

8353

Java from the Very Beginning, Part II

Jezz Kelway

Java Technology Centre, z/OS Service.

IBM Hursley Park Labs, United Kingdom

Objectives



- Learn and understand Java's basic flow of control constructs
- Get some more hands on experience

Agenda



- Part I recap
- Conditional statements
 - if..then..else
 - switch
- Looping constructs
 - Arrays
 - for
 - while
 - do..while
- Exception handling

Recap



- Which of the following identifiers are valid?
 - A) `Big0lLongStringWithMeaninglessName`
 - B) `$int`
 - C) `bytes`
 - D) `$1`
 - E) `finalist`

Recap

- What is the range of values that can be assigned to a variable of type short?
 - A) 0 through $2^{16}-1$
 - B) 0 through $2^{32}-1$
 - C) -2^{15} through $2^{15}-1$
 - D) -2^{31} through $2^{31}-1$
 - E) It depends on the underlying hardware

Recap

- What are the values of x, a and b after executing the following code?

```
int x, a=6, b=7;
```

```
x = a++ + b++;
```

- A) x = 15, a = 7, b = 8
- B) x = 15, a = 6, b = 7
- C) x = 13, a = 7, b = 8
- D) x = 13, a = 6, b = 7

Agenda



- Part I recap
- ➔ Conditional statements
 - if..then..else
 - switch
- Looping constructs
 - Arrays
 - for
 - while
 - do..while
- Exception Handling

if...then...else

```
if (boolean_expr) {  
    then_stmts;  
}
```



```
if (a > 10) {  
    System.out.println("a > 10");  
}
```

- Tests against *boolean* expression (not integer as in C/C++).
- Curly braces delimit blocks of code { }
- If the condition is true, then the statements in the then block are executed.

if...then...else

```
if (expr) {  
    then_stmnts;  
}  
else {  
    else_stmnts;  
}
```



```
if (a < 10) {  
    System.out.println("a < 10");  
}  
else {  
    System.out.println("a >= 10");  
}
```

- If the condition is false, then the statements in the else block are executed.

if..then..else

```
if (expr) {  
    then_stmts;  
}  
else if (expr_1) {  
    else_stmts;  
}  
else {  
    else_stmts;  
}
```



```
int testscore;  
char grade;  
  
if (testscore >= 90) {  
    grade = 'A';  
} else if (testscore >= 80) {  
    grade = 'B';  
} else if (testscore >= 70) {  
    grade = 'C';  
} else if (testscore >= 60) {  
    grade = 'D';  
} else {  
    grade = 'F';  
}
```

switch

- Used to make a choice between multiple alternative execution paths
- Choice must be based on an integer type (byte, short, char or int)

switch

- **switch** (int_expr) {

- **case** x:

- case x stmnts;

- **break**;

- **case** y:

- case y stmnts;

- **break**;

- **case** z:

- case z stmnts;

- **break**;

- **default:**

- default case stmnts;

- **break**;

- }



- **switch** (grade) {

- **case** 'A':

- System.out.println("Outstanding!");

- **break**;

- **case** 'B':

- System.out.println("Well done");

- **break**;

- **case** 'C':

- System.out.println("Satisfactory");

- **break**;

- **default:**

- System.out.println("Fail");

- **break**;

- }

switch



- The "default" case is optional
- The "break" statement is optional, if it is omitted, execution drops through to the next case
 - a common source of errors!

switch

```
•switch ( grade ) {  
    •case 'A':  
    •case 'B':  
    •case 'C':  
        •System.out.println("Pass");  
        •break;  
    •default:  
        •System.out.println("Fail");  
        •break;  
•}
```

Agenda



- Part I recap
- Conditional statements
 - if..then..else
 - switch
- ➔ Looping constructs
 - Arrays
 - for
 - while
 - do..while
- Exception Handling

Arrays

- Declare as *type*[] varName;
 - **int[]** myInts;
- Must allocate memory before use:
 - myInts = **new** int[10];
- General form:
 - f* **elementType[]** arrayName = **new elementType[arraySize]**;

Arrays



- Array indices always start at zero.
- Access array elements using []:
 - `myArray[0] = 5;`
- Special array property *length*
 - `myArray.length`

