Java for the Beginner: Part III of III

SHARE Orlando, February 2008

Oliver Fenton

Java Technology Center, IBM Hursley Labs, Winchester, UK

Session: 8354

Trademarks

IBM, AIX, CICS, DB2, IMS, z/OS, OS/390, S/390, System/390, VisualAge, WebSphere Application Server, WebSphere Studio, z/VM - are trademarks or registered trademarks of the IBM Corporation

Sun, Sun Microsystems, JavaSoft, Java, JavaBeans, JDK, Java 2 Micro Edition, J2ME, Java 2 Standard Edition, J2SE, Java 2 Enterprise Edition, J2EE - are trademarks or registered trademarks of Sun Microsystems Inc.

Agenda

- Part II recap
- Looping constructs
  - for
  - while
  - do .. while
- Exception handling
- Exercise 1 and Exercise 2
- Collections classes
  - Problems with arrays
  - Collection classes and the ArrayList
- Exercise 3
- Java 5 and Generics

Recap

- Which of the following identifiers are valid?
  - A) BigOlLongStringWithMeaninglessName
  - 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

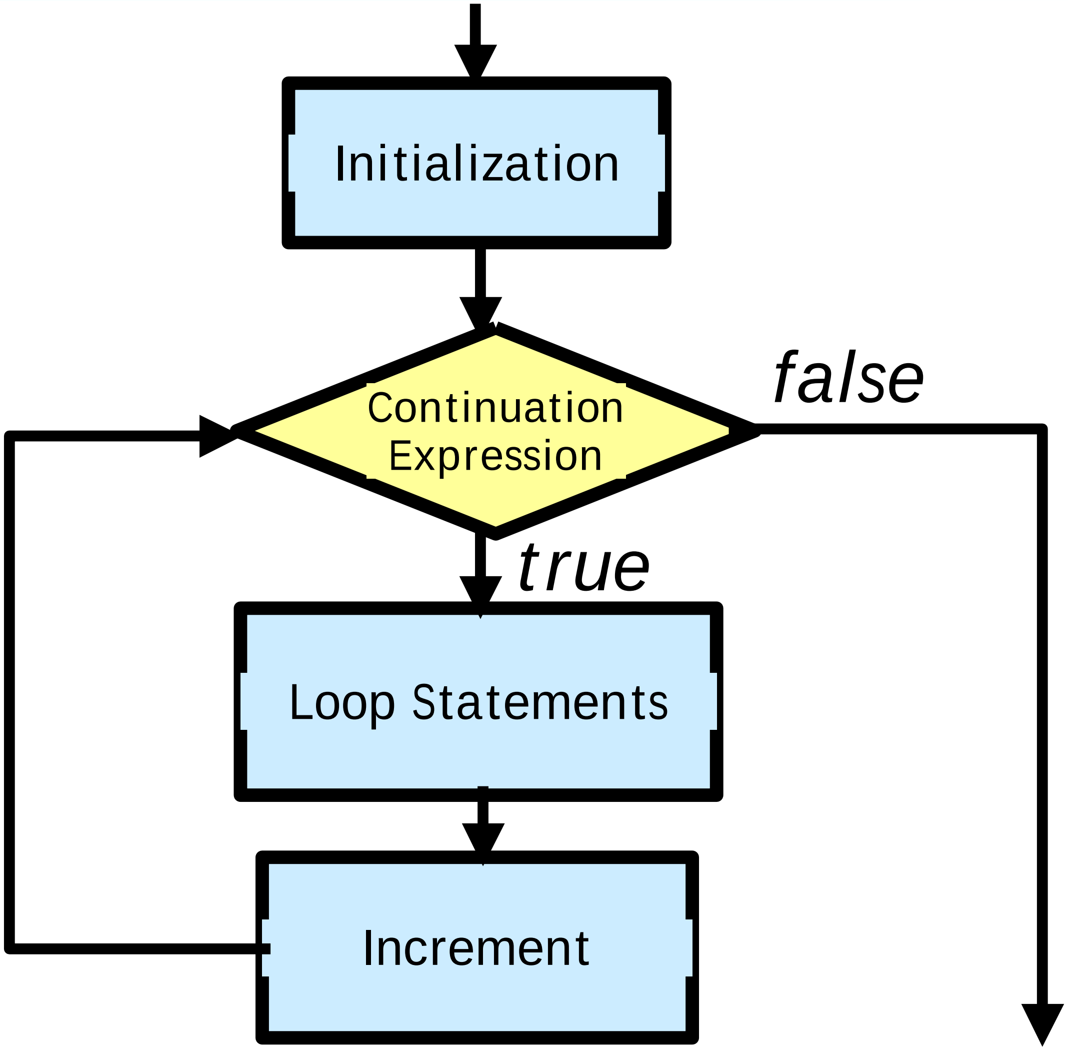
- Part I recap
- Looping constructs
    - for
    - while
    - do .. while
- Exception handling
- Exercise 1 and Exercise 2
- Collections classes
    - Benefits
    - ArrayList
- Exercise 3
- Java 5 and Generics

