Chapter 2: Beginning to Program

CS1: Java Programming
Colorado State University

Original slides by Daniel Liang
Modified by Kris Brown, Ben Say, Wim Bohm
Motivations

- Solve practical problems programmatically
- Java primitive data types
- Strings
- Input/Output
- Constants
Variables

A named container that holds a specific piece of data.

Variables have a type (set of values). Some Java types are:

- int
- double
- char
- String

(more later)
Declaring Variables

```java
int x;        // Declare x to be an integer variable;
double radius; // Declare radius to be a double variable;
char a;       // Declare a to be a character variable;
String s;     // Declare s to be a String variable;
```
Assignment Statements

\[ x = 1; \quad \text{// Assign 1 to } x; \]
\[ \text{radius} = 1.0; \quad \text{// Assign 1.0 to } \text{radius}; \]
\[ a = 'A'; \quad \text{// Assign 'A' to } a; \]
\[ s = "Java"; \quad \text{// Assign "Java" to } s \]
Declaring and Initializing in One Step

- `int x = 1;`
- `double d = 1.4;`
- `String s = "Java";`
Variable names

- A variable name is a sequence of characters that consist of letters, digits, underscores (_), and dollar signs ($).
- A variable name must start with a letter, an underscore (_), or a dollar sign ($). It cannot start with a digit.
- A variable name cannot be a reserved word. (See Appendix A, “Java Keywords,” for a list of reserved words).
- A variable name cannot be true, false, or null.
- A variable name can be of any length.
# Numerical Data Types

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

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

- get the computer to print something to the console
- println prints a line and adds a new line at the end
- print prints the line and continues on the same line
- use for DEBUGGING!!
Simple String Operations

Concatenation:

Use the “+” (plus sign) to concatenate strings

System.out.println(mm + " " + yy);
Simple String Operations

The `length()` method

```java
String theName = "Donald Duck";
int len = theName.length();
```

What is returned by `length()`?
Reading Input from the Console

1. Create a Scanner object

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

2. Use the method `nextDouble()` to obtain a double value. For example,

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

Let’s play with IO in Eclipse
Reading Numbers from the Keyboard

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nextByte()</td>
<td>reads an integer of the byte type.</td>
</tr>
<tr>
<td>nextShort()</td>
<td>reads an integer of the short type.</td>
</tr>
<tr>
<td>nextInt()</td>
<td>reads an integer of the int type.</td>
</tr>
<tr>
<td>nextLong()</td>
<td>reads an integer of the long type.</td>
</tr>
<tr>
<td>nextFloat()</td>
<td>reads a number of the float type.</td>
</tr>
<tr>
<td>nextDouble()</td>
<td>reads a number of the double type.</td>
</tr>
</tbody>
</table>
Variables

// Compute the first area
radius = 1.0;
area = radius * radius * 3.14159;
System.out.println("The area is " + area + " for radius " + radius);

// Compute the second area
radius = 2.0;
area = radius * radius * 3.14159;
System.out.println("The area is " + area + " for radius " + radius);
public class ComputeArea {
    /** Main method */
    public static void main(String[] args) {
        double radius;
        double area;

        // Assign a radius
        radius = 20;

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

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

        // Assign a radius
        radius = 20;

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

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

        // Assign a radius
        radius = 20;

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

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

        // Assign a radius
        radius = 20;

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

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

        // Assign a radius
        radius = 20;

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

        // Display results
        System.out.println("The area for the circle of radius " + radius + " is " + area);
    }
}
Your Turn!

Write code in which a String variable `message` contains “The number of rabbits is”. An integer variable `num` has a value of 129. Concatenate these variables into a String called `report`. Then print, using `report`:

The number of rabbits is 129!!
Lecture 2 Operators, Expressions
Named Constants

final datatype CONSTANTNAME = VALUE;

final double PI = 3.14159;
final int SIZE = 3;
Naming Conventions

● Choose meaningful and descriptive names.
● Variables and method names:
  – Use lowercase. If the name consists of several words, concatenate all in one, use lowercase for the first word, and capitalize the first letter of each subsequent word in the name. For example, the variables radius and area.
Naming Conventions, cont.

- **Class names:**
  - Capitalize the first letter of each word in the name. For example, the class name ComputeArea.

- **Constants:**
  - Capitalize all letters in constants, and use underscores to connect words. For example, the constant PI and MAX.VALUE
# Numeric Operators

<table>
<thead>
<tr>
<th>Name</th>
<th>Meaning</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Addition</td>
<td>34 + 1</td>
<td>35</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td>34.0 - 0.1</td>
<td>33.9</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>300 * 30</td>
<td>9000</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
<td>1.0 / 2.0</td>
<td>0.5</td>
</tr>
<tr>
<td>%</td>
<td>Remainder</td>
<td>20 % 3</td>
<td>2</td>
</tr>
</tbody>
</table>
PEMDAS

What is it? Precedence order in arithmetic expressions:

Parentheses
Exponentiation
Multiplication, Division
Addition, Subtraction

\[ 2 + 3 \times 5 = ? \]
\[ (2 + 3) \times 5 = ? \]

Operators at the same level (\(\ast, /\)) and (\(+, -\)) go from left to right (left associative (\(+, -\)) and left associative (\(\ast, /\)))

\[ 2 - 3 + 5 = ? \]
\[ 12 / 3 \times 6 = ? \]
Integer Division

+, -, *, /, and %

5 / 2 yields an integer 2.
5.0 / 2 yields a double value 2.5

5 % 2 yields 1 (the remainder of the (integer) division)
Modulo/Remainder Operator

Remainder is very useful in programming.

