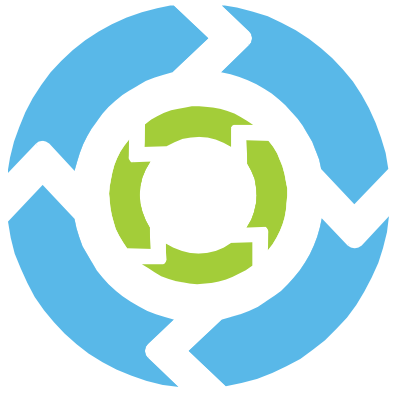
Iteration

while Loop, new operators



Learning Objectives

- Understand the principle of iteration in a program
- Write *while* loops
- Write variations of while loop: counter, sentinel and state controlled loops
- Use additional operators to write concise expressions
- Revisit order of precedence

Iteration (loop) basics

- Many activities are done repeatedly
- For example, consider calculating your GPA
- You have 6 grades from which a GPA has to be calculated
- How about 50 students, each with 6 grades
- The task becomes tedious and time consuming
- We can use loops to make the calculation process much more efficient and the code more compact

Iteration (loop) basics

- We can solve the problem the following way

```
so long there is another student in a list
    read the grades
    add grades to an accumulator
    calculate the grade point average
```

- Three tasks are repeated for each student in the list
- This logic is easily modeled using a while loop

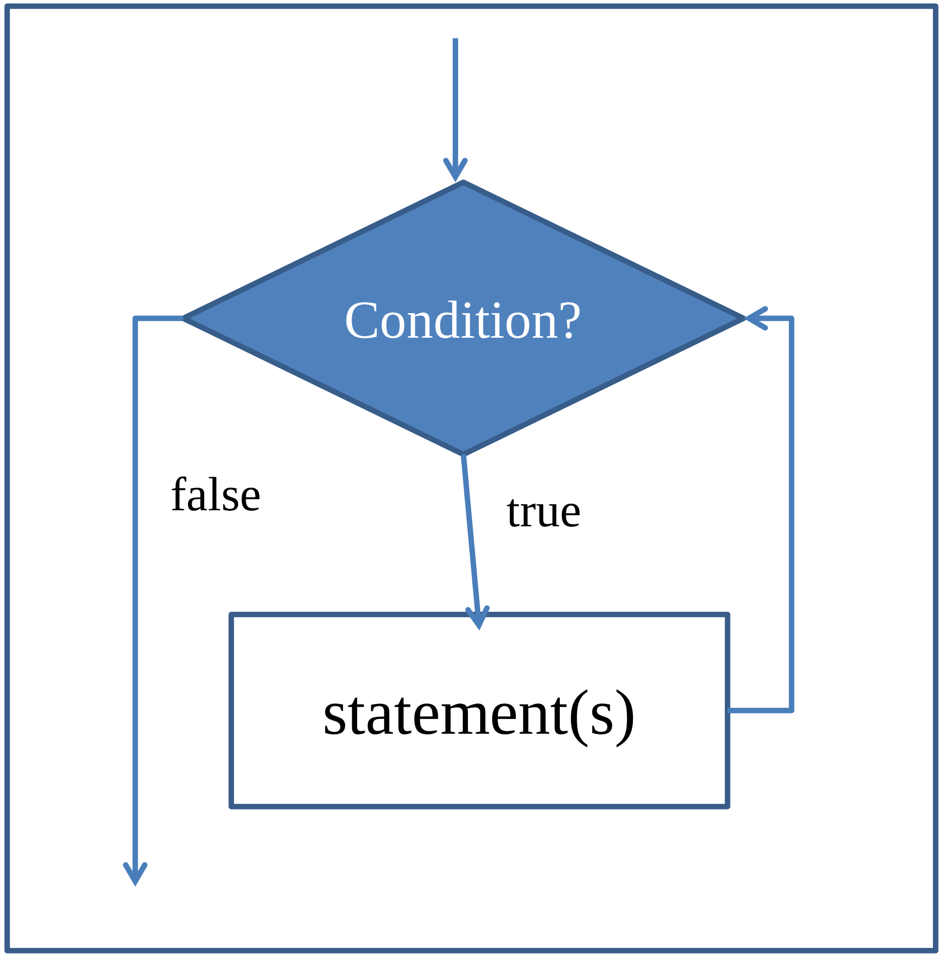
while loop

```
while (loop-continuation-condition) {
        statement(s);  // Loop body
}
```

- Loop-continuation-condition is tested for every iteration of the loop
  - Condition is same as discussed in the decision making section (test condition or conditional expression)
  - Condition is evaluated to true or false
  - Iteration is repeated if the condition is true
  - The while loop can be stated as ‘while condition is true, perform statement(s)’

while loop

```
while (conditional expression) {
        statement(s);
}
```



while loop

- For the loop to execute at least once, loop-continuation-condition must be true to start with
- The condition must become false for the loop to stop
  - **Otherwise it is an Infinite Loop**
- Condition is tested once for each iteration of the loop

while loop – counting example

```
int sum = 0;
int number = 5;
while (number > 3) {
        sum = sum + number;
        number = number - 1;
}
System.out.println(“ Sum = ” + sum);
```

- Java provides a number of other operators
- The following slides discuss these operators before revisiting while loop.

