Programming Fundamentals I
Java Fundamentals

Nguyen Ngoc Minh

Saigon Institute of Technology
Readings

• Readings for these lecture notes
  • Textbook:
    • Tony Gaddis & Godfrey, *Starting Out with Java From Control Structures through Data Structures*
  • Reference:
    • Cay S. Horstmann and Gary Cornell, *Core Java™*
• These lecture notes contain material © Tony Gaddis, and Godfrey Muganda, 2007
Chapter Objectives

Upon completion of this chapter, you should be able to:

• Understand the structure of a simple Java program
• Comment and create help document
• Use variables and primitive data types
• Distinguish primitive & reference variable
• Recognize Java keywords and declare a valid identifier
• Use operators
• Construct an object using new
• Understand the initialization process
• Create a Java program using standard input and output device
Outline

- Parts of a Java Program
- Standard Java library (Java API)
- Variables, Literals, and Identifiers
- Coding Convention
- Primitive Data Types
- Constants
- Assignment & Arithmetic Operators
- Conversion
- The String class
- Scope
- Comments
- Programming style
- Reading keyboard input
- Dialog boxes
Parts of a Java Program

Example

- Comments:
  - Single-line style: line1
  - Multi-line style: line3-line7
- Class header: line 2
- Curly braces:
  - Open brace { : line2, line8
  - Closing brace } : line10, line12
- Method header: line8
- Statement: line 9

```java
public class Simple {
    /*
     * This is another comment that is written in many lines
     */
    public static void main(String[] args) {
        System.out.println("Programming is great fun!");
    }
}
```
Parts of a Java Program

Example

```java
public class Simple {
    public static void main(String[] args) {
        // body of the main method
    }
}
```

- **public class Simple**
- **public static void main(String[] args)**
- **body of the main method**
- **body of the class Simple**
Parts of a Java Program

Comments

• All comments are ignored by the compiler.
• Two styles of inserting comments:
  • Single line: //</br>  Ex: //This line is comment
  • Multi-lines: start comment with /*
    end comment with */
  Ex: /*comment on one
    or
    many lines
  */
Parts of a Java Program

Class Header

• Mark the beginning of a class definition.

• Standard form:

  `<modifier>*  class  <class_name>`

  Ex:  `public class Simple`

• There’re some possible `<modifier>` keywords, but from now on, use only `public`.

• Term public means access to class is unrestricted
Parts of a Java Program
Curly Braces

- When associated with the class header, they define the scope of the class.
- When associated with a method, they define the scope of the method.

```java
public class Simple {
    public static void main(String[] args) {
    }
}
```

- **Scope of class**
- **Scope of method**
Parts of a Java Program

Main Method

• Main method must be declared exactly as:

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

(except the args variable name can be defined by programmer).

• This is the line of code that the java command will run first.

• This method starts the Java program.

• Every Java `application` must have a main method.
Parts of a Java Program

Java statements

- When the program runs, the statements within the main method will be executed sequentially.
- A statement is a ‘instruction’ you want the program perform.
- A semicolon (;) marks the end of a statement in Java.

Ex: `System.out.println("Programming is great fun!");`

What is the purpose of this statement?
Parts of a Java Program

Important Points

• Java is a case-sensitive language.
• All Java programs must be stored in a file with a .java file extension.
• Comments are ignored by the compiler.
• A .java file may contain many classes but may only have one public class.
• If a .java file has a public class, the class must have the same name as the file.
Parts of a Java Program

Important Points

• Java applications must have a main method.
• For every left brace, or opening brace, there must be a corresponding right brace, or closing brace.
• Statements are terminated with semicolons.
• Comments, class headers, method headers, and braces are not considered Java statements.
Outline

- Parts of a Java Program
- **Standard Java library (Java API)**
- Variables, Literals, and Identifiers
- Coding Convention
- Primitive Data Types
- Constants
- Arithmetic & Assignment Operators
- Conversion
- The String class
- Scope
- Comments
- Programming style
- Reading keyboard input
- Dialog boxes
Standard Java library

• The standard Java library is commonly referred to as the Java API.
• We can use classes built in Java API.
• System class contains methods and objects that display messages on standard output device.
• The out object, a member of the System class, contains the methods print and println.
Standard Java library
print and println methods

• The print and println methods send characters to the output device.
  Ex:  
  ```java
  System.out.println("Programming is great fun!");
  ```

• The string inside the parenthesis will be sent to the output.

