# Introduction to Computer Programming

CS102A Lecture 2

James YU
yujq3@sustech.edu.cn

Department of Computer Science and Engineering
Southern University of Science and Technology

Sept. 14, 2020


南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

# Objectives

- Write simple Java applications.
- Use input and output statements.
- Java's primitive types.
- Basic memory concepts.
- Use arithmetic operators.
- The precedence of arithmetic operators.
- Write decision-making statements.
- Use relational and equality operators.

# Introduction

- Java application
  - A computer program that executes when you use the java command to launch the JVM.
- Use tools from the JDK to compile and run programs.

# Our first program in Java

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

```java
// Text printing program

public class Welcome1
{
  // main method begins execution of Java application
  public static void main (String[] args)
  {
    System.out.println("Welcome to Java Programming!");
  } // end method main
} // end class Welcome1
```

# Our first program in Java

- Comments:

```
1 // Text printing program
```

  - `//` indicates that the line is a comment.
  - Used to document programs and improve their readability.
  - Compiler ignores comments.

- Traditional comment, can be spread over several lines:

```
1 /* This is a traditional comment. It
2    can be split over multiple lines */
```

  - This type of comment begins with `/*` and ends with `*/`.

# Traditional vs. end-of-line comments

- Traditional comments do not nest, the first `*/` after the first `/*` will terminate the comment.

```
/*
  /* comment 1 */
     comment 2 */
```

- End-of-line comments can contain anything.

```
// /* This comment is okay */
```

# Our first program in Java

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

- Class declaration:

```
1 public class Welcome1
```

  - Every Java program consists of at least one class that you define.
  - `class` keyword introduces a class declaration and is immediately followed by the *class name*.
  - *Keyword*s are reserved for use by Java and are always **spelled with all lowercase letters**.

# Our first program in Java

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

- Class names:

```
1 public class Welcome1
```

  - By convention, begin with a capital letter and capitalize the first letter of each word they include.
  - **Java is case sensitive** — uppercase and lowercase letters are distinct (not in comments).

# Our first program in Java

- The braces:

```
{
  // Some code
}
```

  - A left brace, , begins the body of every class declaration.
  - A corresponding right brace, , must end each class declaration.
  - Code between braces should be indented (good practice).

# Our first program in Java

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

- Declaring the main method:

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

  - Starting point of Java applications.
  - Parentheses after the identifier main indicate that it's a program building block called a method.
  - Java class declarations normally contain one or more methods.
  - Keyword `void` indicates that this method will not return any information.

# Our first program in Java

- Body of the method declaration:

```
{
  System.out.println("Welcome to Java Programming!");
}
```

  - Enclosed in left and right braces.

- Statement:

```
System.out.println("Welcome to Java Programming!");
```

  - Instructs the computer to perform an action.
  - Print the string of characters contained between the double quotation marks.

# Our first program in Java

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

```
1 System.out.println("Welcome to Java Programming!");
```

- `System.out` object:
  - Standard output object, allows Java applications to display strings in the *command window* from which the Java application executes.
- `System.out.println` method:
  - Displays (or prints) a line of text in the command window.
  - The string in the parentheses is the *argument* to the method.
  - **Positions the output cursor at the beginning of the next line in the command window**.

# Our first program in Java

- Compiling and executing your first Java application
  - To compile the program, type `javac Welcome1.java`.
  - If the program contains no syntax errors, preceding command creates a `Welcome1.class` file (known as the class file) containing the platform-independent Java bytecodes that represent the application.
- To execute the program, type `java Welcome1`.
  - Launches the JVM, which loads the `.class` file for class `Welcome1`.
  - Note that the `.class` file-name extension **is omitted from the preceding command**; otherwise, the JVM will not execute the program.

```
> javac Welcome1.java
> java Welcome1
Welcome to Java Programming!
>
```

# Modify our first Java program

```java
// Text printing program

public class Welcome2 {
  // main method begins execution of Java application
  public static void main (String[] args) {
    System.out.print("Welcome to ");
    System.out.println("Java Programming!");
  } // end method main
} // end class Welcome2
```

# Modify our first Java program

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

- Class `Welcome2` uses two statements to produce the same output as class `Welcome1`:

```
1 System.out.print("Welcome to ");
2 System.out.println("Java Programming!");
```

