Problem Solving & Algorithm Design
with Some Control Structures

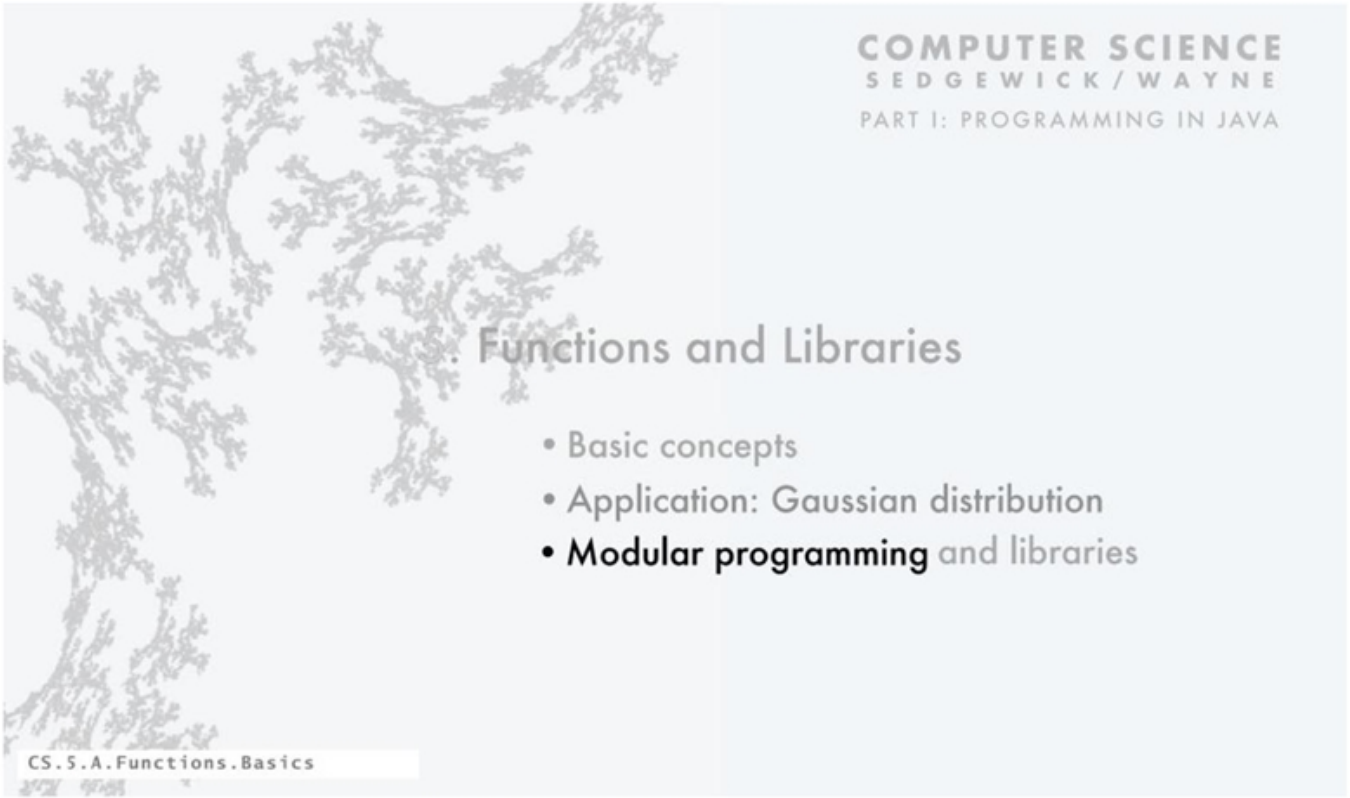
- Dr. 何明昕, He Mingxin, Max
- program06 @ yeah.net

- Email Subject：(AE|A2|A3) + *(Last 4 digits of ID) + Name: TOPIC*
- Sakai: CS102A in 2018A

计算机程序设计基础
Introduction to Computer Programming

Thanks Prof. Stéphane Faroult for his inspiration and efforts on this topic.

Continue from the last notes (03):



Why modular programming?

Modular programming enables

- Independent development of small programs.
- Every programmer to develop and share layers of abstraction.
- Self-documenting code.

Fundamental characteristics

- Separation of client from implementation benefits all *future* clients.
- Contract between implementation and clients (API) benefits all *past* clients.

Challenges

- How to break task into independent modules?
- How to specify API?

Using Static Methods: Functions

Static methods allow you to encode a wide variety of mathematical and data processing functions.

Structuring your code with methods has the following benefits:

- encourages good coding practices by emphasizing discrete, reusable methods;
- encourages self-documenting code through good organization;
- when descriptive names are used, high-level methods can read more like a narrative, reducing the need for comments;
- reduces code duplication.

Two ways to pass parameters to a function(method):

By value – the function gets a copy of the value - numerical values, single chars, expressions can be used

By reference – the function gets the memory address of the real thing – arrays (cannot do otherwise), objects

Functions can return:

- Nothing (`void`)
- Simple values
- References (arrays, objects)

Functions (methods) in Java can be overloaded – It allows several versions of a function, with the same name but different parameters.

Flickr: Tim McCune

Overloading 重载

Different versions of a function

Same name

Often same return type (not always)

Different parameters

In the Math class, you have for instance four different functions called max() that all take parameters of different types and return the greatest. We could have imagined a function taking only double values; but then you couldn't have assigned without casting the returned result to an int. Specific functions are more convenient.

Overloading

Math

`static double`	`max(double a, double b)`	Returns the greater of two double values.
`static float`	`max(float a, float b)`	Returns the greater of two float values.
`static int`	`max(int a, int b)`	Returns the greater of two int values.
`static long`	`max(long a, long b)`	Returns the greater of two long values.

The name is the same because the operation is the same, and you return a type that matches what was passed as parameters.