• The println method places a newline character at the end of whatever is being printed out.
Standard Java library
print and println methods (Example)

• When execute two statements below:

    System.out.println("This is being printed out");
    System.out.println("on two separate lines.");

• Result on output console:

    This is being printed out
    on two separate lines.
Standard Java library

Exercice

• Problem: Write a program to display this message “Java is a object-oriented programming language” in two ways:
  1. The message is displayed in one line
     Java is a object-oriented programming language
  2. The message is displayed in three lines:
     Java is
     a object-oriented
     programming language
Standard Java library
print and println methods

- Use escape sequences to print special characters.
- An escape sequence starts with the backslash character `\` and is followed by one or more control characters.
- Ex: We can solve the previous problem without using println method.

```java
public class PrintAndPrintln {
    public static void main(String[] args) {
        System.out.print("Java is \n");
        System.out.print("a object-oriented \n");
        System.out.print("programming language");
    }
}
```
**Standard Java library**

**Java Escape Sequences**

<table>
<thead>
<tr>
<th>Escape Sequence</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>\n</code> newline</td>
<td>Advances the cursor to the next line for subsequent printing</td>
</tr>
<tr>
<td><code>\t</code> tab</td>
<td>Causes the cursor to skip over to the next tab stop</td>
</tr>
<tr>
<td><code>\b</code> backspace</td>
<td>Causes the cursor to back up, or move left, one position</td>
</tr>
<tr>
<td><code>\r</code> carriage return</td>
<td>Causes the cursor to go to the beginning of the current line, not the next line</td>
</tr>
<tr>
<td><code>\</code> backslash</td>
<td>Causes a backslash to be printed</td>
</tr>
<tr>
<td><code>'</code> single quote</td>
<td>Causes a single quotation mark to be printed</td>
</tr>
<tr>
<td><code>&quot;</code> double quote</td>
<td>Causes a double quotation mark to be printed</td>
</tr>
</tbody>
</table>
Outline

• Parts of a Java Program
• Standard Java library (Java API)
  • Variables, Literals, and Identifiers
• Primitive Data Types
• Constants
• Assignment & Arithmetic Operators
• Conversion
• The String class
• Scope
• Comments
• Programming style
• Reading keyboard input
• Dialog boxes
Variables, Literals, and Identifiers

Variable vs literal

- A variable is a named storage location in the computer’s memory.
- A literal is a value that is written into the code of a program.
- Ex:

```java
//This program has a variable
public class Variable {
    public static void main(String[] args) {
        int value;
        value = 5;
        System.out.print("The value is ");
        System.out.println(value);
    }
}
```
Variables, Literals, and Constants
Variable vs Literal

This line is called a variable declaration.
int value;

The value 5 is stored in memory.
value = 5;

0x000
0x001
0x002
0x003

This is a String literal. It will be printed as is.
System.out.print("The value is ");
System.out.println(value);

The integer 5 will be printed out here.
Notice no quote marks?
Variables, Literals, and Identifiers

Identifier

- An identifier is a name given to a variable, class, or method.
- Identifiers start with a letter (a–z or A–Z), underscore (_), or dollar sign ($).
- Subsequent characters can be letters, underscores, dollar signs, and digits (0-9).
- Identifiers are case-sensitive and have no maximum length.
- Identifiers may not be any of the Java reserved keywords.
## Variables, Literals, and Identifiers

### Java keywords

<table>
<thead>
<tr>
<th>Abstract</th>
<th>Else</th>
<th>Long</th>
<th>Synchronized</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>Extends</td>
<td>Native</td>
<td>This</td>
</tr>
<tr>
<td>Break</td>
<td>Final</td>
<td>New</td>
<td>This</td>
</tr>
<tr>
<td>Byte</td>
<td>Finally</td>
<td>Package</td>
<td>Throw</td>
</tr>
<tr>
<td>Case</td>
<td>Float</td>
<td>Private</td>
<td>Throws</td>
</tr>
<tr>
<td>Catch</td>
<td>For</td>
<td>Protected</td>
<td>Transient</td>
</tr>
<tr>
<td>Char</td>
<td>Goto</td>
<td>Public</td>
<td>Try</td>
</tr>
<tr>
<td>Class</td>
<td>If</td>
<td>Return</td>
<td>Void</td>
</tr>
<tr>
<td>Const</td>
<td>Implements</td>
<td>Short</td>
<td>Void</td>
</tr>
<tr>
<td>Continue</td>
<td>Import</td>
<td>Static</td>
<td>Volatile</td>
</tr>
<tr>
<td>Default</td>
<td>Instanceof</td>
<td>Strictfp</td>
<td>While</td>
</tr>
<tr>
<td>Do</td>
<td>Int</td>
<td>Super</td>
<td></td>
</tr>
<tr>
<td>Double</td>
<td>Interface</td>
<td>Switch</td>
<td></td>
</tr>
</tbody>
</table>
Variables, Literals, and Identifiers
Java Reserved words for literal values

- false
- null
- true
Variables, Literals, and Identifiers
Identifier (Example)

- Which identifiers are valid?
  - UserName
  - myLessons
  - Programming languages
  - _is_valid
  - $EmployeeSalary
  - 12Months
  - Who’sThis?
  - _student1
  - interface
  - int
  - newName
  - New
  - True
  - false
  - nullValue
Variables, Literals, and Identifiers

Naming Convention

• Names should be descriptive more readable, more maintainable.

