Grouping objects

Introduction to collections

The requirement to group objects

- Many applications involve collections of objects:
  - Personal organizers.
  - Library catalogs.
  - Student-record system.
- The number of items to be stored varies.
  - Items added.
  - Items deleted.
A personal notebook

- Notes may be stored.
- Individual notes can be viewed.
- There is no limit to the number of notes.
- It will tell how many notes are stored.
- Explore the notebook1 project.

Class libraries

- Collections of useful classes.
- We don’t have to write everything from scratch.
- Java calls its libraries, packages.
- Grouping objects is a recurring requirement.
  - The java.util package contains classes for doing this.
import java.util.ArrayList;
/**
 * ...
 */
public class Notebook {
    // Storage for an arbitrary number of notes.
    private ArrayList<String> notes;

    /**
     * Perform any initialization required for the
     * notebook.
     */
    public Notebook() {
        notes = new ArrayList<String>();
    }
    ...
}

Collections

- We specify:
  - the type of collection: ArrayList
  - the type of objects it will contain: <String>
- We say, “ArrayList of String”.
Object structures with collections

Adding a third note
Features of the collection

- It increases its capacity as necessary.
- It keeps a private count (`size()` accessor).
- It keeps the objects in order.
- Details of how all this is done are hidden.
  - Does that matter? Does not knowing how prevent us from using it?

Using the collection

```java
public class Notebook {

    private ArrayList<String> notes;
    ... 

    public void storeNote(String note) {
        notes.add(note);
    }

    public int numberOfNotes() {
        return notes.size();
    }

    ... 
}
```

Adding a new note

Returning the number of notes (delegation)
Index numbering

```
public void showNote(int noteNumber)
{
    if(noteNumber < 0) {
        // This is not a valid note number.
    }
    else if(noteNumber < numberOfNotes()) {
        System.out.println(notes.get(noteNumber));
    }
    else {
        // This is not a valid note number.
    }
}
```
Generic classes

- Collections are known as parameterized or generic types.
- `ArrayList` implements list functionality:
  - `add`, `get`, `size`, etc.
- The type parameter says what we want a list of:
  - `ArrayList<Person>`
  - `ArrayList<TicketMachine>`
  - etc.
Review

• Collections allow an arbitrary number of objects to be stored.
• Class libraries usually contain tried-and-tested collection classes.
• Java’s class libraries are called packages.
• We have used the ArrayList class from the java.util package.

Review

• Items may be added and removed.
• Each item has an index.
• Index values may change if items are removed (or further items added).
• The main ArrayList methods are add, get, remove and size.
• ArrayList is a parameterized or generic type.
Grouping objects

Collections and the for-each loop

Interlude:
Some popular errors...
There is a boolean field called ‘isEmpty’...

```java
/**
 * Print out notebook info (number of entries).
 */
public void showStatus()
{
    if(isEmpty == true) {
        System.out.println("Notebook is empty");
    }
    else {
        System.out.print("Notebook holds ");
        System.out.println(notes.size() + " notes");
    }
}
```

What’s wrong here?

This time I have a boolean field called ‘isEmpty’...

```java
/**
 * Print out notebook info (number of entries).
 */
public void showStatus()
{
    if(isEmpty == true) {
        System.out.println("Notebook is empty");
    }
    else {
        System.out.print("Notebook holds ");
        System.out.println(notes.size() + " notes");
    }
}
```

The correct version
What’s wrong here?

/**
 * Store a new note in the notebook. If the notebook is full, save it and start a new one.
 */
public void addNote(String note) {
    if(notes.size() == 100)
        notes.save();
    // starting new notebook
    notes = new ArrayList<String>();

    notes.add(note);
}

This is the same.

/**
 * Store a new note in the notebook. If the notebook is full, save it and start a new one.
 */
public void addNote(String note) {
    if(notes.size() == 100)
        notes.save();

    // starting new notebook
    notes = new ArrayList<String>();

    notes.add(note);
}
** The correct version

```java
/**
 * Store a new note in the notebook. If the
 * notebook is full, save it and start a new one.
 */
public void addNote(String note)
{
    if(notes.size == 100) {
        notes.save();
        // starting new notebook
        notes = new ArrayList<String>();
    }

    notes.add(note);
}
```

Iteration

- We often want to perform some actions an arbitrary number of times.
  - E.g., print all the notes in the notebook. How many are there?
- 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.
Iteration fundamentals

- We often want to repeat some actions over and over.
- Loops provide us with a way to control how many times we repeat those actions.
- With collections, we often want to repeat things once for every object in a particular collection.

