# Software Development 2

More on collections

F27SB

*Today:*

# HOW TO <u>NOT</u> WRITE CODE YOURSELF

# Recap: Arrays

- Fixed size collection.
- Object creation (2 options)

1) `int[] numbers = { 3, 15, 4, 5 };`

2) `numbers = new int[] { 3, 15, 4, 5 };`

- Length

```
int n = numbers.length;
```

No brackets!

# Standard array use

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

DECLARATION

```
...

hourCounts = new int[24];
```

CREATION

```
...

hourCounts[i] = 0;
System.out.println(hourCounts[i]);
```

USE

## Recap: ArrayLists

- There is no pre-defined limit to the number of files.
- Java Class library: pre-defined packages.

```
import java.util.ArrayList;

/**
 * ...
 */
public class MusicOrganizer
{
    //Storage for an arbitrary number of file names.
    private ArrayList<String> files;

    /**
     * Perform any initialisation required for the
     * organizer.
     */
    public MusicOrganizer()
    {
        files = new ArrayList<String>();
    }

    ...
}
```

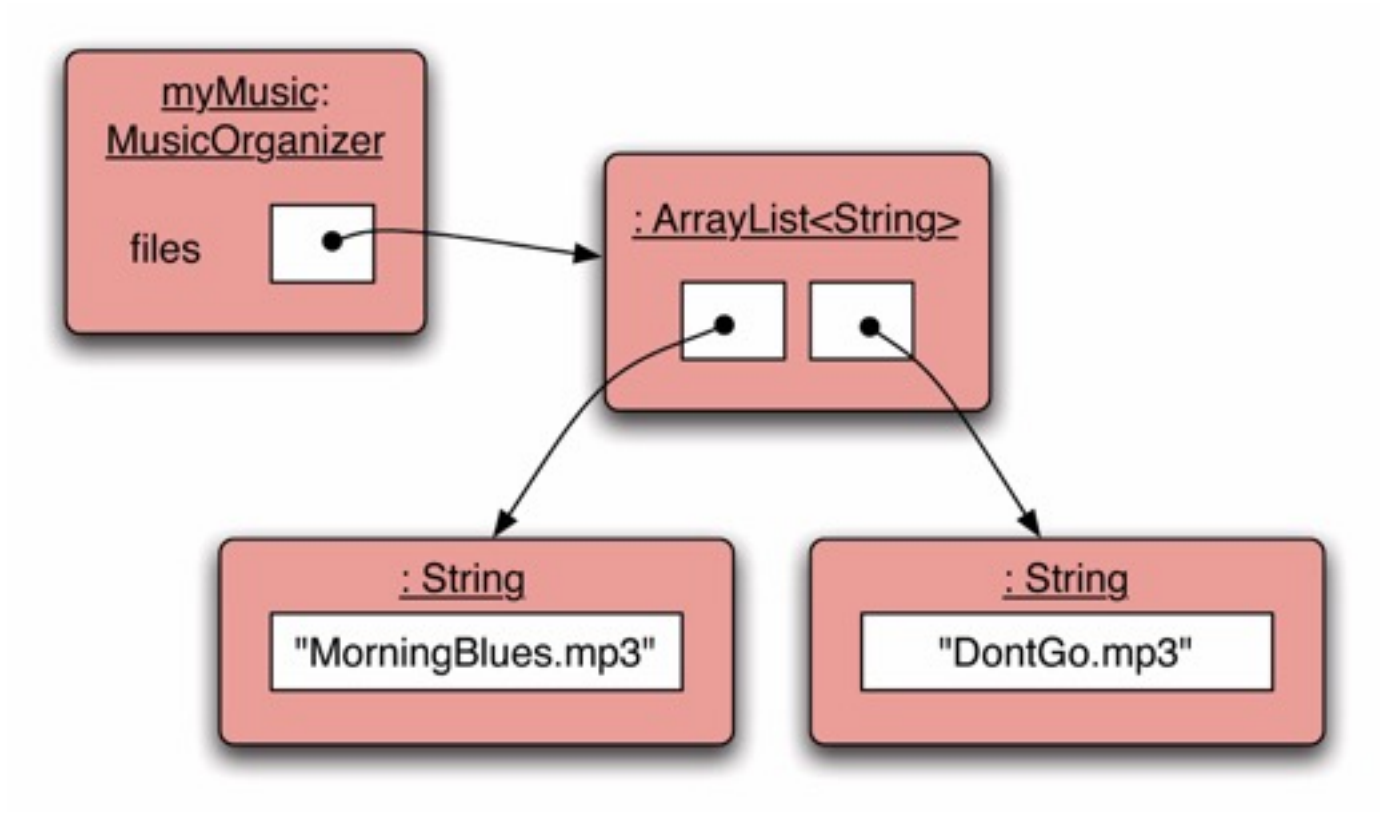
## Alternatively

```
import java.util.ArrayList;

/**
 * ...
 */
public class MusicOrganizer
{
    //Storage for an arbitrary number of file names.
    private ArrayList<String> files;

    /**
     * Perform any initialisation required for the
     * organizer.
     */
    public MusicOrganizer()
    {
        files = new ArrayList<>();
    }

    ...
}
```

