

# Decisions, Iterations and File Processing

---

## Background

---

- For general problem solving we need more capabilities
  - The ability to control which statements are executed
  - The ability to control how often a statement is executed
  - The ability to have iterations, or loops
- We will concentrate first on controlling which statements are executed
  
- Java provides the **if** and **switch** conditional constructs to control whether a statement list is executed
  - The **if** constructs use logical expressions to determine their course of action
  - **while** and **for** are the controls for iterations
  
- Examination begins with logical expressions

## Logical expressions

---

- The branch of mathematics dealing with logical expressions is Boolean algebra
  - Developed by the British mathematician George Boole

## Logical expressions

---

- logical values are true or false
- There are three primary logical operators for manipulating logical values
  - Logical and
  - Logical or
  - Logical not
  
- The operators work as most of us would expect

## Boolean algebra

- We use truth tables to give formal specifications of the operators
- Can create complex logical expressions by combining simple logical expressions

p	q	p and q	p or q	not (p)
False	False	False	False	True
False	True	False	True	True
True	False	False	True	False
True	True	True	True	False

## A boolean type

- Java has the logical type **boolean**
- Type **boolean** has two literal constants
  - **true**
  - **false**
- Operators
  - The and operator is **&&**
  - The or operator is **||**
  - The not operator is **!**

## Defining boolean variables

- Local boolean variables are uninitialized by default

```
boolean isWhitespace;  
boolean receivedAcknowledgement;  
boolean haveFoundMissingLink;
```

isWhitespace	-
receivedAcknowledgement	-
haveFoundMissingLink	-

## Defining boolean variables

- Local boolean variables with initialization

```
boolean canProceed = true;  
boolean preferCyan = false;  
boolean completedSecretMission = true;
```

canProceed	true
preferCyan	false
completedSecretMission	true

## Other operators

---

### □ Equality operators == and !=

- Operator ==
  - Evaluates to true if the operands have the same value; otherwise, evaluates to false
- Operator !=
  - Evaluates to true if the operands have different values; otherwise, evaluates to false
- The operators work with all types of values

## Take care with floating-point values

---

- Consider

```
double a = 1;
double b = 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1
          + 0.1 + 0.1 + 0.1 + 0.1;
double c = .9999999999999999;
```
- Two false Java expressions!

```
a == b      b != c
```
- Two true Java expressions!

```
a != b      b == c
```
- Problem lies with the finite precision of the floating-point types
  - Instead test for closeness with floating-point values

```
Math.abs(a - b) < Small number
```

## Ordering operators

---

- Java provides ordering operators for the primitive types
  - Four ordering operators, <, >, <=, and >=
  - They correspond to mathematical operators of <, >, ≥, and ≤
- Together the equality and ordering operators are known as the relational operators
- False is less than true

## Unicode values

---

- Character comparisons are based on their Unicode values
- Characters '0', '1', ... '9' have expected order
  - Character '0' has the encoding 48
  - Character '1' has the encoding 49, and so on.
- Upper case Latin letters 'A', 'B', ... 'Z' have expected order
  - Character 'A' has the encoding 65, character 'B' has the encoding 66, and so on.
- Lower case Latin letters 'a', 'b', ... 'z' have expected order
  - Character 'a' has the encoding 97
  - Character 'b' has the encoding 98, and so on.

## Evaluation boolean expressions

### □ Suppose

```
char c = '2';  
char d = '3';  
char e = '2';
```

### □ What is the value of

```
c < d  
c < e  
c <= e  
d >= e  
c >= e
```

## Operator precedence revisited

### □ Highest to lowest

- Parentheses
- Unary operators
- Multiplicative operators
- Additive operators
- Relational ordering
- Relational equality
- Logical and
- Logical or
- Assignment

