Chapter 1

Introduction

Objectives

- To understand the meaning of Java language specification, JDK, and IDE.
- To use Java programming style and document programs properly
- To explain the differences between syntax errors, runtime errors, and logic errors
- To develop Java programs using Eclipse

What is programming?

- *Programming* means to create (or develop) software, which is also called a *program.*
- Software developers create software with the help of powerful tools called *programming languages.*

Programming Languages

- There are hundreds of programming languages, and they were developed to make the programming process easier for people.
- This book teaches you how to create programs by using the Java programming language.
- There is no “best” language. Each one has its own strengths and weaknesses.
- However, programs must be converted into the instructions the computer can execute.

Popular high level programming languages

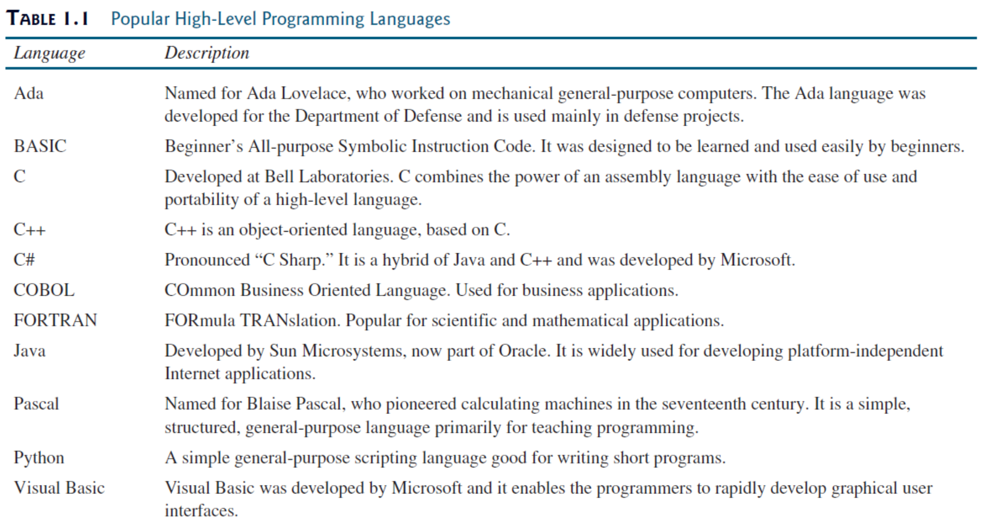


TABLE 1.1 Popular High-Level Programming Languages

Language	*Description*
Ada	Named for Ada Lovelace, who worked on mechanical general-purpose computers. The Ada language was developed for the Department of Defense and is used mainly in defense projects.
BASIC	Beginner's All-purpose Symbolic Instruction Code. It was designed to be learned and used easily by beginners.
C	Developed at Bell Laboratories. C combines the power of an assembly language with the ease of use and portability of a high-level language.
C++	C++ is an object-oriented language, based on C.
C#	Pronounced "C Sharp." It is a hybrid of Java and C++ and was developed by Microsoft.
COBOL	COmmon Business Oriented Language. Used for business applications.
FORTRAN	FORmula TRANslation. Popular for scientific and mathematical applications.
Java	Developed by Sun Microsystems, now part of Oracle. It is widely used for developing platform-independent Internet applications.
Pascal	Named for Blaise Pascal, who pioneered calculating machines in the seventeenth century. It is a simple, structured, general-purpose language primarily for teaching programming.
Python	A simple general-purpose scripting language good for writing short programs.
Visual Basic	Visual Basic was developed by Microsoft and it enables the programmers to rapidly develop graphical user interfaces.

Interpreter vs Compiler

- A program written in a high-level language is called a *source program* or *source code.*
- A source program must be translated into machine code for execution.
- The translation can be done using an *interpreter* or a *compiler.*
  - An interpreter reads one statement from the source code, translates it to the machine code or virtual machine code, and then executes it right away.
    An *interpreter* interprets a computer program into machine language at runtime.
  - A compiler translates the entire source code into a machine-code file, and the machine-code file is then executed.
    That is why compiled programs are usually faster because translation is not necessary if no changes are made to the program.

