Control Structures

Boolean Expressions

- … evaluate to type `boolean`: `true` or `false`
- relational operators: `< > >= <= == !=`
  - operands can have many types, return boolean
- boolean operators: `!` (not) `& &&` (and) `| ||` (or)
- `&, |` vs `&&, ||`:
  - `&&, ||` are "lazy": only evaluate the right operand if necessary.
  - `&, |` are "active": always evaluate the right operand.
- Functions can also return `boolean`.

The `if` Statement

- syntax:

  <cond statement> ::= `if (` <bool expr> `)` <statement>

  <cond statement> ::= `if (` <bool expr> `)` <statement>

  `else` <statement>

- the body of the conditional is a single statement.
  - … but curly braces can be used to make a "compound statement"

    <statement> ::= `{`<statement>`;` <statement>`;` … `}`
  - This applies anywhere a statement can be written.

Example

```
class ConditionalExample {
    public static void main(String[] args) {
        int value = 6;

        if ( value == 6 ) {
            System.out.println("equal");
        } else if ( value < 6 ) {
            System.out.println("less");
        } else {
            System.out.println("more");
        }
    }
}
```

Compound Statements

- There isn't really an "else if" in Java.
  - The `else` applies to the single statement after it.
  - … which happens to be an `if`.
- So, it's really:

```
if ( value == 6 ) { …
} else {
    if ( value < 6 ) { …
    } else { …
    }
}
```

Compound Statements

- This is also legal:

```
if ( value == 6 )
    System.out.println( … );
```

- same as:

```
if ( value == 6 ) {
    System.out.println( … );
}
```

- The {} turn a single statement into a single (compound) statement.
- Only omit the braces if it's obvious.

The `while` Statement

- syntax:

  <while statement> ::= while (<bool expr>)
  <statement>

- Same semantics as other languages.
  - check condition, if true, run the statement, and repeat.
  - The statement will run 0 or more times.

Example

```
int count = 0;
while ( count < 5 ) {
    System.out.print(count);
    System.out.print("  ");
    count++;
}
```

- Output:

```
0 1 2 3 4
```

The `do...while` Statement

- syntax:

  <dowhile statement> ::= do <statement>

  while (<bool expr>);

- run the statement, then check condition, if true, repeat.
  - The statement will run 1 or more times.
  - no equivalent in Python

Example

```
int count = 0;
do {
    System.out.print(count);
    System.out.print("  ");
    count++;
} while ( count < 5 );
```

- Output:

```
0 1 2 3 4
```

The `for` Statement

- syntax:

  <for statement> ::= for (<statement> ; <condition> ;
  <statement>) <statement>

- The three things in the parens are:
  - initialization statement
  - continuation condition
  - "step" statement

Example

```
for ( count=0; count<5; count++ ) {
    System.out.print(count);
    System.out.print(" ");
}
```

- Output:

```
0 1 2 3 4
```

- This is equivalent to…

```
count=0;
while ( count<5 ) {
    System.out.print(count);
    System.out.print(" ");
    count++;
}
```

Others

- The `switch` statement
  - used to give alternatives for many different values of an expression
- The iterator version of `for`
  - loops for each element of an object with an iterator
  - like the Python `for` loop
  - new in Java 5.0.

User Input

- traditionally ugly in Java
  - new in Java 5.0: the `Scanner` class
- Create a scanner object with input from console
  - … then use methods that retrieve the next value of the appropriate type.
- For older versions, see the BearcatScanner package that implements enough of `Scanner` for us.

Example

```
import java.util.Scanner;

class ScannerExample {
    public static void main(String[] args) {
       Scanner sc = new Scanner(System.in);
       int i = sc.nextInt();
       double d = sc.nextDouble();

       System.out.print("First value:  ");
       System.out.println(i);
       System.out.print("Second value: ");
       System.out.println(d);

    }
}
```