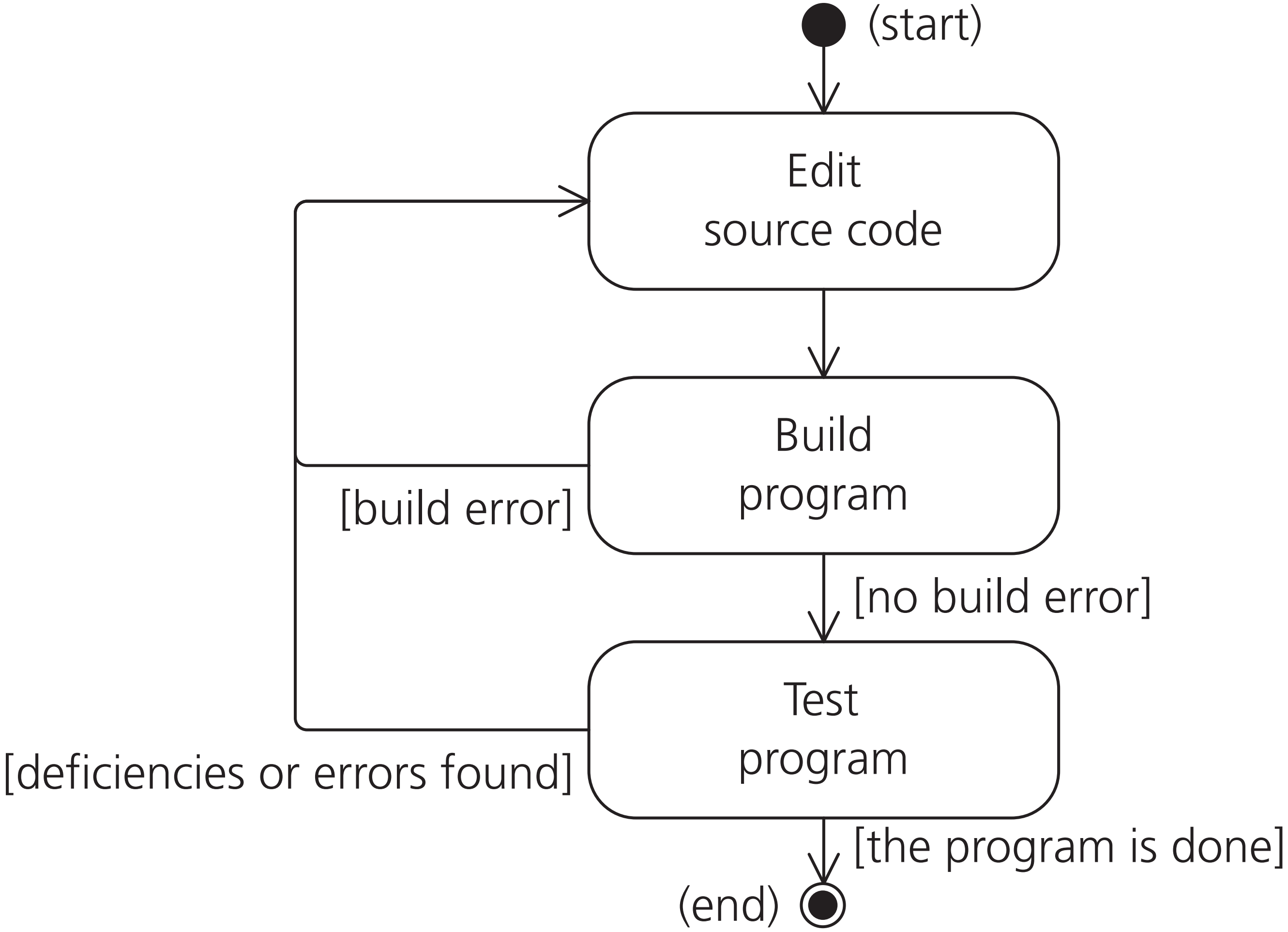
Overview of programming activities

- Activities sufficient for writing small program:



- Many other activities are involved when writing larger programs.

