1. What are the *equivalence partitions* for the parameter `day` of this method?

```java
/**
 * Returns true if the three values represent a valid day
 */
boolean isValidDay(int year, int month, int day){

}
```

Because Feb has 29 days in leap years

[-MAX_INT..0] [1..28] [29] [30] [31] [32..MAX_INT]

Infinity, null, non-int (e.g., strings, doubles) not applicable

2. What are the *boundary values* for the parameter `day` in the question above?

-MAX_INT, 0, 1, 28, 29, 30, 31, 32, MAX_INT

3. Give 10 test inputs you would use for parameter `day` in the question above.

0, 1, 28, 29, 30, 31, 32, -1, 10, 33

Also acceptable –MAX_INT, 10, MAX_INT

Apply heuristics for combining multiple test inputs to improve the E&E of the following test cases, assuming all 6 values in the table need to be tested. underlines indicate invalid values. Point out where the heuristics are contradicted and how to improve the test cases.

SUT: `consume(food, drink)`

Test case	food	drink
TC1	bread	water
TC2	rice	lava
~~TC3~~	~~rock~~	~~acid~~

Heuristic contradicted: Each valid input should appear at least once in a positive test case

Heuristic contradicted: Test invalid inputs individually before combining them

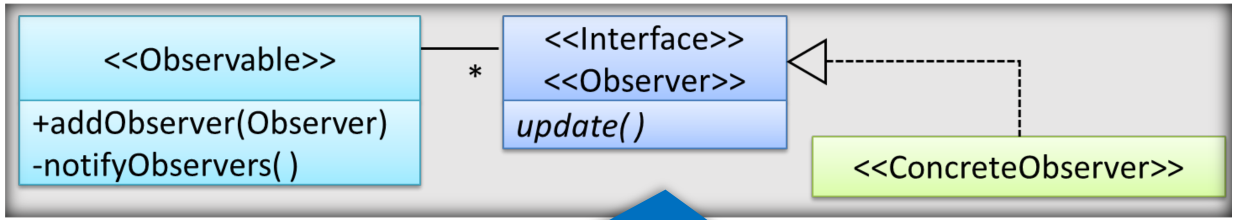
TC4	rice	water
TC5	bread /rice	acid
TC6	rock	water

Fix: Add a *positive* test case containing `rice`

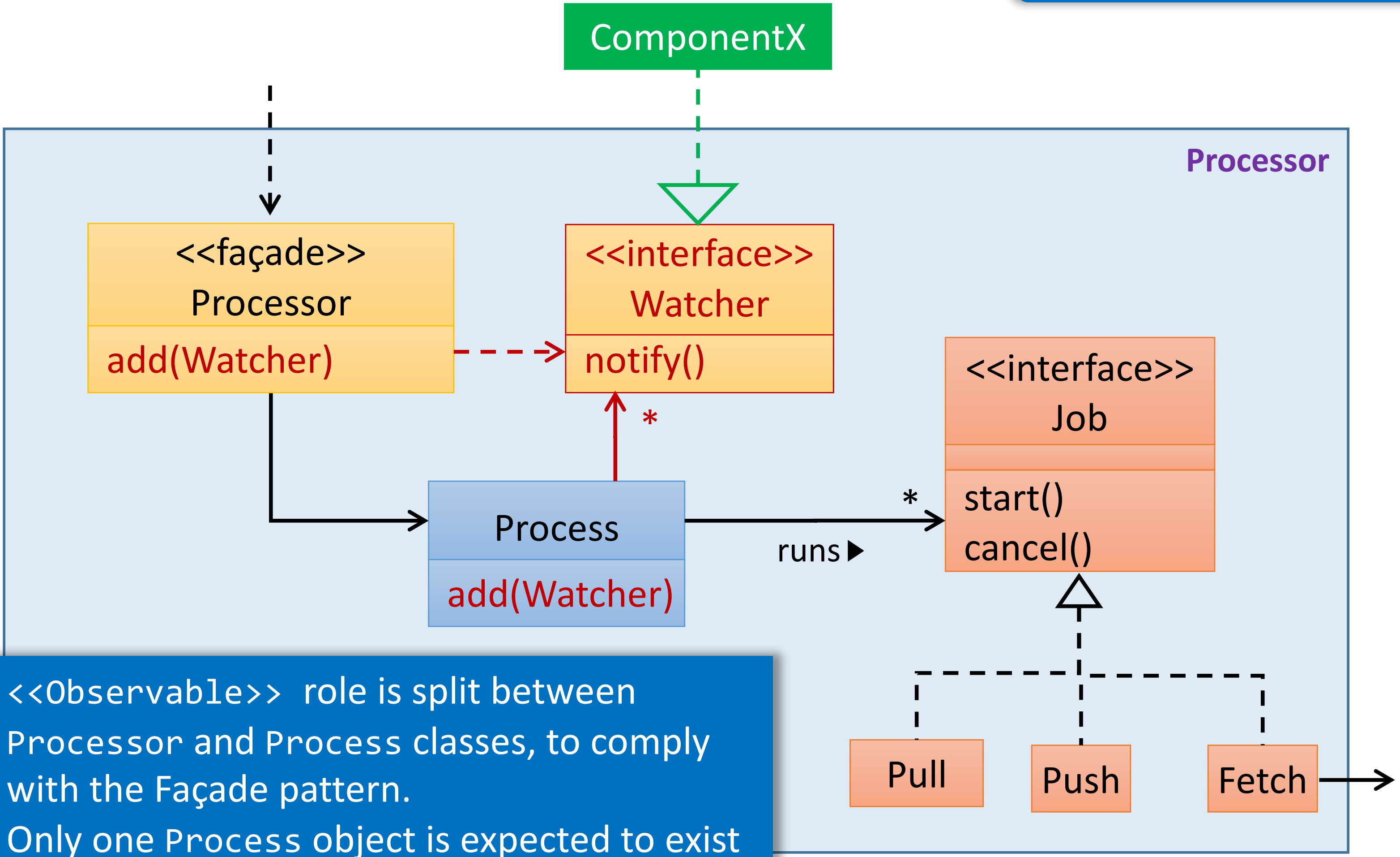
Fix: Split it into two test cases, each containing only one invalid input

