Object-Oriented Software Engineering

Instructor : Huang, Chuen-Min

Teamwork1 ver.1

Group

ID	Name
B10423015	justin
B10423045	Rong
B10423025	Sean
B10323026	Jocelyn
B10323007	Jimmy
B10323001	Ryan
B10423046	Vicky
A10523010	Irene
A10523048	Tony
A10523030	Candy

Date 2018/4/17

Content

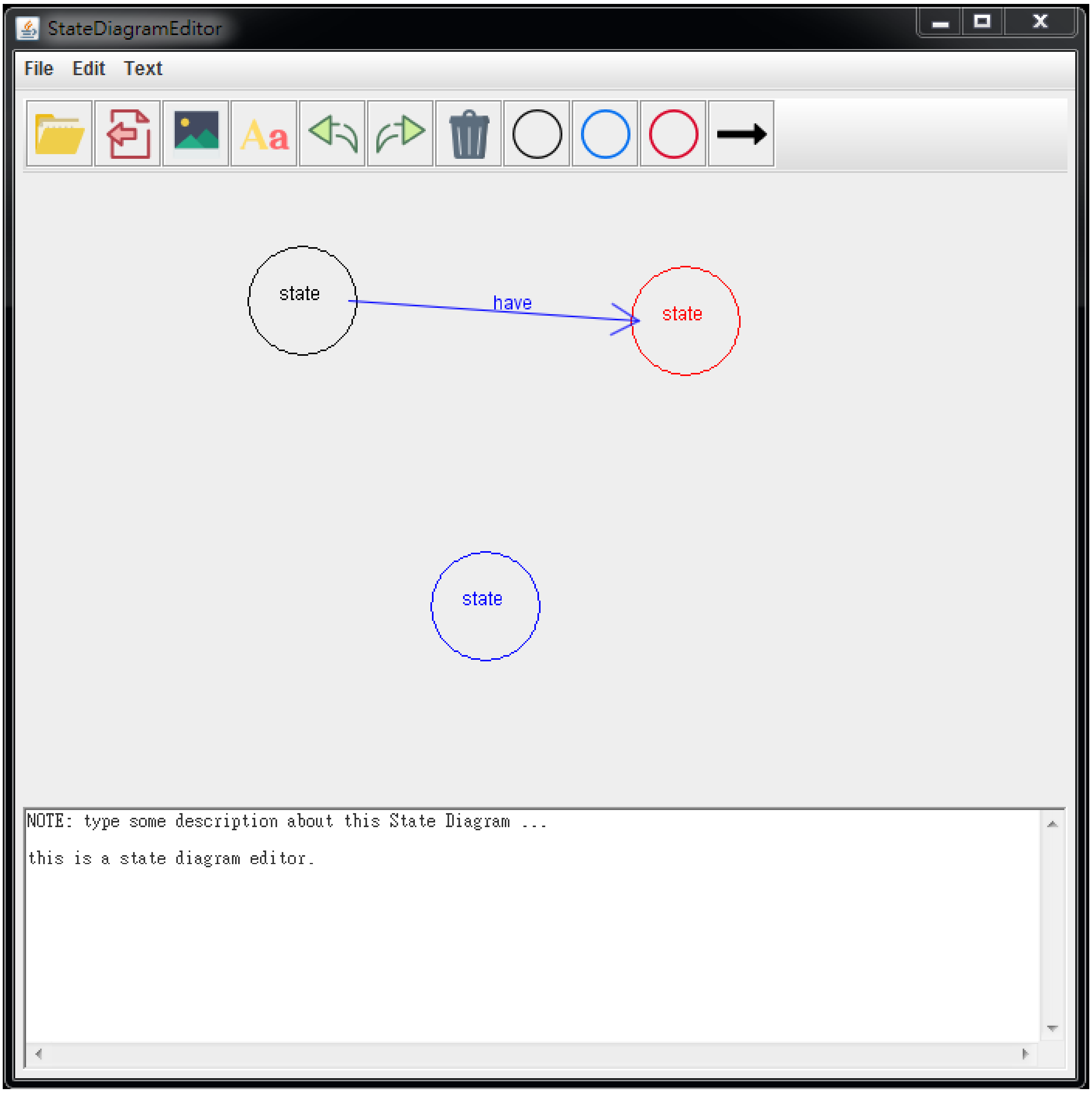
1. Assignments and participation

ID	Name	Participation	Assignments
B10423015	justin	100%	Coding
B10423045	Rong	100%	Paper
B10423025	Sean	100%	Paper Coding
B10323026	Jocelyn	100%	Leader Paper Coding
B10323007	Jimmy	100%	Paper
B10323001	Ryan	100%	Paper
B10423046	Vicky	100%	Paper
A10523010	Irene	100%	Paper GUI
A10523048	Tony	100%	Paper GUI
A10523030	Candy	100%	Paper GUI

2. Describe our application scenarios

We design a State Diagram Editor. It include some functions such as : Import, Export, Draw, Delete, Undo, Redo and Describe.

When user need to use the old file, he/she can import the file what he/she need, and if the user finished his/her file, he/she can export the file in file type or picture type.User can draw a transition or a circle in black, red or green, also, he/she can delete it.If the user want to go back to last step, he/she can use redo function, opposite, if the user want to go to next step, he/she can use redo function.If user can use describe function to describe the state diagram, it will make others know the diagram.

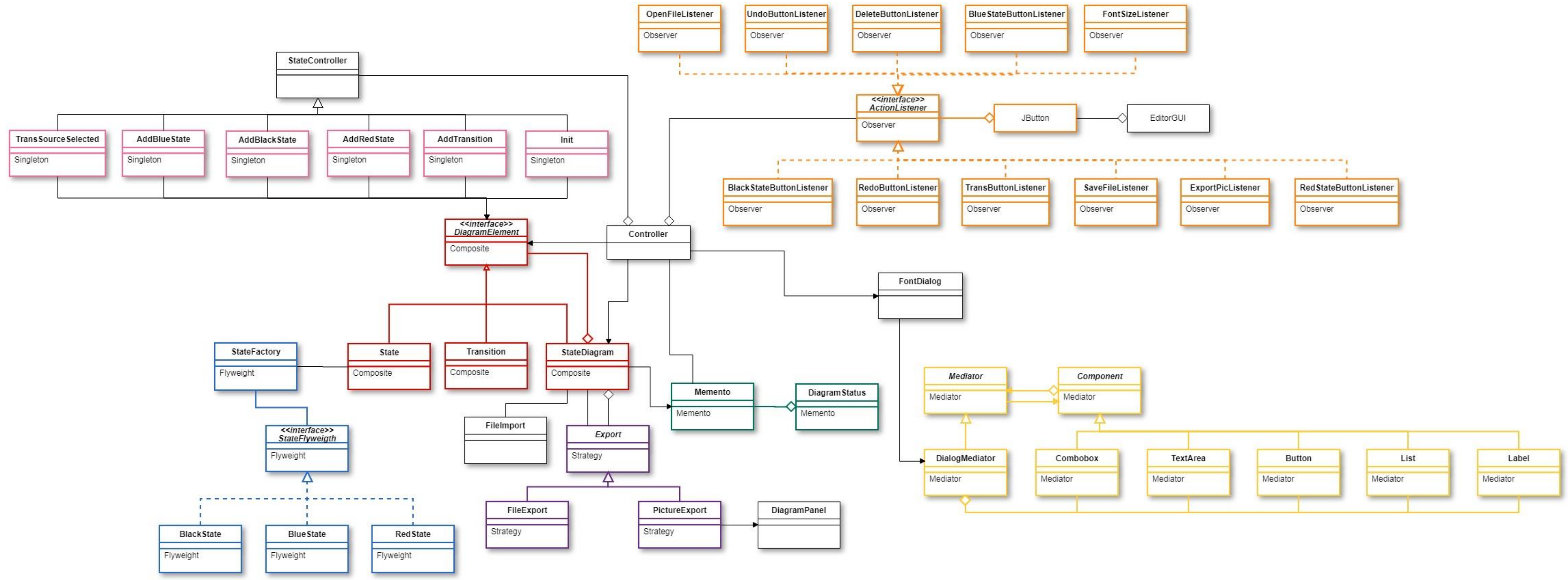


NOTE: type some description about this State Diagram ...

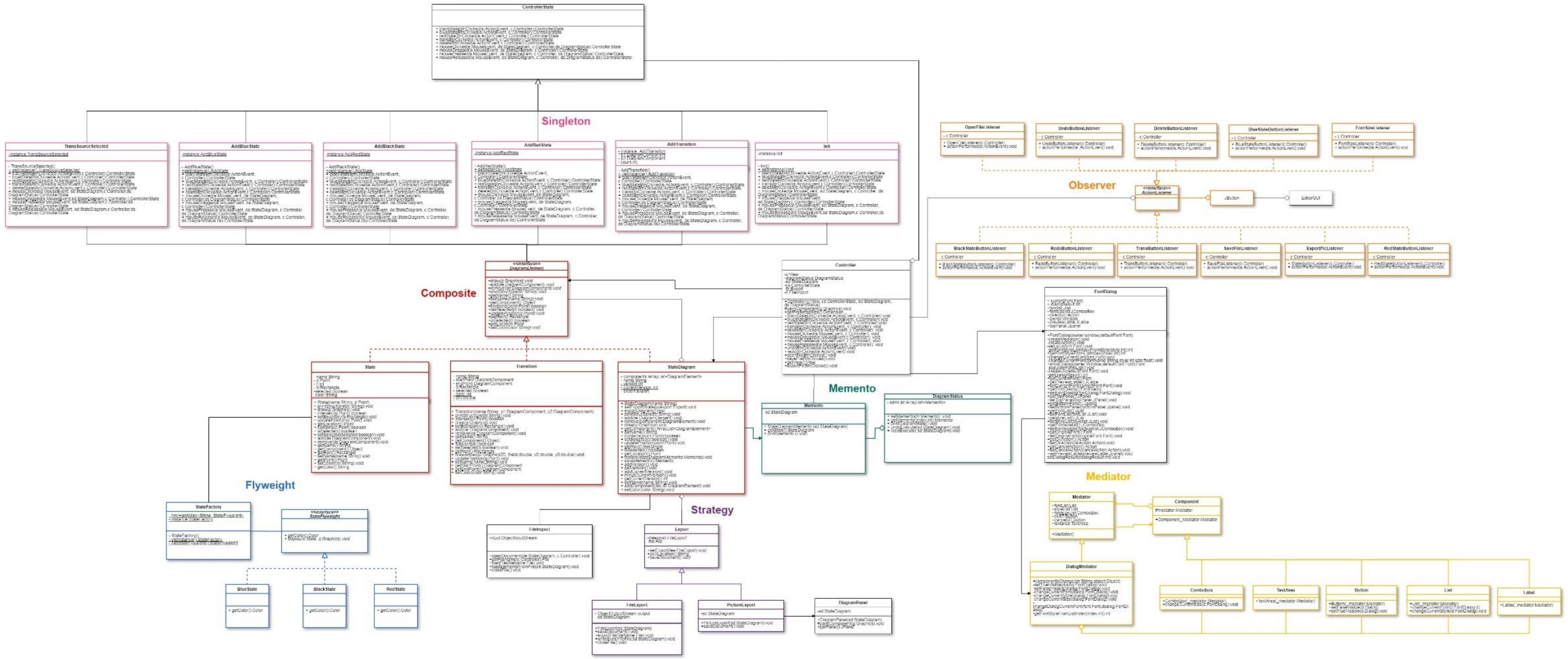
this is a state diagram editor.

