PDFSAM

<u>Code Smells Identified</u>:

1) **<u>Long Method</u>:**

Refactoring type: Extract Method
Source Method: pdfsam-fx/src/main/java/org/pdfsam/ui/commons/RadioButtonDrivenTextFieldsPane.java

Original method in RadioButtonDrivenTextFieldsPane.java**:**

RadioButtonDrivenTextFieldsPane.java

```java
    public void addRow(RadioButton radio, Region field, Text helpIcon) {
        requireNotNullArg(radio, "Cannot add a null radio");
        requireNotNullArg(field, "Cannot add a null field");
        GridPane.setValignment(radio, VPos.BOTTOM);
        GridPane.setValignment(field, VPos.BOTTOM);
        GridPane.setHalignment(radio, HPos.LEFT);
        GridPane.setHalignment(field, HPos.LEFT);
        GridPane.setFillWidth(field, true);
        field.setPrefWidth(300);
        field.setDisable(true);
        radio.selectedProperty().addListener((o, oldVal, newVal) -> {
            field.setDisable(!newVal);
            if (newVal) {
                field.requestFocus();
            }
        });
        radio.setToggleGroup(group);
        add(radio, 0, rows);
        add(field, 1, rows);
        if (nonNull(helpIcon)) {
            add(helpIcon, 2, rows);
        }
        rows++;

    }
```

- The detected lines by JDeodorant in `RadioButtonDrivenTextFieldsPane` class shows the Long Method smell.
- I think not only based on the number of lines in that method, but how a bunch of `Region` and `RadioButton` <u>specific</u> operations like `setPrefixWidth`, `setDisable` and `setToggleGroup` are performed on the input arguments lead JDeodorant to detect this smell.
- This method based on its name is expected to do only adding the row, but it does a lot of pre-processing on the input arguments. It clearly makes sense to move some of these lines into separate methods.
- This is a clear example of the Long Method smell type. Therefore, I think the smell detected here is an actual smell.

Automatic Refactoring by JDeodorant in the IDE:

Method Name: setRadio

```
78•    private void setRadioButton(RadioButton radio, Region field) {
79         radio.selectedProperty().addListener((o, oldVal, newVal) -> {
80             field.setDisable(!newVal);
81             if (newVal) {
82                 field.requestFocus();
83             }
84         });
85         radio.setToggleGroup(group);
86     }
```

Method Name: setField

```
73•    private void setField(Region field) {
74         field.setPrefWidth(300);
75         field.setDisable(true);
76     }
```

Method Name: setFieldAndRadio

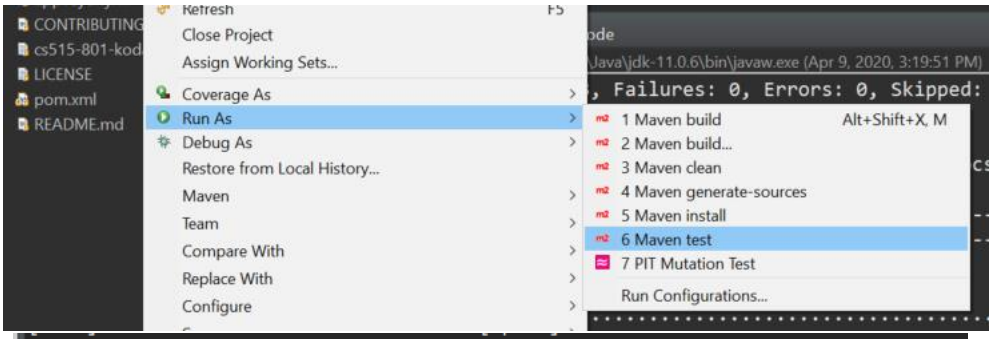
```
68•    private void setFieldAndRadio(RadioButton radio, Region field) {
69         setField(field);
70         setRadioButton(radio, field);
71     }
```

- JDeodorant sees an opportunity to shorten this method by moving the `field` and `radio` related operations into a separate method.
- The action performed by Eclipse IDE's Extract Method is basically moving per-processing logic from `addRow` class into the above new methods.
- I think with this change the methods are just doing what their name suggests. This keep the code clean, easy to understand and maintain.
- After refactoring, this smell is removed from JDeodorant Long Method smells.

Testing:

As the refactoring changes are confined to only one class and the newly added methods are all private. I did not add any new tests. But all the existing tests passed with the modified code.