- `System.out`'s method `print` displays a string.
- Unlike `println`, **`print` does not position the output cursor at the beginning** of the next line in the command window.

```
> javac Welcome2.java
> java Welcome2
Welcome to Java Programming!
>
```

# Modify our first Java program

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

```java
// Text printing program

public class Welcome3 {
  // main method begins execution of Java application
  public static void main (String[] args) {
    System.out.println("Welcome\nto\nJava\nProgramming!");
  } // end method main
} // end class Welcome3
```

```
> javac Welcome3.java
> java Welcome3
Welcome
to
Java
Programming!
>
```

# Modify our first Java program

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

```
1 System.out.println("Welcome\nto\nJava\nProgramming!");
```

- Newline character `\n` tells the `print` methods when to position the output cursor at the beginning of the next line in the command window.
- Newline characters are *white-space character*s.
  - White-space characters do not correspond to a visible mark, but represent horizontal or vertical space in typography.
- The backslash `\` is called an *escape character*, and **forms an escape sequence together with its next character**.
  - Invokes an alternative interpretation on subsequent character.

# Common escape sequences

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

| Escape sequence | Description |
|---|---|
| `\n` | Newline. Position the cursor at the beginning of the next line. |
| `\t` | Horizontal tab. Move the cursor to the next tab stop. |
| `\r` | Carriage return. Position the cursor at the beginning of the current line (do not advance to the next line). Any characters output after the carriage return overwrite the characters previously output on that line. |
| `\\` | Use to print a backslash character. |
| `\"` | Used to print a double-quote character ". `System.out.println("\"in quotes\"");` displays `"in quotes"`. |

# Displaying text with `printf`

```java
// Text printing program

public class Welcome4 {
  // main method begins execution of Java application
  public static void main (String[] args) {
    System.out.printf("%s%n%s\n", "Welcome to", "Java Programming!");
  } // end method main
} // end class Welcome4
```

# Displaying text with `printf`

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

```
1 System.out.printf("%s%n%s\n", "Welcome to", "Java Programming!");
```

- The method `printf` displays “formatted” data.
- It takes a format string as an argument.
- Format specifiers begin with a percent sign (%) and are followed by a character that represents the data type.
  - Format specifier %s is a placeholder for a string.
  - Format specifier %n is a placeholder for a newline.

# Displaying text with `printf`

```
> javac Welcome4.java
> java Welcome4
Welcome to
Java Programming!
>
```

# General vs. specific

- Sometimes we want to write a function to serve the majority, with a very general purpose.
  - Consider buying a standard hamburger set in McDonald's.
- "One Size Can't Fit All"
  - We always have a lot of specific purposes.
  - What if I want more ketchup in my hamburger, more ice and less sugar in my drink?
- `print()` vs. `printf()`

# Primitive data types: Integers

| Type | Size | Range |
|---|---|---|
| `byte` | 8 bits | $-128$ to $+127$ |
| `short` | 16 bits | $-32768$ to $+32767$ |
| `int` | 32 bits | (about) negative two billion to positive two billion |
| `long` | 64 bits | (about) $-10^{38}$ to $+10^{38}$ |

```
1 int year = 2019;
```

# Primitive data types: Floating point numbers

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

| Type | Size | Range |
|---|---|---|
| `float` | 32 bits | $-3.4 \times 10^{38}$ to $+3.4 \times 10^{38}$ |
| `double` | 64 bits | $-1.7 \times 10^{308}$ to $+1.7 \times 10^{308}$ |

```
1 double pi = 3.1415926;
```

# Another application: Adding integers

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

```java
import java.util.Scanner;

public class Addition {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    int number1, number2;
    int sum;

    System.out.print("Enter the first integer: ");
    number1 = input.nextInt();
    System.out.print("Enter the second integer: ");
    number2 = input.nextInt();
    sum = number1 + number2;
    System.out.printf("Sum is %d\n", sum);
  }
}
```

# Another application: Adding integers

```
> javac Addition.java
> java Addition
Enter the first integer: 45
Enter the second integer: 72
Sum is 117
>
```

# Another application: Adding integers

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

- *`import` declaration*: Helps the compiler locate a class that is used in this program.
- Related classes are grouped into *package*s.
- `java.util` package provides commonly-used classes. These classes are collectively called *Java class library*, or Java Application Programming Interface (*Java API*).