3. Class Diagram

Simplify Class Diagram

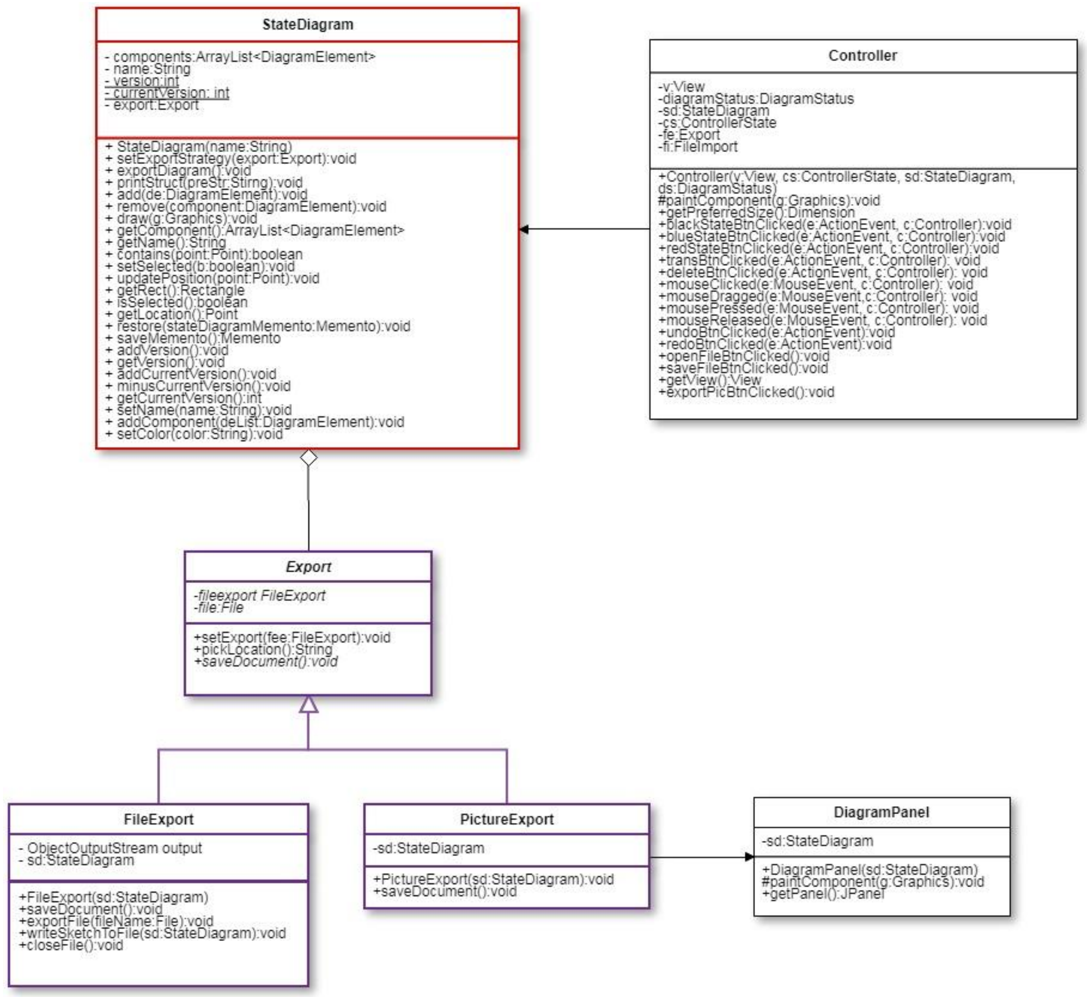


Full Class Diagram



4. Explanation of each pattern using causes

- Strategy Pattern


StateDiagram
- components:ArrayList<DiagramElement>
- name:String
- version:int
- currentVersion: int
- export:Export
+ StateDiagram(name:String)
+ setExportStrategy(export:Export):void
+ exportDiagram():void
+ printStruct(preStr:Stirng):void
+ add(de:DiagramElement):void
+ remove(component:DiagramElement):void
+ draw(g:Graphics):void
+ getComponent():ArrayList<DiagramElement>
+ getName():String
+ contains(point:Point):boolean
+ setSelected(b:boolean):void
+ updatePosition(point:Point):void
+ getRect():Rectangle
+ isSelected():boolean
+ getLocation():Point
+ restore(stateDiagramMemento:Memento):void
+ saveMemento():Memento
+ addVersion():void
+ getVersion():void
+ addCurrentVersion():void
+ minusCurrentVersion():void
+ getCurrentVersion():int
+ setName(name:String):void
+ addComponent(deList:DiagramElement):void
+ setColor(color:String):void
Controller
-v:View
-diagramStatus:DiagramStatus
-sd:StateDiagram
-cs:ControllerState
-fe:Export
-fi:FileImport
+Controller(v:View, cs:ControllerState, sd:StateDiagram, ds:DiagramStatus)
#paintComponent(g:Graphics):void
+getPreferredSize():Dimension
+blackStateBtnClicked(e:ActionEvent, c:Controller):void
+blueStateBtnClicked(e:ActionEvent, c:Controller):void
+redStateBtnClicked(e:ActionEvent, c:Controller):void
+transBtnClicked(e:ActionEvent, c:Controller): void
+deleteBtnClicked(e:ActionEvent, c:Controller): void
+mouseClicked(e:MouseEvent, c:Controller): void
+mouseDragged(e:MouseEvent,c:Controller): void
+mousePressed(e:MouseEvent, c:Controller): void
+mouseReleased(e:MouseEvent, c:Controller): void
+undoBtnClicked(e:ActionEvent):void
+redoBtnClicked(e:ActionEvent):void
+openFileBtnClicked():void
+saveFileBtnClicked():void
+getView():View
+exportPicBtnClicked():void
Export
-fileexport FileExport
-file:File
+setExport(fee:FileExport):void
+pickLocation():String
+saveDocument():void
FileExport
- ObjectOutputStream output
- sd:StateDiagram
+FileExport(sd:StateDiagram)
+saveDocument():void
+exportFile(fileName:File):void
+writeSketchToFile(sd:StateDiagram):void
+closeFile():void
PictureExport
-sd:StateDiagram
+PictureExport(sd:StateDiagram):void
+saveDocument():void
DiagramPanel
-sd:StateDiagram
+DiagramPanel(sd:StateDiagram)
#paintComponent(g:Graphics):void
+getPanel():JPanel

Text Book Name:	In our code Name:
Context	StateDiagram
Abstract Strategy	Export
Concrete Strategy	FileExport 、PictureExport
Clint	Controller

Our State Diagram Editor has two type of Export function. So we use Strategy Pattern to show different Export. They are FileExport and PictureExport.

Controller will pass StateDiagram to suitable Export Class, and calls StateDiagram. StateDiagram will pass itself to Export. Export is Abstract Class. FileExport and PictureExport will extend Export, and does themselves work.

FileExport will read StateDiagram's ArrayList(Component) then writes ArrayList's context in the File.

PictureExport will create new Panel, and calls DiagramPanel class to draw panel. DiagramPanel will call draw method of StateDiagram then draws StateDiagram's Component to new Panel. DiagramPanel will return Panel to PictureExport class, PictureExport will export Panel by PNG.

Coding：

Controller Class

```java
private Export fe;
private FileImport fi = new FileImport();

public void exportFileBtnClicked() {
    // ExportFile
    fe = new FileExport(sd);
    sd.setExportStrategy(fe);
    sd.exportDiagram();

}

public void exportPicBtnClicked() {
    // ExportPicture
    fe = new PictureExport(sd);
    sd.setExportStrategy(fe);
    sd.exportDiagram();

}
```

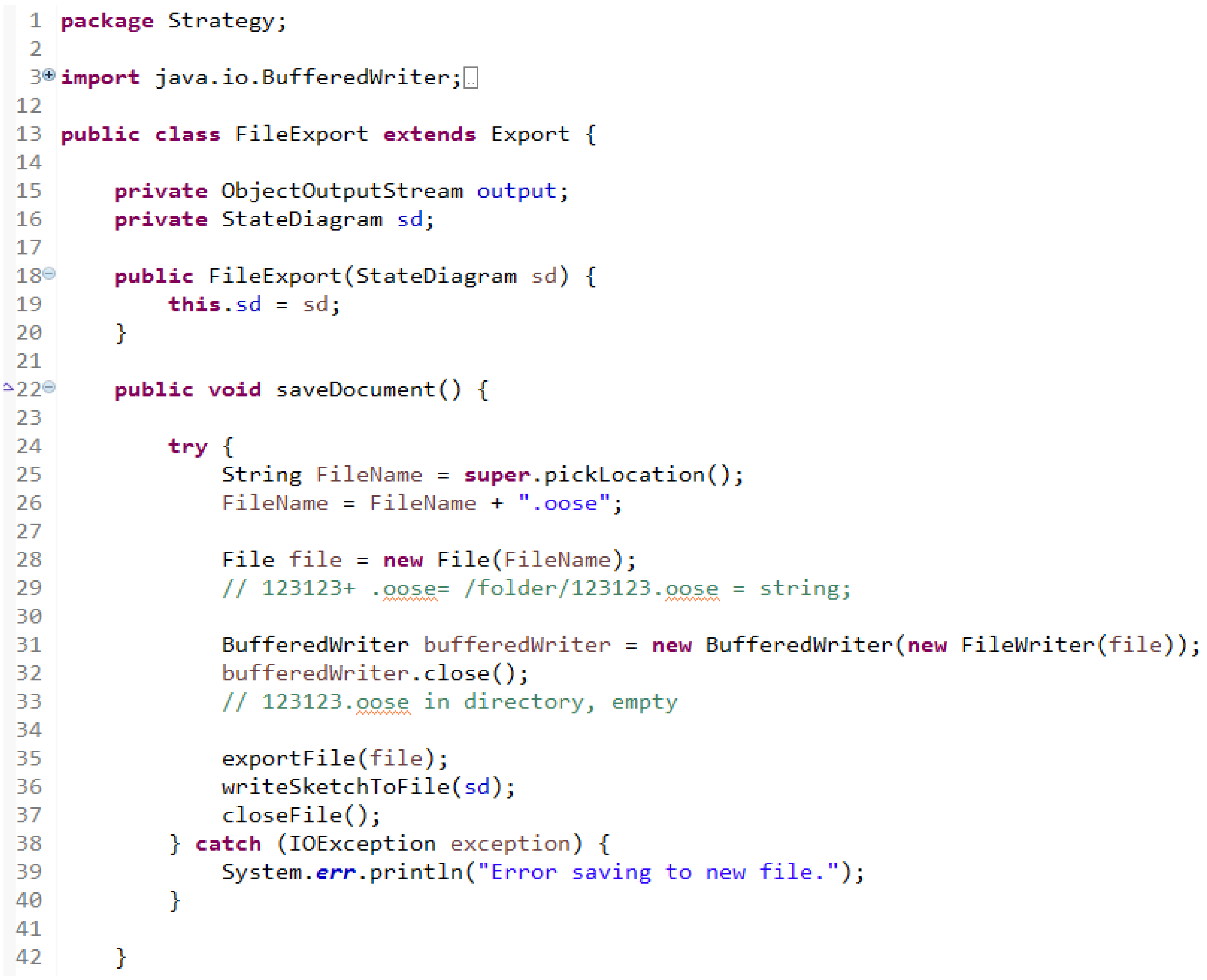
StateDiagram Class：

```java
    private Export export;

     public void setExportStrategy(Export export) {
         this.export = export;

     }

     public void exportDiagram() {
         this.export.saveDocument();
     }
```

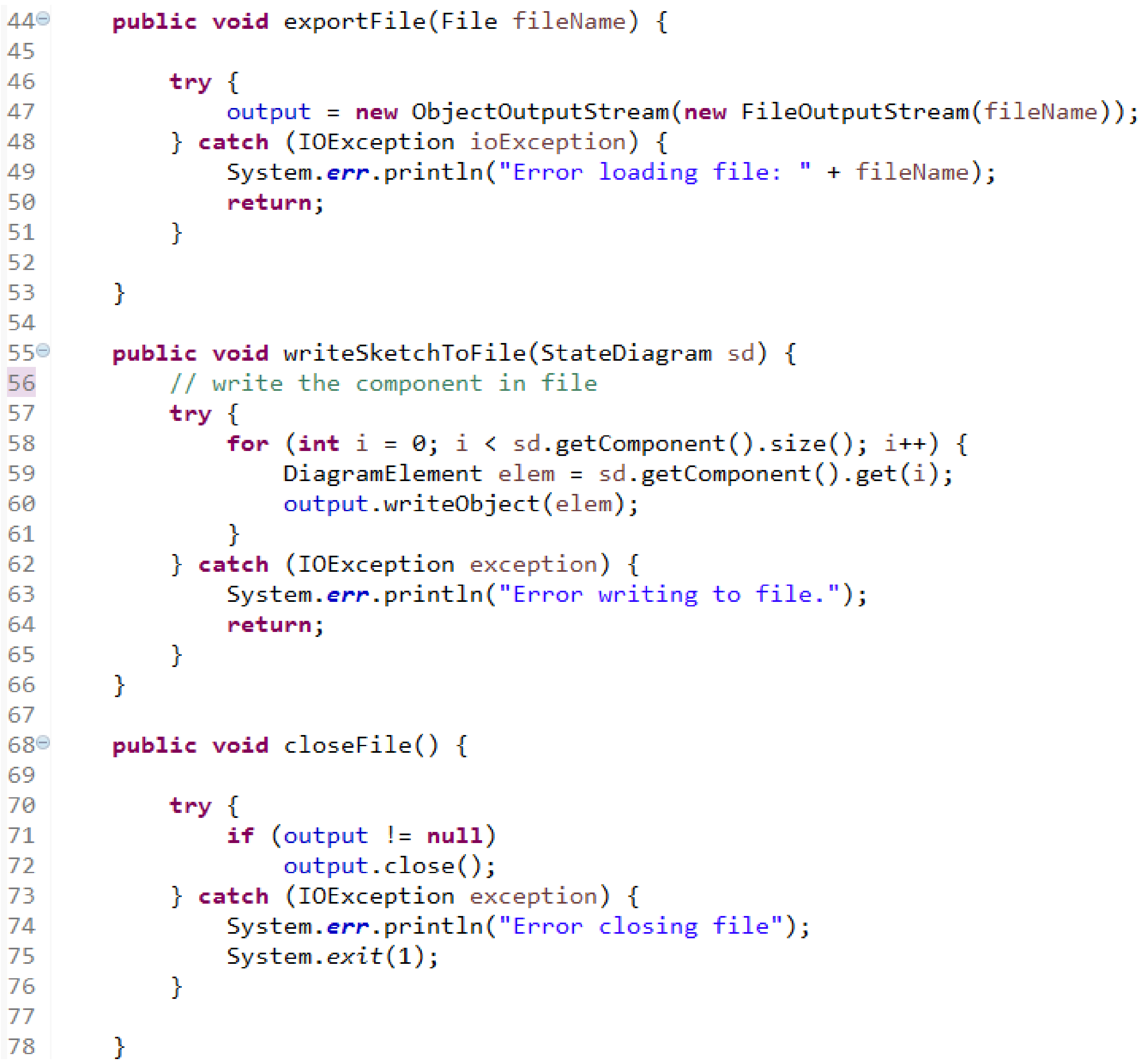
Export Class：

```java
package Strategy;

import java.io.File;

public abstract class Export {

    FileExport fileexport;

    public void setExport(FileExport fee) {
        fileexport = fee;
    }

    private File file;

    public String pickLocation() {

        JFileChooser chooseDirec = new JFileChooser();
        chooseDirec.setFileSelectionMode(JFileChooser.FILES_ONLY);
        chooseDirec.showSaveDialog(null);

        file = chooseDirec.getSelectedFile();
        return file.toString();
    }

    public abstract void saveDocument();

}
```

FileExport Class：

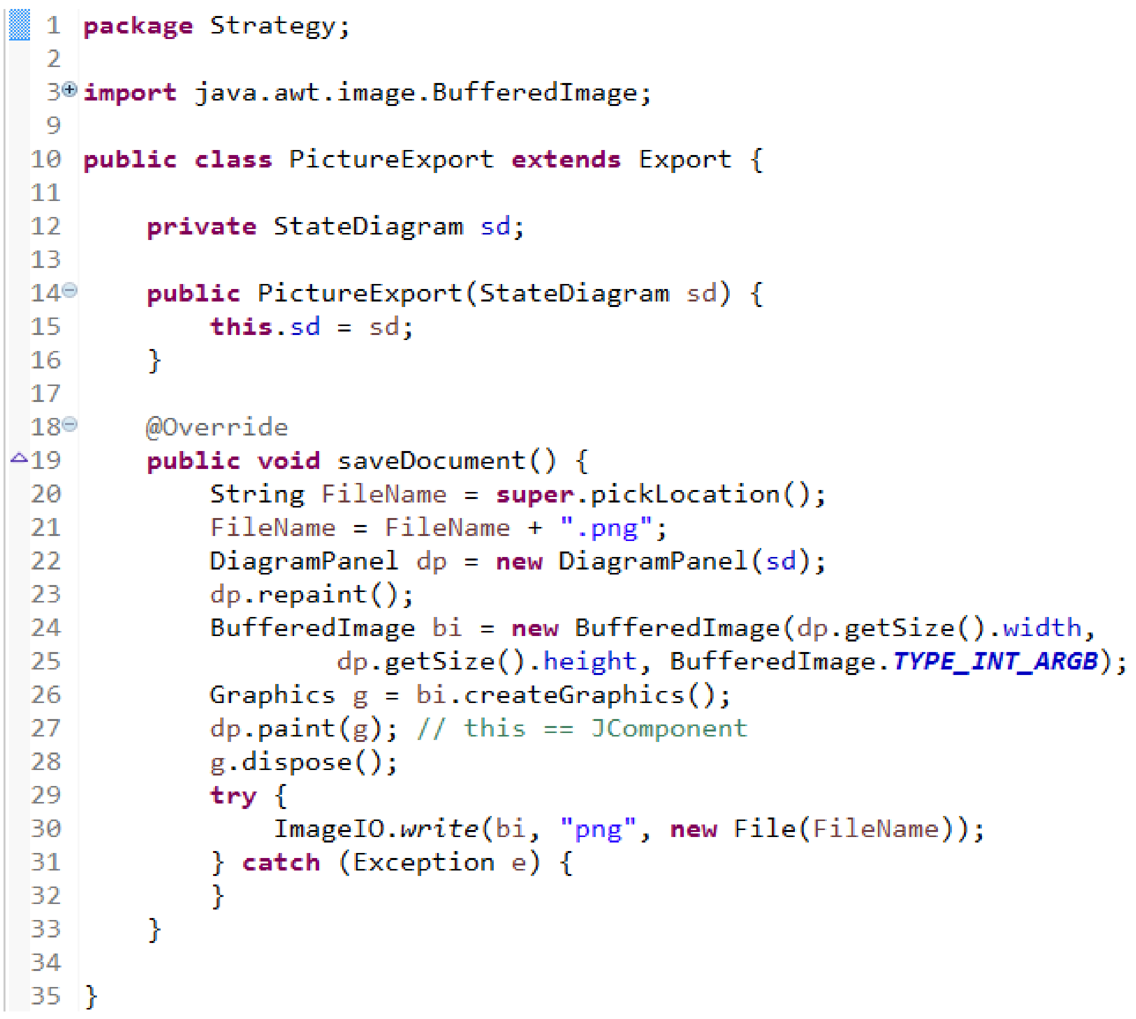


```java
package Strategy;

import java.io.BufferedWriter;

public class FileExport extends Export {

    private ObjectOutputStream output;
    private StateDiagram sd;

    public FileExport(StateDiagram sd) {
        this.sd = sd;
    }

    public void saveDocument() {

        try {
            String FileName = super.pickLocation();
            FileName = FileName + ".oose";

            File file = new File(FileName);
            // 123123+ .oose= /folder/123123.oose = string;

            BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(file));
            bufferedWriter.close();
            // 123123.oose in directory, empty

            exportFile(file);
            writeSketchToFile(sd);
            closeFile();
        } catch (IOException exception) {
            System.err.println("Error saving to new file.");
        }

    }
```