Interpreter vs Compiler (cont'd)

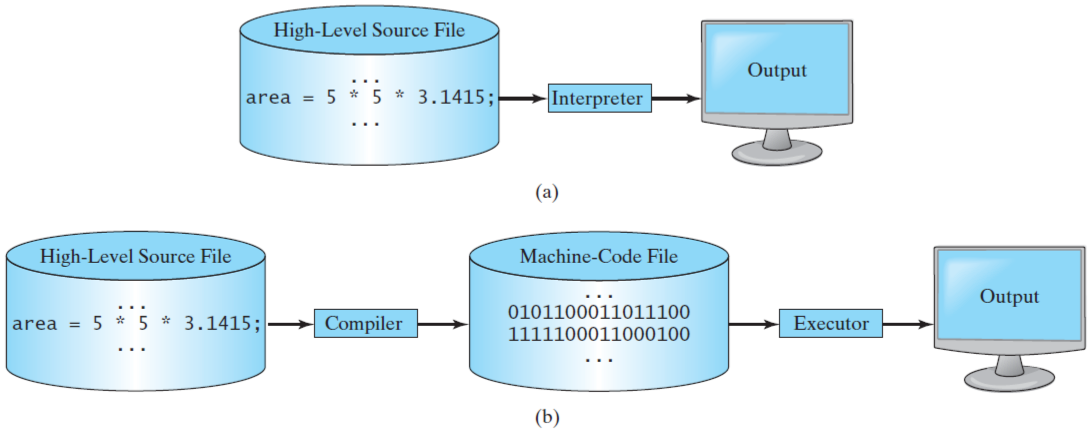


FIGURE 1.4 (a) An interpreter translates and executes a program one statement at a time. (b) A compiler translates the entire source program into a machine-language file for execution.

Java

- Java was developed by a team led by James Gosling at Sun Microsystems. Sun Microsystems was purchased by Oracle in 2010.
- As stated by its designer, Java is *simple*, *object oriented*, *distributed*, *robust*, *secure*, *architecture neutral*, *portable*, *high performance*, *multithreaded*, and *dynamic.*
- Java is a versatile programming language: you can use it to develop applications for desktop computers, servers, and small handheld devices.

Java Language Specification & JDK

- Java syntax is defined in the Java language specification. http://docs.oracle.com/javase/specs/
- The Java Development Kit (JDK), officially named "Java Platform Standard Edition" or "Java SE", is **needed for writing and running Java programs**. https://www.oracle.com/technetwork/java/javase/downloads/index.html
  - **Java 8 and Java 11 are listed because they are LTS (long term support)**
- **You must first install and configure the JDK before you can compile and run programs.**

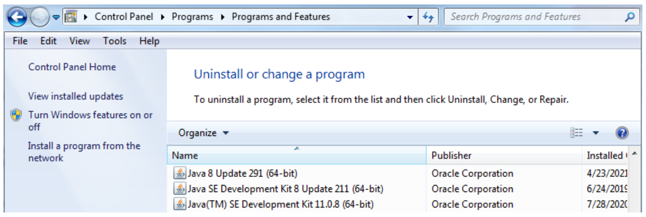
https://dzone.com/articles/difference-between-jdk-vs-jre-vs-jvm

Java Standard Edition (Java SE)

- There are many versions of Java SE (https://en.wikipedia.org/wiki/Java_version_history).
- The latest version is Java SE 18, with Java SE 17 being the latest LTS having support up to September 2029.
- LTS stands for Long Term Support.
- The version in MPI lab in Java SE 8 (LTS) having support for non-commercial use up to December 2030.

What software do you need? – (1) JDK

- **Java SE Development Kit**: Check if jdk is installed in your computer
  - Go to Control Panel-->Program and Features and **check** if **Java /JDK** is listed there.


Control Panel ▸ Programs ▸ Programs and Features
Search Programs and Features
File Edit View Tools Help
Control Panel Home
View installed updates
Turn Windows features on or off
Install a program from the network
Organize

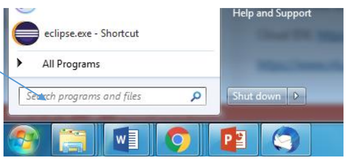
Uninstall or change a program

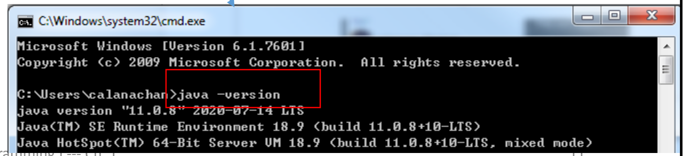
To uninstall a program, select it from the list and then click Uninstall, Change, or Repair.

Name	Publisher	Installed
Java 8 Update 291 (64-bit)	Oracle Corporation	4/23/2021
Java SE Development Kit 8 Update 211 (64-bit)	Oracle Corporation	6/24/2019
Java(TM) SE Development Kit 11.0.8 (64-bit)	Oracle Corporation	7/28/2020

- Or, open command prompt (type 'cmd') and type **java -version**. If you get the **version** info, **Java** is installed correctly and PATH is also set correctly.