# Another application: Adding integers

- Variable declaration statement:

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

- *Variable* represents a location in the computer's memory where a value can be stored for use later in a program.
  - Must be declared with a name and a type before use.
- A variable's *name* enables the program to access the value of the variable in memory.
- A variable's *type* specifies what kind of information is stored at that location in memory.

# Another application: Adding integers

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

- The `Scanner` class enables a program to read data for use in a program.
  - The data can come from many sources, such as the user at the keyboard or a file on disk.
- The equals sign `=` in a declaration indicates that the variable should be *initialized* (i.e., prepared for use in the program) with the result of the expression to the right of the equals sign.

# Another application: Adding integers

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

- The `new` keyword creates an object (we will talk more on objects later).
- Standard input object, `System.in`, enables applications to read bytes of information typed by the user.
- `Scanner` object translates these bytes into types that can be used in a program.

# Another application: Adding integers

- Variable declaration statements:

```
1 int number1, number2;
2 int sum;
```

  - Several variables of the same type may be declared in one declaration with the variable names separated by commas.

# Another application: Adding integers

```
1 System.out.print("Enter the first integer: ");
2 number1 = input.nextInt();
```

- *Prompt* is an output statement that directs the user to take a specific action.
- `System` is a class and part of package `java.lang`.
- The class `System` does not need to be imported at the beginning of the program because **the classes in the java.lang package are imported by default**.

# Another application: Adding integers

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

```
1 System.out.print("Enter the first integer: ");
2 number1 = input.nextInt();
```

- `Scanner` method `nextInt` obtains an integer from the user at the keyboard.
- Program waits for the user to type the number and press the `Enter` key to submit the number.
- The result of the call to method `nextInt` is placed in variable `number1` by using the assignment operator `=`.
  - `number1` gets the value of `input.nextInt()`.

# Another application: Adding integers

- Arithmetic operations:

```
1 sum = number1 + number2;
```

- `sum` gets the value of `number1 + number2`.
- Portions of statements that contain calculations are called *expression*s.

# Another application: Adding integers

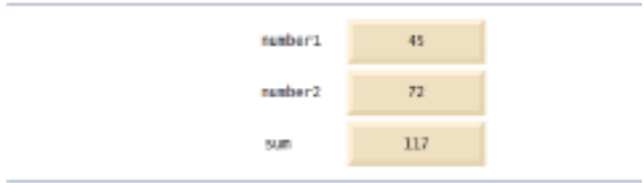
- Integer formatted output:

```
1 System.out.printf("Sum is %d\n", sum);
```

- Format specifier `%d` is a placeholder for an `int` value.
- The letter `d` stands for “decimal integer”.

# Memory concepts

- Variables:
  - Every variable has a *name*, a *type*, a *size* (in bytes) and a *value*.
  - When a new value is placed into a variable, the new value replaces the previous value (if any).
  - The previous value is lost.



# Arithmetic operations

| Operator | Description |
|---|---|
| `+` | Additive operator (also used for `String` concatenation) |
| `-` | Subtraction operator |
| `*` | Multiplication operator |
| `/` | Division operator |
| `%` | Remainder operator |

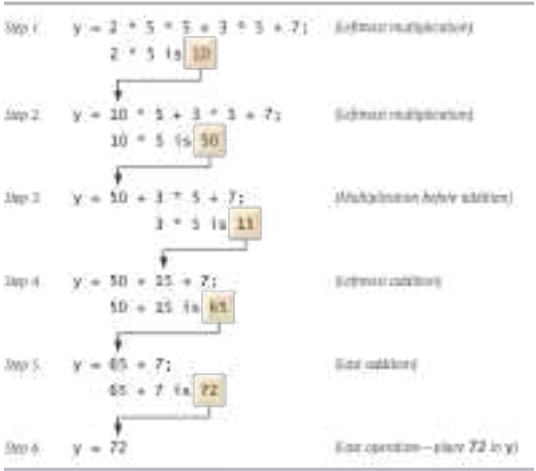
- These are *binary operator*s because they operate on two operands.
- Integer division yields an integer quotient. The fractional part is simply discarded (`3 / 2 = 1`).
- % yields the remainder after division (`10 % 3 = 1`).

