University of HUDDERSFIELD

CFS2160: Software Design and Development

Lecture 14: Using Library Classes

Do not reinvent the wheel.

Tony Jenkins
A.Jenkins@hud.ac.uk

Java

We have now covered the "core" of Java.

We have two remaining things to do:

- Explore the (vast) library of classes available in Java.
- Explore ways to develop more sophisticated object interactions.

Remember that the "trick" in programming is to spot patterns.

Java

We have now covered the "core" of Java.

We have two remaining things to do:

- **Explore the (vast) library of classes available in Java.**
- Explore ways to develop more sophisticated object interactions.

Remember that the "trick" in programming is to spot patterns.

Java

We have now covered the "core" of Java.

We have two remaining things to do:

- Explore the (vast) library of classes available in Java.
- Explore ways to develop more sc
  interactions.

Remember that the "trick" in progra

We will also discuss *methodology*, by which we mean how we program in a structured way.

Java

We have now covered the "core" of Java.

We have two remaining things to do:

- Explore the (vast) library of classes available in Java.
- Explore ways to develop more sc
  interactions.

Remember that the "trick" in progra

And next week we will (probably) look at some ways to automate testing.

A Phone Book

Suppose we want to write a simple "Phone Book" app.

This is a "Collection" of some sort.

Each item in the collection is a pair of Strings (a name and a number).

The collection is searched by the name.

A Phone Book

Suppose we want to write a simple "Phone Book" app.

This is a "Collection" of some sort.

Each item in the collection is a pair of Strings (a name and a number).

The collection is searched by the nar

Why is a phone *number* a String? Because you don't often need to add phone numbers up.

A Phone Book

Suppose we want to write a simple "Phone Book" app.

This is a "Collection" of some sort.

Each item in the collection is a pair of Strings (a name and a number).

The collection is searched by the nar

Why is a phone *number* a String? But you might want to search them, or extract digits.

A Phone Book

Suppose we want to write a simple "Phone Book" app.

This is a "Collection" of some sort.

Each item in the collection is a pair of Strings (a name and a number).

The collection is searched by the nam

Anyway, this is another common *pattern*, so we expect there to be a Library Class.

It's a "Key-Value Pair" pattern.

A Phone Book

Suppose we want to write a simple "Phone Book" app.

This is a "Collection" of some sort.

Each item in the collection is a pair of Strings (a name and a number).

The collection is searched by the nam

Anyway, this is another common *pattern*, so we expect there to be a Library Class.

In Java, this is a "HashMap".

A Phone Book

Suppose we want to write a simple "Phone Book" app.

This is a "Collection" of some sort.

Each item in the collection is a pair of Strings (a name and a number).

The collection is searched by the nam

Anyway, this is another common *pattern*, so we expect there to be a Library Class.

In Python, this is a Dictionary.

A Map

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.

- In our example, the name is the key, and the phone number is the value.
- So we supply a name, and retrieve the corresponding number.

A Map

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.

- In our example, the name is t
  number is the value.
- So we supply a name, and re
  number.

Always be on the lookout for special cases.

Like what happens if the key is not found in the map?

Maps

Searching the Java docs for Maps we find:

https://docs.oracle.com/javase/8/docs/api/java/util/Map.html

which reveals that `Map` is an *interface*.

We also learn that there are many implementations of this interface.

Choosing an Implementation

Looking at the implementations, `HashMap` looks good:

https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html

How would we know this?

- Experience.
- Asking.
- StackOverflow!

Using a Map

As with any library class, there are now some questions:

- How is one created?
- How is one initialised?
- How is it used?

The answers are all in the docs, or can be found in all the usual reference places.

Using a Map

As with any library class, there are now some questions:

- How is one **declared**?
- What **constructors** are there?
- What **methods** are there?

The answers are all in the docs, or can be found in all the usual reference places.

Declaring

A map can be declared and initialised in one step.

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

Declaring

A map can be declared and initialised in one step.

And our code must be aware of the class, so we import it.

```
import java.util.HashMap;
```

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

Populating

The `HashMap` class has the `put` method to add entries.

```
public void fillBook ()
{
  phoneBook.put ("Len Smith", "(01484) 472209");
  phoneBook.put ("Lisa Jones", "(01484) 1234567");
  phoneBook.put ("William Smith", "(0113) 7846251");
}
```

Looking Up

And the `get` method does a lookup.

```
public void lookUpNumber (String name)
{
  String number = phoneBook.get (name);
  System.out.println(name + "'s number is " + number);
}
```

Looking Up

And the `get` method does a lookup.

```
public void lookUpNumber (String name)
{
  String number = phoneBook.get (name);
  System.out.println(name + "'s number is " + number);
}
```

What assumption has been made here?

What do we need to find out?

Better Use

What happens if the `HashMap` is empty?

Or if it doesn't contain the name we seek?

https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html

has all the answers.

Getting used to searching and reading the docs is really very, very important.

More Libraries

Suppose we had an app that needed some sort of "randomness".

We need some way of generating random numbers.

Sounds tricky.

But what should we suspect by now?

More Libraries

Suppose we had an app that needed some sort of "randomness".

We need some way of generating random numbers.

https://docs.oracle.com/javase/8/docs/api/java/util/Random.html

Back to the Phone Book

Reading the docs (or experimenting) would reveal that if a match is not found, `null` is returned.

What should this code do in this case?

```
public String lookUpNumber (String name) {
    return phoneBook.get (name);
}
```

Looking Up

A first effort might be to refactor to return an empty String if no match is found.

The code is easy.

```
public String lookUpNumber (String name) {
    if (phoneBook.get (name) == null) {
        return "";
    }
    return phoneBook.get (name);
}
```

Looking Up

A first effort might be to refactor to return an empty String if no match is found.

The code is easy.