Defining a library of functions

$$\phi(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$$

$$\phi(x, \mu, \sigma) = \frac{1}{\sigma}\phi\left(\frac{x-\mu}{\sigma}\right)$$

```
public class Gaussian
{
   public static double pdf(double x)
   {
      return Math.exp(-x*x / 2) / Math.sqrt(2 * Math.PI);
   }

   public static double pdf(double x, double mu, double sigma)
   {
      return pdf((x - mu) / sigma) / sigma;
   }

   // Stay tuned for more functions.
}
```

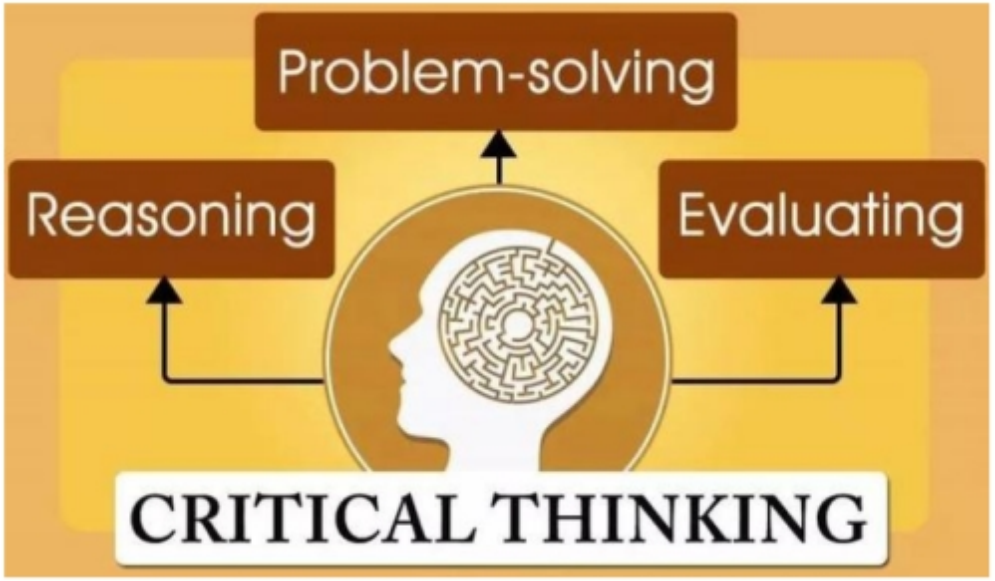
call a function in another module

module named Gaussian.java

functions with different signatures are different (even if names match)

Functions and libraries provide an easy way for any user to *extend* the Java system.

Problem Solving



Functions are much helpful in the problem solving process.

When you are a beginner, when you are asked "write a program that does this and that", you usually have some trouble finding out where to start from.

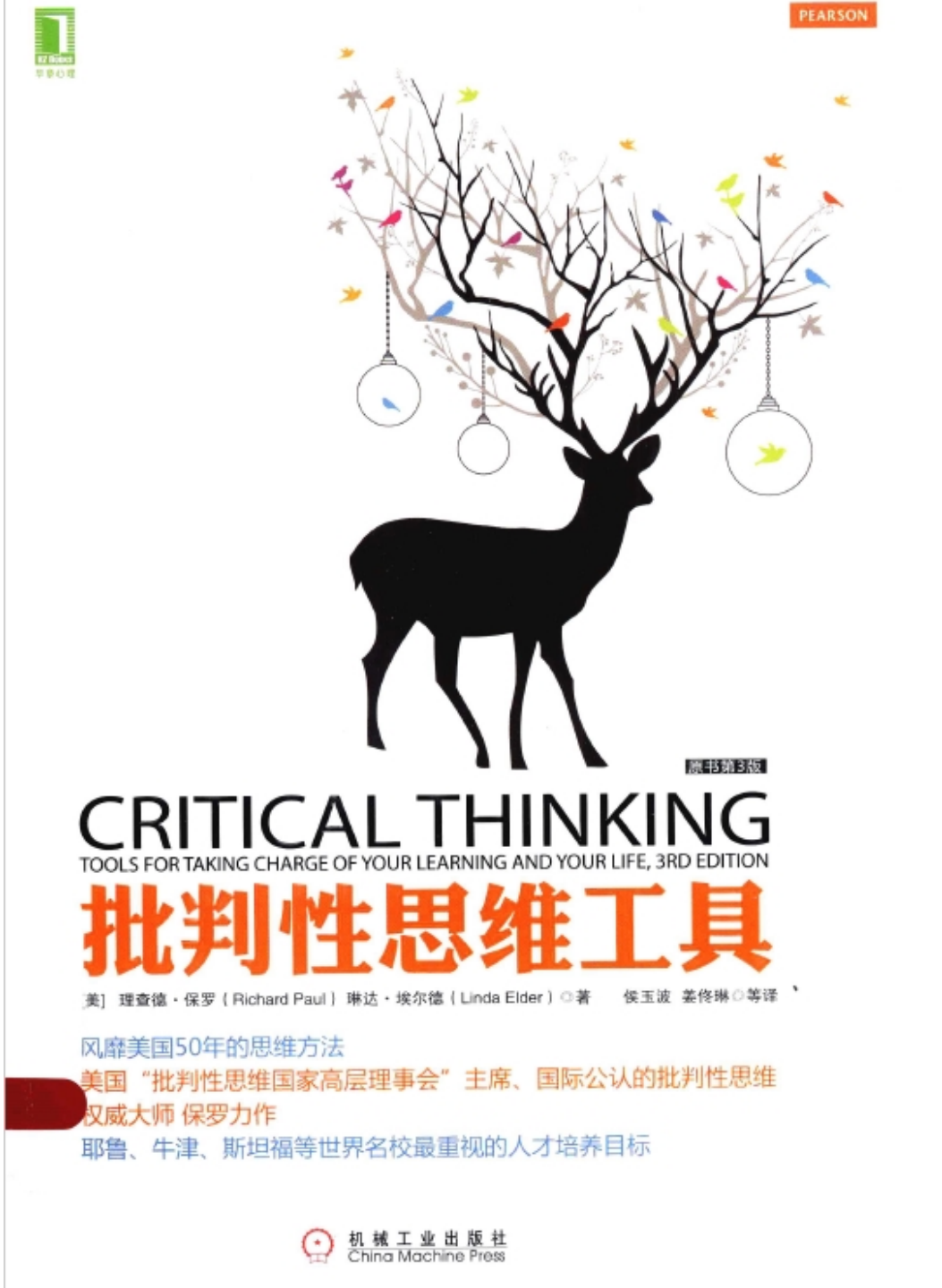
We are going to discuss how people solve programming problems, and also what makes the difference between a good and a not-so-good program.

