# CS3231
# Object Oriented Programming I

Name: _____ (  ) Date: _____

## Chapter 1: Introduction to Java

### Lesson Objectives

*After completing the lesson, the student will be able to*

- *describe the relationship between Java and the World Wide Web*
- *understand the meaning of Java language specification, API, JDK and IDE*
- *explain the basic syntax of a Java program*
- *create, compile and run Java programs*
- *use sound Java style and document programs properly*
- *explain the differences between syntax errors, runtime errors and logic errors*

### 1.1 A Brief History of Java

In 1991, Sun Microsystems was attempting to develop a new technology for programming next generation embedded chips in smart appliances, which Sun expected to be a major new opportunity. Java was developed by a team led by James Gosling at Sun Microsystems.

Initially, Gosling attempted to modify and extend C++ but soon abandoned that in favor of creating a new platform called Green and an entirely new language, which he called Oak, after the tree that stood just outside his office.

The idea of using extending C++ was rejected for several reasons. Because they were developing an embedded system with limited resources, they decided that C++ needed too much memory and that its complexity often led to developer errors.

The language's lack of garbage collection meant that programmers had to manually manage system memory, a challenging and error-prone task. The team also worried about the C++ language's lack of portable facilities for security, distributed programming, and threading. Finally, they wanted a platform that would port easily to all types of devices.

By the summer of 1992, they were able to demonstrate portions of the new platform including the Green OS, the Oak language, the libraries, and the hardware. Oak was renamed Java in 1994 after a trademark search revealed that Oak was used by Oak Technology. The language was also redesigned for developing Web applications. Java 1.0 was finally shipped in 1996. Sun Microsystems was purchased by Oracle in 2010. For the history of Java, see www.java.com/en/javahistory/index.jsp.

Java is often described as a Web programming language because of its use in writing programs called applets that run within a Web browser. That is, you need a Web browser to execute Java applets. Applets allow more dynamic and flexible dissemination of information on the Internet, and this feature alone makes Java an attractive language to learn. However, we are not limited to writing applets in Java. We can write Java applications also.

Figure 1.1 shows a good example of a dynamic web page that carries out sophisticated calculations. The Jmol applet displays molecular structures. By using the mouse, you can rotate and zoom each molecule to better understand its structure. This kind of direct manipulation is not achievable with static web pages, but applets make it possible.

