# Identifiers Assignments Data Types and Operations

# Introduction


- 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.

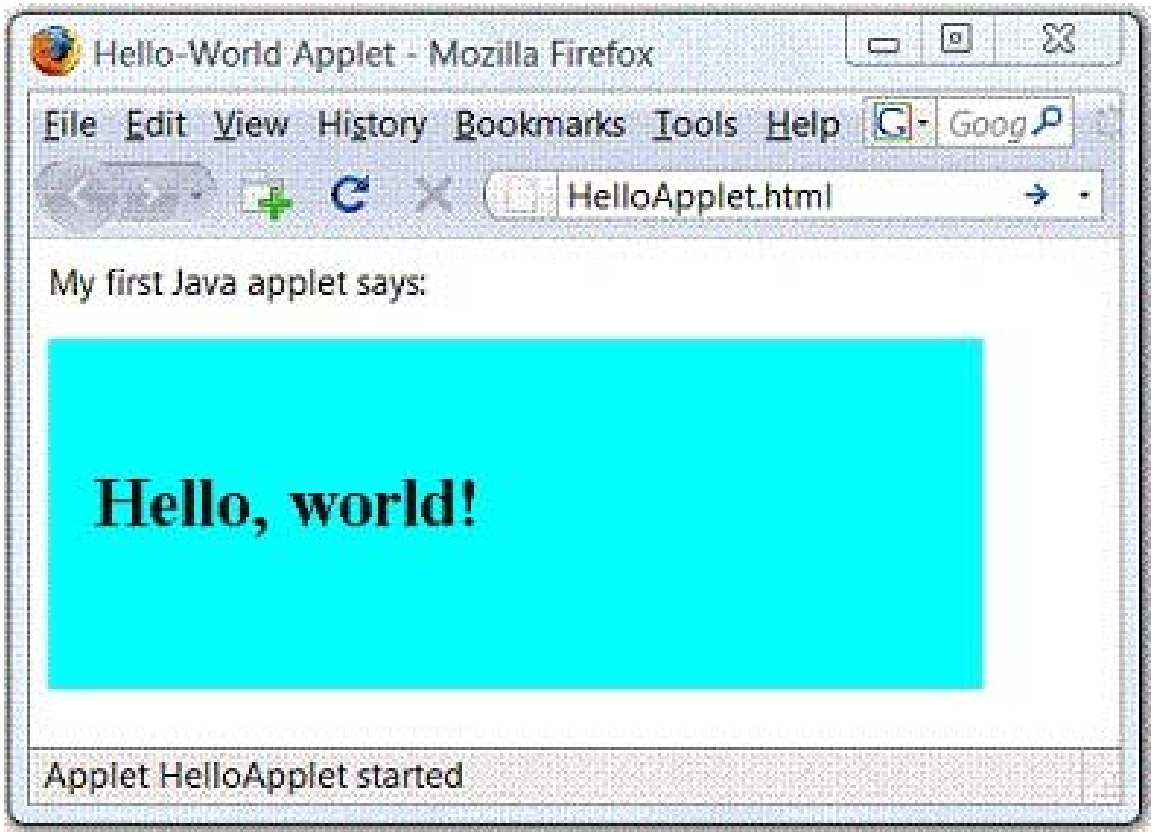
# Java - History


- Java was developed by a team led by James Gosling at Sun Microsystems.
- Sun Microsystems was purchased by Oracle in 2010.
- Originally called Oak, Java was designed in 1991 for use in embedded chips in consumer electronic appliances.
- In 1995, renamed Java, it was redesigned for developing Web applications.

# Java and the Web

- Java initially became attractive because Java programs can be run from a Web browser.
- Such programs are called applets. Applets employ a modern graphical interface with buttons, text fields, text areas, radio buttons, and so on, to interact with users on the Web and process their requests.

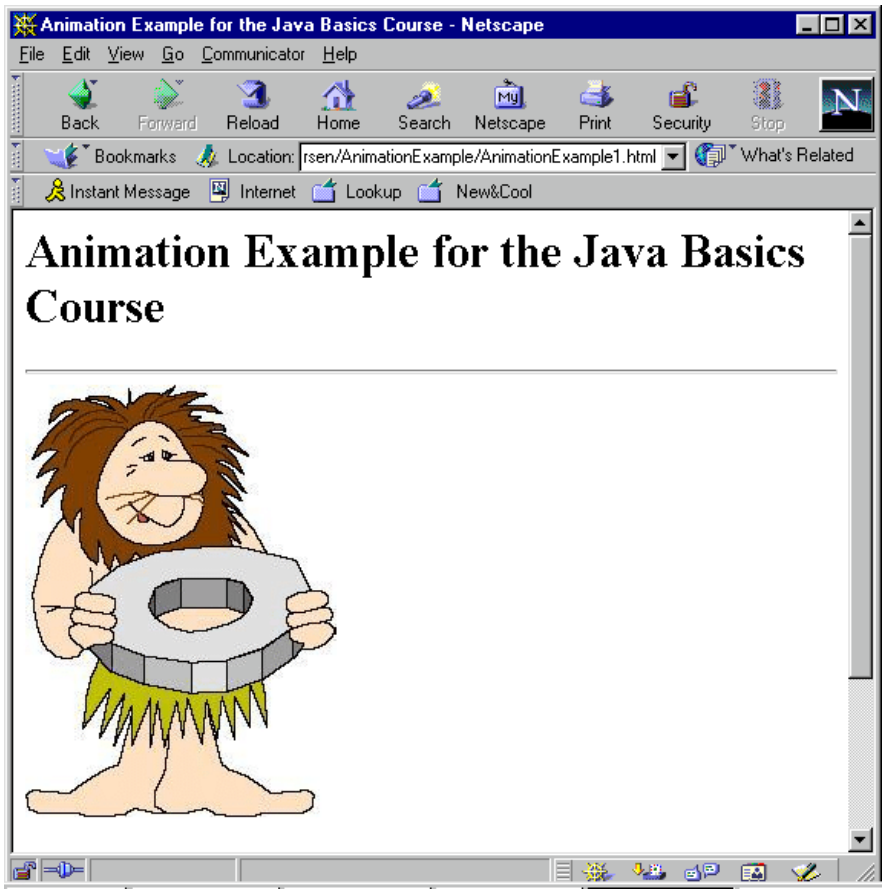
# Java and the Web



# Java and the Web

- Applets make the Web responsive, interactive, and fun to use.
- Applets are embedded in an HTML file. HTML (Hypertext Markup Language) is a simple scripting language for laying out documents, linking documents on the Internet, and bringing images, sound, and video alive on the Web.
- Today, you can use Java to develop rich Internet applications.

# Java and the Web



# Java - API

- The application program interface (API), also known as library, contains predefined classes and interfaces for developing Java programs.

# Java Versions

- Java is a full-fledged and powerful language that can be used in many ways. It comes in three editions:
  - Java Standard Edition (Java SE) to develop client-side applications. The applications can run standalone or as applets running from a Web browser.

# Java Versions

- Java Enterprise Edition (Java EE) to develop server-side applications, such as Java servlets, JavaServer Pages (JSP), and JavaServer Faces (JSF).
- Java Micro Edition (Java ME) to develop applications for mobile devices, such as cell phones.

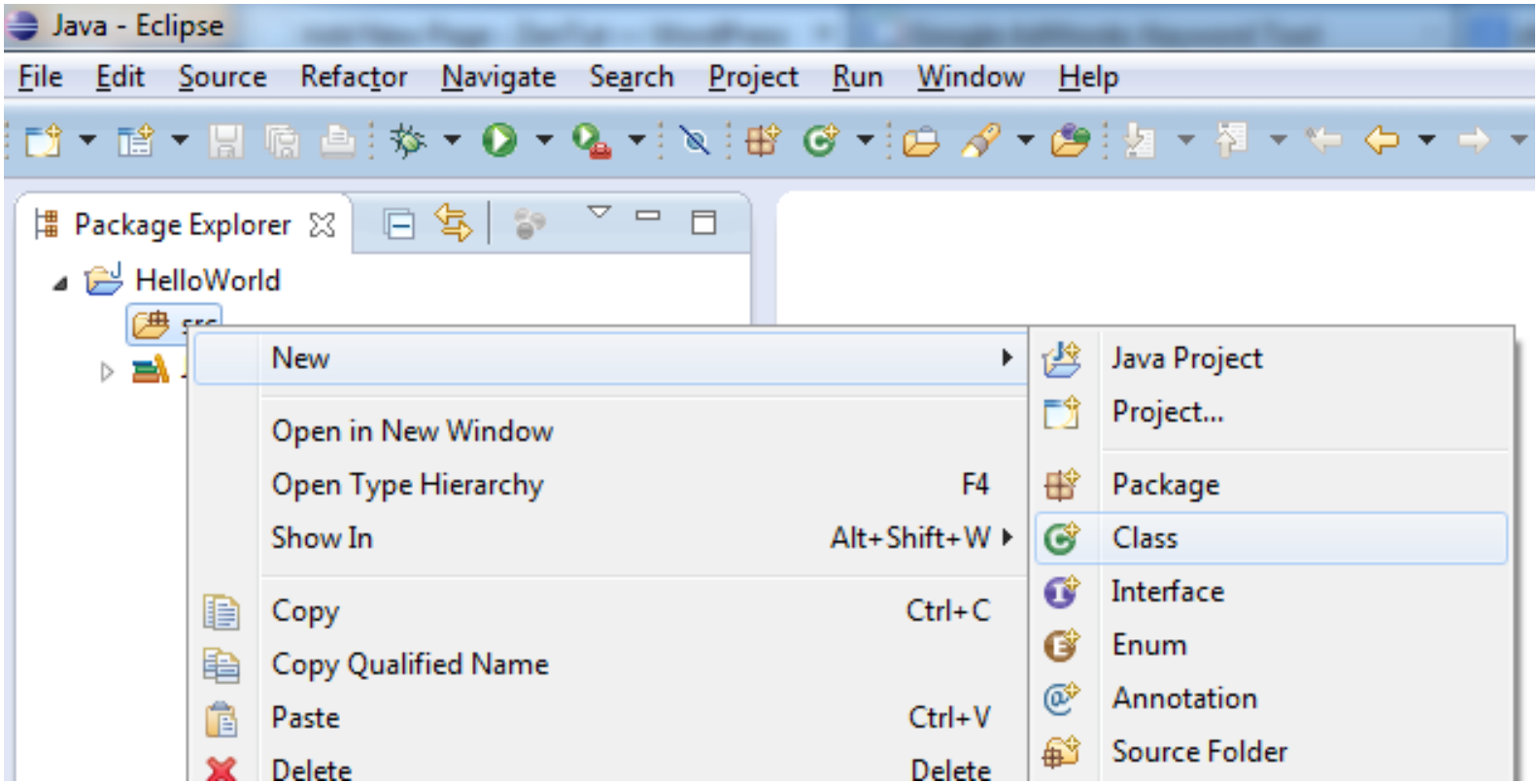
# Java SE and JDK

- Java SE is the foundation upon which all other Java technology is based.
- There are many versions of Java SE. Java SE 8, is used in our course.
- Oracle releases each version with a Java Development Toolkit (JDK).
- For Java SE 8, the Java Development Toolkit is called JDK 1.8 (also known as Java 8 or JDK 8).

