Conditional Execution

Chapter 4
Sequential and Block Statements

Statements are executed one at a time and in order.
Logical Operators

Result is boolean-valued: true or false

- `==` equal to
- `!=` not equal to
- `>` greater than
- `<` less than
- `>=` greater than or equal to
- `<=` less than or equal to
- `&&` logical and
- `||` logical or
- `!` logical not

<table>
<thead>
<tr>
<th>E₁</th>
<th>E₂</th>
<th>E₁ &amp;&amp; E₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>f</td>
<td>f</td>
</tr>
<tr>
<td>f</td>
<td>t</td>
<td>f</td>
</tr>
<tr>
<td>t</td>
<td>f</td>
<td>f</td>
</tr>
<tr>
<td>t</td>
<td>t</td>
<td>t</td>
</tr>
</tbody>
</table>

| E₁ | E₂ | E₁ || E₂ |
|----|----|------|
| f  | f  |   f   |
| f  | t  |   t   |
| t  | f  |   t   |
| t  | t  |   t   |

<table>
<thead>
<tr>
<th>E</th>
<th>!E</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>f</td>
</tr>
<tr>
<td>f</td>
<td>t</td>
</tr>
</tbody>
</table>
**if-Statement**

*Executed if true*

```java
if (expression)
    statement;
```

```
if (line33 < line34)
    line37 = line36 - line35;
```

```
if (income >= 1400 || interest > 750)
{
    mustFile = true;
    deductions = 1;
    adjustedIncome = line37;
}
```

// logical error:
// null statement

```java
if (a < b)
```

Short Circuit Evaluation

A notational convenience

- if (e1 && e2)       // if e1 == false, e2 is not evaluated
- if (e1 || e2)       // if e1 == true, e2 is not evaluated

if (e1 && e2)
  statement;
if (e1)
  statement;

if (c != null && c.isVisible()) // avoid null pointer exception
  // whatever

if (n != 0 && x / n > 1.0)     // avoid / by 0 exception
  // whatever
if-else Statement

Choose one statement

```java
if (expression)
    statement-1;
else
    statement-2;
```

```java
if (line18 > line19)
    taxOwed = line18;
else
    refund = line19;
```
Nested Conditional Statement

Unlimited nesting depth

```c
if (expression-1)
    if (expression-2)
        statement-1;
    else
        statement-2;
else
    if (expression-3)
        statement-3;
    else
        statement-4;
```
else attaches to the last if

```cpp
// error
if (expression-1)
  if (expression-2)
    statement-1;
else
  statement-2;

if (expression-1)
{
  if (expression-2)
    statement-1;
}  
else
  statement-2;
```
if-else-if Ladder

Executes the first statement following a true expression

if (expression-1)
    statement-1;
else if (expression-2)
    statement-2;
else if (expression-3)
    statement-3;
    .
    .
else if (expression-m)
    statement-m;
else
    statement-n
Fencepost Loops

Loops containing multiple but unbalanced tasks

```java
for (int i = 0; i < MAX; i++)
    if (i < MAX - 1)
        System.out.println(post[i] + " ");
    else
        System.out.println(post[i]);

System.out.println(post[0]);
for (int i = 1; i < MAX; i++)
    System.out.println(" "+ post[i]);
```

```java
for (int i = 0; i < MAX - 1; i++)
    System.out.println(post[i] + " ");
System.out.println(post[MAX - 1]);
```
Conditional Expressions

Based on the Conditional Operator ?:

- \((\text{expr } 1) \ ? \ (\text{expr } 2) : (\text{expr } 3)\)
  - If expr 1 is true, expr 2 is the value of the expression
  - If expr 1 is false, expr 3 is the value of the expression
  - Parentheses are not syntactically required but are typically used because ? has a high precedence

- Examples:
  - \(\text{max} = (x > y) \ ? \ x : y;\)
  - \(\text{min} = (x < y) \ ? \ x : y;\)
  - \(\text{index} = (\text{index}+1 == \text{size}) \ ? \ 0 : ++\text{index};\)

- Use ?: if (rule of thumb)
  - the sub-expressions are short enough for the statement to fit on one line, and
  - the statement produces a value
Object Equality