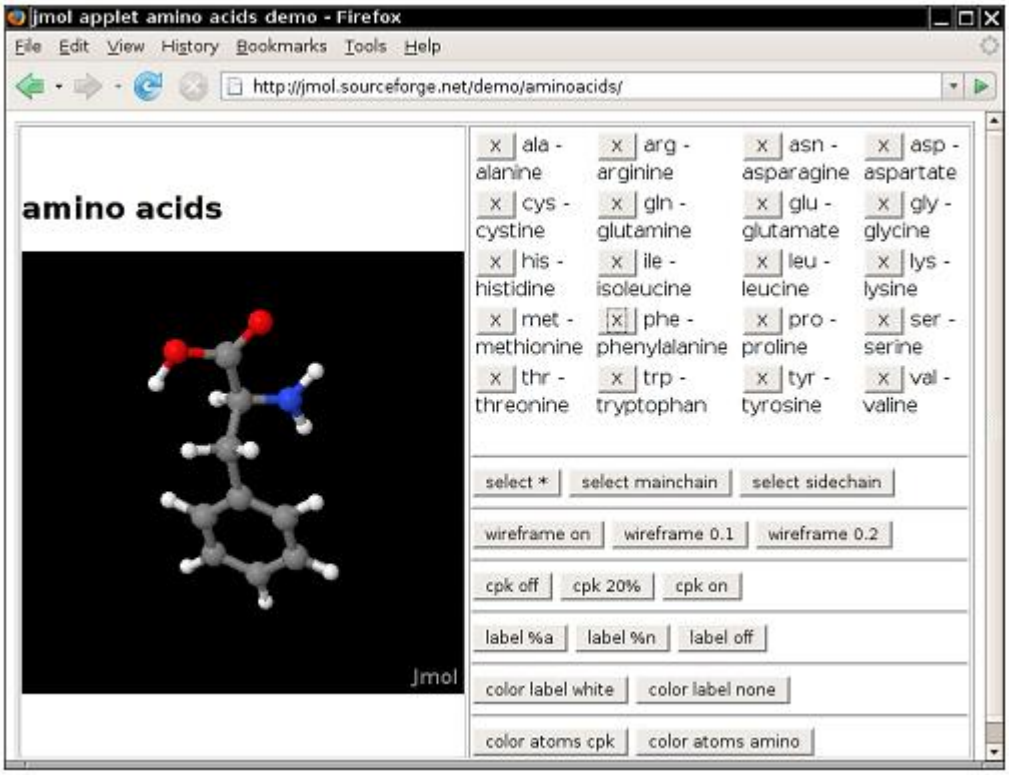


**Figure 1.1** Jmol applet

**A Java application is a complete stand-alone program that does not require a Web browser**. A Java application is analogous to a program we write in other programming languages. In this course, we will focus on Java applications only. Our objective is to teach the fundamentals of object-oriented programming that are applicable to all object-oriented programming languages.

Java is a versatile programming language: you can use it to develop applications for desktop computers, servers, and small handheld devices. The software for Android cell phones is developed using Java.

## 1.2 Features of Java

Java was designed so that anyone can execute programs in their browser without fear. The safety features of the Java language ensure that a program is terminated if it tries to do something unsafe. Having a safe environment is also helpful for anyone learning Java. When you make an error that result in unsafe behaviour, your program is terminated and you receive an accurate error report.

The other benefit of Java is portability. The same Java program will run, without change, on Windows, UNIX, Linux, or Macintosh. In order to achieve portability, the Java compiler does not translate Java programs directly into CPU instructions. Instead, compiled Java programs contain instructions for the Java Virtual Machine (JVM), a program that simulates a real CPU. Portability is another benefit for the beginning student. You do not have to learn how to write programs for different platforms.

Java has become enormously popular. Its rapid rise and wide acceptance can be traced to its design characteristics, particularly its promise that you can write a program once and run it anywhere. As stated by its designer, Java is simple, object oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high performance, multithreaded, and dynamic. *For more details, you may refer to the optional reading material: Characteristics of Java.*

In 2014, the release of Java 8 followed, with the most significant changes to the Java language in almost two decades. Table 1.1 summarizes all the versions since 1996. Java 8 embraces a “functional” style of programming that makes it easy to express computations that can be executed concurrently.

The flagship feature of Java 9 was ***Jigsaw project*** that introduced modularity to monolithic Java SE ecosystem. The primary goal of the Jigsaw project was to make the Java SE Platform and the JDK more easily scalable down to small computing devices.

Modularization of the JDK enables the source code to be completely restructured in order to make it easier to maintain. Module is a self-describing collection of code, data and resources. It consists of **module-info.java** file and one or more packages.

Modules offer stronger **encapsulation** than jars. JDK 9 onwards supports modularization. We will discuss more about modules in Java in the later part of the module.

In September 2017, Mark Reinhold, chief Architect of the Java Platform, proposed to change the release train to "one feature release every six months" rather than the current two-year schedule, and later the proposal took effect.

Java 8 and Java 11 are the currently supported long-term support (LTS) version and Java 10 is the previous supported rapid release version.

All programming languages must evolve to stay relevant, and Java has shown a remarkable capacity to do so.

| Version | Year | New Language Features |
|---|---|---|
| 1.0 | **1996** | **The language itself** |
| **1.1** | **1997** | **Inner classes** |
| **1.2** | **1998** | **The strictfp modifier** |
| **1.3** | **2000** | **None** |
| **1.4** | **2002** | **Assertions** |
| **5.0** | **2004** | **Generic classes, “for each” loop, autoboxing, metadata, enumerations, static import** |
| **6** | **2006** | **None** |
| **7** | **2011** | **Switch with strings, diamond operator, binary literals, exception handling enhancements** |
| **8** | **2014** | **Lambda expressions, interface with default methods, stream and date/time libraries** |
| **9** | **Jul 2017** | **The Java Platform module system. The defining feature for Java 9 is an all-new module system** |
| **10** | **Mar 2018** | **Local-Variable Type Inference etc.** |
| **11** | **Sept 2018** | **JavaFX is no longer included in the JDK.** |