• Variable names should begin with a lower case letter and Internal words start with capital letters:
  Ex: int taxRate

• Class names should be all title case.
  Ex: public class BigLittle

• More Java naming conventions can be found at:

• Variable names and class names are usually nouns or noun phrases.
Outline

- Parts of a Java Program
- Standard Java library (Java API)
- Variables, Literals, and Identifiers
  - **Primitive Data Types**
- Constants
- Assignment & Arithmetic Operators
- Conversion
- The String class
- Scope
- Comments
- Programming style
- Reading keyboard input
- Dialog boxes
Primitive Data Types

- Java has many built in data types.
- There’s two categories of data types: class types and primitive types
- Primitive types are simple values, are not objects
- Java has 8 primitive data types divided into 4 categories:
  - Logical - boolean
  - Textual – char
  - Integral – byte, short, int, and long
  - Floating point – double and float
Primitive Data Types

Variable Declarations

• Variable Declarations take the following form:

  *DataType VariableName*

Ex:

  • byte inches;
  • short month;
  • int speed;
  • long timeStamp;
  • float salesCommission;
  • double distance;
Primitive Data Types
Logical - boolean

- Variable of the boolean type takes one of two literal values: true or false.
- This data type is useful for represent any two states, such as true and false, on and off, or yes and no.

```java
// A program for demonstrating boolean variables
public class TrueFalse {
    public static void main(String[] args) {
        boolean bool;

        bool = true;
        System.out.println(bool);
        bool = false;
        System.out.println(bool);
    }
}
```
Primitive Data Types
Textual - char

- Java char type is used to represent single characters.
- char literals are enclosed in single quote marks ‘’
- Internally, characters are stored as numbers.
- Character data in Java is stored as Unicode characters.
- Each Unicode character takes up 2 bytes in memory.
- The first 256 characters in the Unicode character set are compatible with the ASCII* character set.
### Primitive Data Types

**Textual – char (Example)**

```java
class Letter {
    public static void main(String[] args) {
        char letter;
        // Enclose a char literal in single quotes
        letter = 'A';
        System.out.println(letter);
        // Use code of letter A in Decimal numeral system
        letter = 65;
        System.out.println(letter);
        // Use code of letter A in Hexadecimal numeral system
        letter = '\u0041';
        System.out.println(letter);
    }
}
```
Primitive Data Types

Unicode

Characters are stored in memory as binary numbers.

A

00 65

0000000000010000001

B

00 66

0000000000010000011
Primitive Data Types

Unicode

The binary numbers represent these decimal values.

A

00 65

B

00 66

The binary numbers

00000000001000001

00000000001000011
Primitive Data Types

Unicode

The decimal values represent these characters.
Primitive Data Types
Integral – byte, short, int, and long

- Integral datatype can hold whole numbers such as 5, 10, 23, etc.
- Java has four integral types: byte, short, int, long
- Integer data types cannot hold numbers that have a decimal point in them.
# Primitive Data Types

## Integral – byte, short, int, and long

<table>
<thead>
<tr>
<th>Integer Length</th>
<th>Name or Type</th>
<th>Range</th>
</tr>
</thead>
</table>
| 8 bits         | byte        | From $-2^7$ to $2^7 - 1$  
|                |             | (-128 to +127)        |
| 16 bits        | short       | From $-2^{15}$ to $2^{15} - 1$  
|                |             | (-32,768 to +32,767)   |
| 32 bits        | int         | From $-2^{31}$ to $2^{31} - 1$  
|                |             | (-2,147,483,648 to +2,147,483,647) |
| 64 bits        | long        | From $-2^{63}$ to $2^{63} - 1$  
|                |             | (-9,223,372,036,854,775,808 to +9,223,372,036,854,775,807) |
Primitive Data Types
Integral – byte, short, int, and long

- Literals of integral type can be represented using:
  - Decimal form: 15
  - Octal form: 017
  - Hexadecimal form: 0x000F

- A variable cannot receive value out of its data type range
  Ex: This declaration is not correct
  byte byteNumber = 128
Primitive Data Types
Numerical Bases - Deximal

