REPETITION

Control Flow and Control Structure

- The order in which a program's statements execute is called its control flow.
- A programmer specifies a program's control flow.
- Control Structure
  - Sequence logic structure
  - Selection (Branch) logic structure
  - **Repetition (Loop) logic structure**

Repetition Statements

- Repetition statements allow us to execute a statement multiple times
- Often they are referred to as loops
- Like conditional statements, they are controlled by boolean expressions
- Java has three kinds of repetition statements:
    - `while loop`
    - `do loop`
    - `for loop`
- The programmer should choose the right kind of loop for the situation

The while Statement

- A *while statement* has the following syntax:

```
        while (boolean expression)

            statement;
```

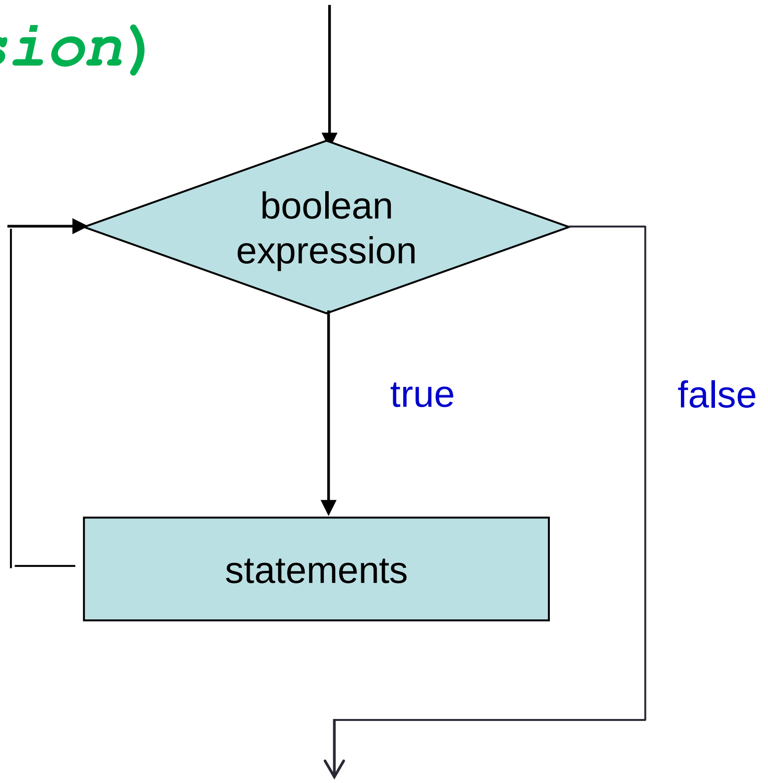
If the `boolean expression` **is true, the** `statement` **is executed**

Then the condition is evaluated again, and if it is still true, the statement is executed again

