

```
package gui;

import filters.ConfigFilter;
import filters.Filter;
import filters.JavaFilter;
import generator.TestInfo;

import java.awt.Component;
import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;

import javax.swing.Box;
import javax.swing.BoxLayout;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JList;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JScrollPane;
import javax.swing.ListSelectionModel;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;

import skeleton.SkeletonMaker;

public class MainFrame extends JFrame{

    private static final long serialVersionUID = 1L;

    private static MainFrame instance = null;

    private static final int TEST_MODE = 0;
    private static final int SKELETON_MODE = 1;
    private int workingMode;

    private ArrayList<String> methodsListData;
```

```
private ArrayList<String> actionsListData;
private JList methodsList, actionsList;
private JMenu instructionMenu, objectMenu, stateMenu, methodMenu;
private JMenuItem instructionDelete, instructionEdit, fileTestGenerate, menuFileOpen, menuFileNew, menuFileExportSkeleton,
fileAddImport, fileSetPackage;
private ast.File tree;
private String lastPath;

//needed only for skeleton mode
private String className, path;

private MainFrame() {
    methodsListData = new ArrayList<String>();
    actionsListData = new ArrayList<String>();
    methodsList = new JList();
    actionsList = new JList();
    lastPath = new String();
    init();
}

/**
 * Initializes all the graphic interface
 */
private void init() {
    setSize(800,600);
    setTitle("JUnit Test Generator");
    this.setBackground(new java.awt.Color(192,192,192));
    setResizable(false);

    //container for the two lists

    Box mainBox = new Box(BoxLayout.X_AXIS);

    mainBox.add(Box.createGlue());

    /*****
     *          LISTS          *
     *****/
    methodsList.setListData(methodsListData.toArray());
    methodsList.setMinimumSize(new Dimension(300,540));
```

```
methodsList.setPreferredSize(new Dimension(300,540));
methodsList.setMaximumSize(new Dimension(300,540));
methodsList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
methodsList.getSelectionModel().addListSelectionListener(new ListSelectionListener() {

    public void valueChanged(ListSelectionEvent arg0) {
        methodSelectionListener();
    }

});

JScrollPane jsp = new JScrollPane(methodsList);
jsp.setMinimumSize(new Dimension(303,543));
jsp.setPreferredSize(new Dimension(303,543));
jsp.setMaximumSize(new Dimension(303,543));
mainBox.add(jsp);

mainBox.add(Box.createGlue());

actionsList.setListData(actionsListData.toArray());
actionsList.setMinimumSize(new Dimension(300,540));
actionsList.setPreferredSize(new Dimension(300,540));
actionsList.setMaximumSize(new Dimension(300,540));
actionsList.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
actionsList.getSelectionModel().addListSelectionListener(new ListSelectionListener() {
    public void valueChanged(ListSelectionEvent arg0) {
        actionSelectionListener();
    }
});

jsp = new JScrollPane(actionsList);
jsp.setMinimumSize(new Dimension(303,543));
jsp.setPreferredSize(new Dimension(303,543));
jsp.setMaximumSize(new Dimension(303,543));
mainBox.add(jsp);

mainBox.add(Box.createGlue());

this.setContentPane(mainBox);

/*****
*           MENUS           *
*****/
```

```
        /*****
        *           FILE           *
        *****/
JMenuBar menuBar = new JMenuBar();
JMenu menuFile = new JMenu("File");
menuFileOpen = new JMenuItem("Open");
menuFileOpen.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        fileOpen();
    }
});
menuFile.add(menuFileOpen);

menuFileNew = new JMenuItem("New");
menuFileNew.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        fileNew();
    }
});
menuFile.add(menuFileNew);

fileTestGenerate = new JMenuItem("Generate Test");
fileTestGenerate.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        if (tree != null) {
            generateTest();
        }
    }
});
fileTestGenerate.setVisible(false);
menuFile.add(fileTestGenerate);

menuFileExportSkeleton = new JMenuItem("Export Skeleton");
