

Data Definition and Classes

Dr. Youna Jung

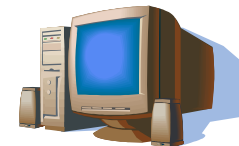
Northeastern University

yo.jung@northeastern.edu



Programs

- ❑ Computer **programs**, known as **software**, are **instructions that tell a computer what to do**.
- ❑ Computers do not understand human languages, so you need to use **computer languages** to communicate with them.
- ❑ Types of programming languages
 - ✓ **Machine language** (Low-level)
 - ✓ **Assembly language**
 - ✓ **High-level language**



1. Machine language

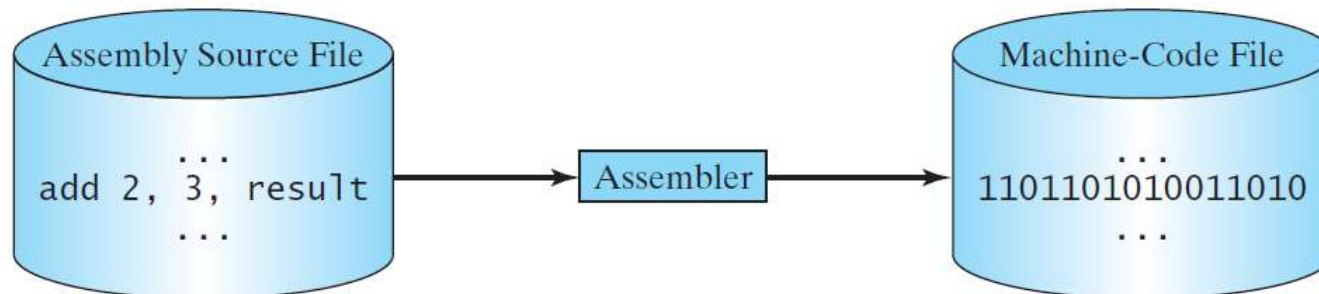
- ❑ The instructions are in the form of **binary code**.
 - ✓ Program with native machine language is a tedious process.
 - ✓ e.g) to **add two numbers 2 and 3**, you might write an instruction in binary like this:

```
1101101010011010
```

2. Assembly languages

- ❑ An alternative to machine languages
 - ✓ developed to **make programming easy**.
 - ✓ uses a **short descriptive word** to represent each of the machine-language **instructions**
 - A program called **assembler** is used to convert **assembly language programs** into **machine code**.
 - e.g) to **add two numbers 2 and 3**, you might write an instruction in **assembly code** like this:

```
ADDF3 R1, R2, R3
```



3. High-Level Languages

- ❑ **English-like** and **easy to learn** and program
 - ✓ e.g) The following is a high-level language statement that **computes the area of a circle with radius 5**:

```
area = 5 * 5 * 3.1415;
```

- ❑ **Platform-independent**
 - ✓ You can write a program in a high-level language and run it in different types of machines.

Popular High-Level 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.

Why Java?

- ❑ Java is a **general purpose programming language**.
- ❑ Java is the **Internet programming language**.
 - ✓ The future of computing is being profoundly influenced by the Internet, and Java promises to remain a big part of that future.
- ❑ Java enables users to develop and deploy applications on the Internet for servers, desktop computers, and small hand-held devices.

Characteristics of Java

Companion
Website

- ☐ Java Is Simple
- ☐ Java Is Object-Oriented
- ☐ Java Is Distributed
- ☐ Java Is Interpreted
- ☐ Java Is Robust
- ☐ Java Is Secure
- ☐ Java Is Architecture-Neutral
- ☐ Java Is Portable
- ☐ Java's Performance
- ☐ Java Is Multithreaded
- ☐ Java Is Dynamic

www.cs.armstrong.edu/liang/JavaCharacteristics.pdf

Characteristics of Java

- ❑ Java Is Simple
- ❑ **Java Is Object-Oriented**
- ❑ Java Is Distributed
- ❑ Java Is Interpreted
- ❑ Java Is Robust
- ❑ Java Is Secure
- ❑ Java Is Architecture-Neutral
- ❑ Java Is Portable
- ❑ Java's Performance
- ❑ Java Is Multithreaded
- ❑ Java Is Dynamic

Java is inherently object-oriented. Although many object-oriented languages began strictly as procedural languages, Java was designed from the start to be object-oriented. Object-oriented programming (OOP) is a popular programming approach that is replacing traditional procedural programming techniques.

One of the central issues in software development is how to reuse code. Object-oriented programming provides great flexibility, modularity, clarity, and reusability through encapsulation, inheritance, and polymorphism.

Java's History

- ❑ **James Gosling** and **Sun Microsystems**
- ❑ Oak
- ❑ Java, May 20, 1995, Sun World
- ❑ HotJava
 - ✓ The first Java-enabled Web browser
- ❑ Early History Website:

<http://www.java.com/en/javahistory/index.jsp>

API, JDK, and IDE

❑ Application Program Interface (API)

- ✓ Known as **library**, contains **predefined classes and interfaces** for developing Java programs
- ✓ <https://docs.oracle.com/javase/10/docs/api/overview-summary.html>

❑ Java Development Toolkit (JDK)

- ✓ JDK 1.02 (1995)
- ✓ JDK 1.1 (1996)
- ✓
- ✓ JDK 10 or Java 10

❑ Integrated Development Environment (IDE)

- ✓ e.g.) **NetBeans**, **Eclipse**, and TextPad

JDK Editions

❑ Java Standard Edition (J2SE)

- ✓ J2SE can be used to develop client-side standalone applications or applets.

❑ Java Enterprise Edition (J2EE)

- ✓ J2EE can be used to develop server-side applications such as Java servlets, Java ServerPages, and Java ServerFaces.

❑ Java Micro Edition (J2ME).

- ✓ J2ME can be used to develop applications for mobile devices such as cell phones.

❑ This book uses J2SE to introduce Java programming.

Supplements on the Companion Website

- ❑ Installing and configuring **JDK**
 - ✓ See **Supplement I.B**
- ❑ Compiling and running Java from the command window for details
 - ✓ See Supplement I.C
- ❑ www.cs.armstrong.edu/liang/intro10e

Get Ready?

☐ IDE Installation

- ✓ NetBeans
- ✓ Eclipse

☐ Creating a Package

- ✓ Check VideoNotes

Anatomy of a Java Program

- ❑ Class name
- ❑ Main method
- ❑ Statements
- ❑ Statement terminator
- ❑ Reserved words
- ❑ Comments
- ❑ Blocks

Class Name

- ❑ Every Java program must have at least one class.
 - ✓ Each class has a name.
 - By convention, class names start with an uppercase letter.
 - Ex) The class name is Welcome.

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


Main() Method

- ❑ In order to run a class, the class must contain a method named `main`.
 - ✓ The program is executed from the `main` method.

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

Statement

❑ A statement represents an **action** or a **sequence of actions**.

✓ **Ex)** `System.out.println ("Welcome to Java!")` is a statement to display the greeting “Welcome to Java!”.

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

Statement Terminator

- ❑ Every statement in Java ends with a semicolon (;).

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

Reserved words (Keywords)

- ❑ Words that have a specific meaning to the compiler so cannot be used for other purposes in the program. (See [Appendix A. Java Keywords](#))
 - ✓ Ex) `class`, `public`, `static`, `void`, `final`, **etc.**

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

Blocks

- ❑ A pair of braces in a program forms a block that groups components of a program.