Type 'cmd' here





```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\calanachan>java -version
java version "11.0.8" 2020-07-14 LTS
Java(TM) SE Runtime Environment 18.9 (build 11.0.8+10-LTS)
Java HotSpot(TM) 64-Bit Server VM 18.9 (build 11.0.8+10-LTS, mixed mode)
```

What software do you need? – (2) IDE

- We need some tools (IDE, **integrated development environment**) to help us write Java programs efficiently.
- We will use Eclipse as the IDE for Java programming
- You need to first install Java Development Kit (JDK), and then Eclipse.

- Java JDK:
  - For JDK 8, https://www.oracle.com/java/technologies/downloads/#java8
  - For the latest Java SE, see https://www.oracle.com/technetwork/java/javase/downloads/index.html
- For Eclipse (PC version):
  - Eclipse Neon 3 (released on 03/23/2017, run on Java 8) from https://www.eclipse.org/downloads/packages/release/neon/3
  - For the latest Eclipse version (which needs at least JDK 11), see https://www.eclipse.org/downloads/

What software do you need? – MacBook users

For MacBook --- https://www.cs.dartmouth.edu/~scot/cs10/mac_install/mac_install.html

- Install jdk 1.8 for mac (the version that I provided in Canvas is for PC only)

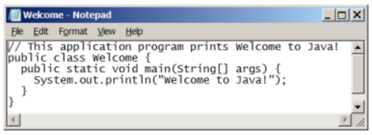
--- https://www.oracle.com/java/technologies/downloads/#java8-mac

- However, if your Macbook was purchased after 2020, the CPU is no longer Intel but ARM architecture (aarch64). In that case, you have to download the latest version of eclipse of aarch64 format and that means you have to use at least JDK 11.

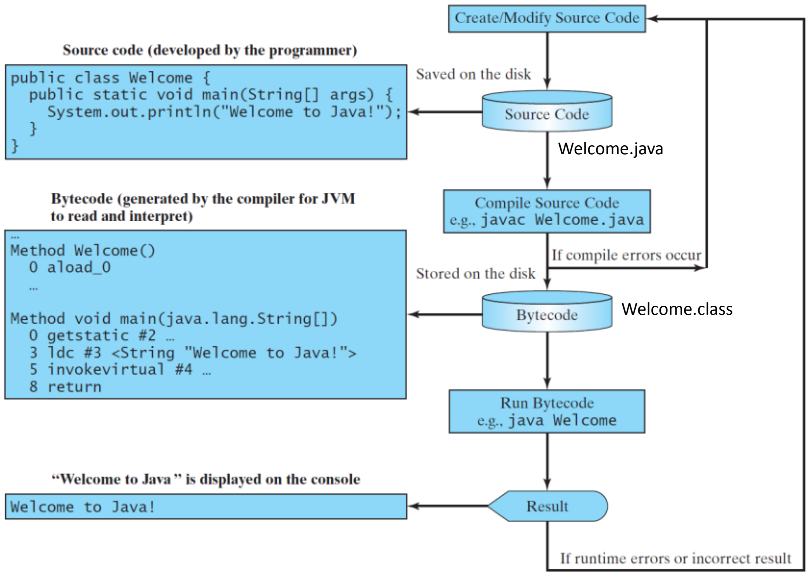
Some tips regarding the JDK

- **Can multiple versions of JDK be installed in one computer?**
  - Yes, you **can**. Just use full path names when invoking javac, java , etc. or set your PATH environment variable to point to the appropriate **jdk**/bin location.
  - The IDEs usually allow defining **multiple** JDKs/JREs, and you **can** choose which one to use for every project.
  - However, bear in mind that some configurations are needed. https://www.happycoders.eu/java/how-to-switch-multiple-java-versions-windows/
  - For the beginning, let's just stick with one version, Java JDK 8 for this module.

Creating, Compiling, and Running Programs



```
// This application program prints Welcome to Java!
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```



Source code (developed by the programmer)

```
public class Welcome {
   public static void main(String[] args) {
      System.out.println("Welcome to Java!");
   }
}
```

Bytecode (generated by the compiler for JVM to read and interpret)

```
…
Method Welcome()
   0 aload_0
   …

Method void main(java.lang.String[])
   0 getstatic #2 …
   3 ldc #3 <String "Welcome to Java!">
   5 invokevirtual #4 …
   8 return
```

"Welcome to Java" is displayed on the console