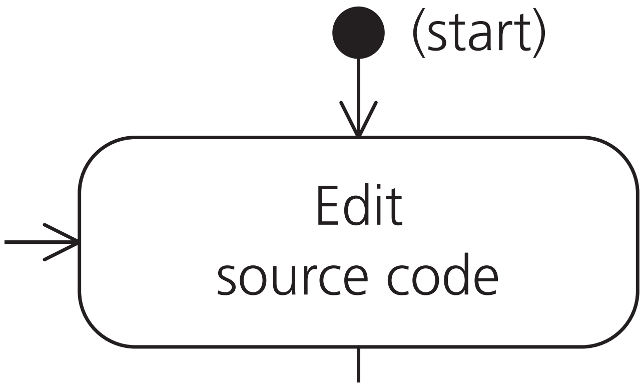
Source code (example)

Class name

```
// (1) This source code file is called SimpleProgram.java
public class SimpleProgram {
  // Print a proverb, and the number of characters in the proverb.
  public static void main(String[] args) {                        // (2)

    System.out.println("A proverb:");                             // (3)

    String proverb = "Practice makes perfect!";                   // (4)
    System.out.println(proverb);                                  // (5)

    int characterCount = proverb.length();                        // (6)
    System.out.println("The proverb has " + characterCount
        + " characters.");
  }
}
```

Editing source code

- We write the source code in text files:
  - Commonly called *source code files*.
  - Describes exactly what tasks the computer should perform.
  - Contain only characters that constitute the actual text of the source code, (no formatting).
  - Choose a good editor for writing source code.



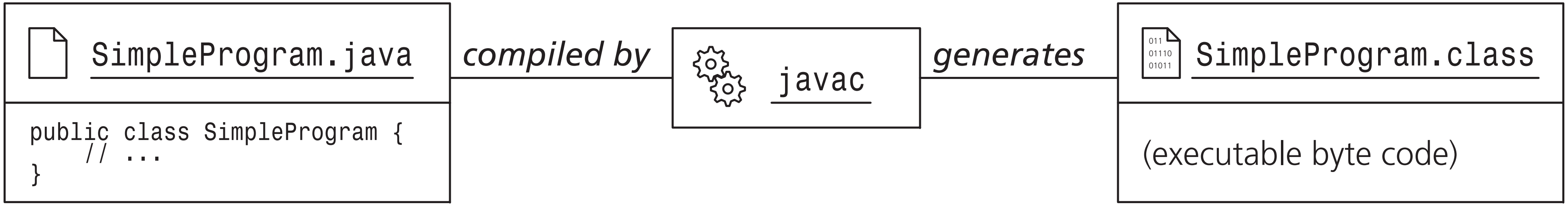
- The compiler requires the source code files to be named according to specific rules:
  - Correct:
    - `SimpleProgram.java`

Class name

  - Incorrect:
    - `simpleprogram.java` *(wrong case)*
    - `SimpleProgram.java`.doc *(wrong extension)*
    - `Simple~1.java`.doc *(Microsoft Windows short-names not allowed)*

Build program: Compiling Java programs

> javac SimpleProgram.java *(Run this on the command-line)*



SimpleProgram.java

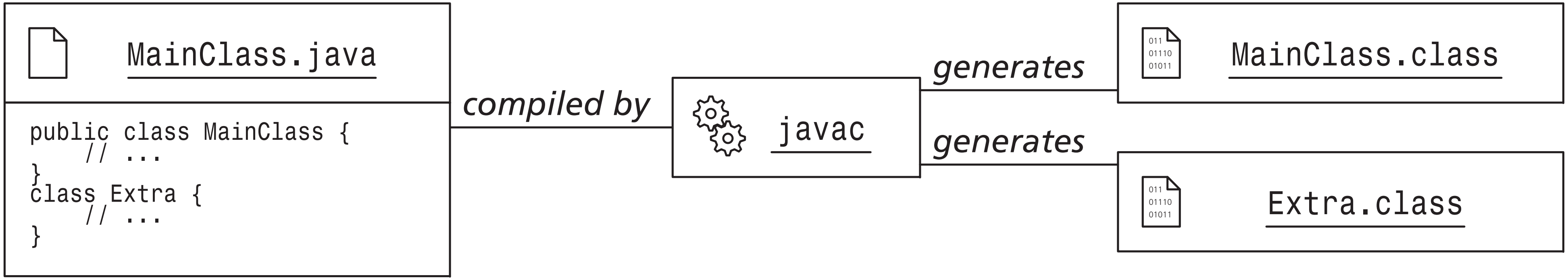
```
public class SimpleProgram {
    // ...
}
```

compiled by javac *generates* SimpleProgram.class

(executable byte code)

(a) One class in the source code file

> javac MainClass.java



MainClass.java

```
public class MainClass {
    // ...
}
class Extra {
    // ...
}
```