**Table 1.1** Versions of Java

## 1.3 Creating, Compiling, and Executing a Java Program

You have to create your program and compile it before it can be executed (Figure 1.3). This process is repetitive, as shown in Figure 1.4. If your program has compile errors, you have to modify the program to fix them, and then recompile it. If your program has runtime errors or does not produce the correct result, you have to modify the program, recompile it, and execute it again.

Java source file must end with the extension `.java` file and if there aren't any syntax errors, the compiler generates a Java bytecode file with a `.class` extension. **The name of the source file must have the same exact name as the public class name**. Once compiled, the `.class` file (bytecode) can be executed by the Java Virtual Machine (JVM). For example, the file for the source code in Program 1-1 should be named `Welcome.java`, since the public class name is `Welcome`.

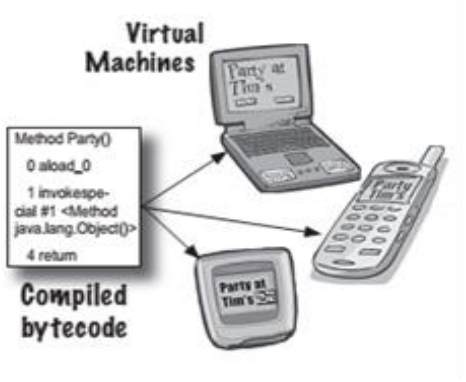


**Figure 1.2** Java Virtual Machines

The Java language is a high-level language, but Java bytecode is a low-level language. The bytecode is similar to machine instructions but is architecture neutral and can run on any platform that has a Java Virtual Machine (JVM) (Figure 1.2). Rather than a physical machine, the virtual machine is a program that interprets Java bytecode. This is one of Java's primary advantages: Java bytecode can run on a variety of hardware platforms and operating systems.

Java source code is compiled into Java bytecode and Java bytecode is interpreted by the JVM. Your Java code may use the code in the Java library. The JVM executes your code along with the code in the library. To execute a Java program is to run the program's bytecode. You can execute the bytecode on any platform with a JVM, which is an interpreter. It translates the individual instructions in the bytecode into the target machine language code one at a time rather than the whole program as a single unit. Each step is executed immediately after it is translated.

When executing a Java program, the JVM first loads the bytecode of the class to memory using a program called the **class loader**. If your program uses other classes, the class loader dynamically loads them just before they are needed. After a class is loaded, the JVM uses a program called the **bytecode verifier** to check the validity of the bytecode and to ensure that the bytecode does not violate Java's security restrictions. Java enforces strict security to make sure that Java class files are not tampered with and do not harm your computer. It is important to take note that if you execute a class file that does not exist, a NoClassDefFoundError will occur. If you execute a class file that does not have a `main` method or you mistype the `main` method (e.g., by typing Main instead of main), a NoSuchMethodError will occur.

You can use any text editor or IDE to create and edit a Java source-code file. Before you can compile a Java program, you need to install JDK (Java SE) on your system.