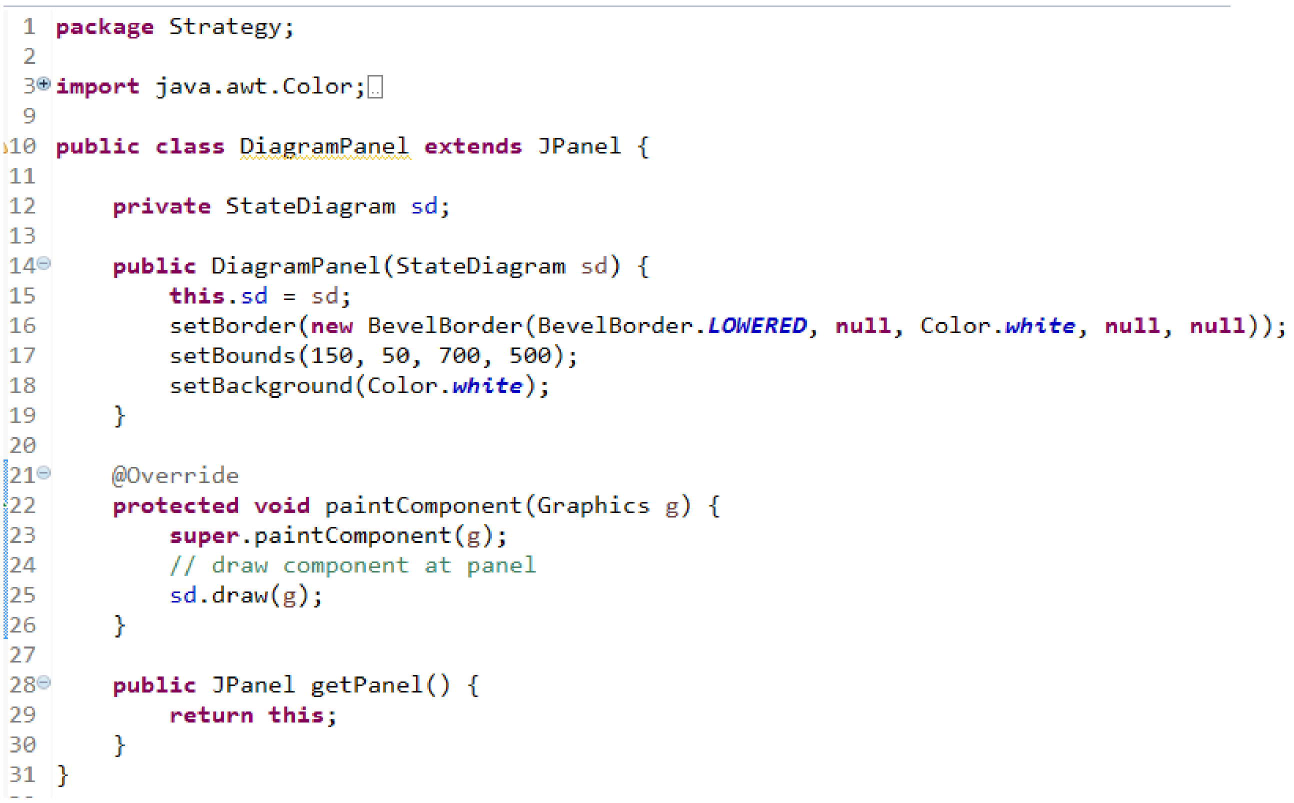
```java
    public void exportFile(File fileName) {

        try {
            output = new ObjectOutputStream(new FileOutputStream(fileName));
        } catch (IOException ioException) {
            System.err.println("Error loading file: " + fileName);
            return;
        }

    }

    public void writeSketchToFile(StateDiagram sd) {
        // write the component in file
        try {
            for (int i = 0; i < sd.getComponent().size(); i++) {
                DiagramElement elem = sd.getComponent().get(i);
                output.writeObject(elem);
            }
        } catch (IOException exception) {
            System.err.println("Error writing to file.");
            return;
        }
    }

    public void closeFile() {

        try {
            if (output != null)
                output.close();
        } catch (IOException exception) {
            System.err.println("Error closing file");
            System.exit(1);
        }

    }
```

PictureExport Class：



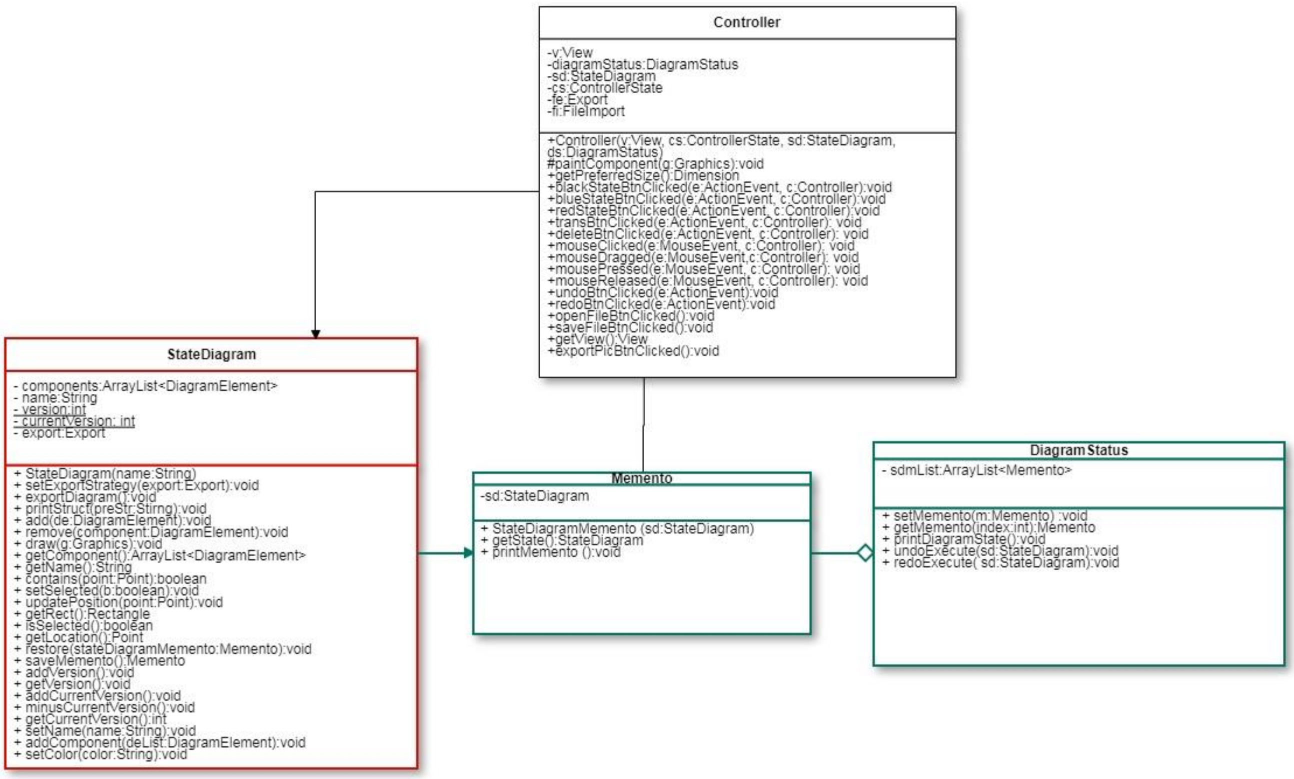
```java
package Strategy;

import java.awt.image.BufferedImage;

public class PictureExport extends Export {

    private StateDiagram sd;

    public PictureExport(StateDiagram sd) {
        this.sd = sd;
    }

    @Override
    public void saveDocument() {
        String FileName = super.pickLocation();
        FileName = FileName + ".png";
        DiagramPanel dp = new DiagramPanel(sd);
        dp.repaint();
        BufferedImage bi = new BufferedImage(dp.getSize().width,
                dp.getSize().height, BufferedImage.TYPE_INT_ARGB);
        Graphics g = bi.createGraphics();
        dp.paint(g); // this == JComponent
        g.dispose();
        try {
            ImageIO.write(bi, "png", new File(FileName));
        } catch (Exception e) {
        }
    }

}
```

DiagramPanel Class：



```java
package Strategy;

import java.awt.Color;

public class DiagramPanel extends JPanel {

    private StateDiagram sd;

    public DiagramPanel(StateDiagram sd) {
        this.sd = sd;
        setBorder(new BevelBorder(BevelBorder.LOWERED, null, Color.white, null, null));
        setBounds(150, 50, 700, 500);
        setBackground(Color.white);
    }

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        // draw component at panel
        sd.draw(g);
    }

    public JPanel getPanel() {
        return this;
    }
}
```

- Memento Pattern


Controller
-v:View
-diagramStatus:DiagramStatus
-sd:StateDiagram
-cs:ControllerState
-fe:Export
-fi:FileImport
+Controller(v:View, cs:ControllerState, sd:StateDiagram, ds:DiagramStatus)
#paintComponent(g:Graphics):void
+getPreferredSize():Dimension
+blackStateBtnClicked(e:ActionEvent, c:Controller):void
+blueStateBtnClicked(e:ActionEvent, c:Controller):void
+redStateBtnClicked(e:ActionEvent, c:Controller):void
+transBtnClicked(e:ActionEvent, c:Controller): void
+deleteBtnClicked(e:ActionEvent, c:Controller): void
+mouseClicked(e:MouseEvent, c:Controller): void
+mouseDragged(e:MouseEvent,c:Controller): void
+mousePressed(e:MouseEvent, c:Controller): void
+mouseReleased(e:MouseEvent, c:Controller): void
+undoBtnClicked(e:ActionEvent):void
+redoBtnClicked(e:ActionEvent):void
+openFileBtnClicked():void
+saveFileBtnClicked():void
+getView():View
+exportPicBtnClicked():void
StateDiagram
- components:ArrayList<DiagramElement>
- name:String
- version:int
- currentVersion: int
- export:Export
+ StateDiagram(name:String)
+ setExportStrategy(export:Export):void
+ exportDiagram():void
+ printStruct(preStr:Stirng):void
+ add(de:DiagramElement):void
+ remove(component:DiagramElement):void
+ draw(g:Graphics):void
+ getComponent():ArrayList<DiagramElement>
+ getName():String
+ contains(point:Point):boolean
+ setSelected(b:boolean):void
+ updatePosition(point:Point):void
+ getRect():Rectangle
+ isSelected():boolean
+ getLocation():Point
+ restore(stateDiagramMemento:Memento):void
+ saveMemento():Memento
+ addVersion():void
+ getVersion():void
+ addCurrentVersion():void
+ minusCurrentVersion():void
+ getCurrentVersion():int
+ setName(name:String):void
+ addComponent(deList:DiagramElement):void
+ setColor(color:String):void
Memento
-sd:StateDiagram
+ StateDiagramMemento (sd:StateDiagram)
+ getState():StateDiagram
+ printMemento ():void
Diagram Status
- sdmList:ArrayList<Memento>
+ setMemento(m:Memento) :void
+ getMemento(index:int):Memento
+ printDiagramState():void
+ undoExecute(sd:StateDiagram):void
+ redoExecute(sd:StateDiagram):void

Text Book Name:	**In our code Name:**
Originator	StateDiagram
Memento	Memento
Caretaker	DiagramStatus
Clint	Controller

Without destroying the Encapsulation, we use Memento Pattern to access the situations of StateDiagram and DiagramState.

StateDiagram has too much responsibility, so we use this pattern to share responsibility and store StateDiagram. It can keep Encapsulation intact and record the current state of StateDiagram, and store the state in DiagramState, also, it can return last state, implement undo and redo function.

StateDiagram create Memento and store the state into DiagramState. DiagramState does not know the internal information and it will not to ask about it. DiagramState can decline coupling and share responsibility with StateDiagram.

Users can use Undo function to keep the state of StateDiagram and use Redo function to restore. When user want to go back to last step or going to next step, the user can use Undo or Redo function.

Coding：

Controller Class：

```
diagramStatus.setMemento(sd.saveMemento()); // initial memento
```

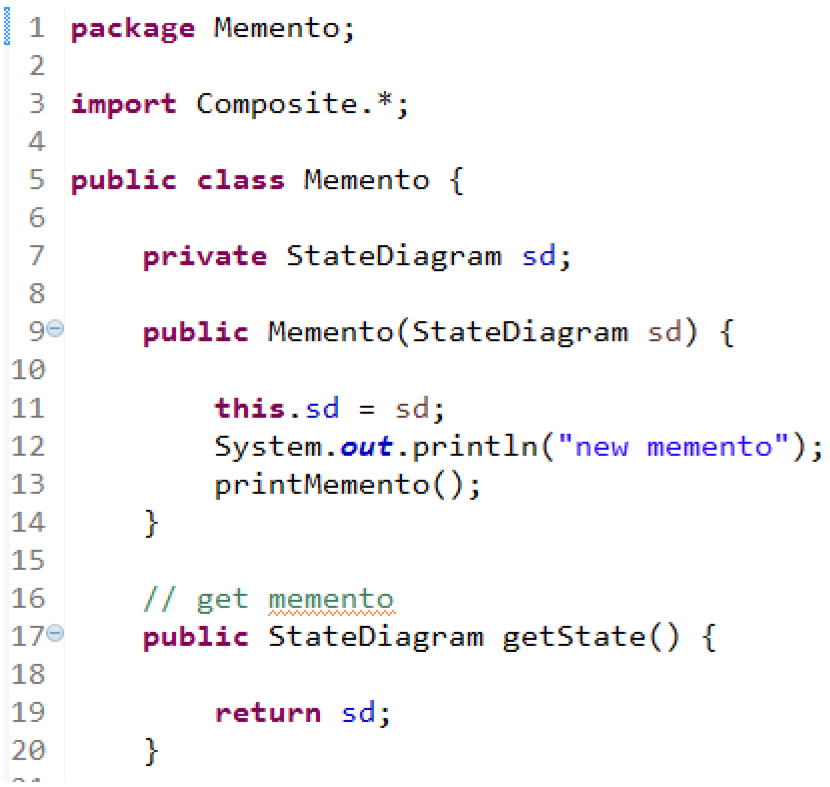
```java
    public void undoBtnClicked(ActionEvent e) {
        System.out.println("undo was clicked");

        // control the range
        if (1 <= sd.getCurrentVersion()) {
            this.v.getRedoButton().setEnabled(true);
            diagramStatus.undoExecute(sd);
            repaint();
        } else {
            this.v.getUndoButton().setEnabled(false);
            System.out.println("command can't execute");
        }
        repaint();
    }

    public void redoBtnClicked(ActionEvent e) {

        System.out.println("redo was clicked");

        // control the range
        if (sd.getCurrentVersion() < (sd.getVersion())) {
            this.v.getUndoButton().setEnabled(true);
            diagramStatus.redoExecute(sd);
            repaint();
        } else {
            this.v.getRedoButton().setEnabled(false);
            System.out.println("command can't execute");
        }

        repaint();
    }
```