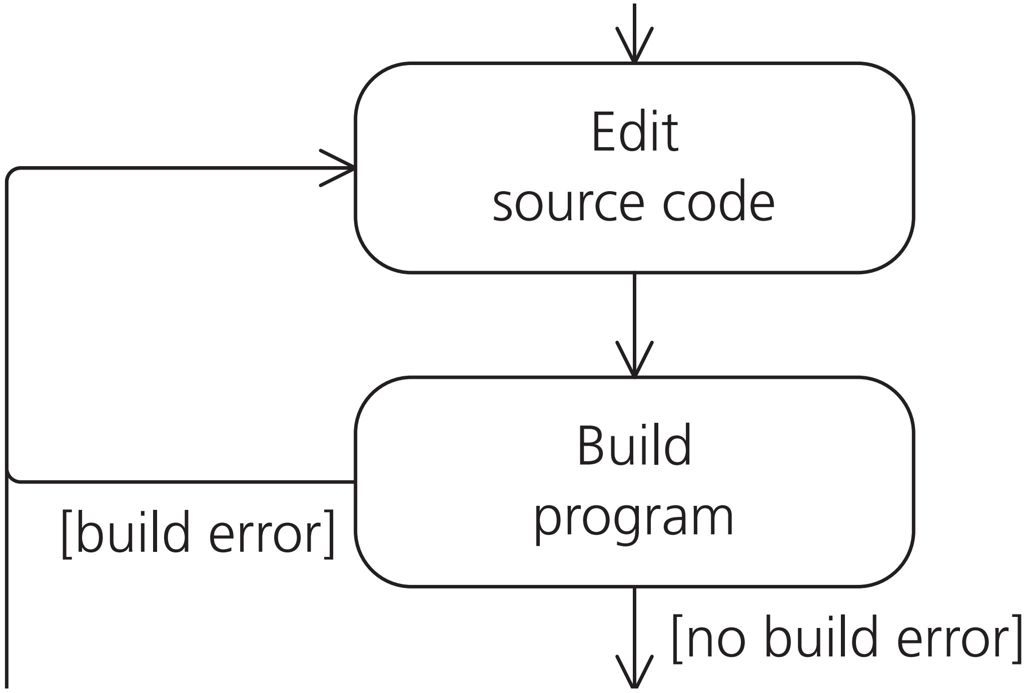
compiled by javac *generates* MainClass.class

generates Extra.class

(b) Two classes in the source code file

Build program: Compilation errors

- The compiler translates source code to byte code.
- It may detect errors in the source.
- The compiler will report any errors and terminate the compilation.
- The errors must be corrected in the source code and the compiler run again to compile the program.



```
> javac SimpleProgram.java
SimpleProgram.java:9: ')' expected
    System.out.println(proverb;                              // (5)
                              ^

1 error
```

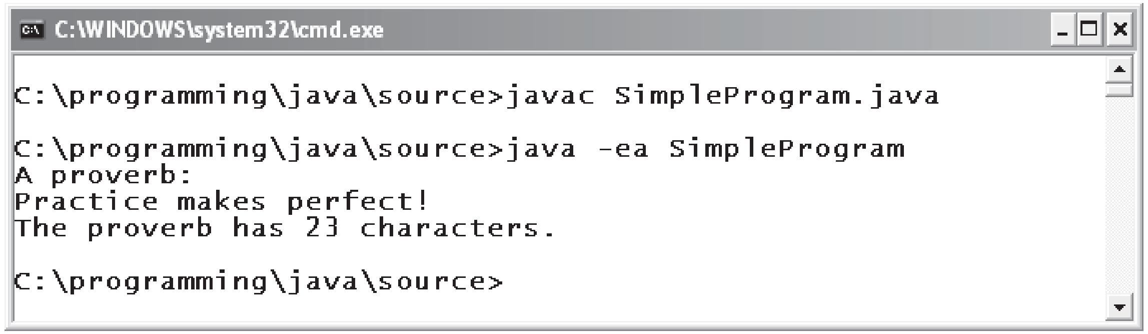
The "^" indicates where the error is located

(Oops, forgot the closing parenthesis.)

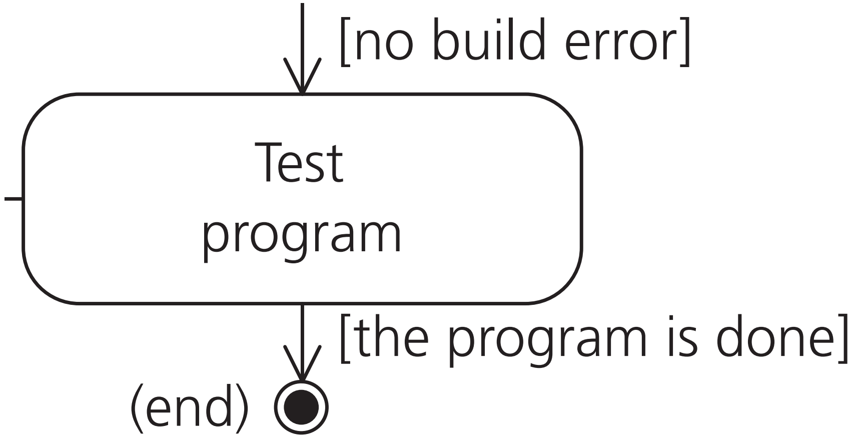
Running Java programs

```
> java -ea SimpleProgram
```

(Run this on the command-line)



```
C:\programming\java\source>javac SimpleProgram.java

C:\programming\java\source>java -ea SimpleProgram
A proverb:
Practice makes perfect!
The proverb has 23 characters.

C:\programming\java\source>
```



- Specify the exact class name, without any “`.class`” or “`.java`” extensions.
- Check the use of upper and lowercase letters in the class name.
- Make sure that the source code has been compiled.