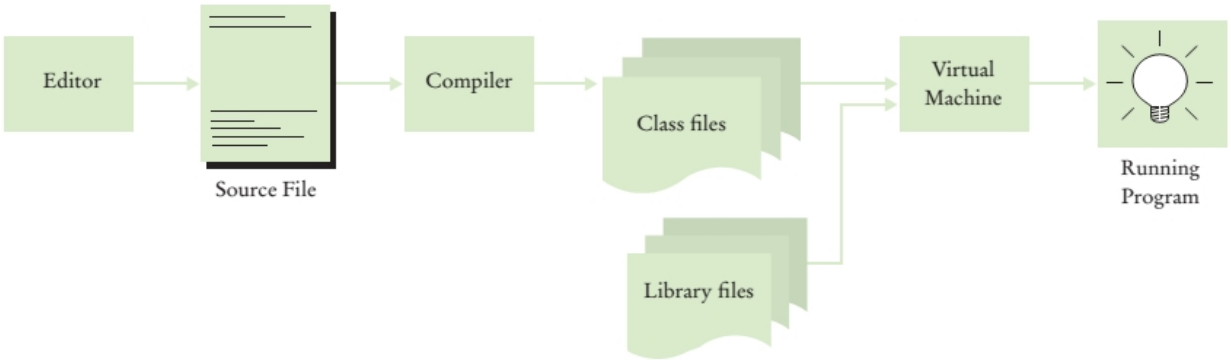

**Figure 1.3** From Source Code to Running Program

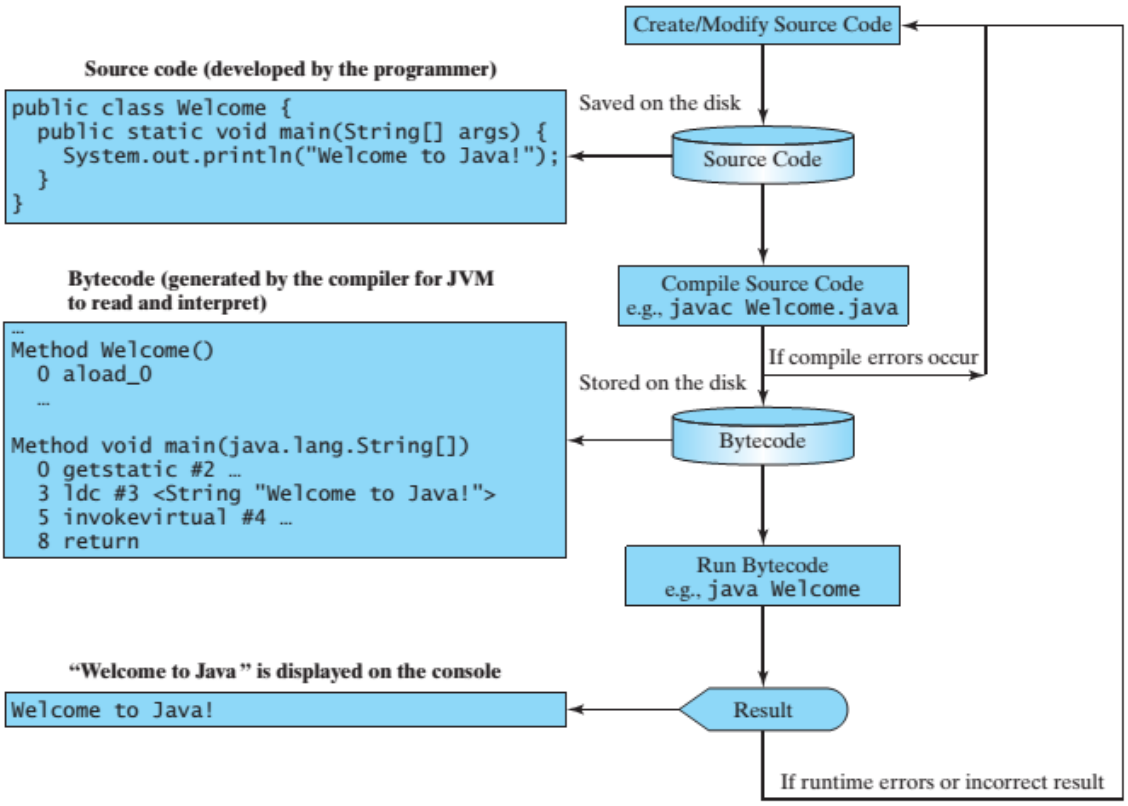


**Figure 1.4 The** Java program-development process consists of repeatedly creating/modifying source code, compiling, and executing programs.

A Java program is executed from the `main` method in the class. PROGRAM 1-1 is a simple Java program that displays the message Welcome to Java! on the console. (The word console is an old computer term that refers to the text entry and display device of a computer. Console input means to receive input from the keyboard, and console output means to display output on the monitor.)