- Deximal numbers use 10 different digits from 0 to 9

\[
\begin{array}{cccccc}
1 & 8 & 2 & 7 & 3 & 6 \\
10^5 & 10^4 & 10^3 & 10^2 & 10^1 & 10^0 \\
6 \times 10^0 &=& 6   \\
3 \times 10^1 &=& 30  \\
7 \times 10^2 &=& 700 \\
2 \times 10^3 &=& 2000 \\
8 \times 10^4 &=& 80000 \\
1 \times 10^5 &=& 100000 \\
\hline
& & & & & 182736
\end{array}
\]
• Octal numbers use 8 different digits from 0 to 7

<table>
<thead>
<tr>
<th>7</th>
<th>1</th>
<th>2</th>
<th>6</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$8^4$</td>
<td>$8^3$</td>
<td>$8^2$</td>
<td>$8^1$</td>
<td>$8^0$</td>
</tr>
</tbody>
</table>

Decimal:

- $3 \times 8^0 = 3$
- $6 \times 8^1 = 48$
- $2 \times 8^2 = 128$
- $1 \times 8^3 = 512$
- $7 \times 8^4 = 28672$

$29363$
Primitive Data Types
Numerical Bases - Hexadecimal

- Hexadecimal numbers use 16 different digits represented by the numbers from 0 to 9 and the letters A, B, C, D, E and F

<table>
<thead>
<tr>
<th>A</th>
<th>2</th>
<th>F</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>16³</td>
<td>16²</td>
<td>16¹</td>
<td>16⁰</td>
</tr>
</tbody>
</table>

decimal:

\[
egin{align*}
7 \times 16^0 &= 7 \\
15 \times 16^1 &= 240 \\
2 \times 16^2 &= 512 \\
10 \times 16^3 &= 40960 \\
\hline
\text{41719}
\end{align*}
\]
Primitive Data Types

Integral - Example

```java
public class IntegralExample {
    public static void main(String[] args) {
        // declare byte variable
        byte byteNum1;
        // declare and assign byte
        byte byteNum2 = -128;
        // declare and assign short
        short shortNum = 32767;
        // declare integer variable;
        int intNum1, intNum2;
        // assign values to int variable
        intNum1 = 32768;
        intNum2 = 2147483647;
    }
}
```
Primitive Data Types
Floating Point

• A numeric literal is a floating point if:
  • includes either a decimal point
    Ex: 2.15
  • or an exponent part (the letter E or e)
    Ex: 6.12E2, 6.5E-1
  • or is followed by the letter F or f (float)
    Ex: 2.5f
  • or the letter D or d (double).
    Ex: 12d
Primitive Data Types
Floating Point

• In Java there are two data types that can represent floating-point numbers.
  • float - also called *single precision* (7 decimal points).
  • double - also called *double precision* (15 decimal points).

• The default type for floating point literals is double.
  Ex: this declaration is not correct
  float taxRate = 0.15
Primitive Data Types
Floating Point

- Floating-point literals can be represented in scientific notation.
  - 47,281.97 == 4.728197 x 10^4.
- Java uses E notation to represent values in scientific notation.
  - 2.5x10^-2 == 2.5E-2
Primitive Data Types

Floating Point

```java
public class FloatingPointExample {

    public static void main(String[] args) {
        // declare and assign floating point
        float fNum = 34.4f;
        // declare and assign double
        double dNum = 1.5;
        // assign value to double variable
        dNum = 12.7d;
        System.out.println(dNum);
    }

    }
```
Outline

• Parts of a Java Program
• Standard Java library (Java API)
• Variables, Literals, and Identifiers
• Primitive Data Types
• Constants
• Assignment & Arithmetic Operators
• Conversion
• The String class
• Scope
• Comments
• Programming style
• Reading keyboard input
• Dialog boxes
Constant

- The `final` key word can be used in a variable declaration to make a variable a named constant.
  
  Ex: `final int NUM_OF_Month = 12`
  
  `final double PI = 3.14`

- When a constant is assigned/initialized with a value, that value cannot change.

- Constant makes the program easy to read and maintain.

- There’re lots of pre-defined constants in Java API.
  
  Ex: `Math.PI`
Outline

• Parts of a Java Program
• Standard Java library (Java API)
• Variables, Literals, and Identifiers
• Primitive Data Types
• Constants
• Assignment & Arithmetic Operators
• Conversion
• The String class
• Scope
• Comments
• Programming style
• Reading keyboard input
• Dialog boxes
Assignment & Arithmetic Operators

In order to store a value in a variable, the assignment statement must be used.

The assignment operator is the equal (\( = \)) sign.

The operand on the left side of the assignment operator must be a variable name.