For loops

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

- initialization 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 construct



For loops example

```
int i;

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

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

- common short hand:

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

While Loops

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

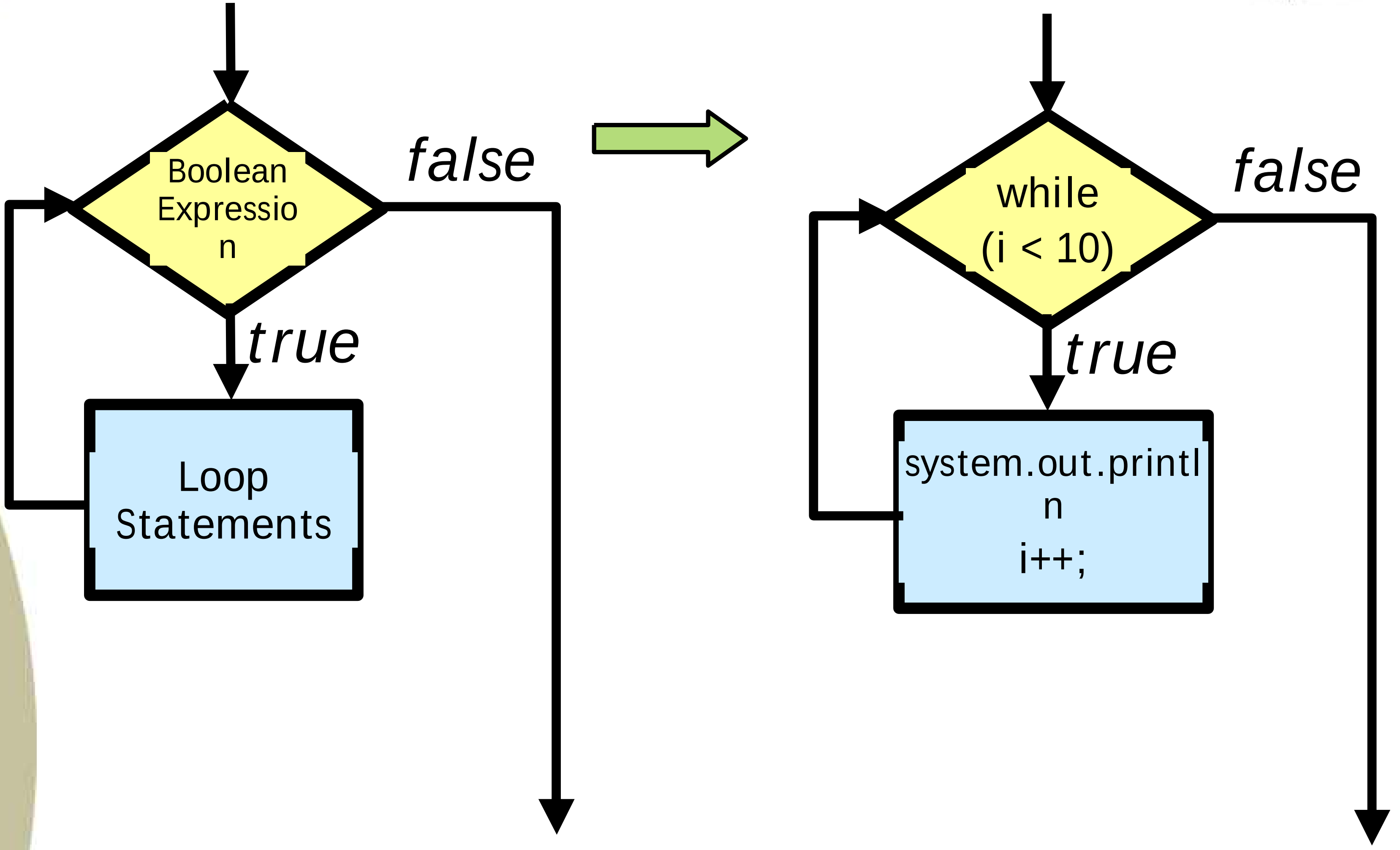
```
int i = 0;