# Java IDE

- The JDK consists of a set of separate programs, each invoked from a command line, for developing and testing Java programs.
- Instead of using the JDK, 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

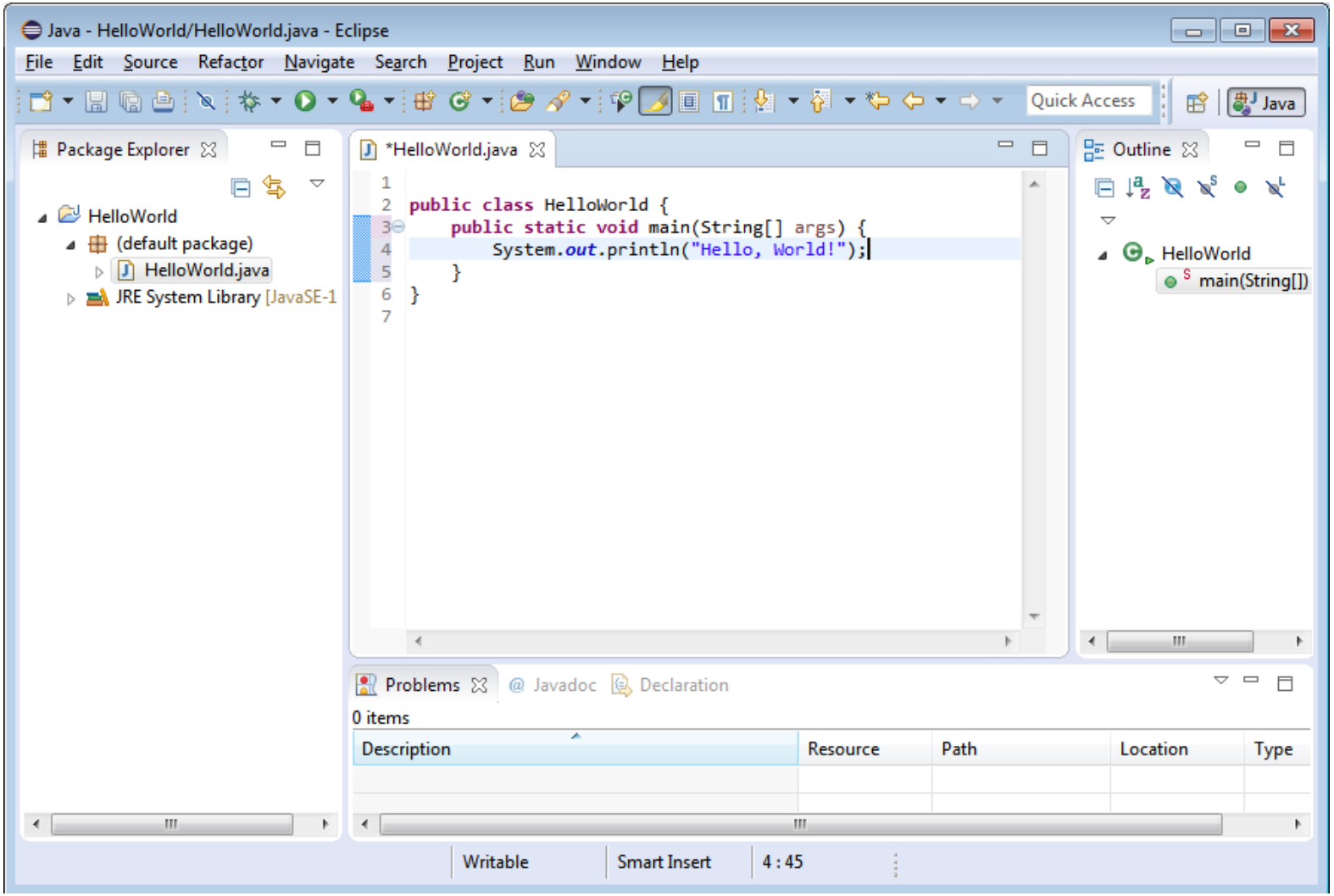
# Java IDE - Eclipse



# Java IDE - Eclipse

- Editing, compiling, building, debugging, and online help are integrated in one graphical user interface.
- You simply enter source code in one window or open an existing file in a window, and then click a button or menu item or press a function key to compile and run the program.

# Java IDE - Eclipse



# A Simple Java Program

**LISTING 1.1** Welcome.java

Class name and file name should match!!!

```
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  }
```

```
Welcome to Java!
```

# A Simple Java Program

- Each class has a name.
- By convention, class names start with an uppercase letter.
- In this example, the class name is Welcome.

# A Simple Java Program

- 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.

# A Simple Java Program

- 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.

# A Simple Java Program

- 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

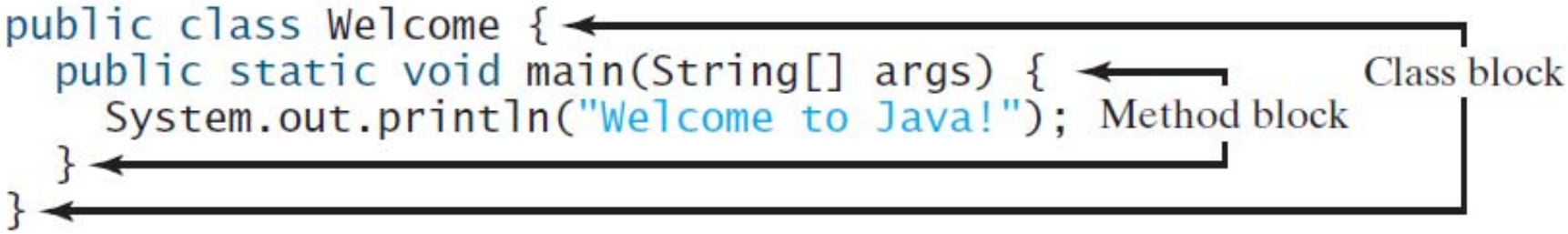
- Reserved words, or keywords, have a specific meaning to the compiler and cannot be used for other purposes in the program.
- For example, when the compiler sees the word class, it understands that the word after class is the name for the class.
- Other reserved words in this program are public, static,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

# Comments

- 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.

# Blocks

- A pair of curly braces {} in a program forms a block that groups the program's components.



# 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. |

# A Second Simple Program

**LISTING 1.2** WelcomeWithThreeMessages.java

```
1  public class WelcomeWithThreeMessages {
2    public static void main(String[] args) {
3      System.out.println("Programming is fun!");
4      System.out.println("Fundamentals First");
5      System.out.println("Problem Driven");
6    }
7  }
```

```
Programming is fun!
Fundamentals First
Problem Driven
```

# A Third Simple Program

- Further, you can perform mathematical computations and display the result on the console.

**LISTING 1.3** ComputeExpression.java

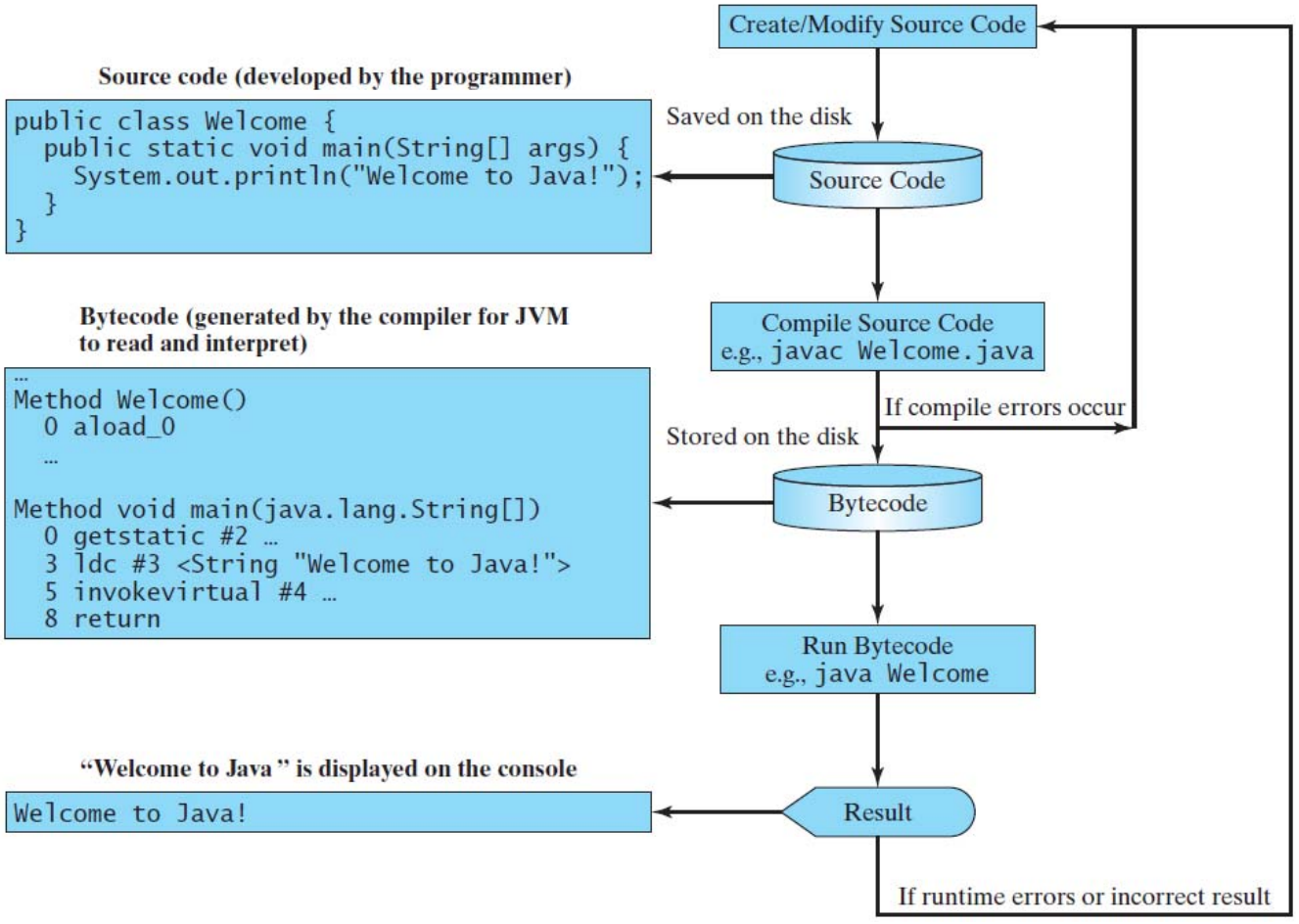
```
1  public class ComputeExpression {
2    public static void main(String[] args) {
3      System.out.println((10.5 + 2 * 3) / (45 - 3.5));
4    }
5  }
```

```
0.39759036144578314
```