Critical Thinking

- 批判性思维
  关键性思考
  建设性思考

- Critical Observing:
  建设性观察

- 系统化认知问题与
  构建解决方案的能力



ALGORITHMS (算法)

With what you have seen so far you can develop algorithms.

Algorithms + Data Structures = Programs

From Wikipedia, the free encyclopedia

Algorithms + Data Structures = Programs[1] is a 1976 book written by Niklaus Wirth covering some of the fundamental topics of computer programming, particularly that algorithms and data structures are inherently related. For example, if one has a sorted list one will use a search algorithm optimal for sorted lists.

The book was one of the most influential computer science books of the time and, like Wirth's other work, was extensively used in education.[2]

Method for getting a result using:

Sequential instructions

Conditional instructions

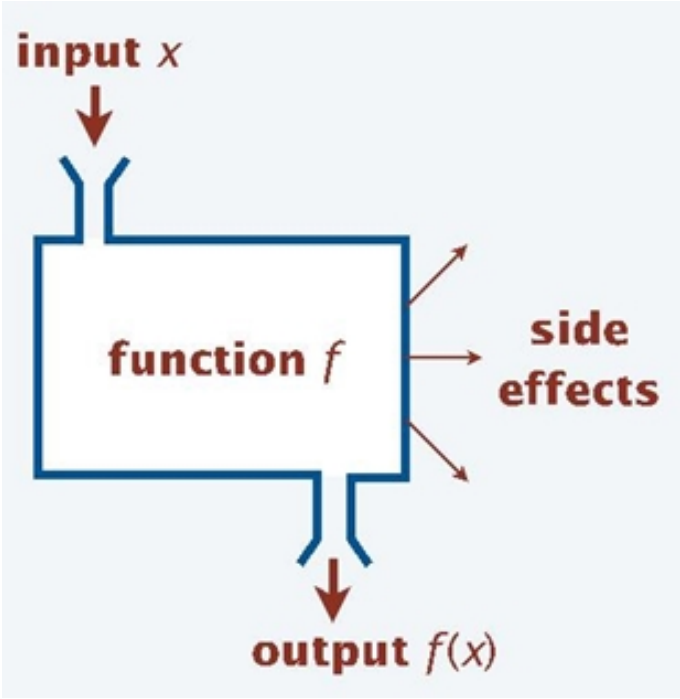
```
(if ... then ... else ...)
```

Loops

Algorithms use nothing more than the constructs we have seen.

The Problem Solving Process

1) What is the problem?



Designing an algorithm starts with defining the problem, which is usually done:

- ✓by seeing what we get as input,
- ✓and defining what we get as output,
- ✓to find out what we should do in-between.

Hence to fight and conquer in all your battles
is not supreme excellence;
supreme excellence consists in breaking the enemy's resistance without fighting.

Sun Tzu, The Art of War (6th century BC)

Flickr:Kanegen

You shouldn't forget the wise words of Sun Tzu, 2,500 years ago. Some people see problems where there are none.

We need critical thinking to help understanding problems correctly.

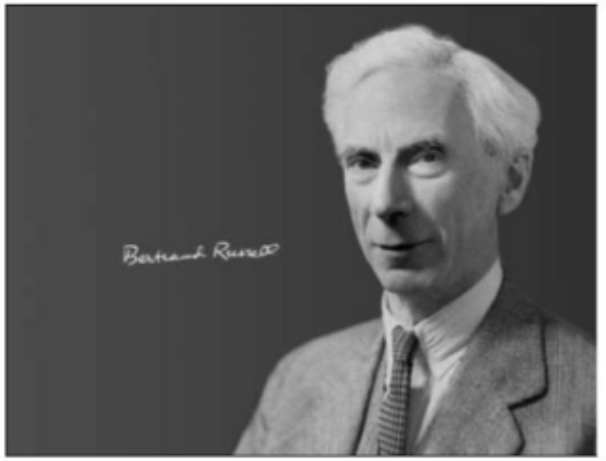
1) What is the problem?

All this requires defining very precisely

- what you are going to do,
- what kind of input you will accept
- and what you will do with unacceptable input.

It's far harder than it seems.

Everything is vague to a degree you do not realize till you have tried to make it precise.