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

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

- boolean_expr evaluated at top of each loop
- Body executed if expr evaluates to true
- Make sure your loop terminates!

While Loops construct



Boolean Expression

false

true

Loop Statements

while (i < 10)

false

true

system.out.println

i++;

do .. while Loops

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

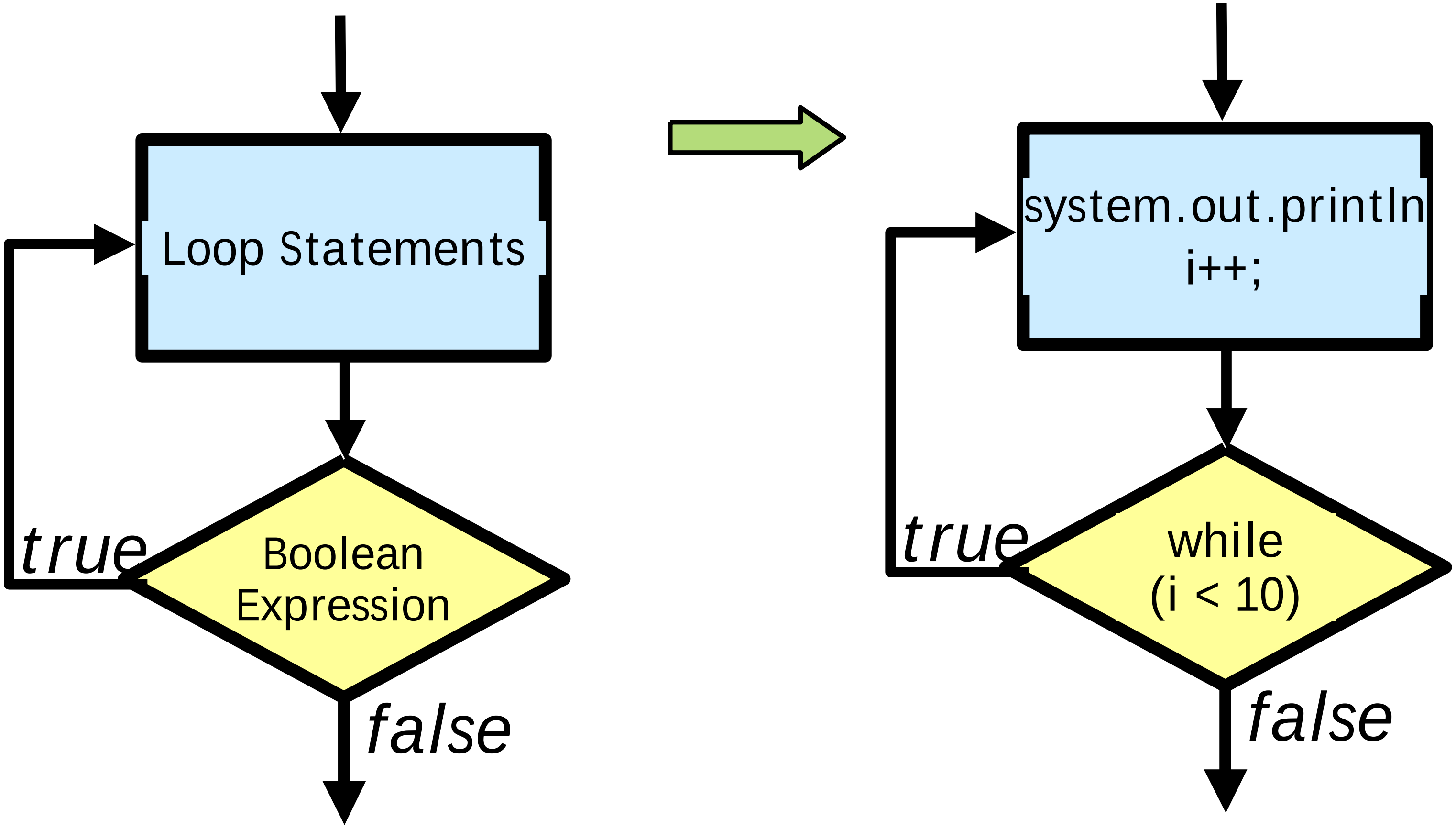
- body executed each time through the loop
- boolean_expr is evaluated at the end of the loop
- body of the loop is always executed at least once

```
int i = 0;

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

➔i = 0
➔i = 1
➔i = 2
➔...
➔i = 9
```