**PROGRAM 1-1**

```
1  public class Welcome {
2   public static void main(String[] args) {
3   // Display message Welcome to Java! on the console
4   System.out.println("Welcome to Java!");
5   }
6  }
```

**PROGRAM OUTPUT**

```
Welcome to Java!
```

Line 1 defines a class. **Every Java program must have at least one class. Each class has a name. By convention, class names start with an uppercase letter.** In this example, the class name is `Welcome`.

Line 2 defines the `main` method. **The program is executed from the `main` method.** A class may contain several methods. **The `main` method is the entry point where the program begins execution. Java source programs are case sensitive. It would be wrong, for example, to replace `main` in the program with `Main`.**

A method is a construct that contains statements. The `main` method in this program contains the `System.out.println` statement. This statement displays the string Welcome to Java!. on the

console (line 4). String is a programming term meaning a sequence of characters. A string must be enclosed in double quotation marks. Every statement in Java ends with a semicolon `;`, known as the statement terminator.

**Reserved words, or keywords, have a specific meaning to the compiler and cannot be used for other purposes in the program**. For example, when the compiler sees the word `class`, it understands that the word after `class` is the name for the class. Other reserved words in this program are `public`, `static`, and `void`.

Line 3 is a comment that documents what the program is and how it is constructed. **Comments help programmers to communicate and understand the program. They are not programming statements and thus are ignored by the compiler.** In Java, comments are preceded by two slashes `//` on a line, called a line comment, or enclosed between `/*` and `*/` on one or several lines, called a block comment or paragraph comment. When the compiler sees `//`, it ignores all text after `//` on the same line. When it sees `/*`, it scans for the next `*/` and ignores any text between `/*` and `*/`. Here are examples of comments:

```
// This application program displays Welcome to Java!
/* This application program displays Welcome to Java! */
/* This application program
 displays Welcome to Java! */
```

A pair of curly braces in a program forms a block that groups the program's components. In Java, each block begins with an opening brace `{` and ends with a closing brace `}`. **Every class has a class block that groups the data and methods of the class**. Similarly, **every method has a method block that groups the statements in the method**. Blocks can be nested, meaning that one block can be placed within another, as shown in Figure 1.5.

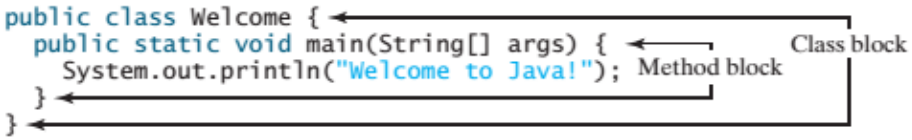


**Figure 1.5** Class block and method block

An opening brace must be matched by a closing brace. Whenever you type an opening brace, immediately type a closing brace to prevent the missing-brace error. Most Java IDEs automatically insert the closing brace for each opening brace.

You have seen several special characters (e.g., `{ }`, `//`, `;`) in the program. They are used in almost every program. Table 1.2 summarizes their uses. The most common errors you will make as you learn to program will be syntax errors. Like any programming language, Java has its own syntax, and you need to write code that conforms to the syntax rules. If your program violates a rule—for example, if the semicolon is missing, a brace is missing, a quotation mark is missing, or a word is misspelled—the Java compiler will report syntax errors. Try to compile the program with these errors and see what the compiler reports.

| Character | Name | Description |
|---|---|---|
| `{}` | Opening and closing braces | Denote a block to enclose statements. |
| `()` | Opening and closing parentheses | Used with methods. |
| `[]` | Opening and closing brackets | Denote an array. |
| `//` | Double slashes | Precede a comment line. |
| `" "` | Opening and closing quotation marks | Enclose a string (i.e., sequence of characters). |
| `;` | Semicolon | Mark the end of a statement. |

**Table 1.2** Summary of Java Versions

You are probably wondering why the `main` method is defined this way and why System.out.println(...) is used to display a message on the console. For the time being, simply accept that this is how things are done. Your questions will be fully answered in subsequent chapters.