Type declaration

Diamond notation

# Object structures with collections



## Using the collection

```
public class MusicOrganizer
{
    private ArrayList<String> files;

    ...

    public void addFile(String filename)
    {
        files.add(filename);
    }

    public int getNumberOfFiles()
    {
        return files.size();
    }

    ...
}
```

ADDING A NEW FILE

RETURNING THE NUMBER OF FILES (DELEGATION)

*Today's lecture*

## MORE RECAP AND PRACTICAL EXAMPLES

## Today’s lecture:

- Recap from SD1 and practical examples using library classes:
  - Random
  - HashMap and HashSet.
    - Flexible size collections.

## A Technical Support System

- A textual, interactive dialog system
- Idea based on ‘Eliza’ by Joseph Weizenbaum (MIT, 1960s)
- The very first chatbot!

## ELIZA

Men are all alike.
IN WHAT WAY
They're always bugging us about something or other.
CAN YOU THINK OF A SPECIFIC EXAMPLE
Well, my boyfriend made me come here.
YOUR BOYFRIEND MADE YOU COME HERE
He says I'm depressed much of the time.
I AM SORRY TO HEAR YOU ARE DEPRESSED
...

http://www.manifestation.com/neurotoys/eliza.php3

## Main loop body

```
Responder responder = new Responder();
InputReader reader = new InputReader();

HashSet<String> input =
    reader.getInput();
...

String response =
    responder.generateResponse();
System.out.println(response);
```