Do Loops



Loop Statements

Boolean Expression

true

false

system.out.println

i++;

while (i < 10)

true

false

Continue statement

- Used to stop / break the current iteration of a loop

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

Continue with Label

- 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
- The break statement has two forms
  - Labeled and unlabeled
  - You can also use an unlabeled break to terminate a for, while, or do-while loop

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

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

The return statement

- The last of the branching statements is the return statement
    - It exits from the current method
    - The control flow returns to where the method was invoked
- The return statement has two forms:
    - One that returns a value -- return ++count;
    - One that doesn't -- return;
- To return a value, simply put the value (or an expression that calculates the value) after the return keyword as indicated above

Agenda

- Part I recap
- Looping constructs
  - for
  - while
  - do .. while
- Exception handling
- Exercise 1 and Exercise 2
- Collections classes
  - Problems with arrays
  - Collection classes and the ArrayList
- Exercise 3
- Java 5 and Generics

Exceptions and Error Handling

- “An exception is an event that occurs during the execution of a program that disrupts the normal flow of the instructions”.
- When an error occurs within a block of code:
  - An 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 a finally block will always be executed

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

Exercise 1

- Print out the Command Line Arguments to a Java program

A Solution for Exercise 1

```
/**
 * 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
```

Exercise 2

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

A Solution for Exercise 2

```
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!
    }
  }
```

Agenda

- Part I recap
- Looping constructs
  - for
  - while
  - do .. while
- Exception handling
- Exercise 1 and Exercise 2
- Collections classes
  - Problems with arrays
  - Collection classes and the ArrayList
- Exercise 3
- Java 5 and Generics

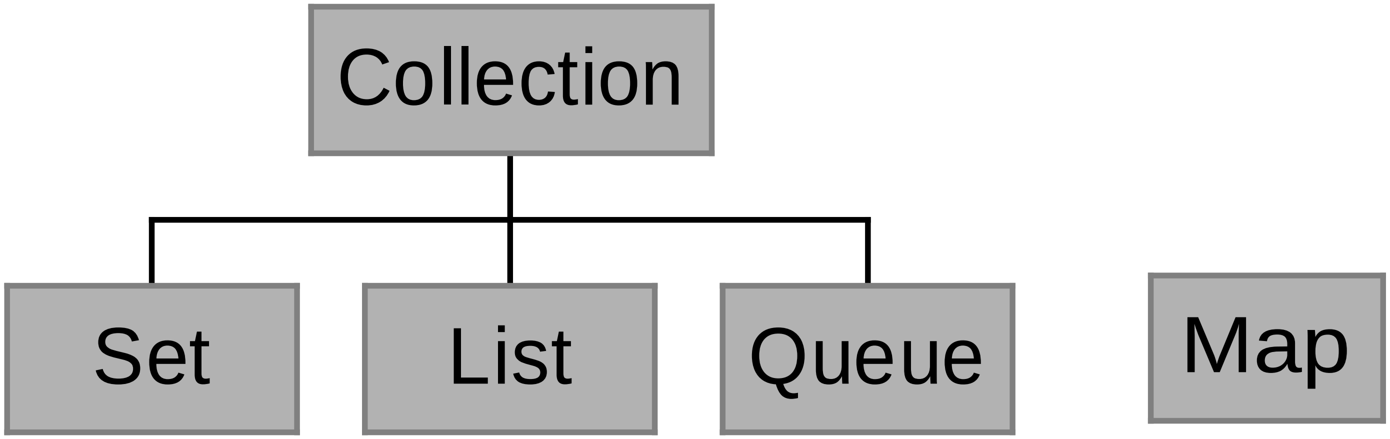
Problems using arrays

- May not know size up front
  - Unable to grow size
- Rigid structures
  - May want unordered container
- Use java.util.collections
  - Provide already defined data structures

