More Selection

Executing Statements Selectively

Chap. 7
(Read §7.1-7.4 & Part of Picture:
Boolean Logic and Digital Design)

Review

We've seen that Java's if statement permits a statement to be executed selectively:

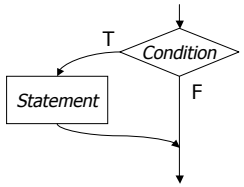
```
if (Expression)
  Statement1
[ else
  Statement2 ]
```

where expression is usually a boolean expression called a *condition*.
From this, the Java if statement can have three different forms:

The Simple if

The first form has no **`else`** or *Statement*$_2$, and is called the *simple if*:

```
if (Condition)
  Statement
```

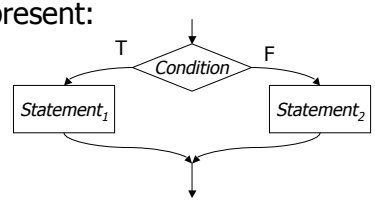


If *Condition* is true, *Statement* is executed; otherwise *Statement* is skipped.

The Two-Branch if

In the second form of if, the **`else`** and *Statement*$_2$ are present:

```
if (Condition)
  Statement1
else
  Statement2
```

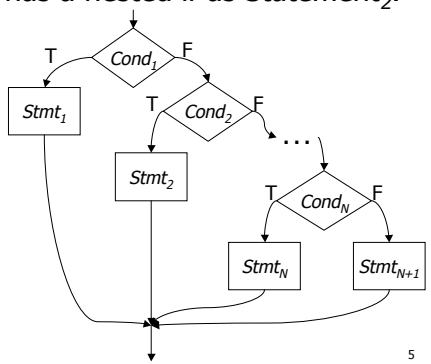


If *Condition* is true, *Statement*$_1$ is executed and *Statement*$_2$ is skipped; otherwise *Statement*$_1$ is skipped and *Statement*$_2$ is executed.

The Multi-branch if

The if's final form has a nested if as *Statement*$_2$:

```
if (Cond1)
  Stmt1
else if (Cond2)
  Stmt2
...
else if (CondN)
  StmtN
else
  StmtN+1
```



Some Potential Problems

1. If x is 5, y is 6, z is 0, what value is assigned to z by:

```
if (x > 5)
  if (y > 5)
    z = x + y;
else
  z = x - y;
```

If this is evaluated as

```
if (x > 5)
  if (y > 5)
    z = x + y;
else
  z = x - y;
// z = ______
```

If this is evaluated as

```
if (x > 5)
  if (y > 5)
    z = x + y;
  else
    z = x - y;
// z = ______
```


This is called the *dangling-else* problem and is resolved by the rule:

> *In a nested* **`if`** *statement, each* **`else`** *is matched with the nearest preceding unmatched* **`if`**.

If we want the else matched with the outer if, enclose the inner if in _____:

```
if (x > 5)
__
  if (y > 5)
    z = x + y;
__
else
  z = x - y;
```

or give the inner if an _____:

```
if (x > 5)
  if (y > 5)
    z = x + y;
  __________
else
  z = x - y;
```

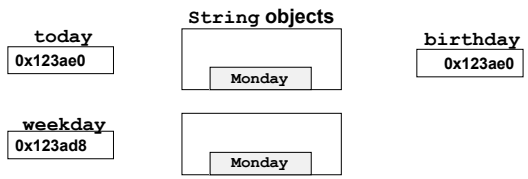

2. Consider the following declarations

```
String
  today = new String("Monday"),
  weekday = new String("Monday"),
  birthday = today;
```

What output will be produced by the following?

```
if (today == weekday)
  theScreen.println("Work hard");
if (today == birthday)
  theScreen.println("Happy birthday");
```

Output:



`today == weekday`? 0x123ae0 == 0x123ad8?

_____

`today == birthday`? 0x123ae0 == 0x123ae0?

_____

Relational operators compare addresses in handles, not the values of the objects they refer to.

Corollary: Classes should provide methods for comparing objects – e.g., String provides **`equals()`** and **`equalsIgnoreCase()`**, **`compareTo()`** and **`compareToIgnoreCase()`**.

```
if ( today.equals(weekday) )
  theScreen.println("Work hard");
if ( today.equals(birthday) )
  theScreen.println("Happy birhday");
```

Output:

Using Selection

Selection is useful anytime you want to execute a statement under particular circumstances.

Example: Suppose we need a method that, given the number of a day of the week (1-7), computes its corresponding name (Sunday-Saturday)?