StateDiamgram Class：

```java
private static int version = 0; // it present how many memento version
private static int currentVersion = 0;// a stop for redo & undo

// get the StateDiagram history in Memento
public void restore(Memento stateDiagramMemento) {
    components = stateDiagramMemento.getState().getComponent();
}
```

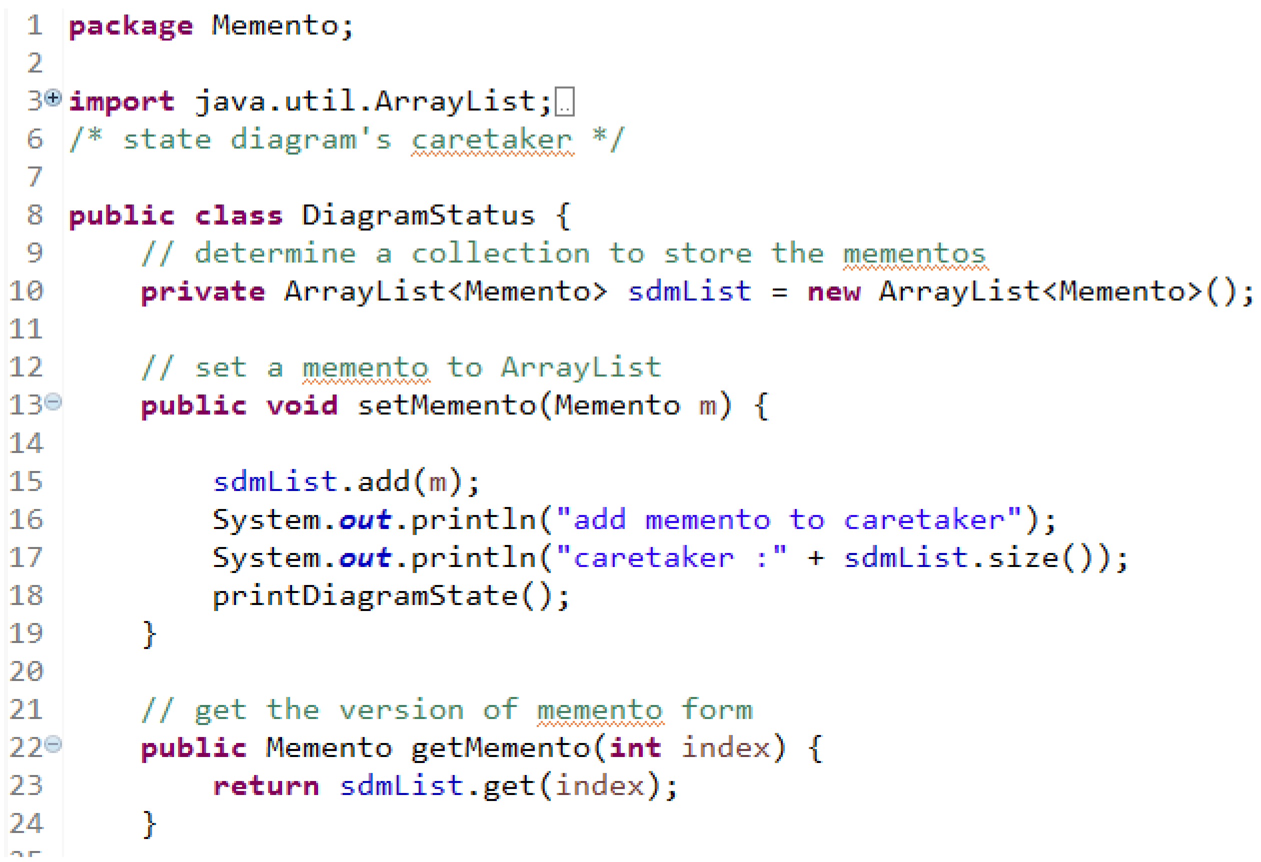
```
 // create a new Memento with a new StateDiagram
 public Memento saveMemento() {
     System.out.println("store the stateDiagram history to memento");

     // create new StateDiagram to store clone copy
     // create new ArrayList to store clone copy
     StateDiagram sd = new StateDiagram(name);
     ArrayList<DiagramElement> adcs = new ArrayList<DiagramElement>();
     adcs = (ArrayList<DiagramElement>) this.components.clone(); // clone
     sd.setComponent(adcs);
     // new StateDiagram set clone copy

     return new Memento(sd);
     // create a memento to store clone StateDiagram
 }

public void addVersion() {

    version += 1;
    System.out.println("version:" + version);
}

public int getVersion() {
    return version;
}

public void addCurrentVersion() {
    currentVersion += 1;
}

public void minusCurrentVersion() {
    currentVersion -= 1;
}

public int getCurrentVersion() {

    return currentVersion;

}
```

Memento Class：



```java
package Memento;

import Composite.*;

public class Memento {

    private StateDiagram sd;

    public Memento(StateDiagram sd) {

        this.sd = sd;
        System.out.println("new memento");
        printMemento();
    }

    // get memento
    public StateDiagram getState() {

        return sd;
    }
```

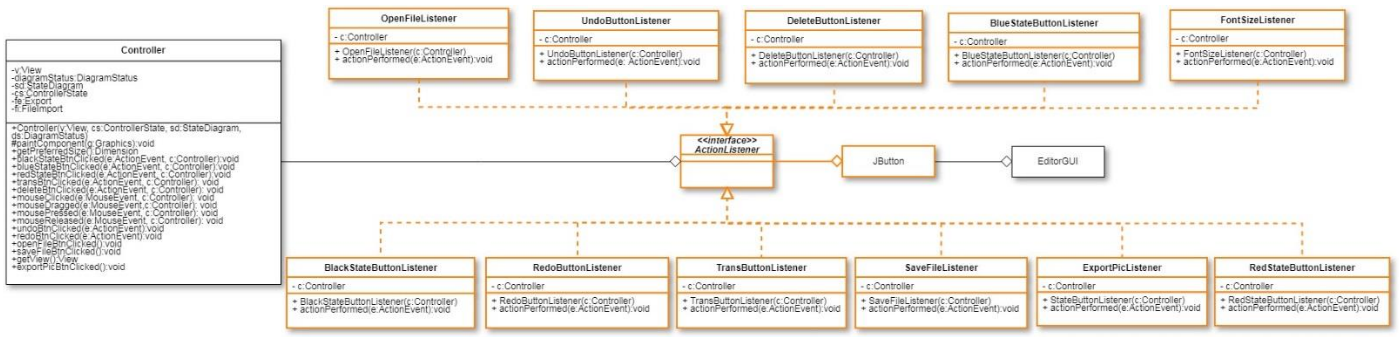
DiagramStatus Class：



```java
package Memento;

import java.util.ArrayList;
/* state diagram's caretaker */

public class DiagramStatus {
    // determine a collection to store the mementos
    private ArrayList<Memento> sdmList = new ArrayList<Memento>();

    // set a memento to ArrayList
    public void setMemento(Memento m) {

        sdmList.add(m);
        System.out.println("add memento to caretaker");
        System.out.println("caretaker :" + sdmList.size());
        printDiagramState();
    }

    // get the version of memento form
    public Memento getMemento(int index) {
        return sdmList.get(index);
    }
```

```
public void undoExecute(StateDiagram sd) {

    // call the StateDigram to do restore() & minusCurrentVersion()
    System.out.println("undo execute");
    sd.restore(this.getMemento(sd.getCurrentVersion() - 1));// back to previous step
    sd.minusCurrentVersion();
}

public void redoExecute(StateDiagram sd) {

    // call the StateDigram to do restore() & addCurrentVersion()
    System.out.println("redo execute");
    sd.restore(this.getMemento(sd.getCurrentVersion() + 1));// go to the next step
    sd.addCurrentVersion();

}
```

- Observer Pattern


Controller
-v:View
-diagramStatus:DiagramStatus
-sd:StateDiagram
-cs:ControllerState
-fe:Export
-fi:FileImport
+Controller(v:View, cs:ControllerState, sd:StateDiagram, ds:DiagramStatus)
#paintComponent(g:Graphics):void
+getPreferredSize():Dimension
+blackStateBtnClicked(e:ActionEvent, c:Controller):void
+blueStateBtnClicked(e:ActionEvent, c:Controller):void
+redStateBtnClicked(e:ActionEvent, c:Controller):void
+transBtnClicked(e:ActionEvent, c:Controller): void
+deleteBtnClicked(e:ActionEvent, c:Controller): void
+mouseClicked(e:MouseEvent, c:Controller): void
+mouseDragged(e:MouseEvent,c:Controller): void
+mousePressed(e:MouseEvent, c:Controller): void
+mouseReleased(e:MouseEvent, c:Controller): void
+undoBtnClicked(e:ActionEvent):void
+redoBtnClicked(e:ActionEvent):void
+openFileBtnClicked():void
+saveFileBtnClicked():void
+getView():View
+exportPicBtnClicked():void
OpenFileListener
- c:Controller
+ OpenFileListener(c:Controller)
+ actionPerformed(e:ActionEvent):void
UndoButtonListener
- c:Controller
+ UndoButtonListener(c:Controller)
+ actionPerformed(e: ActionEvent):void
DeleteButtonListener
- c:Controller
+ DeleteButtonListener(c:Controller)
+ actionPerformed(e:ActionEvent):void
BlueStateButtonListener
- c:Controller
+ BlueStateButtonListener(c:Controller)
+ actionPerformed(e:ActionEvent):void
FontSizeListener
- c:Controller
+ FontSizeListener(c:Controller)
+ actionPerformed(e:ActionEvent):void
<<interface>>
ActionListener
JButton
EditorGUI
BlackStateButtonListener
- c:Controller
+ BlackStateButtonListener(c:Controller)
+ actionPerformed(e:ActionEvent):void
RedoButtonListener
- c:Controller
+ RedoButtonListener(c:Controller)
+ actionPerformed(e:ActionEvent):void
TransButtonListener
- c:Controller
+ TransButtonListener(c:Controller)
+ actionPerformed(e:ActionEvent):void
SaveFileListener
- c:Controller
+ SaveFileListener(c:Controller)
+ actionPerformed(e:ActionEvent):void
ExportPicListener
- c:Controller
+ StateButtonListener(c:Controller)
+ actionPerformed(e:ActionEvent):void
RedStateButtonListener
- c:Controller
+ RedStateButtonListener(c:Controller)
+ actionPerformed(e:ActionEvent):void

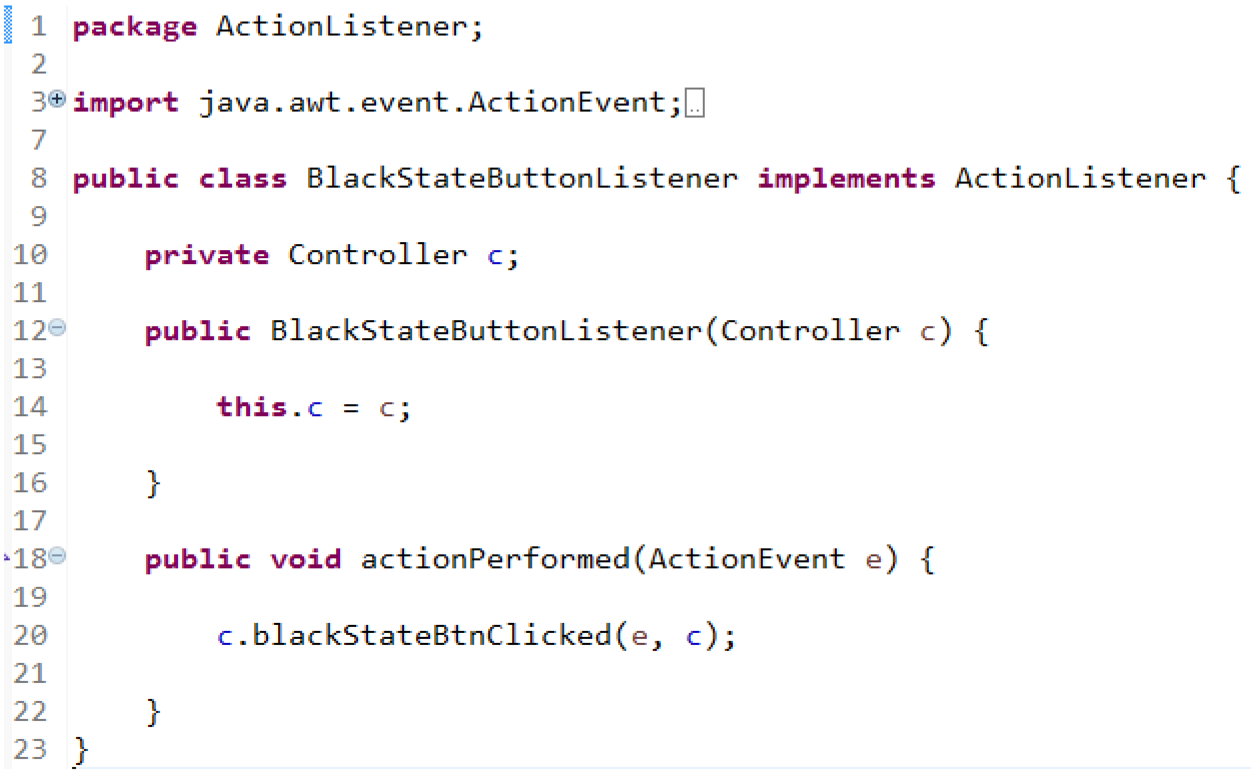
Text Book Name:	**In our code Name:**
Concrete observable	JButton
Abstract Observer	ActionListener
Concrete Observer	BlackStateButtonListener、RedoButtonListener、TransButtonListener、SaveFileListener、ExportPicListener、RedStateButtonListener、FontSizeListener、BlueStateButtonListener、OpenFileListener、DeleteButtonListener、UndoButtonListener
Clint	Controller

Because controller need to know user's state all the time, such as Click, AddBlackState...etc, then it can call the correspond object. ActironListener is a kind of way to implement Observer, so we use Observer Pattern in this situation.

For Example: when user click on a button, ActionListener will know the JButton just be clicked, then MouseClickedListener will touch off and send the value to Controller.