## Main loop structure

```
boolean finished = false;

while(!finished) {

    //do something

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

A common iteration pattern.

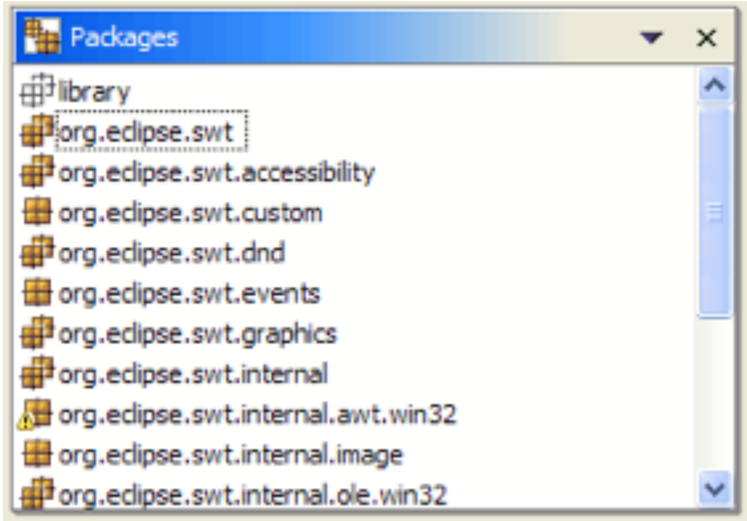
## The exit condition

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

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

- Where does ‘`contains`’ come from?
- What is it? What does it do?
- How can we find out?

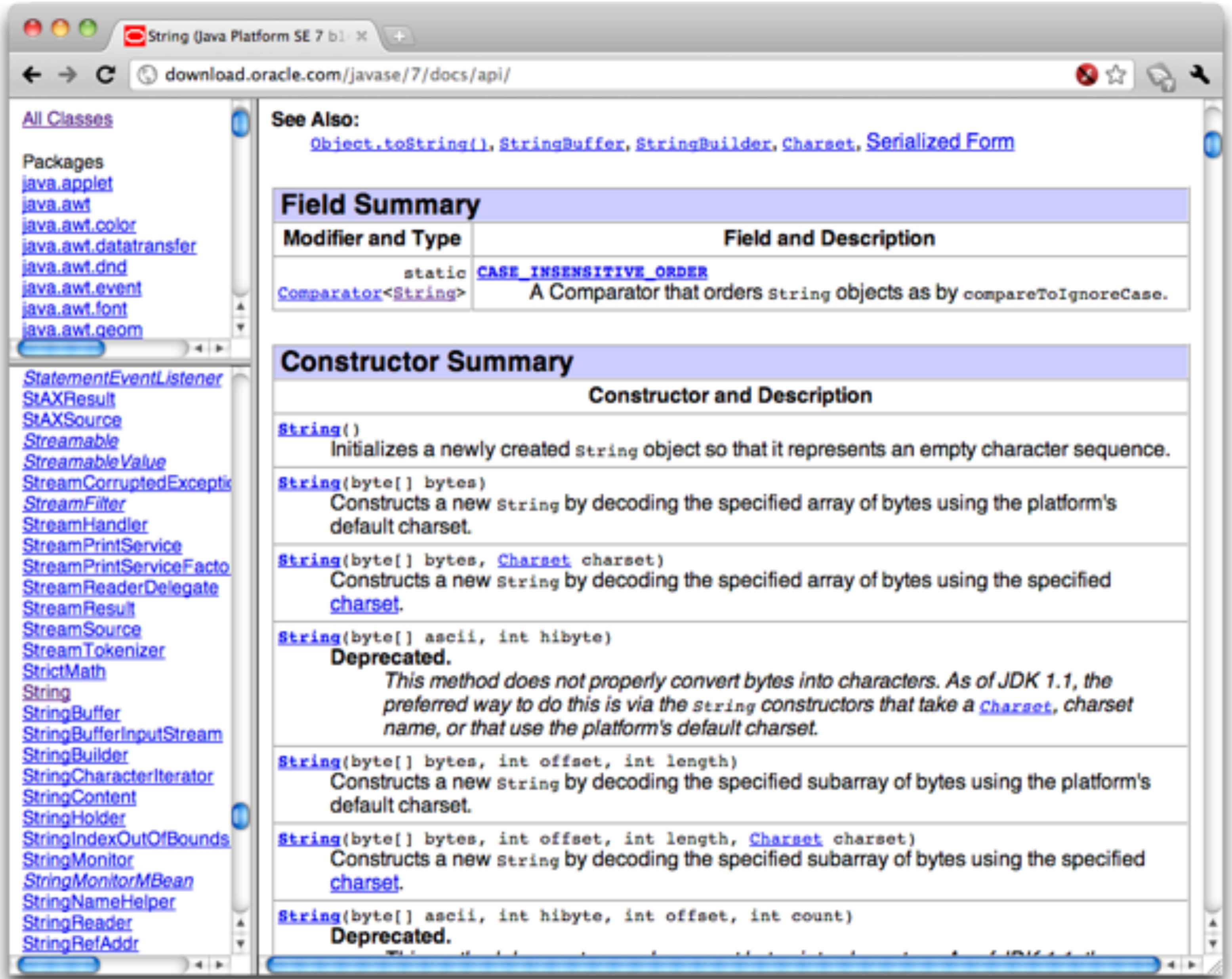
# The Java class library

- 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.


Packages

library
org.eclipse.swt
org.eclipse.swt.accessibility
org.eclipse.swt.custom
org.eclipse.swt.dnd
org.eclipse.swt.events
org.eclipse.swt.graphics
org.eclipse.swt.internal
org.eclipse.swt.internal.awt.win32
org.eclipse.swt.internal.image
org.eclipse.swt.internal.ole.win32

## Official Java API (online)



**See Also:**
`Object.toString()`, `StringBuffer`, `StringBuilder`, `Charset`, Serialized Form

**Field Summary**

| Modifier and Type | Field and Description |
|---|---|
| `static Comparator<String>` | `CASE_INSENSITIVE_ORDER`<br>A Comparator that orders `String` objects as by `compareToIgnoreCase`. |