# Java Compiling

- A Java compiler translates a Java source file into a Java bytecode file.
- If there aren't any syntax errors, the compiler generates a bytecode file with a .class extension, e.g.,Welcome.class.

# Java Compiling



# Java Virtual Machine

- 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).

# Java Virtual Machine

- 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.

# Exercise

- Show the output of the following code:

```
public class Test {
  public static void main(String[] args) {
    System.out.println("3.5 * 4 / 2 - 2.5 is ");
    System.out.println(3.5 * 4 / 2 - 2.5);
  }
}
```

# Class Name and File Name

- With some compilers, the class name could be different than file name if the class is not declared public:

Main.java

```
1  // Online Java Compiler
2  // Use this editor to write, compile and run your Java code online
3
4  class HelloWorld {
5      public static void main(String[] args) {
6          System.out.println("Hello, World!");
7      }
8  }
```

# Reading Input from the Console

- Java uses System.out to refer to the standard output device:

  System.out.println("Hello World!").

- By default, the output device is the display monitor.
- To perform console output, you simply use the println method to display a primitive value or a string to the console.

# Class Scanner

- Java uses System.in to refer to the standard input device:
- By default, the input device is the keyboard.
- Console input is not directly supported in Java, but you can use the Scanner class to create an object to read input from System.in.

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

# Class Scanner

- The syntax new Scanner(System.in) creates an object of the Scanner type.
- The syntax Scanner input declares that input is a variable whose type is Scanner.
- The whole line

  Scanner input = new Scanner(System.in)

  creates a Scanner object and assigns its reference to the variable input.

# Class Scanner Methods

| Method | Description |
| --- | --- |
| `nextInt()` | reads an `int` value from the user |
| `nextFloat()` | reads a `float` value form the user |
| `nextBoolean()` | reads a `boolean` value from the user |
| `nextLine()` | reads a line of text from the user |
| `next()` | reads a word from the user |
| `nextByte()` | reads a `byte` value from the user |
| `nextDouble()` | reads a `doubl`e value from the user |
| `nextShort()` | reads a `short` value from the user |
| `nextLong()` | reads a `long` value from the user |

# Scanner with Different Input Sources

- Here, we creat objects of the Scanner class that will read input from InputStream, File, and String respectively.

```
// read input from the input stream
Scanner sc1 = new Scanner(InputStream input);

// read input from files
Scanner sc2 = new Scanner(File file);

// read input from a string
Scanner sc3 = new Scanner(String str);
```

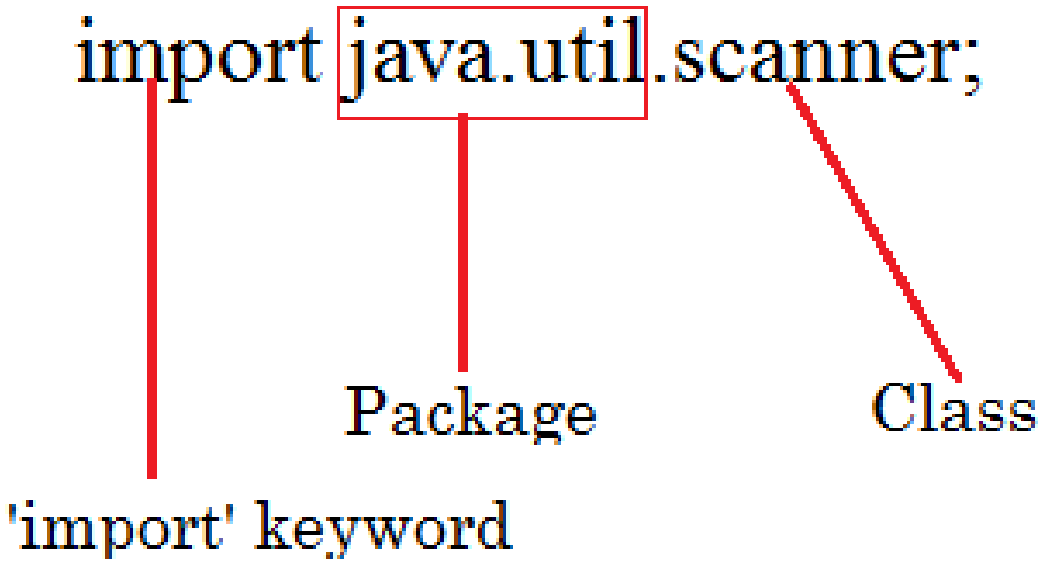
# Java Package: java.util

- In order to be able to use the class Scanner, we need to import the java.util.Scanner package before we can use the Scanner class.

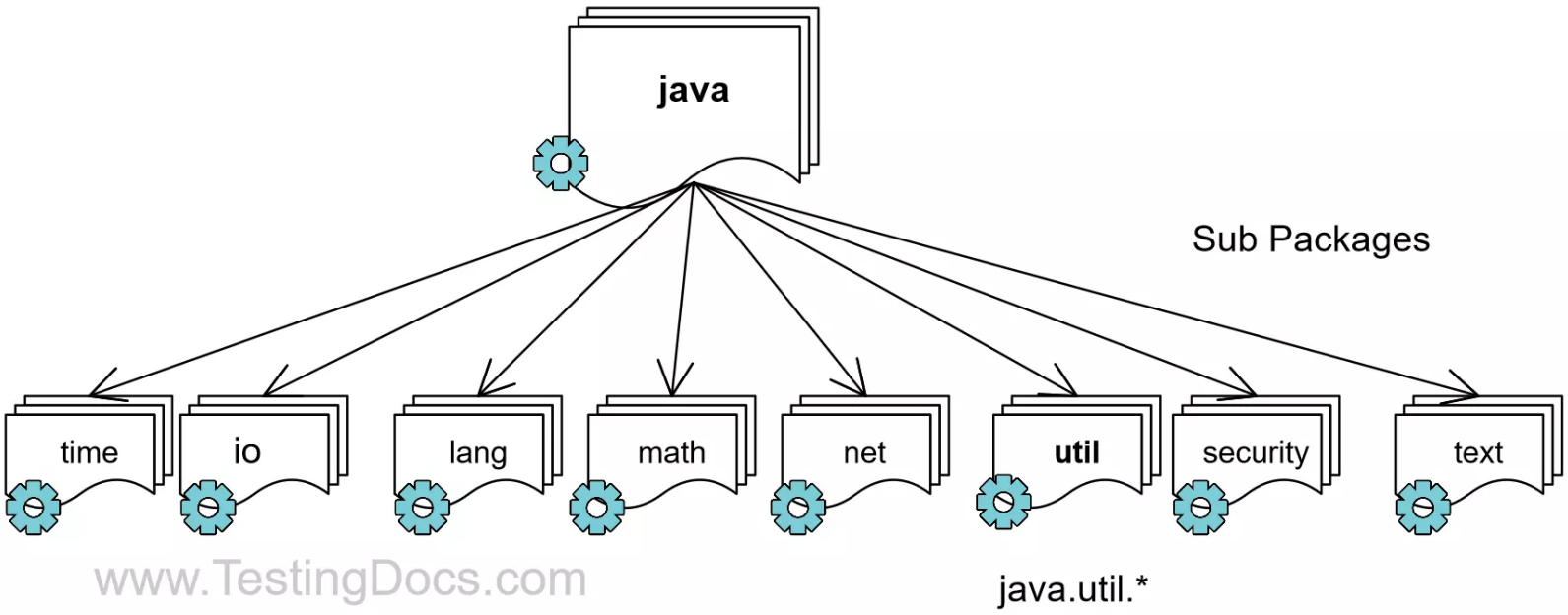
```
import java.util.Scanner;
```

- A package is simply a container that groups related types (Java classes, interfaces, enumerations, and annotations).