Arrays

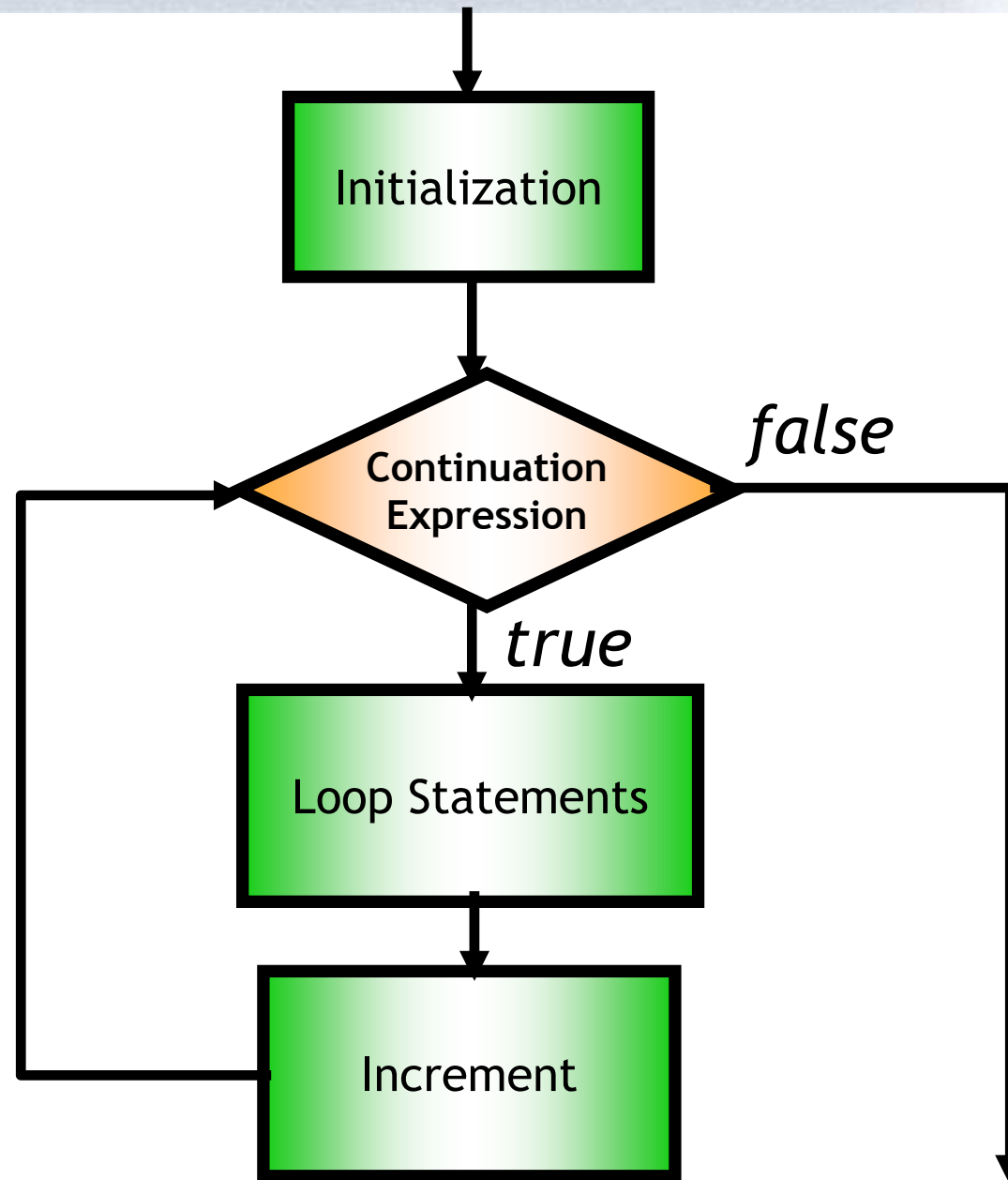
- **int [] squares = new int[5];**
- squares[0] = 0;
- squares[1] = 1;
- squares[2] = 4;
- squares[3] = 9;
- squares[4] = 16;

For loops

```
for (initialisation ; continuation_expr ; increment) {  
    loop_stmts;  
}
```

- ➔ initialisation executed once at beginning
- ➔ increment executed each time round the loop, immediately after the body of the loop
- ➔ continuation_expr is evaluated at the top of the loop on every iteration. The loop terminates when continuation_expr is false.

For loops



For loops

```
int i;
```

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



→ i = 0

→ i = 1

→ i = 2

→ ...