## Evaluating boolean expressions

### □ Suppose

```
boolean p = true;  
boolean q = false;  
boolean r = true;  
boolean s = false;
```

### □ What is the value of

```
p  
!s  
q  
p && r  
q || s  
p && s  
p == q  
q != r  
r == s  
q != s
```

## Evaluating boolean expressions

### □ Suppose

```
int i = 1;  
int j = 2;  
int k = 2;  
char c = '#';  
char d = '%';  
char e = '#';
```

### □ What is the value of

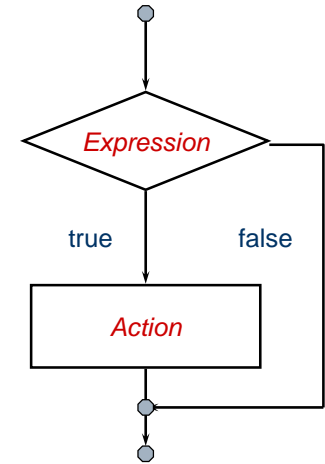
```
j == k  
i == j  
c == e  
c == d  
i != k  
j != k  
d != e  
c != e
```

## Conditional constructs

- Provide
  - Ability to control whether a statement list is executed
- Two constructs
  - If statement
    - if
    - if-else
    - if-else-if
  - Switch statement

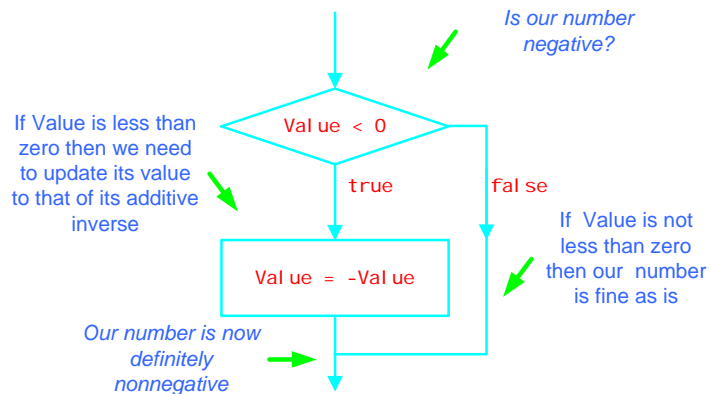
## Basic if statement

- Syntax
  - if (Expression)*  
*Action*
- If the *Expression* is true then execute *Action*
- *Action* is either a single statement or a group of statements within braces
- For us, it will always be a group of statements within braces



## Example

```
if (value < 0) {  
    value = -value;  
}
```



## Why we always use braces

- What is the output?

```
int m = 5;  
int n = 10;
```

```
if (m > n)  
    ++m;  
    ++n;
```

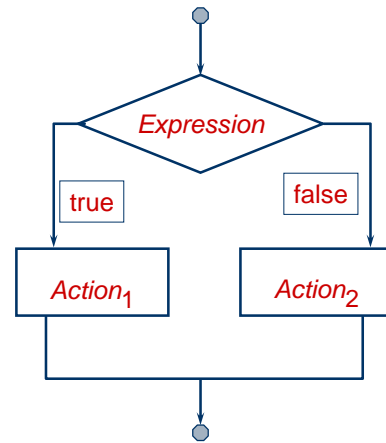
```
System.out.println(" m = " + m + " n = " +  
n);
```

## The if-else statement

- Syntax

```
if (Expression)
    Action1
else
    Action2
```

- If *Expression* is true then execute *Action<sub>1</sub>* otherwise execute *Action<sub>2</sub>*
- The actions are either a single statement or a list of statements within braces



## Testing objects for equality

- Consider

```
// ...
// read two ints, n1 and n2
```

```
if (n1 == n2) {
    System.out.println("Same");
}
else {
    System.out.println("Different");
}
```

What is the output if the user enters 88 both times?  
What is the output if the user enters 88 and 3?

## Testing objects for equality

- Consider

```
// ...
// read two strings, s1 and s2
```

```
if (s1 == s2) {
    System.out.println("Same");
}
else {
    System.out.println("Different");
}
```

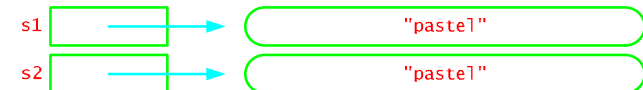
What is the output if the user enters "paste1" both times?