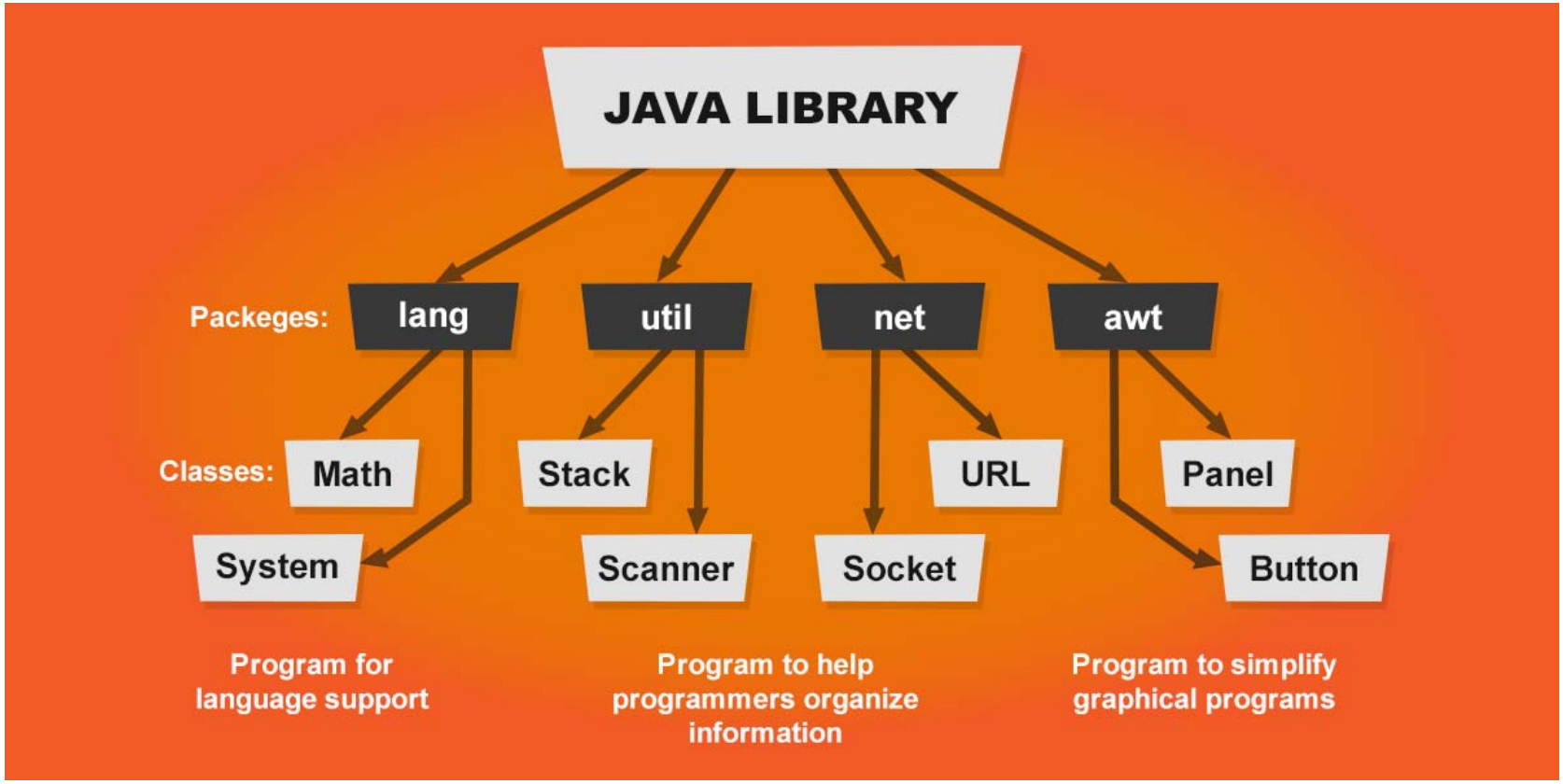
# Java Package: java.util



# Java Packages

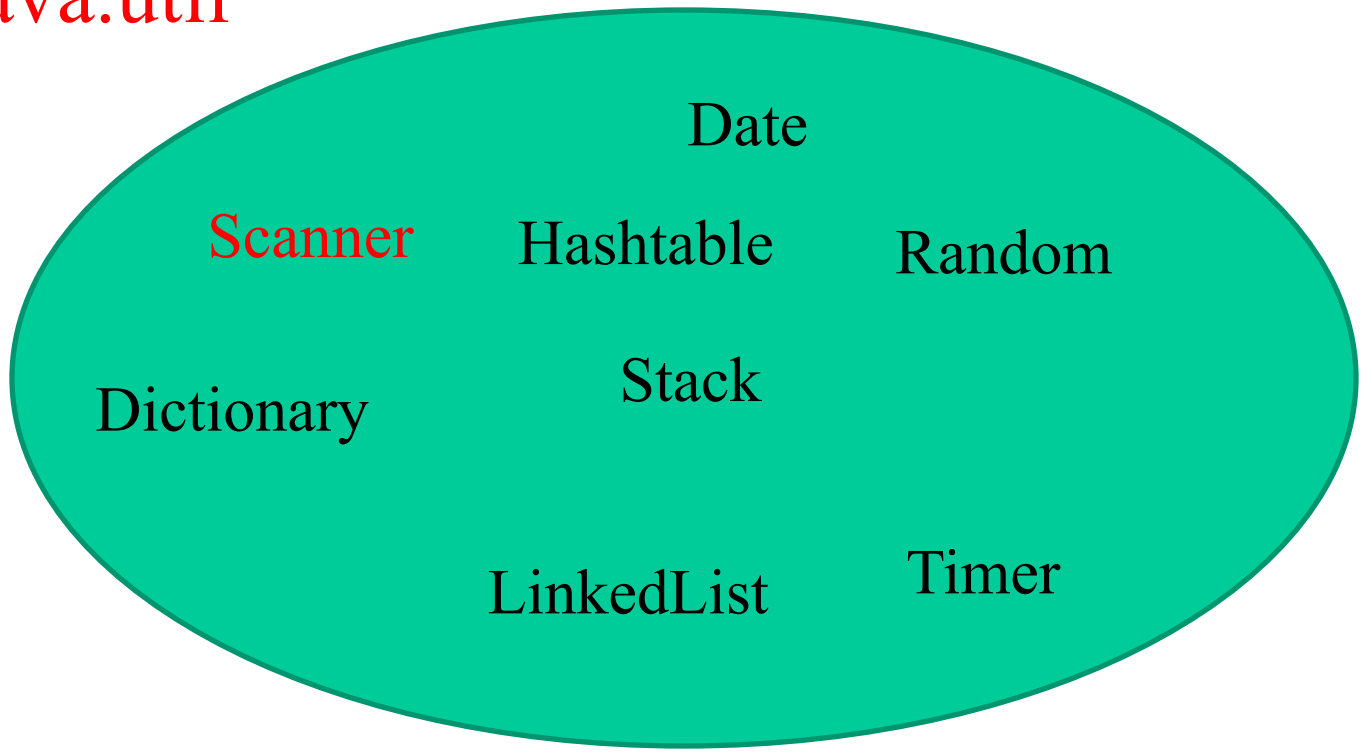


# Java Packages

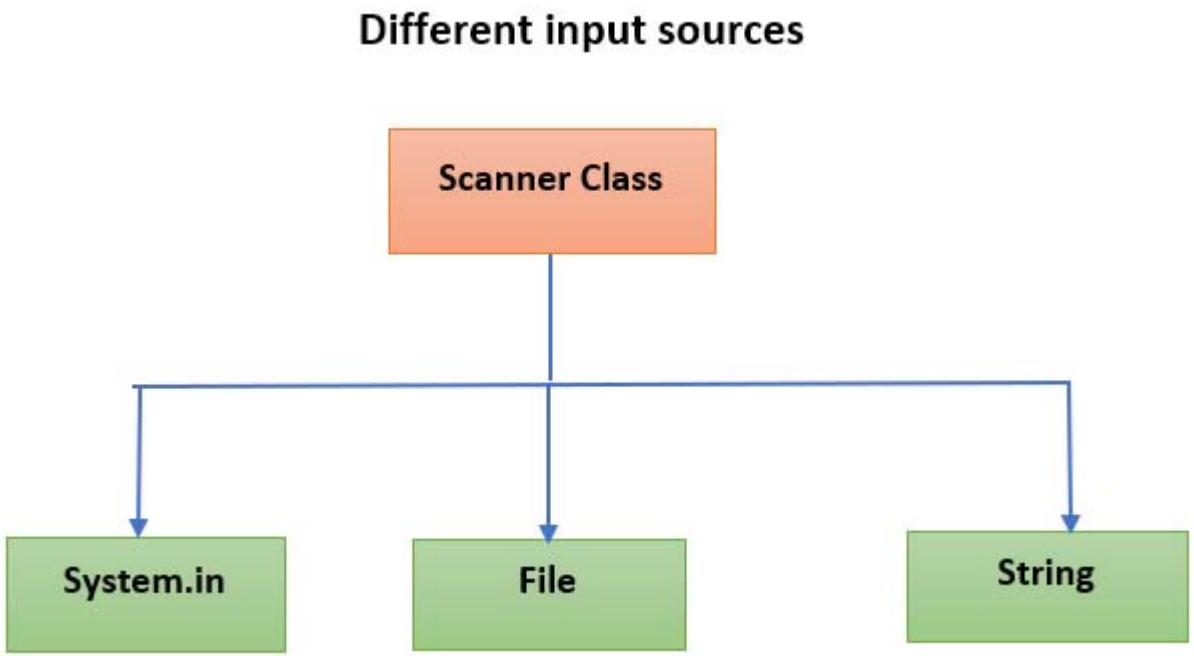


# Java.util classes

- Java.util



# Class Scanner



# Reading Input from the Console

- An object may invoke its methods.
- To invoke a method on an object is to ask the object to perform a task.
- You can invoke the nextDouble() method to read a double value as follows:

