04 Variables and Assignments

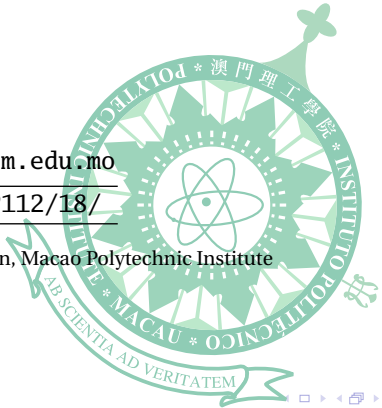
Instructor: Ke Wei（柯韋）

A319 ✆ Ext. 6452 ✉ wke@ipm.edu.mo

http://brouwer.ipm.edu.mo/COMP112/18/

Bachelor of Science in Computing, School of Public Administration, Macao Polytechnic Institute

September 07, 2018



Outline

Computing the Area of a Circle

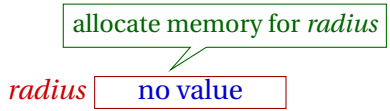
Let's *trace* the execution of this program.

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

Computing the Area of a Circle

Let's *trace* the execution of this program.

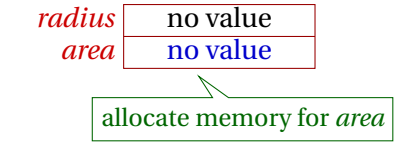
```
1  public class ComputeArea {
2      // Main method
3      public static void main(String[] args) {
4        →double radius;
5          double area;
6          // Assign a radius
7          radius = 20;
8          // Compute area
9          area = radius * radius * 3.14159;
10         // Display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



Computing the Area of a Circle

Let's *trace* the execution of this program.

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



Computing the Area of a Circle

Let's *trace* the execution of this program.

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

assign 20 to *radius*

radius	20
area	no value

Computing the Area of a Circle

Let's *trace* the execution of this program.

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

radius	20
area	1256.636

compute area and assign the result to variable *area*

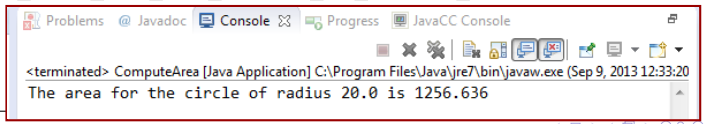
Computing the Area of a Circle

Let's *trace* the execution of this program.

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

radius	20
area	1256.636

print a message to the console



Reading Input from the Console

1. Import the *Scanner* class.

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

2. Create a *Scanner* object to process the standard *input stream*.

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

3. Through the *scanner* object, call the methods *next*(), *nextByte*(), *nextShort*(), *nextInt*(), *nextLong*(), *nextFloat*(), *nextDouble*(), or *nextBoolean*() to obtain a string, `byte`, `short`, `int`, `long`, `float`, `double`, or `boolean` value. For example,

   ```
   System.out.print("Enter a double value: ");
   double d = scanner.nextDouble();
   ```

4. Close the *scanner* object when all the input is finished.

   ```
   scanner.close();
   ```

Using *Scanner* in *try-with-resources* Block

The new *try-with-resources* statement beginning from Java 7 auto-closes the scanner.

1. Import the *Scanner* class.
2. Declare the variables to input before the `try` block. For example,

```
double d;
```

3. Create a *Scanner* object in the `try` block header to process the standard *input stream*.
4. Put the input statements in the `try` block body. For example,

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

The *scanner* object is closed automatically when the execution goes beyond the `try` block.

Keywords and Identifiers

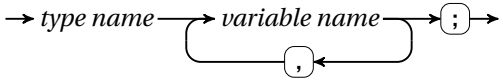
- Keywords (also called reserved words) are names for built-in entities (such as built-in types), and other language constructs (such as control flow statements).
- Identifiers are user defined names for variables, classes, methods and other entities.

Good Identifiers	Bad Identifiers
`num`	`2er`
`_Name`	`power of two`
`weight_of_2_cats`	`Henry-King`

- An identifier is a sequence of characters that consists of letters, digits, underscores (_), and dollar signs ($).
- An identifier cannot be a keyword.
  - See Appendix A, "Java Keywords", for a list of reserved words.
- An identifier cannot be `true`, `false`, or `null`.
  - These words seem like keywords, but they are actually *literals*, just like floating point literal 3.14 and string literal `"Hello"`.

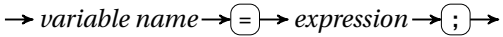
Using Variables

- **Declaration**. Before a variable can be used, we must declare it and give it a type.