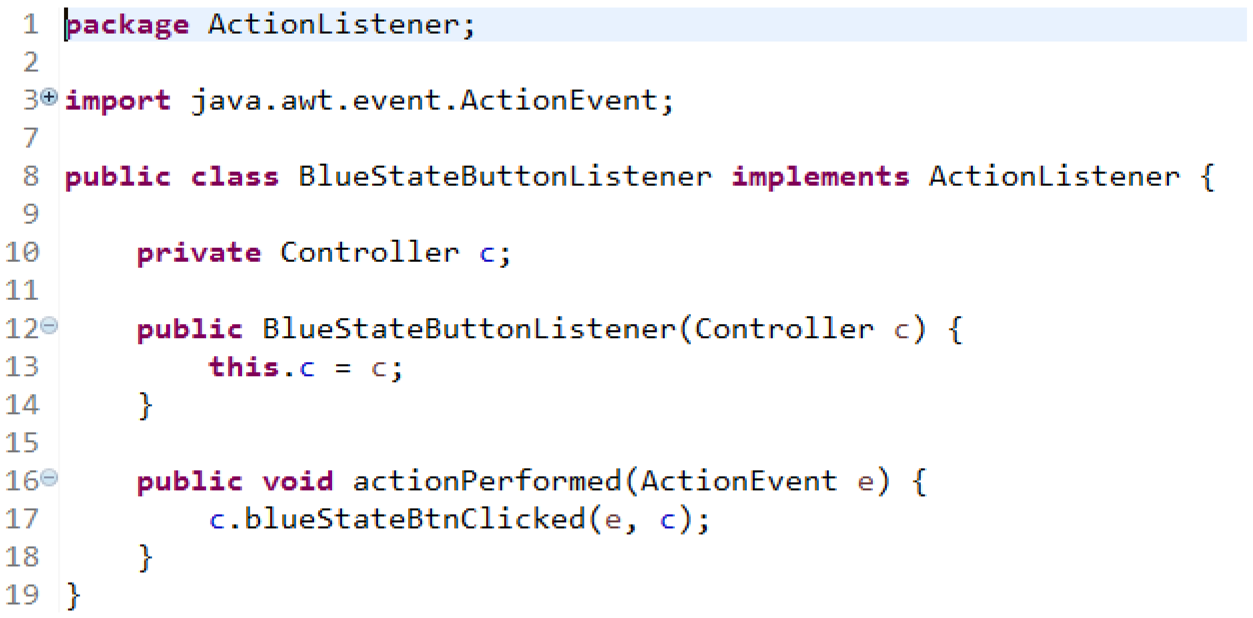
Coding：

BlackStateButtonListener Class：



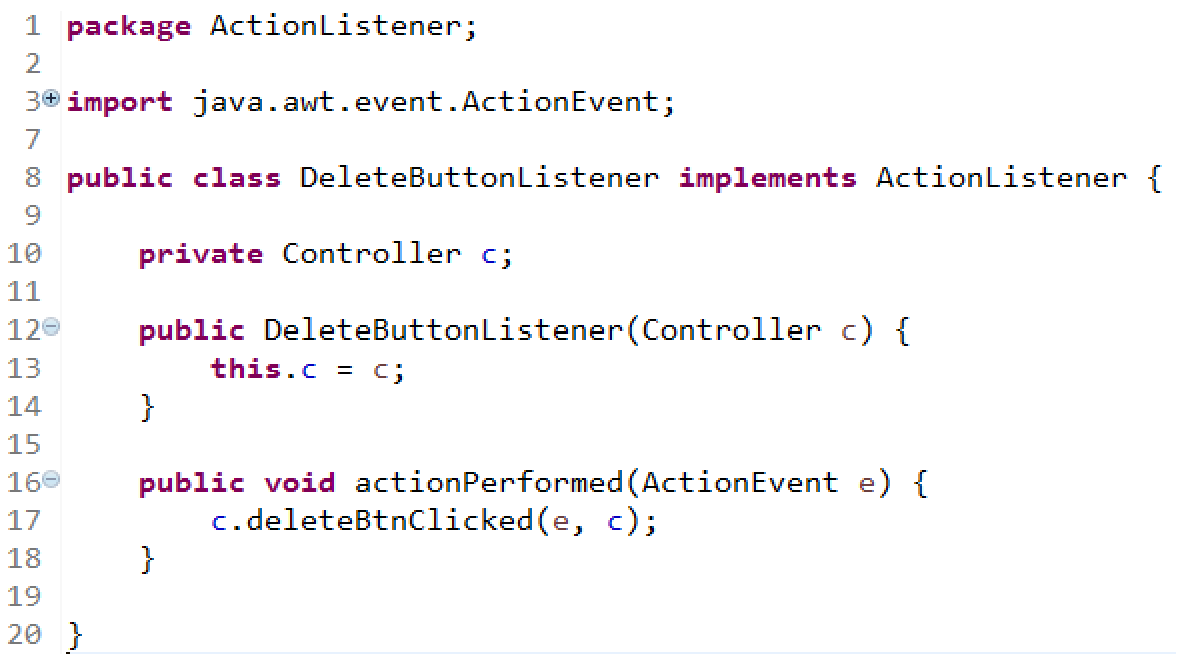
```java
package ActionListener;

import java.awt.event.ActionEvent;

public class BlackStateButtonListener implements ActionListener {

    private Controller c;

    public BlackStateButtonListener(Controller c) {

        this.c = c;

    }

    public void actionPerformed(ActionEvent e) {

        c.blackStateBtnClicked(e, c);

    }
}
```

BlueStateButtonListener Class：



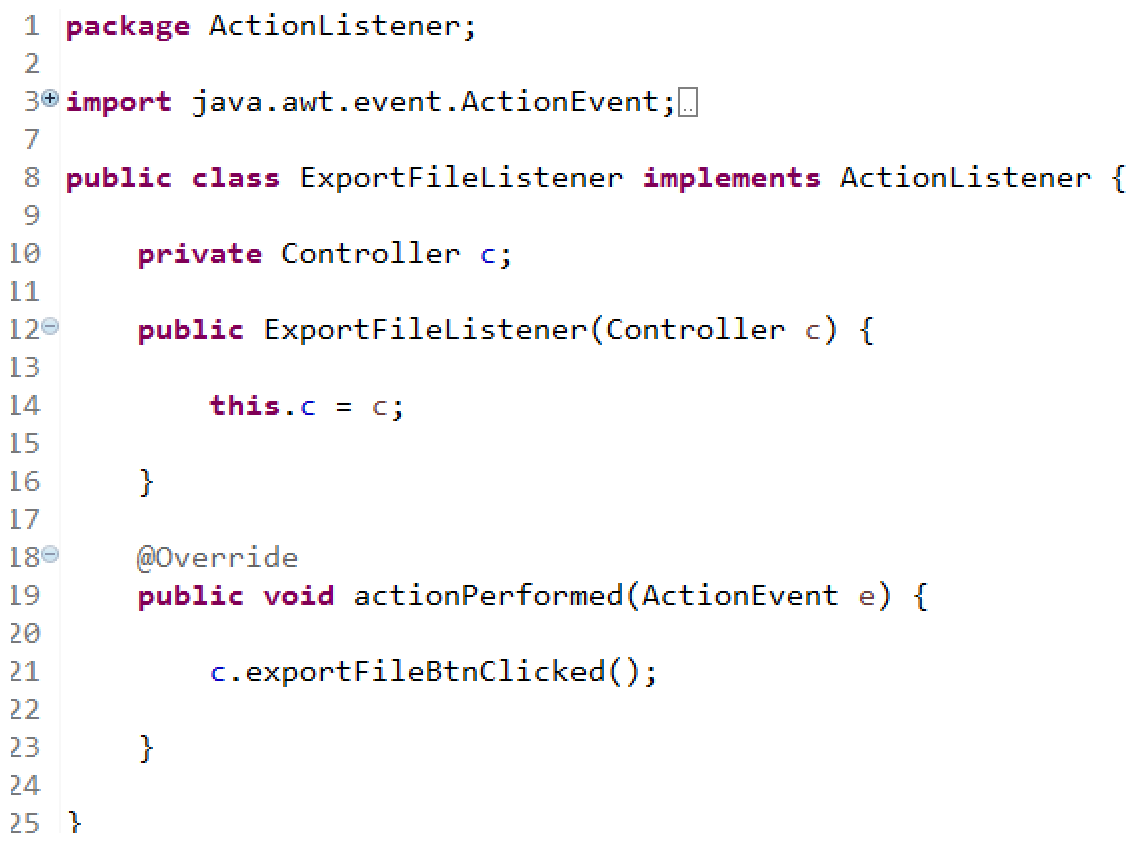
```java
package ActionListener;

import java.awt.event.ActionEvent;

public class BlueStateButtonListener implements ActionListener {

    private Controller c;

    public BlueStateButtonListener(Controller c) {
        this.c = c;
    }

    public void actionPerformed(ActionEvent e) {
        c.blueStateBtnClicked(e, c);
    }
}
```

DeleteButtonListener Class：



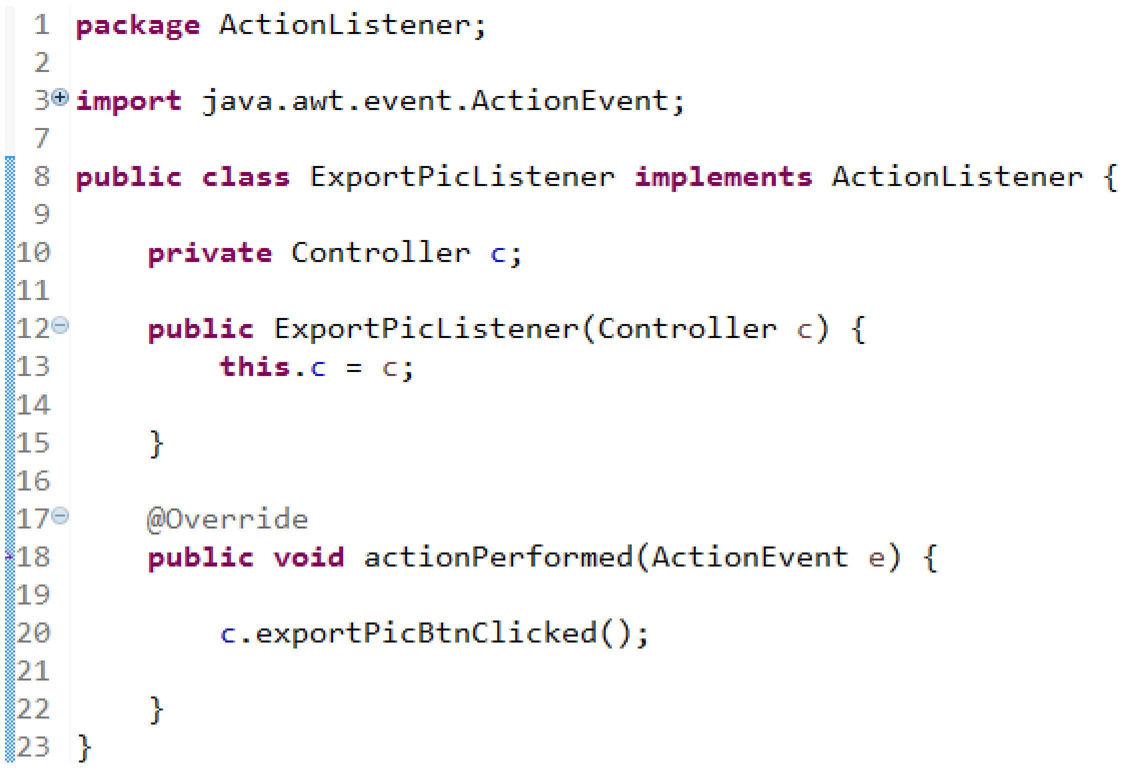
```java
package ActionListener;

import java.awt.event.ActionEvent;

public class DeleteButtonListener implements ActionListener {

    private Controller c;

    public DeleteButtonListener(Controller c) {
        this.c = c;
    }

    public void actionPerformed(ActionEvent e) {
        c.deleteBtnClicked(e, c);
    }

}
```

ExportFileListener Class：



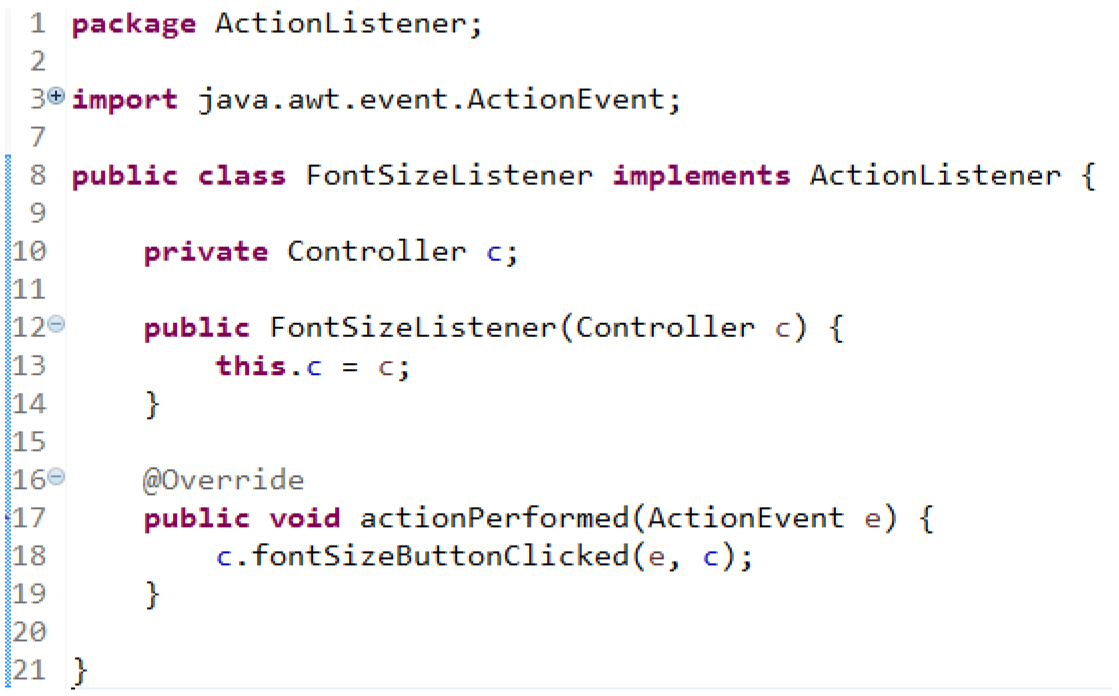
```java
package ActionListener;

import java.awt.event.ActionEvent;

public class ExportFileListener implements ActionListener {

    private Controller c;

    public ExportFileListener(Controller c) {

        this.c = c;

    }

    @Override
    public void actionPerformed(ActionEvent e) {

        c.exportFileBtnClicked();

    }

}
```