A FEW OPERATORS

++ operator

- ++ is an increment operator
- -- is a decrement operator
- Both ++ and – are Unary operators – one operand only
- Most commonly used with integers
- 1 is added or subtracted to the variable
- Java allows use with floating-point type – adds or subtracts 1
- Example:

```
    int num = 100;
    num++;          // num = 101
    num--;          // num = 100
```

++ operator

- Both ++ and -- can be placed before a variable
  - Postfix when used after the variable
  - Prefix when used before the variable
- Called preincrement or predecrement
- Example:

```
int num = 100;
++num;                  // num = 101
++num;                  // num = 102
--num;          // num = 101
--num;          // num = 100
--num;          // num = 99
```

++ operator

- Both increments and decrement (++ and --) can be used in an expression
- Example: increment

```
int num = 10;
int newNum = 10 * num++;
// ++ is postfix increment
// newNum = 10 * 10;  num = 11
```

- When used as increment or decrement (postfix), the value of variable is used first before increment or decrement

++ operator

- Both preincrements and predecrement (prefix) can also be used in an expression
- Example: preincrement

```
int num = 10;
int newNum = 10 * ++num;
// ++ is prefix increment
// newNum = 10 * 11;  num = 11
```

- When used as preincrement or predecrement (prefix), the value of variable is increased or decreased first before being used in the expression

++ operator

Another Example:

```
int count = 0, result = 0, firstNum = 10;
count++;                // count is now 1
result = count++ * --firstNum + 10;
```

⇒result = 1 * 9 + 10

⇒result = 9 + 10

result = 19   count = 2   firstNum = 9

- Order of operations: ++ -- (Unary operators are Right to left)

++ operator

Yet another example:

```
double x = 1.0;
double y = 5.0;
double z = x-- + (++y)
```

Result:

```
z = 1 + 6 => 7
x = 0
y = 6
```

Compound Arithmetic Operators

Operator	Operation	Example
+=	Addition	answer += 2;
- =	Subtraction	answer - = 2;
* =	Multiplication	answer *= 2;
/=	Division	answer /= 2;
%=	Modulus	answer %= 2;

Order of Operation

Category	Operators	Associativity
Unary	+ - ++ --	Right
Multiplicative	* / %	Left
Additive	+ -	Left
Relational	< > <= >=	Left
Equality	= = !=	Left
Assignment	= *= /= %= += - =	Right

BACK TO WHILE LOOP

Sentinel-Controlled while loop

- Used when a loop reads data from the console or a file
  - The number of data items to be read is unknown
- The program can rely on a sentinel – a last known value to stop the loop
- Sentinel value is an extreme value, or a dummy value
- Sentinel value should not be a legitimate or expected data
- Sentinel value should not be processed

Sentinel-Controlled while loop

```
import java.util.Scanner;
public class LoopTester {
  public static void main(String[] args) {
     Scanner in = new Scanner(System.in);
     int myData = 0;
     int count = 0;
     while (myData != -99) {
        System.out.println("Enter a number, -99 to stop");
        count++;
        myData = in.nextInt();
     }
     System.out.println("Iteration count = "+ count);
  }
}
```

Sentinel value

This code has one problem, it iterates at least once
The reported value of count is off by one

Sentinel-Controlled (improved)

```
import java.util.Scanner;
public class LoopTester {
   public static void main(String[] args) {
      Scanner in = new Scanner(System.in);
      int myData = 0;
      int count = 0;
      System.out.println("Enter a number, -99 to stop");
      myData = in.nextInt();
      while (myData != -99) {
          System.out.println("Enter a number, -99 to stop");
          count++;
          myData = in.nextInt();
      }
      System.out.println("Iteration count = "+ count);
   }
}
```

Priming Read

In sync with iterations

State-Controlled while Loop

- State controlled loops are a variation of Sentinel-controlled loop
- The sentinel is replaced by a boolean flag variable

```
boolean moreData = true;
while (moreData) {
        statement;
        statement;
        if (someCondition) {
                moreData = false;
        }
}
```

Key requirement to stop the loop