Collection Classes

- What is available
  - Set
    - cannot contain duplicate elements
  - List
    - ordered collection or sequence
    - can contain duplicate elements
  - Queue
    - hold multiple elements prior to processing
    - additional insertion, extraction, and inspection operations
  - Map
    - maps keys to values
    - cannot contain duplicate keys

Collection Classes



Collection

Set

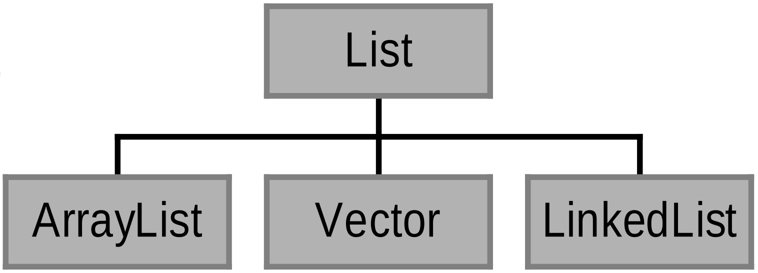
List

Queue

Map

ArrayList

- import java.utils.*;
- List myList = new ArrayList();
  - Why use List rather than ArrayList
- myList.add(Object)
- myList.get(int)
- myList.contains(Object)



ArrayList

- Complete list of methods

  http://java.sun.com/j2se/1.5.0/docs/api/java/util/ArrayList.html

ArrayList

```
import java.util.*;

public class ArrayListExample1 {

    public static void main(String[] args) {

        List theChildren = new ArrayList();

        theChildren.add("Jon");
        theChildren.add("Jane");

        System.out.println("number of children: " + theChildren.size());
        System.out.println("First item: " + theChildren.get(0));
        System.out.println("Second item: " + theChildren.get(1));
    }
}
```

Exercise 3

- Modify the CommandLine program to store the arguments in a ArrayList
- Query this array list to see if it contains a specific value

Problem

```
try {
    System.out.println("Last element: " +
   theArguments.get(theArguments.size()-1));
} catch (java.lang.ArrayIndexOutOfBoundsException e) {
        System.out.println("Accessing array with: " +
                (theArguments.size()-1));
        e.printStackTrace();
}
```

Agenda

- Part I recap
- Looping constructs
  - for
  - while
  - do .. while
- Exception handling
- Exercise 1 and Exercise 2
- Collections classes
  - Problems with arrays
  - Collection classes and the ArrayList
- Exercise 3
- Java 5 and Generics

Problems with the exercise

- The solution worked without warnings for Java 1.4.2 but not for Java 5
- Against theArguments.add(i); is the warning:

Type safety: The method addElement(Object) belongs to the raw type Vector. References to generic type Vector<E> should be parameterized

Java 1.4.2 and type checking

- In Java 1.4.2 (and before) type checking was the responsibility of the programmer. List entries are of class Object.

```
List theArguments = new ArrayList();

theArguments.add((String) "Hello");

String element = (String) the Arguments.get(0);
```

- Not very nice, the program has to do all the type checking

  1. Prone to mishtakes
  2. Casting produces ugly code

Java 5 and Generics

- Java 5 introduces Generics

  http://java.sun.com/j2se/1.5/pdf/generics-tutorial.pdf

  // Before Java 5 ArrayLists entries are Objects

  List theArguments = new ArrayList();

  // In Java 5 it is possible to define the class of entries eg String

  List <String> theArguments = new ArrayList <String> ();

Java 5 and Generics

- So instead of (pre Java 5)

```
List theArguments = new ArrayList();
theArguments.add((String) "Hello");
String element = (String) the Arguments.get(0);
```

- we have:

```
List <String> theArguments = new ArrayList <String> ();
theArguments.add("Hello");
String element = the Arguments.get(0);
```

- Now the JVM does the type checking

Review

- Part I recap
- Looping constructs
  - for
  - while
  - do .. while
- Exception handling
- Exercise 1 and Exercise 2
- Collections classes
  - Problems with arrays
  - Collection classes and the ArrayList
- Exercise 3
- Java 5 and Generics