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

# Complete Example

**LISTING 2.2** ComputeAreaWithConsoleInput.java

```
1  import java.util.Scanner; // Scanner is in the java.util package
2
3  public class ComputeAreaWithConsoleInput {
4    public static void main(String[] args) {
5      // Create a Scanner object
6      Scanner input = new Scanner(System.in);
7
8      // Prompt the user to enter a radius
9      System.out.print("Enter a number for radius: ");
10     double radius = input.nextDouble();
11
12     // Compute area
13     double area = radius * radius * 3.14159;
14
15     // Display results
16     System.out.println("The area for the circle of radius " +
17       radius + " is " + area);
18   }
19 }
```

import class

create a Scanner

read a double

# Complete Example

```
Enter a number for radius:  2.5 Enter
The area for the circle of radius 2.5 is 19.6349375
```

```
Enter a number for radius:  23 Enter
The area for the circle of radius 23.0 is 1661.90111
```

# Example: Java Scanner nextInt()

```java
import java.util.Scanner;

class Main {
  public static void main(String[] args) {

    // creates a Scanner object
    Scanner input = new Scanner(System.in);

    System.out.println("Enter an integer: ");

    // reads an int value
    int data1 = input.nextInt();

    System.out.println("Using nextInt(): " + data1);

    input.close();
  }
}
```

```
Enter an integer:
22
Using nextInt(): 22
```

# Example: Java Scanner next()

```java
import java.util.Scanner;

class Main {
  public static void main(String[] args) {

    // creates an object of Scanner
    Scanner input = new Scanner(System.in);
    System.out.print("Enter your name: ");

    // reads the entire word
    String value = input.next();
    System.out.println("Using next(): " + value);

    input.close();
  }
}
```

```
Enter your name: Jonny Walker
Using next(): Jonny
```

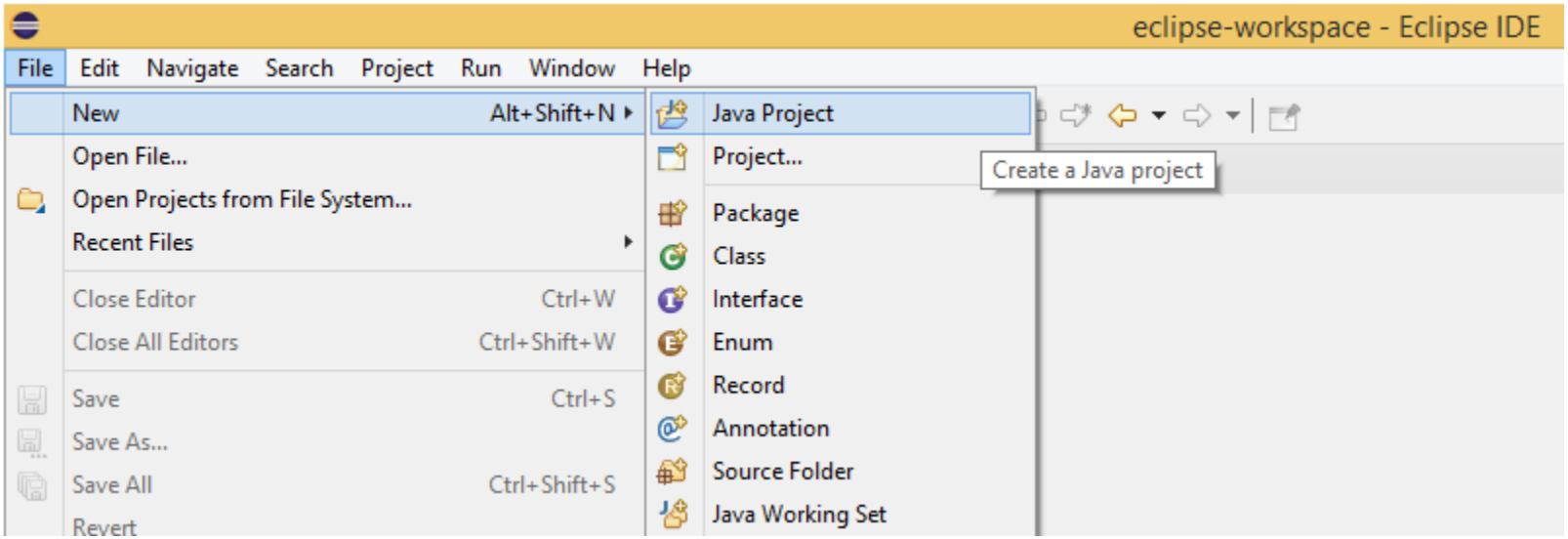
# Example: Java Scanner nextLine()

```java
import java.util.Scanner;

class Main {
  public static void main(String[] args) {

    // creates an object of Scanner
    Scanner input = new Scanner(System.in);
    System.out.print("Enter your name: ");

    // reads the entire line
    String value = input.nextLine();
    System.out.println("Using nextLine(): " + value);

    input.close();
  }
}
```

```
Enter your name: Jonny Walker
Using nextLine(): Jonny Walker
```

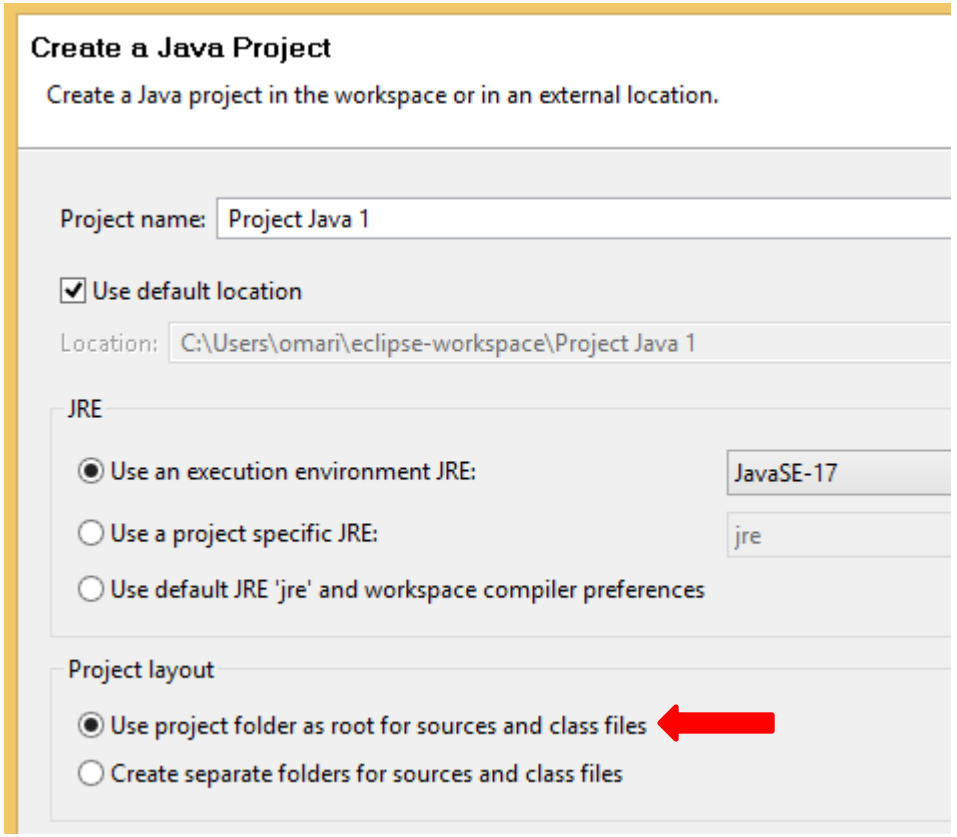
# Creating a Java Project

- Before creating Java programs in Eclipse, you need to first create a project to hold all files.



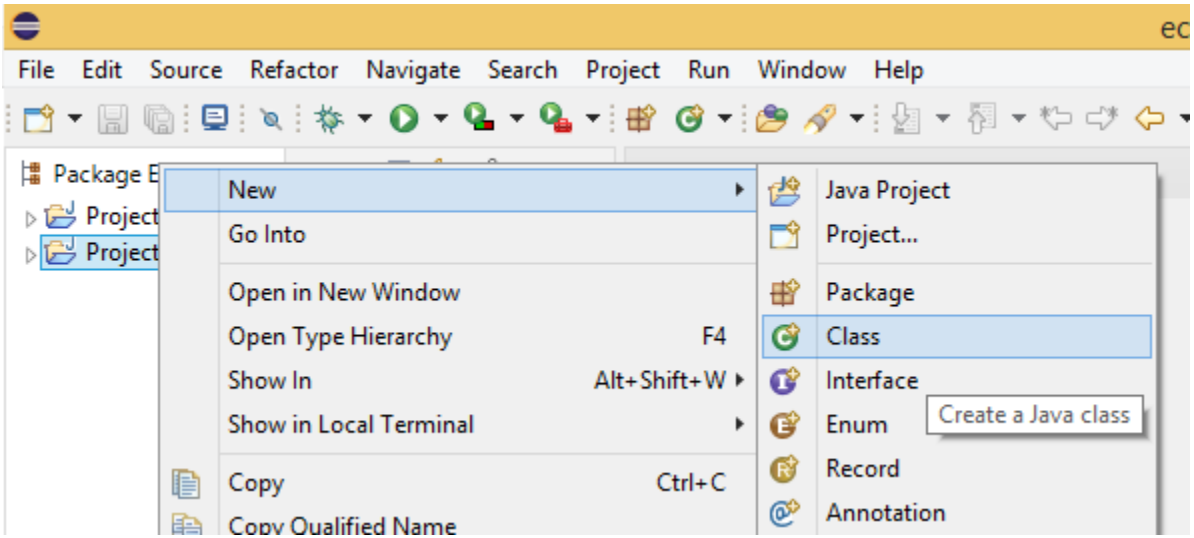
# Creating a Java Project

- Make sure that you selected the options 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.



