Introduction to Mathematical Operators

- `*  /  %  +  -` are the mathematical operators
- `*  /  %` have a higher precedence than `+` or `-`

```
double myVal = a + b % d – c * d / b;
```

- Is the same as:

```
double myVal = (a + (b % d)) –
                    ((c * d) / b);
```

Statements & Blocks

- A simple statement is a command terminated by a semi-colon:

  ```
  name = “Fred”;
  ```

- A block is a compound statement enclosed in curly brackets:

  ```
  {
      name1 = “Fred”; name2 = “Bill”;
  }
  ```

- Blocks may contain other blocks

Flow of Control

- Java executes one statement after the other in the order they are written
- Many Java statements are flow control statements:

Alternation:	if, if else, switch
Looping:	for, while, do while
Escapes:	break, continue, return

If – The Conditional Statement

- The if statement evaluates an expression and if that evaluation is true then the specified action is taken

```
if ( x < 10 ) x = 10;
```

- If the value of x is less than 10, make x equal to 10
- It could have been written:

```
if ( x < 10 )
x = 10;
```

- Or, alternatively:

```
if ( x < 10 ) { x = 10; }
```

Relational Operators

==	Equal (careful)
!=	Not equal
>=	Greater than or equal
<=	Less than or equal
>	Greater than
<	Less than

If… else

- The if … else statement evaluates an expression and performs one action if that evaluation is true or a different action if it is false.

```
if (x != oldx) {
  System.out.print(“x was changed”);
}
else {
  System.out.print(“x is unchanged”);
}
```

Nested if ... else

```
if ( myVal > 100 ) {
  if ( remainderOn == true) {
     myVal = mVal % 100;
  }
  else {
   myVal = myVal / 100.0;
  }
}
else
{
  System.out.print(“myVal is in range”);
}
```

else if

- Useful for choosing between alternatives:

```
if ( n == 1 ) {
  // execute code block #1
}
else if ( j == 2 ) {
  // execute code block #2
}
else {
  // if all previous tests have failed,
  execute code block #3
}
```

A Warning...

WRONG!

```
if( i == j )
     if ( j == k )
      System.out.print(
         “i equals k”);
     else
      System.out.print(
      “i is not equal
      to j”);
```

CORRECT!

```
if( i == j ) {
  if ( j == k )
  System.out.print(
      “i equals k”);
}
else
  System.out.print(“i
  is not equal to j”);
   // Correct!
```

The switch Statement

```
switch ( n ) {
  case 1:
   // execute code block #1
   break;
  case 2:
   // execute code block #2
   break;
   default:
   // if all previous tests fail then
   //execute code block #4
   break;
}
```

The for loop

- Loop n times

```
for ( i = 0; i < n; n++ ) {
  // this code body will execute n times
  // ifrom  0 to n-1
}
```

- Nested for:

```
for ( j = 0; j < 10; j++ ) {
  for ( i = 0; i < 20; i++ ){
   // this code body will execute 200 times
  }
}
```

while loops

```
while(response == 1) {
  System.out.print( “ID =” + userID[n]);
  n++;
  response = readInt( “Enter “);
}
```

What is the minimum number of times the loop is executed?

What is the maximum number of times?

do {... } while loops

```
do {
  System.out.print( “ID =” + userID[n] );
  n++;
  response = readInt( “Enter ” );
}while (response == 1);
```

What is the minimum number of times the loop is executed?

What is the maximum number of times?

Break

- A break statement causes an exit from the innermost containing while, do, for or switch statement.

```
for ( int i = 0; i < maxID, i++ ) {
  if ( userID[i] == targetID ) {
   index = i;
   break;
  }
} // program jumps here after break
```

Continue

- Can only be used with while, do or for.
- The continue statement causes the innermost loop to start the next iteration immediately

```
for ( int i = 0; i < maxID; i++ ) {
  if ( userID[i] != -1 ) continue;
  System.out.print( “UserID ” + i + “ :” +
   userID);
}
```

Application & Scope

- These if and else statements are very easy to use and user friendly.
- It help in complex statements.
- It can created nested of statements.
- These conditions use in C, Java, C++ or many oops oriented application.