menuFileExportSkeleton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        exportSkeleton();
    }
});
menuFileExportSkeleton.setVisible(false);
menuFile.add(menuFileExportSkeleton);

fileAddImport = new JMenuItem("Add Import");
```

```
fileAddImport.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent arg0) {  
        addImport();  
    }  
});  
fileAddImport.setVisible(false);  
menuFile.add(fileAddImport);
```

```
fileSetPackage = new JMenuItem("Set Package");  
fileSetPackage.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent arg0) {  
        setPackage();  
    }  
});  
fileSetPackage.setVisible(false);  
menuFile.add(fileSetPackage);
```

```
menuBar.add(menuFile);
```

```
    /*****  
    *          STATE          *  
    *****/
```

```
stateMenu = new JMenu("State");  
JMenuItem stateSave = new JMenuItem("Save");  
stateSave.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent arg0) {  
        saveTestInfo();  
    }  
});
```

```
JMenuItem stateLoad = new JMenuItem("Load");  
stateLoad.addActionListener(new ActionListener() {  
    public void actionPerformed(ActionEvent arg0) {  
        loadTestInfo();  
    }  
});
```

```
stateMenu.add(stateLoad);  
stateMenu.add(stateSave);
```

```
menuBar.add(stateMenu);
```

```
stateMenu.setVisible(false);

        /*****
        *   CREATE OBJECT   *
        *****/
objectMenu = new JMenu("Object");
JMenuItem objectCreate = new JMenuItem("Create");
objectCreate.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        if (tree != null)
            createObject();
    }
});
objectMenu.add(objectCreate);
objectMenu.setVisible(false);
menuBar.add(objectMenu);

        /*****
        *   INSTRUCTION     *
        *****/
instructionMenu = new JMenu("Instruction");
JMenuItem instructionCreate = new JMenuItem("Create");
instructionCreate.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        createAssert();
    }
});
instructionMenu.add(instructionCreate);
instructionEdit = new JMenuItem("Edit Selected");
instructionEdit.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        editAction();
    }
});
instructionEdit.setVisible(false);
instructionMenu.add(instructionEdit);
instructionDelete = new JMenuItem("Delete Selected");
```

```
instructionDelete.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        deleteAction();
    }
});
instructionDelete.setVisible(false);
instructionMenu.add(instructionDelete);
instructionMenu.setVisible(false);
menuBar.add(instructionMenu);

    /*****
    *      METHODS      *
    *****/
methodMenu = new JMenu("Method and Constructors");
JMenuItem methodNew = new JMenuItem("New Method");
methodNew.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        newMethod();
    }
});

JMenuItem constructorNew = new JMenuItem("New Constructor");
constructorNew.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        newConstructor();
    }
});

methodMenu.add(methodNew);
methodMenu.add(constructorNew);

methodMenu.setVisible(false);
menuBar.add(methodMenu);

this.setJMenuBar(menuBar);

    /*****
    *      END  MENUS      *
    *****/
```

```
        this.setDefaultCloseOperation(EXIT_ON_CLOSE);
        setVisible(true);

        //autoFileOpen();
    }

    /**
     * show a dialog to select a configuration file and loads its content.
     */
    private void loadTestInfo() {
        JFileChooser fc = new JFileChooser();
        fc.setFileFilter(ConfigFilter.getInstance());
        fc.setAcceptAllFileFilterUsed(false);
        fc.setCurrentDirectory(new java.io.File(lastPath));
        int returnVal = fc.showOpenDialog(this);
        if (returnVal == JFileChooser.APPROVE_OPTION) {
            java.io.File file = fc.getSelectedFile();
            lastPath = file.getParent();
            try {
                TestInfo.getInstance().loadFromFile(file.getAbsolutePath(), methodsListData);
                methodSelectionListener();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }

    /**
     * shows a dialog to select a configuration file to store all the information of the tests.
