CSPP 511-01: Introduction to Object-Oriented Programming

Harri Hakula

Ryerson 256, tel. 773-702-8584
hhakula@cs.uchicago.edu

July 3, 2000
Outline

- Flow of Control
- Statements and Blocks
- Boolean Expressions
- Branching Statements
- Loop Statements
Flow of Control

Flow of control refers to the order in which actions are performed in programs. In simple programs every statement is executed in the sequential order in which they were written.

In normal programs, two kinds of statements are used:

- A branching statement chooses one action from a list of two or more possible actions.
- A loop statement repeats some action again and again until some stopping condition is met.
Statements and Blocks

The two basic statements are expression statements and declaration statements. As we have seen, declaration statements can appear anywhere inside a block. All statements are terminated by the semicolon.

The shortest statement is the empty statement - a semicolon by itself.
Curly braces, { and }, group zero or more statements into a block. A block can be used where any single statement is allowed because a block is a statement, albeit a compound one.

... 
{
  // New block
  ...
... 
}

/* End of block followed by an empty statement */
...
...
Scope of a Variable

Local variables exist only as long as the block containing their declaration is executing. The block is their scope. Consider:

```c
...{
    // New block
    int a = 1;
    ...
}
    // End of block
a += 1;  // Error: Outside of scope
...
Boolean Expressions

A boolean expression is an expression that is either true or false. The value of a boolean expression can be stored in a variable of type boolean. The simplest boolean expressions are comparisons of two things such as numbers.

    boolean answer = 1 < 2;
    // answer is always true!
# Comparison Operators

<table>
<thead>
<tr>
<th>Math</th>
<th>Name</th>
<th>JAVA</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>equal to</td>
<td>==</td>
<td>answer == true</td>
</tr>
<tr>
<td>≠</td>
<td>not equal to</td>
<td>!=</td>
<td>answer != true</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
<td>&gt;</td>
<td>two &gt; 1</td>
</tr>
<tr>
<td>≥</td>
<td>greater than or equal to</td>
<td>&gt;=</td>
<td>iq &gt;= 90</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than</td>
<td>&lt;</td>
<td>one &lt; two</td>
</tr>
<tr>
<td>≤</td>
<td>less than or equal to</td>
<td>&lt;=</td>
<td>two &lt;= pair</td>
</tr>
</tbody>
</table>
Comparison operators are defined only for the primitive types. Therefore, comparison of two Strings would not have the intended meaning.

For user-defined types, comparisons must be implemented separately. String API is

```java
equals (OtherString)

equalsIgnoreCase (OtherString)
```
Logical Operators

More complicated boolean expressions can be created with the use of logical operators. Here we introduce only binary operators **and** and **or** and unary **not**.

**not** (!)

! (two == three)

**and** (&&)

(two > 1) && (two < three)

**or** (||)

(two > 1) || (two < three)

**and** and **or** are conditional, because if the left hand side test determines the value, the right hand test is not evaluated.
Branching Statements

Only one way to branch is required. However, since certain situations occur commonly in programs, there are two branching statements in JAVA.

**if-else** Choose between two possible actions.

**switch** Choose an action from a set of possible actions.
If-Else

if( value >= 100 )
{ System.out.println( "OK" ); }
else if((value > 0) && (value < 100))
{ System.out.println( "Acceptable" ); }
else
{
    System.out.println( "Error: Negative." );
}
If-Else Cntd.

Syntax template:

```java
if (Boolean Expression 1)
{
    Action 1;
}
else if (Boolean Expression 2)
{
    Action 2;
}
else
{
    Default action.
}
```
Switch

int val = 2;
switch(val) {
    case 1: System.out.println( "one" );
    break;
    case 2:
    case 3: System.out.println(
        "either two or three" );
    break;
    default:
    System.out.println( "wow" );
    break;
}
Switch Cntd.

```java
switch (Controlling Expression) {
    case CaseLabel1:
        statement; ... statement;
        break;
    case CaseLabel2:
        statement; ... statement;
        break;
    default:
        statement; ... statement;
        break;
}
```
Loop Statements

A portion of the program that repeats a statement or group of statements is a loop. The statement (or group of statements) to be repeated is called the body of the loop. Each repetition of the loop body is called the iteration of the loop.

For certain cases there are special loop structures.

One must always define the entry and exit points of the loop.
While

while (Boolean Expression)
   
   Body

The Body may either be a simple statement or a compound statement.

int val = 2;
while(val < 100)
{
   ...
   val *= 2;
}
Do-While

Often it is necessary to execute the body at least once.

\[ \text{do} \]

\[
\quad \text{Body}
\]

\[ \text{while} \ (\text{Boolean Expression}); \]

\[ \text{int \ val} = 101; \]

\[ \text{do} \]

\[
\quad \{ \\
\quad \quad \cdots \\
\quad \quad \text{val} = \text{val} \times 2; \\
\quad \} \text{ while(val < 100);} \\
\]
Do-While Cntd.

We can define Do-While in terms of While:

```plaintext
do
    Body
while (Boolean Expression);
```

is equivalent to

```plaintext
Body
while (Boolean Expression)
    Body

where every occurrence of the Body is the same.
For

for (Initializing Action; Boolean Expression; Action After Each Iteration)

Body

int i;
for(i = 0; i < 10; ++i)
{
   System.out.println( i );
}
For cntd.

However, a functionally same loop can be written as:

```java
for(int i = 0; i < 10; ++i)
{
    System.out.println( i );
}
```

There is a subtle difference. The scopes of the variable ’i’s are different. The easy way to see this is to define for in terms of while.
For cntd.

for (Initializing Action; Boolean Expression; Action After Each Iteration)

Body

is equivalent to (Note the added block!)

{

    Initializing Action;

    while (Boolean Expression)
    {
        Body
        Action After Each Iteration;
    }
}
Exiting a Loop

There are three ways of altering the flow inside the body of a loop.

**break**  Breaks any loop.

**continue**  Jumps to the end of the body.

**System.exit(0)**  Exits the whole program.
Exiting a Loop Cntd.

```java
public class ExitLoop {
    public static void main(String args[]) {
        for(int i = 0; i < 10; ++i) {
            if(i == 5) continue;
            else if(i == 7) break;
            System.out.print(i + " ");
        }
    }
}
```

prints 0 1 2 3 4 6.