SETS, MAPS AND ITERATORS
Overview

- Further library classes
  - Set
  - Map

- Iterators for collection classes

- References: Barnes and Kölling: Chapter 5 and see the world-of-zuul game in Chapter 6

- Slides Source: Objects First with Java - A Practical Introduction using BlueJ, © David J. Barnes, Michael Kölling
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.
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:
  
  ```java
  import java.util.ArrayList;
  ```

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

- Good style recommends: import classes named explicitly (B&K app J)

- Importation does not involve source code insertion.
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
<table>
<thead>
<tr>
<th>Modifier and Type</th>
<th>Method and Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td><code>charAt(int index)</code></td>
</tr>
<tr>
<td></td>
<td>Returns the char value at the specified index.</td>
</tr>
<tr>
<td>int</td>
<td><code>codePointAt(int index)</code></td>
</tr>
<tr>
<td></td>
<td>Returns the character (Unicode code point) at the specified index.</td>
</tr>
<tr>
<td>int</td>
<td><code>codePointBefore(int index)</code></td>
</tr>
<tr>
<td></td>
<td>Returns the character (Unicode code point) before the specified index.</td>
</tr>
<tr>
<td>int</td>
<td><code>codePointCount(int beginIndex, int endIndex)</code></td>
</tr>
<tr>
<td></td>
<td>Returns the number of Unicode code points in the specified textrange of this String.</td>
</tr>
<tr>
<td>int</td>
<td><code>compareTo(String anotherString)</code></td>
</tr>
<tr>
<td></td>
<td>Compares two strings lexicographically.</td>
</tr>
<tr>
<td>int</td>
<td><code>compareToIgnoreCase(String str)</code></td>
</tr>
<tr>
<td></td>
<td>Compares two strings lexicographically, ignoring case differences.</td>
</tr>
<tr>
<td>String</td>
<td><code>concat(String str)</code></td>
</tr>
<tr>
<td></td>
<td>Concatenates the specified string to the end of this string.</td>
</tr>
<tr>
<td>boolean</td>
<td><code>contains(CharSequence s)</code></td>
</tr>
<tr>
<td></td>
<td>Returns true if and only if this string contains the specified sequence of char values.</td>
</tr>
<tr>
<td>boolean</td>
<td><code>contentEquals(CharSequence cs)</code></td>
</tr>
<tr>
<td></td>
<td>Compares this string to the specified CharSequence.</td>
</tr>
<tr>
<td>boolean</td>
<td><code>contentEquals(StringBuffer sb)</code></td>
</tr>
<tr>
<td></td>
<td>Compares this string to the specified StringBuffer.</td>
</tr>
<tr>
<td>static String</td>
<td><code>copyValueOf(char[] data)</code></td>
</tr>
<tr>
<td></td>
<td>Returns a String that represents the character sequence in the array specified.</td>
</tr>
<tr>
<td>static String</td>
<td><code>copyValueOf(char[] data, int offset, int count)</code></td>
</tr>
<tr>
<td></td>
<td>Returns a String that represents the character sequence in the array specified.</td>
</tr>
<tr>
<td>boolean</td>
<td><code>endsWith(String suffix)</code></td>
</tr>
<tr>
<td></td>
<td>Tests if this string ends with the specified suffix.</td>
</tr>
<tr>
<td>boolean</td>
<td><code>equals(Object anObject)</code></td>
</tr>
<tr>
<td></td>
<td>Compares this string to the specified object.</td>
</tr>
<tr>
<td>boolean</td>
<td><code>equalsIgnoreCase(String anotherString)</code></td>
</tr>
<tr>
<td></td>
<td>Compares this String to another String, ignoring case considerations.</td>
</tr>
<tr>
<td>static String</td>
<td><code>format(Locale l, String format, Object... args)</code></td>
</tr>
<tr>
<td></td>
<td>Returns a formatted string using the specified locale, format string, and arguments.</td>
</tr>
<tr>
<td>static String</td>
<td><code>format(String format, Object... args)</code></td>
</tr>
<tr>
<td></td>
<td>Returns a formatted string using the specified format string and arguments.</td>
</tr>
<tr>
<td>byte[]</td>
<td><code>getBytes()</code></td>
</tr>
<tr>
<td></td>
<td>Encodes this String into a sequence of bytes using the platform's default charset, storing the result into</td>
</tr>
</tbody>
</table>
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
SETS AND MAPS
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 with code for an ArrayList!
```
Sets

- The `Set<E>` class represents mathematical sets with members of type E: sets are collections with no duplicates.
- The `HashSet` class implements the `Set` interface, backed by a hash table (that is, an efficient index).
- Set operations include:
  - Membership: `boolean contains(Object o)`
  - Size of the set: `int size()`
  - Test for empty set: `boolean isEmpty()`
  - Insert an element (if not already present): `boolean add(E e)`
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

```
:HashMap

<table>
<thead>
<tr>
<th>Name</th>
<th>Phone Number</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

```java
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.
ITERATORS
Until now …

for (Track track: tracks) { … }  
use to process every item in a collection

int index = 2;
Track track = tracks.get(index);
use to access a particular item (when you know its index)

TODAY: iterators – for more flexible looping
**Iterator and iterator()**

• Collections have an `iterator()` method.
• This returns an `Iterator` object.
• `Iterator<E>` has three methods:
  
  • `boolean hasNext()`  
  • `E next()`  
  • `void remove()`
Iterator

- An Iterator is an object that provides functionality to iterate over all elements of a collection

- import java.util.Iterator;
Using an Iterator object

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

public void listAllFiles()
{
    Iterator<Track> it = files.iterator();
    while(it.hasNext()) {
        Track tk = it.next();
        System.out.println(tk.getDetails());
    }
}
```
myList/List

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

hasNext()?

correct

correct
myList:List

:Element ➔ :Element ➔ :Element ➔ :Element

:Iterator ➔ :Iterator

hasNext()? ✔
next()
myList: List

: Element
: Element
: Element
: Element

: Iterator
: Iterator

hasNext()?
next()
myList: List

:Element → :Element → :Element → :Element

:Iterator → :Iterator

hasNext()? ✔
next()
myList:List

:Element ➔ :Element ➔ :Element ➔ :Element

:Iterator

hasNext()?
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.
    - Use to remove from a collection.

- Iteration is an important programming pattern.
Removing from a collection

```java
Iterator<Track> it = tracks.iterator();
while(it.hasNext()) {
    Track t = it.next();
    String artist = t.getArtist();
    if(artist.equals(artistToToRemove)) {
        it.remove();
    }
}
```

Use the `Iterator`'s `remove` method.
RANDOM LIBRARY
Using Random

- The library class `Random` can be used to generate a sequence of pseudorandom 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());
```
Review

• The Set class represents mathematical sets: collections with no duplicates
• The HashSet class implements the Set interface, backed by a hash table (that is, an efficient index)
• The Map<K,V> class maps keys of type K to values of type V
• The HashMap class implements the Map interface
• All collection classes provide special Iterator objects that provide sequential access to a whole collection
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).