Address vs. content

When applied to objects, the == operator compares the contents of the referenced variables or object handles (i.e., it compares the memory address of two objects)

```java
p0 = new Point(5, 10);
p1 = new Point(5, 10);
p2 = p1;
p3 = new point(1, 2);
```

- `p0 == p1` is false; 123 ≠ 127
- `p1 == p2` is true; 127 = 127
- `p0.equals(p1)` is true; 5 = 5 and 10 = 10
- `p0 == p3` is false; 123 ≠ 131
- `p0.equals(p3)`; 5 ≠ 1 and 10 ≠ 2
Strings: equals() vs. ==

The string literal pool

- String literals (string constants appearing in a program) are entered into a "pool" at compile time
  - References to identical string literals refer to a single object
  - Strings created during execution are not placed in the pool

```java
String s1 = "Hello world";
String s2 = "Hello world";
String s3 = "Hello";
s3 += " world";
```
Min / Max Loops

Finding the smallest / largest value

```java
int max; int min; int next;
min = max = first value;
loop:
{
    next = next value;
    if (next > max)
        max = next;
    if (next < min)
        min = next;
} 
int max = Integer.MAX_VALUE ;
int min = Integer.MIN_VALUE ;
int next;
loop:
{
    next = next value;
    if (next > max)
        max = next;
    if (next < min)
        min = next;
}
```
Exception Handling Steps

Java’s™ keywords

- Claiming an exception
  - `throws`: Introduces a comma-separated list of exceptions that a method might throw

- Throwing an exception
  - `throw`: Launch an exception object to indicate that an exception situation has occurred

- Catching an exception
  - `try`: Define a block of code that anticipates exceptions
  - `catch`: Define a block of code that deals with a specific exception
  - `finally`: Define a block of code that executes regardless of whether an exception is generated or not
**throws and Exception Lists**

Claiming or advertising errors with an *exception specification*

- **`throws`** introduces a comma-separated list of possible exceptions that may be generated within a method
- Checked exceptions must be advertised or claimed
- Unchecked exceptions are not advertised or claimed

```java
public void push(int data) throws Overflow {
    ...
}

public int pop() throws Underflow {
    ...
}
```
throwing Exceptions

Announcing errors

- An exception object is instantiated, constructed, and launched or thrown (manipulated by a thread)
- Execution resumes somewhere but not following the throw statement

```java
int pop() throws Underflow
{
    if (sp > 0)
        return StackMem[--sp];
    else
        throw new Underflow("EMPTY Stack");
}
```
try and catch Blocks

Dealing with errors

- **try** block encloses code that may generate an exception
- **catch** block(s) deal with specific exceptions
  - Must catch all checked exceptions that are not claimed with **throws**
  - Can catch all exceptions that can be thrown
- When an exception is caught
  - Warn the user, deal with the exception, return the program to a known or stable state where execution can continue
  - Warn the user, clean up, perform an orderly shut-down

```java
try
  {  S.push(i);
      S.pop();
  }
catch(Overflow E)
  { System.err.print("ERROR: ");
      E.Display();
  }
catch(Underflow E)
  { System.err.print("ERROR: ");
      E.Display();
  }
```
Program Flow with Exceptions

Tracing execution flow when exception occur

```java
public static void function(myStack S) throws Overflow, Underflow
{
   for (int i = 0; i <= 100; i++)  // one too many
      S.Push(i);
}

class myStack{
   Stack s;

   public void Push(int i){ s.push(i); }

   public int Pop(){ return s.pop(); }

   public boolean IsEmpty(){ return s.isEmpty(); }
}

public static void main(String args[ ])
{
   myStack S = new myStack( );

   try {
      function(S);
   }
   catch(Overflow E) {
      System.out.print("ERROR: ");
      E.Display( );
   }
}
```
Program Flow Illustrated

Where does execution resume?

```
try {
    push()
}
catch (Overflow E) {
}
```

```
function() {
} catch (Overflow E) {
}
```

```
push() throws Overflow {
}
```

```
function() {
    push()
}
```

```
push() throws Overflow {
}
```