Tests passed on the modified code:




```
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary for PDFsam 4.1.2-SNAPSHOT:
[INFO]
[INFO] PDFsam ............................................ SUCCESS [  0.263 s]
[INFO] PDFsam i18n ....................................... SUCCESS [ 27.056 s]
[INFO] PDFsam core ....................................... SUCCESS [ 35.819 s]
[INFO] PDFsam JavaFx views ............................... SUCCESS [05:21 min]
[INFO] PDFsam service .................................... SUCCESS [ 41.981 s]
[INFO] PDFsam merge module ............................... SUCCESS [ 24.103 s]
[INFO] PDFsam simple split module ........................ SUCCESS [ 42.978 s]
[INFO] PDFsam split by size module ....................... SUCCESS [ 24.185 s]
[INFO] PDFsam split by bookmarks module .................. SUCCESS [ 37.940 s]
[INFO] PDFsam alternate mix module ....................... SUCCESS [ 16.735 s]
[INFO] PDFsam rotate module .............................. SUCCESS [ 21.155 s]
[INFO] PDFsam extract module ............................. SUCCESS [ 26.503 s]
[INFO] PDFsam desktop client application .................. SUCCESS [02:43 min]
[INFO] PDFsam Basic Edition application ................... SUCCESS [ 11.377 s]
[INFO] PDFsam docs ....................................... SUCCESS [  0.001 s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  13:17 min
[INFO] Finished at: 2020-04-09T15:33:11-06:00
[INFO] ------------------------------------------------------------------------
```

2) God Class:

Refactoring type: Extract Class
Source Entity: pdfsam-core/src/main/java/org/pdfsam/module/ModuleDescriptorBuilder.java

- A class is tagged with God Classs smell type when it is handling or doing too much.
- In this case 'ModuleDescriptionBuilder' class seems to be holding a lot of member variables that are associated with module. And JDeoDorant clearly identified this smell.
- This class holds module related data which increases the size of the class. This is beyond the responsibility of this cases. So, I think is an actual smell.

Original class ModuleDescriptorBuilder.java:

ModuleDescriptorBuilder.java

```java
public final class ModuleDescriptorBuilder {

    private ModuleCategory category;
    private ModuleInputOutputType[] inputTypes;
    private String name;
    private String description;
    private int priority = ModulePriority.DEFAULT.getPriority();
    private String supportURL;

    private ModuleDescriptorBuilder() {
        // hide
    }

    public ModuleDescriptorBuilder category(ModuleCategory category) {
        this.category = category;
        return this;
    }

    public ModuleDescriptorBuilder inputTypes(ModuleInputOutputType... inputTypes) {
        this.inputTypes = inputTypes;
        return this;
    }

    public ModuleDescriptorBuilder name(String name) {
        this.name = name;
        return this;
    }

    public ModuleDescriptorBuilder description(String description) {
        this.description = description;
        return this;
    }

    public ModuleDescriptorBuilder priority(int priority) {
        this.priority = priority;
        return this;
    }

    public ModuleDescriptorBuilder priority(ModulePriority priority) {
        this.priority = priority.getPriority();
        return this;
    }

    public ModuleDescriptorBuilder supportURL(String supportURL) {
```

Automatic Refactoring:

New Class: Data (nested class in ModuleDescriptorBuilder)

ModuleDescriptorBuilder.java

```java
public final class ModuleDescriptorBuilder {

    public class Data {
        public ModuleCategory category;
        public ModuleInputOutputType[] inputTypes;
        public String name;
        public String description;
        public int priority;
        public String supportURL;

        public Data(int priority) {
            this.priority = priority;
        }
    }

    private Data data =  new Data(ModulePriority.DEFAULT.getPriority());

    private ModuleDescriptorBuilder() {
        // hide
    }

    public ModuleDescriptorBuilder category(ModuleCategory category) {
        this.data.category = category;
        return this;
    }

    public ModuleDescriptorBuilder inputTypes(ModuleInputOutputType... inputTypes) {
```

- Instead of `ModuleDescriptorBuilder` holding all the private member variables, the Eclipse IDE's extract class suggested to separate the data class either into a top level or a nested class.
- I extracted the data into a public nested class and the changes compiled.
- I think this refactoring made the class looks much cleaner, easier to understand and maintain.
- Although adding new variables and associated into the existing class seems easy, it is better to evaluate if adding them into a separate class makes sense. This helps in the long run with maintaining the code.
- After refactoring, this smell is removed from JDeodorant God Class smells.