The statement is executed repeatedly until the condition becomes false

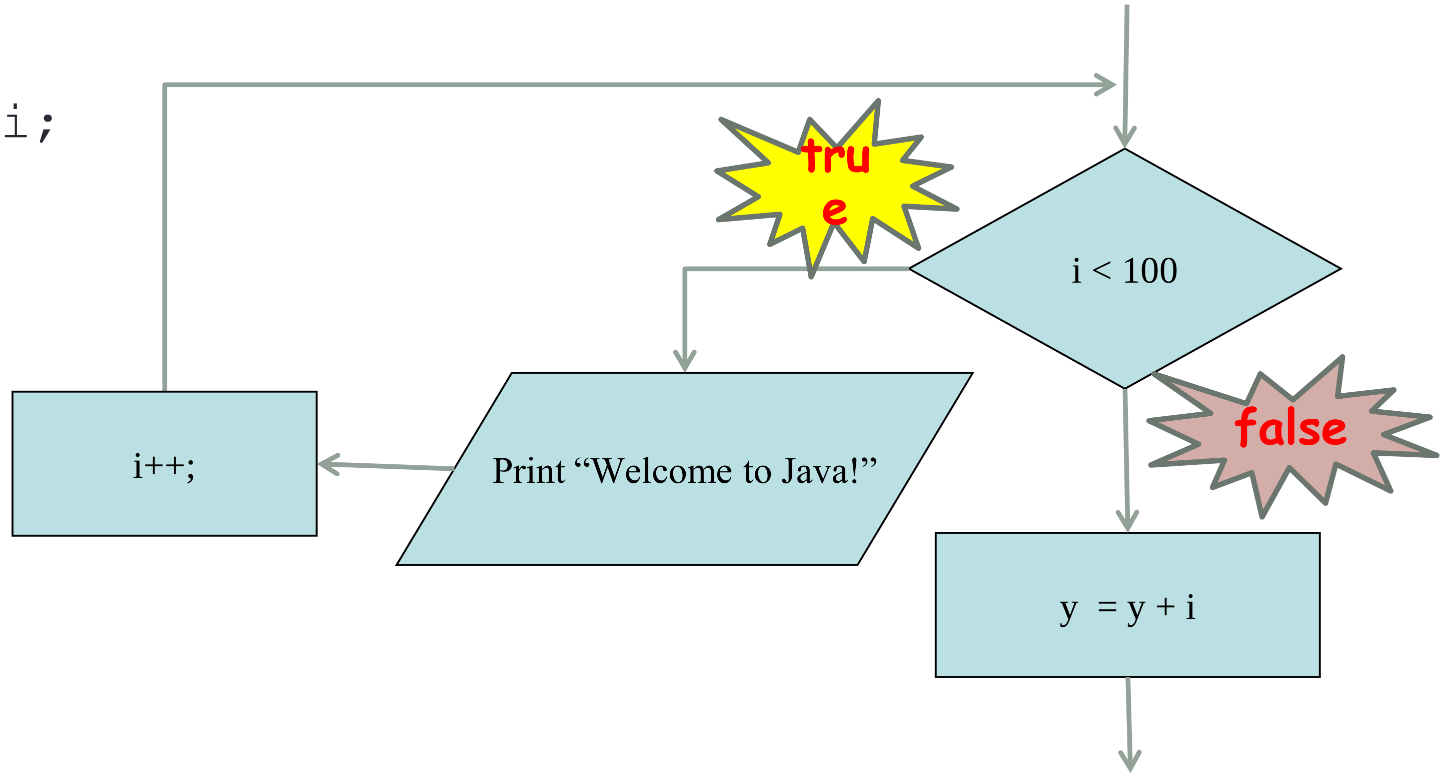
Logic of a while Loop

```
while (boolean expression)
    statement;
```



while Loop Flow Chart, cont.

```
int i = 0;
while (i < 100) {
   System.out.println("Welcome to Java!");
   i++;
}
y = y + i;
```



The while Statement

- Let's look at some examples of loop processing
- A loop can be used to maintain a running sum
- A sentinel value (flag value or signal value) is a special input value that represents the end of input
- A loop can also be used for input validation, making a program more robust

Example: Average.java

```
int sum = 0, value, count = 0;
double average;
Scanner scan = new Scanner (System.in);
System.out.print("Enter number(0 to quit): ");
value = scan.nextInt();

while(value != 0) {
   count++;
   sum += value;
   System.out.print("Enter number(0 to quit): ");
   value = scan.nextInt();
}
System.out.println ();
if (count == 0)
   System.out.println ("No values were entered.");
else {
   average = (double)sum / count;
   System.out.println ("The average is " + average);
}
```

sentinel value

maintain running sum

Example: WinPercentage.java

```
final int NUM_GAMES = 12;
int won;
double ratio;
Scanner scan = new Scanner (System.in);
System.out.print("Enter the number of games won (0 to "
   + NUM_GAMES + "): ");
won = scan.nextInt();
while (won < 0 || won > NUM_GAMES){
  System.out.print ("Invalid input. Please reenter: ");
  won = scan.nextInt();
}
ratio = (double)won / NUM_GAMES;
System.out.println ("\nWinning percentage: " + ratio*100 + "%"));
```

input validation

Infinite Loops

- The body of a `while` loop eventually must make the condition false
- If not, it is called an *infinite loop*, which will execute until the user interrupts the program
- This is a common logical error
- You should always double check the logic of a program to ensure that your loops will terminate normally

