Assignment Two

Brae & Emily & Tom

April 22, 2018

CSSE2002: Programming in the Large

Getting Started

1. Download `supplied.zip` and `doc.zip` from blackboard
2. Load the extracted contents of `supplied.zip` into IntelliJ - refer to week two slides

Getting Started

1. Download `supplied.zip` and `doc.zip` from blackboard
2. Load the extracted contents of `supplied.zip` into IntelliJ - refer to week two slides

The supplied code **will not compile** when initially loaded. Some code needs to be written (by you) before it will compile. This is stated in the task sheet.

Where To Start?

Start with classes that don't have dependencies (e.g. Pair)

MapIO is hard (IO has not yet been covered)

Start with classes top of the hierarchy (e.g. Thing)

The order specified in the task sheet is a good starting point

Where To Start?

Start with classes that don't have dependencies (e.g. Pair)

MapIO is hard (IO has not yet been covered)

Start with classes top of the hierarchy (e.g. Thing)

The order specified in the task sheet is a good starting point

Where To Start?

Start with classes that don't have dependencies (e.g. Pair)

MapIO is hard (IO has not yet been covered)

Start with classes top of the hierarchy (e.g. Thing)

The order specified in the task sheet is a good starting point

Where To Start?

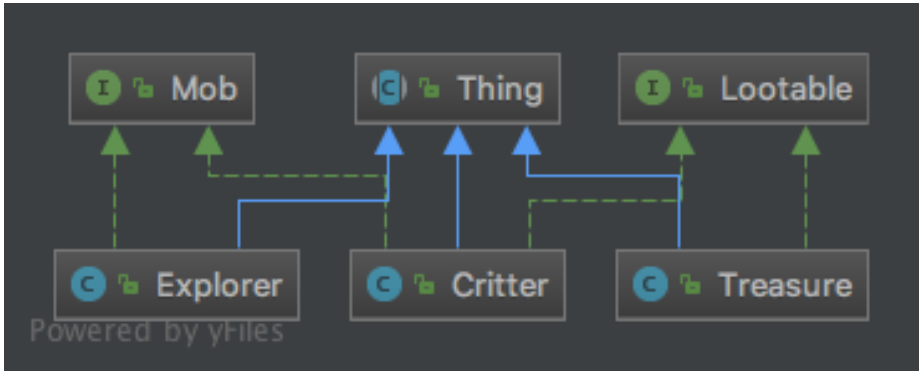
Start with classes that don't have dependencies (e.g. Pair)

MapIO is hard (IO has not yet been covered)

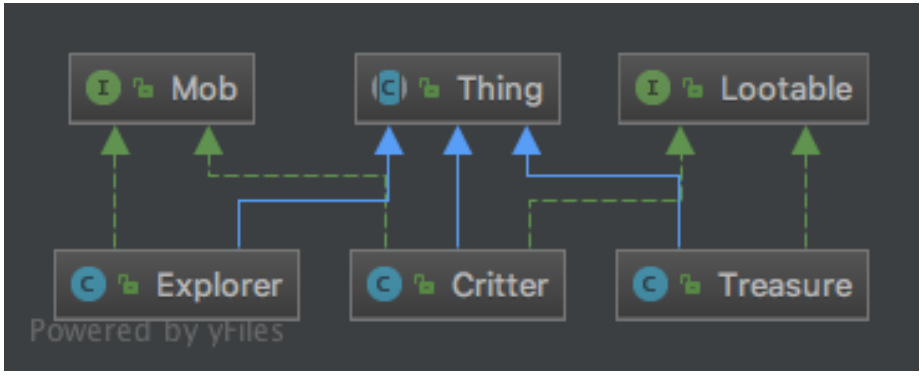
Start with classes top of the hierarchy (e.g. Thing)

