**Code smells and refactoring**

1. **Code smells:**

After analyzing the given code, following four code smells have been identified which are defined below. Two of them are Class-level smells and the remaining two are Method-level smells.

1. **Class-level smells:**
   * **Large class:**

The SimpleNotePad class is unnecessary large in the given code which affects the code readability and maintainability. As ActionListener interface should be implemented to a private listener class and the object of this class should be used as a listener in **addActionListener()** method of GUI items so that the class size can be reduced to make it more readable and maintainable.

* + **Refused Bequest - Unnecessary Inheritance:**

In java, each class can inherit just one other class which makes it necessary to use inheritance wisely so that we can inherit any required class in future. As the given code is inheriting a JFrame class and all work can be done without inheriting this JFrame class so we used HAS-A relation to avoid unnecessary inheritance and created & used the object of JFrame class.

1. **Method-level smells**
   * **Inconsistent Names - Excessively short identifiers:**

As the name of a variable should reflect its function unless the function is obvious but in the given code all variables were almost consists of two letters at maximum which makes the code unreadable and unchangeable. So, I changed the variable names to an understandable, readable and meaningful name

* + **Long method:**

The given code consists of so much long methods which should be divided into some smaller methods. Like the **actionCommand(ActionEvent e)** method has grown unnecessary long which can be divided into number of different methods. As this effect the reusability of code. So, I created different methods to avoid redundancy and call them while implementing some functionality and where they are required.

1. **Refactoring:**

I refactored the code to make it more readable, understandable, scalable and maintainable by applying the following practices.

* + Line # 34: changed the variable from mb to menuBar, because it is JMenuBar type and this variable name is more readable.
  + Line # 35: changed the variable from fm to fileMenu because it is of JMenu type and this variable name is more readable.
  + Line # 36: changed the variable from em to editMenu, because because it is of JMenu type and this variable name is more readable.
  + Line # 37: changed the variable from d to textPane, because it is of JTextPane type and this variable name is more readable.
  + Line # 38: changed the variable from nf to newFileMenuItem, because it is of JMenuItem type and this variable name is more readable.
  + Line # 39: changed the variable from sf to saveFileMenuItem, because it is of JMenuItem type and this variable name is more readable.
  + Line # 40: changed the variable from pf to printFileMenuItem, because it is of JMenuItem type and this variable name is more readable.
  + Line # 41: changed the variable from pf to printFileMenuItem, because it is of JMenuItem type and this variable name is more readable.
  + Line # 42: changed the variable from c to copyMenuItem, because it is of JMenuItem type and this variable name is more readable.
  + Line # 43: changed the variable from p to pasteMenuItem, because it is of JMenuItem type and this variable name is more readable.
  + Line # 44: changed the variable from u to undoMenuitem, because it is of JMenuItem type and this variable name is more readable.
  + I created a new inner class NotepadListener and added the object of this class in addActionListener method, because it hides the details to get access from outside and reduce the class size of SimpleNotePad class and make the code more maintainable and readable.
  + I have created all methods for every functionality provided by the JMenuItems of SimpleNotepad class and call that specific method when the respective JMenuItem is clicked because this makes the code more reusable and scalable and avoid redundancy in code and we can use one method anytime and anywhere in the code when it is required.
  + I also created an object of JFrame class to avoid using inheritance to save it for a good time and make this frame visible in main method by calling setVisible method instead of in the constructor to make it easily visible and hide when we need.