## 1.4 Programming Style and Documentation

Good programming style and proper documentation make a program easy to read and help programmers prevent errors. Programming style deals with what programs look like. A program can compile and run properly even if written on only one line, but writing it all on one line would be bad programming style because it would be hard to read. Documentation is the body of explanatory remarks and comments pertaining to a program. Programming style and documentation are as important as coding. Good programming style and appropriate documentation reduce the chance of errors and make programs easy to read.

### 1.4.1 Appropriate Comments and Comment Styles

Include a summary at the beginning of the program that explains what the program does, its key features, and any unique techniques it uses. In a long program, you should also include comments that introduce each major step and explain anything that is difficult to read. It is important to make comments concise so that they do not crowd the program or make it difficult to read. In addition to line comments (beginning with `//`) and block comments (beginning with `/*`), Java supports comments of a special type, referred to as javadoc comments.

What is Javadoc?

Javadoc is a tool which comes with JDK and it is used for generating Java code documentation in HTML format from Java source code, which requires documentation in a predefined format. javadoc comments begin with `/**` and end with `*/`. They can be extracted into an HTML file using the JDK's javadoc command. PROGRAM 1-2 shows how a javadoc comments are written.

**PROGRAM 1-2**

```
1  /**
2  * The HelloWorld program implements an application that
3  * simply displays "Hello World!" to the standard output.
4  *
5  * @author  Claude Chua
6  * @version 1.0
7  */
8  public class HelloWorld {
9     public static void main(String[] args) {
10       // Prints Hello, World! on standard output.
11       System.out.println("Hello World!");
12    }
13 }
```

Use javadoc comments `/** ... */` for commenting on an entire class or an entire method. These comments must precede the class or the method header in order to be extracted into a javadoc HTML file. For commenting on steps inside a method, use line comments `//`.

To see an example of a javadoc HTML file generation, check out: *Introduction to JavaDoc*.

- https://www.youtube.com/watch?v=CJxMwbJPisw
- https://www.youtube.com/watch?v=Ls-NHebXY20

### 1.4.2 Proper Indentation and Spacing

A consistent indentation style makes programs clear and easy to read, debug, and maintain. Indentation is used to illustrate the structural relationships between a program's components or statements. Java can read the program even if all of the statements are on the same long line, but humans find it easier to read and maintain code that is aligned properly. Indent each subcomponent or statement at least two spaces more than the construct within which it is nested.

A single space should be added on both sides of a binary operator, as shown in the following statement:

```
System.out.println(3+4*4);          Bad style
System.out.println(3 + 4 * 4);      Good style
```

### 1.4.3 Block Styles

A block is a group of statements surrounded by braces. There are two popular styles, next-line style and end-of-line style, as shown below.

```
public class Test
{
  public static void main(String[] args)
  {
    System.out.println("Block Styles");
  }
}
```

Next-line style

```
public class Test {
  public static void main(String[] args) {
    System.out.println("Block Styles");
  }
}
```

End-of-line style

The next-line style aligns braces vertically and makes programs easy to read, whereas the end-of-line style saves space and may help avoid some subtle programming errors. Both are acceptable block styles. The choice depends on personal or organizational preference. You should use a block style consistently—mixing styles is not recommended.

## 1.5 Programming Errors

Programming errors can be categorized into three types: syntax errors, runtime errors, and logic errors.

### 1.5.1 Syntax Errors

Errors that are detected by the compiler are called **syntax errors** or compile errors. Syntax errors result from errors in code construction, such as mistyping a keyword, omitting some necessary punctuation, or using an opening brace without a corresponding closing brace. These errors are usually easy to detect because the compiler tells you where they are and what caused them. For example, PROGRAM 1-3 has a syntax error.

**PROGRAM 1-3**

```
1  public class ShowSyntaxErrors {
2   public static void main(String[] args) {
3    System.out.println("Welcome to Java);
4   }
5  }
```

If you were to run the program four errors are reported, but the program actually has two errors:

- The keyword `void` is missing before main in line 2.
- The string Welcome to Java should be closed with a closing quotation mark in line 3.