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

Class block

Method block

Special Symbols

Character Name		Description
{ }	Opening and closing braces	Denotes a block to enclose statements.
()	Opening and closing parentheses	Used with methods.
[]	Opening and closing brackets	Denotes an array.
//	Double slashes	Precedes a comment line.
" "	Opening and closing quotation marks	Enclosing a string
;	Semicolon	Marks the end of a statement.

{ ... }

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

(...)

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


▪
;

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

// ...

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

""
...

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

A Simple Java Program

Welcome.java

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



[Welcome](#)

Trace a Program Execution

Enter main method

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

Trace a Program Execution

Execute statement

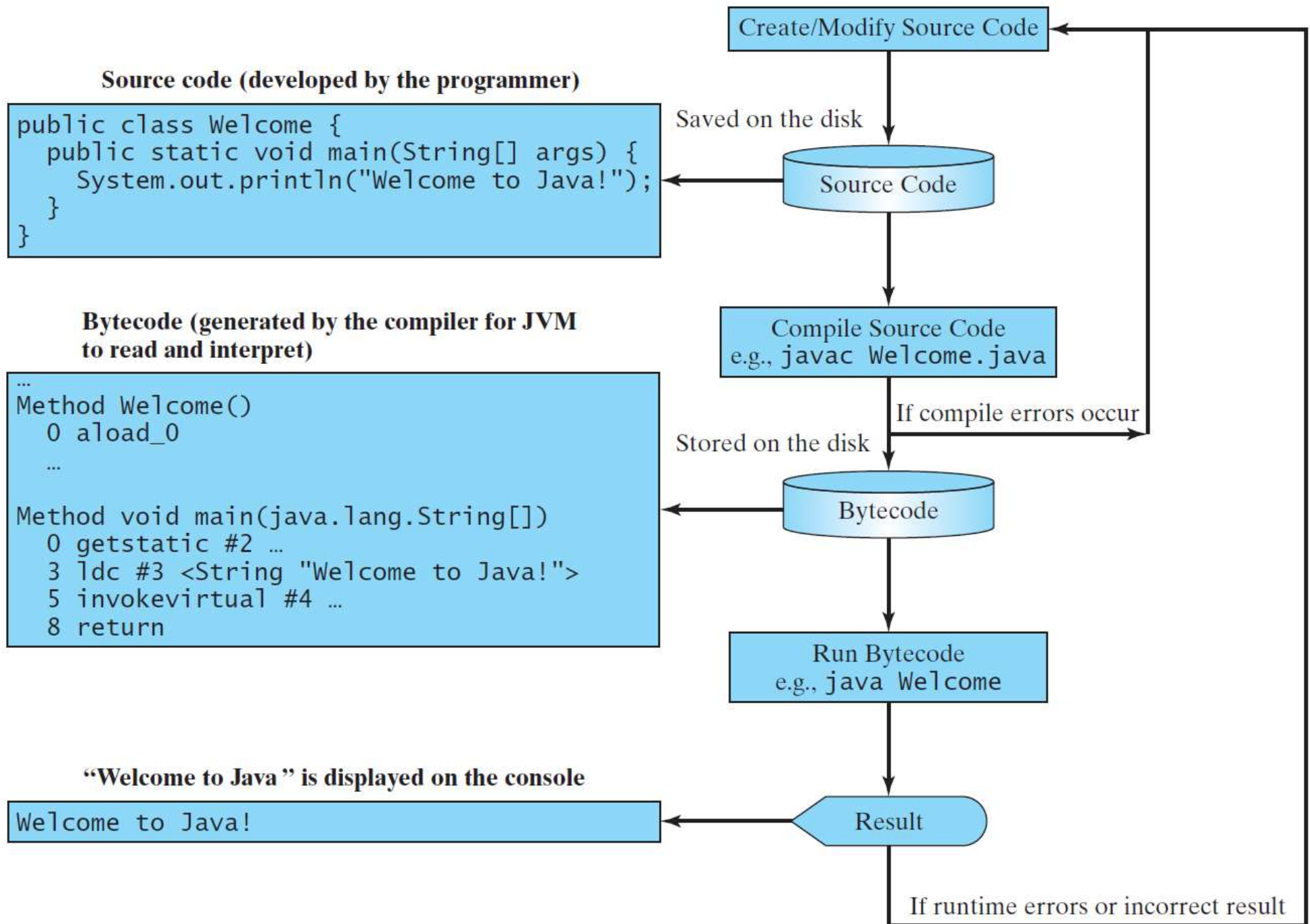
```
// This program prints Welcome to Java!  
public class Welcome {  
    public static void main(String[] args) {  
        System.out.println("Welcome to Java!");  
    }  
}
```

Trace a Program Execution

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

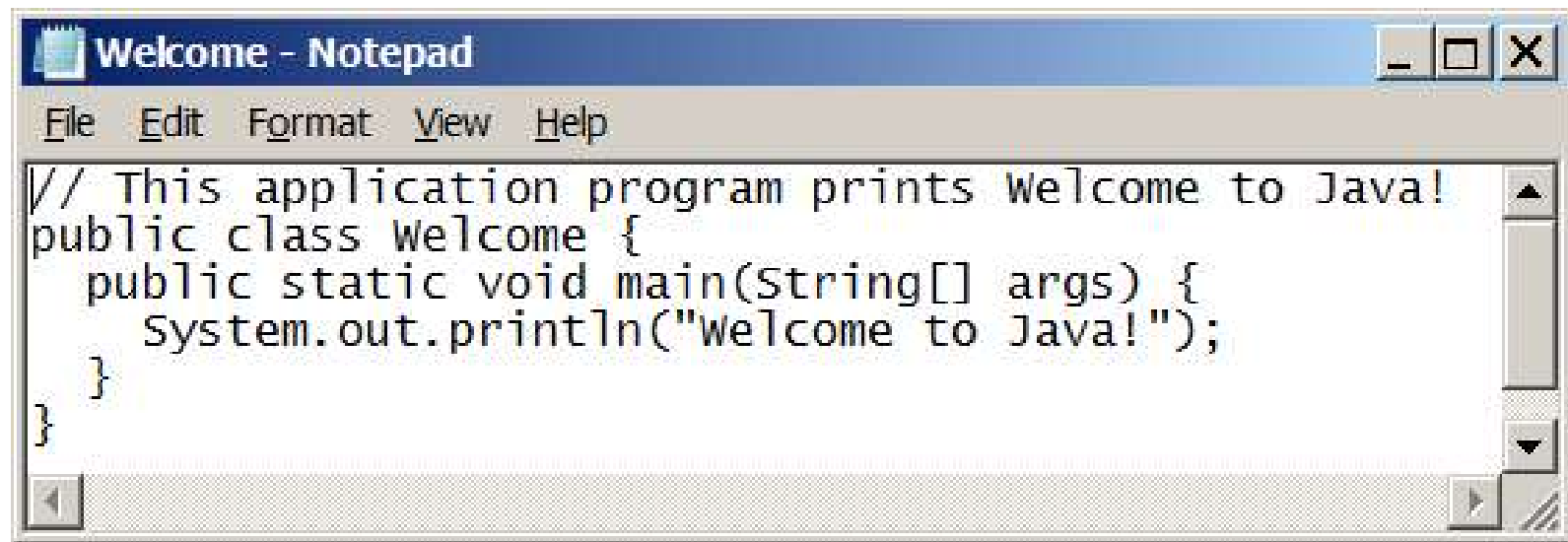
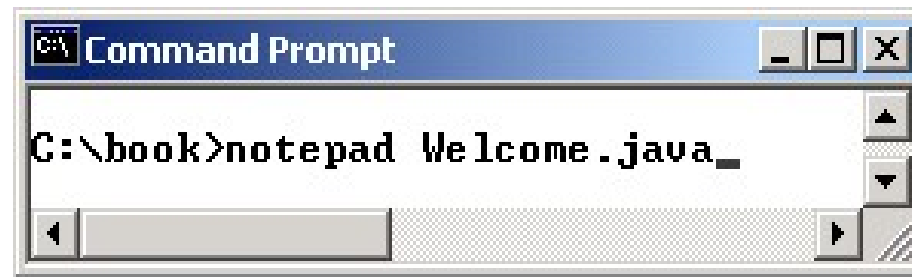


print a message to the
console



Creating and Editing using Notepad

- ❑ To use **NotePad**, type *notepad Welcome.java* from the DOS prompt.



