More sophisticated behavior

Using library classes to implement some more advanced functionality

Main concepts to be covered

• Using library classes
• Reading documentation
The Java class library

- Thousands of classes
- Tens of thousands of methods
- Many useful classes that make life much easier
- A competent Java programmer must be able to work with the libraries.

Working with the library

You should:
- know some important classes by name;
- know how to find out about other classes.

Remember:
- We only need to know the interface, not the implementation.
A Technical Support System

- A textual dialog system
- Idea based on ‘Eliza’ by Joseph Weizenbaum (MIT, 1960s)
- *Explore...*

Main loop structure

```java
boolean finished = false;

while(!finished) {
    do something
    if(exit condition) {
        finished = true;
    } else {
        do something more
    }
}
```
Main loop body

```java
String input = reader.getInput();
...
String response = responder.generateResponse();
System.out.println(response);
```

The exit condition

```java
String input = reader.getInput();

if(input.startsWith("bye")) {
    finished = true;
}
```

- Where does ‘startsWith’ come from?
- What is it? What does it do?
- How can we find out?
Reading class documentation

- Documentation of the Java libraries in HTML format;
- Readable in a web browser
- Class API: *Application Programmers’ Interface*
- Interface description for all library classes

Interface vs implementation

*The documentation includes*
- the name of the class;
- a general description of the class;
- a list of constructors and methods
- return values and parameters for constructors and methods
- a description of the purpose of each constructor and method

the *interface* of the class
Interface vs implementation

*The documentation does not include*

- private fields (most fields are private)
- private methods
- the bodies (source code) for each method

---

Using library classes

- Classes from the library must be imported using an *import* statement (except classes from `java.lang`).
- They can then be used like classes from the current project.
Improve input processing

- What happens if we type “Bye” or “bye”?
  - String input = reader.getInput().trim();
  - if(input.toLowerCase().equals("bye"))

Immutable Objects

- An object is said to be immutable if its contents or state cannot be changed once it has been created.
- Strings are an example of immutable objects.
Packages and import

- Classes are organised in packages.
- Single classes may be imported:
  
  ```java
  import java.util.ArrayList;
  ```

- Whole packages can be imported:

  ```java
  import java.util.*;
  ```

Using Random

- The library class `Random` can be used to generate random numbers

  ```java
  import java.util.Random;
  ...
  Random randomGenerator = new Random();
  ...
  int index1 = randomGenerator.nextInt();
  int index2 = randomGenerator.nextInt(100);
  ```
Generating random responses

```java
public Responder()
{
    randomGenerator = new Random();
    responses = new ArrayList<String>();
    fillResponses();
}

public String generateResponse()
{
    int index = randomGenerator.nextInt(responses.size());
    return responses.get(index);
}

public void fillResponses()
{
    ...
}
```

Parameterized class

- **ArrayList<E>:** The class definition needs to be completed with type names in angle brackets
  - boolean add(E o)
  - E get(index)
Review

• Java has an extensive class library.
• A good programmer must be familiar with the library.
• The documentation tells us what we need to know to use a class (interface).

More sophisticated behaviour

Using library classes to implement some more advanced functionality
What’s wrong with this?

class NoteBook {
    private ArrayList<String> notes;

    public NoteBook() {
        ArrayList<String> notes = new ArrayList<String>();
    }

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

Main concepts to be covered

- More library classes
- Writing documentation
Using sets

```java
import java.util.HashSet;
...
HashSet<String> mySet = new HashSet<String>();
mySet.add("one");
mySet.add("two");
mySet.add("three");

for(String element : mySet) {
    do something with element
}
```

Compare this to ArrayList code!

Tokenising Strings

```java
public HashSet<String> getInput()
{
    System.out.print("> ");
    String inputLine =
        reader.nextLine().trim().toLowerCase();

    String[] wordArray = inputLine.split(" ");
    HashSet<String> words = new HashSet<String>();

    for(String word : wordArray) {
        words.add(word);
    }
    return words;
}
```
Maps

- Maps are collections that contain pairs of values.
- Pairs consist of a **key** and a **value**.
- Lookup works by supplying a key, and retrieving a value.
- An example: a telephone book.

Using maps

- A map with Strings as keys and values

```java
HashMap

"Charles Nguyen" "(531) 9392 4587"
"Lisa Jones" "(402) 4536 4674"
"William H. Smith" "(998) 5488 0123"
```
Using maps

HashMap <String, String> phoneBook =
new HashMap<String, String>();

phoneBook.put("Charles Nguyen", "(531) 9392 4587");
phoneBook.put("Lisa Jones", "(402) 4536 4674");
phoneBook.put("William H. Smith", "(998) 5488 0123");

String phoneNumber = phoneBook.get("Lisa Jones");
System.out.println(phoneNumber);

Writing class documentation

• Your own classes should be documented the same way library classes are.
• Other people should be able to use your class without reading the implementation.
• Make your class a 'library class'!
Elements of documentation

*Documentation for a class should include:*  
- the class name  
- a comment describing the overall purpose and characteristics of the class  
- a version number  
- the authors’ names  
- documentation for each constructor and each method

Elements of documentation

*The documentation for each constructor and method should include:*  
- the name of the method  
- the return type  
- the parameter names and types  
- a description of the purpose and function of the method  
- a description of each parameter  
- a description of the value returned
Class comment:

/**
 * The Responder class represents a response generator object. It is used to generate an automatic response.
 * @author Michael Kölling and David J. Barnes
 * @version 1.0 (30.Mar.2006)
 */

Method comment:

/**
 * Read a line of text from standard input (the text terminal), and return it as a set of words.
 * @param prompt A prompt to print to screen.
 * @return A set of Strings, where each String is one of the words typed by the user
 */
public HashSet<String> getInput(String prompt) {
    ...
}
Public vs private

- Public attributes (fields, constructors, methods) are accessible to other classes.
- Fields should not be public.
- Private attributes are accessible only within the same class.
- Only methods that are intended for other classes should be public.

Information hiding

- Data belonging to one object is hidden from other objects.
- Know what an object can do, not how it does it.
- Information hiding increases the level of independence.
- Independence of modules is important for large systems and maintenance.
Class variables

```
BouncingBall

gravity 3
```

- `private`: access modifier, as usual
- `static`: class variable
- `final`: constant

```java
private static final int gravity = 3;
```
Review

• Java has an extensive class library.
• A good programmer must be familiar with the library.
• The documentation tells us what we need to know to use a class (interface).
• The implementation is hidden (information hiding).
• We document our classes so that the interface can be read on its own (class comment, method comments).

Acknowledgement

The original authors of these slides are the authors of the textbook. The instructor made necessary modifications, with permissions from the authors.