Objects and Operations

How to make an omelette:

1. Open() the refrigerator
2. Take out() an egg carton
3. Open() the egg carton
4. Take out() two eggs
5. Close() the egg carton
6. ...

Legend:

Operation: operation name()

Object: object name

The *type* of the object determines the operations that can be performed on it:

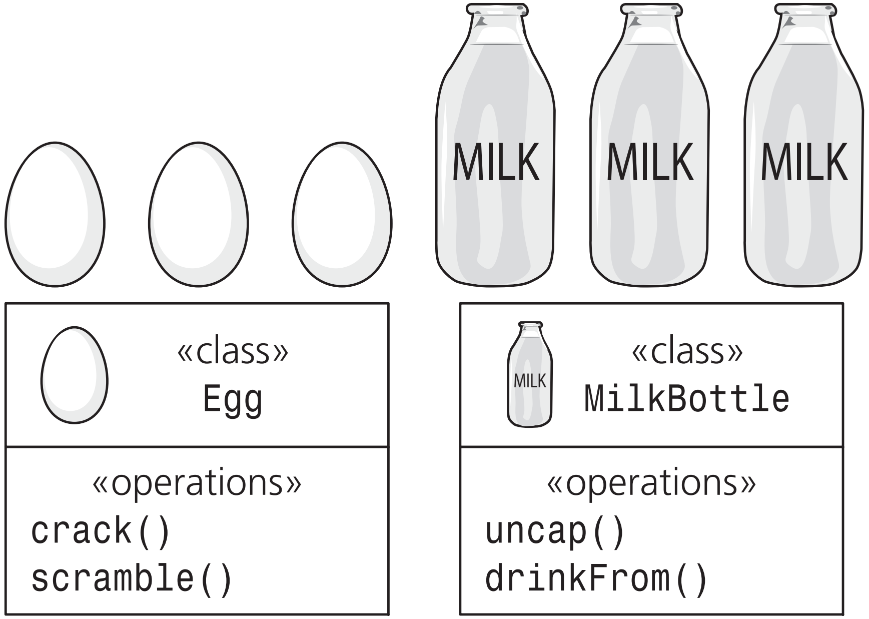
- ~~Open() the frying pan~~ *(a frying pan cannot be opened)*

Object based programming (OBP)

