# CO452 Programming Concepts

Lecture 2

Selection, Iteration and Enumerate Types

Sequence, Selection and Iteration

❖ **Sequence** mandates that statements be executed in order (line by line)

❖ **Selection** (conditional) statements will execute a **block of code once** when the condition is true

❖ **Iteration** allows us to **repeat** statements within a block **whilst** the condition is **true**
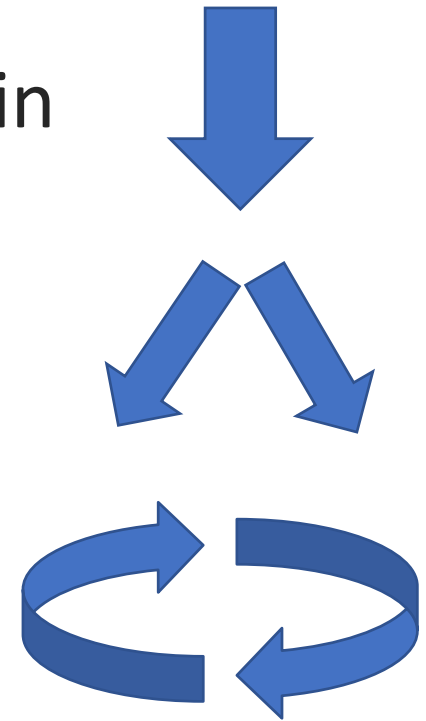
Selection

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

```java
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

```java
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**

```java
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

```java
if(mark >= 0 && mark <= 39)
    System.out.println("This is a failed attempt");
else if (mark >= 40 && mark <= 49)
    System.out.println("This is a pass");
else if (mark >= 50 && mark <= 59)
    System.out.println("This is a 2:2");
else if (mark >= 60 && mark <= 69)
    System.out.println("This is a 2:1");
else if (mark >=70 && 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_F && mark <= GRADE_D)
    System.out.println("This is a pass");
else if (mark > GRADE_D && mark <= GRADE_C)
    System.out.println("This is a 2:2");
else if (mark > GRADE_C && mark <= GRADE_B)
    System.out.println("This is a 2:1");
else if (mark > GRADE_B && mark <= GRADE_A)
    System.out.println("This is a 1st");
else
    System.out.println("This is an invalid mark");
```

Take out lower bounds

```java
if (mark < GRADE_NS || mark > GRADE_A)
    System.out.println("This is an invalid mark");
else if (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
    System.out.println("This is a 1st");
```

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

```java
char grade;

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

```java
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 Grades

```java
if (mark < Grades.NS.getValue() || mark > Grades.A.getValue())
    System.out.println("This is an invalid mark");
else if (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
    System.out.println("This is a 1st");
```

Iteration

Iteration

Iteration allows us to repeat statements within a block providing the condition is **true**

while

do while

for

for each

while

while loop

The code within the braces of an **while** statement will execute **whilst** the comparison evaluates to **true**

```java
int count = 1;

while(count <= 3)
{
  System.out.println("This loop has executed " + count + " times");
  count = count + 1;

}
```

while loop

The code within the braces of an **while** statement will execute **whilst** the comparison evaluates to **true**

```java
int count = 1;

while(count <= 3)
{
  System.out.println("This loop has executed " + count + " times");
  count = count + 1;
}
```

Variable initialisation

Condition: is the value of count < 3 ?

If yes, execute the contents of the loop

Increase the value of count by 1

do while

do while loop

Due to the order of statements, the body of the **do while** loop **will execute at least once** (even if comparison is false)

```java
int count = 0;

do
{

    System.out.println("This loop has executed " + (count+1) + " times");
    count = count + 1;

}
while(count < 3);
```

do while loop

Due to the order of statements, the body of the **do while** loop **will execute at least once** (even if comparison is false)

```
int count = 0;

do        No condition to check! So enter the braces
{
    System.out.println("This loop has executed " + (count+1) + " times");
    count = count + 1;

}
while(count < 3);        Check at the end of braces – remember ;
```

for

for loop

The for loop has three parts:

(1) variable initialisation   **(2) condition**   **(3) increment**

```java
for(int count = 0; count < 3; count++)
{
    System.out.println("This loop has executed " + (count+1) + " times");
}
```

for loop

```java
for(int count = 0; count < 3; count++)
{
  System.out.println("This loop has executed... ");
}
```

for loop

Initialise count to 0

```java
for(int count = 0; count < 3; count++)
{
  System.out.println("This loop has executed... ");
}
```

for loop

Is 0 < 3 ?

```
for(int count = 0; count < 3; count++)
{
  System.out.println("This loop has executed... ");
}
```

for loop

```
for(int count = 0; count < 3; count++)
{
  System.out.println("This loop has executed... ");
}
```

Yes! (true) Therefore execute code in braces

for loop

```java
for(int count = 0; count < 3; count++)
{
  System.out.println("This loop has executed... ");
}
```

for loop

count now has value of 1

```
for(int count = 0; count < 3; count++)
{
  System.out.println("This loop has executed... ");
}
```

for loop

We don't re-initialise count back to 0!

```java
for(int count = 0; count < 3; count++)
{
  System.out.println("This loop has executed... ");
}
```

for loop

Is 1 < 3 ?

```
for(int count = 0; count < 3; count++)
{
  System.out.println("This loop has executed... ");
}
```

for loop

```
for(int count = 0; count < 3; count++)
{
  System.out.println("This loop has executed... ");
}
```

Yes! (true) Therefore execute code in braces

for loop

After executing code in braces, increment count

```
for(int count = 0; count < 3; count++)
{
  System.out.println("This loop has executed... ");
}
```

for loop

Count now has the value of 2

```
for(int count = 0; count < 3; count++)
{
  System.out.println("This loop has executed... ");
}
```

for loop

Is 2 < 3 ?

```java
for(int count = 0; count < 3; count++)
{
  System.out.println("This loop has executed... ");
}
```

for loop

```
for(int count = 0; count < 3; count++)
{
  System.out.println("This loop has executed... ");
}
```

Yes! (true) Therefore execute code in braces

for loop

After executing code in braces, increment count

```java
for(int count = 0; count < 3; count++)
{
  System.out.println("This loop has executed... ");
}
```

for loop

Count now has
the value of 3

```
for(int count = 0; count < 3; count++)
{
  System.out.println("This loop has executed... ");
}
```

for loop

Is 3 < 3 ?

```java
for(int count = 0; count < 3; count++)
{
  System.out.println("This loop has executed... ");
}
```

for loop

```
for(int count = 0; count < 3; count++)
{
  System.out.println("This loop has executed... ");
}
```

No! (false) Therefore end loop and continue program

```
//continue with program...
```

for each

for each item in the collection

The for each loop can be used to iterate through collections of objects.

Requires an object to be declared of the type of item that is in the collection:

```
for(Student student : students)
{
    student.print();
}
```

for each item in the collection

The for each loop can be used to iterate through collections of objects.

Requires an object to be declared of the type of item that is in the collection:

Type	variable	ArrayList

```
for(Student student : students)
{
    student.print();    call print on each item in the ArrayList
}
```