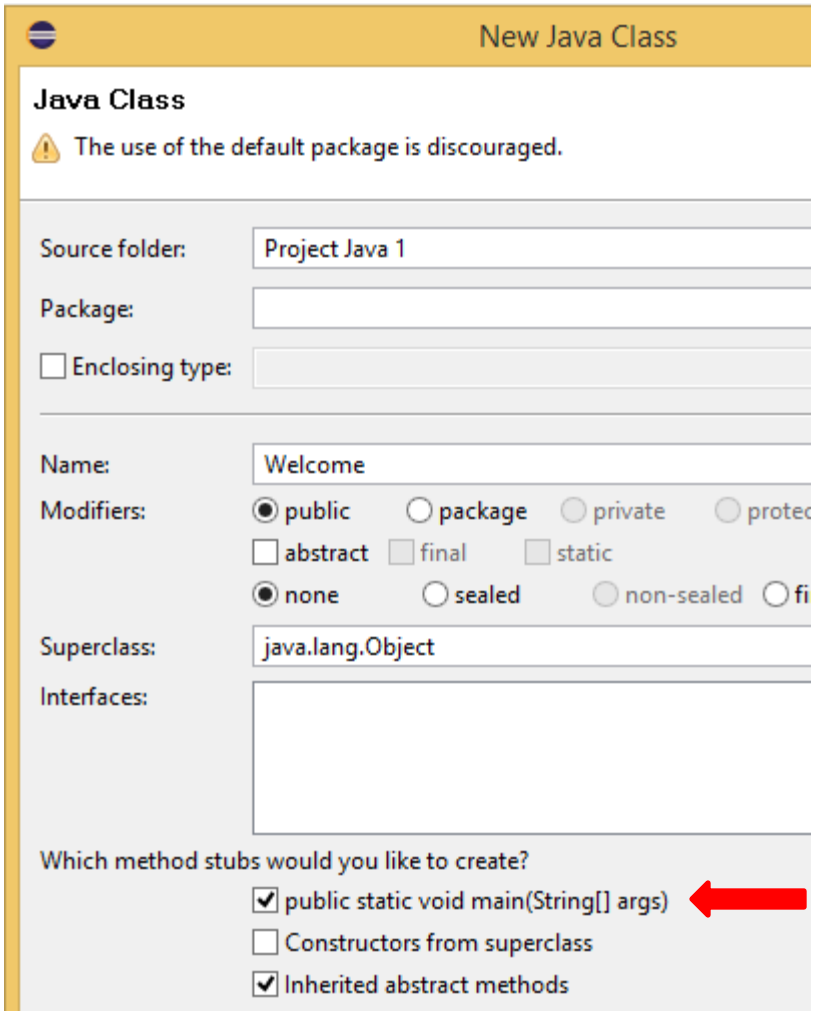
# Creating a Java Class

- Right click the project just created (on the left), then select new, and then select class.



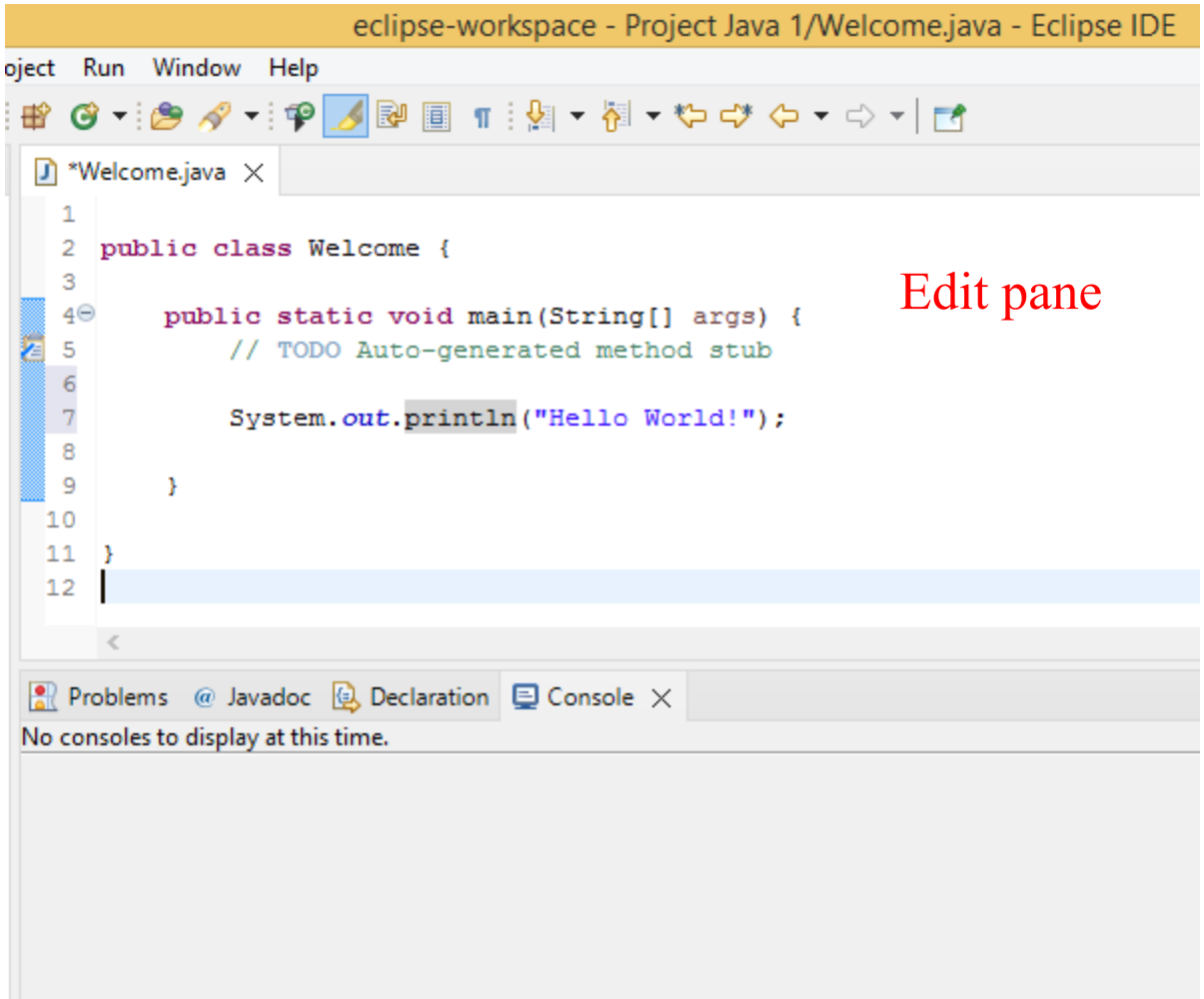
# Creating a Java Class

- Select a name for the class (e.g., Welcome), and select the option public static void main.



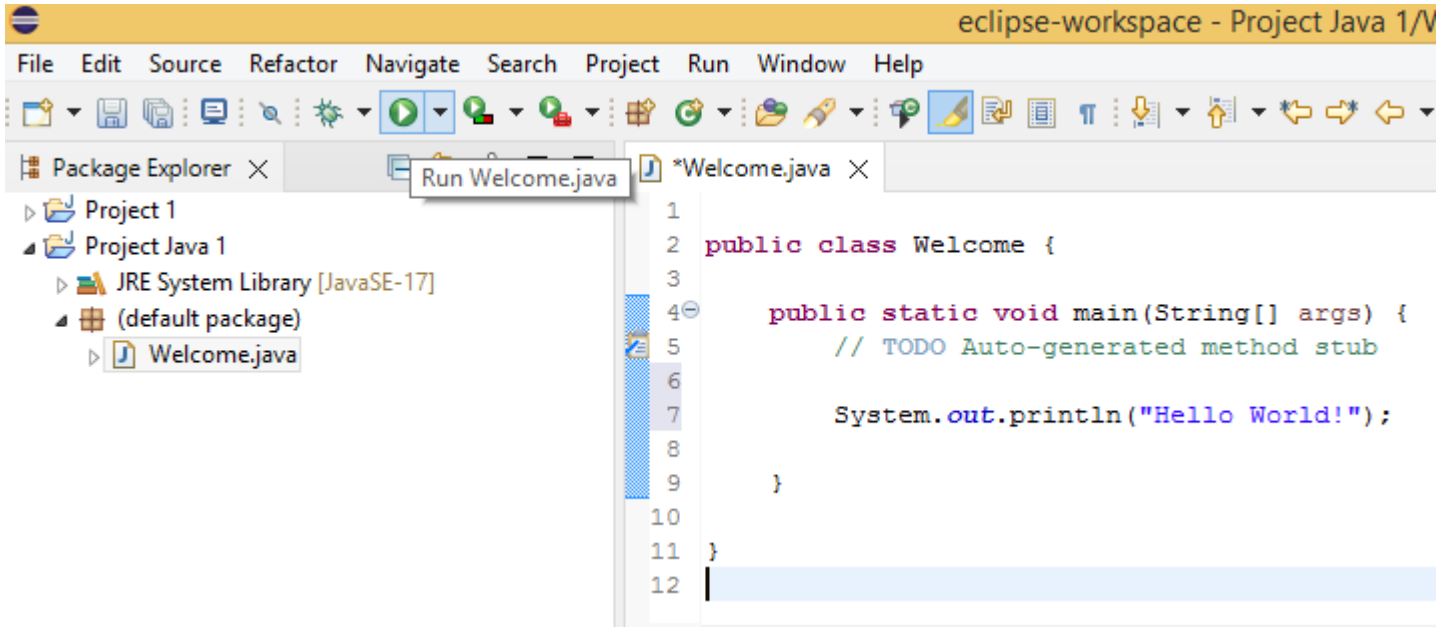
# Adding code to a Java Class

- Add java code to the edit pane like System.out.println().