WelcomeWithThreeMessages.java

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



[WelcomeWithThreeMessages](#)

Run

Animation

ComputeExpression.java

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

0.39759036144578314



[ComputeExpression](#)

Run

Animation

Programming Style and Documentation

- ❑ Appropriate Comments
- ❑ Naming Conventions
- ❑ Proper Indentation and Spacing Lines
- ❑ Block Styles

Appropriate Comments

- ❑ Include a summary at the beginning of the program
 - ✓ to explain what the program does, its key features, its supporting data structures, and any unique techniques it uses.
 - ✓ include your name, class section, instructor, date, and a brief description.

Naming Conventions

- ❑ Choose **meaningful** and **descriptive** names.
- ❑ Class names:
 - ✓ **Capitalize the first letter of each word in the name.**
 - e.g.) The class name `ComputeExpression`.

Proper Indentation and Spacing

□ Indentation

- ✓ For better readability
- ✓ Indent two spaces at least, or just use a tab.

□ Spacing

- ✓ Use blank line to separate segments of the code.

Block Styles

❑ **Next-line style** vs. **End-of-line style** for braces.

*Next-line
style*

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

*End-of-line
style*

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


Programming Errors

❑ Syntax Errors

- ✓ Detected by the **compiler**

❑ Runtime Errors

- ✓ Causes the program to **abort**

❑ Logic Errors

- ✓ Produces **incorrect result**

1) Syntax Errors

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

** Syntax Error*




ShowSyntaxErrors

Run

2) Runtime Errors

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

 *Runtime Error- Division by zero*



ShowRuntimeErrors

Run

3) Logic Errors

```
public class ShowLogicErrors {  
    public static void main(String[] args) {  
        System.out.println("Celsius 35 is Fahrenheit degree ");  
        System.out.println((9 / 5) * 35 + 32);  
        System.out.println((9.0 / 5) * 35 + 32);  
    }  
}
```

❌ *Logic Error- Fahrenheit 67 degree is wrong. (should be 95)*
In Java, the division for integers is the quotient (the fractional part is truncated (so $9/5 \rightarrow 1$, not 1.8))



[ShowLogicErrors](#)

Run

Writing a simple program

- ❑ Let's consider the simple problem of **computing the area of a circle**.
 1. **Designing algorithms**
 - ✓ *Algorithm* – How a problem is solved by listing the actions that need to be taken and the order of their execution.
 2. **Translating algorithms into programming code**
 - ✓ *Pseudocode* – Algorithms can be described in natural languages mixed with some programming code

Writing a simple program

Algorithm

1. Read in the circle's radius
2. Compute the area using the following formula:

$$area = radius \times radius \times \pi$$

3. Display the result

Pseudocode

```
Public class ComputeArea {  
    public static void main(String[] args) {  
        //Step1: Read in radius  
        //Step2: compute area  
        //Step3: Display the area  
    }  
}
```

Step 1: Read in radius

❑ Needs to **read the radius** entered by the user

1) **Reading** the radius

- **Assign a fixed value** to radius OR
- **Prompt the user to designate** the circle's radius

2) **Storing** the radius in the program

- A **variable** represents **a value stored in the computer's memory**
 - Every variable has a **name**, a **type**, a **size**, and a **value**
 - Use descriptive names for variables
 - ✓ e.g.) **radius** for radius and **area** for area
 - Specify data types of variables
 - ✓ **Data type** – the kind of data stored in a variable
 - ✓ Java provides simple data types for representing integers, real numbers, characters, and Boolean types (**Primitive data types**)

Numerical Data Types

☐ Java has 6 numeric types

Name	Range	Storage Size
<code>byte</code>	-2^7 to $2^7 - 1$ (-128 to 127)	8-bit signed
<code>short</code>	-2^{15} to $2^{15} - 1$ (-32768 to 32767)	16-bit signed
<code>int</code>	-2^{31} to $2^{31} - 1$ (-2147483648 to 2147483647)	32-bit signed
<code>long</code>	-2^{63} to $2^{63} - 1$ (i.e., -9223372036854775808 to 9223372036854775807)	64-bit signed
<code>float</code>	Negative range: -3.4028235E+38 to -1.4E-45 Positive range: 1.4E-45 to 3.4028235E+38	32-bit IEEE 754
<code>double</code>	Negative range: -1.7976931348623157E+308 to -4.9E-324 Positive range: 4.9E-324 to 1.7976931348623157E+308	64-bit IEEE 754

Step 1: Read in radius

```
Public class ComputeArea {  
    public static void main(String[] args) {  
        double radius; //Declare radius  
        double area;    //Declare area  
  
        radius = 20;    //Assign a radius. Now radius is 20.  
  
        //Step2: compute area  
        //Step3: Display the area  
    }  
}
```

Step 2: Compute Area

```
Public class ComputeArea {  
    public static void main(String[] args) {  
        double radius; //Declare radius  
        double area;    //Declare area  
  
        radius = 20;    //Assign a radius. Now radius is 20.  
  
        //Compute area  
        area = radius * radius * 3.14159  
  
        //Step3: Display the area  
    }  
}
```

Step 3: Display the area

```
Public class ComputeArea {  
    public static void main(String[] args) {  
        double radius; //Declare radius  
        double area;   //Declare area  
  
        radius = 20;    //Assign a radius. Now radius is 20.  
  
        //Compute area  
        area = radius * radius * 3.14159  
  
        //Display the area  
        System.out.println("The area for the circle of radius " +  
            radius + " is " + area);  
    }  
}
```

+ has 2 meaning:
1) Addition
2) Concatenation

!! Caution !!

A string cannot cross lines in the source code.

