11 The `switch` Statement

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

October 8, 2018

Outline

Reading Characters from a Stream

- When we have a text file, such as a Java source file or an HTML file, we want to read the file character by character.
- For reading characters from a text file, we use the *FileReader* class.

```
FileReader r = new FileReader("MyClass.java");
```

- For reading characters from an input stream, we use the *InputStreamReader* class.

```
InputStreamReader r = new InputStreamReader(System.in);
```

- Both classes are in the *java.io* package.
- The following code echoes the input characters to the standard output.

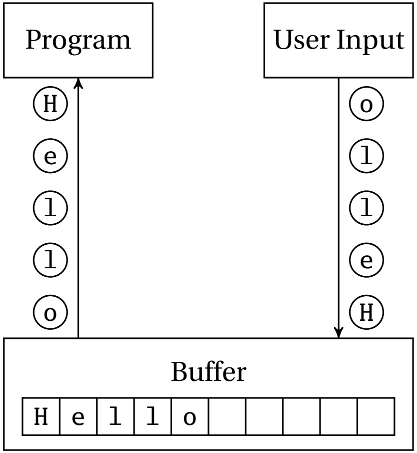
```
int c;
while ( (c = r.read()) != -1 ) // check the end-of-file (EOF) condition
    System.out.print((char)c);
```

- A Ctrl-Z (Windows) or a Ctrl-D (Unix) generates the EOF for console input.

Input Buffers

- If we echo the standard input stream *System.in* to the output, we will get:

```
Hello, there.  I would Enter
Hello, there.  I would
like a Enter
like a
bag of potatoes. Enter
bag of potatoes.
Ctrl-Z or Ctrl-D
```



- The *System.in* stream is *buffered*. The input characters will not be available until the user press Enter.
- The input characters are first collected into a buffer.
  - Full-buffered: the buffer is *flushed* when it is full.
  - Line-buffered: the buffer is *flushed* on a newline.

Multi-way Selection

- Sometimes, a program needs to choose among several alternatives. The choice is made by a case analysis on a value.
- For example, we need to increment the counter of either blank, newline or other characters based on the input:

```
InputStreamReader r = new InputStreamReader(System.in);
int blankCnt = 0, newlineCnt = 0, otherCnt = 0;
int c;
while ( (c = r.read()) != -1 && c != '#' ) {
    if ( c == '␣' ) ++blankCnt;
    else if ( c == '\n' ) ++newlineCnt;
    else ++otherCnt;
}
```

- The case analysis is fixed on the value of *c*, this branching pattern is quite common, and worth being a primitive of the language.

The switch Statement

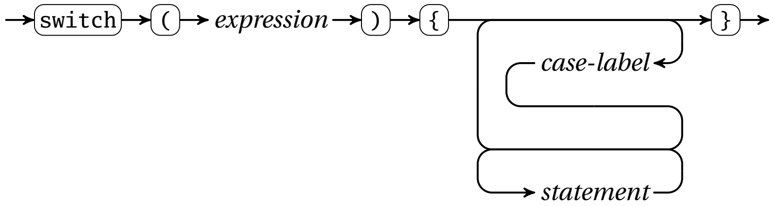
- The `switch` statement transfers control to a *case-label* within its body. It has the following form:

```
switch ( expression ) {
    case constant_1:   statement-list_1
    case constant_2:   statement-list_2
    ...
    case constant_n:   statement-list_n
    default:           statement-list_d
}
```

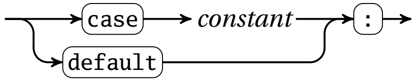
- If the *expression* evaluates to $constant_i$, then the control is transferred to the "`case` $constant_i$" case-label.
- If no constant is matched, then the control is transferred to the `default` case-label.

Syntax Diagram of switch

switch statement



case-label



Converting if ... else if to switch

- The character counting program can use the switch statement:

```
while ( (c = r.read()) != -1 && c != '#' )
    switch ( c ) {
        case '␣':
            ++blankCnt;
            break;
        case '\n':
            ++newlineCnt;
            break;
        default:
            ++otherCnt;
    }
```

- The type of the *expression* to switch on must be char, byte, short, int or *String*.
- No two case constants within the same switch statement can have the same value.

break Statements in a switch Body

- A `break` statement in the `switch` body exits the body, just like in a loop body. So, in the previous example, the `break` statements exit the `switch` body, not the `while` body.
- Case-labels (including `default`) are just labels. They do not affect the execution sequence, specifically, they do not stop the previous statements to exit the `switch` body.
- You must use `break` to exit, otherwise, the execution flow continues.
- Multiple case-labels can appear in front of a statement that multiple cases can have the same processing.
- The `default` case-label can be omitted. If present, at most once. If `default` is omitted and there is no case matched, the entire `switch` body is skipped.

Counting Vowels

The following program counts the number of each vowel in character input stream *r*. It must handle upper and lower case vowels.

```
InputStreamReader r = new InputStreamReader(System.in);
int aCnt = 0, eCnt = 0, iCnt = 0, oCnt = 0, uCnt = 0;
int ch;
while ( (ch = r.read()) != -1 && ch != '#' ) {
    switch ( ch ) {
        case 'a': case 'A':  aCnt++;  break;
        case 'e': case 'E':  eCnt++;  break;
        case 'i': case 'I':  iCnt++;  break;
        case 'o': case 'O':  oCnt++;  break;
        case 'u': case 'U':  uCnt++;  break;
    } // end of switch
} // end of while loop
```

International Dialing Codes — Nested switch

International dialing codes are prefix codes. We decode them following a *tree*.

```
StringReader r = new StringReader(code);
switch ( r.read() ) {
case '1':
   region = "US␣or␣Canada"; break;
case '3':
   switch ( r.read() ) {
   case '0':
      region = "Greece"; break;
   case '1':
      region = "Netherlands"; break;
   } break;
```

```
case '8':
   switch ( r.read() ) {
   case '6':
      region = "China"; break;
   case '5':
      switch ( r.read() ) {
      case '2':
         region = "Hong␣Kong"; break;
      case '3':
         region = "Macau"; break;
      } break;
     } break;
}
```

Reading Homework

Textbook

- Section 3.1 – 3.2, 3.10.

Internet

- Standard streams (`http://en.wikipedia.org/wiki/Standard_streams`).
- Switch statement (`http://en.wikipedia.org/wiki/Switch_statement`).

Self-test

- 3.31 – 3.34 (`http://tiger.armstrong.edu/selftest/selftest9e?chapter=3`).