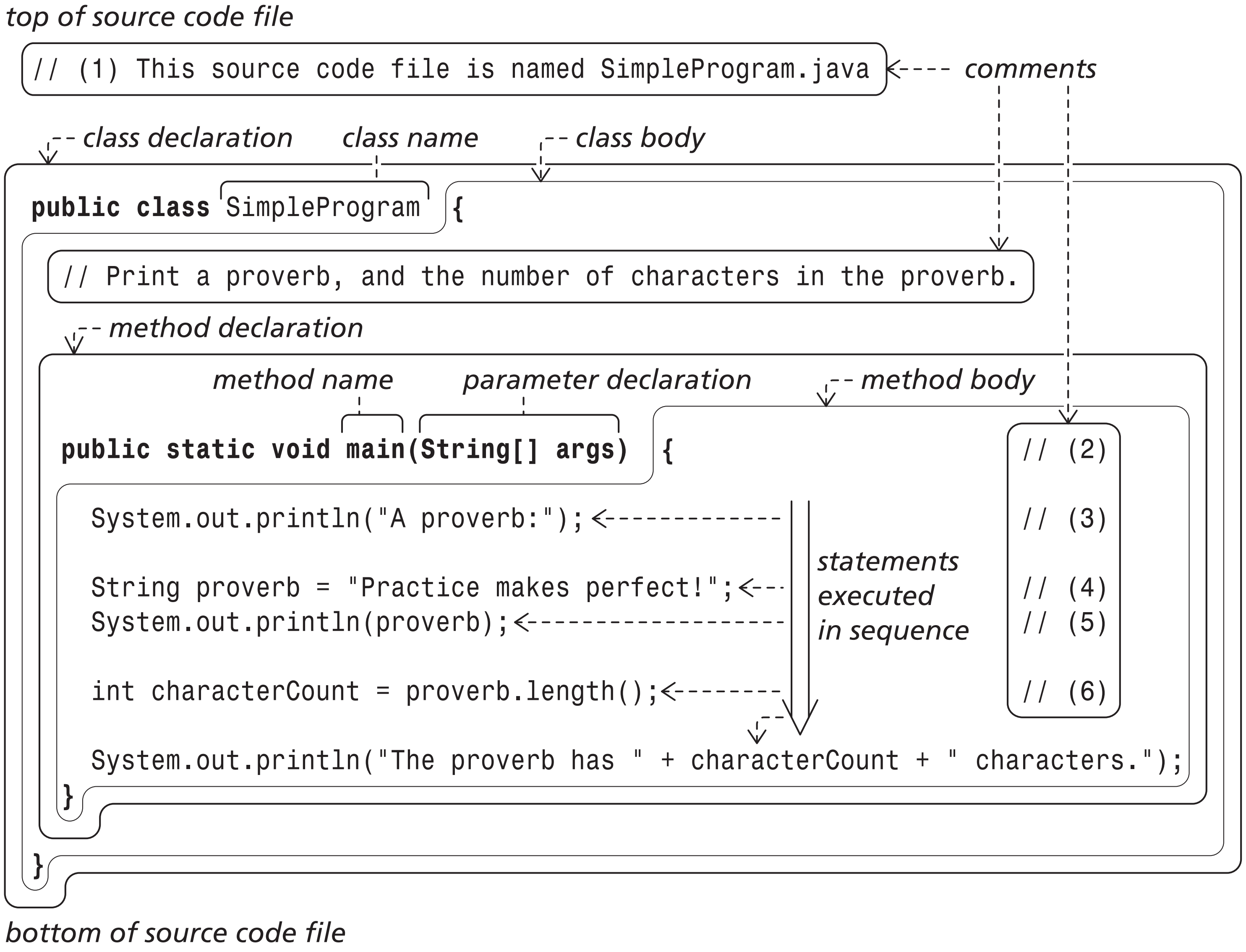
- Describing tasks as operations executed on objects.
- Define objects that are useful for the problem you’re trying to solve.
- E.g. for a program to keep track of library loans, create objects representing...
  - **tangible items**: books, journals, audio tapes
  - **non-tangible concepts**: lending date, information about library users
- Programs usually have more than one object of the same type.

Objects
multiple of each type

Classes
one for each type of object
(the class is the type)



The Java programming language



top of source code file

comments

class declaration

class name

class body

method declaration

method name

parameter declaration

method body

statements executed in sequence

bottom of source code file

```
// (1) This source code file is named SimpleProgram.java
public class SimpleProgram {
  // Print a proverb, and the number of characters in the proverb.
  public static void main(String[] args) {                                     // (2)
    System.out.println("A proverb:");                                           // (3)

    String proverb = "Practice makes perfect!";                                 // (4)
    System.out.println(proverb);                                                // (5)

    int characterCount = proverb.length();                                      // (6)

    System.out.println("The proverb has " + characterCount + " characters.");
  }
}
```

Comments and indentation

`// This is a source code comment.` *(ignored by compiler)*

This will technically work...

```
public static void main(String[]args){System.out.println("A proverb:");
String proverb="Practice makes perfect!";System.out.println(proverb);int
characterCount=proverb.length();System.out.println("The proverb has "+
characterCount+" characters.");}
```

...but don’t do it.
Please.