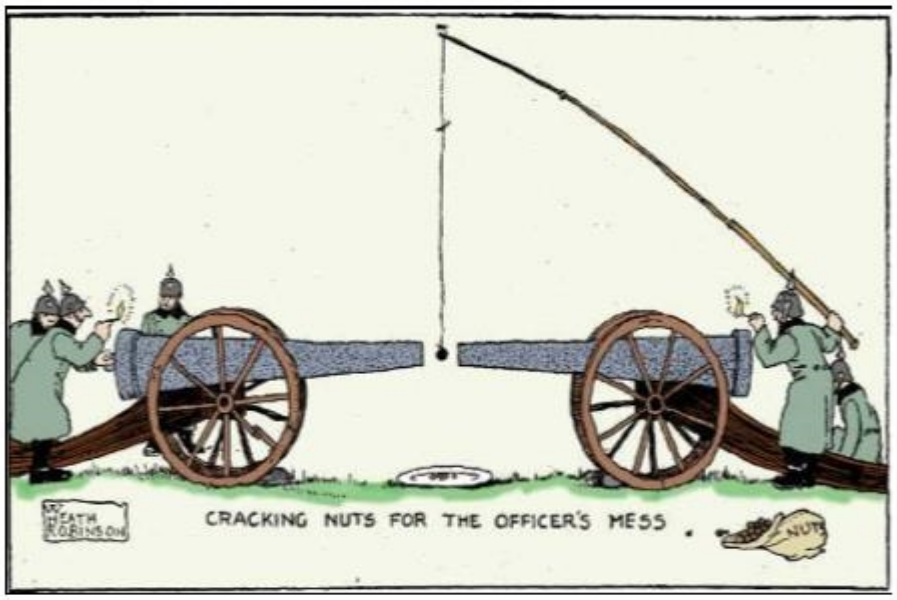
1872-1970

2) Outline a solution

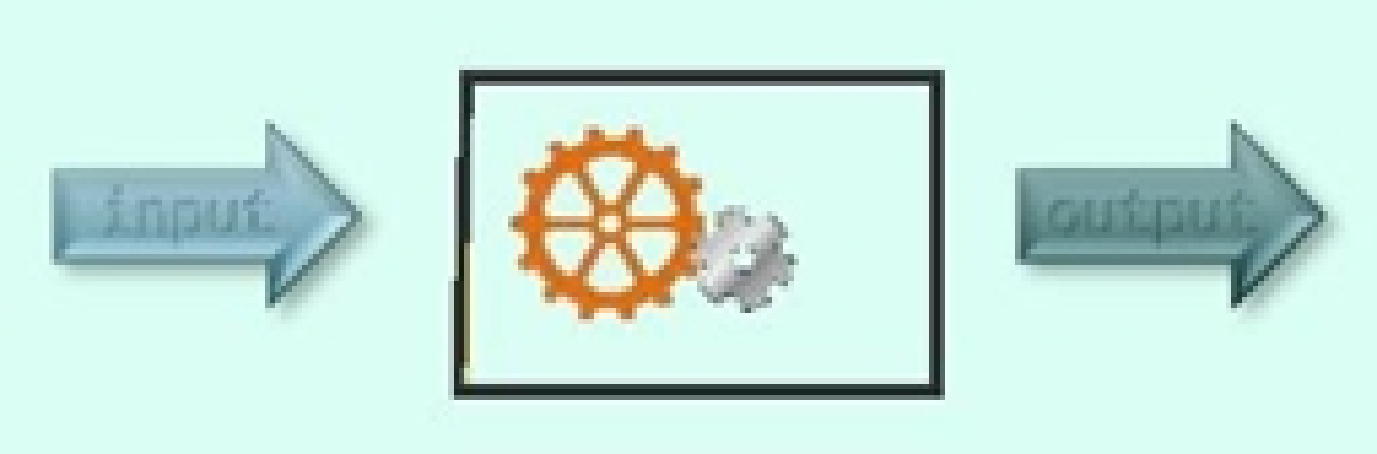
Once you know what precisely you want to do, it's time to think about how you could do it.

Once again, not always as easy as it looks.

There is always a human tendency to think first of the most complicated and less efficient solution.



3) Design an algorithm



Then you can turn your idea into a clean algorithm.

3) Design an algorithm

Finite number of well-defined steps.

- Terminates
- Exact result
- Reaches wanted precision

An algorithm is a procedure you must follow step by step. Sometimes, because precision isn't infinite in a computer (think of pi), termination is decided by the precision of an approximate result.

3) Design an algorithm

Several possibilities

Which is the best one?

It's quite common that there are several ways of solving a problem - several algorithms. For instance, you can solve a system of equations using either Gaussian elimination or Cramer's method. There is a need for assessing an algorithm vs another algorithm.

3) Design an algorithm

Speed

Precision

Storage

The most common characteristics used for comparing algorithms are speed, precision and storage.

3) Design an algorithm

Elegance

Something that may surprise you is that a lot of attention is often brought to the elegance of an algorithm. You can write an elegant algorithm in the same way as you can write elegant prose or poetry.

Simple is difficult

It's far, far harder to write an elegant algorithm that solves a problem in an easily understandable way than to write a clunky algorithm that simply does the job.

Flickr: Steve Johnson

3) Design an algorithm

Pseudo-code

Algorithms are usually written in pseudo-code.

What is pseudo-code? As the name says, not really code. The purpose of pseudo-code is to describe the key points of an algorithm (important variables and data structures, tests, loops) in a way that is free from programming language syntax. Several different, and all valid, examples follow.

3) Design an algorithm

Set total to zero
Set grade counter to one
While grade counter is less than or equal to ten
    Input the next grade
    Add the grade into the total
Set the class average to the total divided by ten
Print the class average.

3) Design an algorithm

```
Set moveCount to 1
FOR each row on the board
   FOR each column on the board
      IF gameBoard position (row, column) is occupied THEN
         CALL findAdjacentTiles with row, column
         INCREMENT moveCount
      END IF
   END FOR
END FOR
```

3) Design an algorithm

```
For (i = 1; i <= 100; i++) {
   set printNumber to true;
   If i is divisible by 3
      print "Fizz";
      set printNumber to false;
   If i is divisible by 5
      print "Buzz";
      set printNumber to false;
   If printNumber, print i;
   print a newline;
}
```

Some people find writing seudocode that almost looks like real code easier, but it's up to you.

4) Convert algorithm in pseudocode to programming language

Once pseudocode is written, the hard work is done.

If the pseudocode is good, any programmer, should be able to turn it into a program in a computer language he masters.

5) Verify operations

Check that everything works according to plans.

Test boundaries and invalid values.

When modifying a program, check that what used to work still does.

Five-Steps: Problem Solving Process

1. Define the problem
2. Outline a solution
3. Design an algorithm
4. Convert the algorithm into a program
5. Verify correct operation of the program

Alternate method to algorithms:

Sometimes there is no method that allows you to obtain the result in a reasonable amount of time (playing chess is an example, there are many others)

short-cuts

"good enough" result

The word "heuristic" is used to describe a method that is commonsensical but approximate. Frequently used in operations research.

Don't use heuristic where there is a perfectly good Algorithm.

Primitive Types & Operators
Structured Programming

Reviews: Types, literals(constant values), variables, operators

Type(数据类型): A set of **values**
together with **operations** on them
and a **limited storage** with specific format.

- 8 Primitive types(基本类型) :

  boolean, char, byte, short, int, long, float, double

- Some used reference (object) types (引用类型，引用对象):
  String, Scanner, InputStream, OutputStream, array of a type

Data Types — Integer Data Types

Java's basic data types are almost identical to C/C++ data types. The main difference is that Java's types all have a set size regardless of platform, while C/C++'s data types only have a minimum number of bits, which causes some variation between platforms. It also has the same two categories; Integer and Floating Point.