     */
    private void saveTestInfo() {
        JFileChooser fc = new JFileChooser();
        fc.setFileFilter(ConfigFilter.getInstance());
        fc.setAcceptAllFileFilterUsed(false);
        fc.setCurrentDirectory(new java.io.File(lastPath));
        int returnVal = fc.showSaveDialog(this);
        if (returnVal == JFileChooser.APPROVE_OPTION) {
            java.io.File file = fc.getSelectedFile();
            lastPath = file.getParent();
            try {
                if (Filter.compareExt(file.getAbsolutePath(), "cfg")) {

```

```
        TestInfo.getInstance().saveToFile(file.getAbsolutePath());
    }else{
        TestInfo.getInstance().saveToFile(file.getAbsolutePath()+".cfg");
    }
}catch(Exception e){
    e.printStackTrace();
}
}

private void exportSkeleton(){
    SkeletonMaker.getInstance().createSkeleton(path + className + ".java");
}

private void generateTest(){//.java
    JFileChooser fc = new JFileChooser();
    fc.setFileFilter(JavaFilter.getInstance());
    fc.setAcceptAllFileFilterUsed(false);
    fc.setCurrentDirectory(new java.io.File(lastPath));
    int returnVal = fc.showSaveDialog(this);
    if(returnVal == JFileChooser.APPROVE_OPTION) {
        java.io.File file = fc.getSelectedFile();
        lastPath = file.getParent();
        try{
            if(Filter.compareExt(file.getAbsolutePath(), "java")){
                TestInfo.getInstance().toFile(file.getAbsolutePath());
            }else{
                TestInfo.getInstance().toFile(file.getAbsolutePath() + ".java");
            }
        }catch(Exception e){
            e.printStackTrace();
        }
    }
}

private void fileOpen(){
    JFileChooser fc = new JFileChooser();
    fc.setFileFilter(JavaFilter.getInstance());
    fc.setAcceptAllFileFilterUsed(false);
    fc.setCurrentDirectory(new java.io.File(lastPath));
    int returnVal = fc.showOpenDialog(this);
    if (returnVal == JFileChooser.APPROVE_OPTION) {
        java.io.File file = fc.getSelectedFile();
    }
```

```
lastPath = file.getParent();
try {
    analyzer.Parser p = new analyzer.Parser(file.getAbsolutePath());
    tree = p.Analyze();
    if(tree != null){
        methodsListData.clear();
        ArrayList<ast.Method> constructors = tree.fileclass.getConstructors();
        for (int i = 0; i < constructors.size(); i++){
            methodsListData.add(constructors.get(i).toString());
        }
        ArrayList<ast.Method> methodslist = tree.fileclass.getMethods();
        for (int i = 0; i < methodslist.size(); i++){
            methodsListData.add(methodslist.get(i).toString());
        }
        methodsList.setListData(methodsListData.toArray());
        TestInfo.getInstance().setPackage(tree.getPackage());
        TestInfo.getInstance().setClassName(tree.fileclass.getName());
        TestInfo.getInstance().setImports(tree.getImports());
        stateMenu.setVisible(true);
        fileTestGenerate.setVisible(true);
        menuFileOpen.setVisible(false);
        menuFileNew.setVisible(false);
        workingMode = TEST_MODE;
    }else{
        msgBox("Something went wrong while Parsing the Java file.");
    }
} catch (Exception e) {
    msgBox("Something went wrong while Parsing the Java file: \r\n" + e.getMessage());
}
} else {
    // error
}
}

private void setPackage(){
    String myPackage = JOptionPane.showInputDialog(null,
        "Package",
        "Write here the name of the package:",
        JOptionPane.QUESTION_MESSAGE);
    if (myPackage != null){
```

```
SkeletonMaker.getInstance().setPackage(myPackage);
    }
}
private void addImport(){
    String myImport = JOptionPane.showInputDialog(null,
        "Import text:",
        "Adding Imports",
        JOptionPane.QUESTION_MESSAGE);
    if (myImport != null){
        SkeletonMaker.getInstance().addImport(myImport);
    }
}
private void fileNew(){
    JFileChooser fc = new JFileChooser();
    fc.setFileFilter(JavaFilter.getInstance());
    fc.setAcceptAllFileFilterUsed(false);
    fc.setCurrentDirectory(new java.io.File(lastPath));
    int returnVal = fc.showSaveDialog(this);
    if(returnVal == JFileChooser.APPROVE_OPTION) {
        java.io.File file = fc.getSelectedFile();
        lastPath = file.getParent();
        try{
            String filePath = file.getAbsolutePath();
            try{
                path = filePath.substring(0,filePath.lastIndexOf('\\'));
            }catch(java.lang.StringIndexOutOfBoundsException e){
                path = filePath.substring(0,filePath.lastIndexOf('/')); //Unix compatibility
            }

            if(Filter.compareExt(filePath, "java")){
                className = filePath.substring(path.length()+1,filePath.lastIndexOf('.'));
            }else{
                className = filePath.substring(path.length()+1,filePath.length());
            }

            setTitle("Creating: " + className);