ExportPicListener Class：



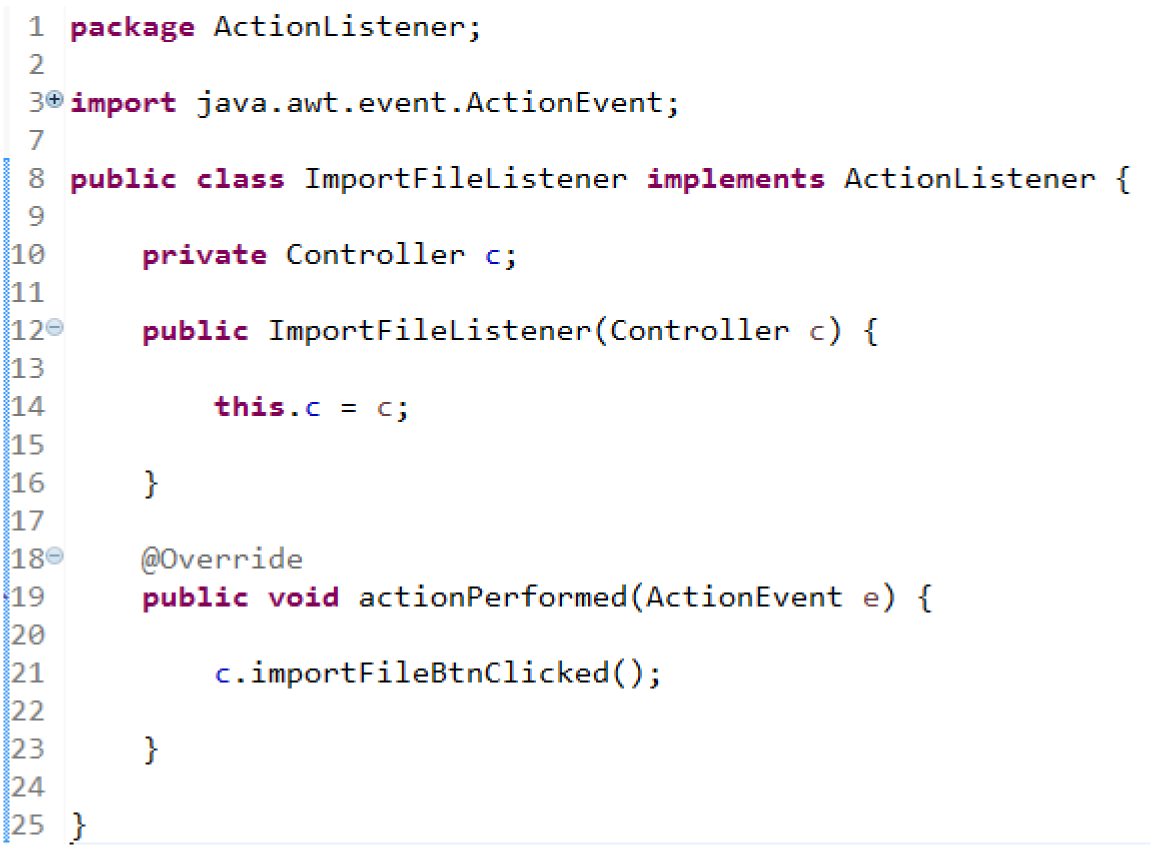
```java
package ActionListener;

import java.awt.event.ActionEvent;

public class ExportPicListener implements ActionListener {

    private Controller c;

    public ExportPicListener(Controller c) {
        this.c = c;

    }

    @Override
    public void actionPerformed(ActionEvent e) {

        c.exportPicBtnClicked();

    }
}
```

FontSizeListener Class：



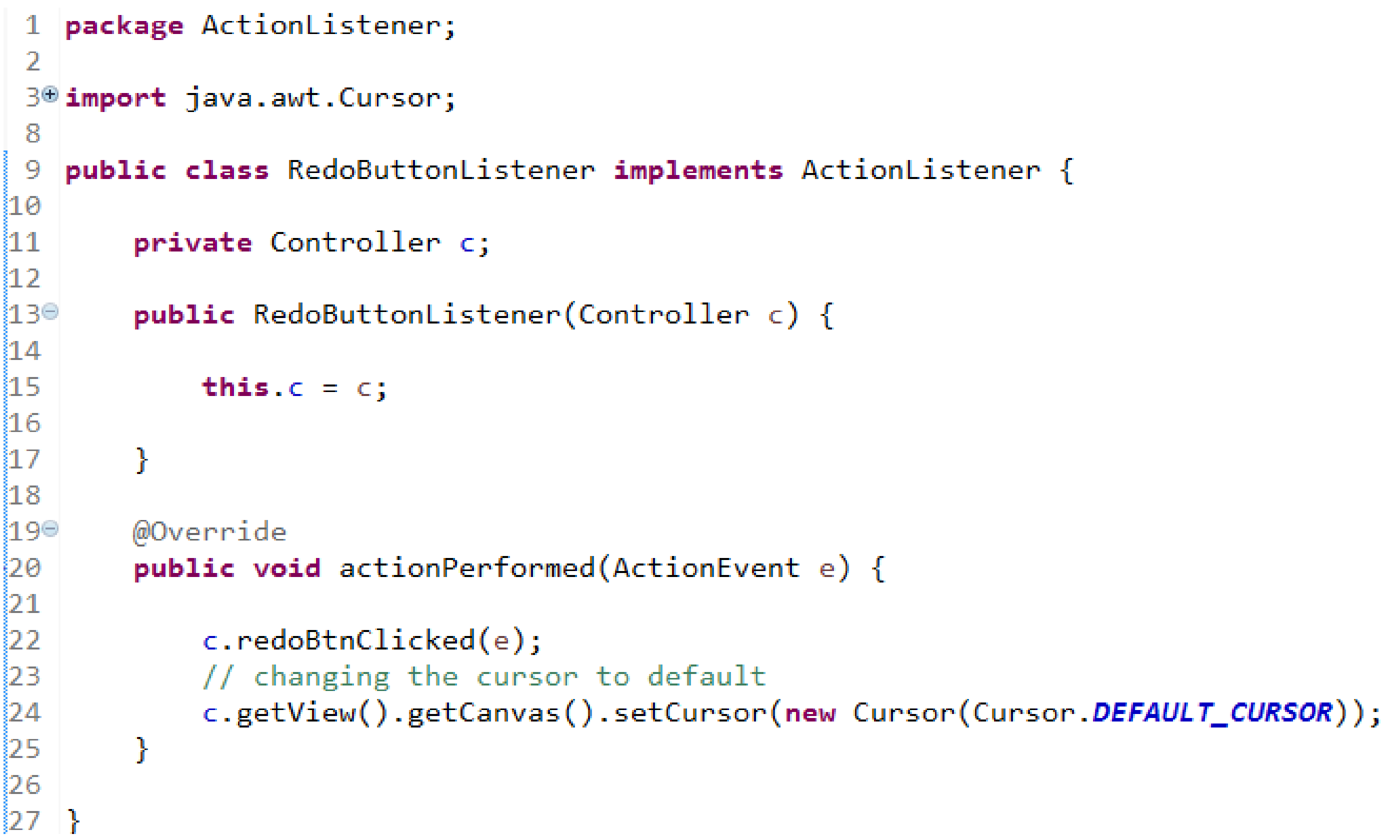
```java
package ActionListener;

import java.awt.event.ActionEvent;

public class FontSizeListener implements ActionListener {

    private Controller c;

    public FontSizeListener(Controller c) {
        this.c = c;
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        c.fontSizeButtonClicked(e, c);
    }

}
```

ImportFileListener Class：



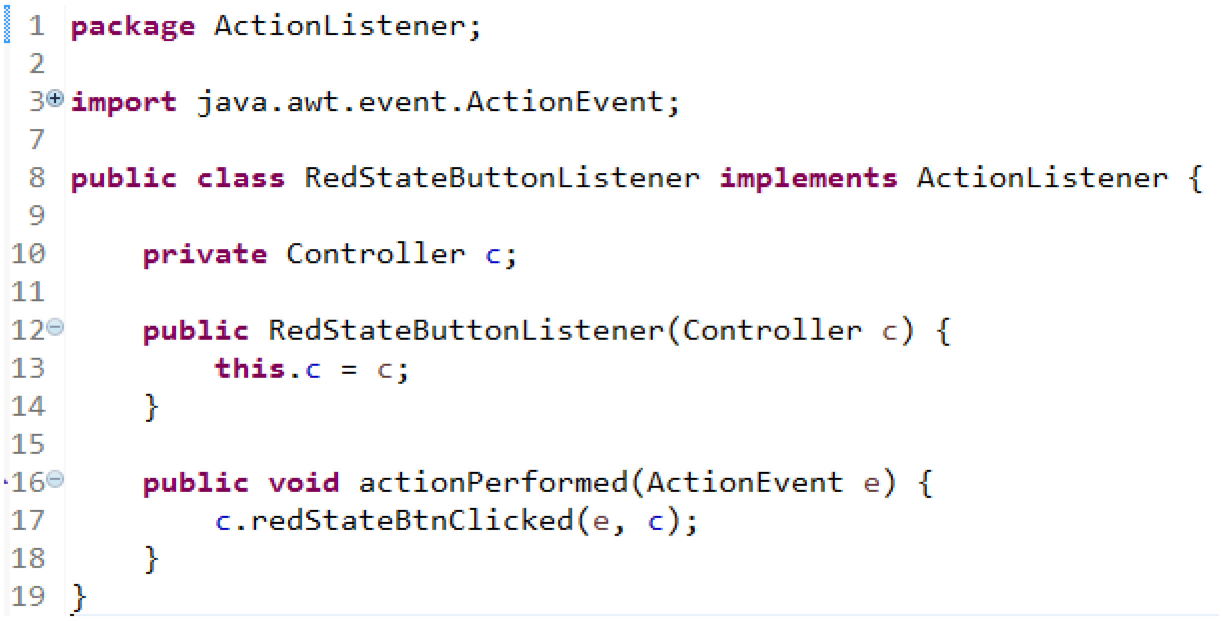
```java
package ActionListener;

import java.awt.event.ActionEvent;

public class ImportFileListener implements ActionListener {

    private Controller c;

    public ImportFileListener(Controller c) {

        this.c = c;

    }

    @Override
    public void actionPerformed(ActionEvent e) {

        c.importFileBtnClicked();

    }

}
```

RedoButtonListener Class：



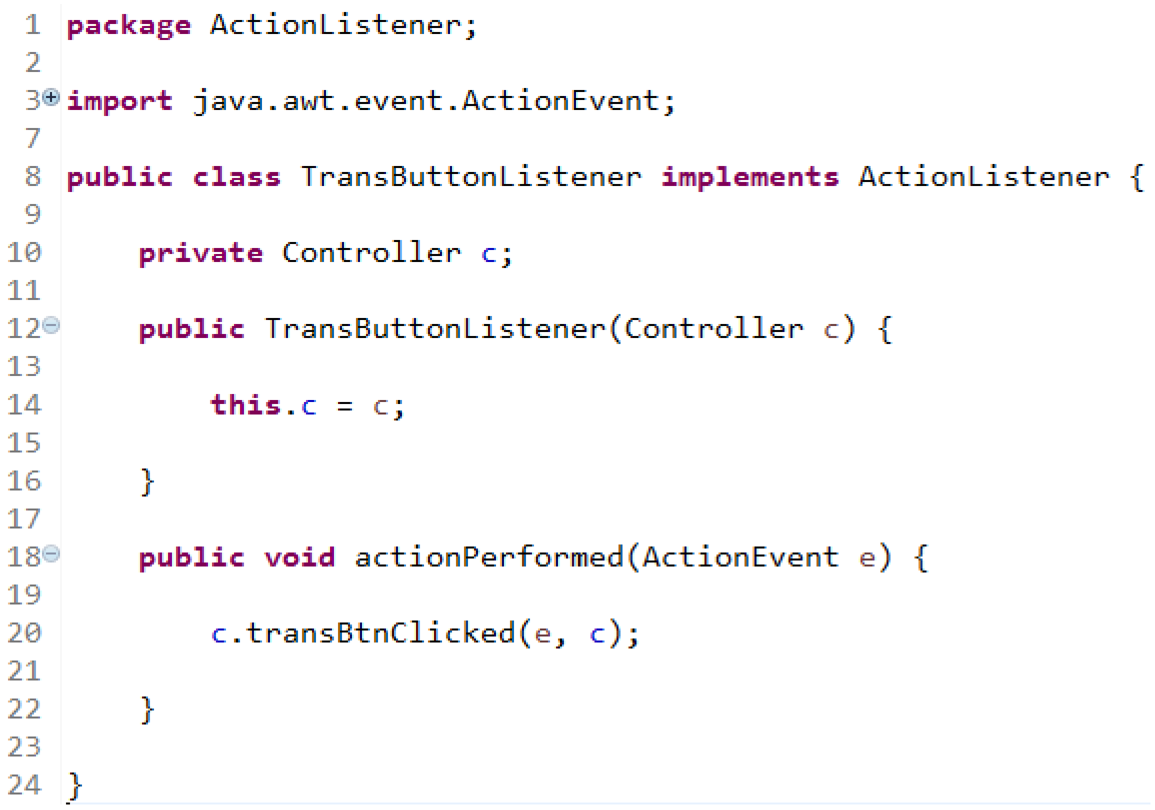
```java
package ActionListener;

import java.awt.Cursor;

public class RedoButtonListener implements ActionListener {

    private Controller c;

    public RedoButtonListener(Controller c) {

        this.c = c;

    }

    @Override
    public void actionPerformed(ActionEvent e) {

        c.redoBtnClicked(e);
        // changing the cursor to default
        c.getView().getCanvas().setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
    }

}
```

RedStateButtonListener Class：



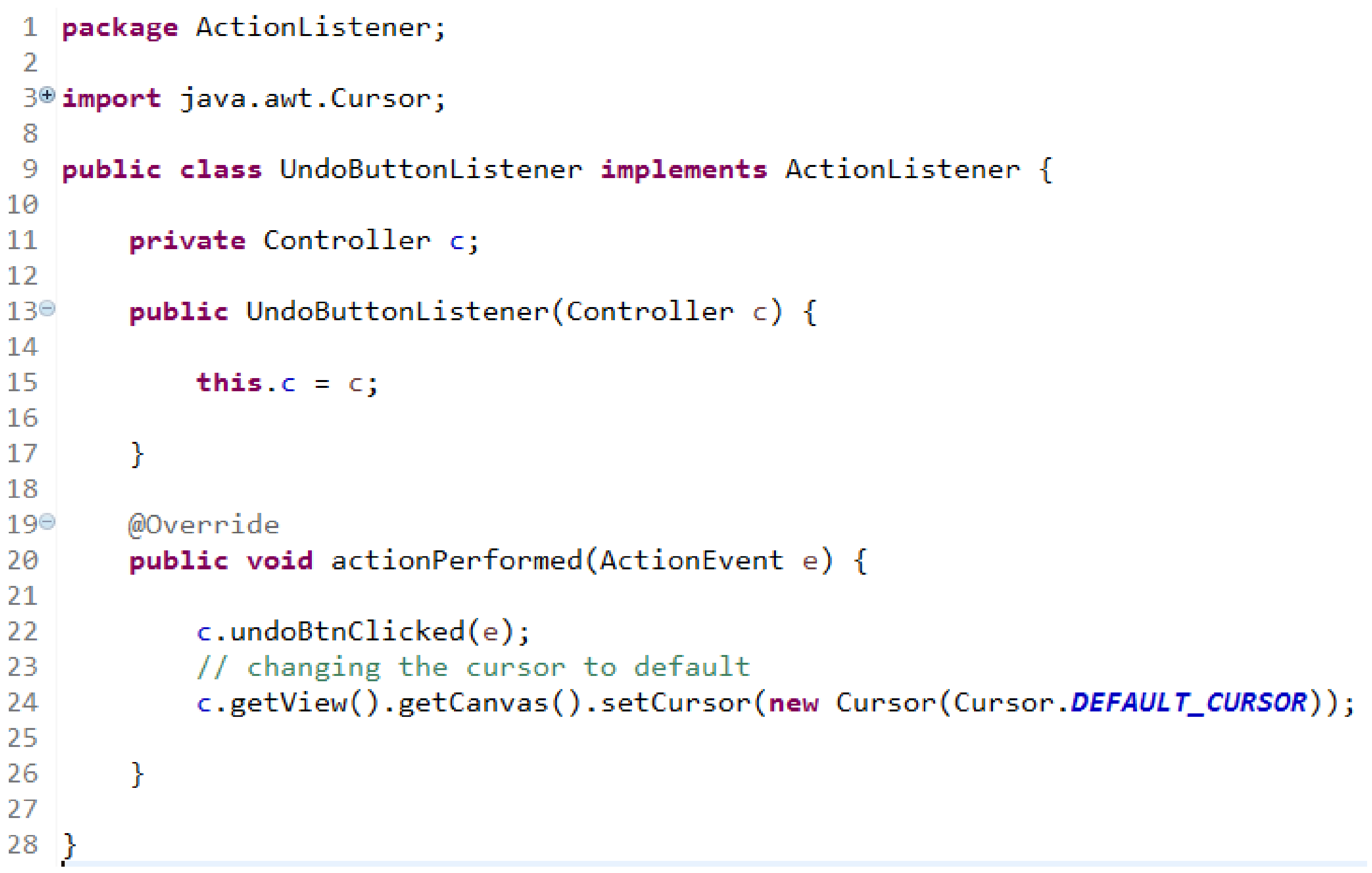
```java
package ActionListener;

import java.awt.event.ActionEvent;

public class RedStateButtonListener implements ActionListener {

    private Controller c;

    public RedStateButtonListener(Controller c) {
        this.c = c;
    }

    public void actionPerformed(ActionEvent e) {
        c.redStateBtnClicked(e, c);
    }
}
```

