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

```
1  /**
2   * Returns true if the three values represent a valid day
3   */
4  boolean isValidDay(int year, int month, int day){
5
6  }
```

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?

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

OK to test -MAX_INT and MAX_INT too (but less important than the ones given)

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

[7 boundary values] -1, 10, 33

Also acceptable –MAX_INT, 10, MAX_INT

null, 1.2 : these will be rejected by the compiler

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

Heuristic violated: No more than one invalid input per test case

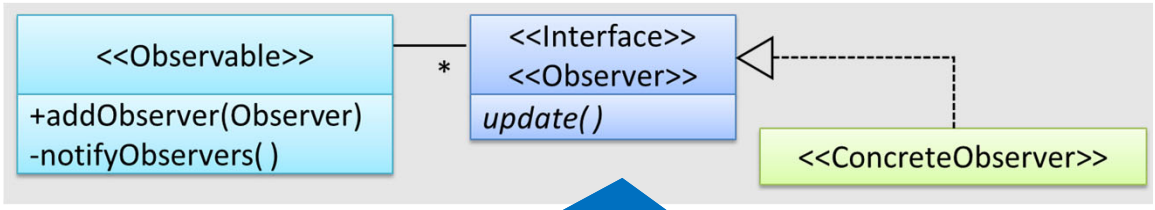
TC4	rice	water
TC5	bread /rice	acid
TC6	rock	water

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

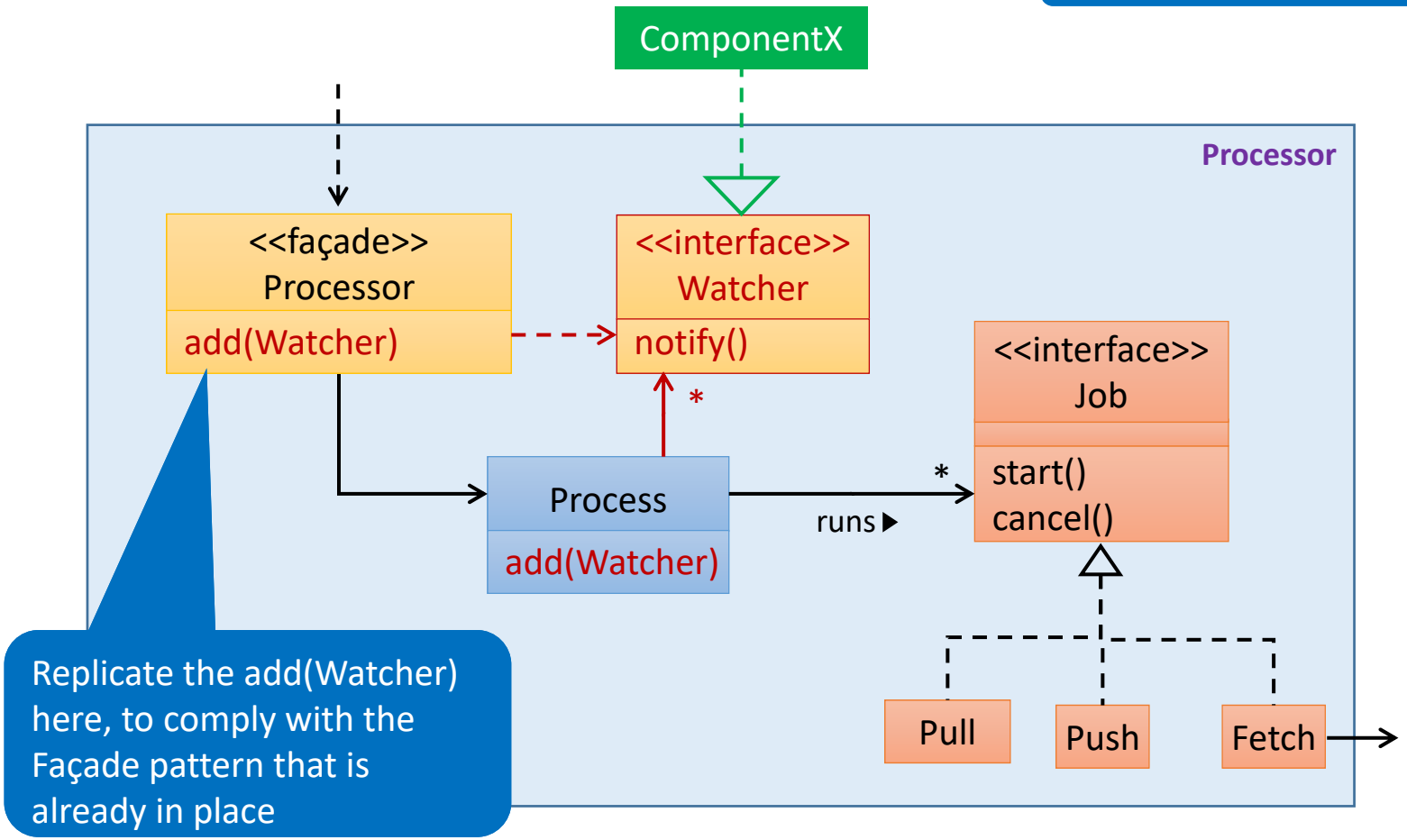
Fix: Split it into two test cases, each containing only one invalid input