Algorithm

0. Receive dayNumber.

1. If dayNumber == 1:
   Return "Sunday".
   Else if dayNumber == 2:
   Return "Monday".
   Else if dayNumber == 3:
   Return "Tuesday".
   Else if dayNumber == 4:
   Return "Wednesday".
   Else if dayNumber == 5:
   Return "Thursday".
   Else if dayNumber == 6:
   Return "Friday".
   Else if dayNumber == 7:
   Return "Saturday".
   Else
   Display an error message, and return "".

Coding 1

Such an algorithm can be coded using a multi-branch if:

```
public static String dayName(int dayNumber)
{
   if (dayNumber == 1)
      return "Sunday";
   else if (dayNumber == 2)
      return "Monday";
   else if (dayNumber == 3)
      return "Tuesday";
   else if (dayNumber == 4)
      return "Wednesday";
   else if (dayNumber == 5)
      return "Thursday";
   else if (dayNumber == 6)
      return "Friday";
   else if (dayNumber == 7)
      return "Saturday";
   else
   {
      System.err.println(
               "\n** DayName: invalid day number");
      return "";
   }
}
```

13

Drawback

The multi-branch if has _____*execution* time:

- Computing "Sunday" requires ___ comparison(s)
- Computing "Monday" requires ___ comparison(s)
- ...
- Computing "Saturday" requires ___ comparison(s)

 Computations that are "later" in the if take longer.

There are situations where the time to select one of many statements must be _____.

14

A Solution

The _____ statement provides an alternative:

```
public static String dayName(int dayNumber)
{
   ____________________________
   ___
      ____________________________
      ____________________________
      case 3: return "Tuesday";
      case 4: return "Wednesday";
      case 5: return "Thursday";
      case 6: return "Friday";
      case 7: return "Saturday";
      ___________
         System.err.println(
                  "\n** dayName: invalid day number");
         return "";
   ___
}
```

Need not be in order nor consecutive

15

The switch Statement

The switch statement provides multi-branch selection, but guarantees _____*time,* regardless of which branch is selected.

Thus, the time to select

```
return "Saturday";
```

is identical to the time to select

```
return "Sunday";
```

if a switch statement is used to select them.

16

The switch Statement (ii)

Pattern:

```
switch (Expression)
{
 caseList1  StatementList1
 caseList2  StatementList2
 ...
 caseListN  StatementListN
 default:   StatementListN+1
}
```

where *Expression* is an _____-compatible expression, each *caseList* is one or more *cases* of this form:

```
case ConstantValue :
```

and each *StatementList* usually ends with a _____ or_____ statement.

17

Example

Switch statements can use any *integer-compatible type*:

```
public static double
   straightPercentageCutOff(char letterGrade)
{
  switch(letterGrade)
  {
    case 'A': return 90.0;
    case 'B': return 80.0;
    case 'C': return 70.0;
    case 'D': return 60.0;
    case 'F': return 0.0;
    default:
      System.err.println(" \n** Invalid letter grade: "
              + letterGrade
              + " received by straightPercentageCutOff\n"
              + "-- returning 999");
      return 999;
  }
}
```

They _____ with **`string`** or **`double`** values.

18

In class example -- inverse of **`straightPercentageCutOff()`**:
numeric score letter grade

```
public static char letterGrade(double
score)
{

}
```

19

Another Restriction

To use the switch, the common algorithm pattern is:

- if (*expression* == *CONSTANT1*)
- {*statementlist1*}
- else if (*expression* == *CONSTANT2*)
- {*statementlist2*}
- ...

The pattern of a switch statement used to implement it is:

switch (*expression*)
{
  case $CONSTANT_1$:
    $statementlist_1$
  case $CONSTANT_2$:
    $statementlist_2$
  …
  case $CONSTANT_n$:
    $statementlist_n$
  default:
    $statementlist_{n+1}$
}

20

Warning

C++ switch statements exhibit _____ *behavior*.

1. *expression* is evaluated.
2. If *expression* == $CONSTANT_i$, control jumps to the $statementlist_i$ associated with $CONSTANT_i$.
3. Control _____within the switch statement until:
   a. The end of the switch is reached;
   b. A **`break`** is executed, terminating the switch;
   c. A **`return`** is executed, terminating the function; or
   d. Execution is terminated, e.g., with **`exit()`**.

