Chapter 5

`if`

- `if` (*expression*)
  statement
- differences compared to Python
  - no colon
  - `else if`, not `elif`
  - statements grouped by *blocks* not indentation

if-else

```
if (expression)
    statement
else
    statement
```

if-else if-else

```
if (expression)
    statement
else if (expression)
    statement
else
    statement
```

careful – order is important!

Block Statements

```
if (expression)
{
    statement
}
else
{
    statement
}
```

Nested `if` Statements

```
if (expression)
{
    if (expression)
        statement

}
else
    statement
```

Equality and Relational Operators

operator	meaning
==	equal to
!=	not equal to
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to

lower precedence than arithmetic operators

Comparing Floating Point Values

- two values only equal (==) when all binary digits match
- probably not what you want due to rounding errors
- check if values are sufficiently close

```
final double EPSILON = 1e-14;
if (Math.abs(f1 - f2) < EPSILON)
```

Comparing Characters

- lexicographic ordering dependent upon Unicode character set
  - `'a'` is less than `'b'`
  - `'A'` is less than `'B'`
  - `'0'` is less than `'1'`
  - `'B'` is less than `'a'`

Comparing Strings

```
String name1, name2;

// returns true if name1 and name2 refer
// to the same object (same memory
// location)
if (name1 == name2) // works in Python!


// returns true if name1 and name2
// contain the same characters
if (name1.equals(name2))
```

String.compareTo

```
String name1, name2;
int result = name1.compareTo(name2);
```

- returns 0 if `name1` and `name2` contain the same characters
- returns a value < 0 if `name1` is < than `name2`
- returns a value > 0 if `name1` is > than `name2`
- based on lexicographic order

Comparing Objects

- same as comparing `String` (which are objects)
- the == operator tests if two object references are identical
- the `equals` method tests if two objects are "equal"
  - the `equals` method must be defined for the class
  - what "equals" mean is defined by the class

`null` reference

- the value of no object
- cannot invoke methods on a `null` reference
- test for `null` references using the == operator

Logical Operators

logical operator	Java	Python	Precedence
NOT	`!`	`not`	highest
AND	`&&`	`and`	middle
OR	`||`	`or`	lowest

Not (Logical Complement) Truth Table

a	! a
FALSE	TRUE
TRUE	FALSE

And/Or Truth Table

`a`	`b`	`a && b`	`a || b`
FALSE	FALSE	FALSE	FALSE
FALSE	TRUE	FALSE	TRUE
TRUE	FALSE	FALSE	TRUE
TRUE	TRUE	TRUE	TRUE

De Morgan's Laws

- `!(a && b)`
  - equivalent to: `(!a) || (!b)`
- `!(a || b)`
  - equivalent to: `(!a) && (!b)`

Prove

`!(a && b) ⇔(!a) || (!b)`

a	b	!(a&&b)	(!a)||(!b)
FALSE	FALSE	TRUE	TRUE
FALSE	TRUE	TRUE	TRUE
TRUE	FALSE	TRUE	TRUE
TRUE	TRUE	FALSE	FALSE

Short Circuit

- if the left operand solely determines result of logical operator, right operand is not evaluated
  - (just like Python)
- Example:

```
if (count != 0 && total/count > MAX)
```

Conditional / Ternary Operator

- devil spawn from C
- also know as the conditional operator

```
operand1 ? operand2 : operand3
```

- translated:

```
if(operand1 == true)
    return operand2;
else
    return operand3;
```

switch

- another conditional statement
- preferred over `if` when evaluating several discrete values as opposed to a few ranges of values
- flow of control jumps to branch matching conditional expression and continues from there

```
switch (integerExpression)
{
    case constantExpression:
        statement;
    [case constantExpression:
        statement;]
    ...
    [default:
        statement;]
}
```

```
switch (gradeNumber)
{
   case 6:
   case 7:
   case 8:
      System.out.println(“Junior High”);
      break;
   case 9:
      System.out.println(“Freshmen”);
      break;
   case 10:
      System.out.println(“Sophomores”);
      break;
   case 11:
      System.out.println(“Juniors”);
      break;
   case 12:
      System.out.println(“Seniors”);
      break;
   default:
      System.out.println(“Elementary”);
}
```

Components of a `switch`

- `case`
  - constant expression matched by switch
- `break`
  - flow of control leaves switch block
  - (without break flow on control continues into next `case`)
- `default`
  - matches everything not matched by a `case`

Enumerations

- a set of objects that represent a related set of choices
- supported by switch statement
- usually compared with == operator
- enumerations capitalized like classes; enumerated values, like constants

Example

```
public enum FilingStatus {SINGLE,
   MARRIED, MARRIED_FILING_SEPARATELY}
...
```

```
FilingStatus status =
   FilingStatus.SINGLE;
...

if(status == FilingStatus.SINGLE)
   ...
```

Post Increment and Decrement Operators

- equivalent to adding or subtracting 1
- no Python equivalent
- returns the value and then increments or decrements

```
b = 7;
a = b++; // a==7; b==8
c = b--; // c==8; b==7
```

Pre Increment and Decrement Operators

- increments or decrements and then returns the value

```
b = 7;
a = ++b; // a==8; b==8
c = --b; // c==7; b==7
```

Assignment Operators

- perform the specified operation and then assign the resulting value
- `+=, -=, *=, /=, %=`
- `x += y` is equivalent to `x = x + y`