Conditional Execution

Software Development 1 (F27SA)

Michael Lones

Week 2, lecture 2

Today's Lecture

- What is conditional execution?
- `if…then…else` statements
- `switch…case` statements
- The ternary operator

What is Conditional Execution?

This is where execution can take **different paths** through a program, depending upon whether a particular **condition** is true or not

- Different sections of code get executed depending upon the states of variables in the program
- Conditional execution is used extensively during programming as a way of making decisions

if…then…else

This is the most common form of conditional execution, and is found in every language

- Usually referred to as an *if statement*
- **if** some condition is true **then** do this **else** do this
- In Java:

```
if(condition)
    // execute this code
else
    // execute this code
```

if...then...else

IfThenElseDemo.java

```
public class IfThenElseDemo {
   public static void main(String[] args) {
      int age = 18;
      if(age > 40)
         System.out.println("You're old!");
      else
         System.out.println("You're young!");
   }
}
```

Terminal

```
$ java IfThenElseDemo
You're young!
```

Blocks

If you want to execute more than one statement, you need to use a block, i.e. curly braces:

```
if(condition) {
    // execute these
    // multiple statements
}
else {
    // execute these
    // multiple statements
}
```

if…then…

IfThenDemo.java

```
public class IfThenDemo {
   public static void main(String[] args) {
      int age = 18;
      if(age > 40) {
         System.out.println("You're old!");
         System.out.println("I respect you.");
      }
   }
}
```

Terminal

```
$ java IfThenDemo
```

if…then…

IfThenDemo.java

```
public class IfThenDemo {
   public static void main(String[] args) {
      int age = 18;
      if(age > 40) {
         System.out.println("You're old!");
         System.out.println("I respect you.");
      }
   }
}
```

The else part is optional

Terminal

```
$ java IfThenDemo
```

Conditions

Conditions are expressions that typically use **Boolean operators**. Commonly used ones are:

- **>** : greater than
- **<** : less than
- **>=** : greater than or equal to
- **<=** : less than or equal to
- **==** : equal to
- **!=** : not equal to

Conditions

Conditions are expressions that typically use **Boolean operators**. Commonly used ones are:

- **>** : greater than
- **<** : less than
- **>=** : greater than or equal to
- **<=** : less than or equal to
- **==** : equal to ← A common (and hard to spot) error is to use the assignment operator (=) rather than ==
- **!=** : not equal to

A quick aside: Booleans

They are called Boolean operators because they evaluate to a Boolean value (i.e. **true** or **false**)

- Which could be stored in a Boolean variable
- So, these two code fragments are equivalent:

```
int age = 18;

if(age > 40) {
   ...
}
```

```
int age = 18;
boolean isOld = age > 40;
if(isOld) {
   ...
}
```

A quick aside: Booleans

Here's an example where the result of a Boolean operator is printed out, giving a Boolean value:

IntCompare.java

```
public class IntCompare {
   public static void main(String[] args) {
      int a = 5;
      int b = 5;
      System.out.println( a == b );
   }
}
```

Terminal

```
$ java StringCompare
true
```

A quick aside: Booleans

Here's an example where the result of a Boolean operator is printed out, giving a Boolean value:

IntCompare.java

```
public class IntCompare {
   public static void main(String[] args) {
      int a = 5;
      int b = 5;
      System.out.println( a == b );
   }
}
```

Assignment

Equal to operator

Terminal

```
$ java StringCompare
true
```

Another quick aside: Strings

However, note that Strings (and other object types) can <u>not</u> be compared using operators

StringCompareOperator.java

```
import java.util.Scanner;

public class StringCompareOperator {
   public static void main(String[] args) {
      Scanner scan = new Scanner(System.in);
      String a = scan.next();
      System.out.println( a == "hello" );
   }
}
```

