Q. Why Java?

A. The programs that we are writing are very similar to their counterparts in several other languages, so our choice of language is not crucial. We use Java because it is widely available, embraces a full set of modern abstractions, and has a variety of automatic checks for mistakes in programs, so it is suitable for learning to pro- gram. There is no perfect language, and you certainly will be programming in other languages in the future.

Q. Do I really have to type in the programs in the book to try them out? I believe that you ran them and that they produce the indicated output.

A. Everyone should type in and run Hello World. Your understanding will be greatly magniﬁed if you also run Use Argument, try it on various inputs, and modify it to test different ideas of your own. To save some typing, you can ﬁnd all of the code in this book (and much more) on the booksite. This site also has information about installing and running Java on your computer, answers to selected exercises, web links, and other extra information that you may ﬁnd useful or interesting.

Q. What is the meaning of the words public, static and void?

A. These keywords specify certain properties of main() that you will learn about later in the book. For the moment, we just include these keywords in the code (be- cause they are required) but do not refer to them in the text.

Q. What is the meaning of the //, /\*, and \*/ character sequences in the code?

A. They denote comments, which are ignored by the compiler. A comment is either text in between

/\* and \*/ or at the end of a line after //. As with most online code, the code on the booksite is liberally annotated with comments that explain what it does; we use fewer comments in code in this book because the accompanying text and ﬁgures provide the explanation.

Q. What are Java’s rules regarding tabs, spaces, and newline characters?

A. Such characters are known as whitespace characters. Java compilers consider all whitespace in program text to be equivalent. For example, we could write Hel-

loWorld as follows: public class HelloWorld { public static void main ( String [] args) { System.out.print("Hello, World") ; System.out. println() ;} } But we do normally adhere to spacing and indenting conventions when we write Java programs, just as we always indent paragraphs and lines consistently when we write prose or poetry.

Q. What are the rules regarding quotation marks?

A. Material inside quotation marks is an exception to the rule deﬁned in the pre- vious question: things within quotes are taken literally so that you can precisely specify what gets printed. If you put any number of successive spaces within the quotes, you get that number of spaces in the output. If you accidentally omit a quotation mark, the compiler may get very confused, because it needs that mark to distinguish between characters in the string and other parts of the program.

Q. What happens when you omit a brace or misspell one of the words, like public or static or void or main?

A. It depends upon precisely what you do. Such errors are called syntax errors and are usually caught by the compiler.

For example,

if you make a program Bad that is exactly the same as HelloWorld except that you omit the line containing the ﬁrst left brace (and change the program name from HelloWorld to Bad), you get the following helpful message:

% javac Bad.java

Bad.java:2: '{' expected

public static void main(String[] args)

^

1 error

From this message, you might correctly surmise that you need to insert a left brace. But the compiler may not be able to tell you exactly what mistake you made, so the error message may be hard to understand. For example, if you omit the second left brace instead of the ﬁrst one, you get the following messages:

% javac Bad.java

Bad.java:4: ';' expected

System.out.print("Hello, World");

^

Bad.java:7: 'class' or 'interface' expected

}

^

Bad.java:8: 'class' or 'interface' expected

^

3 errors

One way to get used to such messages is to intentionally introduce mistakes into a simple program and then see what happens. Whatever the error message says, you should treat the compiler as a friend, for it is just trying to tell you that something is wrong with your program.

Q. Can a program use more than one command-line argument?

A. Yes, you can use many arguments, though we normally use just a few. Note that the count starts at 0, so you refer to the ﬁrst argument as args[0], the second one as args[1], the third one as args[2], and so forth.

Q. What Java methods are available for me to use?

A. There are literally thousands of them. We introduce them to you in a deliberate fashion (starting in the next section) to avoid overwhelming you with choices.

Q. When I ran UseArgument, I got a strange error message. What’s the problem? A. Most likely, you forgot to include a command-line argument:

% java UseArgument

Hi, Exception in thread “main”

java.lang.ArrayIndexOutOfBoundsException: 0

at UseArgument.main(UseArgument.java:6)

The JVM is complaining that you ran the program but did not type an argument as promised.