# Arithmetic expressions

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

- Arithmetic expressions must be written in straight-line form to facilitate entering programs into the computer.
- Expression “a divided by b” must be written as `a / b` (rather than $\frac{a}{b}$), so that all constants, variables and operators appear in a straight line.
- Parentheses are used to group terms in expressions in the same manner as in algebraic expressions: `a * ( b + c )`.
- In case of nested parentheses, the expression in the innermost set of parentheses is evaluated first: `( ( a + b ) * c )`.

# Operator precedence



# Integer divisions

```
int a = 9;
int b = 2;
float f = a / b - 1.5f;
System.out.printf("The result is %f\n", f);
```

- What is the result?
  - Java drops the fractional part for integer divisions.
  - What to do if you really want to calculate `9 / 2 - 1.5`?

# Errors

- Compilation error:
  - occurs at compile time, reported by the compiler.
  - e.g. syntax error,
  - e.g. assigning a float value to an int variable.
- Logic error:
  - a logic error is a bug in a program that causes it to operate incorrectly,
  - occurs at run time.

```
1 float average(float a, float b) {
2   return a + b / 2;
3 }
```

- What is the result?

# Decision making: Equality and relational operators



- Condition:
  - An expression that can be `true` or `false`.
- `if` selection statement allows a program to make a decision based on a condition's value.
- Equality operators (`==` and `!=`), relational operators (`>`, `<`, `>=` and `<=`).

# Decision making: Equality and relational operators

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

```java
import java.util.Scanner;

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

    int number1, number2;

    System.out.print("Enter the first integer: ");
    number1 = input.nextInt();
    System.out.print("Enter the second integer: ");
    number2 = input.nextInt();

    if (number1 == number2)
      System.out.printf("%d == %d\n", number1, number2);
```

# Decision making: Equality and relational operators

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

```
16      if (number1 != number2)
17        System.out.printf("%d != %d\n", number1, number2);
18      if (number1 < number2)
19        System.out.printf("%d < %d\n", number1, number2);
20      if (number1 > number2)
21        System.out.printf("%d > %d\n", number1, number2);
22      if (number1 <= number2)
23        System.out.printf("%d <= %d\n", number1, number2);
24      if (number1 >= number2)
25        System.out.printf("%d >= %d\n", number1, number2);
26    }
27  }
```

# Decision making: Equality and relational operators

```
> javac Comparison.java
> java Comparison
Enter the first integer: 777
Enter the second integer: 777
777 == 777
777 <= 777
777 >= 777
> javac Comparison.java
> java Comparison
Enter the first integer: 100
Enter the second integer: 200
100 != 200
100 < 200
100 <= 200
```

```
> javac Comparison.java
> java Comparison
Enter the first integer: 200
Enter the second integer: 100
200 != 100
200 > 100
200 >= 100
>
```

# Primitive types vs. reference types

- Java types are divided into two categories: primitive types and reference types.
- Primitive types are the basic types of data.
  - `byte`, `short`, `int`, `long`, `float`, `double`, `boolean`, `char`.
  - A primitive-type variable can store one value of its declared type.

| Type | Description | Default value | Size | Example code |
|---|---|---|---|---|
| `boolean` | Truth value | `false` | 1 bit | `boolean b = false;` |
| `char` | Unicode character | `\0000` | 16 bits | `char c = 'z';` |

# Primitive types vs. reference types

- All non-primitive types are reference types, including instantiable classes and arrays (an array is a container object that holds a fixed number of values of a single type).
  - `Scanner`, `String`, `String[]`, `int[]`.
- Programs use reference-type variables to store the locations of objects in memory.

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

- Such a variable is said to refer to an object in the program. Objects that are referenced may each contain many instance variables of primitive and reference types.

# Primitive types vs. reference types

- Reference-type variables, if not explicitly initialized, are initialized by default to the value `null` (reference to nothing).
- To call methods of an object, you need to use the reference to the object: `myGradeBook.displayMessage();`.
- Primitive-type variables (e.g., int variables) do not refer to objects, so **such variables cannot be used to call methods**.

# Java API specification

- `https://docs.oracle.com/javase/8/docs/api/`.
- Provides detailed description of each Java API (a library method), including the API behavior, required parameters, return types etc.