```
Welcome to Java!
```

- Class name
- Main method
- Statements
- Statement terminator
- Comments

A simple Java program

- A Java program is executed from the **main** method in the class.
- The name of the file must be the same as the name of the class.

LISTING 1.1 Welcome.java

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

Every Java program has at least one class. Class names start with an uppercase letter.

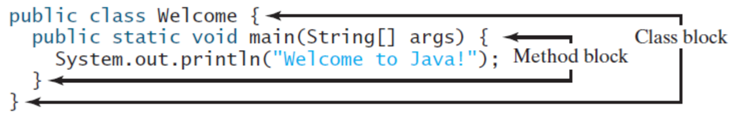
A class may contain several methods. The **main** method is where the program begins execution.

Comments are not programming statements and are ignored by the compiler.

This statement displays the string **Welcome to Java!**

Class block, Method block

- A pair of curly braces in a program forms a *block* that groups the program's components. A block begins with an opening brace (**{**) and ends with a closing brace (**}**).



```
public class Welcome {
  public static void main(String[] args) {
    System.out.println("Welcome to Java!");
  }
}
```

- In a method, there are *statements*, which include variable declarations, assignments, *invocations* to methods and control flow statements.
- Java source programs are case sensitive. It would be wrong, for example, to replace **main** in the program with **Main.**

Some rules regarding the Java Program

- Every statement in Java ends with a semicolon (;), known as the *statement terminator.*
- A string must be enclosed in double quotation marks.

LISTING 1.1 Welcome.java

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

- *Reserved words*, or *keywords*, have a specific meaning to the compiler and cannot be used for other purposes in the program. Examples of reserved words in this program are **class, public, static,** and **void**.
- A *line comment is* preceded by two slashes (**//**)*,* or enclosed between **/*** and ***/** on one or several lines, called a *block comment* or *paragraph comment*.

Special characters

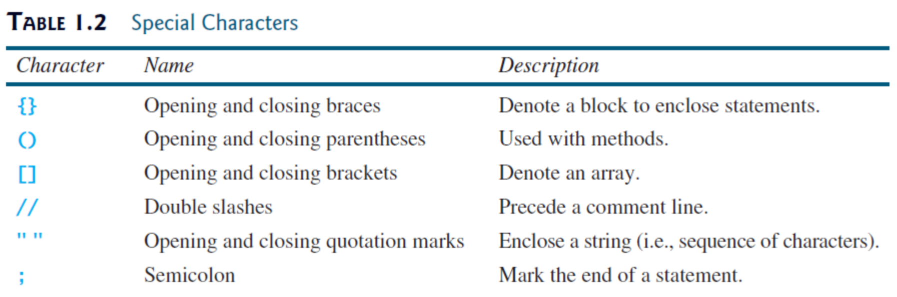


TABLE 1.2 Special Characters

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.

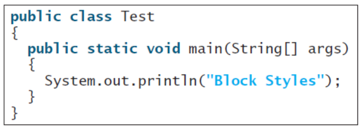
- If your program violates a rule—for example, if the semicolon is missing, a brace is missing, a quotation mark is missing, or a word is misspelled—the Java compiler will report syntax errors.

Creating and Compiling a Java Program

- *You save a Java program in a .java file and compile it into a .class file. The .class file is executed by the Java Virtual Machine.*
- 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.
- The source file must end with the extension **.java** and must have the same exact name as the public class name. For example, the file for the source code in Listing 1.1 should be named **Welcome.java**, since the public class name is **Welcome**.
- If there aren't any syntax errors, the *compiler* generates a bytecode file with a **.class** extension. Thus, the preceding command generates a file named **Welcome.class**

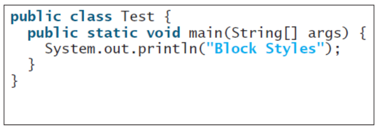
Programming style and documentation

- Proper Indentation and Spacing
  - *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 line, but humans find it easier to read and maintain code that is aligned properly.
- 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.



```
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");
  }
}
```

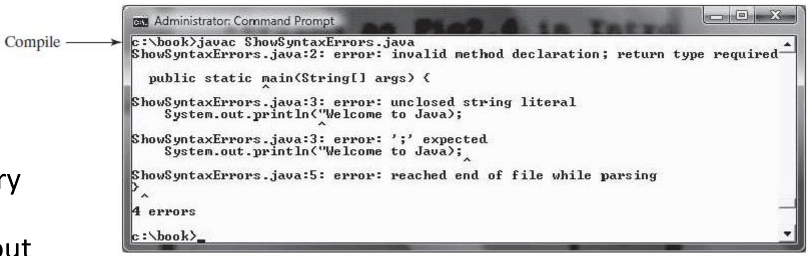
End-of-line style

Programming Errors

- *Programming errors can be categorized into three types: syntax errors, runtime errors, and logic errors.*
- Syntax Errors
  - Detected by the compiler.
  - 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
  - Causes the program to abort.
  - Not difficult to find, either, since the reasons and locations for the errors are displayed on the console when the program aborts.
- Logic Errors
  - Produces incorrect result.
  - Finding logic errors, on the other hand, can be very challenging. In the upcoming chapters, you will learn the techniques of tracing programs and finding logic errors.

Programming Errors – Syntax errors

- 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.



