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.
• Library classes are often inter-related.
• Arranged into packages.
Working with the library

• A competent Java programmer must be able to work with the libraries.
• 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, interactive dialog system
- Idea based on ‘Eliza’ by Joseph Weizenbaum (MIT, 1960s)
- Explore tech-support-complete ...
Main loop structure

```java
boolean finished = false;

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

A common iteration pattern.
Main loop body

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
### Field Summary

<table>
<thead>
<tr>
<th>Modifier and Type</th>
<th>Field and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>static Comparator&lt;String&gt;</td>
<td>A Comparator that orders string objects as by compareIgnoreCase.</td>
</tr>
</tbody>
</table>

### Constructor Summary

- **String()**
  - Initializes a newly created `String` object so that it represents an empty character sequence.

- **String(byte[] bytes)**
  - Constructs a new `String` by decoding the specified array of bytes using the platform's default charset.

- **String(byte[] bytes, Charset charset)**
  - Constructs a new `String` by decoding the specified array of bytes using the specified charset.

- **String(byte[] ascii, int hibyte)**
  - **Deprecated.**
  - This method does not properly convert bytes into characters. As of JDK 1.1, the preferred way to do this is via the `String` constructors that take a `Charset`, charset name, or that use the platform's default charset.

- **String(byte[] bytes, int offset, int length)**
  - Constructs a new `String` by decoding the specified subarray of bytes using the platform's default charset.

- **String(byte[] bytes, int offset, int length, Charset charset)**
  - Constructs a new `String` by decoding the specified subarray of bytes using the specified charset.

- **String(byte[] ascii, int hibyte, int offset, int count)**
  - **Deprecated.**
  - This method is deprecated as of JDK 1.1.
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) of methods

the *implementation* of the class
Documentation for startsWith

- **startsWith**
  - public boolean startsWith(String prefix)
- Tests if this string starts with the specified prefix.
- Parameters:
  - prefix - the prefix.
- Returns:
  - true if the ...; false otherwise
Methods from String

- contains
- endsWith
- indexOf
- substring
- toUpperCase
- trim
- Beware: strings are immutable!
Using library classes

- Classes organized into packages.
- Classes from the library must be imported using an `import` statement (except classes from the `java.lang` package).
- They can then be used like classes from the current project.
Packages and import

• Single classes may be imported:
  import java.util.ArrayList;

• Whole packages can be imported:
  import java.util.*;

• Importation does not involve source code insertion.
Using Random

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

```java
import java.util.Random;
...
Random rand = new Random();
...
int num = rand.nextInt();
int value = 1 + rand.nextInt(100);
int index = rand.nextInt(list.size());
```
Selecting random responses

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

public void fillResponses()
{
    fill responses with a selection of response strings
}

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

• The documentation includes provision for a type parameter:
  - `ArrayList<E>`

• These type names reappear in the parameters and return types:
  - `E get(int index)`
  - `boolean add(E e)`
Parameterized classes

• The types in the documentation are placeholders for the types we use in practice:
  – An `ArrayList<TicketMachine>` actually has methods:
    – `TicketMachine get(int index)`
    – `boolean add(TicketMachine e)`
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 (its interface).
• Some classes are parameterized with additional types.
  • Parameterized classes are also known as *generic classes* or *generic types*. 
More sophisticated behavior

Using library classes to implement some more advanced functionality
Main concepts to be covered

- Further library classes
  - Set
  - Map
- Writing documentation
  - javadoc
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
}
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.
- Example: a telephone book.
Using maps

- A map with strings as keys and values

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Charles Nguyen&quot;</td>
<td>&quot;(531) 9392 4587&quot;</td>
</tr>
<tr>
<td>&quot;Lisa Jones&quot;</td>
<td>&quot;(402) 4536 4674&quot;</td>
</tr>
<tr>
<td>&quot;William H. Smith&quot;</td>
<td>&quot;(998) 5488 0123&quot;</td>
</tr>
</tbody>
</table>
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);
List, Map and Set

- Alternative ways to group objects.
- Varying implementations available:
  - ArrayList, LinkedList
  - HashSet, TreeSet
- But HashMap is unrelated to HashSet, despite similar names.
- The second word reveals organizational relatedness.
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 potential '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:

```java
/**
 * 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 (2011.07.31)
*/
```
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 elements are accessible to objects of other classes:
  • Fields, constructors and methods
• Fields should not be public.
• Private elements are accessible only to objects of 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.
Code completion

• The BlueJ editor supports lookup of methods.
• Use Ctrl-space after a method-call dot to bring up a list of available methods.
• Use Return to select a highlighted method.
Code completion in BlueJ
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).
Class and constant variables
Class variables

- A class variable is shared between all instances of the class.
- In fact, it belongs to the class and exists independent of any instances.
- Designated by the `static` keyword.
- Public static variables are accessed via the class name; e.g.:
  - `Thermometer.boilingPoint`
Class variables
Constants

• A variable, once set, can have its value fixed.
• Designated by the final keyword.
  – final int max = list.size();
• Final fields must be set in their declaration or the constructor.
• Combing static and final is common.
Class constants

• **static**: class variable
• **final**: constant

private static final int gravity = 3;

• Public visibility is less of an issue with **final** fields.
• Upper-case names often used for class constants:

  public static final int BOILING_POINT = 100;