Infinite Loops

- An example of an infinite loop:

```
int count = 1;
while (count <= 25){
   System.out.println(count);
   count = count - 1;
}
```

This loop will continue executing until interrupted (Control-C) or until an underflow error occurs

Nested Loops

- Similar to nested `if` statements, loops can be nested as well
- That is, the body of a loop can contain another loop
- For each iteration of the outer loop, the inner loop iterates completely

Example: PalindromeTester.java

```
String str, another = "y";
int left, right;
Scanner scan = new Scanner (System.in);
while(another.equalsIgnoreCase("y")) {
   System.out.println ("Enter a potential palindrome:");
   str = scan.nextLine();
   left = 0;
   right = str.length() - 1;
   while (str.charAt(left)==str.charAt(right)&& left < right){
        left++;
        right--;
   }
   String prefix = "That string IS ";
   if (left < right)
     System.out.println(prefix + "NOT a palindrome.\n");
   else
     System.out.println(prefix +"a palindrome.\n");
   System.out.print("Test another palindrome (y/n)? ");
   another = scan.nextLine();
}
```

Nested Loops

- How many times will the string `"Here"` be printed?

```
count1 = 1;
while (count1 <= 10){
    count2 = 1;
    while (count2 <= 20){
        System.out.println ("Here");
        count2++;
    }
    count1++;
}
```

The do Statement

- A *do statement* has the following syntax:

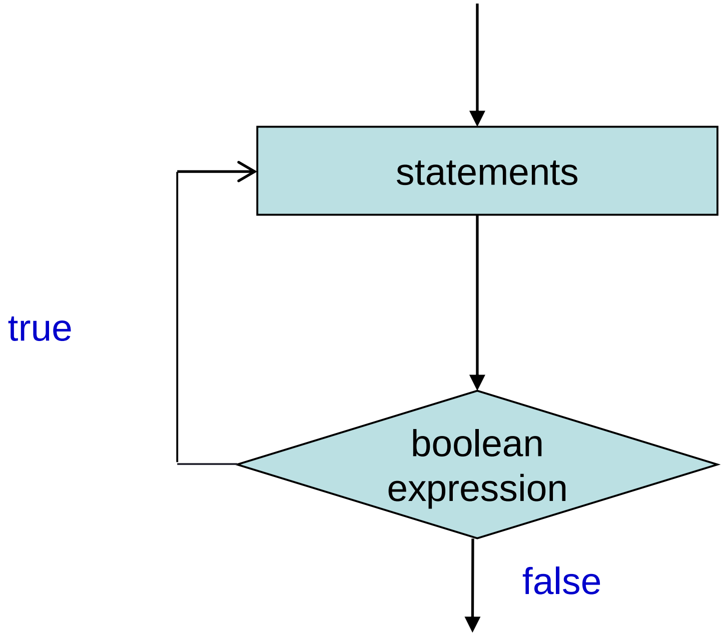
```
do {
    statement;
}
while (boolean expression);
```

The `statement` **is executed once initially, and then the** `boolean expression` **is evaluated**

The statement is executed repeatedly until the condition becomes false

Logic of a do Loop

```
do {
    statement;
}
while (boolean expression);
```



Example: ReverseNumber.java

```
int number, lastDigit, reverse = 0;
Scanner scan = new Scanner (System.in);
System.out.print ("Enter a positive integer: ");
number = scan.nextInt();
do {
    lastDigit = number % 10;
    reverse = (reverse * 10) + lastDigit;
    number = number / 10;
} while (number > 0);

System.out.println ("That number reversed is " +reverse);
```

Caution

- Similarly, the following loop is also wrong:

```
int i=0;
while (i<10);{
   System.out.println("i is " + i);
   i++;
}
```