The operand on the right side must be either a literal or expression that evaluates to a type that is compatible with the type of the variable.
Assignment & Arithmetic Operators

assignment operator (Example)

How to correct one bug in this example?

```
public class Assignment {

    public static void main(String[] args) {
        // declare variables
        int numOfStudents;
        int numOfScholarship;
        int totalFinancialAid;

        /*
         * declare float constant to hold the money value for
         * each scholarship
         */
        final float scholarshipValue = 2000;
        // assign int literal to variable
        numOfStudents = 100;
        numOfScholarship = 5;
        // assign an expression to int variable
        totalFinancialAid = numOfScholarship * scholarshipValue;
        System.out.print("The total amount of financial aid is:");
        System.out.print(totalFinancialAid);
    }
}
```
Assignment & Arithmetic Operators

Initialization

- The variables must be declared and initialized before used.

Ex: This program has one bug.

```java
public class Assignment {

    public static void main(String[] args) {
        // declare variables
        int numOfStudents;
        int numOfScholarship;
        float totalFinancialAid;
        /*
         * declare float constant to hold the money value for each scholarship
        */

        final float scholarshipValue;
        // assign int literal to variable
        numOfStudents = 100;
        numOfScholarship = 5;
        // assign an expression to int variable
        totalFinancialAid = numOfScholarship * scholarshipValue;
        System.out.print("The total amount of financial aid is:");
        System.out.print(totalFinancialAid);
    }
}
```
Arithmetic Operators

Arithmetic operations are used for manipulating numeric values.

Three types of operators: unary, binary, ternary

Unary operator: require 1 operand.
Ex: -5

Binary operator: require 2 operands.
Ex: 3+2

Ternary operator: require 3 operands.
## Arithmetic Operators

**binary arithmetic operators**

- Java has 5 arithmetic operators.
- Each operator must have a left and right operator.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
<td>Binary</td>
<td>total = cost + tax;</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td>Binary</td>
<td>cost = total – tax;</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>Binary</td>
<td>tax = cost * rate;</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
<td>Binary</td>
<td>salePrice = original / 2;</td>
</tr>
<tr>
<td>%</td>
<td>Modulus</td>
<td>Binary</td>
<td>remainder = value % 5;</td>
</tr>
</tbody>
</table>
Arithmetic Operators

binary arithmetic operators (Example)

```java
public class Wages {
    public static void main(String[] args) {
        double regularWages;  // The calculated regular wages
        double basePay = 25;  // the base pay rate
        double regularHours = 40;  // The hours worked less overtime
        double overtimeWages;  // Overtime wages
        double overtimePay = 37.5;  // Overtime pay rate
        double overtimeHours = 10;  // Overtime hours worked
        double totalWages;  // Total wages

        regularWages = basePay * regularHours;
        overtimeWages = overtimePay * overtimeHours;
        totalWages = regularWages + overtimeWages;

        System.out.println("Wages for this week are $" + totalWages);
    }
}```
### Arithmetic Operators

**Operator Precedence**

- Mathematical expressions can be very complex.
- There is a set order in which arithmetic operations will be carried out.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Associativity</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Higher Priority</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- (unary negation)</td>
<td>Right to left</td>
<td>(x = -4 + 3;)</td>
<td>-1</td>
</tr>
<tr>
<td>* / %</td>
<td>Left to right</td>
<td>(x = -4 + 4 % 3 \times 13 + 2;)</td>
<td>11</td>
</tr>
<tr>
<td>Lower Priority</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+ -</td>
<td>Left to right</td>
<td>(x = 6 + 3 - 4 + 6 \times 3;)</td>
<td>23</td>
</tr>
</tbody>
</table>
Arithmetic Operators

Grouping with Parenthesis

- When parenthesis are used in an expression, the inner most parenthesis are processed first.
- If two sets of parenthesis are at the same level, they are processed left to right.

\[
x = \left( \left( 4 \times 5 \right) / \left( 5 - 2 \right) \right) - 25; \quad // \text{result} = -19
\]
## Combined Assignment Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example</th>
<th>Equivalent</th>
<th>Value of variable after operation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>+=</code></td>
<td><code>x += 5;</code></td>
<td><code>x = x + 5;</code></td>
<td>The old value of <code>x</code> plus 5.</td>
</tr>
<tr>
<td><code>-=</code></td>
<td><code>y -= 2;</code></td>
<td><code>y = y - 2;</code></td>
<td>The old value of <code>y</code> minus 2</td>
</tr>
<tr>
<td><code>*=</code></td>
<td><code>z *= 10;</code></td>
<td><code>z = z * 10;</code></td>
<td>The old value of <code>z</code> times 10</td>
</tr>
<tr>
<td><code>/=</code></td>
<td><code>a /= b;</code></td>
<td><code>a = a / b;</code></td>
<td>The old value of <code>a</code> divided by <code>b</code>.</td>
</tr>
<tr>
<td><code>%=</code></td>
<td><code>c %= 3;</code></td>
<td><code>c = c % 3;</code></td>
<td>The remainder of the division of the old value of <code>c</code> divided by 3.</td>
</tr>
</tbody>
</table>
Outline

- Parts of a Java Program
- Standard Java library (Java API)
- Variables, Literals, and Identifiers
- Coding Convention
- Primitive Data Types
- Constants
- Assignment & Arithmetic Operators
  - Conversion
- The String class
- Scope
- Comments
- Programming style
- Reading keyboard input
- Dialog boxes
Conversion

- When the value’s data type is not compatible with the variable’s data type, there should be a conversion.
- Two types of conversion: **widening** and **narrowing**
- Java automatically perform widening conversion when necessary.
- The programmer must perform narrowing conversion as need.
Conversion
Primitive data type ranking

- double: Highest Rank
- float
- long
- int
- short: Lowest Rank
- byte
Conversion
Widening (Promotion)

- Widening conversion means converting the lower-ranked type to the higher-ranked type.
  Ex: long longValue;
      int intValue = 10;
      longValue = intValue;
- Widening conversion doesn’t cause a loss of data
Conversion
Narrowing (Promotion)

• Narrowing conversion means converting the higher-ranked type to the lower-ranked type.
• Narrowing conversion may cause a loss of data.
• If the programmer doesn’t manually perform narrowing conversion in assignment statements → compile-time error.
• The cast operator can be used to perform converting.
Conversion
Cast operator

- Cast operator is unary operator
- To convert to a specific data type, use this syntax:
  \[(\text{data\_type}) \text{ value}\]
  
  Ex:  \[\text{float floatNum} = (\text{float})3.5;\]
       \[\text{float floatNum} = (\text{float})(3.5*2+1);\]
- The operator precedes the value being converted.
Conversion

some rules

• When values of the byte or short data type are used in arithmetic expression, they are temporarily converted to \texttt{int} values.