```
c:\book>javac ShowSyntaxErrors.java
ShowSyntaxErrors.java:2: error: invalid method declaration; return type required
  public static main(String[] args) {
                ^
ShowSyntaxErrors.java:3: error: unclosed string literal
    System.out.println("Welcome to Java);
                       ^
ShowSyntaxErrors.java:3: error: ';' expected
    System.out.println("Welcome to Java);
                                         ^
ShowSyntaxErrors.java:5: error: reached end of file while parsing
}
 ^
4 errors

c:\book>_
```

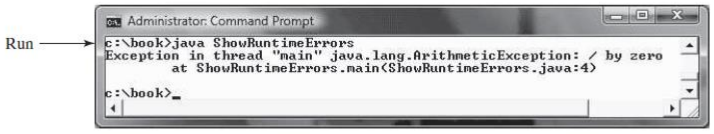
Four errors are reported, but the program actually has two errors:
- The keyword **void** is missing before **main** in line 2.
- The string **Welcome to Java** should be closed with a closing quotation mark in line 3.

Programming Errors – Runtime errors

- 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. 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.

LISTING 1.5 ShowRuntimeErrors.java

```java
public class ShowRuntimeErrors {
  public static void main(String[] args) {
    System.out.println(1 / 0);
  }
}
```

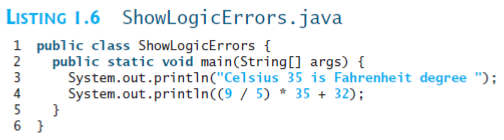


```
c:\book>java ShowRuntimeErrors
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at ShowRuntimeErrors.main(ShowRuntimeErrors.java:4)