Wrong

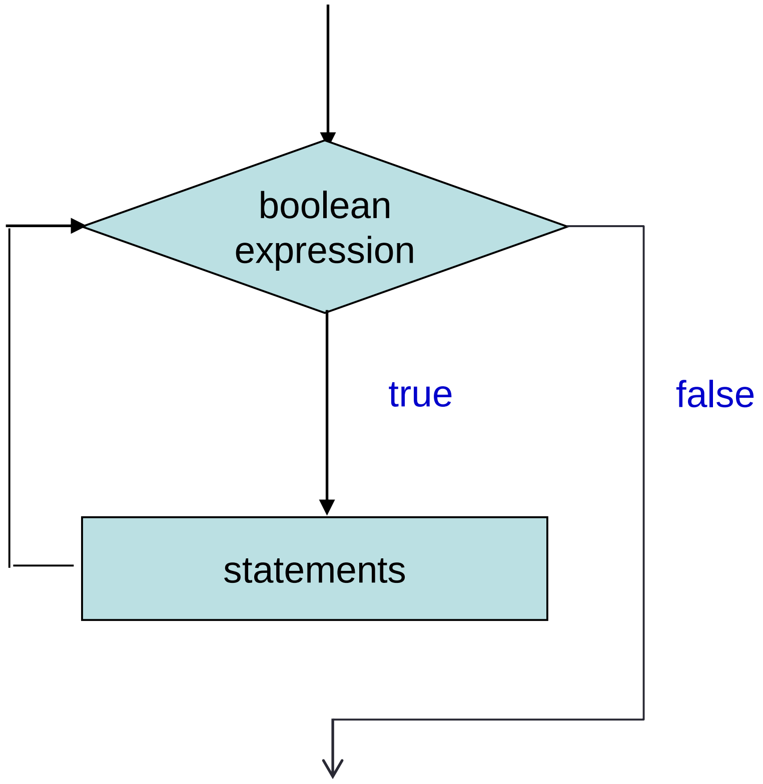
- In the case of the do loop, the following semicolon is needed to end the loop.

```
int i=0;
do {
   System.out.println("i is " + i);
   i++;
} while (i<10);
```

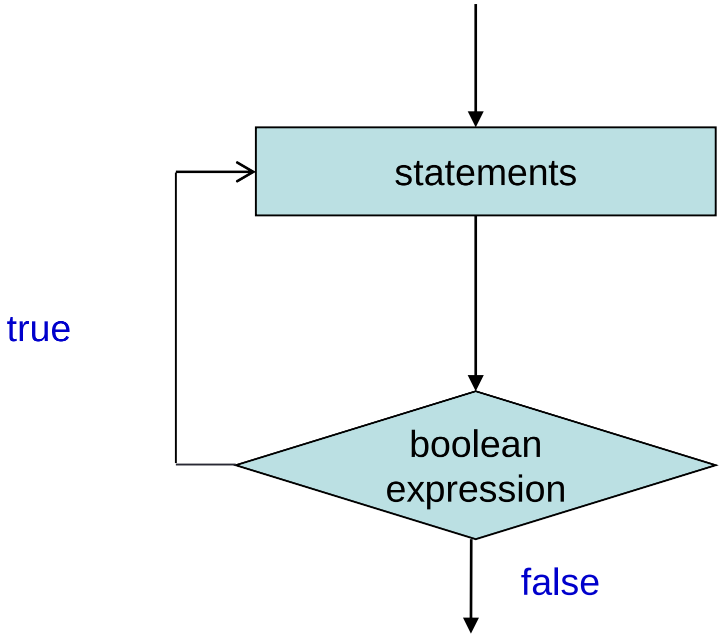
Correct

Comparing while and do..while

while



do-while



The for Statement

- A *for statement* has the following syntax:

The ***initialization*** is executed once before the loop begins

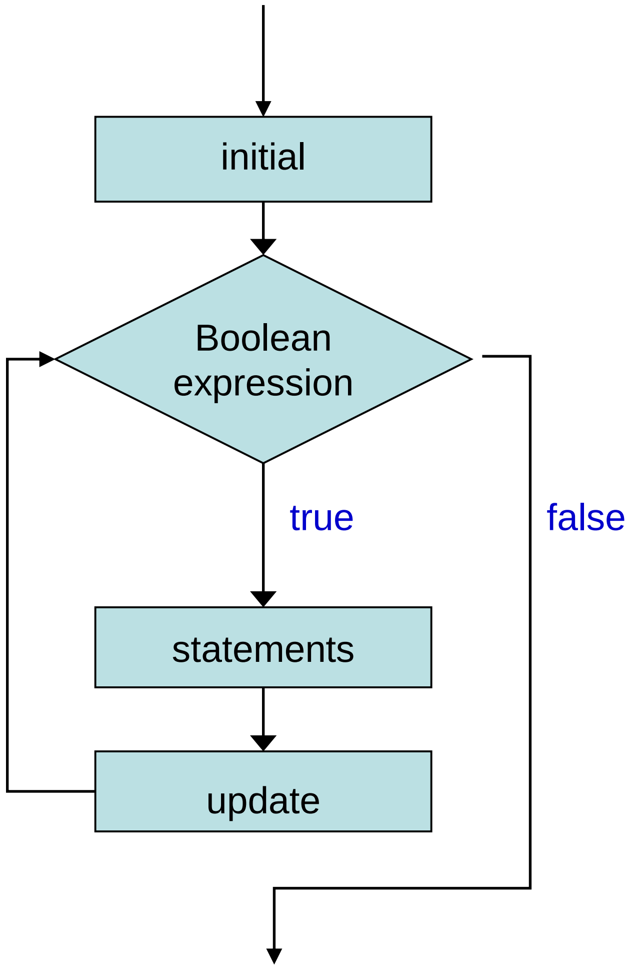
The ***statement*** is executed until the ***condition*** becomes false

```
for ( initialization ; boolean expression ; increment )
    statement;
```

The ***increment*** portion is executed at the end of each iteration

The for Loop Flow Chart

```
for ( initialization ; boolean expression ; increment )
    statement;
```



For Loop vs. while Loop

```
int i = 0;
while (i < 100) {
     System.out.println("Welcome to Java! ” + i);
     i++;
}

int  i;
for (i = 0; i < 100; i++) {
     System.out.println("Welcome to Java! ” + i);
}
```

Caution

- while, do..while and for loops control only one statements
- If you have more statements, you need to group together into a *block statement* delimited by braces

```
while(){
   statements
}

 do{
    statements
 }while();

 for(){
     statements
 }
```

Caution

- Adding a semicolon at the end of the for clause before the loop body is a common mistake, as shown below:

```
for (int i=0; i<10; i++);
{
  System.out.println("i is " + i);
}
```

Wrong

//cannot find variable i

Factorial

- *N* Factorial == *N!*
- *N!* == N * (N-1) * (N-2) * (N-3) * . . . 4 * 3 * 2 * 1
- *N* must be a positive integer or zero, and *0!* is defined to be 1.
- For example,