```java
public class Conversion {
    public static void main(String[] args) {
        short a, b;
        a = 1;
        b = a + 1;
    }
}
```
Conversion

some rules

• When 2 operands of a binary operator are primitive numeric types, the result type is the highest-ranked type of either operand, or int

```java
public class Conversion {
    public static void main(String[] args) {
        short a;
        int b;
        long c = 2;
        a = 1;
        b = a + c;
    }
}
```
Conversion
Example (Narrowing, lost data)

```java
public class NarrowingConversion {
    public static void main(String[] args) {
        int a;
        long b;
        float c;
        b = 5;
        a = (int)b; // it's ok because 5 in range of int type
        c = 3.14f;
        b = (long)c; // lost data
        System.out.println("a="+a);
        System.out.println("b="+b);
        System.out.println("c="+c);
    }
}
```
Outline

- Parts of a Java Program
- Standard Java library (Java API)
- Variables, Literals, and Identifiers
- Coding Convention
- Primitive Data Types
- Constants
- Assignment & Arithmetic Operators
- Conversion
- The String class
- Scope
- Comments
- Programming style
- Reading keyboard input
- Dialog boxes
String Class

• This is one of class types built-in Java.
• This class is used to hold & manipulate a series of characters.
• Syntax to declare a String variable: `String var_name;`
• Syntax to create a String object:
  ```java
  var_name = "string literals" ;
  var_nam = new String("string literals");
  ```
String Class
Example

```java
public class StringDemo {
    public static void main(String[] args) {
        // declare and initialize a String variable
        String question = "Have you ever heard about this logan? ";
        // declare a String variable
        String javaSlogan;
        // Assign String literals to a String variable
        javaSlogan = new String("Write once run any where");
        // concatenate Strings and print to output console
        System.out.println(question + "\n" + javaSlogan);
    }
}
```
String Class

methods

• The String class contains many methods used to manipulation of String objects.

Ex: toUpperCase(), length(), charAt()

• String objects are immutable, meaning that they cannot be changed.
String Class

methods (Example)

```java
public class StringMethods {

    public static void main(String[] args) {
        String message = "Java is Great Fun!";
        String upper = message.toUpperCase();
        String lower = message.toLowerCase();
        char letter = message.charAt(0);
        int stringSize = message.length();

        System.out.println(message);
        System.out.println(upper);
        System.out.println(lower);
        System.out.println(letter);
        System.out.println(stringSize);
    }
}
```
The + Operator

- The + operator can be used in two ways:
  - as a concatenation operator
  - as an addition operator
- If either side of the + operator is a String, the result will be a String.
- Ex:
  ```java
  System.out.println("Hello " + "World");
  System.out.println("The value is: " + 5);
  System.out.println("The value is: " + value);
  System.out.println("The value is: " + '/n' + 5);
  ```
String Concatenation

- A String literal value cannot span lines in a Java source code file.
- The String concatenation operator can be used to fix this problem.
- String concatenation can join various data types.
- The Concatenation operator can be used to format complex String objects.