JavaLiveExample: ComputeArea.java

```
1 public class ComputeArea {  
2     public static void main(String[] args) {  
3         double radius; // Declare radius  
4         double area; // Declare area  
5  
6         // Assign a radius  
7         radius = 20; // New value is radius  
8  
9         // Compute area  
10        area = radius * radius * 3.14159;  
11  
12        // Display results  
13        System.out.println("The area for the circle of radius " +  
14        radius + " is " + area);  
15    }  
16 }  
17
```

Compile/Run

Reset

You can modify the code and run it again

```
c:\book>javac ComputeArea.java
```

```
Compiled successful
```

```
c:\book>java ComputeArea
```

```
The area for the circle of radius 20.0 is 1256.636
```

```
c:\book>
```



[ComputeArea](#)

Animation

Run

Trace a Program Execution

```
public class ComputeArea {  
    /** Main method */  
    public static void main(String[] args) {  
        double radius;  
        double area;  
  
        // Assign a radius  
        radius = 20;  
  
        // Compute area  
        area = radius * radius * 3.14159;  
  
        // Display results  
        System.out.println("The area for the  
            circle of radius " + radius + " is "  
            + area);  
    }  
}
```

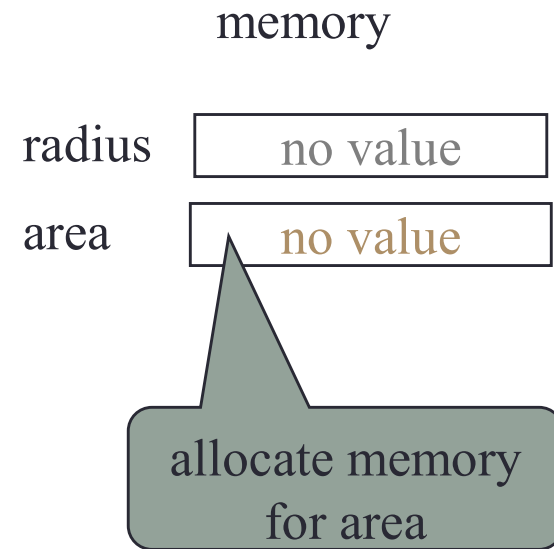
radius

allocate memory
for radius

no value

Trace a Program Execution

```
public class ComputeArea {  
    /** Main method */  
    public static void main(String[] args) {  
        double radius;  
        double area;  
  
        // Assign a radius  
        radius = 20;  
  
        // Compute area  
        area = radius * radius * 3.14159;  
  
        // Display results  
        System.out.println("The area for the  
            circle of radius " + radius + " is "  
            + area);  
    }  
}
```



Trace a Program Execution

```
public class ComputeArea {  
    /** Main method */  
    public static void main(String[] args) {  
        double radius;  
        double area;  
  
        // Assign a radius  
        radius = 20;  
  
        // Compute area  
        area = radius * radius * 3.14159;  
  
        // Display results  
        System.out.println("The area for the  
            circle of radius " + radius + " is "  
            + area);  
    }  
}
```

assign 20 to radius

radius

20.0

area

no value

Trace a Program Execution

```
public class ComputeArea {  
    /** Main method */  
    public static void main(String[] args) {  
        double radius;  
        double area;  
  
        // Assign a radius  
        radius = 20;  
  
        // Compute area  
        area = radius * radius * 3.14159;  
  
        // Display results  
        System.out.println("The area for the  
            circle of radius " + radius + " is "  
            + area);  
    }  
}
```

memory

radius

20.0

area

1256.636

compute area and assign it
to variable area

Trace a Program Execution

```
public class ComputeArea {  
    /** Main method */  
    public static void main(String[] args) {  
        double radius;  
        double area;  
  
        // Assign a radius  
        radius = 20;  
  
        // Compute area  
        area = radius * radius * 3.14159;  
  
        // Display results  
        System.out.println("The area for the  
            circle of radius " + radius + " is "  
            + area);  
    }  
}
```

memory

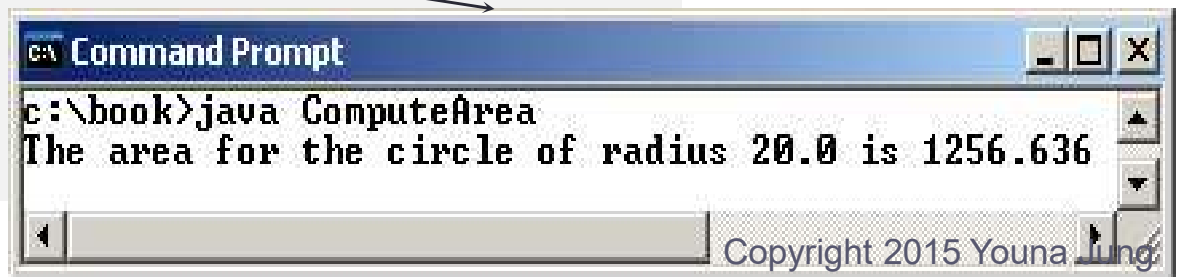
radius

20

area

1256.636

print a message to the
console



The screenshot shows a Command Prompt window with the title bar 'Command Prompt'. The command prompt shows the command 'c:\book>java ComputeArea' and the output 'The area for the circle of radius 20.0 is 1256.636'. A copyright notice 'Copyright 2015 Youna Jung' is visible at the bottom right of the window.

```
Command Prompt  
c:\book>java ComputeArea  
The area for the circle of radius 20.0 is 1256.636  
Copyright 2015 Youna Jung
```

Reading Input from the Console

- ❑ Enable the program to accept input from the user → use the **Scanner** class

1. Impart `java.util.Scanner` package

```
import java.util.Scanner;
```

2. Create a Scanner object

```
Scanner input = new Scanner(System.in);
```

- ✓ `System.in` – The standard input device (Keyboard)
- ✓ `System.out` – The standard output device (Monitor)

3. Use the method `nextDouble()` to obtain to a double value.

```
System.out.print("Enter a double value: ");  
Scanner input = new Scanner(System.in);  
double d = input.nextDouble();
```

Reading Numbers from the Keyboard

```
Scanner input = new Scanner(System.in);  
int value = input.nextInt();
```

Method	Description
<code>nextByte()</code>	reads an integer of the <code>byte</code> type.
<code>nextShort()</code>	reads an integer of the <code>short</code> type.
<code>nextInt()</code>	reads an integer of the <code>int</code> type.
<code>nextLong()</code>	reads an integer of the <code>long</code> type.
<code>nextFloat()</code>	reads a number of the <code>float</code> type.
<code>nextDouble()</code>	reads a number of the <code>double</code> type.

ComputeAreaWithConsoleInput.java

```
import java.util.Scanner; // Scanner is in the java.util package

public class ComputeAreaWithConsoleInput {
    public static void main(String[] args) {
        // Create a Scanner object
        Scanner input = new Scanner(System.in);

        // Prompt the user to enter a radius
        System.out.print("Enter a number for radius: ");
        double radius = input.nextDouble();

        // Compute area
        double area = radius * radius * 3.14159;

        // Display result
        System.out.println("The area for the circle of radius " +
            radius + " is " + area);
    }
}
```



[ComputeAreaWithConsoleInput](#)

prompt

Enter a number for radius: 3.85

The area for the circle of radius 3.85 is 46.566217775000005

ComputeAverage.java

```
import java.util.Scanner; // Scanner is in the java.util package

public class ComputeAverage {
    public static void main(String[] args) {
        // Create a Scanner object
        Scanner input = new Scanner(System.in);

        // Prompt the user to enter three numbers
        System.out.print("Enter three numbers: ");
        double number1 = input.nextDouble();
        double number2 = input.nextDouble();
        double number3 = input.nextDouble();

        // Compute average
        double average = (number1 + number2 + number3) / 3;

        // Display result
        System.out.println("The average of " + number1 + " " + number2
            + " " + number3 + " is " + average);
    }
}
```

Enter three numbers: 1.5 2.3 7.1

The average of 1.5 2.3 7.1 is 3.6333333333333333



[ComputeAverage](#)

Identifiers

❑ The names that identify the elements

- ✓ e.g.) classes, methods, and variables.

