P2: Advanced Java & Exam Preparation

Claudio Corrodi

May 18 2018
public interface Addressable {
    public String getStreet();
    public String getCity();

    public String getFullAddress();
}
public interface Addressable {
  public String getStreet();
  public String getCity();

  public default String getFullAddress() {
    return getStreet() + "", " + getCity();
  }
}
public class Letter implements Addressable {
    private String street;
    private String city;
    public Letter(String street, String city) {
        this.street = street;
        this.city = city;
    }
    public String getCity() {
        return city;
    }
    public String getStreet() {
        return street;
    }
    public static void main(String[] args) {
        Letter l = new Letter("123 AnyStreet", "AnyCity");
        System.out.println(l.getFullAddress());
        // prints "123 AnyStreet, AnyCity"
    }
}
public class Letter implements Addressable {

    // ... 

    @Override
    public String getFullAddress() {
        return "Destination: " + getStreet() + ": " + getCity();
    }

    public static void main(String[] args) {
        Letter l = new Letter("123 AnyStreet", "AnyCity");
        System.out.println(l.getFullAddress());
        // prints "Destination: 123 AnyStreet: AnyCity"
    }
}
Java 8: Default Methods

```java
public interface Int1 {
    public default String doSomething () {
        return "Int1.doSomething";
    }
}

class MyClass implements Int1, Int2 {
}
```

Does not compile! MyClass inherits two different methods with the same name. Select which inherited method you want to use!
Java 8: Default Methods

```java
public interface Int1 {
    public default String doSomething () {
        return "Int1.doSomething";
    }
}

public interface Int2 {
    public default String doSomething () {
        return "Int2.doSomething";
    }
}

public class MyClass implements Int1, Int2 {
}
```

Does not compile! MyClass inherits two different methods with the same name.
Java 8: Default Methods

```java
public interface Int1 {
    public default String doSomething () {
        return "Int1.doSomething";
    }
}

public interface Int2 {
    public default String doSomething () {
        return "Int2.doSomething";
    }
}

public class MyClass implements Int1, Int2 {
    public String doSomething () {
        return Int1.super.doSomething();
    }
}
```

Select which inherited method you want to use!
public interface Addressable {
    public String getStreet();
    public String getCity();

    public default String getFullAddress() {
        return getStreet() + ", " + getCity();
    }

    public static void print(Addressable addressable) {
        System.out.println(addressable.getFullAddress());
    }
}
Java 8: Lambdas

```java
List<String> myList = Arrays.asList("e1", "e2", "e3");
for (String element : myList) {
    System.out.println(element);
}

// Anonymous inner class
myList.forEach(
    new Consumer<String>() {
        public void accept(String element) {
            System.out.println(element);
        }
    };

myList.forEach((String element) -> System.out.println(element));
myList.forEach(System.out::println);
```
Java 8: Lambdas

List<String> myList = Arrays.asList("e1", "e2", "e3");

for (String element : myList) {
    System.out.println(element);
}

// Anonymous inner class
myList.forEach(new Consumer<String>() {
    public void accept(String element) {
        System.out.println(element);
    }
});

// Lambda expression
myList.forEach((String element) -> System.out.println(element));

myList.forEach(System.out::println);
Java 8: Lambdas

List<String> myList = Arrays.asList("e1", "e2", "e3");
for (String element : myList) {
    System.out.println(element);
}

// Anonymous inner class
myList.forEach(new Consumer<String>() {
    public void accept(String element) {
        System.out.println(element);
    }
});

mylist.forEach((String element) -> System.out.println(element));
myList.forEach(System.out::println);
Java 8: Lambdas

```java
List<String> myList = Arrays.asList("e1", "e2", "e3");
for (String element : myList) {
    System.out.println(element);
}

// Anonymous inner class
myList.forEach(new Consumer<String>() {
    public void accept(String element) {
        System.out.println(element);
    }
});

myList.forEach((String element) -> System.out.println(element));
myList.forEach(element -> System.out.println(element));
```
Java 8: Lambdas

```java
List<String> myList = Arrays.asList("e1", "e2", "e3");
for (String element : myList) {
    System.out.println(element);
}
```

// Anonymous inner class
```java
myList.forEach(new Consumer<String>() {
    public void accept(String element) {
        System.out.println(element);
    }
});
```

```java
myList.forEach((String element) -> System.out.println(element));
```

```java
myList.forEach(System.out::println);
```
Exam sample questions

Claudio Corrodi

May 18 2018
Terminology

- Why do *god classes* and *data classes* often occur together?

- When should you call `super()` in a constructor and why?

- What is *iterative development*, and how does it differ from the *waterfall* model?

- What are the advantages of using the Model-View-Controller pattern?
Explain the observer pattern on an example use case of your choice. Include the following in your answer:

- Provide example code.
- Provide an UML diagram of the classes involved.
- State one advantage and one disadvantage of using the Observer pattern to implement a GUI. Use less than 100 words.
Design Patterns

You should be able to do this for all the patterns from the lecture and from the lab, for example, adapter, proxy, visitor, builder, null object, ... (and more!)
Fix these JavaDoc comments.

```
/*
 * The <i>Algorithm</i> defines how a value for a file is computed.
 * It must be sure that multiple calls for the same file results in the same value.
 * The implementing class should implement a useful toString() method.
 */
public interface Algorithm {
    // ...
}
```
Write a JUnit test that verifies that line ?? works as expected.

```java
import java.util.*;

public class Spreadsheet {
    private int[][] contents;
    private int rows;
    private int cols;

    /** JavaDoc omitted */
    public void setCellValue(int row, int col, int value) {
        if (row < 0 || row > this.rows - 1) {
            throw new IllegalArgumentException();
        }
        if (col < 0 || col > this.cols - 1) {
            throw new IllegalArgumentException();
        }
        this.contents[row][col] = value;
    }
}
```
What is the Law of Demeter? Does the following code satisfy the Law of Demeter? If not, where does it violate it?

```java
/**
 * Play the game with the given scripted player.
 * // more JavaDoc omitted.
 */
public void runWithScriptedPlayer(ScriptedPlayer player) {
    assert isValidGame();
    Queue<Command> commands = player.getInputQueue();
    while (!isOver() && !commands.isEmpty()) {
        execute(commands.top());
        commands.pop();
    }
    if (isOver()) {
        print("The scripted player successfully solved the level.");
    } else {
        print("The scripted player failed to solve the level.");
    }
}
```
Smalltalk

Explain what the following Smalltalk code does in 100 words.

```smalltalk
rows: rows columns: columns tabulate: aBlock
| a i |
a := Array new: rows*columns
i := 0.
1 to: rows do: [:row |
    1 to: columns do: [:column |
        a at: (i := i+1) put: (aBlock value: row value: column) ] ].
^ a
```
Exercises

- Finish exercise 8 and all other pending ones as soon as possible.

- Exercise 9 will be published after the Smalltalk lecture.

- We will write a note in your status.md file once you passed enough exercises.