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

if-Statement
Executed if true

if (expression)
  statement;

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

if (income >= 1400 || interest > 750) // logical error
  deductions = 1;
  adjustedIncome = line37;
  if (a < b) ; // null statement

if-else Statement
Choose one statement

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

if (line18 > line19)
  taxOwed = line18;
else
  refund = line19;

Short Circuit Evaluation
A notational convenience

if (e1 && e2) // e1 == false, e2 is not evaluated
if (e1 || e2) // e1 == true, e2 is not evaluated
if (c != null && c.isVisible()) // avoid null pointer exception
  // whatever
if (n != 0 && x / n > 1.0) // avoid / by 0 exception
  // whatever
**Nested Conditional Statement**

Unlimited nesting depth

```
if (expression-1)
  if (expression-2)
    statement-1;
  else
    statement-2;
else
  if (expression-3)
    statement-3;
  else
    statement-4;
```

**The Dangling else Problem**

```
else attaches to the last if
```

```
// 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...
else if (expression-m)
  statement-m;
else
  statement-n
```

**Fencepost Loops**

Loops containing multiple but unbalanced tasks

```
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]);
```

```
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 `?:`

```
(expr 1) ? (expr 2) : (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:**
- `max = (x > y) ? x : y`;
- `min = (x < y) ? x : y`;
- `index = (index+1 == size) ? 0 : ++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)

```
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

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;
```

```java
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

```java
try
{
    S.push(i);
    S.pop();
}
catch (Overflow E)
{
    System.err.print("ERROR ");
    E.Display();
}
```

```java
catch (Underflow E)
{
    System.err.print("ERROR ");
    E.Display();
}
```

- 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
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();
    }

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?