Terminal

```
$ java StringCompare
hello
false
```

Another quick aside: Strings

Instead, you have to use string's "**equals method**"

something that will be explained in Part 2 of SD1

StringCompareEquals.java

```
import java.util.Scanner;

public class StringCompareEquals {
   public static void main(String[] args) {
      Scanner scan = new Scanner(System.in);
      String a = scan.next();
      System.out.println( a.equals("hello") );
   }
}
```

Terminal

```
$ java StringCompare
hello
true
```

Conditions

More complex conditions can be built using these Boolean logic operators:

- **&&** "and" which is only true when **both** of its operands are true
- **||** "or" which is true when **at least one** of its operands are true
- **!** "not" which is true when its operand is false, and vice versa

Conditions

ConditionDemo.java

```
public class ConditionDemo {
   public static void main(String[] args) {
      int age = 70;
      int wealth = 150000;
      if(age>60 && wealth>100000)
         System.out.println(
           "Time to retire!");
   }
}
```

Terminal

```
$ java ConditionDemo
Time to retire!
```

Precedence (recap)

When constructing more complex expressions, you have to be aware of precedence rules

- This applies whenever operators are used, both in **logic** expressions and **mathematical** expressions
- Operators with highest precedence always get applied first, regardless of where they occur within an expression
- If you're not aware of the precedence rules, then your program might have unexpected behaviour

Precedence

Operators in order of precedence (high to low):

- ()
- ++, --
- - [minus], !
- *, /, %
- +, -
- <, <, >=, <=
- ==, !=
- &&, ||
- =, +=, -=, *=, /-

Precedence

Operators in order of precedence (high to low):

- ()
- ++, --
- - [minus], !
- *, /, %
- +, -
- <, <, >=, <=
- ==, !=
- &&, ||
- =, +=, -=, *=, /-

If you want to make sure a sub-expression gets evaluated first, then enclose it in parentheses

Precedence

Operators in order of precedence (high to low):

- ()
- ++, --
- - [minus], !
- *, /, %
- +, -
- <, <, >=, <=
- ==, !=
- &&, ||
- =, +=, -=, *=, /-

"Not" has higher precedence than "and" and "or", meaning:

`!a&&b` is not the same as

`!(a&&b)`

Precedence

Operators in order of precedence (high to low):

- ()
- ++, --
- - [minus], !
- *, /, %
- +, -
- <, <, >=, <=
- ==, !=
- &&, ||
- =, +=, -=, *=, /-

"And" and "or" have low precedence, meaning you often don't need to use parentheses:

`a>b||b==c` is the same as `(a>b)||(b==c)`

Nested `if`s

It is possible, and even common, to have an if statement inside another if statement, e.g.

```
if(age>40) {
  if(wealth>500000)
    System.out.println("Probably tory");
  else
    System.out.println("Probably not tory");
}
else
    System.out.println("Probably labour");
```