→ i = 9

common shorthand:

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

Exercise



- Print out the command line arguments to a Java program

A Solution



```
/**  
 * A Java application to list the command line arguments  
 */  
class CommandLine {  
  
    public static void main(String [] args) {  
  
        for (int i = 0; i < args.length; i++) {  
            System.out.println("Argument " + i + " = " + args[i]);  
        }  
  
    } // end of main method  
  
} // end of class
```

While Loops

```
while ( boolean_expr ) {  
    stmnts;  
}
```

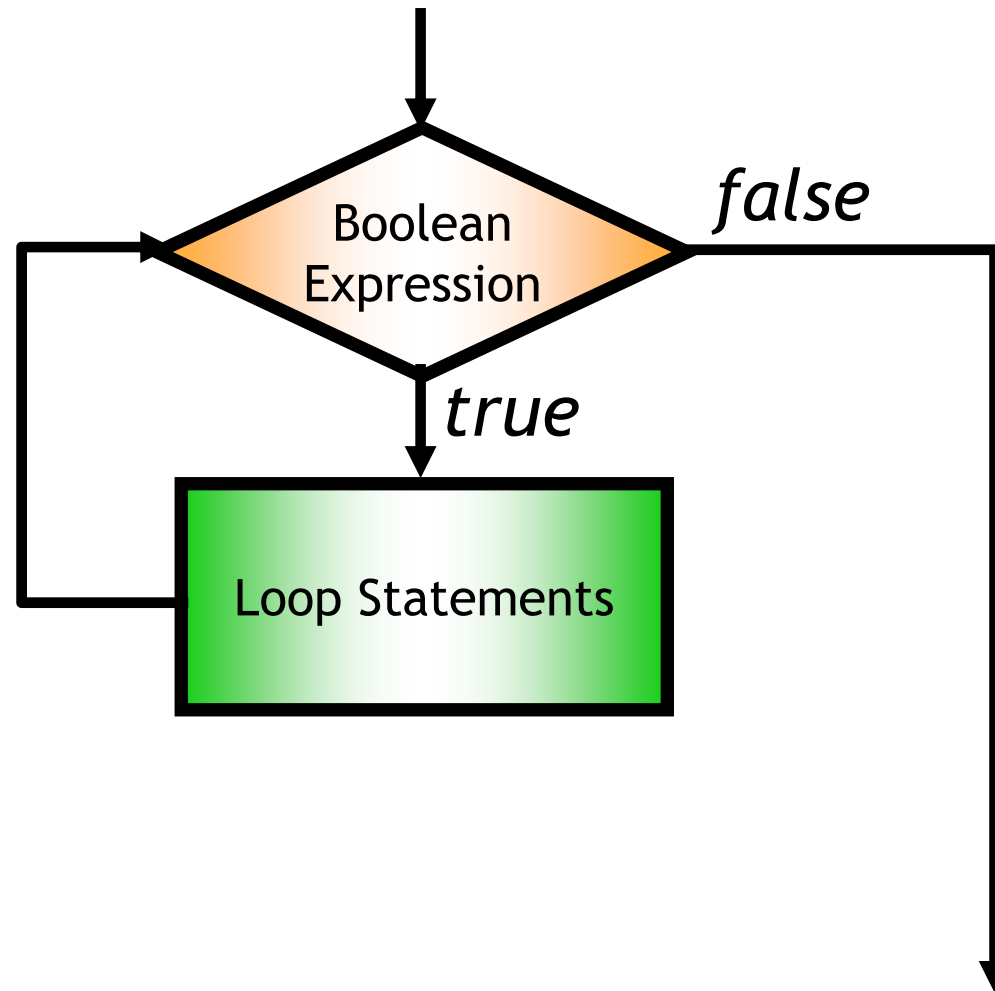


```
int i = 0;  
  
while (i < 10) {  
    System.out.println("i = " + i);  
    i++;  
}
```

- expr evaluated at top of each loop
- body executed if expr evaluates to true
- make sure your loop terminates!

```
→ i = 0  
→ i = 1  
→ i = 2  
→ ...  
→ i = 9
```