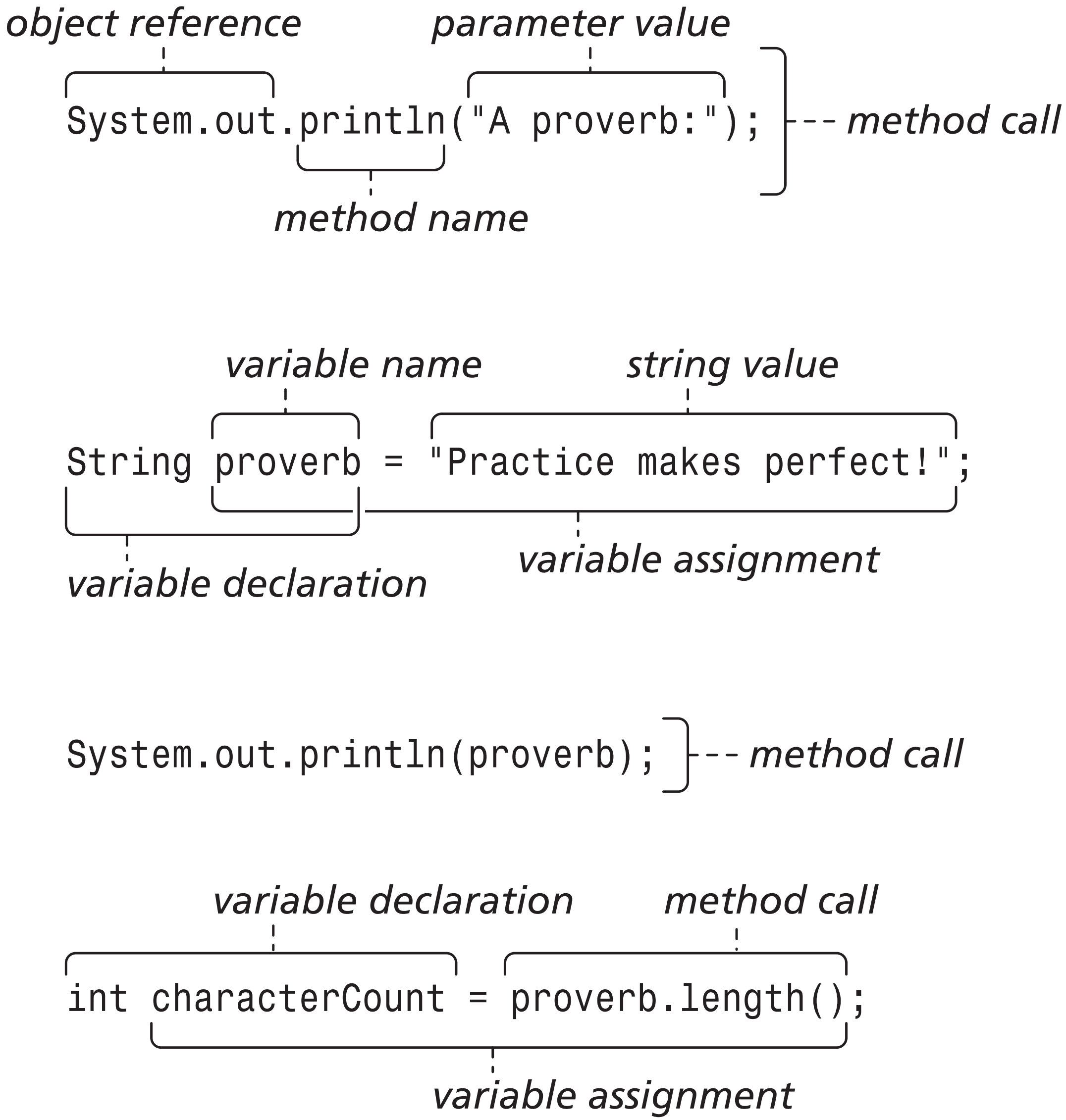
- Use proper indentation:
  - It makes the source code easier to read and modify.
  - **Java convention:** use four spaces for each indentation step

Program entry point

```
public static void main(String[] args) {
        ... method body containing statements that will be executed one by one...
}
```

- For a Java program to be executable, it must define exactly one `main()` method.
- For *very small* programs:
  - one source code file
  - primary class in the file that contains the `main()` method
- For *larger* programs:
  - split the source code into several files
  - one class in each file
  - only one file containing the `main()` method

Statements



object reference

parameter value

```
System.out.println("A proverb:");
```

method call

method name

variable name

string value

```
String proverb = "Practice makes perfect!";
```

variable declaration

variable assignment

```
System.out.println(proverb);
```

method call

variable declaration

method call

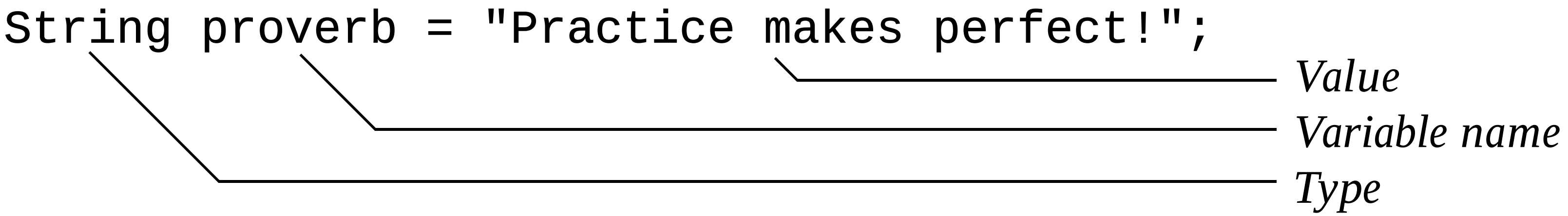
```
int characterCount = proverb.length();
```

variable assignment

Variables

- named locations in the computer's internal storage (memory)
- holds values during program execution
- often used by methods to hold intermediate results
- storing numeric values is very common
- storing other types of values is also possible

- Store a value in a variable:



```
String proverb = "Practice makes perfect!";
```