For-each loop pseudo code

For each element in collection, do the things in the loop body.
A Java example

```java
/**
 * 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

Review

- Loop statements allow a block of statements to be repeated.
- The for-each loop allows iteration over a whole collection.
Grouping objects

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.
### While loop pseudo code

**General form of a while loop**

```
while (loop condition) {
  loop body
}
```

**Pseudo-code expression of the actions of a while loop**

```
while we wish to continue, do the things in the loop body
```

### A Java example

```java
/**
 * 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*
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.

While without a collection

```java
// Print all even numbers from 0 to 30.
int index = 0;
while(index <= 30) {
    System.out.println(index);
    index = index + 2;
}
```
Searching a collection

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, or we searched the whole
// collection.

Grouping objects

Identity vs equality
Side note: String equality

```java
if(input == "bye") {
    ...
}
if(input.equals("bye")) {
    ...
}
```

Strings should always be compared with `.equals`

Identity vs equality 1

Other (non-String) objects:

```java
:Person
  "Fred"
```

```java
:Person
  "Jill"
```

```java
person1 == person2 ?
```
Identity vs equality 2

Other (non-String) objects:

\[ \text{:Person} \quad \text{“Fred”} \quad \text{person1} \quad \text{:Person} \quad \text{“Fred”} \quad \text{person2} \]

\[ \text{person1 \ == \ person2 ?} \]

Identity vs equality 3

Other (non-String) objects:

\[ \text{:Person} \quad \text{“Fred”} \quad \text{person1} \quad \text{:Person} \quad \text{“Fred”} \quad \text{person2} \]

\[ \text{person1 \ == \ person2 ?} \]
Identity vs equality (Strings)

String input = reader.getInput();
if(input == "bye") {
    ...
}

Equality vs identity (may be) false!

Identity vs equality (Strings)

String input = reader.getInput();
if(input.equals("bye")) {
    ...
}

Equality tests equality true!
Grouping objects

Identity vs equality

myList: List

myList.iterator()
Element e = myList.iterator().next();
myList: List

hasNext()?

next()
Using an Iterator object

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

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

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 variable.
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.

Grouping objects

Arrays
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.

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.
Creating an array object

```java
public class LogAnalyzer
{
    private int[] hourCounts;  // Array variable declaration
    private LogfileReader reader;

    public LogAnalyzer()
    {
        hourCounts = new int[24];  // Array object creation
        reader = new LogfileReader();
    }
    ...
}
```

The hourCounts array

![Diagram of the hourCounts array](image)
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]++;`

Standard array use

```java
private int[] hourCounts;
private String[] names;
...

hourCounts = new int[24];
...

hourcounts[i] = 0;
hourcounts[i]++;
System.out.println(hourcounts[i]);
```
Array literals

private int[] numbers = { 3, 15, 4, 5 };

System.out.println(numbers[i]);

- Array literals can only be used in initialisations.

Array length

private int[] numbers = { 3, 15, 4, 5 };

int n = numbers.length;

- Note: ‘length’ is not a method!!
The for loop

- There are two variations of the for loop, *for-each* and *for*.
- The for loop is often used to iterate a fixed number of times.
- Often used with a variable that changes a fixed amount on each iteration.

For loop pseudo-code

General form of a for loop

```java
for(initialization; condition; post-body action) {
    statements to be repeated
}
```

Equivalent in while-loop form

```java
initialization;
while(condition) {
    statements to be repeated
    post-body action
}
```
A Java example

**for loop version**

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

**while loop version**

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

Practice

- Given an array of numbers, print out all the numbers in the array, using a for loop.

```java
int[] numbers = { 4, 1, 22, 9, 14, 3, 9};
for ...
```
Practice

• Fill an array with the Fibonacci sequence.

0 1 1 2 3 5 8 13 21 34 ...

```java
int[] fib = new int[100];
fib[0] = 0;
fib[1] = 1;
for ...
```

for loop with bigger step

```java
// Print multiples of 3 that are below 40.
for(int num = 3; num < 40; num = num + 3) {
    System.out.println(num);
}
```
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.