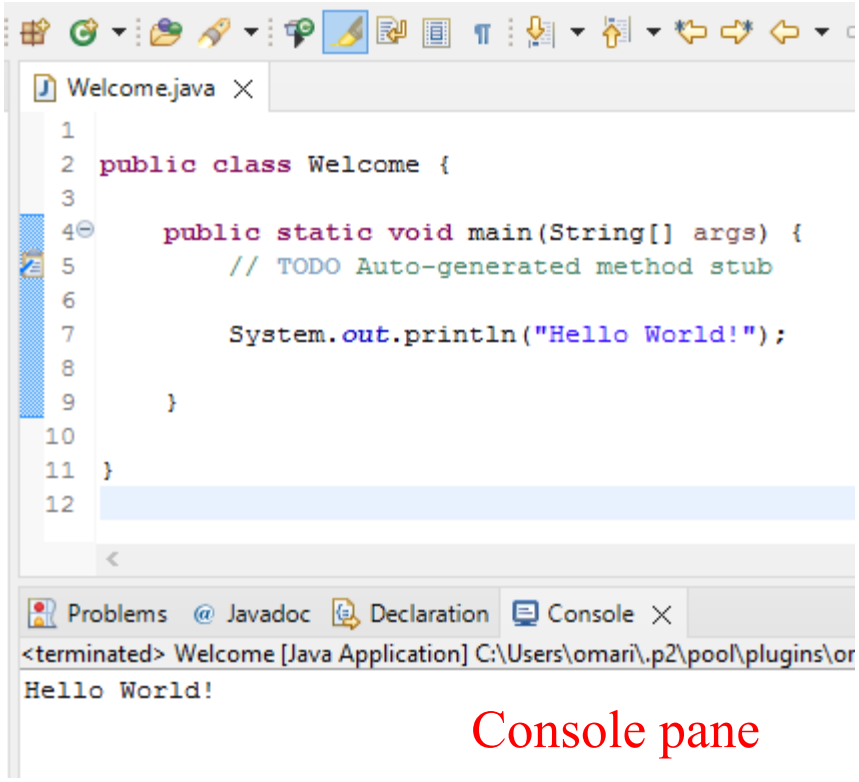
# Compiling a Java Class

- Click on the green play button to run the java code.



# Output pane in Eclipse

- The output is shown down at the console pane.




Console pane

# Identifiers

- Identifiers are the names that identify the elements such as classes, methods, and variables in a program.
  - An identifier is a sequence of characters that consists of letters, digits, underscores (_), and dollar signs ($).
  - An identifier must start with a letter, an underscore (_), or a dollar sign ($). It cannot start with a digit.

# Identifiers - Rules

- An identifier cannot be a reserved word.
- An identifier cannot be true, false, or null.
- An identifier can be of any length.
- For example, $2, ComputeArea, area, radius, and print are legal identifiers, whereas 2A and d+4 are not because they do not follow the rules.
- The Java compiler detects illegal identifiers and reports syntax errors.
- Since Java is case sensitive, area, Area, and AREA are all different identifiers.

# Variables

- Variables are used to store values to be used later in a program.
- They are called variables because their values can be changed.

```
1  // Compute the first area
2  radius = 1.0;                                          radius:  1.0
3  area = radius * radius * 3.14159;                        area:  3.14159
4  System.out.println("The area is " + area + " for radius " + radius);
5
6  // Compute the second area
7  radius = 2.0;                                          radius:  2.0
8  area = radius * radius * 3.14159;                        area:  12.56636
9  System.out.println("The area is " + area + " for radius " + radius);
```

# Variables

- Variables are for representing data of a certain type.
- To use a variable, you declare it by telling the compiler its name as well as what type of data it can store.
- The variable declaration tells the compiler to allocate appropriate memory space for the variable based on its data type.

# Variables and Data Types

- The syntax for declaring a variable is

  datatype variableName;

- Here are some examples of variable declarations:

```
int count; // Declare count to be an integer variable
double radius; // Declare radius to be a double variable
double interestRate; // Declare interestRate to be a double
    variable
```

- These examples use the data types int and double.

# Variables and Data Types

- The variables are separated by commas.
- For example,

```
int i, j, k; // Declare i, j, and k as int
```

# Variables and Data Types

- Variables often have initial values. You can declare a variable and initialize it in one step.

```
int count = 1;
```

- This is equivalent to the next two statements:

```
int count;
count = 1;
```

- You can also use a shorthand form to declare and initialize variables of the same type together.

```
int i = 1, j = 2;
```

# Variables Declaration Rules

- A variable must be declared before it can be assigned a value.
- A variable declared in a method must be assigned a value before it can be used.
- Whenever possible, declare a variable and assign its initial value in one step.
- This will make the program easy to read and avoid programming errors.

# Assignments

- After a variable is declared, you can assign a value to it by using an assignment statement.
- In Java, the equal sign (=) is used as the assignment operator.
- The syntax for assignment statements is as follows:

variable = expression;

# Assignments

- An expression represents a computation involving values, variables, and operators that, taking them together, evaluates to a value.

```
int y = 1;                          // Assign 1 to variable y
double radius = 1.0;                // Assign 1.0 to variable radius
int x = 5 * (3 / 2);                // Assign the value of the expression to x
x = y + 1;                          // Assign the addition of y and 1 to x
double area = radius * radius * 3.14159; // Compute area
```

# Assignments

- If a value is assigned to multiple variables, you can use this syntax:

  ```
  i = j = k = 1;
  ```

- which is equivalent to

  ```
  k = 1;
  j = k;
  i = j;
  ```

# Constants

- A named constant is an identifier that represents a permanent value.
- The value of a variable may change during the execution of a program, but a named constant, or simply constant, represents permanent data that never changes.

# Constants

- Here is the syntax for declaring a constant:

  final datatype CONSTANTNAME = value;

- A constant must be declared and initialized in the same statement.
- The word final is a Java keyword for declaring a constant.

# Constants

- For example, you can declare pi as a constant:

**LISTING 2.4** ComputeAreaWithConstant.java

```
1  import java.util.Scanner; // Scanner is in the java.util package
2
3  public class ComputeAreaWithConstant {
4    public static void main(String[] args) {
5      final double PI = 3.14159; // Declare a constant
6
7      // Create a Scanner object
8      Scanner input = new Scanner(System.in);
9
10     // Prompt the user to enter a radius
11     System.out.print("Enter a number for radius: ");
12     double radius = input.nextDouble();
13
14     // Compute area
15     double area = radius * radius * PI;
16
17     // Display result
18     System.out.println("The area for the circle of radius " +
19       radius + " is " + area);
20   }
21 }
```

# Constants

- There are three benefits of using constants:
  - (1) you don't have to repeatedly type the same value if it is used multiple times;
  - (2) if you have to change the constant value (e.g., from 3.14 to 3.14159 for PI), you need to change it only in a single location in the source code;
  - (3) a descriptive name for a constant makes the program easy to read.

# Naming Conventions in Java

- Use lowercase for variables and methods.
  - for example, the variables radius and area and the method print.
  - If a variable/method name has many syllables, capitalize the syllables' first letters starting from the second syllable: loanAmount, secondSemesterYear2000.

# Naming Conventions in Java

- Capitalize the first letter of each word in a class name.
  - for example, the class names Welcome and System.
  - If a class name has many syllables, capitalize the syllables' first letters: Compute Area.
- Capitalize every letter in a constant, and use underscores between words
  - for example, the constants PI and MAX_VALUE.

# Numeric Data Types

**TABLE 2.1** Numeric Data Types

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

# Numeric Data Types

- Every data type has a range of values.
- The compiler allocates memory space for each variable or constant according to its data type.
- Java provides eight primitive data types for numeric values, characters, and Boolean values.
- Numeric data types are: byte, short, int, long, float and double.

# Reading Numbers from the Keyboard

**TABLE 2.2** Methods for `Scanner` Objects

| *Method* | *Description* |
|---|---|
| `nextByte()` | reads an integer of the `byte` type. |
| `nextShort()` | reads an integer of the `short` type. |
| `nextInt()` | reads an integer of the `int` type. |
| `nextLong()` | reads an integer of the `long` type. |
| `nextFloat()` | reads a number of the `float` type. |
| `nextDouble()` | reads a number of the `double` type. |