```java
System.out.println("The following will be printed in a tabbed format:");

  First = 5 * 6 + ","
  Second = (6 + 4) + "",
  Third = 16.7 + ");
```
String Concatenation

Example

```java
public class StringConcatenation {

    public static void main(String[] args) {
        // Concatenate String on multiple lines
        System.out.println("The String concatenation operator" +
                           "can be used to" +
                           "concat String on multiple lines");

        // String concatenation can join various data types.
        System.out.println("We can join a string to " +
                           "a number like this: " + 5);

        // The Concatenation operator can be used to format
        // complex String objects.
       浍/ System.out.println("The following will be printed "
                                + "in a tabbed format: "
                                + "\n\tFirst = " + 5 * 6 + ","
                                + "\n\tSecond = " + (6 + 4) + ","
                                + "\n\tThird = " + 16.7 + ".");
    }
}
```
Primitive vs. Reference Variables

- Variables that are associated with objects only contain the memory address where the object is located.
- This is called a *reference* or *pointer*.

```java
String cityName = "Charleston"
```

The object that contains the character string “Charleston”
Outline

- Parts of a Java Program
- Standard Java library (Java API)
- Variables, Literals, and Identifiers
- Coding Convention
- Primitive Data Types
- Constants
- Assignment & Arithmetic Operators
- Conversion
- The String class
- Scope
- Comments
- Coding Convention
- Programming style
- Reading keyboard input
- Dialog boxes
Scope

• A variable’s **scope** is the part of the program that has access to a variable.

• Variables declared inside a method (like the main method) are called local variables.

• Local variables’ scope begins at the declaration of the variable and ends at the end of the method in which it was declared.
Scope

Example

```java
public class Scope {
    public static void main(String[] args) {
        System.out.println(value);
        int value;
        value = 5;
    }
}
```
Outline

- Parts of a Java Program
- Standard Java library (Java API)
- Variables, Literals, and Identifiers
- Coding Convention
- Primitive Data Types
- Constants
- Assignment & Arithmetic Operators
- Conversion
- The String class
- Scope
- Comments
- Programming style
- Reading keyboard input
- Dialog boxes
Comment

• **Java provides three methods for commenting code.**

<table>
<thead>
<tr>
<th>Comment Style</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>//</td>
<td>Single line comment. Anything after the // on the line will be ignored by the compiler.</td>
</tr>
<tr>
<td>/* ... */</td>
<td>Block comment. Everything beginning with /* and ending with the first */ will be ignored by the compiler. This comment type cannot be nested.</td>
</tr>
<tr>
<td>/** ... */</td>
<td>Javadoc comment. This is a special version of the previous block comment that allows comments to be documented by the javadoc utility program. Everything beginning with the /** and ending with the first */ will be ignored by the compiler. This comment type cannot be nested.</td>
</tr>
</tbody>
</table>
Comment

Example

```java
/**
 * This class creates a program that calculates company payroll
 */

public class CommentJavaDoc {
    /**
     * The main method is the program's starting point.
     */

    public static void main(String[] args) {
        double payRate; // Holds the hourly pay rate
        double hours; // Holds the hours worked;
        int employeeNumber; // Holds the employee number;

        // The remainder of this program is Omitted.
    }
}
```
Comment

Javadoc

• Javadoc comments can be built into HTML documentation.

• To create the documentation:
  • Run the javadoc program with the source file as an argument
  • Ex: javadoc CommentJavaDoc.java

• The javadoc program will create index.html and several other documentation files in the same directory as the input file.
Comment

Javadoc

Class CommentJavaDoc

java.lang.Object
   CommentJavaDoc

public class CommentJavaDoc
extends java.lang.Object

This class creates a program that calculates company payroll

Constructor Summary

CommentJavaDoc()  

Method Summary

static void main(java.lang.String[] args)
   The main method is the program's starting point.
Outline

• Parts of a Java Program
• Standard Java library (Java API)
• Variables, Literals, and Identifiers
• Coding Convention
• Primitive Data Types
• Constants
• Assignment & Arithmetic Operators
• Conversion
• The String class
• Scope
• Comments
• Programming style
• Reading keyboard input
• Dialog boxes
Programming style

• Use spaces, indentations, blank lines, and punctuation characters to write a program’s source code in understandable style.
• Whitespace characters are ignored by the compiler.
• Ex:

```java
public class Compact {
    public static void main(String[] args) {
        int shares = 220;
        double averagePrice = 14.67;
        System.out.println("There were "+ shares + " shares sold at "+ averagePrice + " per share.");
    }
}
```
Programming style

Example

```java
public class Readable {
    public static void main(String[] args) {
        int shares = 220;
        double averagePrice = 14.67;