❑ All identifiers must obey the following rules

- ✓ An identifier is a sequence of characters that consist of 1) letters, 2) digits, 3) underscores (_), and 4) dollar signs (\$).
- ✓ An identifier must start with a letter, an underscore (_), or a dollar sign (\$). It cannot start with a digit.
- ✓ An identifier cannot be a reserved word.
 - See Appendix A, “Java Keywords,” for a list of reserved words.
- ✓ An identifier cannot be true, false, or null.
- ✓ An identifier can be of any length.
- ✓ Java is case sensitive.

Q) Which identifiers' names are valid?

: \$2, ComputeArea, area, radius, print, ~~2A~~, ~~d+4~~

Variables

- ❑ Represent **values that may be changed** in the program
- ❑ To use a variable

1) Declaration

- Tells the compiler to **allocate appropriate memory** space for the variable based on its data type

```
Datatype variableName;
```

➤e.g.) `int count, i, k;`
`double radius;`

Assignment operator

You can assign a value to a variable by using an assignment statement

2) Assignment/Initialization

- We can assign an actual value to a variable.

➤e.g.) `int count = 1;`

Declaring Variables

```
int x;           // Declare x to be an integer variable;  
double radius; // Declare radius to be a double variable;  
char a;          // Declare a to be a character variable;
```

Assignment Statements

```
x = 1;           // Assign 1 to x;  
radius = 1.0;   // Assign 1.0 to radius;  
a = 'A';        // Assign 'A' to a;
```

Declaring and Initializing in 1 Step

```
int x = 1;  
double d = 1.4;
```


Assignment

- ❑ After a variable is declared, you can assign a value to it by using an assignment statement

```
Variable = expression;
```

- ✓ *Assignment operator (=)*
 - In mathematics, $x = 2 * x + 1$ denotes an equation
 - However, in java, $x = 2 * x + 1$ is an assignment statement that evaluates the expression $2 * x + 1$ and assigns the result to x .
- ✓ *Expression* represents a computation involving **values**, **variables**, and **operators** that taking them together, evaluates to a value.

Assignment

- ❑ You must place the **variable** name to the **left** of the assignment operator.

✓ e.g) `1 = x;` **Error!**

- ❑ The **data type** of the variable on the left must be **compatible** with the data type of the **value** on the right

✓ e.g) `int x = 1.0;` **Error!**

Named Constant (Constant)

- ❑ An **identifier** that represents a **permanent value**

```
Final datatype CONSTANTNAME = value;
```

- ✓ A constant **must be declared and initialized in the SAME statement.**

- ❑ Benefits of using constants

- 1) You **don't have to repeatedly type the same value**
- 2) If you have to change the constant value, you need to change it only in a single location in the code
- 3) A descriptive name for a constant makes the program easy to read

Named Constant (Constant)

```
import java.util.Scanner;
public class ComputeAreaWithConstant{
    public static void main(String[] args) {

        final double PI = 3.14159; //declare a constant
        Scanner input = new Scanner(System.in);

        System.out.print("Enter a number for radius: ");
        double radius = input.nextDouble();
        double area = radius * radius * PI;

        System.out.println("The area for the circle of radius " +
            radius + " is " + area);
    }
}
```

Naming Conventions

- ❑ Choose meaningful and descriptive names.
- ❑ **Variables** and **method** names:
 - ✓ Use lowercase.
 - ✓ If the name consists of several words, concatenate all in one, use **lowercase for the first word**, and **capitalize the first letter of each subsequent word** in the name.
 - E.g) **r**adius, **a**rea, and **c**ompute**A**rea.

Naming Conventions, cont.

❑ Class names

- ✓ Capitalize the first letter of each word in the name.
 - e.g) `ComputeArea` class

❑ Constants

- ✓ Capitalize all letters in constants, and use underscores to connect words.
 - e.g) `PI` and `MAX_VALUE`

Numeric Operators

❑ Java has 5 standard arithmetic operators

Name	Meaning	Example	Result
+	Addition	34 + 1	35
-	Subtraction	34.0 - 0.1	33.9
*	Multiplication	300 * 30	9000
/	Division	1.0 / 2.0	0.5
%	Remainder	20 % 3	2

Division

- ❑ Both operands of a division are integers, the result is the quotient and the fractional part is truncated

- ✓ e.g) $5/2$ yields an integer 2 (Not 2.5)

- ❑ To perform a float-point division, one of the operands must be a floating-point number.

- ✓ e.g) $5.0/2$ yields a double value 2.5

Remainder (Modulo)

❑ Modulo yields the remainder after division.

✓ e.g) $7 \% 3$ yields 1

✓ e.g) $3 \% 7$ yields 3

✓ e.g) $12 \% 4$ yields 0

❑ The remainder is **negative only if the dividend is negative**

✓ e.g) $-7 \% 3$ yields -1

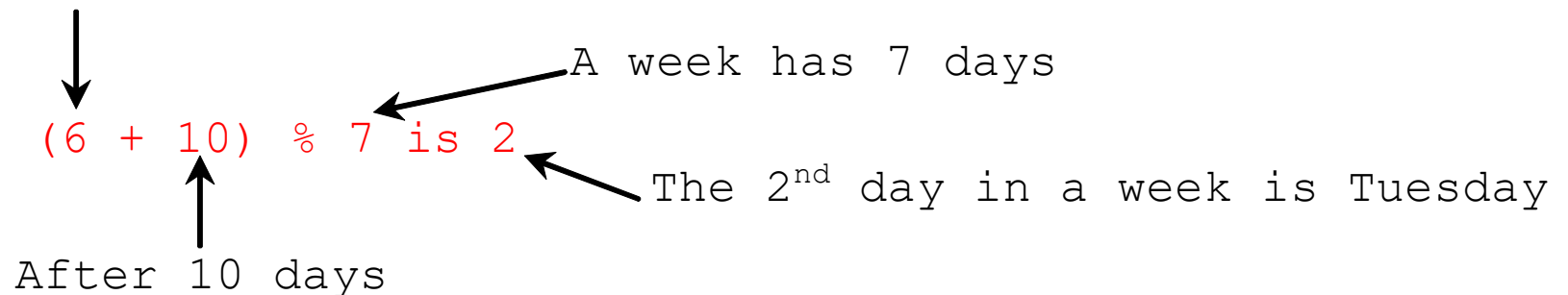
✓ e.g) $-12 \% 4$ yields 0

✓ e.g) $-26 \% -8$ yields -2

Remainder

- ❑ An **even number % 2** is always **0** and an **odd number % 2** is always **1**.
- ❑ Suppose today is **Saturday** and you and your friends are **going to meet in 10 days**. What day is in 10 days?
 - ✓ You can find that day is Tuesday using the following expression:

Saturday is the 6th day in a week



Problem: Displaying Time

- ❑ Write a program that obtains **minutes** and **remaining seconds** from **an amount of time in seconds**.
 - ✓ ex) User input: 500 seconds → 8 minutes and 20 seconds



DisplayTime

Run

DisplayTime.java

```
import java.util.Scanner;

public class DisplayTime {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        // Prompt the user for input
        System.out.print("Enter an integer for seconds: ");
        int seconds = input.nextInt();

        int minutes = seconds / 60; // Find minutes in seconds
        int remainingSeconds = seconds % 60; // Seconds remaining
        System.out.println(seconds + " seconds is " + minutes +
            " minutes and " + remainingSeconds + " seconds");
    }
}
```

```
c:\book>javac DisplayTime.java
Compiled successful
```

```
c:\book>java DisplayTime
Enter an integer for seconds: 500
500 seconds is 8 minutes and 20 seconds
```

Copyright 2015 Youna Jung

NOTE

- ❑ Calculations involving **floating-point numbers** are **approximated** because these numbers are not stored with complete accuracy.
 - ✓ e.x) `System.out.println(1.0 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1);`
 - ➔ displays 0.5000000000000000001, NOT 0.5
 - ✓ `System.out.println(1.0 - 0.9);`
 - ➔ displays 0.0999999999999999998, NOT 0.1.