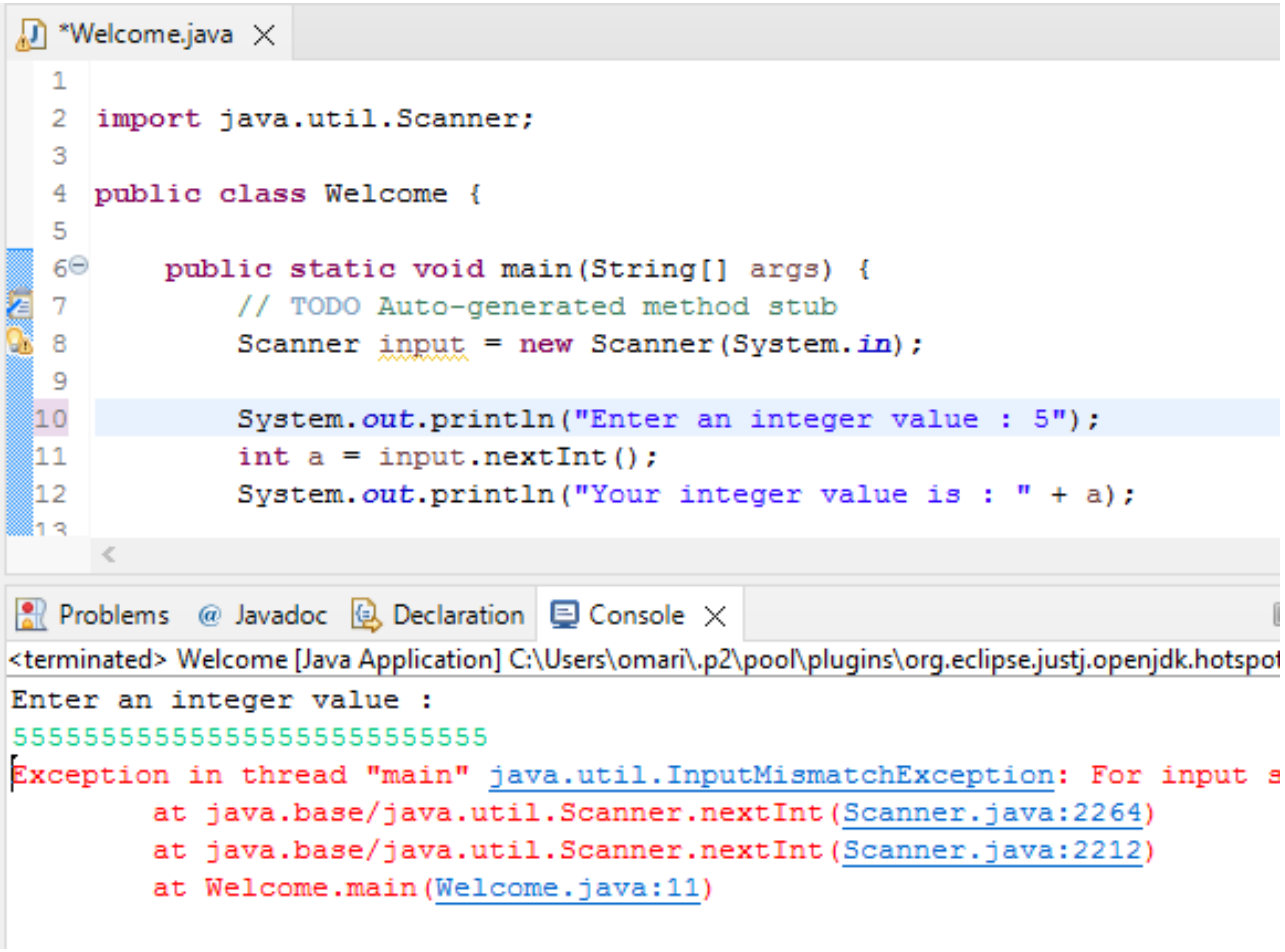
# Reading Numbers from the Keyboard

- Here are examples for reading values of various types from the keyboard:

```
1  Scanner input = new Scanner(System.in);
2  System.out.print("Enter a byte value: ");
3  byte byteValue = input.nextByte();
4
5  System.out.print("Enter a short value: ");
6  short shortValue = input.nextShort();
7
8  System.out.print("Enter an int value: ");
9  int intValue = input.nextInt();
10
11 System.out.print("Enter a long value: ");
12 long longValue = input.nextLong();
13
14 System.out.print("Enter a float value: ");
15 float floatValue = input.nextFloat();
```

# Reading Numbers from the Keyboard

- If you enter a value with an incorrect range or format, a runtime error would occur.



# Numeric Operators

- The operators for numeric data types include the standard arithmetic operators: addition (+), subtraction (–), multiplication (*), division (/), and remainder (%).

**TABLE 2.3** Numeric 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 |

# Example

**LISTING 2.5** DisplayTime.java

```
1  import java.util.Scanner;
2
3  public class DisplayTime {
4    public static void main(String[] args) {
5      Scanner input = new Scanner(System.in);
6      // Prompt the user for input
7      System.out.print("Enter an integer for seconds: ");
8      int seconds = input.nextInt();
9
10     int minutes = seconds / 60; // Find minutes in seconds
11     int remainingSeconds = seconds % 60; // Seconds remaining
12     System.out.println(seconds + " seconds is " + minutes +
13       " minutes and " + remainingSeconds + " seconds");
14   }
15 }
```

```
Enter an integer for seconds: 500 Enter
500 seconds is 8 minutes and 20 seconds
```

# Exponent Operations

- The 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.

```
System.out.println(Math.pow(2, 3)); // Displays 8.0
System.out.println(Math.pow(4, 0.5)); // Displays 2.0
System.out.println(Math.pow(2.5, 2)); // Displays 6.25
System.out.println(Math.pow(2.5, -2)); // Displays 0.16
```

# Numeric Literals - Integers

- A literal is a constant value that appears directly in a program.
  - int numberOfYears = 34;
  - double weight = 0.305;
  - long distance = 2147483648L
- Binary values: 0B1111 (means 15).
- Octal values: 07777 (means 4095).
- Hexadecimal values: 0XFFFF (means 65365).

# Numeric Literals - Floats

- Floating-point literals are written with a decimal point.
- By default, a floating-point literal is treated as a double type value.
- For example, 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, and you can make a number a double by appending the letter d or D.
- For example, you can use 100.2f or 100.2F for a float number, and 100.2d or 100.2D for a double number.

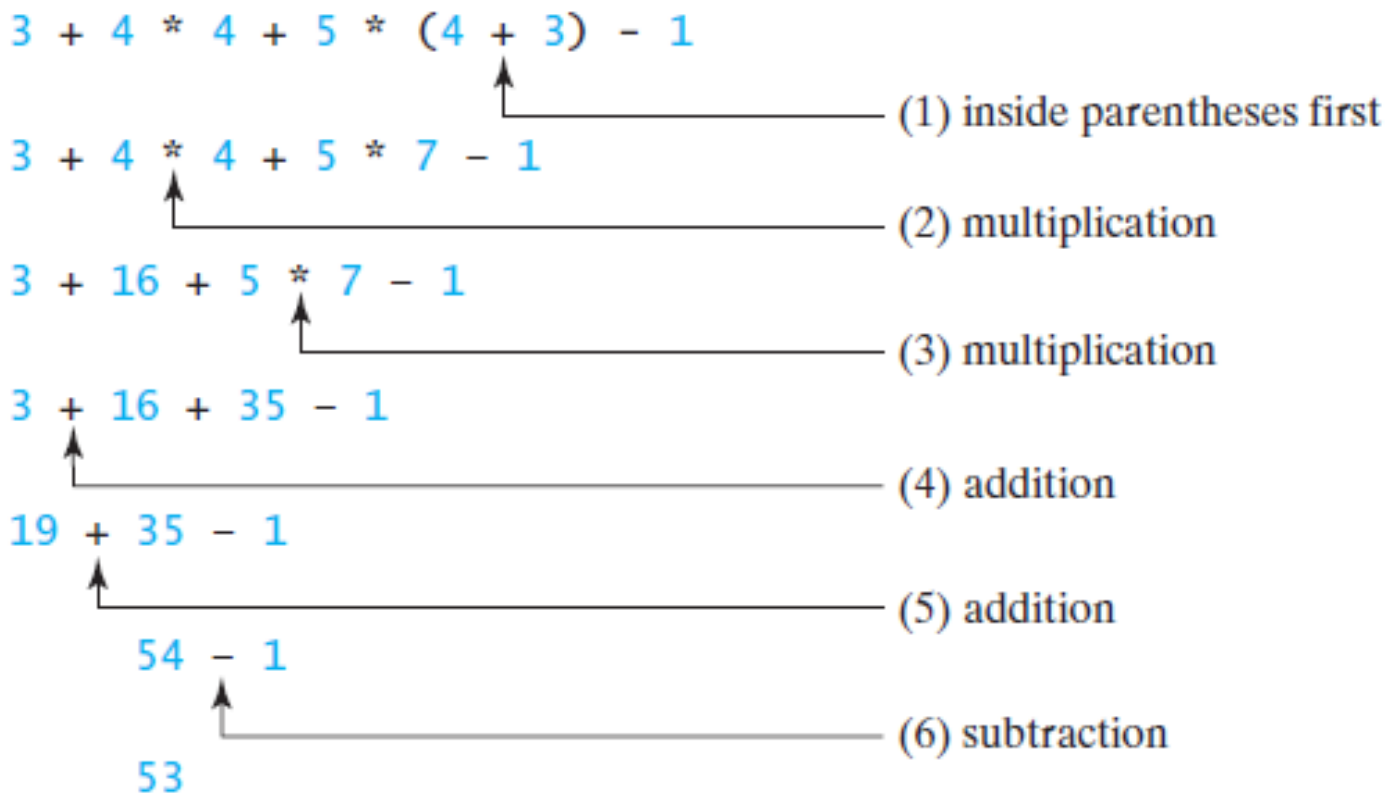
# Numeric Literals - Floats

- A special syntax is used to write scientific notation numbers.
- For example, 1.23456 * $10^2$ is written as 1.23456E2 or 1.23456E+2 and 1.23456 * 10-2 as 1.23456E-2.
- E (or e) represents an exponent and can be in either lowercase or uppercase.

# Evaluating Expressions and Operator Precedence

- Java expressions are evaluated in the same way as arithmetic expressions.
- Multiplication, division, and remainder operators are applied first.
- If an expression contains several multiplication, division, and remainder operators, they are applied from left to right.
- Addition and subtraction operators are applied last.
- If an expression contains several addition and subtraction operators, they are applied from left to right.

# Evaluating Expressions and Operator Precedence



# Augmented Assignment Operators

- Java allows you to combine assignment and operators using an augmented (or compound) assignment operator

**TABLE 2.4** Augmented Assignment Operators

| *Operator* | *Name* | *Example* | *Equivalent* |
|---|---|---|---|
| `+=` | Addition assignment | `i += 8` | `i = i + 8` |
| `-=` | Subtraction assignment | `i -= 8` | `i = i - 8` |
| `*=` | Multiplication assignment | `i *= 8` | `i = i * 8` |
| `/=` | Division assignment | `i /= 8` | `i = i / 8` |
| `%=` | Remainder assignment | `i %= 8` | `i = i % 8` |

# Increment and Decrement Operators

- The increment operator (++) and decrement operator (– –) are for incrementing and decrementing a variable by 1.

**TABLE 2.5** Increment and Decrement Operators

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

# Increment and Decrement Operators

- Example:

```
double x = 1.0;
double y = 5.0;
double z = x–– + (++y);
```

- After all three lines are executed, y becomes 6.0, z becomes 7.0, and x becomes 0.0.

# Numeric Type Conversions

- Floating-point numbers can be converted into integers using explicit casting.
- If an integer and a floating-point number are involved in a binary operation, Java automatically converts the integer to a floating-point value.
- So, 3 * 4.5 is same as 3.0 * 4.5.

# Numeric Type Conversions

- The syntax for casting a type is to specify the target type in parentheses, followed by the variable's name or the value to be cast.
- For example, the following statement

  System.out.println((int) 1.7);

  displays 1.
- When a double value is cast into an int value, the fractional part is truncated.

# Numeric Type Conversions

- The following statement

  System.out.println((double)1 / 2);

  displays 0.5, because 1 is cast to 1.0 first, then 1.0 is divided by 2.

- However, the statement
- System.out.println(1 / 2);

  displays 0, because 1 and 2 are both integers and the resulting value should also be an integer.