Integer types:

Name	Size	Range	Notes
boolean	1	**true** or **false**	like C++'s bool, but it can't be assigned with a number
char	16	0 to 65535	This is bigger than C's char. This is because Java strings are Unicode, not ASCII. It's also unsigned by default.
byte	8	-128 to 127	Standard issue 'byte'
short	16	-32768 to 32767	just like a **char**, but signed by default
int	32	-2147483648 to 2147483647	standard-issue integer number type
long	64	-9223372036854775808 to 9223372036854775807	For very huge integers

Floating Point types:

Name	Size	Range	Notes
float	32	+/- 1.4023×10^{-45} to $3.4028 \times 10^{+38}$	general purpose real-number
double	64	+/- 4.9406×10^{-324} to 1.7977×10^{308}	higher-precision real number

All data types in Java are always signed (except for boolean, of course). Also note that **long** is guaranteed to be 64 bits long.

Operators						Associativity	Type
`++`	`--`					right to left	unary postfix
`++`	`--`	`+`	`-`	`!`	(*type*)	right to left	unary prefix
`*`	`/`	`%`				left to right	multiplicative
`+`	`-`					left to right	additive
`<`	`<=`	`>`	`>=`			left to right	relational
`==`	`!=`					left to right	equality
`&`						left to right	boolean logical AND
`^`						left to right	boolean logical exclusive OR
`|`						left to right	boolean logical inclusive OR
`&&`						left to right	conditional AND
`||`						left to right	conditional OR
`?:`						right to left	conditional
`=`	`+=`	`-=`	`*=`	`/=`	`%=`	right to left	assignment

Fig. 5.19 | Precedence/associativity of the operators discussed so far.

Logical Operators

- Java's logical operators enable you to form more complex conditions by combining simple conditions.
- The logical operators are :

  && (conditional AND)

  || (conditional OR)

  & (boolean logical AND)

  | (boolean logical inclusive OR)

  ^ (boolean logical exclusive OR)

  ! (logical NOT).

- [Note: The &, | and ^ operators are also bitwise operators when they are applied to integral operands.]

Logical Operators (Cont.)

- The parts of an expression containing && or || operators are evaluated only until it's known whether the condition is true or false.
- This feature of conditional AND and conditional OR expressions is called short-circuit evaluation.
- Operator && has a higher precedence than operator ||.
- Both operators associate from left to right.

Conditional operator (?:)

- Conditional operator (?:)—short hand if…else.
- Ternary operator (takes three operands)
- Operands and ?: form a conditional expression (produce a value)

  condition ? *value4true* : *value4false*

- Operand to the left of the ? is a boolean expression— condition evaluates to a boolean value (true or false)
- Second operand (value4true , between the ? and :) is the value if the boolean expression condition is true
- Third operand (value4false, to the right of the :) is the value if the boolean expression condition is false.

- Example:

```
System.out.println(
   studentGrade >= 60 ? "Passed" : "Failed"
);
```

- Evaluates to the string "Passed" if the boolean expression studentGrade >= 60 is true and to the string "Failed" if it is false.

- Example:

```
System.out.print(
   n + " bird" + (n <= 1 ? "" : "s")
);
```

- Output like:

```
0 bird  1 bird  2 birds  3 birds  ..  32767 birds
```

Structured Programming Summary

- Structured programming promotes simplicity.
- Bohm and Jacopini: Only three forms of control are needed to implement an algorithm:
  - Sequence
  - Selection
  - Repetition
- The sequence structure is trivial. Simply list the statements to execute in the order in which they should execute.