Example

What will the following display, if the value of **`dayNumber`** is 4?

```
switch(dayNumber)
{
  case 1:  theScreen.print("Sunday");
  case 2:  theScreen.print("Monday");
  case 3:  theScreen.print("Tuesday");
  case 4:  theScreen.print("Wednesday");
  case 5:  theScreen.print("Thursday");
  case 6:  theScreen.print("Friday");
  case 7:  theScreen.print("Saturday");
  default: theScreen.println("Error!");
}
```

Output:

Solution

To avoid the "drop-though" behavior, we need to add _____ at the end of each case:

```
switch(dayNumber)
{
  case 1: cout << "Sunday";
          _________
  case 2: cout << "Monday";
          _________
  case 3: cout << "Tuesday";
          _________
  case 4: cout << "Wednesday";
          break;
  case 5: cout << "Thursday";
          break;
  case 6: cout << "Friday";
          break;
  case 7: cout << "Saturday";
          break;
  default: cout << "Error!" << endl;
}
```

Output when **`dayNumber`** is 4:

Selection: When to use **`switch`**

Use the switch statement for selection when

- You are comparing <u>integer-compatible types</u> (i.e., int, long, short, char, ...); *and*
- Your algorithm is of the form:

  if (*expression* == $CONSTANT_1$) $statementlist_1$
  else if (*expression* == $CONSTANT_2$) $statementlist_2$
  …
  else if (*expression* == $CONSTANT_n$) $statementlist_n$
  else $statementlist_{n+1}$

Selection: When to use `if`

Use the if statement when

- You are comparing non-integer-compatible types (i.e., double, string, ...); *or*
- Your algorithm is of the more general form:

  if (*condition*$_1$) *statementlist*$_1$
  else if (*condition*$_2$) *statementlist*$_2$
  …
  else if (*condition*$_n$) *statementlist*$_n$
  else *statementlist*$_{n+1}$

  where the *condition*$_i$ don't all have the form *expression* == *CONSTANT*$_i$ with the *expression* the same in each condition.

Example (Lab 7):

1. Menu:

```
Please enter:
   + to add two numbers;
   - to subtract two numbers;
   * to multiple two numbers; or
   / to divide two numbers.
-->
```

2. Read a choice
3. Use a(n) _____ (switch or if) statement to process the choice

Multi-Branch Selection: Conditional Expressions

- Thereis a _____ operator: _____
  - it takes _____ operands
- Syntax:

  `condition ? expression`$_1$ **`: expression`**$_2$

  where:
  - **`condition`** is a boolean expression
  - **`expression`**$_1$ and **`expression`**$_2$ are of compatible types

Example: Find the smaller of two numbers:

```
public static int largerOf(int v1, int v2)
{
  return _________  _________  _________;
}
```

Condition

Value returned if condition is true

Value returned if condition is false

Example: Print a date – e.g., 10/21/02, 11/01/02

```
theScreen.print(
          month + "/" +
          _________________________ + day
          _________________________+ year );
```

Summary

- Java provides two selective execution statements:
  - The if statement.
  - The switch statement.
- The if statement is more general and can be used to solve any problem requiring selective behavior.
- The switch is more specialized, since it can only be used in special circumstances (equality comparisons), and on certain data types (integer-compatible).
- Java also has a ternary operator

  `? :`

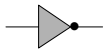
  used to form *conditional expressions* that can be used within other expressions – somewhat like putting an if statement inside an expression.

Part of the Picture: Boolean Logic & Digital Design

- Arithmetic operations performed by the CPU are carried out by logic circuits made up of three basic electronic components which mimic logical operators:

AND gate 

OR gate 

NOT gate (inverter) 

- Logic circuits can be represented by boolean expressions.
- Basic axioms of Boolean algebra can be used to simplify these circuits.

Circuit Design: A Binary Half-Adder

- Truth table

	0	1
0	0	1
1	1	10

digit1	digit2	carry	sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

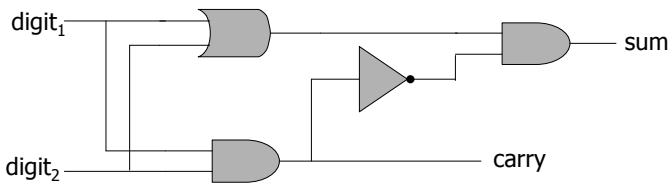
- Boolean expression equivalent:

```
boolean carry = digit1 && digit2,
        sum = (digit1 || digit2) &&
              !(digit1 && digit2);
```



```
boolean carry = digit1 && digit2,
        sum = (digit1 || digit2) &&
              !(digit1 && digit2);
```

- Digital circuit equivalent:



- Note binary half-adder class, source code, Figure 7.9, test driver Figure 7.10