## Testing objects for equality

- When it is executed

```
// ...
// read two strings, s1 and s2
```

- Memory looks like



- As a result no matter what is entered *s1* and *s2* are not the same
  - They refer to different objects

## Testing operators for equality

---

- Consider

```
// ...  
// read two strings, s1 and s2
```

```
if (s1.equals(s2)) {  
    System.out.println("Same");  
}  
else {  
    System.out.println("Different");  
}
```

← Tests whether s1 and s2 represent the same object

All objects have a method `equals()`. Their implementation is class-specific. The String `equals()` method – like many others – tests for equivalence in representation

## Some handy String class methods

---

- `isDigit()`
  - Tests whether character is numeric
- `isLetter()`
  - Tests whether character is alphabetic
- `isLowerCase()`
  - Tests whether character is lowercase alphabetic
- `isWhiteSpace()`
  - Tests whether character is one of the space, tab, formfeed, or newline characters

## Some handy String class methods

---

- `isUpperCase()`
  - Tests whether character is uppercase alphabetic
- `toLowerCase()`
  - If the character is alphabetic then the lowercase equivalent of the character is returned; otherwise, the character is returned
- `toUpperCase()`
  - If the character is alphabetic then the uppercase equivalent of the character is returned; otherwise, the character is returned

## If-else-if

---

- Consider

```
if (number == 0) {  
    System.out.println("zero");  
}  
else {  
    if (number > 0) {  
        System.out.println("positive");  
    }  
    else {  
        System.out.println("negative");  
    }  
}
```

## If-else-if

- Better

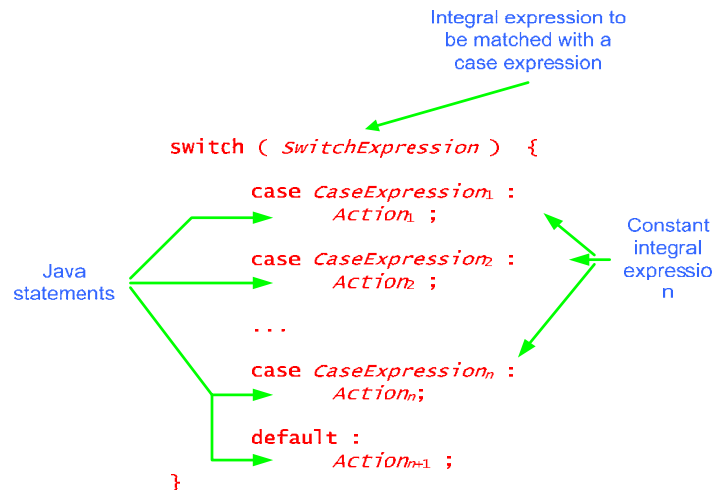
```
if (number == 0) {
    System.out.println("zero");
}
else if (number > 0) {
    System.out.println("positive");
}
else {
    System.out.println("negative");
}
```

Same results as previous segment – but this segment better expresses the meaning of what is going on

## Switch statement

- Software engineers often confronted with programming tasks where required action depends on the values of integer expressions
  - The if-else-if construct can be used
    - Separately compare the desired expression to a particular value
      - If the expression and value are equal, then perform the appropriate action
- Because such programming tasks occur frequently
  - Java includes a switch statement
    - The task is often more readable with the switch than with the if-else-if

## Switch statement



## Testing for vowel-ness

```
switch (ch) {
    case 'a': case 'A':
    case 'e': case 'E':
    case 'i': case 'I':
    case 'o': case 'O':
    case 'u': case 'U':
        System.out.println("vowel ");
        break; ← The break causes an exiting of the switch
    default :
        System.out.println("not a vowel ");
}
```