If you want to provide the ability for *other* components to get notified when a `Job` is finished running, without the `Processor` component becoming dependent on those other components,

1. which design pattern would you use?
2. modify the above design accordingly.

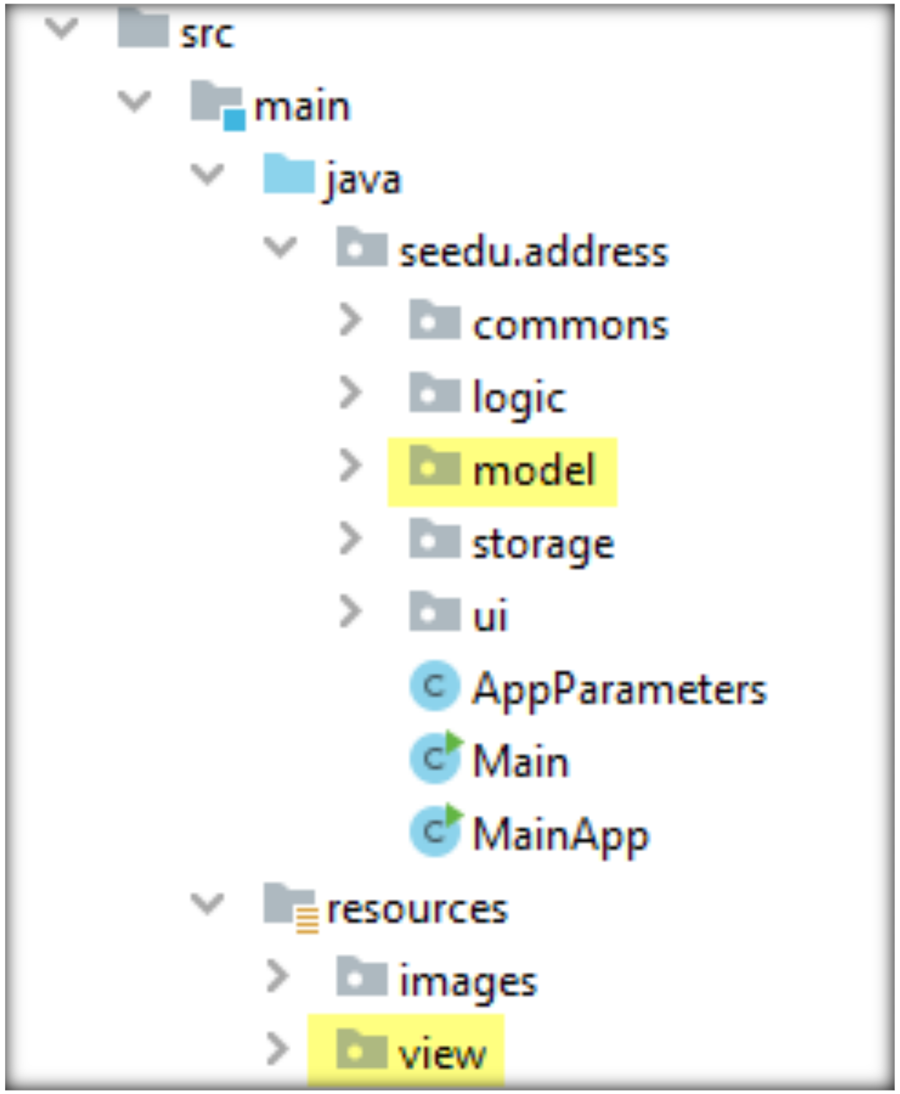


Answer: Observer pattern



