B L13-4

Ingredients of methods

- We want a way to define methods. I.e. to have a part of the program that executes when the main program asks for it.
- We want a way to call methods from the main program (or from other methods).
- We want a way for the main program to give the routine some values to work on, and for the method to return a value to caller.
- We already have something that do this exactly. E.g. `Math.sqrt` is a method. The caller will give it a double value (e.g. 4.0), and it returns a double value (e.g. 2.0) back to the caller.
- To call a method, we write the method name, and then a pair of parentheses enclosing what values to give it. E.g. `in.readDouble()`, and `Math.Sqrt(4.0)`.

0911B L13-5

Defining a method

- The secret is, *you already know how to write a method!* The "main" line is actually a method definition.
- A method is always defined within a class. It looks like:

```
public static returnType methodName(formalArgs) {
    statements
}
```
- `returnType` is the type of value returned by the method. If the method returns nothing, we place `void` there instead.
- By convention, methods are namedLikeThis, i.e. with words capitalized except the first, and concatenated directly.
- `formalArgs` contains a list of argument to pass and their names in the method, in the form `argType argName`.

0911B L13-6

Defining a method (cont'd)

- Statements in a method (its body) are executed one by one when the method is called.
- The statements can use the arguments defined by *formalArgs* just like a variable.
- If a return statement is executed, the method exits immediately. If the returnType is not void, it also specifies a value to return to the caller (e.g. `return 20;` gives the value 20 back to the caller).
- N.B.: "formalArgs" means "formal arguments". They give a way for the method body to refer to the arguments. When the method is called, it specifies the "actual arguments", the values passed by the caller.

0911B L13-7

Example: Print a tree (again)

```
/* Print a tree */
import chapman.io.*;
public class Tree {
    public static void main(String[] args) {
        StdIn in = new StdIn();
        System.out.print("Width of tree = ");
        int width = in.readInt();

    }

    /* Print a line containing '*' characters,
       from column starStart to column starEnd. */
    public static void printLine(int starStart, int starEnd) {

    }
}
```

0911B L13-8

Class announcement

- **Upcoming workshop:** Coming workshop is on Tuesday Oct 31, 2000 and Wednesday Nov 1, 2000, about program debugging.
- **Assignment deadline:** Remember that the soft deadline is on the coming Monday.
- **Reading week:** No lecture on Oct 23, 26, 27. During lecture hour of Oct 23, I'll hold a tutorial letting you to ask any question you want. *No new lecture material will be covered.* I.e., you can safely skip them if you understand the lectures. This can be repeated on Oct 26 and 27 if attendance is high.

0911B L13-9