TransButtonListener Class：



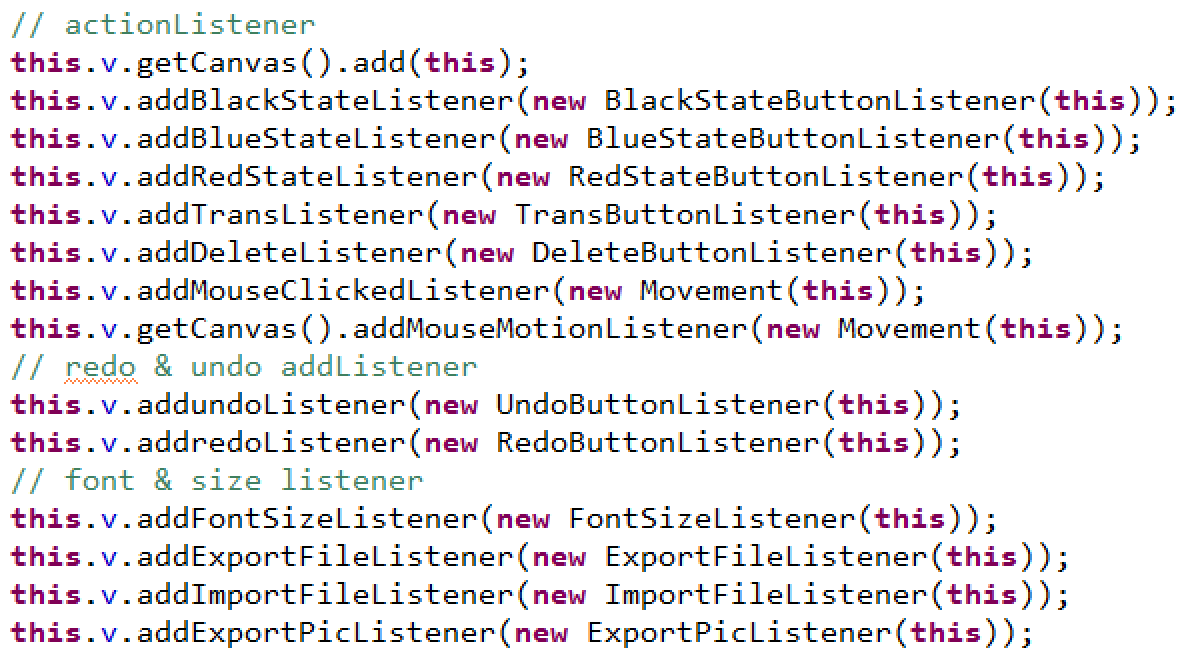
```java
package ActionListener;

import java.awt.event.ActionEvent;

public class TransButtonListener implements ActionListener {

    private Controller c;

    public TransButtonListener(Controller c) {

        this.c = c;

    }

    public void actionPerformed(ActionEvent e) {

        c.transBtnClicked(e, c);

    }

}
```

UndoButtonListener Class：



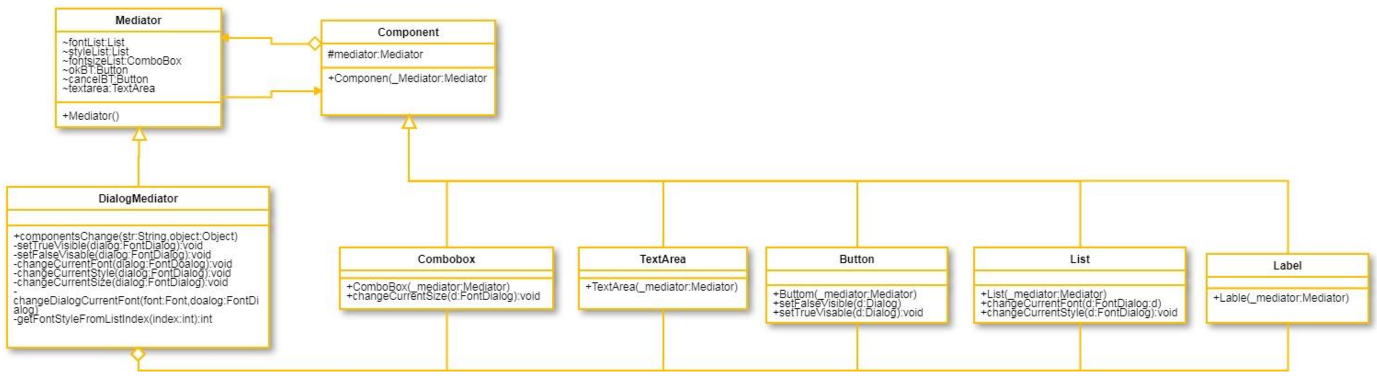
```java
package ActionListener;

import java.awt.Cursor;

public class UndoButtonListener implements ActionListener {

    private Controller c;

    public UndoButtonListener(Controller c) {

        this.c = c;

    }

    @Override
    public void actionPerformed(ActionEvent e) {

        c.undoBtnClicked(e);
        // changing the cursor to default
        c.getView().getCanvas().setCursor(new Cursor(Cursor.DEFAULT_CURSOR));

    }

}
```

Controller Class：



```java
// actionListener
this.v.getCanvas().add(this);
this.v.addBlackStateListener(new BlackStateButtonListener(this));
this.v.addBlueStateListener(new BlueStateButtonListener(this));
this.v.addRedStateListener(new RedStateButtonListener(this));
this.v.addTransListener(new TransButtonListener(this));
this.v.addDeleteListener(new DeleteButtonListener(this));
this.v.addMouseClickedListener(new Movement(this));
this.v.getCanvas().addMouseMotionListener(new Movement(this));
// redo & undo addListener
this.v.addundoListener(new UndoButtonListener(this));
this.v.addredoListener(new RedoButtonListener(this));
// font & size listener
this.v.addFontSizeListener(new FontSizeListener(this));
this.v.addExportFileListener(new ExportFileListener(this));
this.v.addImportFileListener(new ImportFileListener(this));
this.v.addExportPicListener(new ExportPicListener(this));
```

- Mediator Pattern


Mediator
~fontList:List
~styleList:List
~fontsizeList:ComboBox
~okBT:Button
~cancelBT:Button
~textarea:TextArea
+Mediator()
Component
#mediator:Mediator
+Componen(_Mediator:Mediator
DialogMediator
+componentsChange(str:String,object:Object)
-setTrueVisible(dialog:FontDialog):void
-setFalseVisable(dialog:FontDialog):void
-changeCurrentFont(dialog:FontDoalog):void
-changeCurrentStyle(dialog:FontDialog):void
-changeCurrentSize(dialog:FontDialog):void
-changeDialogCurrentFont(font:Font,doalog:FontDialog)
-getFontStyleFromListIndex(index:int):int
Combobox
+ComboBox(_mediator:Mediator)
+changeCurrentSize(d:FontDialog):void
TextArea
+TextArea(_mediator:Mediator)
Button
+Buttom(_mediator:Mediator)
+setFalseVisible(d:Dialog)
+setTrueVisable(d:Dialog):void
List
+List(_mediator:Mediator)
+changeCurrentFont(d:FontDialog:d)
+changeCurrentStyle(d:FontDialog):void
Label
+Lable(_mediator:Mediator)

Text Book Name:	**In our code Name:**
Abstract Mediator	Mediator
Concrete Mediator	DialogMediator
Abstract Colleague	Component
Concrete Colleague	Button、ComboBox、Label、List、TextArea
Clint	FontDialog

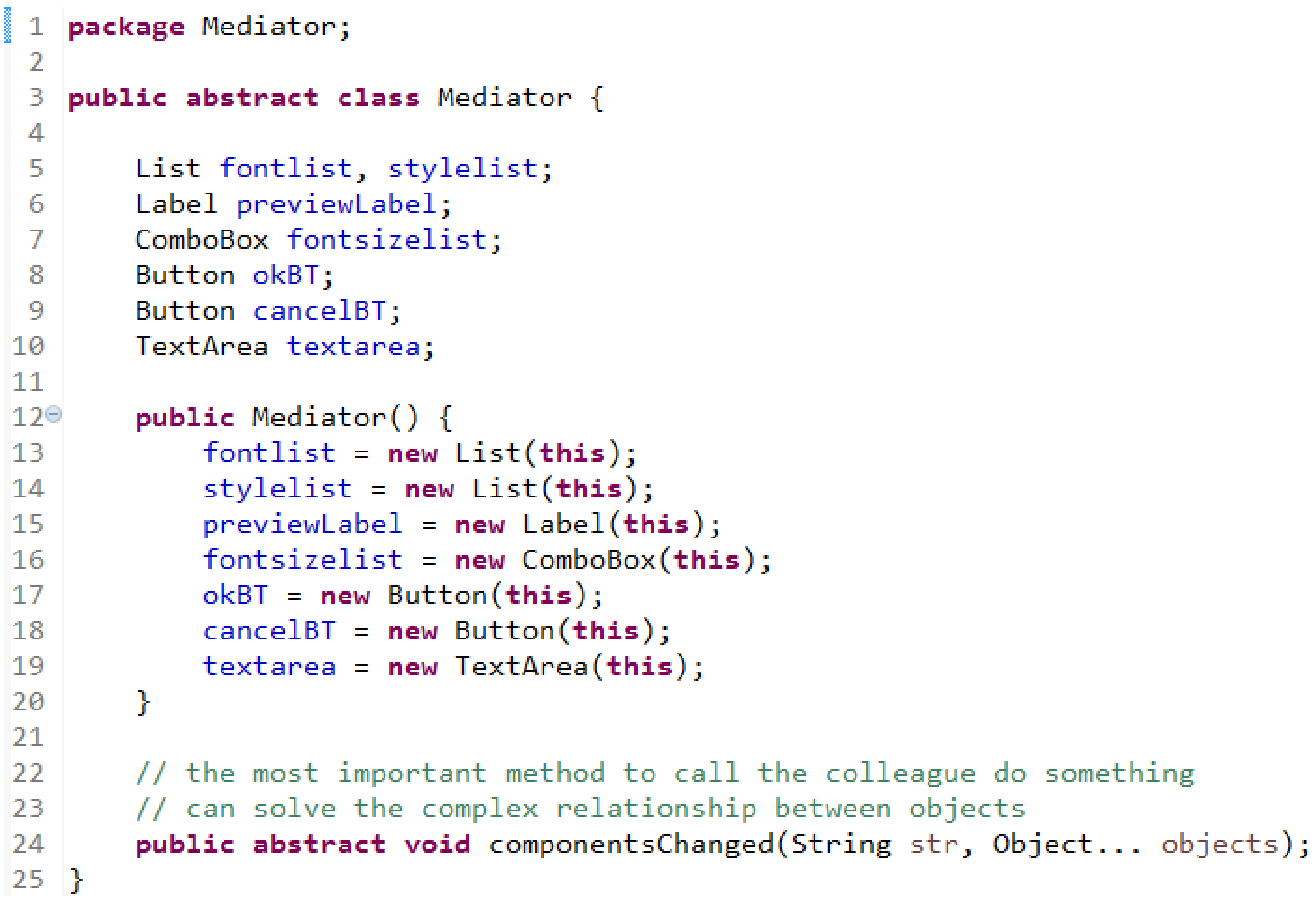
Using the mediator can decoupling the highly complex structure of the Colluegues[Button、ComboBox、Label、List、TextArea]. In our design when we Select the Font / Style / Size, it will trigger other component. If without the Mediator Patter, it will make the high coupling / hard to reuse / hard to expend the system, so we use Mediator Pattern to solves this kind of problems.

Set up the font will use many setting like : styles, size...etc, because when apply these settings, it will affect the content what presented by previewLabel, it has high interactivity, so we use Mediator to coordinate setting and reduce coupling.

DialoMediator coordinate with component class. When the component class want to call or pass the parameter to another component. It must call DialoMediator or pass the parameter to DialoMediator. Then DialoMediator will call or pass the parameter to another component.