```
int x, y;          // Declare x and y to be two integer variables.
double radius;     // Declare radius to be a double variable.
char a;            // Declare a to be a character variable.
```

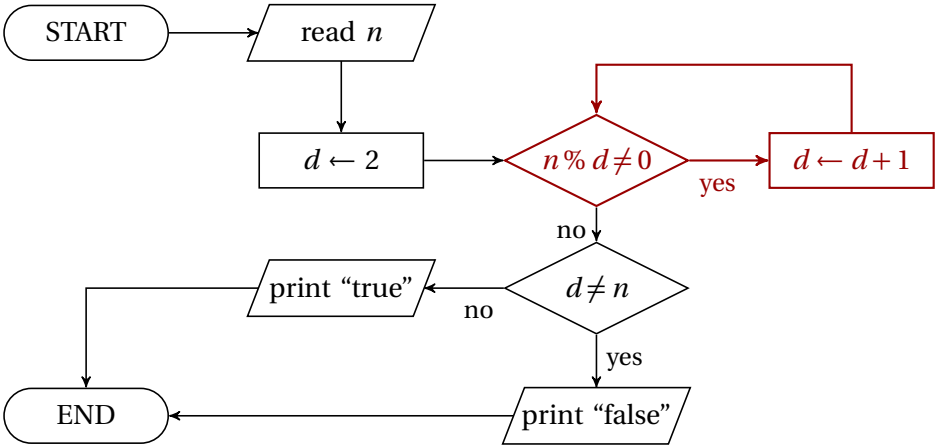
- **Assignment**. We can assign a value to a variable, the value can be a constant, or a value taken from another variable, or the result of an *expression*.




```
x = 1;              // Assign 1 to x.
radius = 1.0;       // Assign 1.0 to radius.
a = 'A';            // Assign 'A' to a.
radius = radius*2;  // Double the radius.
```

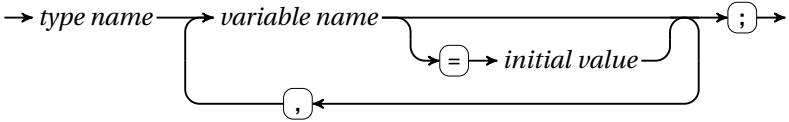
Primality Test — Self-increment

The simplest way to determine whether an input number is a prime number is trial division.



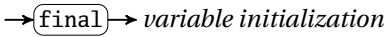
Variable Initialization and Named Constants

- **Variable initialization**. We can give an initial value to a variable when we declare it.




```
int x = 1;
double d = 1.4, g = 7.8;
```

- **Named constant declaration**. We may also declare variables that cannot be assigned with new values, these variables are called *named constants*.




```
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*, and the method *computeArea*.
- **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`

Integer Data Types

Name	Range	Storage Size
`byte`	-2^7 to -2^7-1 (-128 to 127)	8-bit
`short`	-2^{15} to $2^{15}-1$ (-32768 to 32767)	16-bit
`int`	-2^{31} to $2^{31}-1$ (-2147483648 to 2147483647)	32-bit
`long`	-2^{63} to $2^{63}-1$ (-9223372036854775808 to 9223372036854775807)	64-bit

Integer Operations

Name	Meaning	Example	Result
*	Multiplication	`300 * 30`	9000
/	Division	`5 / 2`	2
		`9 / (-3)`	-3
%	Modulo	`20 % 3`	2
		`(-20) % 6`	-2
+	Addition	`34 + 1`	35
		`(-2) + 3`	1
-	Subtraction	`34 - 3`	31
		`3 - 15`	-12

Integer Division and Modulo Operation

- Integer division truncates the result *towards zero* (the nearest integer between the result and zero).

  `5/2` yields 2 and `-17/3` yields −5,
  `5.0/2` yields a double value 2.5 and `-17.0/3` yields −5.666666666666667.

- The modulo operation returns the remainder of the division: `5%2` yields 1.
- Remainder is very useful in programming. For example, an even number % 2 is always 0 and an odd number % 2 is always 1. So you can use this property to determine whether a number is even or odd.
- Suppose today is Saturday and you and your friends are going to meet in 10 days. What day is in 10 days? You can find that day is Tuesday using the following expression:

```
(6 + 10) % 7 // the result is 2.
```

Write a program that obtains hours and minutes from seconds.

Truncation and Negative Remainders

- Euclidean division is the process of division of two integers a and b, which produces a quotient q and a remainder r. These four integers satisfy,

$$a = bq + r. \tag{1}$$

  When $0 \leq r < |b|$, q and r are unique.
- With the truncation in Java's integer division, bq is not always less than a, when negative numbers are involved. For example,

  `-5 / 2` yields `-2` and `2 * (-2)` yields `-4`

  To ensure that 1 holds, we must have a negative remainder -1 in this case.
- As a result, we can always use (1) to determine whether the remainder is negative.

```
-5 %  2  yields -1        7 % (-4) yields  3
-8 %  3  yields -2       -9 % (-4) yields -1
```

- As we have observed, in Java, the sign of the remainder agrees with the sign of the dividend, regardless the sign of the divisor.

Reading Homework

Textbook

- Section 2.6 – 2.9.
- Check Point 2.1 – 2.11.

Internet

- Naming convention (`http://en.wikipedia.org/wiki/Naming_convention_(programming)`).

Self-test

- 2.10 – 2.30 (`http://tiger.armstrong.edu/selftest/selftest9e?chapter=2`).