- `<<Observable>>` role is split between `Processor` and `Process` classes, to comply with the Façade pattern.
- Only one `Process` object is expected to exist at any time (use singleton pattern?)

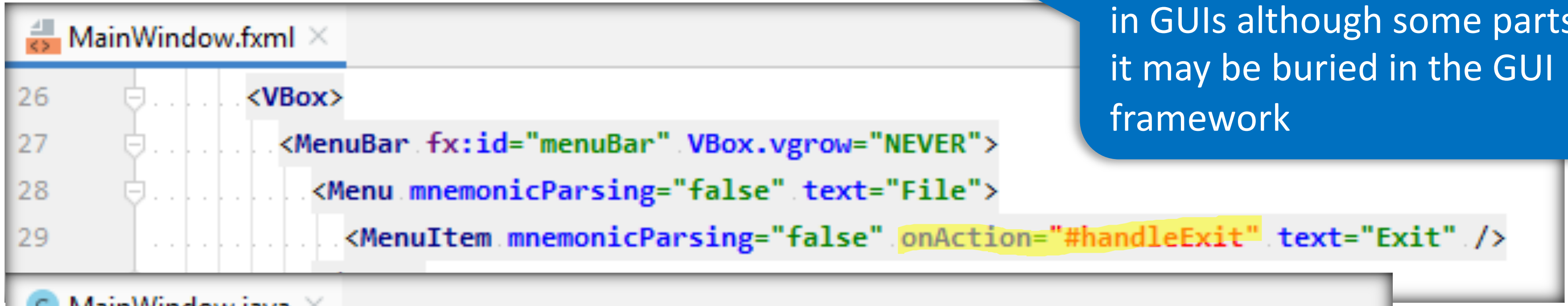
Does AB3 use the *MVC* pattern?



There is evidence that the design takes some inspiration from the MVC pattern

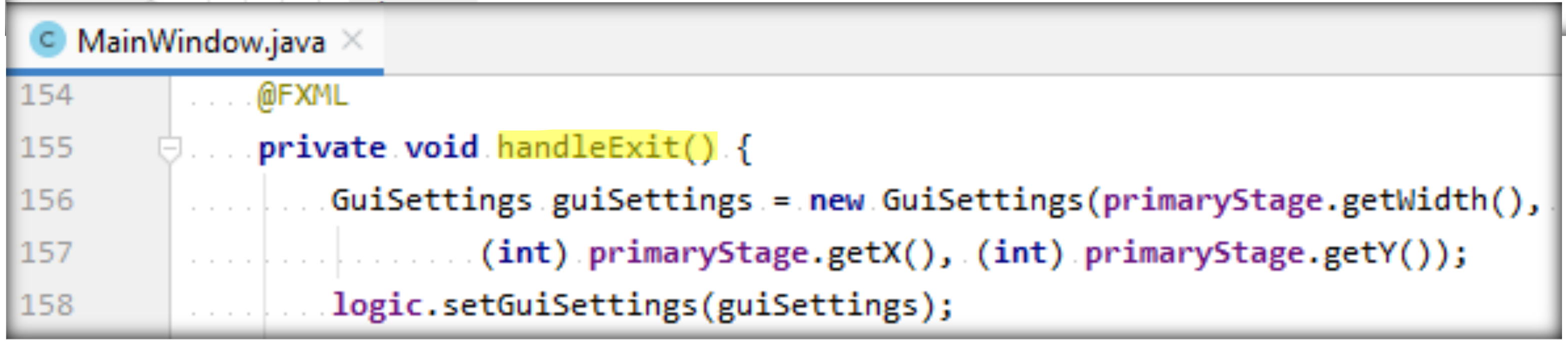
Does AB3 use the *Observer* pattern?