c:\book>_
```

FIGURE 1.11 The runtime error causes the program to terminate abnormally.

Programming Errors – Logic errors

- 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, suppose you wrote the program in Listing 1.6 to convert Celsius **35** degrees to a Fahrenheit degree:



LISTING 1.6 ShowLogicErrors.java

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

  - 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**.

Eclipse IDE

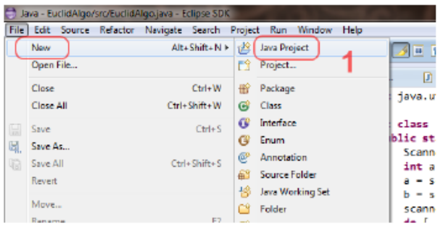
- You can use a Java development tool (e.g., NetBeans, Eclipse, and TextPad)—software that provides an *integrated development environment (IDE)* for developing Java programs quickly.
- Eclipse IDE 2020-06 is still be able to run against Java 8. But here is a reminder that further releases will require Java 11.

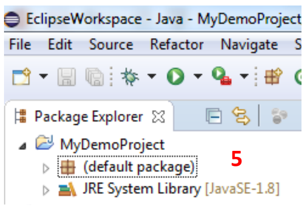
Eclipse: Developing Java Programs

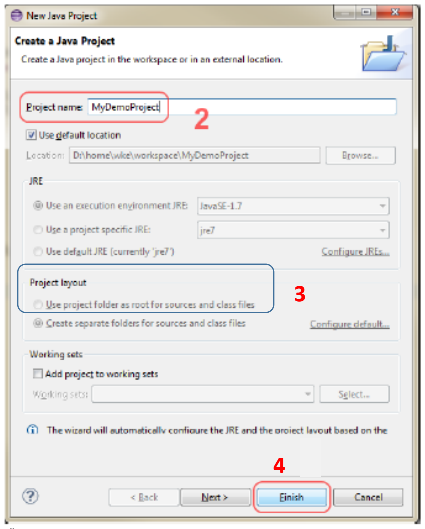
To start with Eclipse, create a project to hold all the files with the steps below:

1. Choose *File→ New → Java Project* to display the New Project wizard.
2. Type **MyDemoProject** in the Project name field. As you type, the Location field is automatically set by default. You may customize the location for your project.
3. Configure Project Layout:
   - By default, Eclipse stores certain project files in the project folder, your Java source code in the project's src sub-folder, and the class files produced by the Java compiler in the project's bin sub-folder. This organization benefits large complex projects but complicates smaller projects.
   - Eclipse can be configured to, by default, create a *new* project with all its files in the project folder. Select the option *Use project folder as root for sources and class files* so that the .java and .class files are in the same folder for easy access.
4. Click *Finish* to create the project.

Eclipse: Developing Java Programs




EclipseWorkspace - Java - MyDemoProject
File Edit Source Refactor Navigate
Package Explorer
MyDemoProject
(default package)
JRE System Library [JavaSE-1.8]
5



Project name: MyDemoProject

[✓] Use default location

Location: D:\home\wke\workspace\MyDemoProject

JRE

(●) Use an execution environment JRE: JavaSE-1.7

( ) Use a project specific JRE: jre7

( ) Use default JRE (currently 'jre7') Configure JREs...

Project layout

( ) Use project folder as root for sources and class files

(●) Create separate folders for sources and class files Configure default...

Working sets

[ ] Add project to working sets

Working sets: Select...

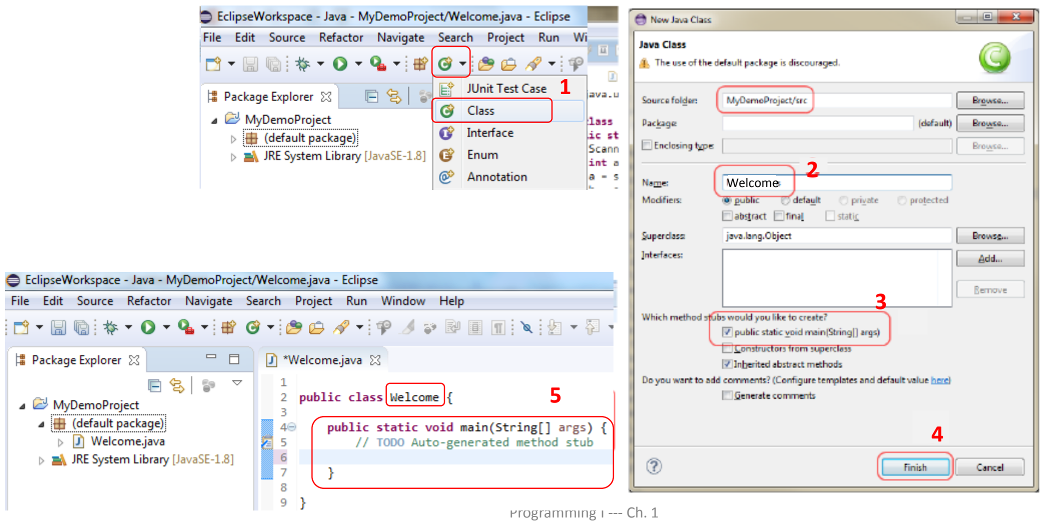
The wizard will automatically configure the JRE and the project layout based on the

Eclipse: Creating a Java Class

After a project is created, you can create Java programs in the project using the following steps:

1. Choose *File→ New → Class* to display the New Java Class wizard.
2. Type **Welcome** in the Name field.
3. Check the option *public static void main(String[] args)*.
4. Click *Finish* to generate the template for the source code Welcome.java.
5. Now, you can import necessary classes before the main class, and write statements in the main method.

Eclipse: Creating a Java Class


EclipseWorkspace - Java - MyDemoProject/Welcome.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Wi
Package Explorer
MyDemoProject
(default package)
JRE System Library [JavaSE-1.8]
JUnit Test Case
1
Class
Interface
Enum
Annotation
New Java Class
Java Class
2
3
4
Browse...
Add...
Remove
Finish
Cancel
EclipseWorkspace - Java - MyDemoProject/Welcome.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
*Welcome.java
MyDemoProject
(default package)
Welcome.java
JRE System Library [JavaSE-1.8]
5

The use of the default package is discouraged.

Source folder: MyDemoProject/src

Package: (default)

[ ] Enclosing type

Name: Welcome

Modifiers: [✓ public] default private protected

[ ] abstract [ ] final [ ] static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

[✓] public static void main(String[] args)

[ ] Constructors from superclass

[✓] Inherited abstract methods

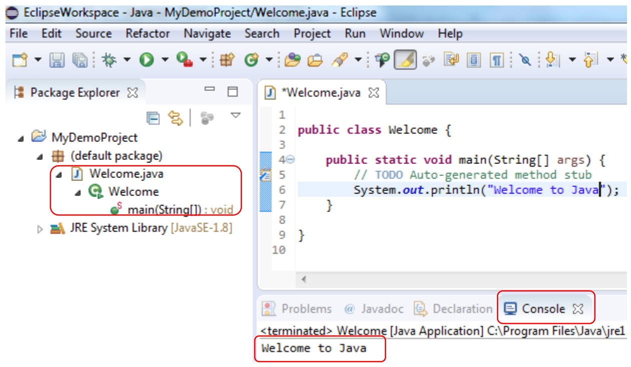
Do you want to add comments? (Configure templates and default value here)

[ ] Generate comments