            methodMenu.setVisible(true);
            workingMode = SKELETON_MODE;
            menuFileOpen.setVisible(false);
            menuFileNew.setVisible(false);
        }
    }
}
```

```
        fileSetPackage.setVisible(true);
        fileAddImport.setVisible(true);
        menuFileExportSkeleton.setVisible(true);
        SkeletonMaker.getInstance().setClassName(className);
    }catch(Exception e){
        e.printStackTrace();
    }
}

private void createObject(){
    if(methodsList.getSelectedIndex() >= 0 && methodsList.getSelectedIndex() < tree.fileclass.getConstructors().size()){
        this.setEnabled(false);
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                new CreateObjectUI(tree.fileclass.getConstructors().get(methodsList.getSelectedIndex()));
            }
        });
    }else{
        msgBox("You need to select the constructor that you want to use.");
    }
}

private void newMethod(){
    this.setEnabled(false);
    javax.swing.SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            new CreateMethodConstructorUI();
        }
    });
}

private void newConstructor(){
    this.setEnabled(false);
    javax.swing.SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            new CreateMethodConstructorUI(className);
        }
    });
}
```

```
private void createAssert() {
    if (TestInfo.getInstance().getObjectsString().equals(new ArrayList<String>())) {
        msgBox("You need to create at least 1 object before creating Asserts");
    } else if (methodsList.getSelectedIndex() < tree.fileclass.getConstructors().size()) {
        msgBox("You must choose a Method (not a constructor) before trying to create an Assert");
    } else {
        this.setEnabled(false);
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                new CreateInstructionUI(tree.fileclass.getMethods().get(methodsList.getSelectedIndex() - tree.fileclass.
getConstructors().size()));
            }
        });
    }
}

private void methodSelectionListener() {
    if (workingMode == TEST_MODE) {
        if (methodsList.getSelectedIndex() != -1) {
            if (methodsList.getSelectedIndex() < tree.fileclass.getConstructors().size()) {
                actionsListData = TestInfo.getInstance().getObjectsString();
                if (workingMode == TEST_MODE) {
                    instructionMenu.setVisible(false);
                    objectMenu.setVisible(true);
                }
            } else {
                actionsListData = TestInfo.getInstance().getThingsToDoString((String) methodsList.getSelectedValue());
                instructionMenu.setVisible(true);
                objectMenu.setVisible(false);
            }
        } else {
            instructionMenu.setVisible(false);
            objectMenu.setVisible(false);
        }
        actionsList.setListData(actionsListData.toArray());
    } else if (workingMode == SKELETON_MODE) {
    }
}
```

```
private void actionSelectionListener() {
    if(methodsList.getSelectedIndex() >= tree.fileclass.getConstructors().size() && actionsList.getSelectedIndex() != -1){
        instructionDelete.setVisible(true);
        instructionEdit.setVisible(true);
    }else{
        instructionDelete.setVisible(false);
        instructionEdit.setVisible(false);
    }
}

private void editAction(){
    this.setEnabled(false);
    javax.swing.SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            new CreateInstructionUI(tree.fileclass.getMethods().get(methodsList.getSelectedIndex()-tree.fileclass.
getConstructors().size()),actionsList.getSelectedIndex());
        }
    });
}

private void deleteAction(){
    TestInfo.getInstance().deleteThingsToDo(methodsList.getSelectedValue().toString(), actionsList.getSelectedIndex());
    methodSelectionListener();
}

private void msgBox(String msg){
    JOptionPane.showMessageDialog(((Component)this), msg);
}

/**
 * Unlocks the window after having closed a child window
 */
public void unlock(){
    this.setEnabled(true);
    methodSelectionListener();
    if(workingMode == SKELETON_MODE){
        methodsListData = SkeletonMaker.getInstance().getMethodsArray();
        methodsList.setListData(methodsListData.toArray());
    }
}

/**
 * Following Singleton pattern, returns the only instance of the class MainFrame
 * @return the only instance of the class MainFrame
 */
```

```
    */
    public static MainFrame getInstance() {
        if(instance == null){
            javax.swing.SwingUtilities.invokeLater(new Runnable() {
                public void run() {
                    instance = new MainFrame();

                }
            });
        }
        return instance;
    }
    /**
     * Main that launches the application
     * @param params Unused
     */
    public static void main(String[] params){
        getInstance();
    }
}
```

////////////////////////////////////