Scarborough College
University of Toronto
Fall 1997

CSC A06F: Introduction to Computer Programming
Lecture notes 3: The elements of programs — Part 2

Reading:

- Lewis and Loftus, chapters 3 and 4.

The *if-else* statement

```
if (condition)
    statement1;
else
    statement2;
```

Just like the normal *if* statement, but if the condition is false, statement2 is executed.

One or the other is executed. Never both.

Nested ifs

Can 'nest' if-statements within each other to make more complex decisions.

```
if (age < 19)
   if (selection.equals ("beer"))
      System.out.println("Not legal.");
   else
      System.out.println("OK");
else
   System.out.println("OK");
```


Block statements

The syntax of the if-statement says it only allows one statement.

So, how do we make it do several things when the condition is true?

A **block statement** gathers a group of statements together. Java treats it like a single statement.

Use {} (brace brackets) to mark the beginning and end of the block.

E.g.,

```
if (selection.equals ("beer")) {
   int years = 19 - age;
   System.out.println("Not legal.");
   System.out.println("Come back in " + years);
}
else {
   ...
}
```

Flow of control

Normally the computer executes your program from top to bottom, one statement at a time.

The **flow of control** is *linear* .

The if-statement changes the flow of control:
a statement may or may not be executed, depending on the condition.

There are other ways to change the flow of control:

- loops
- calls to methods

Repetition and loops

A loop is a way to have the same statements executed many times in a row.

Why would you want this?

Several kinds of loop: one is the `while` loop.

The `while` loop

```
while (condition)
    statement;
```

Executes `statement` if `condition` is true.

Then goes back and checks the `condition` again. If it's still true, executes `statement` again.

Keeps 'looping' until the `condition` is false. E.g.,

```
int sum = 0;
int number = Integer.parseInt (stdin.readLine());

while ( number > 0 ) {
   sum = sum + number;
   number = Integer.parseInt (stdin.readLine());
}
```

How does the condition become true?

Another example

Complete this code fragment:

```
final int N = 10;
int sum = 0;

// this loop sums up all the integers from 1 to N
while (           ) {







}
```

What happens if the condition never becomes true?

Objects

Breaking a system up into **objects** that communicate with one another is a good way to design a system.

An object in a system is like a real-world object. It has three parts:

- identity
- state
- behaviour

Can have many objects that are similar.
I.e., , that have same behaviour, but different states and identities

A **class** is used to define an object (or a set of similar objects).

A class is a ***blueprint*** for building lots of similar objects.

Objects in Java

To use an object in Java, you need:

- a variable
- a class

Then, you have to *build* the object. E.g.,

```
String student;

student = new String("Phil Edmonds");
```

Notes:

- `student` is a variable that is declared to contain `String` objects.
- `String` is a class that is defined in the Java API.
- Different from primitive types. Must use `new` to build and initialize an object.

Assignment with objects

Can also use assignment with objects:

```
String student2;
student2 = student;
```

Although it looks the same as regular assignment, it works a little differently.

This makes `student2` contain the ***exact same object*** as `student` does. (Same identity, same everything).

Whereas assignment for primitive data types ***copies*** the data value. So you have 2 separate, but same, data values.

Using an object: calling its methods

Objects have **methods** . You can **call** these to tell the object to do something.

Examples:

```
System.out.println("Hello world!");
           ^^^^^^^^^^^^^^^^^^^^^^^


token = stdin.readLine();
              ^^^^^^^^^^


if (selection.equals("beer") {}
              ^^^^^^^^^^^^^^
```

You use the dot operator (`.`) to call a method of an object.

Syntax:

```
object-reference.method-name (parameters);
```

How calls to methods work

This makes the flow of control change. What happens?

- the computer remembers this spot in the code;
- it then starts running the statements in the method;
- when finished these, it 'returns' to the spot, and continues on from there.

Some methods of `String`

```
length()

equals(Object obj)

compareTo(String str)

indexOf (char ch)

charAt(int index)
```