```

public class Welcome {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

    }

}
```

Eclipse: Compiling and Running a Class

- When you save your program source file, the program is compiled automatically by Eclipse, and the binary ".class" file is created.
- To run the program, right-click the class in the project to display a context menu. Choose *Run, Java Application* in the context menu to run the class. The output is displayed in the Console pane.


EclipseWorkspace - Java - MyDemoProject/Welcome.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
MyDemoProject
(default package)
Welcome.java
Welcome
main(String[]) : void
JRE System Library [JavaSE-1.8]
*Welcome.java
Problems
Javadoc
Declaration
Console

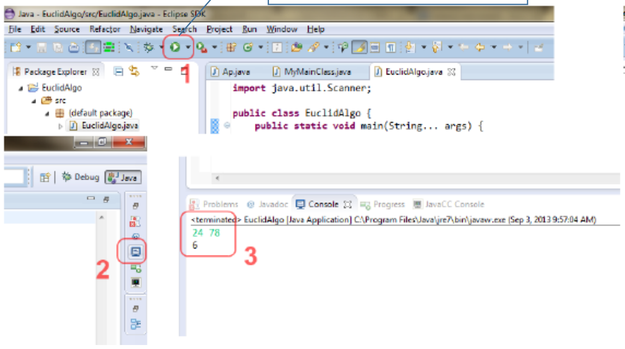
```
public class Welcome {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        System.out.println("Welcome to Java");
    }

}
```

```
<terminated> Welcome [Java Application] C:\Program Files\Java\jre1
Welcome to Java
```

- Eclipse allows you to keep more than one programs in a project.
- To run a particular program, open and right-click on the source file ⇒ Run As ⇒ Java Application.
- Clicking the "Run" button (with a "Play" icon) runs the recently-run program (based on the previous configuration).

Eclipse: Error mark

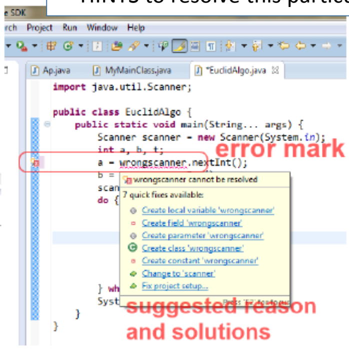
Button to run a program



```
import java.util.Scanner;

public class EuclidAlgo {
    public static void main(String... args) {
```

```
<terminated> EuclidAlgo [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Sep 3, 2013 9:57:04 AM)
24 78
6
```

- ORANGE LIGHT-BULB (for HINTS) next to the ERROR RED-CROSS to get a list of HINTS to resolve this particular error



```
import java.util.Scanner;