Handles all of the other cases

## Java looping

### Options

- while
- do-while
- for

- Allow programs to control how many times a statement list is executed

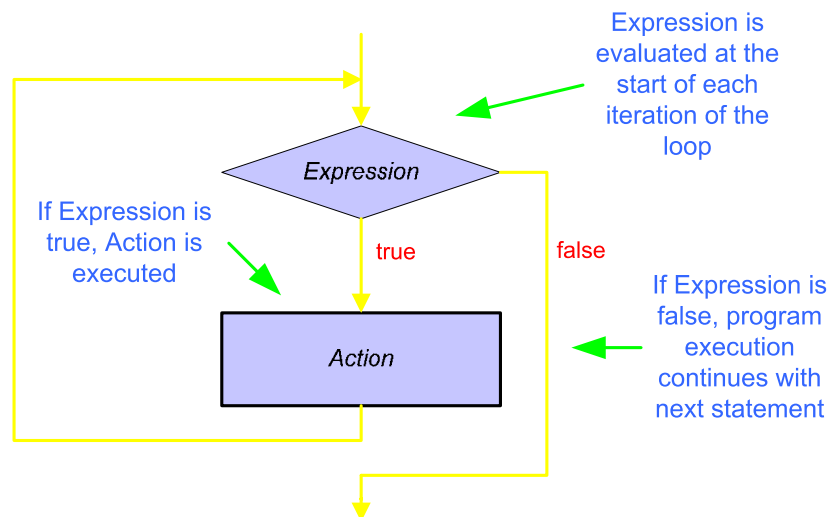
## While syntax and semantics

**while ( Expression ) Action**

Logical expression that determines whether Action is to be executed — if Expression evaluates to true, then Action is executed; otherwise, the loop is terminated

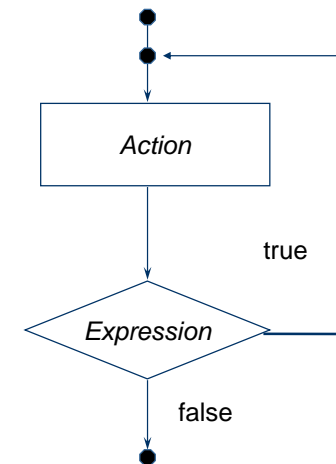
Action is either a single statement or a statement list within braces. The action is also known as the body of the loop. After the body is executed, the test expression is reevaluated. If the expression evaluates to true, the body is executed again. The process repeats until the test expression evaluates to false

## While Semantics



## The do-while statement

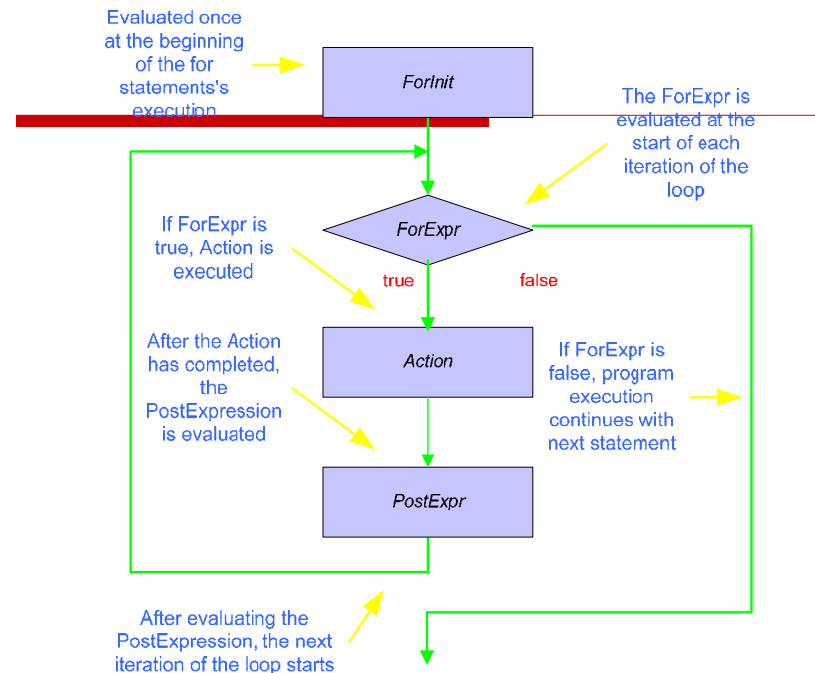
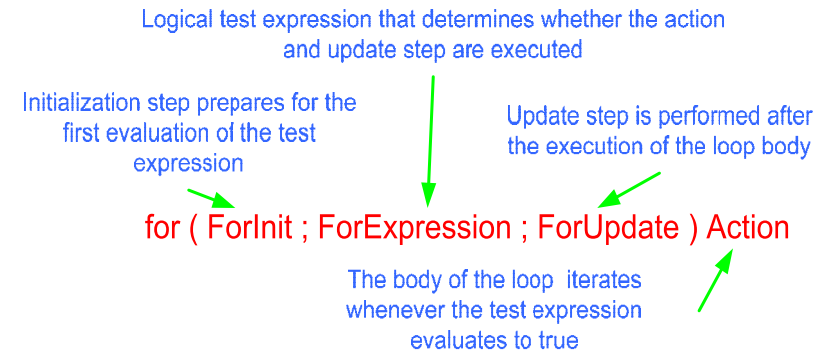
- Syntax
  - do Action
  - while ( Expression )
- Semantics
  - Execute Action
  - If Expression is true then execute Action again
  - Repeat this process until Expression evaluates to false
- Action is either a single statement or a group of statements within braces