While Loops



Do Loops

```
do {  
    stmnts;  
} while ( boolean_expr );
```

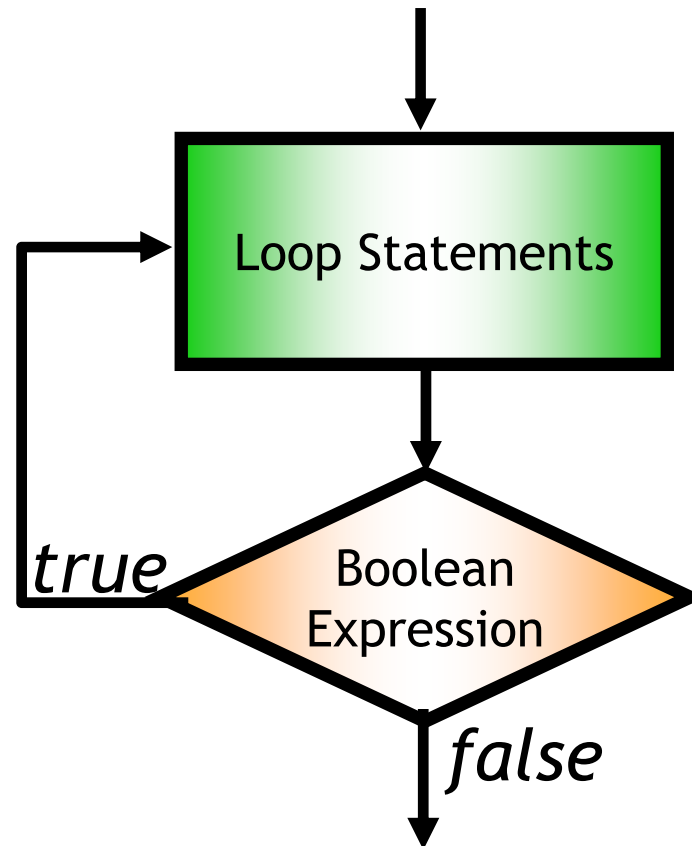


- body executed each time through the loop
- boolean_expr is evaluated at the end of the loop
- body of the loop is always executed at least once

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

```
→ i = 0  
→ i = 1  
→ i = 2  
→ ...  
→ i = 9
```

Do Loops



Continue



- Used to abandon execution of the body of a loop, or a number of nested loops
- A "structured" form of goto...

```
for ( int i = 0; i < array.length; i++ ) {  
    if ( !array[ i ].needsProcessing( ) ) {  
        continue;  
    }  
    // process element...  
}
```

Continue

- Use labels for nested loops
- Can label opening statement of do, while and for loops

```
mainLoop: for ( int i = 0; i < array.length; i++ ) {  
    for ( int j =0; j < array[ i ].length; j++ ) {  
        if ( !array[ i ][ j ].needsProcessing( ) ) {  
            continue mainLoop;  
        }  
        // process element...  
    }  
}
```

Break

- Like continue, but abandons entire loop instead of current iteration
- Can also use labels on break statements

```
for ( int i = 0; i < array.length; i++ ) {  
    if ( array[ i ] == 0 ) {  
        break;    // stop processing at first zero entry  
    }  
    // process element...  
}
```

Agenda



- Part I recap
- Conditional statements
 - if..then..else
 - switch
- Looping constructs
 - Arrays
 - for
 - while
 - do..while

➔ Exception handling

Exceptions and Error Handling



- "An *exception* is an event that occurs during the execution of a program that disrupts the normal flow of instructions".
- When an error occurs within a block of code:
 - ➔ An exception is created containing information about the error
 - ➔ The exception is passed to the runtime system
 - ➔ The runtime system searches backwards through the call stack to find an exception *handler*
 - ➔ if a handler is found, control passes to the handler, else the program exits

Catching Exceptions

- Surround code which may cause an error in a try block, and place one or more catch blocks after it

```
FileReader fileReader;  
try {  
    fileReader = new FileReader("input.txt");  
    // read from file etc...  
    ...  
    fileReader.close(); // done!  
}  
catch (FileNotFoundException notFoundEx) {  
    // handle file not found  
}  
catch (IOException ioEx) {  
    // handle error closing file  
}
```

And Finally

- A finally block may follow a try and its associated catch blocks
- The code in the finally block will always be executed

```
try {  
    ...  
}  
catch (...) { }  
catch (...) { }  
finally {  
    // tidy up code...  
}
```

Exercise



- Improve the "FilePrinter" program so that it handles errors gracefully.

A Solution

```
FileReader fileReader = null; // declare outside of the scope of the try block
```

```
try {  
    fileReader = new FileReader(fileName);  
    int c;  
    while ( (c = fileReader.read()) != -1) {  
        System.out.print((char)c);  
    }  
}  
  
catch (FileNotFoundException notFoundEx) {  
    System.out.println("Could not open " + fileName);  
}  
  
catch (IOException ioEx) {  
    System.out.println("Error reading from " + fileName);  
}  
  
finally {  
    System.out.println();  
    if (fileReader != null) {  
        try { fileReader.close(); }  
        catch (IOException ioEx) { ; } // nothing we can do now!  
    }  
}
```

Summary



- Two main forms of conditional constructs:
 - if..then...else
 - switch
- Three looping constructs
 - for, do, while
- Error handling via try, catch, finally