1 Peer-review

1.1 Design Principles/Design Patterns

The following sections contain information about the design patterns that were implemented fully or to an extent in the program.

1.1.1 Factory Pattern

In the class **ImageServiceFactory.java** there is a implementation of Factory Pattern. This class has a static method which takes a File as a parameter to decide the file type, and returns a (new) JPGImage or PNGImage depending on the parameter. The factory is used in the PaintITController class, and effectively removes the dependency from PaintITController to PNGImage/JPGImage and replaces it with a dependency to the factory.

1.1.2 Bridge Pattern

The class **LineTool.java** implements a Bridge Pattern by composing a LineTool object with another object implementing a draw method. It is however not an implementation of the Strategy Pattern, as the name **DesignStrategy** might suggest, because the behaviour of the implemented method can't be changed at runtime.

1.1.3 Template Method Pattern

In the class **ColorTool.java**, which is abstract, there are 3 methods (onMousePressed, onMouseDragged, onMouseReleased) that acts as templates for the subclasses to override. Although, in two out of the three classes which extends **ColorTool.java** the method onMouseReleased is completely empty. This means that those two classes are forced to implement a method which they are never going to use. Furthermore, the onMousePressed method is used in the exact same way in each of the subclasses, which would suggest that this method should be moved to the superclass. Therefore, the only correct implementation of Template Method Pattern is the onMouseDragged method, which is used by every subclass but with distinctive funcionality.

1.2 Documentation

The methods are well-documented with a purpose as well as a thorough description of the parameters. All classes have a documentation, however, they do not present enought information. Authors and date of creation are missing. The documentation of which classes use the specific class and vice versa is not present.

1.3 Name Usage

Proper names are used for both methods and classes. It is easy to desipher what the methods do, and the names are short and camelcased.

It is a little bit misleading with a package that is named **View** (after the **MVC**-pattern) that only contains Controller-classes. We assume that these are controllers for all the views.

1.4 Modular Design

The classes are divided into packages depending on their responsibilities. thus it is easier to exchange or reuse these modules in other purposes. The **Memory** package has no dependencies, and **Tools** has only a dependency to **Memory**.

1.5 Abstractions

ColorTool is an abstract class that abstracts functionality from its subclasses **LineTool**, **ColorPipette** and **FillTool**. This makes it possible for the classes to create their own version of a certain method that all subclasses share.

The classes **Position**, **Pixel**, **PixelImage** and **ModelImage** use composition which is prefered over inheritance. A **ModelImage** has a **PixelImage**, a **PixelImage** a matrix of **Pixels** and a **Pixel** a **Position**. This seems logical to us.

1.6 Tests

All tests pass. the class coverage is 100%, the method coverage is 71%, and the line coverage is 77%. The coverage is adequate.

1.7 MVC Implementation

The enum **SelectedTool** is placed inside the **Controller** package, which leads to a dependency from the Model to the Controller. A quick solution is to move the enum outside the package so no dependency occurs.

There is also a circular dependency between **PaintITController** and **MainFxmlController**, which should be avoidable.

1.8 Understandability

As mentioned previously, the names of the classes and methods are descriptive and precise. The program overall is easy to understand since most of it is thoroughly commented.

It is clear what the icons in the program do, given that you have used a similar program before. A needed addition would be tool-tips for the icons so that beginners get to know what these do.