Value

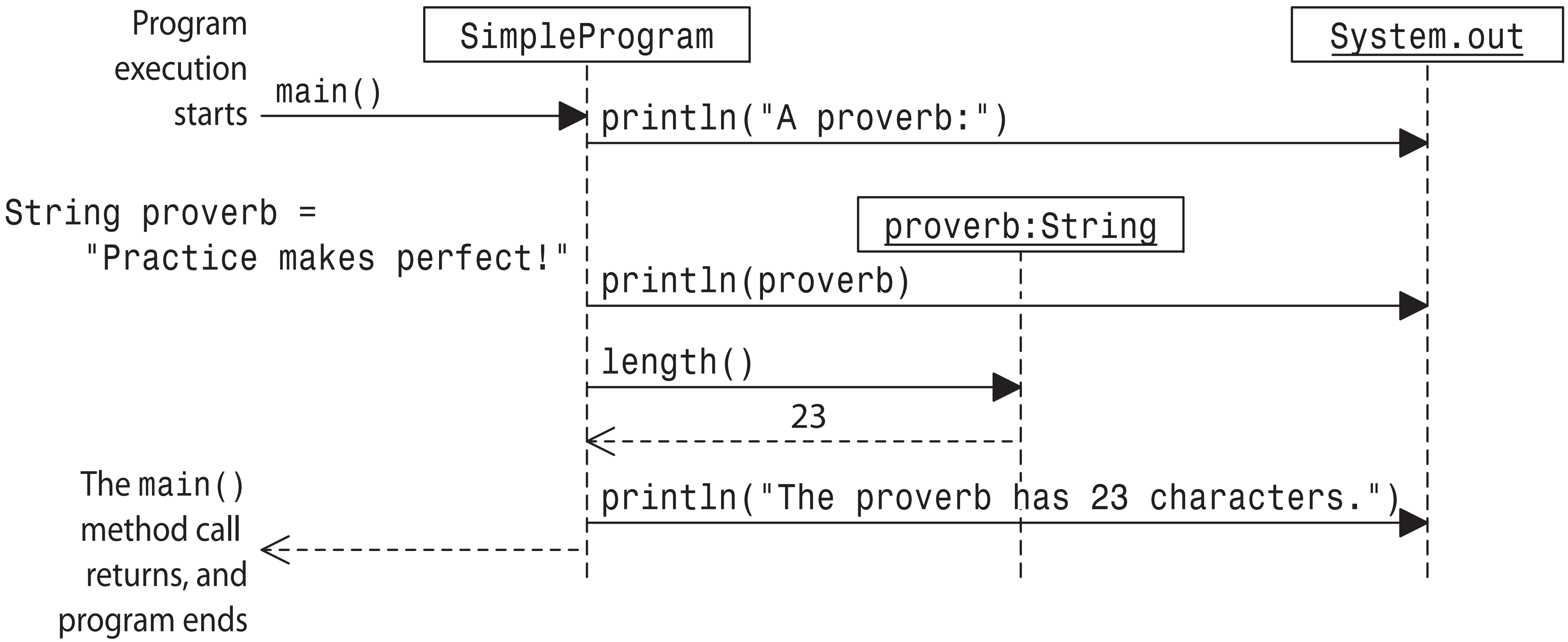
Variable name

Type

- Later, use value by referring to the variable:

```
System.out.println(proverb);
```

