CO452
Programming Concepts

Lecture 3

Selection and Enumerate types

Selection

Selection allows us to choose between blocks of code based on whether a comparison evaluates to **true or false**

- if, else if, else blocks
- switch

if

if statement

The code within the braces of an **if** statement will execute **if** the comparison evaluates to **true**
(in this case, **if** **mark** is **less than zero**)

```
if(mark < 0)
{
   System.out.println("Mark must be greater than 0");
}
```

More comparisons

```
if(mark < 0)
{
   System.out.println("Mark must be greater than 0");
}
if(mark > 100)
{
   System.out.println("Mark must be less than 100");
}
```

Operators

Comparison Operators

==	Equality
!=	Inequality
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal

Java String Equality

Whilst the equality operator (==) can be applied to primitive data (int, char, boolean), Strings are classes, so **the equality operator would compare memory addresses of String objects** rather than the values stored in each object

Use the method **equals** to compare the values stored at String variables rather than comparing memory addresses

```
if(name.equals("Nick"))
```

Logical Operators

&&	AND operator (**ALL comparisons** must be true)
||	OR operator (**only one comparison** must be true)
!	NOT operator (reverses the evaluation of comparison)

Example of the OR operator

The OR operator **||** requires **at least one** comparison to be true

```
if(mark < 0 || mark > 100)
{
   System.out.println("Mark must between 0 and 100");
}
```

Example of the AND operator

The AND operator **&&** requires **all** comparisons to be **true**

```
if(mark >= 0 && mark <= 100)
{
   System.out.println("This is a valid mark");
}
```

else and else if

else statement

The else block executes **if** the evaluation is **false**

```
if(mark >= 0 && mark <= 100)
{
   System.out.println("This is a valid mark");
}
else
{
    System.out.println("This is an invalid mark");
}
```

else if statements

```
if(mark >= 0 && mark <= 39)
    System.out.println("This is a failed attempt");
else if (mark <= 49)
    System.out.println("This is a pass");
else if (mark <= 59)
    System.out.println("This is a 2:2");
else if (mark <= 69)
    System.out.println("This is a 2:1");
else if (mark <= 100)
    System.out.println("This is a 1st");
else
   System.out.println("This is an invalid mark");
```

Declaring constants

```
public final int GRADE_NS = 0;
public final int GRADE_F = 39;
public final int GRADE_D = 49;
public final int GRADE_C = 59;
public final int GRADE_B = 69;
public final int GRADE_A = 100;
```

Applying constants

```
if(mark >= GRADE_NS && mark <= GRADE_F)
    System.out.println("This is a failed attempt");
else if (mark <= GRADE_D)
    System.out.println("This is a pass");
else if (mark <= GRADE_C)
    System.out.println("This is a 2:2");
else if (mark <= GRADE_B)
    System.out.println("This is a 2:1");
else if (mark <= GRADE_A)
    System.out.println("This is a 1st");
else
   System.out.println("This is an invalid mark");
```

switch

How does the switch statement work?

The switch statement can compare the values of a variable against cases. Cases of the switch statement are tested individually in sequence and code executes when the case matches the value of the variable. The default is the equivalent of the else statement.

However, when a case executes, the remainder of the cases within the switch block also execute! The break statement can guard against this ‘fall-through mechanism’.

Switch example

```
char grade;

switch(grade)
{
  case 'F' :    System.out.println("This is a failed attempt"); break;
  case 'D' :   System.out.println("This is a pass"); break;
  case 'C' :   System.out.println("This is a 2:2"); break;
  case 'B' :   System.out.println("This is a 2:1"); break;
  case 'A' :   System.out.println("This is a 1st"); break;
  default  :   System.out.println("This is an invalid mark");
}
```

Enumerate type

What is an enumerate type?

Enumerate types are user defined types which have a limited number of values

This makes for easier validation, when utilising selection structures, given that the valid (acceptable) values are limited/finite

Letter grades: A, B, C, D, E, F

Directions: North, South, East, West

Music genres: Rock, Pop, Blues, Classical, R&B, Country

Basic enumerate type for Grades

```
public enum Grades
{
   NS,  F,  D,  C,  B,  A;
}
```

These would be assigned integer values by default:
NS = 0, F = 1, D = 2, C = 3, B = 4, A = 5

```
public enum Grades
{
    NS  (0), F  (39),  D  (49), C  (59), B  (69),  A  (100);

    private final int value;

    private Grades(int value)
    {
        this.value = value;
    }

    public int getValue()
    {
        return value;
    }
}
```

Applying enum for Grade

```
if(mark >= Grades.NS.getValue() && mark <= Grades.F.getValue())
    System.out.println("This is a failed attempt");
else if (mark <= Grades.D.getValue())
    System.out.println("This is a pass");
else if (mark <= Grades.C.getValue())
    System.out.println("This is a 2:2");
else if (mark <= Grades.B.getValue())
    System.out.println("This is a 2:1");
else if (mark <= Grades.A.getValue())
    System.out.println("This is a 1st");
else
    System.out.println("This is an invalid mark");
```