# Loop design

- Questions to consider in loop design and analysis
  - What initialization is necessary for the loop's test expression?
  - What initialization is necessary for the loop's processing?
  - What causes the loop to terminate?
  - What actions should the loop perform?
  - What actions are necessary to prepare for the next iteration of the loop?
  - What conditions are true and what conditions are false when the loop is terminated?
  - When the loop completes what actions are need to prepare for subsequent program processing?

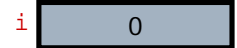
# For statement syntax



# Execution Trace

```
for (int i = 0; i < 3; ++i) {
    System.out.println("i is " + i);
}

System.out.println("all done");
```



## Execution Trace

```
for (int i = 0; i < 3; ++i) {  
    System.out.println("i is " + i);  
}
```

```
System.out.println("all done");
```

i 0

## Execution Trace

```
for (int i = 0; i < 3; ++i) {  
    System.out.println("i is " + i);  
}
```

```
System.out.println("all done");
```

i is 0

i 0

## Execution Trace

```
for (int i = 0; i < 3; ++i) {  
    System.out.println("i is " + i);  
}
```

```
System.out.println("all done");
```

i is 0

i 0

## Execution Trace

```
for (int i = 0; i < 3; ++i) {  
    System.out.println("i is " + i);  
}
```

```
System.out.println("all done");
```

i is 0

i 1

## Execution Trace

```
for (int i = 0; i < 3; ++i) {  
    System.out.println("i is " + i);  
}
```

```
System.out.println("all done");
```

```
i is 0
```

i 1

## Execution Trace

```
for (int i = 0; i < 3; ++i) {  
    System.out.println("i is " + i);  
}
```

```
System.out.println("all done");
```

```
i is 0
```

```
i is 1
```

i 1

## Execution Trace

```
for (int i = 0; i < 3; ++i) {  
    System.out.println("i is " + i);  
}
```

```
System.out.println("all done");
```

```
i is 0
```

```
i is 1
```

i 1

## Execution Trace

```
for (int i = 0; i < 3; ++i) {  
    System.out.println("i is " + i);  
}
```

```
System.out.println("all done");
```

```
i is 0
```

```
i is 1
```

i 2

## Execution Trace

```
for (int i = 0; i < 3; ++i) {  
    System.out.println("i is " + i);  
}
```

```
System.out.println("all done");
```

```
i is 0  
i is 1
```

i 2

## Execution Trace

```
for (int i = 0; i < 3; ++i) {  
    System.out.println("i is " + i);  
}
```

```
System.out.println("all done");
```

```
i is 0  
i is 1  
i is 2
```