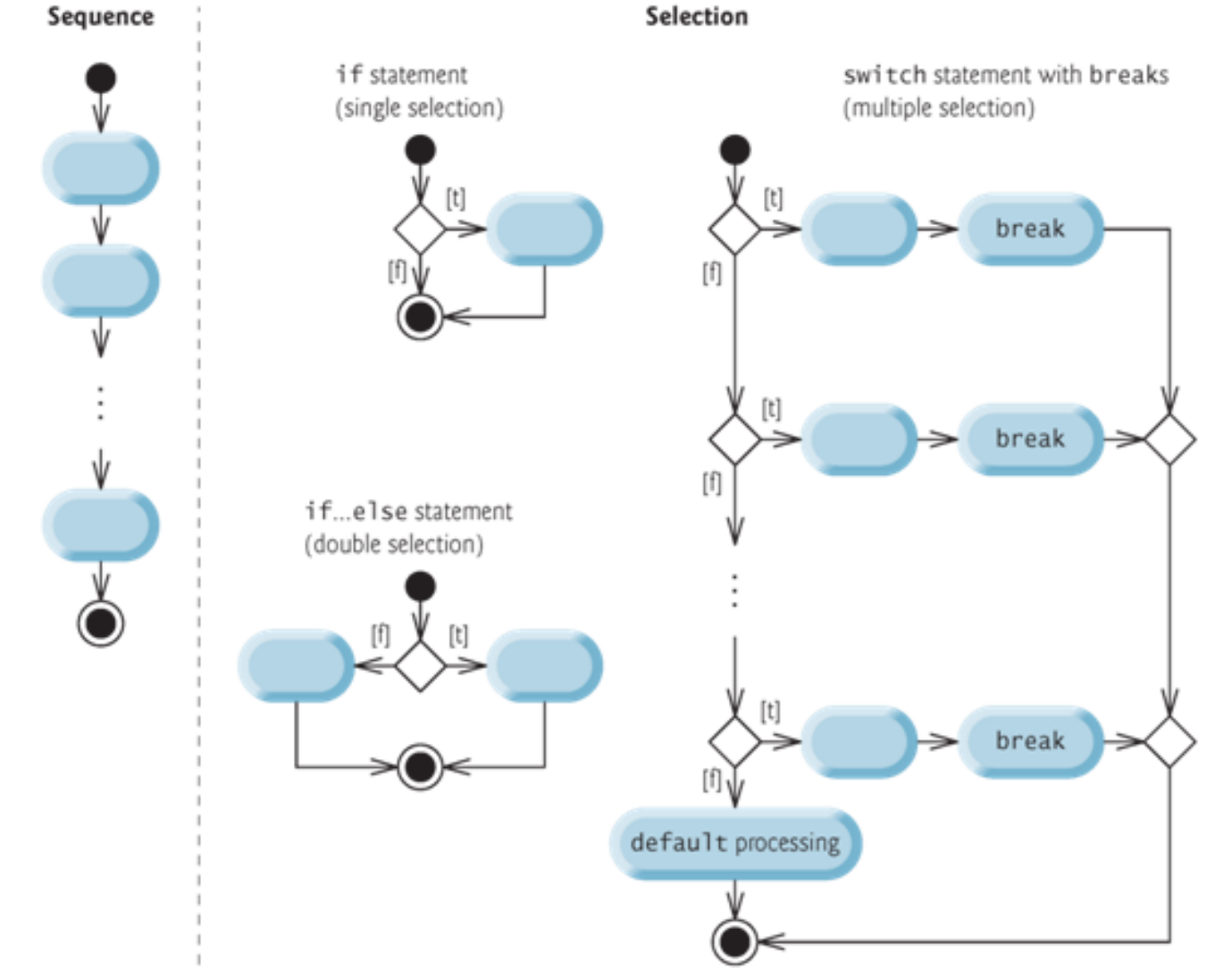


Fig. 5.20 | Java's single-entry/single-exit sequence, selection and repetition statements. (Part 1 of 2.)

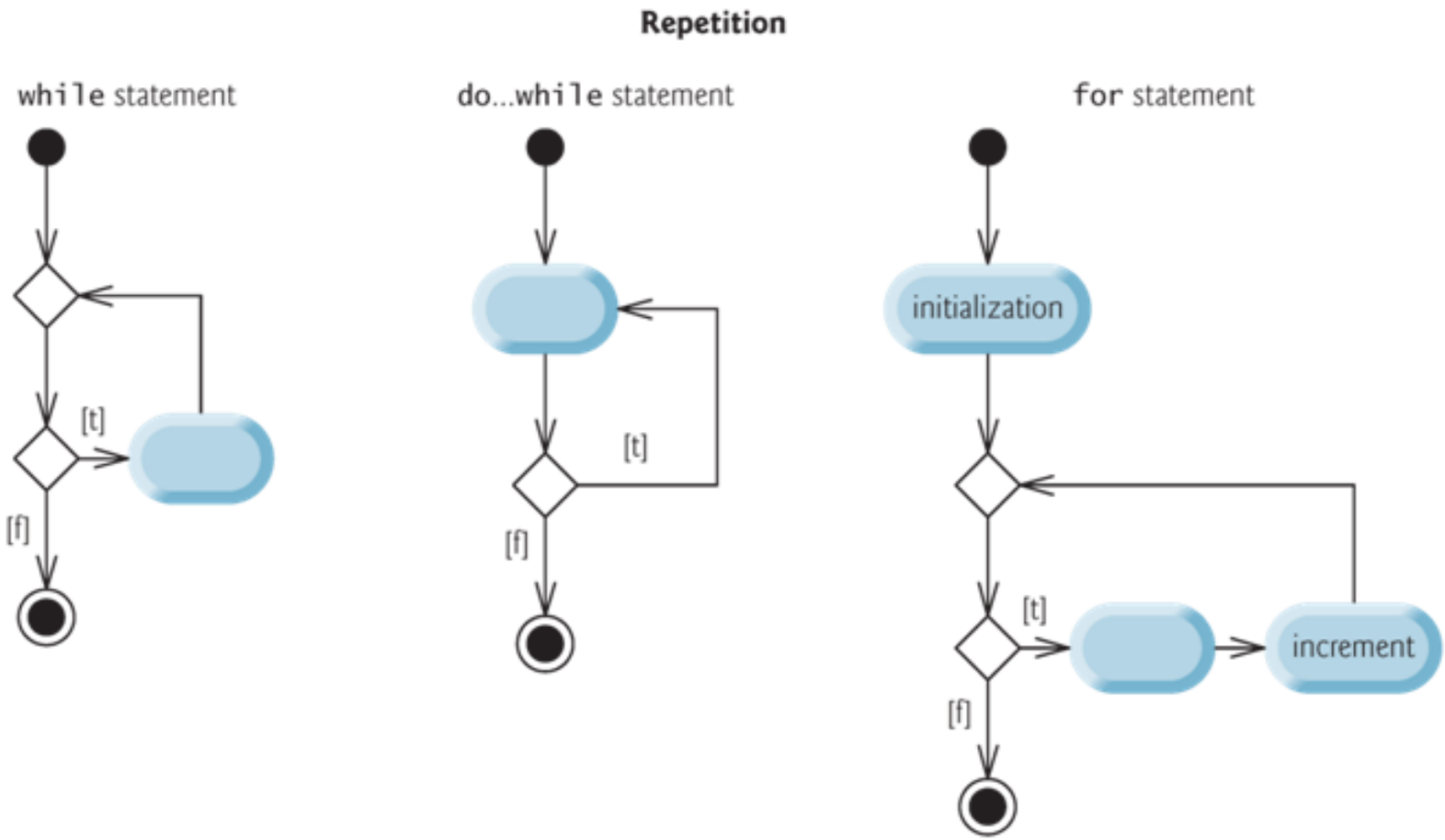


Fig. 5.20 | Java's single-entry/single-exit sequence, selection and repetition statements. (Part 2 of 2.)

Structured Programming Summary (Cont.)

- Selection is implemented in one of three ways:
  - if statement (single selection)
  - if…else statement (double selection)
  - switch statement (multiple selection)
- The simple if statement is sufficient to provide any form of selection—everything that can be done with the if…else statement and the switch statement can be implemented by combining if statements.