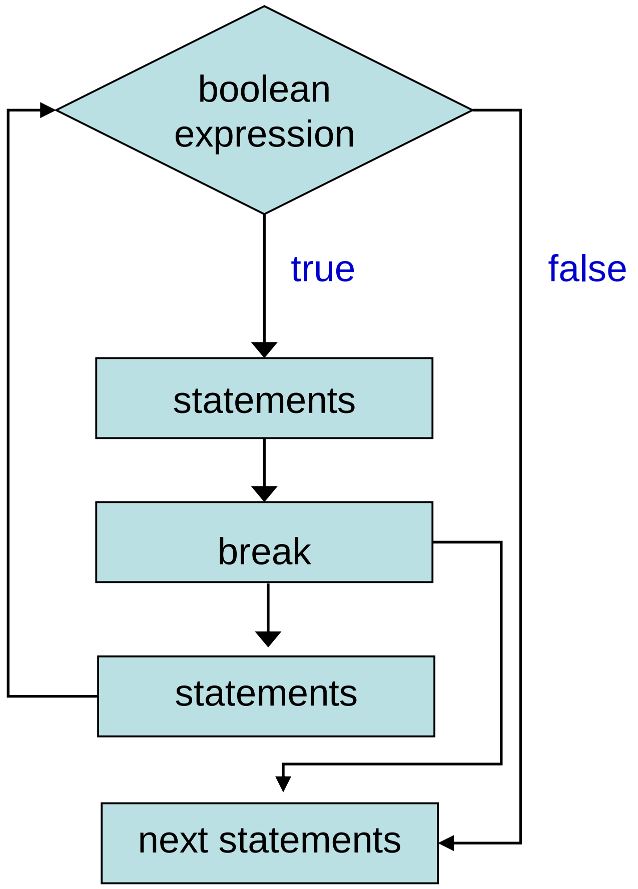
  6! == 6 * 5 * 4 * 3 * 2 * 1 == 720

Example: Factorial

```
int n = sc.nextInt();
if (n > 0) {
    int fact=1;
    for (int i = n; i>1; i--)
        fact*=i;    // fact = fact * i;
    System.out.println(n + "! = "+ fact);
} else {
    System.out.println("Invalid number !!! ");
}
```

The `break` Keyword

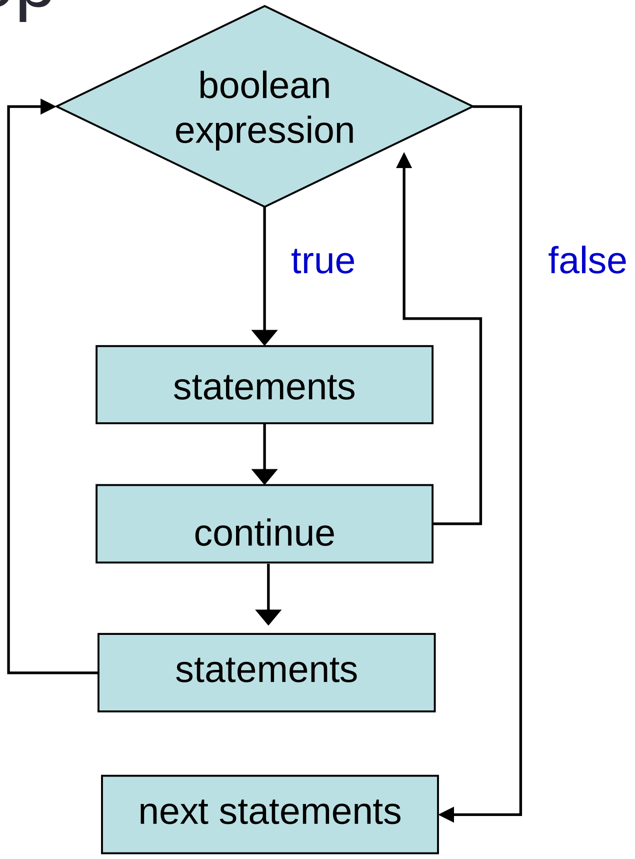
- The break statement immediately jumps to the end of loop



break uses for
- **switch**
- **while**
- **do..while**
- **for**

The `continue` Keyword

- The continue statement immediately jumps to the next iteration of loop



continue uses for
- **while**
- **do..while**
- **for**

break

```
int i,j;
System.out.println("Prime numbers between 1 to 50 : ");
for (i = 1;i <= 50;i++ ){
   for (j = 2;j < i;j++ ){
      if(i % j == 0)
         break;
   }
   if(i == j)
      System.out.print("  " + i);
}
```

continue

```
String searchMe = "peter piper picked a peck of pickled peppers";
int max = searchMe.length();
int numPs = 0;
for (int i = 0; i < max; i++) {
   //interested only in p’s
   if (searchMe.charAt(i) != 'p')
      continue;
  //process p’s
   numPs++;
}
System.out.println("Found " + numPs + " p's in the string.");
```