Control Structures

The Big Picture

Blocks – defined by {}

- a block is a group of statements
  - defines the body of a class definition, a subroutine definition, a branch of an `if` statement, …
  - defines the *scope* of a variable definition (i.e. where that variable's name can be used)

Making choices – (`if` statements)

- syntax and semantics
- tricky cases
- programming with `if` statements

Scope

- *scope* refers to where a name can be used
- the scope of a variable is from the point of declaration to the end of the closest enclosing block

```
{
  System.out.println(a);
  System.out.println(b);
  int a = 10;
  {
    System.out.println(a);
    System.out.println(b);
    int b = 20;
    System.out.println(a);
    System.out.println(b);
  }
  System.out.println(a);
  System.out.println(b);
}
```

illegal statements are crossed out – symbol not found

Conditionals (`if` Statements)

- purpose: to execute statements only in certain circumstances

```
if ( condition ) {
  statements
}
```

`if` statement

execute *statements* if *condition* is true, do nothing if *condition* is false

```
if ( condition ) {
  statements
} else {
  statements2
}
```

`if-else` statement

execute *statements* if *condition* is true, execute *statements2* if *condition* is false

```
if ( condition1 ) {
  statements1
} else if ( condition2 ) {
  statements2
} else if ( condition3 ) {
  statements3
} else {
  statementsN
}
```

`else if` statement

execute *statements1* if *condition1* is true,
execute *statements2* if *condition1* is false and *condition2* is true,
execute *statements3* if both *condition1* and *condition2* are false and *condition3* is true,
otherwise execute *statementsN*

- there can be any number of `else if` clauses
- the final `else` can be omitted

Conditions

Conditions are expressions whose value is of type `boolean`.

- relational operators – result is `true` or `false`
  - test for equality or inequality: ==, !=
    - applies to primitive types (`int`, `double`, `char`, `boolean`, …)
    - ⚠ due to precision issues, not ideal for `doubles`
  - comparison: <, >, <=, >=
    - applies to numeric types (`int`, `double`, …) and `char`
- logical operators – apply to `boolean` values
  - && – and
    - `a && b` is `true` if both a and b are `true`, `false` otherwise
  - || – or
    - `a || b` is `true` if either a and b (or both) are `true`, `false` otherwise
  - ! – not
    - `!a` is `true` if a is `false` and `false` if a is `true`

⚠ for `Strings` *s1*, *s2* use *s1*`.equals(`*s2*`)` or `!`*s1*`.equals(`*s2*`)`


```
int a, b;
a = 5;
b = 10;
if ( b >= a ) {
  System.out.println("b");
  b = 2;
} else {
  System.out.println("a");
}
System.out.println(b);
```

output is

b
2

only the first branch with a true condition is done

```
int points;
points = 85;
if ( points >= 90 ) {
  System.out.println("A");
} else if ( points >= 80 ) {
  System.out.println("B");
} else if ( points >= 70 ) {
  System.out.println("C");
} else if ( points >= 60 ) {
  System.out.println("D");
} else {
  System.out.println("F");
}
```

output is

B

```
int points;
points = 85;
if ( points >= 60 ) {
  System.out.println(“D”);
} else if ( points >= 70 ) {
  System.out.println(“C”);
} else if ( points >= 80 ) {
  System.out.println(“B”);
} else if ( points >= 90 ) {
  System.out.println(“A”);
} else {
  System.out.println(“F”);
}
```

output is

D


```
if ( s.length() > 4 )
  if ( s.length() < 10 )
    System.out.println("medium");
else
  System.out.println("short");
```

```
if ( s.length() > 4 ) {
  if ( s.length() < 10 ) {
    System.out.println(“medium”);
  } else {
    System.out.println(“short”);
  }
}
```

"hi"

"hello"

"greetings, earthling!"

pro tip: while { } is not required if *statements* only contains one statement, always include { }

Programming With `ifs`

Strategy –

- recognize that the task involves doing different things in different circumstances
- get more specific – identify which variation
  - either do or don't do something
  - two alternatives
  - more than two alternatives, do nothing is not an option
  - more than two alternatives, do nothing is an option
- match the variation to the syntax
  - do or don't do → `if` statement
  - two alternatives → `if-else` statement
  - more than two alternatives → `else if` statement, with (do nothing is not an option) or without (do nothing is an option) the final `else`
- fill in the elements in the pattern
  - alternatives → *statements*
  - circumstances under which to do each alternative → *conditions*