

## 4 Grouping objects (cont.)

### Iteration

### Main concepts to be covered

- Iteration with **Loops**
- Collection traversal with **Iterators**
- Fixed-size collections - **Arrays**

Object oriented programming, DAT042, D2, 11/12, Ip 1

Lecture 4 2

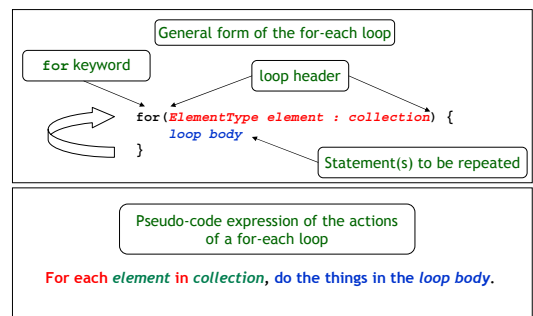
### Iteration

- We often want to perform some actions an arbitrary number of times.
  - E.g., print all the elements in a collection.
- Most programming languages include *loop statements* to make this possible.
- Java has several sorts of loop statement.
  - We will start with its *for-each* loop.
- With collections, we often want to repeat things once for every object in a particular collection.

Object oriented programming, DAT042, D2, 11/12, Ip 1

Lecture 4 3

### For-each loop pseudo code



Object oriented programming, DAT042, D2, 11/12, Ip 1

Lecture 4 4

### A Java example

```
/**  
 * List all notes in the notebook.  
 */  
public void listNotes()  
{  
    for(String note : notes) {  
        System.out.println(note);  
    }  
}
```

for each note in notes, print out note

Object oriented programming, DAT042, D2, 11/12, Ip 1

Lecture 4 5

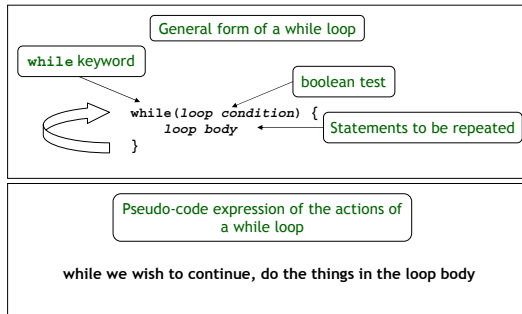
### The while loop

- A for-each loop repeats the loop body for each object in a collection.
- Sometimes we require more variation than this.
- We can use a boolean condition to decide whether or not to keep going.
- A while loop provides this control.

Object oriented programming, DAT042, D2, 11/12, Ip 1

Lecture 4 6

## While loop pseudo code



Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 4 7

## A Java example

```

private ArrayList<String> notes;
...

/**
 * List all notes in the notebook.
 */
public void listNotes()
{
    int index = 0;
    while(index < notes.size()) {
        System.out.println(notes.get(index));
        index++;
    }
}
    
```

**Increment *index* by 1**

**while the value of *index* is less than the size of the collection, print the next note, and then increment *index***

Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 4 8

## Using a non-generic list

```

private ArrayList notes;
// "Old" Java style
// the list elements have type Object
...

/**
 * List all notes in the notebook.
 */
public void listNotes()
{
    int index = 0;
    while(index < notes.size()) {
        String note = (String)notes.get(index);
        System.out.println(note);
        index++;
    }
}
    
```

**type cast Object to String**

Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 4 9

## for-each versus while

- for-each:
  - easier to write.
  - safer: it is guaranteed to stop.
- while:
  - we don't *have* to process the whole collection.
  - doesn't even have to be used with a collection.
  - take care: could be an *infinite loop*.

Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 4 10

## Searching a collection using a boolean

```

int index = 0;
boolean found = false;
while(index < notes.size() && !found) {
    String note = notes.get(index);
    if(note.contains(searchString)) {
        // We don't need to keep looking.
        found = true;
    }
    else {
        index++;
    }
}
// Either we found it (found is true),
// or we searched the whole collection
// (found is false).
    
```

Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 4 11

## Searching a collection using break

```

int index = 0;
while(index < notes.size()) {
    String note = notes.get(index);
    if(note.contains(searchString)) {
        // We don't need to keep looking.
        break;
    }
    else {
        index++;
    }
}
// Either we found it: index < notes.size()
// or we searched the whole collection:
// index == notes.size();
    
```

Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 4 12

## Searching a collection using a smarter condition

```
int index = 0;
while( index < notes.size() &&
      ! notes.get(index).contains(searchString) )
    index++;

// Either we found it: index < notes.size()
// or we searched the whole collection:
// index == notes.size();
```

Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 4 13

## Using an Iterator object

```
java.util.Iterator returns an Iterator object
Iterator<ElementType> it = myCollection.iterator();
while(it.hasNext()) {
    call it.next() to get the next object
    do something with that object
}
```

```
public void listNotes()
{
    Iterator<String> it = notes.iterator();
    while(it.hasNext()) {
        System.out.println(it.next());
    }
}
```

Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 4 14

## Using an Iterator object

```
Iterator<Integer> it = myCollection.iterator();
while(it.hasNext()) {
    if( it.next() > 0 ) {
        System.out.println(it.next());
    }
}
```

What's wrong here?

Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 4 15

## Using an Iterator object

```
Iterator<Integer> it = myCollection.iterator();
while(it.hasNext()) {
    int element = it.next();
    if( element > 0 ) {
        System.out.println(element);
    }
}
```

Normally next() is called once in each iteration

Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 4 16

## Index versus Iterator

- Ways to iterate over a collection:
  - for-each loop.
    - Use if we want to process every element.
  - while loop.
    - Use if we might want to stop part way through.
    - Use for repetition that doesn't involve a collection.
  - Iterator object.
    - Use if we might want to stop part way through.
    - Often used with collections where indexed access is not very efficient, or impossible.
- Iteration is an important *programming pattern*.

Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 4 17

## Collections and primitive types

- Java collections such as ArrayList can only store object types - not primitive types.
- Each primitive type has a corresponding *wrapper class* that stores a primitive value as an object (Boxing).

```
ArrayList<Integer> al = new ArrayList<Integer>();
int i = 123;
al.add(new Integer(i));
i = (al.get(0)).intValue();
```

box an int

unbox the Integer

Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 4 18

## Collections and primitive types

- Java 5 (and later versions) performs boxing and unboxing automatically!

```
ArrayList<Integer> al = new ArrayList<Integer>();  
  
int i = 123;  
  
al.add(i);           // auto boxing  
  
i = notes.get(0);    // auto unboxing
```

Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 4 19

## The *auction* project

- The *auction* project provides further illustration of collections and iteration.
- One further point to follow up: the `null` value.
  - Used to indicate, 'no object'.
  - We can test if an object variable holds the null value.

Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 4 20

## Review

- Loop statements allow a block of statements to be repeated.
- The for-each loop allows iteration over a whole collection.
- The while loop allows the repetition to be controlled by a boolean expression.
- All collection classes provide special *Iterator* objects that provide sequential access to a whole collection.

Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 4 21

## Fixed-size collections

- Sometimes the maximum collection size can be pre-determined.
- Programming languages usually offer a special fixed-size collection type: an *array*.
- Java arrays can store objects or primitive-type values.
- Arrays use a special syntax.

Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 4 22

## The *weblog-analyzer* project

- Web server records details of each access.
- Supports webmaster's tasks.
  - Most popular pages.
  - Busiest periods.
  - How much data is being delivered.
  - Broken references.
- Analyze accesses by hour.

Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 4 23

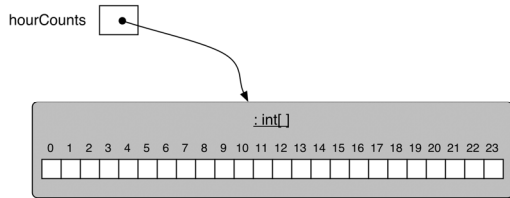
## Creating an array object

```
public class LogAnalyzer  
{  
    private int[] hourCounts; ← Array variable declaration  
    private LogfileReader reader;  
  
    public LogAnalyzer()  
    {  
        hourCounts = new int[24]; ← Array object creation  
        reader = new LogfileReader();  
    }  
    ...  
}
```

Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 4 24

## The hourCounts array



Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 4 25

## Using an array

- Square-bracket notation is used to access an array element: `hourCounts[...]`
- Elements are used like ordinary variables.
  - On the left of an assignment:
    - `hourCounts[hour] = ...;`
  - In an expression:
    - `adjusted = hourCounts[hour] - 3;`
    - `hourCounts[hour]++;`

Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 4 26

## Using an array

- Initialization

```
ElemType[] arr = {value1,value2,...}
arr = new ElemType[] {value1,value2,...}
```
  - Number of elements

```
arr.length
```
  - Assignment and copy
    - Shallow copy:

```
arr2 = arr1 // reference copy
```
    - Deep copy (=cloning):

```
arr2 = (ElemType[])arr1.clone()
System.arraycopy(arr1,pos1,arr2,pos2,n)
```
  - Comparison

```
arr1 == arr2 // identity
Arrays.equals(arr1,arr2) // equality
```
- See `java.util.Arrays` for more array utilities

Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 4 27

## Array traversal with for-loops

### for loop version

```
for(int hour = 0; hour < hourCounts.length; hour++) {
    System.out.println(hour + ": " + hourCounts[hour]);
}
```

### for-each loop version

```
for(int count : hourCounts) {
    System.out.println(hour + ": " + count);
}
```

Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 4 28

## Review

- Arrays are appropriate where a fixed-size collection is required.
- Arrays use special syntax.
- For loops offer an alternative to while loops when the number of repetitions is known.
- For loops are used when an index variable is required.

Object oriented programming, DAT042, D2, 11/12, lp 1

Lecture 4 29