Exponent Operations

- ❑ `Math.pow(a, b)` method can be used to compute a^b .
 - ✓ The `pow` method is defined in the `Math` class in the Java API.
 - ✓ ex) `System.out.println(Math.pow(2, 3));` → 8.0
 - ✓ ex) `System.out.println(Math.pow(4, 0.5));` → 2.0
 - ✓ ex) `System.out.println(Math.pow(2.5, 2));` → 6.25
 - ✓ ex) `System.out.println(Math.pow(2.5, -2));` → 0.16

Check point

- ❑ Write answers to the problems below and explanations.
 - ✓ 2.10
 - ✓ 2.11
 - ✓ 2.12
 - ✓ 2.14
 - ✓ 2.16
 - ✓ 2.17

Number Literals

- ❑ A **literal** is a constant value that appears directly in the program.
 - ✓ Ex) 34, 1,000,000, and 5.0 are literals in the following statements:

```
int i = 34;  
long x = 1000000;  
double d = 5.0;
```


Integer Literals

- ❑ An integer literal can be assigned to an integer variable as long as it can fit into the variable.
 - ✓ A compilation error would occur if the literal were too large for the variable to hold.
Error!
– Ex) `byte b = 1000;` *Because 1000 cannot be stored in a byte type variable.
byte (-128 to 127)*
- ❑ An integer literal is assumed to be of the `int` type
 - ✓ value is -2^{31} (-2147483648) ~ $2^{31}-1$ (2147483647).
 - ✓ To denote an integer literal of the `long` type, append it with the letter **L**(preferred) or **l**.

Floating-Point Literals

- ❑ Floating-point literals are written with a decimal point.
 - ✓ By default, a floating-point literal is treated as a `double` type.
 - Ex) `5.0` is considered a `double` value, not a `float` value.
 - ✓ You can make a number a `float` by appending the letter `f` or `F`
 - Ex) `100.2f` or `100.2F` for a `float` number
 - ✓ You can make a number a `double` by appending `d` or `D`.
 - Ex) `100.2d` or `100.2D` for a `double` number.

double vs. float

- ❑ The **double** type values are **more accurate** than the `float` type values.

```
System.out.println("1.0 / 3.0 is " + 1.0 / 3.0);
```

displays `1.0 / 3.0 is 0.3333333333333333`

16 digits

```
System.out.println("1.0F / 3.0F is " + 1.0F / 3.0F);
```

displays `1.0F / 3.0F is 0.33333334`

8 digits

Scientific Notation

- ❑ Floating-point literals can also be specified in scientific notation
- ❑ E (or e) represents an exponent and it can be either in lowercase or uppercase.
 - ✓ Ex) $1.23456e+2$, same as $1.23456e2 == 123.456$
 - ✓ Ex) $1.23456e-2 == 0.0123456$

Arithmetic Expressions

$$\frac{3 + 4x}{5} - \frac{10(y - 5)(a + b + c)}{x} + 9\left(\frac{4}{x} + \frac{9 + x}{y}\right)$$

is translated to java code as below:

```
(3+4*x) / 5 - 10 * (y-5) * (a+b+c) / x + 9 * (4/x + (9+x) / y)
```

Operator Precedence Rule

See Appendix. C

- 1) Operators contained **within pairs of parentheses** are evaluated first.
 - ✓ The **inner** parentheses is evaluated **first**
- 2) **Multiplication**, **division**, and **remainder** operators are applied first (left → right)
- 3) **Addition** and **subtraction** are applied last (left → right)

Operator Precedence Rule

$$3 + 4 * 4 + 5 * (4 + 3) - 1$$



(1) inside parentheses first

$$3 + 4 * 4 + 5 * 7 - 1$$



(2) multiplication

$$3 + 16 + 5 * 7 - 1$$



(3) multiplication

$$3 + 16 + 35 - 1$$



(4) addition

$$19 + 35 - 1$$



(5) addition

$$54 - 1$$



(6) subtraction

$$53$$

Problem: Converting Temperatures

- ❑ Write a program that converts a Fahrenheit degree to Celsius using the formula:

$$celsius = (\frac{5}{9})(fahrenheit - 32)$$

- ❑ You have to program

```
celsius = (5.0/9) * (Fahrenheit - 32);
```



[FahrenheitToCelsius](#)

Run

FahrenheitToCelsius.java

```
import java.util.Scanner;

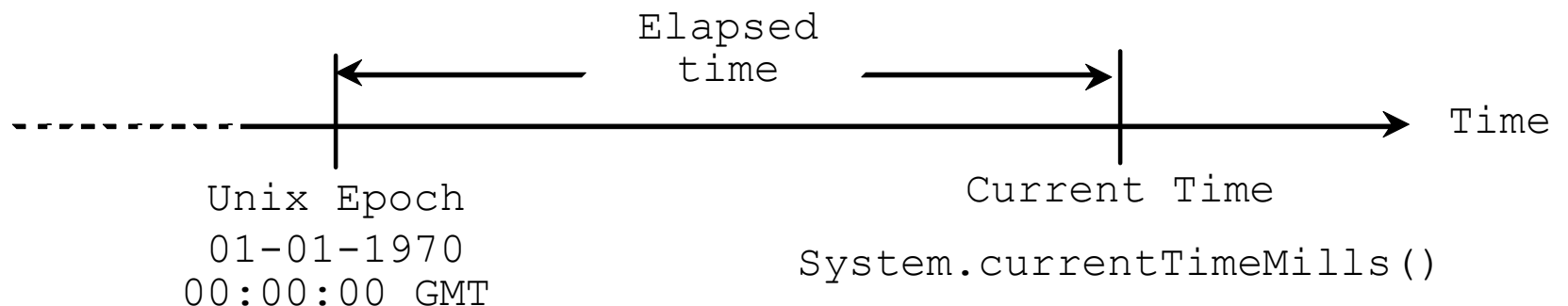
public class FahrenheitToCelsius {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        System.out.print("Enter a degree in Fahrenheit: ");
        double fahrenheit = input.nextDouble();

        // Convert Fahrenheit to Celsius
        double celsius = (5.0 / 9) * (fahrenheit - 32);
        System.out.println("Fahrenheit " + fahrenheit + " is "
            + celsius + " in Celsius");
    }
}
```