An even number % 2 is 0 and an odd number % 2 is 1.

Suppose today is Saturday and you and your friends are going to meet in 10 days. What day is in 10 days? Tuesday; use the following expression:

Saturday is the 6th day in a week

\[(6 + 10) \% 7 = 2\]

A week has 7 days

After 10 days

The 2nd day in a week is Tuesday

Which day is Sunday?
Relationship of / and %

- For all integers p and q: $p = (p/q) \times q + p \% q$

- Terminology:
  - p: dividend
  - q: divisor
  - p/q: quotient
  - p%q: remainder

The remainder is negative only if the dividend is negative. Integer division rounds towards 0: $3/2 = 1$, $-3/2 = -1$
Exponent Operations

// these are (Math)library methods

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
Number Literals

A *literal* is a constant value that appears directly in the program. For example, 34, 1,000,000, and 5.0 are literals in the following statements:

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

An integer literal can be assigned to an integer variable as long as it can fit into the variable.

```
byte b1 = 100;
byte b2 = 1000; // compiler error, WHY?
```

An integer literal is assumed to be of the int type, whose value is between $-2^{31}$ (-2147483648) to $2^{31}-1$ (2147483647).
Floating-Point Literals

Floating-point literals are written with a decimal point. By default, a floating-point literal is treated as a double type value.

```java
double d0 = 0.5;
double d1 = 100.2d;
double d2 = 100.1D;
float f1 = 100.2f;
float f2 = 100.3F;
```
double vs. float

The double type values are more accurate than the float type values. For example,

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

displays \(1.0 \div 3.0 \text{ is } 0.3333333333333333\) \(\text{16 digits}\)

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

displays \(1.0F \div 3.0F \text{ is } 0.3333334\) \(\text{7 digits}\)
Scientific Notation

Floating-point literals can also be specified in scientific notation, for example, 1.23456e+2, same as 1.23456e2, is equivalent to 123.456, and 1.23456e-2 is equivalent to 0.0123456. E (or e) represents an exponent and it can be either in lowercase or uppercase.
# Numeric data types

<table>
<thead>
<tr>
<th>Name</th>
<th>Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>8 bit signed integer</td>
</tr>
<tr>
<td>short</td>
<td>16 bit signed integer</td>
</tr>
<tr>
<td>Int</td>
<td>32 bit signed integer</td>
</tr>
<tr>
<td>long</td>
<td>64 bit signed integer</td>
</tr>
<tr>
<td>float</td>
<td>32 bit floating point number</td>
</tr>
<tr>
<td>double</td>
<td>64 bit floating point number</td>
</tr>
</tbody>
</table>

For value ranges see book. E.g.:

- byte range: -128 to 127
- int range: ~ 2E10 to 2E10
Arithmetic Expressions

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

is translated to

\[
\frac{3+4x}{5} - \frac{10(y-5)(a+b+c)}{x} + 9\left(\frac{4}{x} + \frac{9+x}{y}\right)
\]
How to Evaluate an Expression

Apply the precedence and associativity rules discussed earlier To determine the order of evaluation.

Parentheses ( ) allow changing the order of evaluationin

\[
3 + 4 \times 4 + 5 \times (4 + 3) - 1
\]

1. inside parentheses first

\[
3 + 4 \times 4 + 5 \times 7 - 1
\]

2. multiplication

\[
3 + 16 + 5 \times 7 - 1
\]

3. multiplication

\[
3 + 16 + 35 - 1
\]

4. addition

\[
19 + 35 - 1
\]

5. addition

\[
54 - 1
\]

6. subtraction

\[
53
\]
Full parenthesization

- In full parenthesization every sub expression is parenthesized. It explicitly defines the evaluation order in an expression:

\[ 5 \times (3 - 2) - 3 + 4 \rightarrow (((5 \times (3-2)) - 3) + 4) \]
Augmented Assignment Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Example</th>
<th>Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>+=</td>
<td>Addition assignment</td>
<td>i += 8</td>
<td>i = i + 8</td>
</tr>
<tr>
<td>-=</td>
<td>Subtraction assignment</td>
<td>i -= 8</td>
<td>i = i - 8</td>
</tr>
<tr>
<td>*=</td>
<td>Multiplication assignment</td>
<td>i *= 8</td>
<td>i = i * 8</td>
</tr>
<tr>
<td>/=</td>
<td>Division assignment</td>
<td>i /= 8</td>
<td>i = i / 8</td>
</tr>
<tr>
<td>%=</td>
<td>Remainder assignment</td>
<td>i %= 8</td>
<td>i = i % 8</td>
</tr>
</tbody>
</table>
## Increment and Decrement Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Description</th>
<th>Example (assume $i = 1$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$++\text{var}$</td>
<td>preincrement</td>
<td>Increment $\text{var}$ by 1, and use the new $\text{var}$ value in the statement</td>
<td>$\text{int } j = ++i; \quad \text{// } j \text{ is } 2, \text{ i is } 2$</td>
</tr>
<tr>
<td>$\text{var}++$</td>
<td>postincrement</td>
<td>Increment $\text{var}$ by 1, but use the original $\text{var}$ value in the statement</td>
<td>$\text{int } j = i++; \quad \text{// } j \text{ is } 1, \text{ i is } 2$</td>
</tr>
<tr>
<td>$--\text{var}$</td>
<td>predecrement</td>
<td>Decrement $\text{var}$ by 1, and use the new $\text{var}$ value in the statement</td>
<td>$\text{int } j = --i; \quad \text{// } j \text{ is } 0, \text{ i is } 0$</td>
</tr>
<tr>
<td>$\text{var}--$</td>
<td>postdecrement</td>
<td>Decrement $\text{var}$ by 1, and use the original $\text{var}$ value in the statement</td>
<td>$\text{int } j = i--; \quad \text{// } j \text{ is } 1, \text{ i is } 0$</td>
</tr>
</tbody>
</table>
Increment and Decrement Operators, cont.

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

Same effect as

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

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

Same effect as

```java
i = i + 1;
int newNum = 10 * i;
```
Increment and Decrement Operators, cont.

Using increment and decrement operators makes expressions short, but it also makes them complex and difficult to read. Avoid using these operators in expressions that modify multiple variables, or the same variable for multiple times such as this: int k = ++i + i.
Assignment Expressions and Assignment Statements

Prior to Java 2, all the expressions can be used as statements. Since Java 2, only the following types of expressions can be statements:

- `variable op= expression;` // Where op is +, -, *, /, or %
- `++variable;`
- `variable++;`
- `--variable;`
- `variable--;`

(because they are implicit assignment statements)
Type Casting

Implicit casting

```java
double d = 3;  // (type widening)
```

Explicit casting

```java
int i = (int)3.0;  // (type narrowing)
int i = (int)3.9;  // (fraction part is truncated)
```

What is wrong?

```java
int x = 5 / 2.0;
```

Range increases:

```
byte, short, int, long, float, double
```
***Conversion Rules***

When performing a binary operation involving two operands of different types, Java automatically converts the operand based on the following rules:

1. If one of the operands is double, the other is converted into double.
2. Otherwise, if one of the operands is float, the other is converted into float.
3. Otherwise, if one of the operands is long, the other is converted into long.
4. Otherwise, both operands are converted into int.
Casting in an Augmented Expression

In Java, an augmented expression of the form \( x_1 \ op= \ x_2 \) is implemented as \( x_1 = (T)(x_1 \ op \ x_2) \), where \( T \) is the type for \( x_1 \). Therefore, the following code is correct.

```java
int sum = 0;
sum += 4.5; // sum becomes 4 after this statement
// is equivalent to sum = (int)(sum + 4.5).
```
Common Errors and Pitfalls

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

- Common Pitfall 1: Redundant Input Objects
Common Error 1: Undeclared/Uninitialized Variables and Unused Variables

double interestRate = 0.05;
double interest = interestRate * 45;
Common Error 2: Integer Overflow

```java
int value = 2147483647 + 1;
// value will actually be -2147483648
```
Common Error 3: Unintended Integer Division

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

(a)

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

(b)
Common Pitfall 4: Redundant Input Objects

Scanner input = new Scanner(System.in);
System.out.print("Enter an integer: ");
int v1 = input.nextInt();

Scanner input1 = new Scanner(System.in);
System.out.print("Enter a double value: ");
double v2 = input1.nextDouble();

Let’s go find out what happens . . .
Your Turn!

Write a program that stores the remainder of dividing the integer variable $i$ by integer variable $j$ in an integer variable named $k$. Use a Scanner object to get those variables from a user at the keyboard. Print $k$ to the console.

Enter $i$:
Enter $j$:
Result $k$:
You can format your decimal numbers using, for example, the following:

```
System.out.printf("Total is: $%,.2f%n", dblTotal);
```

You will learn about printf in recitation.
NOTE

Calculations involving floating-point numbers are approximate because these numbers are not stored with complete accuracy. For example,

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

displays 0.5000000000000001, not 0.5, and

System.out.println(1.0 - 0.9);

displays 0.09999999999999998, not 0.1.

Integers are stored precisely. Therefore, calculations with integers yield a precise integer result.
Common Error 3: Round-off Errors

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

Let’s go look what happens . . .