But this would require very specific code in the program using the class to handle the error.

```
public String lookUpNumber (String name) {
    if (phoneBook.get (name) == null) {
        return "";
    }
    return phoneBook.get (name);
}
```

```
String number = c.lookUpNumber ("Donald");

if (!number.equals ("")) {
   System.out.println (number);
}
```

Looking Up

A second thought might be to make the method return a Boolean to show success or failure.

But this idea breaks because in Java the method must always return the same type.

```
public boolean lookUpNumber (String name) {
    if (phoneBook.get (name) == null) {
        return false;
    }
    return phoneBook.get (name); // true?
}
```

Looking Up

Thirdly, we could return to the docs, thinking maybe there is a method that can check if a key exists within the class.

There is!

```
boolean containsKey (Object key)
```

```
public String lookUpNumber (String name) {
    if (phoneBook.get (name) == null) {
        return false;
    }
    return phoneBook.get (name);
}
```

Looking Up

Thirdly, we could return to the docs, thinking maybe there is a method that can check if a key exists within the class.

There is!

```
boolean containsKey (Object key)
```

So the program using the HashMap could check before it calls the lookup method.

```
public String lookUpNumber (String name) {
    if (phoneBook.get (name) == null) {
        return false;
    }
    return phoneBook.get (name);
}
```

```
String name = "Donald";

if (c.containsKey (name) {
   String number = c.lookUpNumber (name);
   System.out.println (name);
}
```

Looking Up

This solution is plausible, but breaks if we design the classes properly.

We want to *encapsulate* the Phone Book in a single class, and use it from another program.

Which means we would have to add a `contains` method to the Phone Book class.

Looking Up

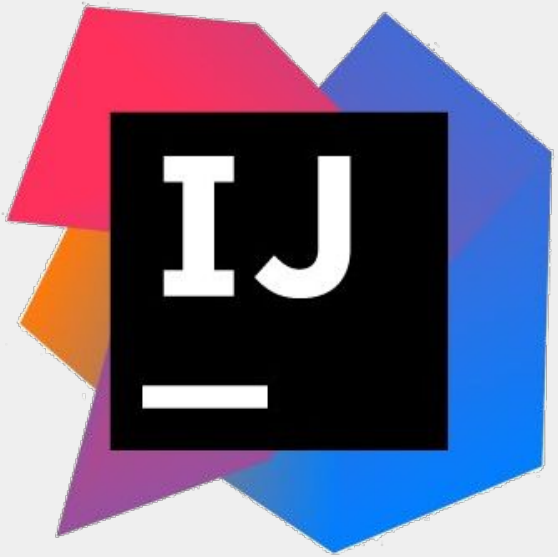
This solution is plausible, but breaks if we design the classes properly.

We want to *encapsulate* the Phone Book in a single class, and use it from another program.

Which means we would have to add Phone Book class.

This could actually be a useful piece of design.
Let's look at that setup now.

IntelliJ Demo Time



Looking Up

This solution is plausible, but breaks if we design the classes properly.

We want to *encapsulate* the Phone Book in a single class, and use it from another program.

Which means we would have to add Phone Book class.

That works.

But remember LBYL and EAFP.

Looking Up

This solution is plausible, but breaks if we design the classes properly.

We want to *encapsulate* the Phone Book in a single class, and use it from another program.

Which means we would have to add Phone Book class.

We have also arguably broken encapsulation by giving away details of how the Phone Book class (`Contacts`) is implemented.

EAFP

It is cumbersome to require the program using the Phone Book to check if the number is there first.

(It is also not how this would work in real life.)

It is surely better to assume the number is there and, if it isn't, to signal that something has gone wrong.

EAFP

It is cumbersome to require the program using the Phone Book to check if the number is there first.

(It is also not how this would work in real life.)

It is surely better to assume the num
signal that something has gone wro

So, as in Python, an Exception will be needed.

Looking Up

So we refactor the lookup method to throw (raise) an Exception if there is no match.

```
public String lookUpNumber (String name) {

  if (!phoneBook.containsKey (name)) {
    throw new NoSuchElementException ();
  }
  return phoneBook.get (name);
}
```

Looking Up

So we refactor the lookup method to throw (raise) an Exception if there is no match.

And then catch it in the program using the class.

```
public String lookUpNumber (String name) {

  if (!phoneBook.containsKey (name)) {
    throw new NoSuchElementException ();
  }
  return phoneBook.get (name);
}

try {
  System.out.println (c.lookUpNumber (name));
}
catch (NoSuchElementException e) {
  System.out.println ("Not found. Sorry.");
}
```

Looking Up

So we refactor the lookup method to throw (raise) an Exception if there is no match.

And then catch it in the program using the class

```
public String lookUpNumber (String name) {

  if (!phoneBook.containsKey (name)) {
    throw new NoSuchElementException ();
  }
  return phoneBook.get (name);
}

try {
  System.out.println (c.lookUpNumber (name));
}
catch (NoSuchElementException e) {
  System.out.println ("Not found. Sorry.");
}
```

Looking good, but the name of the Exception could be clearer.

Hierarchies

Java classes form a hierarchy.

`Object` is at the top, and all other classes *inherit* from `Object`.

(You can see this at the top of any page in the docs.)

So, we can define our own Exception, by *inheriting* all the properties of existing Exceptions.

Hierarchies

Java classes form a hierarchy.

`Object` is at the top, and all other classes *inherit* from `Object`.

(You can see this at the top of any page in the docs.)

So, we can define our own Exceptior
properties of existing Exceptions.

That's for later.
Today we'll just look at the code.

Assessment

There is no log book this term.

This means that I can show you *my* solutions to the practicals.

But remember that there are many ways to write a program, so just copying code from mine will probably not work …

IntelliJ Demo Time