**Constructor Summary**

| Constructor and Description |
|---|
| `String()`<br>Initializes a newly created `String` object so that it represents an empty character sequence. |
| `String(byte[] bytes)`<br>Constructs a new `String` by decoding the specified array of bytes using the platform's default charset. |
| `String(byte[] bytes, Charset charset)`<br>Constructs a new `String` by decoding the specified array of bytes using the specified charset. |
| `String(byte[] ascii, int hibyte)`<br>**Deprecated.**<br>*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)`<br>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)`<br>Constructs a new `String` by decoding the specified subarray of bytes using the specified charset. |
| `String(byte[] ascii, int hibyte, int offset, int count)`<br>**Deprecated.** |

## Using library classes

- Classes organised 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:

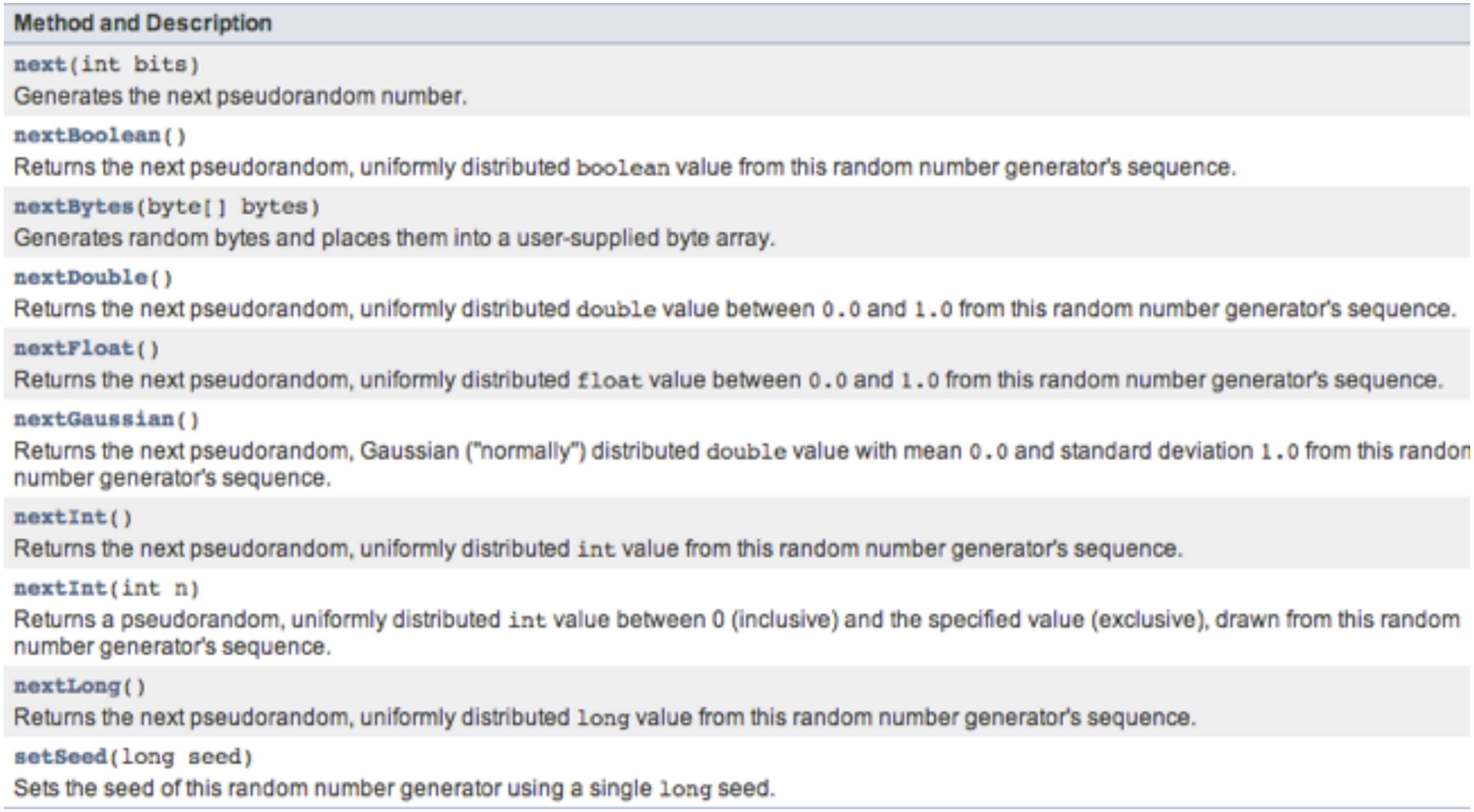
```
import java.util.*;
```

- Importation does not involve source code insertion.

## 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);
}
```