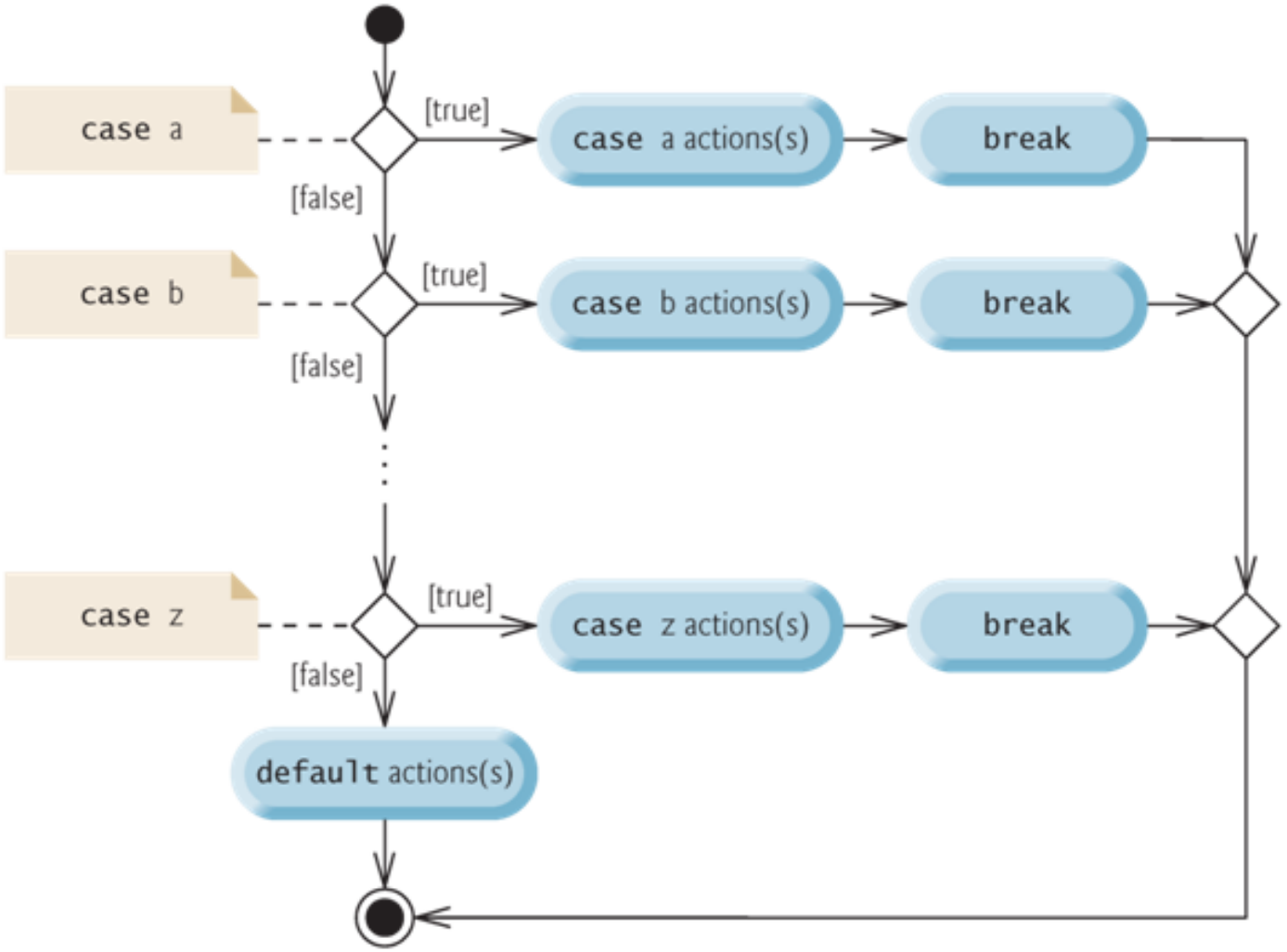
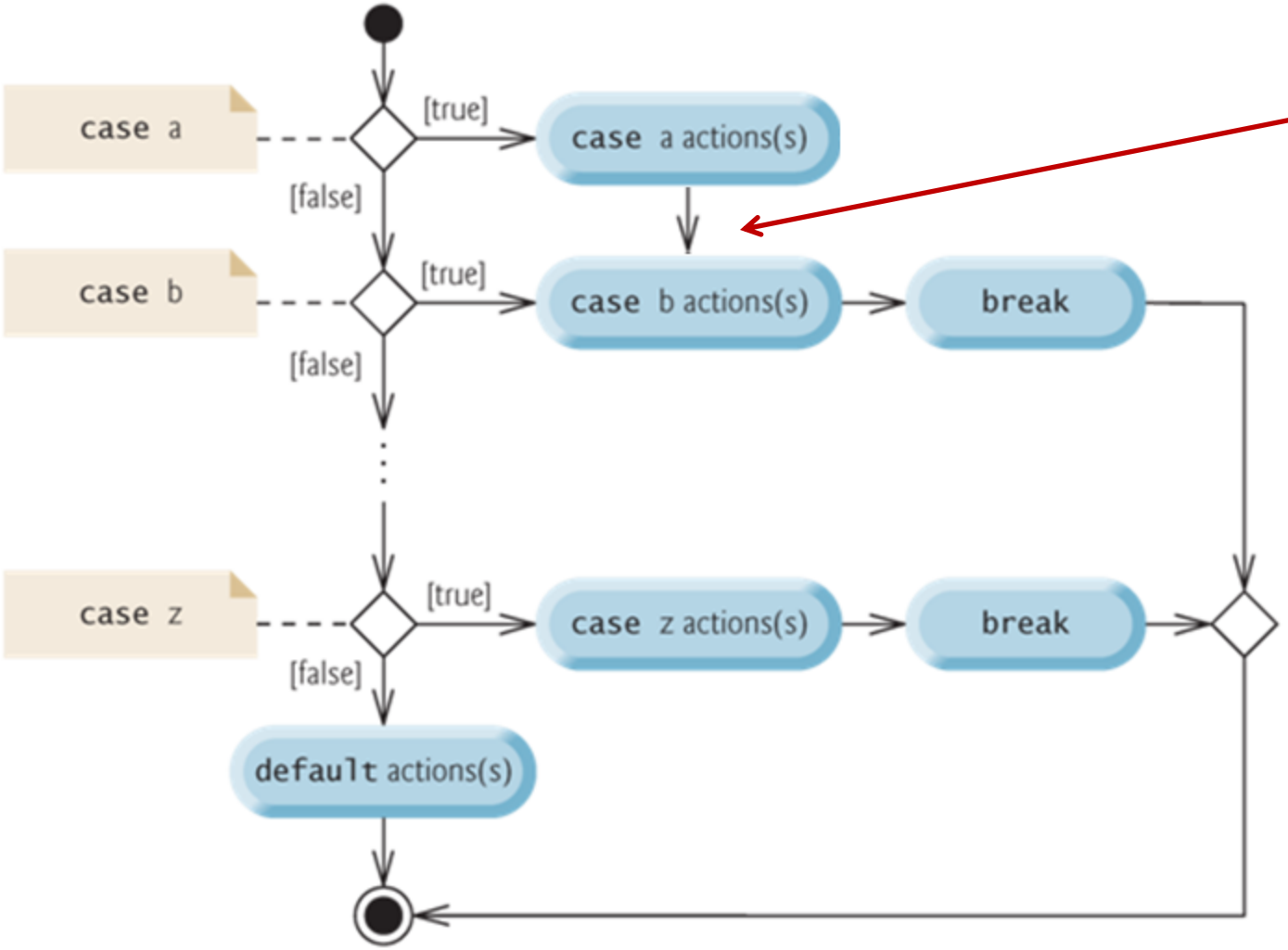