Nested `if`s

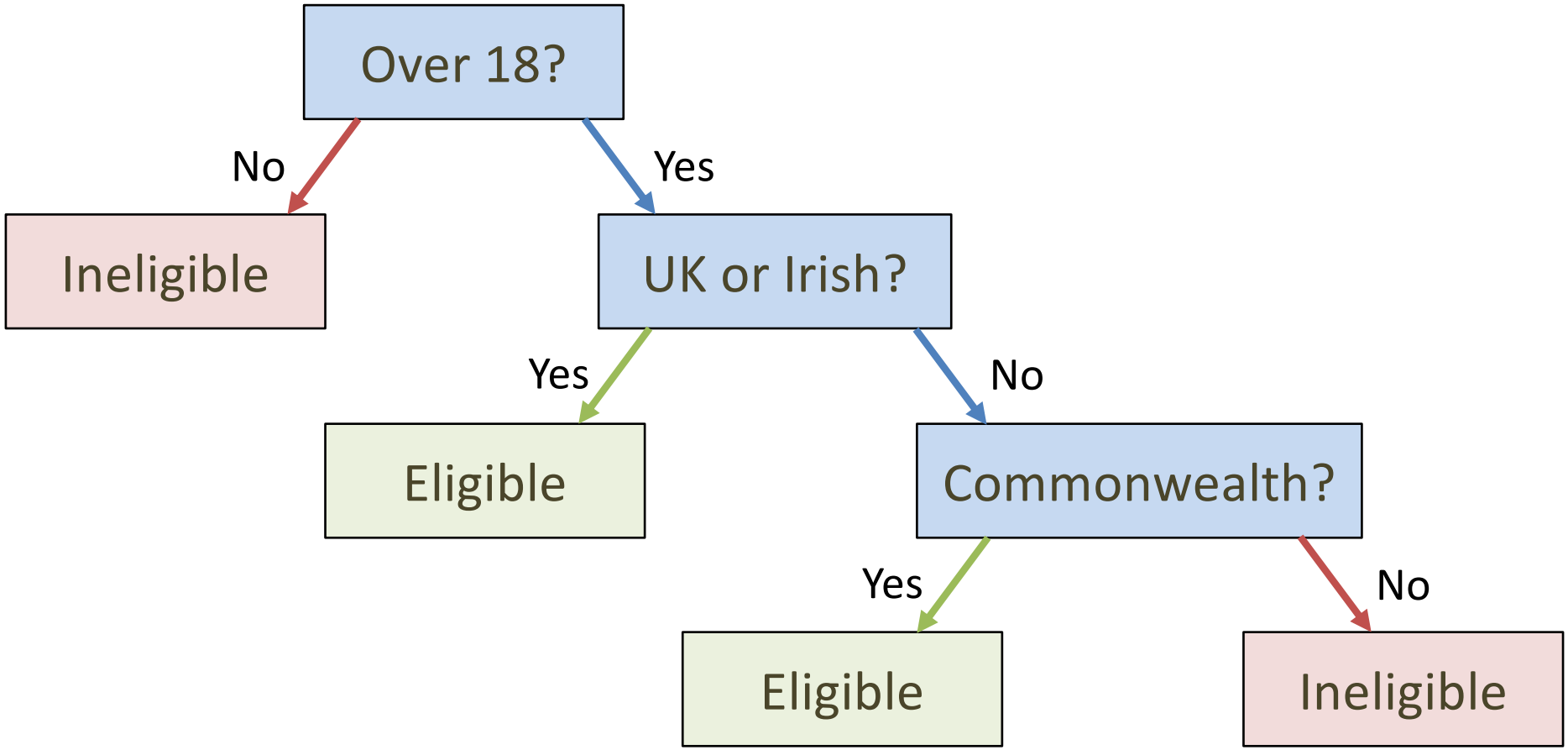
Which is equivalent to, but more readable, than

```
if(age>40 && wealth>500000)
   System.out.println("Probably tory");
else
   if(age>40 && wealth<500000)
      System.out.println("Probably not tory");
else
    System.out.println("Probably labour");
```

There are often multiple ways of coding the same behaviour. Some are better than others.

`if` Example

Are you eligible to vote in a UK general election?



`if` Example

Are you eligible to vote in a UK general election?

VoteEligibility.java

```
public class VoteEligibility {
    public static void main(String[] args) {
        // first, declare variables
        int age;
        boolean isUKorIrish;
        boolean isCommonwealth = false;
```

It is standard practice to declare all the important variables before you start writing the program's logic.