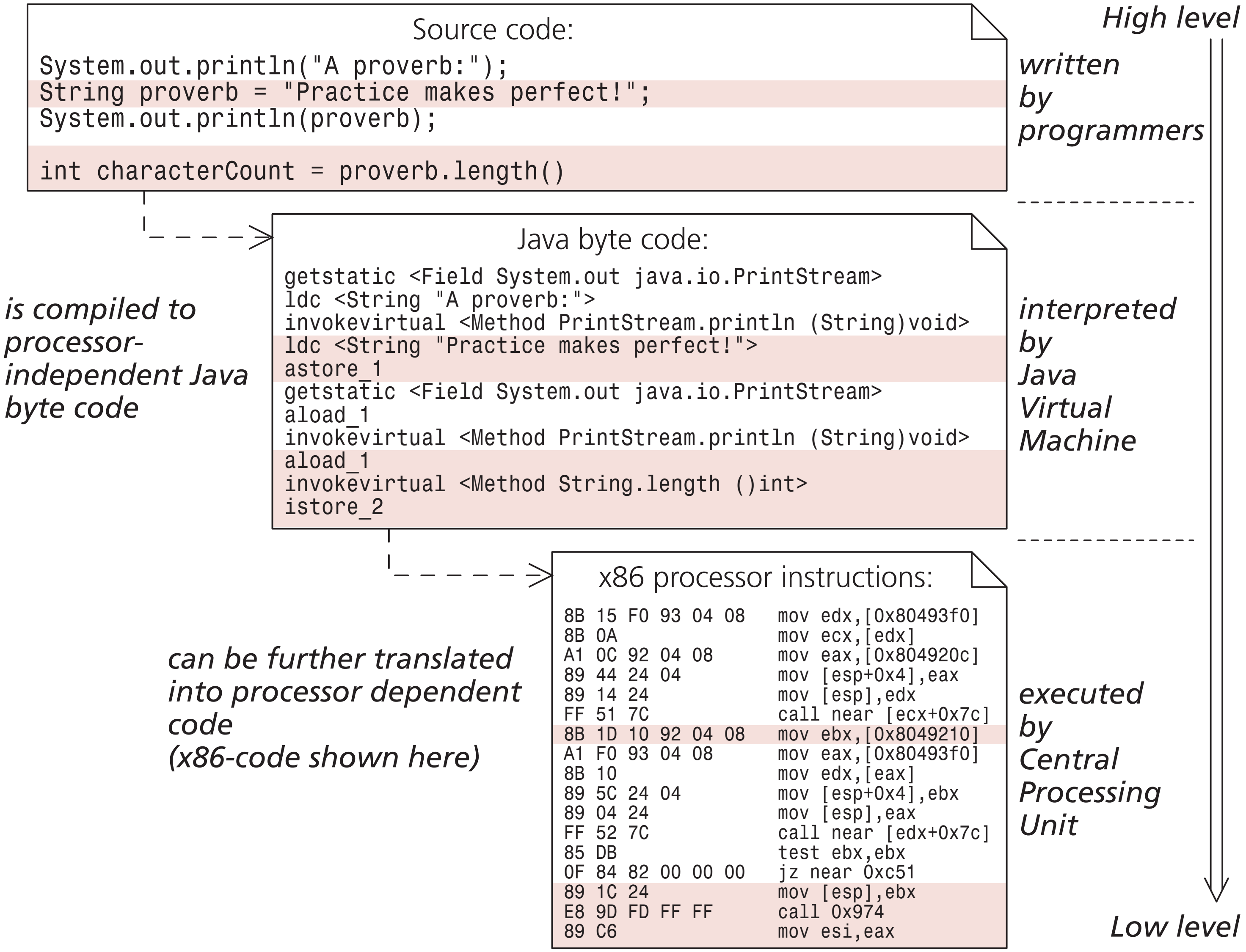
Sequence of method calls during program execution



Byte code and the Java Virtual Machine

- Java programming language:
  - a high-level language
  - provides a rich set of language
  - natural for humans to read
- Java byte code:
  - a low-level language
  - provides a small set of basic instructions
  - suited for execution by machines
  - platform independent
- Java Virtual Machine (JVM):
  - a program that interprets byte code instructions
  - not a physical machine...
    ...but behaves much in the same way as a central processing unit (CPU)
  - may virtual machines interpret the byte code directly
  - or recompile it to platform specific *machine code* during execution
  - implementations exist for several platforms (Windows, Solaris, Linux)

Program code at several levels



Source code:

```
System.out.println("A proverb:");
String proverb = "Practice makes perfect!";
System.out.println(proverb);

int characterCount = proverb.length()
```

is compiled to processor-independent Java byte code

Java byte code:

```
getstatic <Field System.out java.io.PrintStream>
ldc <String "A proverb:">
invokevirtual <Method PrintStream.println (String)void>
ldc <String "Practice makes perfect!">
astore_1
getstatic <Field System.out java.io.PrintStream>
aload_1
invokevirtual <Method PrintStream.println (String)void>
aload_1
invokevirtual <Method String.length ()int>
istore_2
```

can be further translated into processor dependent code (x86-code shown here)

x86 processor instructions:

```
8B 15 F0 93 04 08   mov edx,[0x80493f0]
8B 0A               mov ecx,[edx]
A1 0C 92 04 08      mov eax,[0x804920c]
89 44 24 04         mov [esp+0x4],eax
89 14 24            mov [esp],edx
FF 51 7C            call near [ecx+0x7c]
8B 1D 10 92 04 08   mov ebx,[0x8049210]
A1 F0 93 04 08      mov eax,[0x80493f0]
8B 10               mov edx,[eax]
89 5C 24 04         mov [esp+0x4],ebx
89 04 24            mov [esp],eax
FF 52 7C            call near [edx+0x7c]
85 DB               test ebx,ebx
0F 84 82 00 00 00   jz near 0xc51
89 1C 24            mov [esp],ebx
E8 9D FD FF FF      call 0x974
89 C6               mov esi,eax
```

High level

written by programmers

interpreted by Java Virtual Machine

executed by Central Processing Unit

Low level