i 2

## Execution Trace

```
for (int i = 0; i < 3; ++i) {  
    System.out.println("i is " + i);  
}
```

```
System.out.println("all done");
```

```
i is 0  
i is 1  
i is 2
```

i 2

## Execution Trace

```
for (int i = 0; i < 3; ++i) {  
    System.out.println("i is " + i);  
}
```

```
System.out.println("all done");
```

```
i is 0  
i is 1  
i is 2
```

i 3

## Execution Trace

```
for (int i = 0; i < 3; ++i) {  
    System.out.println("i is " + i);  
}
```

i 3

```
System.out.println("all done");
```

```
i is 0  
i is 1  
i is 2
```

## Execution Trace

```
for (int i = 0; i < 3; ++i) {  
    System.out.println("i is " + i);  
}
```

3

```
System.out.println("all done");
```

```
i is 0  
i is 1  
i is 2  
all done
```

Variable *i* has gone  
out of scope – it  
is *local* to the loop

## Nested loops

```
int m = 2;  
int n = 3;  
for (int i = 0; i < n; ++i) {  
    System.out.println("i is " + i);  
    for (int j = 0; j < m; ++j) {  
        System.out.println("  j is " + j);  
    }  
}
```

## Nested loops

```
int m = 2;  
int n = 3;  
for (int i = 0; i < n; ++i) {  
    System.out.println("i is " + i);  
    for (int j = 0; j < m; ++j) {  
        System.out.println("  j is " + j);  
    }  
}
```

```
i is 0  
  j is 0  
  j is 1  
i is 1  
  j is 0  
  j is 1  
i is 2  
  j is 0  
  j is 1
```

# File Processing: Reading a file

## □ Background

You used this a lot:

```
BufferedReader stdin = new BufferedReader( new InputStreamReader(System.in) );
```

A file is a logical concept that represents a particular hardware device like a monitor or a keyboard, so, you can do the following:

```
BufferedReader fileIn = new BufferedReader( new FileReader("filename") );
```

```
String currentLine = fileIn.readLine();
```

...

```
fileIn.close();
```

Processing data from the standard input stream, i.e., keyboard



# Reading a file

```
BufferedReader stdin = new BufferedReader(  
    new InputStreamReader(System.in));
```

```
System.out.print("Filename: ");
```

```
String filename = stdin.readLine();
```

```
FileReader reader = new FileReader(filename);
```

```
BufferedReader fileIn = new BufferedReader(  
    reader);
```

```
while (fileIn.readLine()) {  
    String currentLine = fileIn.readLine();  
    System.out.println(currentLine);  
}
```

```
fileIn.close();
```

# Example: FileLister.java

```
// Lists contents of a user-specified file  
import java.io.*;  
  
public class FileLister {  
  
    // main(): application entry point  
    public static void main(String [] args) throws IOException {  
        // set up standard input stream  
        BufferedReader stdin = new BufferedReader(  
            new InputStreamReader( System.in ) );  
  
        // determine filename  
        System.out.print("Filename: ");  
        String filename = stdin.readLine();  
  
        // set up file stream  
        BufferedReader fileIn = new BufferedReader(  
            new FileReader( filename ) );  
  
        // process lines one by one  
        String currentLine = fileIn.readLine(); // get first line  
  
        while (currentLine != null) {  
            // display current line  
            System.out.println(currentLine);  
  
            // get next line  
            currentLine = fileIn.readLine();  
        }  
  
        // close the file stream  
        fileIn.close();  
    }  
}
```

*Question to you:  
How to write to  
a file?*

# Assignment, Due Mar. 6, before noon

- 5.2
- 5.34
- 6.23
- Implement a class called DatafileHandler, which can
  1. open and read a data file
  2. count how many data values are in the file
  3. report how many with score "0".
  4. find the maximum value
  5. find the minimum value
  6. calculate the average value

Then write a program to use this class, prompt a user for a filename, then output the results of filename, total numbers, numbers of "0"s, maximum value, minimum value, and average on the screen.

A sample data file is attached.  
(assignment1score.txt)