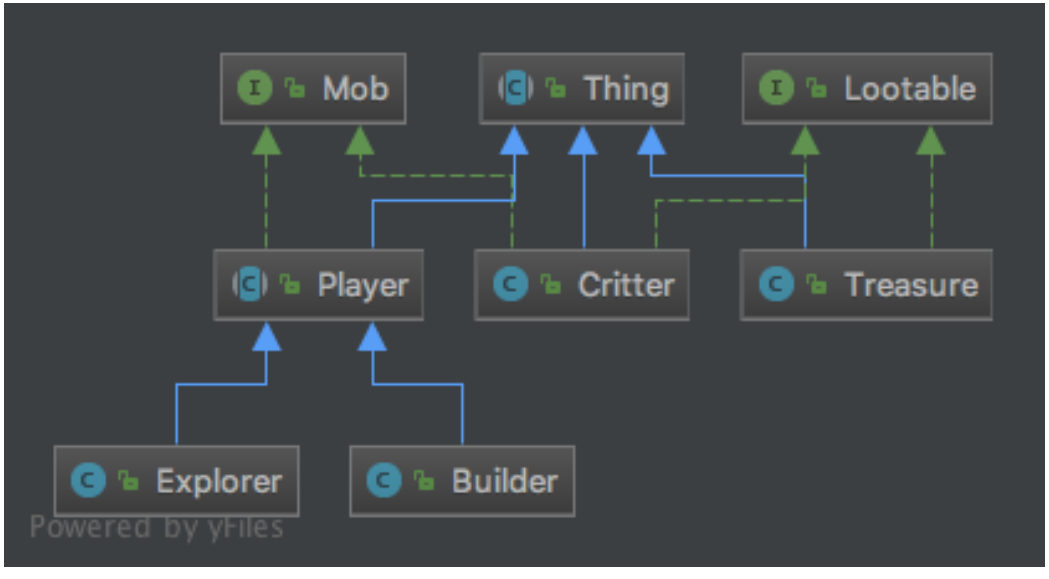
The order specified in the task sheet is a good starting point

What Has Changed?



What Has Changed?





Joel's Solution

In Joel's solution he has used `Collections.unmodifiableList` and `Collections.unmodifiableMap` when returning a instance variables. Do I need to know this?

Joel's Solution

In Joel's solution he has used `Collections.unmodifiableList` and `Collections.unmodifiableMap` when returning a instance variables. Do I need to know this?

No. You were not expected to know these methods.

Joel's Solution

In Joel's solution he has used `Collections.unmodifiableList` and `Collections.unmodifiableMap` when returning a instance variables. Do I need to know this?

No. You were not expected to know these methods.

```
return Collections.unmodifiableList(this.contents);

return new ArrayList<Thing>(this.contents);

List<Thing> results = new ArrayList<Thing>();
for (Thing thing : this.contents) {
        results.add(thing);
}
return results;
```

Exceptions

Exceptions do not need to be complicated.

```
public class CrawlException extends Exception {}

public class NullRoomException extends CrawlException
    {}
```

There is no need to add a constructor or any extra methods to exceptions.

Common Mistakes - Abstract Thing

The JavaDocs specified that the `Thing` class was an abstract class.

```
public class Thing {
        ...
}

public abstract class Thing {
        ...
}
```

Common Mistakes - Code Duplication

Code duplication for replacing strings.

Common Mistakes - Code Duplication

Code duplication for replacing strings.

Don't do this

```
public Thing(String shortDescription, String
    longDescription) {
    this.shortDescription = shortDescription.replace('
        \n', '*').replace(';', '*').replace('\r', '*')
        ;
    this.longDescription = longDescription.replace('\n
        ', '*').replace(';', '*').replace('\r', '*');
}

protected void setShort(String shortDescription) {
    this.shortDescription = shortDescription.replace('
        \n', '*').replace(';', '*').replace('\r', '*')
        ;
}
```

Common Mistakes - Code Duplication

Code duplication for replacing strings.

Abstract away repeated code

```
public Thing(String shortDescription, String
    longDescription) {
    this.shortDescription = replaceDescription(
        shortDescription);
    this.longDescription = replaceDescription(
        shortDescription);
}

private String replaceDescrition(String description) {
    return description.replace('\n', '*')
            .replace(';', '*')
            .replace('\r', '*');
}
```

Common Mistakes - String Comparison

Strings need to be compared using the .equals method not the comparison operator.

```
String first = "hello";
String second = "hello";

if (first == second) {} // wrong
if (first.equals(second)) {} // right
```

Common Mistakes - Style

Horizontal space is required on both sides of any binary or ternary operator.

Separate any reserved word, such as if, for or catch, from an open parenthesis (() that follows it on that line.

```
if (x) {} // right
if(x){} //wrong
if (x){} //wrong
if(x) {} //wrong
```

Common Mistakes - Style

Name Case

Variable names should be in camelCase

Class names should be in PascalCase

Constants should be in `SCREAMING_SNAKE_CASE`

Method names should be in camelCase

Common Mistakes - Style

Name Case

Variable names should be in camelCase

Class names should be in PascalCase

Constants should be in `SCREAMING_SNAKE_CASE`

Method names should be in camelCase

```
public class MyClassName {
        private static final int MAX_SCORE = 1000;
        private int bestScore = 0;

        public int getBestScore() {
                int myScore = 1;
                return myScore;
        }
}
```

Common Mistakes - Style

More on style in your tutorials...