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,

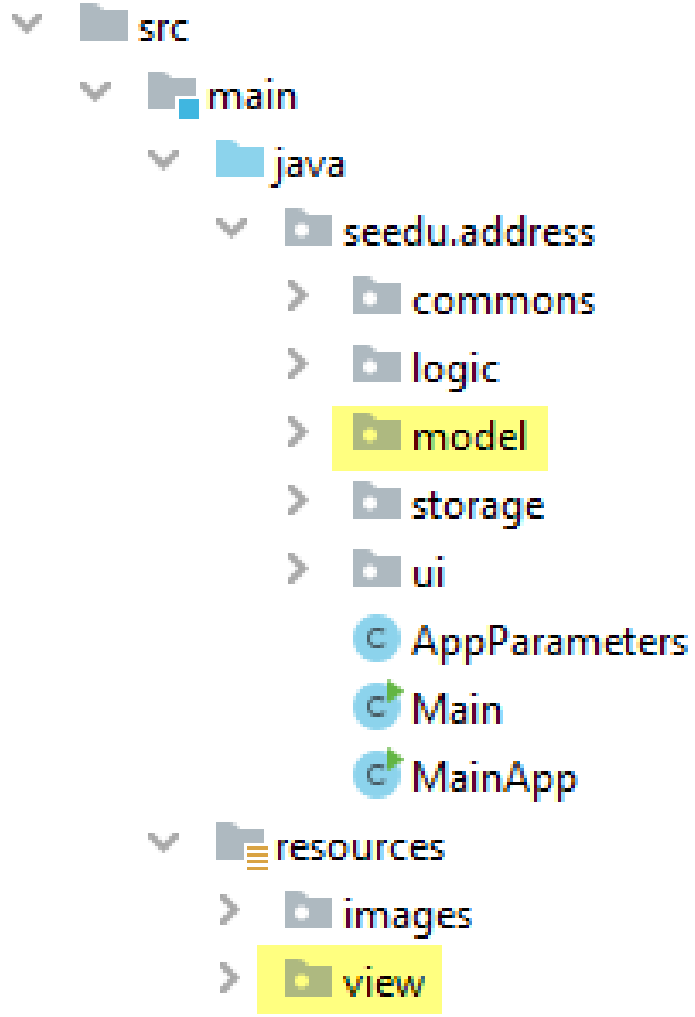
1. which design pattern would you use?
2. modify the above design accordingly.



Answer: Observer pattern



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

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

MainWindow.java

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

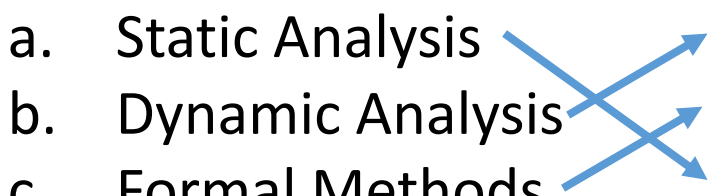
CommandBox.java

```
24      public CommandBox(CommandExecutor commandExecutor) {
25          super(FXML);
26          this.commandExecutor = commandExecutor;
27          // calls #setStyleToDefault() whenever there is a change to the text of the co
28          commandTextField.textProperty().addListener((unused1, unused2, unused3) -> set
29      }
```

```
5   import javafx.collections.ObservableList;
6   import javafx.fxml.FXML;
7   import javafx.scene.control.ListCell;
8   import javafx.scene.control.ListView;
```

[Bonus question]

Match each QA technique to the most matching item in the second column



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