# What does it do? See Java API



| Method and Description |
|---|
| `next(int bits)`<br>Generates the next pseudorandom number. |
| `nextBoolean()`<br>Returns the next pseudorandom, uniformly distributed `boolean` value from this random number generator's sequence. |
| `nextBytes(byte[] bytes)`<br>Generates random bytes and places them into a user-supplied byte array. |
| `nextDouble()`<br>Returns the next pseudorandom, uniformly distributed `double` value between `0.0` and `1.0` from this random number generator's sequence. |
| `nextFloat()`<br>Returns the next pseudorandom, uniformly distributed `float` value between `0.0` and `1.0` from this random number generator's sequence. |
| `nextGaussian()`<br>Returns the next pseudorandom, Gaussian ("normally") distributed `double` value with mean `0.0` and standard deviation `1.0` from this randon number generator's sequence. |
| `nextInt()`<br>Returns the next pseudorandom, uniformly distributed `int` value from this random number generator's sequence. |
| `nextInt(int n)`<br>Returns a pseudorandom, uniformly distributed `int` value between 0 (inclusive) and the specified value (exclusive), drawn from this random number generator's sequence. |
| `nextLong()`<br>Returns the next pseudorandom, uniformly distributed `long` value from this random number generator's sequence. |
| `setSeed(long seed)`<br>Sets the seed of this random number generator using a single `long` seed. |

http://docs.oracle.com/javase/7/docs/api/

## Using Random

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

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

More collections

## HASHMAP AND HASHSET

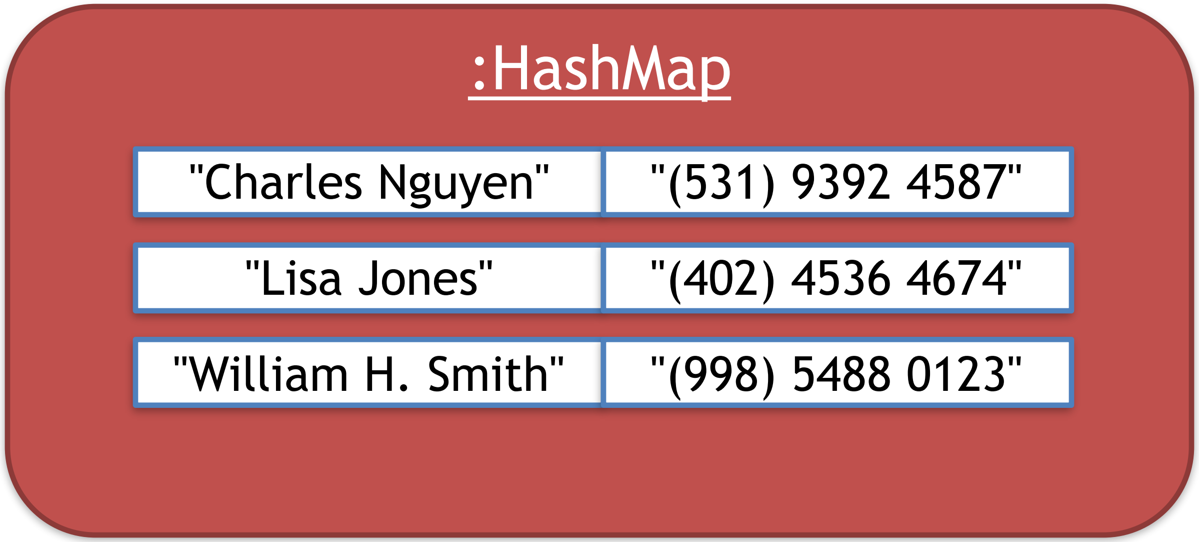
# Maps

- Maps are collections that contain pairs of values.
- Pairs consist of a <u>key</u> and a <u>value</u>.
- Lookup works by supplying a key,
  and retrieving a value.
- Example: a telephone book.

- In other languages also call it “dictionary”, e.g. Python.

# Using maps

- A map with strings as keys and values



| | |
|---|---|
| "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);
```

## Using sets

```
import java.util.HashSet;


...
HashSet<String> mySet = new HashSet<String>();

mySet.add("one");
mySet.add("two");
mySet.add(”one");

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

Compare with code for an ArrayList!

# Tokenising Strings

```
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;
}
```

# List, Map and Set

- Alternative ways to group objects.
- Varying implementations available:
  - `ArrayList, LinkedList`
  - `HashSet, TreeSet`
- Sets do not hold duplicates.
- But `HashMap` is unrelated to `HashSet`, despite similar names.
- The second word reveals organisational relatedness.

## 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).

## THAT’S IT!

## Homework

- Read chapter 5.1-5.10  (all).

- Look up List, Set, and Map in the Java API!
- http://docs.oracle.com/javase/7/docs/api/