        System.out.println("There were " + shares + ", shares sold at $" +
                          averagePrice + " per share.");
    }
}
```
Outline

- Parts of a Java Program
- Standard Java library (Java API)
- Variables, Literals, and Identifiers
- Coding Convention
- Primitive Data Types
- Constants
- Assignment & Arithmetic Operators
- Conversion
- The String class
- Scope
- Comments
- Programming style
- **Reading keyboard input**
- Dialog boxes
Reading keyboard input
The Scanner Class

- Java was designed primarily to receive input from a graphical user interface (GUI).
- Getting information from the keyboard in a console application is not convenient.
- We can use the Scanner class to simplify standard input.
- Ex: Payroll.java
  
  InputProblem.java
Reading keyboard input
The Scanner Class (con’t)

- The Scanner class is defined in java.util, so we import java.util.Scanner;
- Scanner objects work with System.in
- To create a Scanner object
  - Scanner keyboard = new Scanner (System.in)
- Scanner class methods are listed in Table 2-18 in the text.
Outline

• Parts of a Java Program
• Standard Java library (Java API)
• Variables, Literals, and Identifiers
• Coding Convention
• Primitive Data Types
• Constants
• Assignment & Arithmetic Operators
• Conversion
• The String class
• Scope
• Comments
• Programming style
• Reading keyboard input
  • Dialog boxes
Dialog boxes

• A dialog box is a small graphical window used to display message or request the user’s input.

• We can use `JOptionPane` class in Java API to create different kinds of dialog boxes.

• **Message Dialog**: display a message and an OK button

• **Input Dialog**: prompts the user for input and provides a text field, an OK button, and a Cancel button

• We must import class `JOptionPane` before using. Ex: `import javax.swing.JOptionPane;`
import javax.swing.JOptionPane;

public class NamesDialog {
    /**
     * This program demonstrates using dialogs with JOptionPane.
     */
    public static void main(String[] args) {
        String firstName;  // The user's first name
        String middleName;  // The user's middle name
        String lastName;   // The user's last name

        // Using Input Dialog
        firstName = JOptionPane.showInputDialog("What is your first name?");
        middleName = JOptionPane.showInputDialog("What is your middle name?");
        lastName = JOptionPane.showInputDialog("What is your last name?");

        // Using Message Dialog
        JOptionPane.showMessageDialog(null, "Hello " + lastName + " " +
                                          middleName + " " +
                                          firstName);
        System.exit(0);
    }
}
Converting String to Numbers

• The showInputDialog returns the string
• If we want to use this input value in a math operation, we must convert it to a number.
• There’re different methods to convert a string to a specific numeric data type.
# Converting String to Numbers (con’t)

<table>
<thead>
<tr>
<th>Method</th>
<th>Use this method to …</th>
<th>Example Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte.parseByte</td>
<td>Convert a string to a byte</td>
<td>Byte num;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>num = Byte.parseByte(str)</td>
</tr>
<tr>
<td>Double.parseDouble</td>
<td>Convert a string to a double</td>
<td>double num;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>num = Double.parseDouble(str)</td>
</tr>
<tr>
<td>Float.parseFloat</td>
<td>Convert a string to a float</td>
<td>float num;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>num = Float.parseFloat(str)</td>
</tr>
<tr>
<td>Integer.parseInt</td>
<td>Convert a string to a int</td>
<td>int num;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>num = Integer.parseInt(str)</td>
</tr>
<tr>
<td>Long.parseLong</td>
<td>Convert a string to a long</td>
<td>long num;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>num = Long.parseLong(str)</td>
</tr>
<tr>
<td>Short.parseShort</td>
<td>Convert a string to a short</td>
<td>short num;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>num = Short.parseShort(str)</td>
</tr>
</tbody>
</table>
Converting String to Numbers

Example

```java
import javax.swing.JOptionPane;

public class PayrollDialog {

    public static void main(String[] args) {
        String inputString; // For reading input
        String name;        // To hold a name
        int hours;          // Hours worked
        double payRate;     // Hourly pay rate
        double grossPay;    // Gross pay

        // Get the user's name.
        name = JOptionPane.showInputDialog("What is your name? ");
        // Get the number of hours worked this week.
        inputString = JOptionPane.showInputDialog("How many hours did you work this week? ");
        hours = Integer.parseInt(inputString);
        // Get the user's hourly pay rate
        inputString = JOptionPane.showInputDialog("What is your hourly pay rate? ");
        payRate = Double.parseDouble(inputString);
        // Calculate the gross pay.
        grossPay = hours * payRate;
        // Display the resulting information.
        System.out.println("Hello " + name);
        System.out.println("Your gross pay is $" + grossPay);
        System.exit(0);
    }
}
```