Since a single error will often display many lines of compile errors, it is a good practice to fix errors from the top line and work downward. Fixing errors that occur earlier in the program may also fix additional errors that occur later.

### 1.5.2 Runtime Errors

Runtime errors are errors that cause a program to terminate abnormally. They occur while a program is running if the environment detects an operation that is impossible to carry out. Input mistakes typically cause runtime errors. An input error occurs when the program is waiting for the user to enter a value, but the user enters a value that the program cannot handle. For instance, if the program expects to read in a number, but instead the user enters a string, this causes data-type errors to occur in the program.

Another example of runtime errors is division by zero. This happens when the divisor is zero for integer divisions. For instance, PROGRAM 1-4 would cause a runtime error.

**PROGRAM 1-4**

```
1  public class ShowSyntaxErrors {
2   public static void main(String[] args) {
3    System.out.println(1 / 0);
4   }
5  }
```

### 1.5.3 Logic Errors

Logic errors occur when a program does not perform the way it was intended to. Errors of this kind occur for many different reasons. For example, PROGRAM 1-5 attempts to convert Celsius 35 degrees to a Fahrenheit degree:

**PROGRAM 1-5**

```
1  public class ShowSyntaxErrors {
2   public static void main(String[] args) {
3    System.out.println("Celsius 35 is Fahrenheit degree ");
4    System.out.println((9 / 5) * 35 + 32);
5   }
6  }
```

**PROGRAM OUTPUT**

```
Celsius 35 is Fahrenheit degree
67
```

You will get Fahrenheit 67 degrees, which is wrong. It should be 95.0. In Java, the division for integers is the quotient—the fractional part is truncated—so in Java 9 / 5 is 1. To get the correct result, you need to use 9.0 / 5, which results in 1.8.

In general, syntax errors are easy to find and easy to correct because the compiler gives indications as to where the errors came from and why they are wrong. Runtime errors are not difficult to find, either, since the reasons and locations for the errors are displayed on the console when the program aborts. Finding logic errors, on the other hand, can be very challenging.

## 1.6 Common Errors

Missing a closing brace, missing a semicolon, missing quotation marks for strings, and misspelling names are common errors for new programmers.

**Common Error 1: Missing Braces**
The braces are used to denote a block in the program. Each opening brace must be matched by a closing brace. A common error is missing the closing brace. To avoid this error, type a closing brace whenever an opening brace is typed, as shown in the following example.

```
public class Welcome {

}
```
← Type this closing brace right away to match the opening brace

If you use an IDE such as Eclipse, the IDE automatically inserts a closing brace for each opening brace typed.

**Common Error 2: Missing Semicolons**
Each statement ends with a statement terminator (;). Often, a new programmer forgets to place a statement terminator for the last statement in a block, as shown in the following example.

```
public static void main(String[] args) {
  System.out.println("Programming is fun!");
  System.out.println("Fundamentals First");
  System.out.println("Problem Driven")
}
```
↑ Missing a semicolon

**Common Error 3: Missing Quotation Marks**
A string must be placed inside the quotation marks. Often, a new programmer forgets to place a quotation mark at the end of a string, as shown in the following example.

```
System.out.println("Problem Driven );
```
↑ Missing a quotation mark

If you use an IDE such as Eclipse, the IDE automatically inserts a closing quotation mark for each opening quotation mark typed.

**Common Error 4: Misspelling Names**
Java is case sensitive. Misspelling names is a common error for new programmers. For example, the word main is misspelled as Main and String is misspelled as string in the following code.

```
public class Test {
  public static void Main(string[] args) {
    System.out.println((10.5 + 2 * 3) / (45 - 3.5));
  }
}
```

**[Reference]**

[1] Core Java Volume 1 – Fundamentals 10th Ed, Cay Horstmann, 2016.
[2] Introduction to Java Programming Comprehensive Version 10th Ed, Daniel Liang, 2016.
[3] Big Java: Early Objects 6th Ed, Cay Horstmann, 2016.
[4] https://en.wikipedia.org/wiki/Java_(software_platform)#History