Observer pattern is often used in GUIs although some parts of it may be buried in the GUI framework



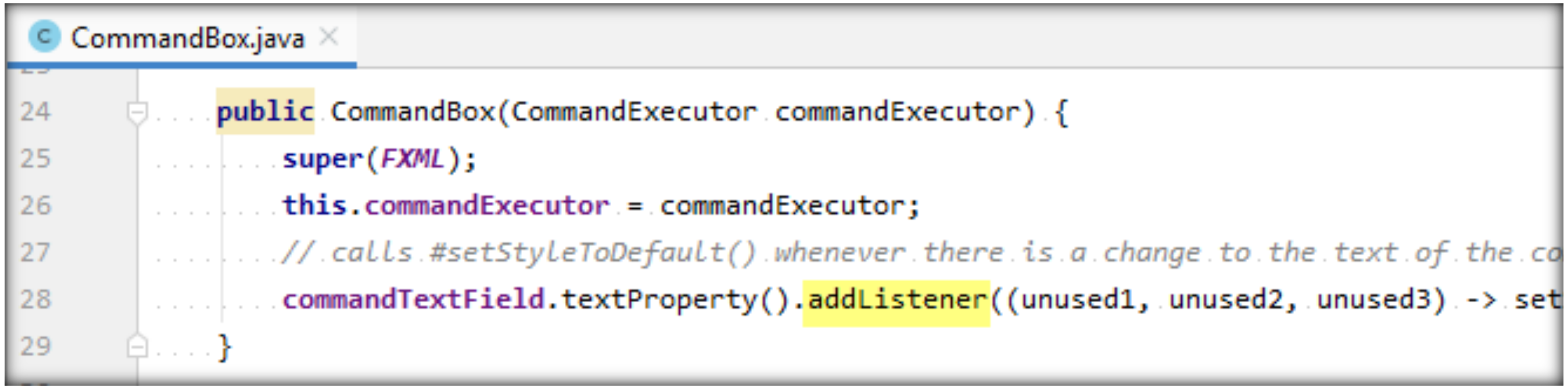
MainWindow.fxml

```
<VBox>
    <MenuBar fx:id="menuBar" VBox.vgrow="NEVER">
        <Menu mnemonicParsing="false" text="File">
            <MenuItem mnemonicParsing="false" onAction="#handleExit" text="Exit" />
```



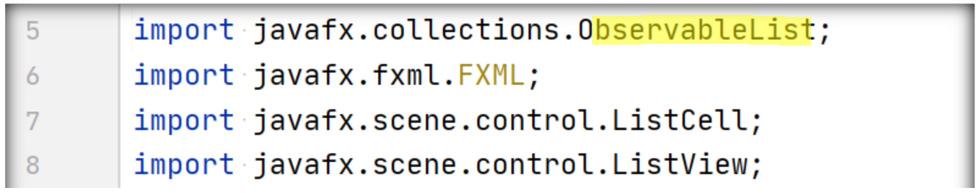
MainWindow.java

```
    @FXML
    private void handleExit() {
        GuiSettings guiSettings = new GuiSettings(primaryStage.getWidth(),
                (int) primaryStage.getX(), (int) primaryStage.getY());
        logic.setGuiSettings(guiSettings);
```



CommandBox.java

```
    public CommandBox(CommandExecutor commandExecutor) {
        super(FXML);
        this.commandExecutor = commandExecutor;
        // calls #setStyleToDefault() whenever there is a change to the text of the co
        commandTextField.textProperty().addListener((unused1, unused2, unused3) -> set
    }
```



```
import javafx.collections.ObservableList;
import javafx.fxml.FXML;
import javafx.scene.control.ListCell;
import javafx.scene.control.ListView;
```

[Bonus question]
Match each QA technique to the most matching item in the second column

$$\begin{tikzcd}
\text{a. Static Analysis} \arrow[rdd] & \text{1. Coverage} \\
\text{b. Dynamic Analysis} \arrow[ru] & \text{2. Auto-pilot software} \\
\text{c. Formal Methods} \arrow[ru] & \text{3. Checkstyle}
\end{tikzcd}$$

The use of *formal methods* is expensive but worth the cost for some safety-critical software