public class EuclidAlgo {
    public static void main(String... args) {
        Scanner scanner = new Scanner(System.in);
        int a, b, t;
        a = wrongscanner.nextInt();
```

wrongscanner cannot be resolved
7 quick fixes available:
- Create local variable 'wrongscanner'
- Create field 'wrongscanner'
- Create parameter 'wrongscanner'
- Create class 'wrongscanner'
- Create constant 'wrongscanner'
- Change to 'scanner'
- Fix project setup...

- SYNTAX WARNING: marked by a orange triangular exclamation sign. Unlike errors, warnings may or may not cause problems. You can RUN your program with warnings.

Eclipse Workspace

- A Workspace is a folder on your hard-drive where Eclipse stores your java projects.
- Java code is organized in projects in Eclipse.
- An eclipse workspace can contain any number of projects.

Eclipse Workspace: Closing projects

- A project can be either in the open state or closed state.
- You will see your project in Package Explorer on the left of your screen.
- If a project is not under active development, it can be closed, but they still reside on the local file system.
- To close a project, from the Project select the Close Project menu item.
- A closed project is visible in the Package Explorer view but its contents cannot be edited using the Eclipse user interface.
- To reopen a closed project, in the Package Explorer view, select the closed project and click on the Project menu and select Open Project.

Eclipse Workspace: perspectives

- An eclipse window can have multiple perspectives open in it but only one perspective can be active at any point of time.
- A perspective controls what appears in some menus and tool bars.
- The default perspective is called java. Another useful perspective is the debug perspective. (Windows→ Perspective → Open Perspective).
- To stop running an application:
  - Open the Debug perspective
  - Select process in Devices list (bottom right)
  - Hit **Stop** button (top right of Devices pane)

Eclipse Workspace: deleting projects

There are two options to delete a project via Edit → Delete:

- To delete a project and remove its contents from the file system
- To delete a project from the workspace without removing its contents from the file system

Eclipse: Export function

Saving (exporting) a project in a different location for later use.

- If you work in the lab and you want to save your work to a memory stick.
- To export an entire project right click on the project and select Export… → General → File System and then press the **Next >** button to browse to the location you want to save the project to.

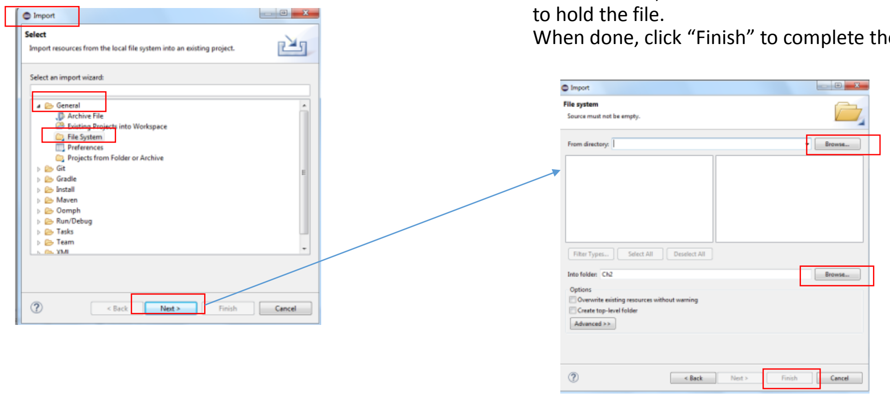
Eclipse: open an existing project

- In Eclipse, to open an existing project which is copied from another source, there are two options:
  - Use File→ Open Projects From File System
    This works for Eclipse projects with .project descriptor
  - Use its **Import** function
    File→ Import… → General → Existing projects into Workspace, then select the radio option: *Select root directory* and specify the project's directory path.
    - Import is for projects that you have created either using Eclipse or other compatible IDEs, and that have a project file associated to them.
    - You can use import to copy an existing project to a new project folder inside the workspace directory.
  - It is common mistake to choose File → Open File… ✘✘
- **Adding Existing Classes to A Project:**
  File→ Import… → General → File System

Eclipse: **Adding Existing Classes to A Project**

- File→ Import… → General → File System → Next

- In the following dialog box, specify where to get the source file, and confirm the destination folder to hold the file.
  When done, click "Finish" to complete the import.



Import resources from the local file system into an existing project.

Select an import wizard:

Source must not be empty.

Into folder: Ch2

Chapter Summary

- Computer *programming* is the writing of instructions (i.e., code) for computers to perform.
- *High-level languages* are English-like and easy to learn and program. A program written in a high-level language is called a *source program.*
- A *compiler* is a software program that translates the source program into a *machine language program.*
- Java is platform independent, meaning that you can write a program once and run it on any computer.
- The Java source file name must match the public class name in the program. Java source code files must end with the **.java** extension.

Chapter Summary

- The **main** method is the entry point where the program starts when it is executed.
- 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.
- Java source programs are case sensitive.
- Programming errors can be categorized into three types: *syntax errors*, *runtime errors*, and *logic errors.*

Organization of the Book

- **Part I: Fundamentals of Programming (Chapters 1–8)**
- **Part II: Object-Oriented Programming (Chapters 9–13, and 17)**
- **Part III: GUI Programming (Chapters 14–16 and Bonus Chapter 31)**
- **Part IV: Data Structures and Algorithms (Chapters 18–30 and Bonus Chapters 42–43)**
- **Part V: Advanced Java Programming (Chapters 32-41, 44)**

Exercises

1. What is the Java source filename extension, and what is the Java bytecode filename extension?
2. What does IDE stand for? What does JDK stand for?
3. What is a keyword? List some Java keywords.
4. Is Java case sensitive?
5. What is a comment? Is the comment ignored by the compiler? How do you denote a comment line and a comment paragraph?
6. What is the statement to display a string on the console?

Exercises (cont'd)

7. If you forget to put a closing quotation mark on a string, what kind of error will be raised (syntax error, runtime error, or logic error)?
8. Suppose you write a program for computing the perimeter of a rectangle and you mistakenly write your program so that it computes the area of a rectangle. What kind of error is this (syntax error, runtime error, or logic error)? (5 marks)
9. Suppose you define a Java class as follows, the source code should be stored in a file named ___________________. (5 marks)

```
public class Test {

}
```

Exercises (cont’d)

10. Identify and fix the errors in the following code:

```
public class Welcome {
  public void Main(String[] args) {
    System.out.println('Welcome to Java!);
  }
)
```