`if` Example

```
// obtain information from the user
Scanner scan = new Scanner(System.in);
System.out.println("Please enter age in years: ");
age = scan.nextInt();

System.out.println("Are you a UK or Irish citizen?
    (true/false)");
isUKorIrish = scan.nextBoolean();

if(!isUKorIrish) {
    // we only need to ask this if we know they
    // are not UK or Irish citizens
    System.out.print("Are you a Commonwealth citizen
        and resident in the UK? (true/false)");
    isCommonwealth = scan.nextBoolean();
}
```

`if` Example

```
// summarise what the user entered
System.out.println("You are "+age+" years old.");
if(isUKorIrish) {
    System.out.println("You are a UK or Irish
                        citizen.");
}
else {
    if(isCommonwealth)
        System.out.println("You are a resident
                            Commonwealth citizen.");
    else {
        System.out.println("You are not a UK,
                            Irish, or resident
                            Commonwealth
                            citizen.");
    }
}
```

`if` Example

```
        // indicate whether the user is eligible to vote
        if(age >= 18 && (isUKorIrish || isCommonwealth)) {
            System.out.println("You can vote!");
        }
        else {
            System.out.println("Sorry, you are not eligible
                                to vote.");
        }
    }
}
```

Note the use of parentheses in the `if` condition. This is required because || has the same precedence as &&

Exercise

Complete this program so that aubergines (which are 5-15cm, neither yellow nor red, and are not square) are correctly identified:

IfExercise.java

```
public class IfExercise {
   public static void main(String[] args) {
      double size = 10;
      boolean yellow = false, red = false;
      boolean square = false;
      if(<add code here>)
         System.out.println("Aubergine!");
   }
}
```

switch…case

This is another kind of conditional execution construct found in most languages

- The code executed depends on the value of a variable

```
switch(variable) {
   case value1:    // execute some code
                   break;
   case value2:    // execute some code
                   break;
   default:        // execute this code
                   // if no cases match
}
```

switch…case

This is another kind of conditional execution construct found in most languages

- The code executed depends on the value of a variable

```
switch(variable) {
   case value1:    // execute some code
                   break;
   case value2:    // execute some code
                   break;
   default:        // execute this code
                   // if no cases match
}
```

These breaks are very important. Without them, execution will run on to the code in the next case

switch…case

This is another kind of conditional execution construct found in most languages

- The code executed depends on the value of a variable

```
switch(variable) {
   case value1:    // execute some code
                   break;
   case value2:    // execute some code
                   break;
   default:        // execute this code
                   // if no cases match
}
```

"Some code" can span multiple lines, without the need for curly braces { … }

switch…case

This is another kind of conditional execution construct found in most languages

- The code executed depends on the value of a variable

```
switch(variable) {
   case value1:    // execute some code
                   break;
   case value2:    // execute some code
                   break;
   default:        // execute this code
                   // if no cases match
}
```

The default part is optional

switch…case

It is equivalent to this `if` statement:

```
if(variable==value1) {
   // execute some code
}
else if(variable==value2) {
   // execute some code
}
else {
   // execute this code
}
```

The `switch…case` version is more readable. However, it can only be used for simple conditions.

switch…case

Example: is this letter a vowel or a consonant?

VowelConsonantSwitch.java

```
public class VowelConsonantSwitch {
   public static void main(String[] args) {
      String letter = "a";
      switch(letter.toUpperCase()) {
         case "A":
         case "E":
         case "I":
         case "O":
         case "U": System.out.println("Vowel");
                   break;
         default:  System.out.println("Consonant");
      }
   }
}
```

switch…case

Example: is this letter a vowel or a consonant?

VowelConsonantSwitch.java

```
public class VowelConsonantSwitch {
   public static void main(String[] args) {
      String letter = "a";
      switch(letter.toUpperCase()) {
         case "A":
         case "E":
         case "I":
         case "O":
         case "U": System.out.println("Vowel");
                   break;
         default:  System.out.println("Consonant");
      }
   }
}
```

Note the lack of break statements, causing execution to run on to "U"

switch...case

Here's the `if` version, which is less readable:

VowelConsonantIf.java

```
public class VowelConsonantIf {
   public static void main(String[] args) {
      String letter = "a";
      letter = letter.toUpperCase();
      if(letter.equals("A") || letter.equals("E") ||
         letter.equals("I") || letter.equals("O") ||
         letter.equals("U"))
         System.out.println("Vowel");
      else
         System.out.println("Consonant");
   }
}
```

switch...case

Here's the `if` version, which is less readable:

VowelConsonantIf.java

```
public class VowelConsonantIf {
   public static void main(String[] args) {
      String letter = "a";
      letter = letter.toUpperCase();
      if(letter.equals("A") || letter.equals("E") ||
         letter.equals("I") || letter.equals("O") ||
         letter.equals("U"))
         System.out.println("Vowel");
      else
         System.out.println("Consonant");
   }
}
```

Since you can't use == with strings

Ternary Operator

Many languages, including Java and the C family, have something called the **ternary operator**

- This provides another form of conditional execution. It's not used often, but can be very useful

```
variable = condition ? value1 : value2;
```

If `condition` is true, `variable` is assigned `value1`

If `condition` is false, `variable` is assigned `value2`

Ternary Operator

Remember this example from earlier?

IfThenElseDemo.java

```
public class IfThenElseDemo {
   public static void main(String[] args) {
      int age = 18;
      if(age > 40)
         System.out.println("You're old!");
      else
         System.out.println("You're young!");
   }
}
```

Ternary Operator

We could rewrite it using the ternary operator:

TernaryDemo.java

```
public class TernaryDemo {
   public static void main(String[] args) {
      int age = 18;
      String state = age>40 ? "old" : "young";
      System.out.println("You're " + state);
   }
}
```

The ternary operator is sometimes useful for reducing the length of code, though often at the expense of clarity.

Ternary Operator

We could rewrite it using the ternary operator:

TernaryDemo.java

```
public class TernaryDemo {
   public static void main(String[] args) {
      int age = 18;
      String state = age>40 ? "old" : "young";
      System.out.println("You're " + state);
   }
}
```

?: has very low precedence (just above =), so you don't need to put parenthesis around its operands

The ternary operator is sometimes useful for reducing the length of code, though often at the expense of clarity.

Quiz

What is the output of this program fragment?

```
boolean a = true;
boolean b = false;
System.out.println(a || b ? "Yes!" : "No!");
```

A. `Yes!`

B. `No!`

C. No output is produced

D. A compiler error is generated

Quiz

What is the output of this program fragment?

```
int b = 10;
b /= 2;
if(b = 5)
   System.out.println("Yes!");
else
   System.out.println("No!");
```

A. `Yes!`

B. `No!`

C. No output is produced

D. A compiler error is generated

Quiz

What is the output of this program fragment?

```
int b = 10;
b /= 2;
if(b == 5)
   System.out.println("Yes!");
   System.out.println("Yes!");
else
   System.out.println("No!");
```

A. `Yes! Yes!`

B. `No!`

C. No output is produced

D. A compiler error is generated

Quiz

What is the output of this program fragment?

```
int b = 1;
switch(b) {
   case(1): System.out.println("Yes!");
   case(2): System.out.println("No!");
}
```

A. `Yes!`

B. `No!`

C. A compiler error is generated

D. None of the above

Summary

- Java provides three forms of conditional execution.
- `if` statements are the most flexible.
- `if` statements can be nested.
- Boolean operators have precedence rules.
- `switch...case` can be used when you only need to test the value of a variable.
- `switch...case` is often more readable than `if`
- The ternary operator can be used to reduce code length, but can make it harder to read.

Next Lab

You'll be doing **assessed** exercises involving:

- variables, types, operators, expressions
- input and output
- if…else and switch…case

So please make sure you're up to speed on these!

Tutorial 2

Contains **formative** exercises involving:

- variables, types, operators, expressions
- input and output
- if…else and switch…case

I'll put solutions to these exercises on Vision before the lab session takes place.

Next Week

- Introduction to IDEs and Eclipse
- Iteration statements (better known as loops!)