Problem: Displaying Current Time

- ❑ Write a program that displays current time in GMT in the format hour:minute:second such as 1:45:19.
 - The `currentTimeMillis()` method in the `System` class returns the current time in milliseconds since the midnight, January 1, 1970 GMT.
 - You can use this method to obtain the current time, and then compute the current second, minute, and hour as follows.



```
public class ShowCurrentTime {  
    public static void main(String[] args) {  
        // Obtain the total milliseconds since midnight, Jan 1, 1970  
        long totalMilliseconds = System.currentTimeMillis();  
  
        // Obtain the total seconds since midnight, Jan 1, 1970  
        long totalSeconds = totalMilliseconds / 1000;  
  
        // Compute the current second in the minute in the hour  
        long currentSecond = totalSeconds % 60;  
  
        // Obtain the total minutes  
        long totalMinutes = totalSeconds / 60;  
  
        // Compute the current minute in the hour  
        long currentMinute = totalMinutes % 60;  
  
        // Obtain the total hours  
        long totalHours = totalMinutes / 60;  
  
        // Compute the current hour  
        long currentHour = totalHours % 24;  
  
        // Display results  
        System.out.println("Current time is " + currentHour + ":"  
            + currentMinute + ":" + currentSecond + " GMT");    }  
}
```

Augmented Assignment Operators

- ❑ The operators `+`, `-`, `*`, `/`, and `%` can be combined with the assignment operator to form augmented operators
 - ✓ The augmented assignment operator is performed last after all the other operators in the expression are evaluated
 - e.g) `x /= 4 + 5.5 * 1.5` is same as `x = x / (4 + 5.5 * 1.5)`

<i>Operator</i>	<i>Name</i>	<i>Example</i>	<i>Equivalent</i>
<code>+=</code>	Addition assignment	<code>i += 8</code>	<code>i = i + 8</code>
<code>-=</code>	Subtraction assignment	<code>i -= 8</code>	<code>i = i - 8</code>
<code>*=</code>	Multiplication assignment	<code>i *= 8</code>	<code>i = i * 8</code>
<code>/=</code>	Division assignment	<code>i /= 8</code>	<code>i = i / 8</code>
<code>%=</code>	Remainder assignment	<code>i %= 8</code>	<code>i = i % 8</code>

Increment and Decrement Operators

- ❑ For incrementing and decrementing a variable by 1
 - ✓ `i++` (plus plus), `i--` (minus minus), `++i`, `--i`

<i>Operator</i>	<i>Name</i>	<i>Description</i>	<i>Example (assume $i = 1$)</i>
<code>++var</code>	preincrement	Increment <code>var</code> by <code>1</code> , and use the new <code>var</code> value in the statement	<code>int j = ++i;</code> // <code>j</code> is 2, <code>i</code> is 2
<code>var++</code>	postincrement	Increment <code>var</code> by <code>1</code> , but use the original <code>var</code> value in the statement	<code>int j = i++;</code> // <code>j</code> is 1, <code>i</code> is 2
<code>--var</code>	predecrement	Decrement <code>var</code> by <code>1</code> , and use the new <code>var</code> value in the statement	<code>int j = --i;</code> // <code>j</code> is 0, <code>i</code> is 0
<code>var--</code>	postdecrement	Decrement <code>var</code> by <code>1</code> , and use the original <code>var</code> value in the statement	<code>int j = i--;</code> // <code>j</code> is 1, <code>i</code> is 0

Increment and Decrement Operators

```
int i = 10;
```

```
int newNum = 10 * i++;
```

Same effect as

```
int newNum = 10 * i;  
i = i + 1;
```

I is 11, newNum is 100

```
int i = 10;
```

```
int newNum = 10 * (++i);
```

Same effect as

```
i = i + 1;  
int newNum = 10 * i;
```

I is 11, newNum is 110

Increment and Decrement Operators

- ❑ Using increment and decrement operators makes expressions **short**, but it also makes them **complex** and **difficult** to read.
- ❑ Avoid using these operators in expressions that modify multiple variables, or the same variable for multiple times.
 - ✓ Ex) `int k = ++i + i ;`

Numeric Type Conversion

- ❑ You can always assign a value to a numeric variable whose type supports a larger range of value (**widening a type**)
 - ✓ ex) assign an `int` value to a `long` variable → OK
- ❑ However, you cannot assign a value to a variable of a type with a smaller range (**narrowing a type**), unless you explicitly use type casting.
 - ✓ ex) `int k = 5 / 2.0;` **Dangerous! Result is a double type (2.5) but k is an int type → Casting! → becomes 2**

Conversion Rules

- 1) If one of the operands is `double`, the other is converted into `double`.
- 2) Otherwise, if one of the operands is `float`, the other is converted into `float`.
- 3) Otherwise, if one of the operands is `long`, the other is converted into `long`.
- 4) Otherwise, `both` operands are converted into `int`.

Type Casting

❑ Casting is an operation that converts a value of one data type into a value of another data type.

1) Implicit casting

– e.g) `double d = 3;` (Type widening)

2) Explicit casting

(target type) variable name or the value to be cast

– e.g) `int i = (int)3.0;` (type narrowing)

i becomes 3

– e.g) `int i = (int)3.9;` (Fraction part is truncated)

i becomes 3

– e.g) `System.out.println((int) 1.7);`

Displays 1

– e.g) `System.out.println((double) 1 / 2);`

Displays 0.5

– e.g) `System.out.println(1 / 2);`

Displays 0

range increases



`byte, short, int, long, float, double`

Caution

- ❑ If **casting** is not used in case of **type narrowing**, a **compile error** will occur.

```
double d = 2.0;  
int i = d;           // Error!
```

- ❑ Casting **does not change** the variable being cast.

```
double d = 2.5;  
int i = (int)d; // i becomes 2, but d is still 2.5
```

Caution

- ❑ In java, an augmented expression of the form **x1 op= x2** is implemented as **x1 = (t) (x1 op x2)**, where t is the type for $x1$.

```
int sum = 0;  
sum += 4.5;
```

// sum becomes 4 after this statement.

// sum += 4.5; is equivalent to

// **sum = (int) (sum + 4.5);**

Caution

- ❑ To assign a variable of the `int` type to a variable of the `short` or `byte` type, **explicit casting MUST** be used.

```
int i = 1;  
byte b = i;           // Error! Because explicit casting is required
```

- ✓ However, so long as the integer literal is within the permissible range of the target variable, explicit casting is not needed to assign an integer literal to a variable of the `short` or `byte` type.

Problem: Keeping Two Digits After Decimal Points

❑ Write a program that displays the sales tax with two digits after the decimal point.

- 1) read your input (purchase amount)
 - using **Scanner**
- 2) compute the sales tax (6%)
 - $\text{Tax} = \text{purchase amount} \times 0.06$
- 3) display the sales tax with two digits after the decimal point
 - `(int)(tax * 100) / 100.0`



SalesTax

Run

Problem: Computing Loan Payments

❑ This program lets the user enter the **annual interest rate(%)**, **number of years**, and **loan amount**, and then computes **monthly payment** and **total payment**.