Fig. 5.11 | `switch` multiple-selection statement UML activity diagram with `break` statements.



Falling down without break

Fig. 5.11 | `switch` multiple-selection statement UML activity diagram with `break` statements.

switch Multiple-Selection Statement

- As of Java SE 7, you can use Strings or an enum in a switch statement's controlling expression and in case labels as in:

```
switch (city) {
   case "Maynard":
      zipCode = "01754";
      break;
   case "Marlborough":
      zipCode = "01752";
      break;
   case "Framingham":
      zipCode = "01701";
      break;
}
```

Structured Programming Summary (Cont.)

- Repetition is implemented in one of three ways:
  - while statement
  - do…while statement
  - for statement
- The while statement is sufficient to provide any form of repetition. Everything that can be done with do…while and for can be done with the while statement.

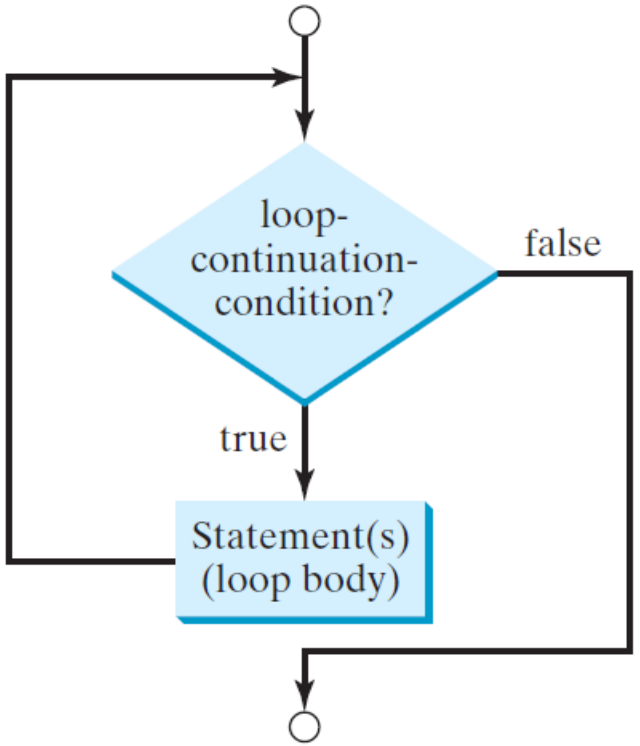
Basic Loops (Repetitions)

1) WhileStatement: 0+ Loop (while repetition)
2) DoStatement: 1+ Loop (do-while repetition)
3) ForStatement: Stepwise Increment Loop (for repetition)
4) ForEachStatement: Enhanced for loop, interative in a collection (for each repetition)

1) WhileStatement: 0+ Loop (while repetition)

Syntax:

```
while (condition)      // loopContinuationCondition
    statement          // ; is or                    nt
```



Normally:

```
initialization;
while (condition) {
    statements
}
```

2) DoStatement: 1+ Loop (do-while repetition)

Syntax:

```
do
    statement          // ; is one part of a simple statement
while (condition);
```

It is equivalent to

```
statement
while (condition)
   statement
```

Normally:

```
initialization;
do {
    statements
} while (condition);
```

3) ForStatement: Stepwise Increment Loop (for repetition)

Syntax:

```
for (initialization; condition; increments)
    statement          // ; is one part of a simple statement
// initialization like:  n = 0  or int  i = 0    or   a = xxx, b = nnn, c = yyyy
// increments like:  i++   or   i++, a = xxx, b = nnn, c = yyyy
```

Normally:

Local initialization

```
initializationA;
for (initialization; condition; increments) {
  statements
}
```

4) ForEachStatement: Enhanced for loop, Iterative in a collection (for each repetition)

Syntax:

```
for (type e : collection)
    statement          // ; is one part of a simple statement
```

Normally:

```
initialization;
for (type e : collection) {
  statements
}
```

break and continue Statements

- The break statement, when executed in a while, for, do…while or switch, causes immediate exit from that statement.
  - Execution continues with the first statement after the control statement.
  - Common uses of the break statement are to escape early from a loop or to skip the remainder of a switch.

- The continue statement, when executed in a while, for or do…while, skips the remaining statements in the loop body and proceeds with the next iteration of the loop.

```java
public class BreakTest {
   public static void main (String[] args) {
      int count;
      for (count = 1; count <= 10; count++) {
         if (count == 5) break;
         System.out.printf( "%d ", count );
      }
      System.out.printf( "\nBroke out of loop at count = %d\n", count );
   }
}
```

```
1 2 3 4
Broke out of loop at count = 5
```

```java
public class ContinueTest {
   public static void main (String[] args) {
      for (int count = 1; count <= 10; count++) {
         if (count == 5) continue;
         System.out.printf( "%d ", count );
      }
      System.out.println( "\nUsed continue to skip printing 5" );
   }
}
```

```
1 2 3 4 6 7 8 9 10
Used continue to skip printing 5
```

Examples continued in series of WarmUp04 notes …