If statements : the basic usage.

If(statement1){

}

If(statement2){

}

A better way to do it.

If(input > 10){

}

Else{

}

The above still does the same thing.

> , < , == , != are some of the operators you can use with inside conditional statements.

Custom objects or classes cannot be compared using these operators but they have class operators like input.equals.

**Complex conditions**

If (input.contains(sOne) || input.contains(sTwo) && false){

System.out.println(“TRUE”);

}else{

System.out.println(“FALSE”);

}

In this situation thr && executes first and then the || and this can make things a bit confusing.

But you can use “()” to tell java which statement to execute first in a if statement.

1. g if((input.contains(sOne) || input.contains(sTwo) && false) willl compute the statments in the brakets first before going to the &&.

To simplify things even further we can store the boolean expressions inside boolean values before we evaluate the if statement for more readability , e.g :

Boolean bool1 = input.contains(sOne);

Boolean bool2 = input.contains(sTwo);

If(bool1 || bool2) && false){

System.out.print(“TRUE”);

}.

Avoid complex conditionals because often times there is a more elegant solution.

**The switch statement**

Another way to control the flow of a program.

If vs Switch statements

Int x = 1;

If(x == 1){

System.out.println(“RED”)

}

If(x == 2){

System.out.println(“BLUE”)

}

If(x == 3){

System.out.println(“GREEN”)

}

If((x != 1) && (x != 2) && (x != 3)){

System.out.println(“RED”)

}

This strategy works but it will get very tedious once we deal with many cases of x and also it can be very difficult to read/follow. The solution to this is the switch statements.

Int x = 1;

Switch(x){  
case 1:

System.out.println(“RED”)

case 2:

System.out.println(“BLUE”)

case 3:

System.out.println(“GREEN”)

default:

System.out.println(“NONE”)

}

As you can see the switch statment code does the exact same task as the if statements but its a more elegant and easy to code/read solution.

But there is one problem with the above switch statment. Without breaks the statements can actually run multiple cases. The correct of doing switch statements is doing breaks.

Switch(x){  
case 1:

System.out.println(“RED”);

break;

case 2:

System.out.println(“BLUE”);

break;

case 3:

System.out.println(“GREEN”);

break;

default:

System.out.println(“NONE”)

break;

}

**While and Do while loops**

Loops allow us to execute a block of java code over and over again.

Example 1

Int I = 5;

while(i > 5){

System.out.println(“Hello world”);

i--;

}

Example 2

Import java.util.\*;

Scanner reader = new Scanner(System.in);

String input;

String all = “”;

Int i = 5;

While(i > 0){

Input = reader.nextLine();

all += input;

I--;

}

System.out.println(all);

Example 3

String input;

String all = “”;

While(!input.equals(STOP)){

Input = reader.nextLine();

}

This above code will gives us an error on the line of the while loop because input is not yet set and is not a primitive , so the computer cannot possibly call the input.equals() method. This is where the do while comes in :

String input;

String all = “”;

do{

Input = reader.nextLine();

all += input;

}

While(!input.equals(STOP)){

The above will suprisingly not give us an error because of what is happening with the do while executes the code block first before doing the conditional checks so the value of input will be set by then.

**For loops**

/\*int I = 1;

While(i <= 100 ){

System.out.println(i);

i++;

}\*/

for(int i = 1; i <= 100; i++){

System.out.println()

}