1) prompt the user to enter the *annual interest rate*, *the number of years*, and *the loan amount*

2) convert *the annual interest rate* to *the monthly interest rate*

– convert *an annual interest rate* in percent format into a decimal

➤ *annual rate* / 100

– divide *the annual rate* by 12

➤ *annual rate* / 12

➤ ex) *annual rate* = 4.5 → *The monthly rate* = 4.5 / 1200 = 0.00375

$$\text{Monthly rate} = \text{annual rate} / 1200$$

3) compute *the monthly payment* using the following formula

$$\text{monthlyPayment} = \frac{\text{loanAmount} \times \text{monthlyInterestRate}}{1 - \frac{1}{(1 + \text{monthlyInterestRate})^{\text{numberOfYears} \times 12}}}$$

4) compute *the total payment*

– *total payment* = *monthly payment* × 12 × *the number of years*

5) display *the monthly payment* and *total payment*.

```

import java.util.Scanner;

public class ComputeLoan {

    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Enter yearly interest rate, for example 8.25: ");
        double annualInterestRate = input.nextDouble();
        double monthlyInterestRate = annualInterestRate / 1200;

        System.out.print("Enter number of years as an integer: ");
        int numberOfYears = input.nextInt();

        System.out.print("Enter loan amount, for example 120000.95: ");
        double loanAmount = input.nextDouble();

        double monthlyPayment = loanAmount * monthlyInterestRate /
            (1 - 1 / Math.pow(1+monthlyInterestRate, numberOfYears*12));
        double totalPayment = monthlyPayment * numberOfYears * 12;

        System.out.println("The monthly payment is $" + (int)(monthlyPayment*100)/100.0);
        System.out.println("The total payment is $" + (int)(totalPayment*100) / 100.0);
    }
}

```

```

c:\book>javac ComputeLoan.java
Compiled successful

c:\book>java ComputeLoan
Enter yearly interest rate, for example 8.25: 5.75
Enter number of years as an integer, for example 5: 15
Enter loan amount, for example 120000.95: 25000.0
The monthly payment is $207.6
The total payment is $37368.45

```



[ComputeLoan](#)

Problem: Monetary Units

- ❑ This program lets the user **enter the amount in decimal** representing dollars and cents and output a report listing **the monetary equivalent in single dollars, quarters, dimes, nickels, and pennies**.
 - 1) Prompt the user to enter the amount as a decimal number. ex) 11.56
 - 2) Convert the amount into **cents**: $cents = amount * 100 = 11.56 * 100 = 1156$
 - 3) Find the number of **dollars** and the remaining cents:
 $dollars = cents / 100 = 1156 / 100 = 11$
 $remaining = cents \% 100 = 1156 \% 100 = 56$
 - 4) Find the number of **quarters** and the remaining cents :
 $quarters = remaining / 25 = 56 / 25 = 2$
 $remaining = remaining \% 25 = 56 \% 25 = 6$
 - 5) Find the number of **dimes** and the remaining cents:
 $dimes = remaining / 10 = 6 / 10 = 0$
 $remaining = remaining \% 10 = 6 \% 10 = 6$
 - 6) Find the number of **nickels** and the number of **pennies**:
 $nickels = remaining / 5 = 6 / 5 = 1$
 $pennies = remaining \% 5 = 6 \% 5 = 1$
 - 7) Display the result.

```
import java.util.Scanner;
```

```
public class ComputeChange {
```

```
    public static void main(String[] args) {
```

```
        Scanner input = new Scanner(System.in);
```

```
        System.out.print("Enter an amount in double, for example 11.56: ");
```

```
        double amount = input.nextDouble();
```

```
        int remainingAmount = (int)(amount * 100);
```

```
        int numberOfOneDollars = remainingAmount / 100;
```

```
        remainingAmount = remainingAmount % 100;
```

```
        int numberOfQuarters = remainingAmount / 25;
```

```
        remainingAmount = remainingAmount % 25;
```

```
        int numberOfDimes = remainingAmount / 10;
```

```
        remainingAmount = remainingAmount % 10;
```

```
        int numberOfNickels = remainingAmount / 5;
```

```
        remainingAmount = remainingAmount % 5;
```

```
        int numberOfPennies = remainingAmount;
```

```
        System.out.println("Your amount " + amount + " consists of");
```

```
        System.out.println("        " + numberOfOneDollars + " dollars");
```

```
        System.out.println("        " + numberOfQuarters + " quarters ");
```

```
        System.out.println("        " + numberOfDimes + " dimes");
```

```
        System.out.println("        " + numberOfNickels + " nickels");
```

```
        System.out.println("        " + numberOfPennies + " pennies");
```

```
    }
```

```
}
```

```
c:\book>java ComputeChange
Enter an amount in double, for example 11.56: 11.56
Your amount 11.56 consists of
    11 dollars
     2 quarters
     0 dimes
     1 nickels
     1 pennies
```

 [ComputeChange](#)

Common Errors and Pitfalls

- ❑ Common Error 1: Undeclared/Uninitialized Variables and Unused Variables
- ❑ Common Error 2: Integer Overflow
- ❑ Common Error 3: Round-off Errors
- ❑ Common Error 4: Unintended Integer Division
- ❑ Common Error 5: Redundant Input Objects

- ❑ Common Pitfall 1: Redundant Input Objects

Common Error 1: Undeclared/Uninitialized Variables and Unused Variables

- ❑ A variable must be declared with a type and assigned a value before using it.

```
double interestRate = 0.05;  
double interest = interestrate * 45;
```

Java is **case sensitive** → *interestRate* and *interestrate* are two different variable → *Interestrate* is not declared and initialized

- ❑ If a variable is declared, but not used in the program, it might be a potential programming error.
 - ✓ You should remove the unused variable from code.

```
double interestRate = 0.05;  
double taxRate = 0.05;  
double interest = interestRate * 45;  
System.out.println("Interest is " + interest);
```

taxRate is **never used** → remove *taxRate*

Common Error 2: Integer Overflow

- ❑ Numbers are stored with a limited number of digits. When a variable is assigned **a value that is too large** (in size) to be stored, it causes **overflow**.

- ✓ Java does not report warnings on overflow!

Overflow! The largest value that can be stored in a variable of the `int` type is **2147483647** → 2147483648 will be too LARGE for an `int` value.

```
int value = 2147483647 + 1;  
// value will actually be 2147483648
```

- ❑ When **a floating-point number is too small** (too close to zero) to be stored, it causes **underflow**.
- ✓ Java approximates it to **zero**.

Common Error 3: Round-off Errors

- ❑ A round-off error (rounding error) is **the difference between the calculated approximation of a number and its exact mathematical value.**
 - ✓ Since the number of digits that can be stored in a variable is limited → round-off errors are inevitable.
 - ✓ Calculations involving floating-point numbers are approximated because these numbers are not stored with complete accurate.

```
System.out.println(1.0 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1);  
System.out.println(1.0 - 0.9);
```

Displays **0.5000000000000001**, not 0.5

Displays **0.09999999999999998**, not 0.1

Common Error 4: Unintended Integer Division

- ❑ When two operands are integer, the / operator performs an **integer division**, the result of the operation is an integer.
 - ✓ **The fractional part is truncated** (Unintended Integer Division)
- ❑ To force two integers to perform a floating-point division, **make one of the integers into a floating-point number**.

Displays 1.0

```
int number1 = 1;  
int number2 = 2;  
double average = (number1 + number2) / 2;  
System.out.println(average);
```

(a)

Displays 1.5

```
int number1 = 1;  
int number2 = 2;  
double average = (number1 + number2) / 2.0;  
System.out.println(average);
```

(b)

Common Pitfall 1: Redundant Input Objects

- ✓ The code is not wrong, but **inefficient**.
 - It creates two input objects unnecessarily and may lead to some subtle errors.

BAD!

```
Scanner input = new Scanner(System.in);  
System.out.print("Enter an integer: ");  
int v1 = input.nextInt();  
  
Scanner input1 = new Scanner(System.in);  
System.out.print("Enter a double value: ");  
double v2 = input1.nextDouble();
```

GOOD!

```
Scanner input = new Scanner(System.in);  
System.out.print("Enter an integer: ");  
int v1 = input.nextInt();  
System.out.print("Enter a double value: ");  
double v2 = input.nextDouble();
```