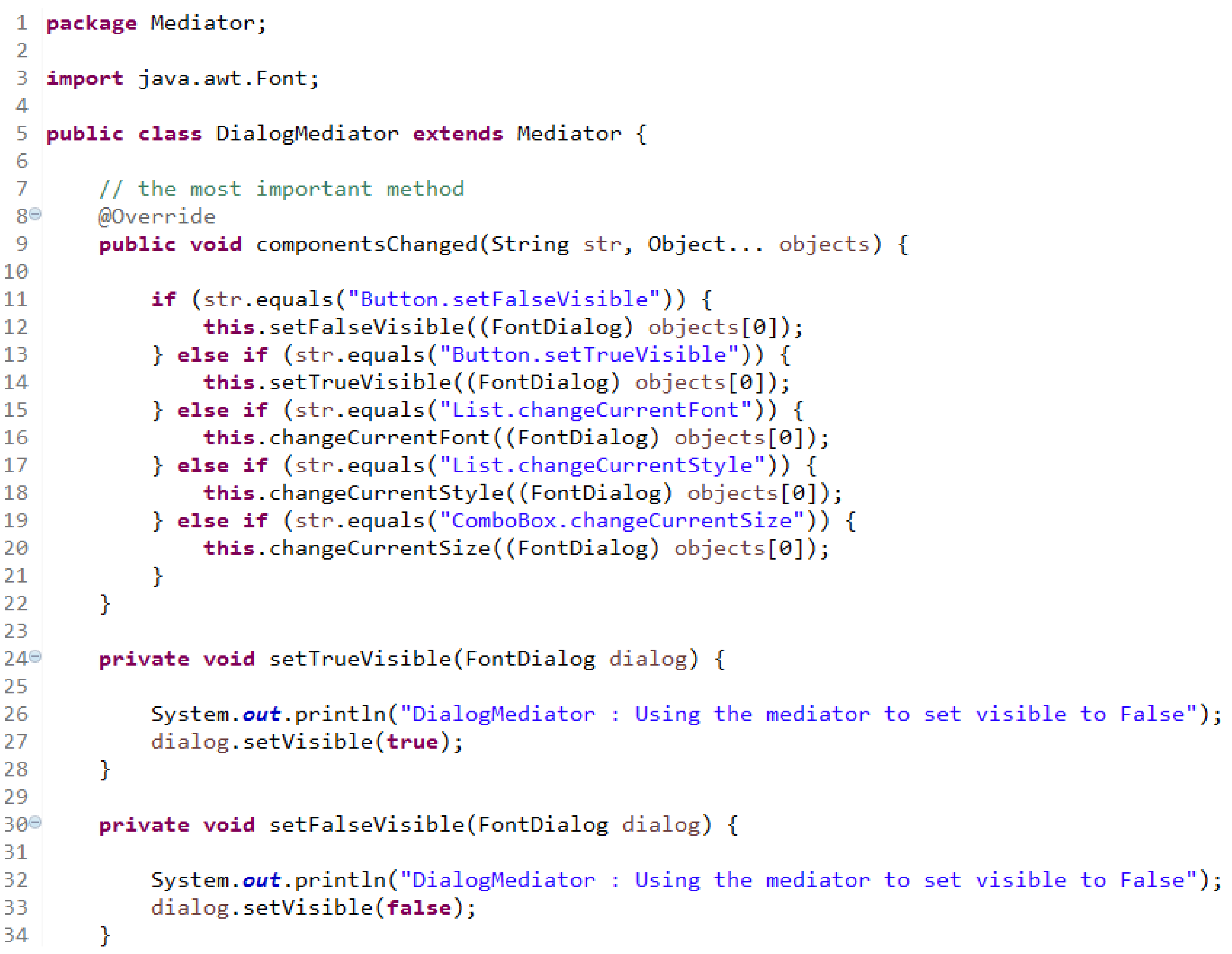
Coding：

Mediator Class：

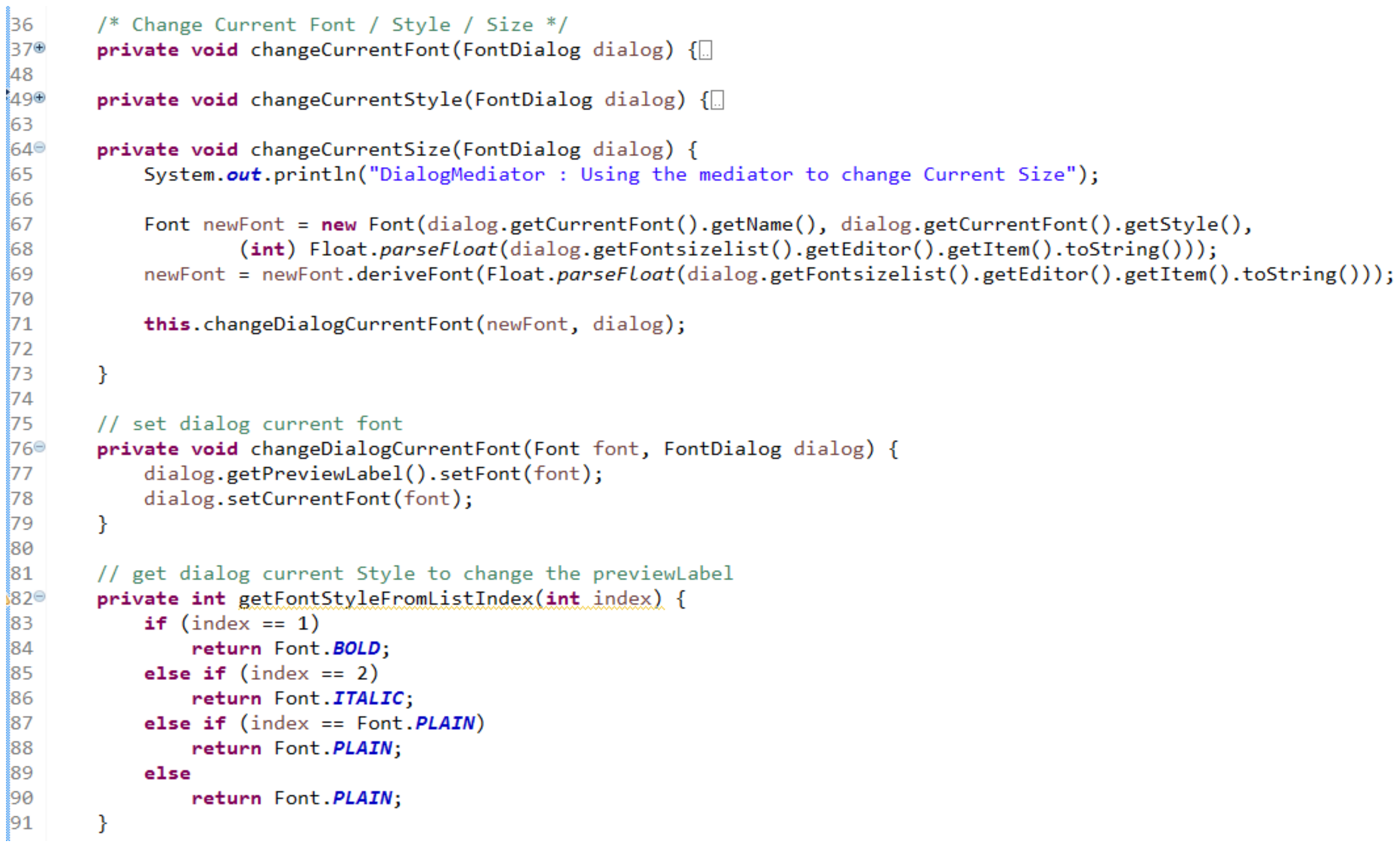


```java
package Mediator;

public abstract class Mediator {

    List fontlist, stylelist;
    Label previewLabel;
    ComboBox fontsizelist;
    Button okBT;
    Button cancelBT;
    TextArea textarea;

    public Mediator() {
        fontlist = new List(this);
        stylelist = new List(this);
        previewLabel = new Label(this);
        fontsizelist = new ComboBox(this);
        okBT = new Button(this);
        cancelBT = new Button(this);
        textarea = new TextArea(this);
    }

    // the most important method to call the colleague do something
    // can solve the complex relationship between objects
    public abstract void componentsChanged(String str, Object... objects);
}
```

DialogMediator Class：



```java
package Mediator;

import java.awt.Font;

public class DialogMediator extends Mediator {

    // the most important method
    @Override
    public void componentsChanged(String str, Object... objects) {

        if (str.equals("Button.setFalseVisible")) {
            this.setFalseVisible((FontDialog) objects[0]);
        } else if (str.equals("Button.setTrueVisible")) {
            this.setTrueVisible((FontDialog) objects[0]);
        } else if (str.equals("List.changeCurrentFont")) {
            this.changeCurrentFont((FontDialog) objects[0]);
        } else if (str.equals("List.changeCurrentStyle")) {
            this.changeCurrentStyle((FontDialog) objects[0]);
        } else if (str.equals("ComboBox.changeCurrentSize")) {
            this.changeCurrentSize((FontDialog) objects[0]);
        }
    }

    private void setTrueVisible(FontDialog dialog) {

        System.out.println("DialogMediator : Using the mediator to set visible to False");
        dialog.setVisible(true);
    }

    private void setFalseVisible(FontDialog dialog) {

        System.out.println("DialogMediator : Using the mediator to set visible to False");
        dialog.setVisible(false);
    }
```



```java
    /* Change Current Font / Style / Size */
    private void changeCurrentFont(FontDialog dialog) {

    private void changeCurrentStyle(FontDialog dialog) {

    private void changeCurrentSize(FontDialog dialog) {
        System.out.println("DialogMediator : Using the mediator to change Current Size");

        Font newFont = new Font(dialog.getCurrentFont().getName(), dialog.getCurrentFont().getStyle(),
                (int) Float.parseFloat(dialog.getFontsizelist().getEditor().getItem().toString()));
        newFont = newFont.deriveFont(Float.parseFloat(dialog.getFontsizelist().getEditor().getItem().toString()));

        this.changeDialogCurrentFont(newFont, dialog);

    }

    // set dialog current font
    private void changeDialogCurrentFont(Font font, FontDialog dialog) {
        dialog.getPreviewLabel().setFont(font);
        dialog.setCurrentFont(font);
    }

    // get dialog current Style to change the previewLabel
    private int getFontStyleFromListIndex(int index) {
        if (index == 1)
            return Font.BOLD;
        else if (index == 2)
            return Font.ITALIC;
        else if (index == Font.PLAIN)
            return Font.PLAIN;
        else
            return Font.PLAIN;
    }
```

Component Class：

```java
package Mediator;

//abstract Colleague
public abstract class Component {
    protected Mediator mediator;

    public Component(Mediator _mediator) {
        this.mediator = _mediator;
    }
}
```

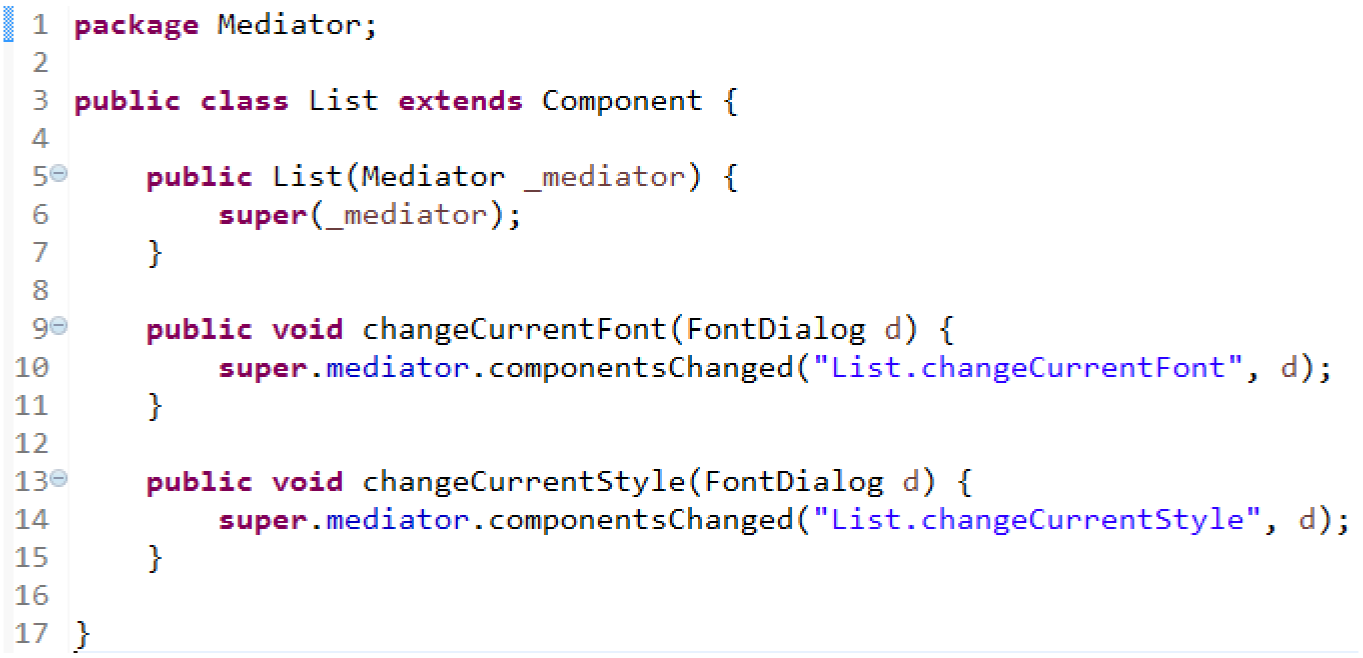
Button Class：

```java
package Mediator;

import java.awt.Dialog;

public class Button extends Component {
    public Button(Mediator _mediator) {
        super(_mediator);
    }

    // set the dialog Visible to false
    public void setFalseVisible(Dialog d) {
        super.mediator.componentsChanged("Button.setFalseVisible", d);
    }

    // set the dialog Visible to Ture
    public void setTrueVisible(Dialog d) {
        super.mediator.componentsChanged("Button.setTrueVisible", d);
    }
}
```

Label Class：

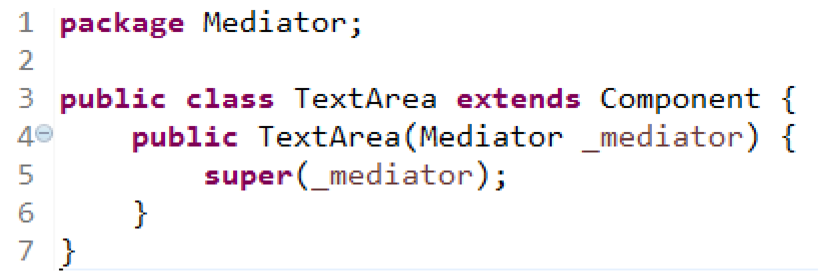
```java
package Mediator;

public class Label extends Component {

    public Label(Mediator _mediator) {
        super(_mediator);
    }

}
```

List Class：



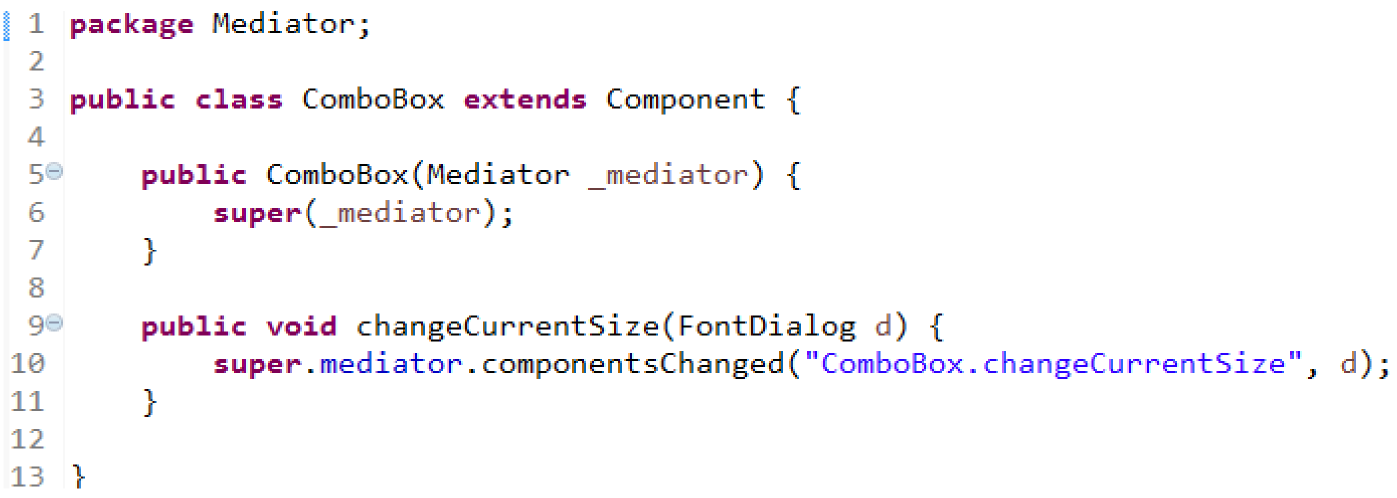
```java
package Mediator;

public class List extends Component {

    public List(Mediator _mediator) {
        super(_mediator);
    }

    public void changeCurrentFont(FontDialog d) {
        super.mediator.componentsChanged("List.changeCurrentFont", d);
    }

    public void changeCurrentStyle(FontDialog d) {
        super.mediator.componentsChanged("List.changeCurrentStyle", d);
    }

}
```

TextArea Class：



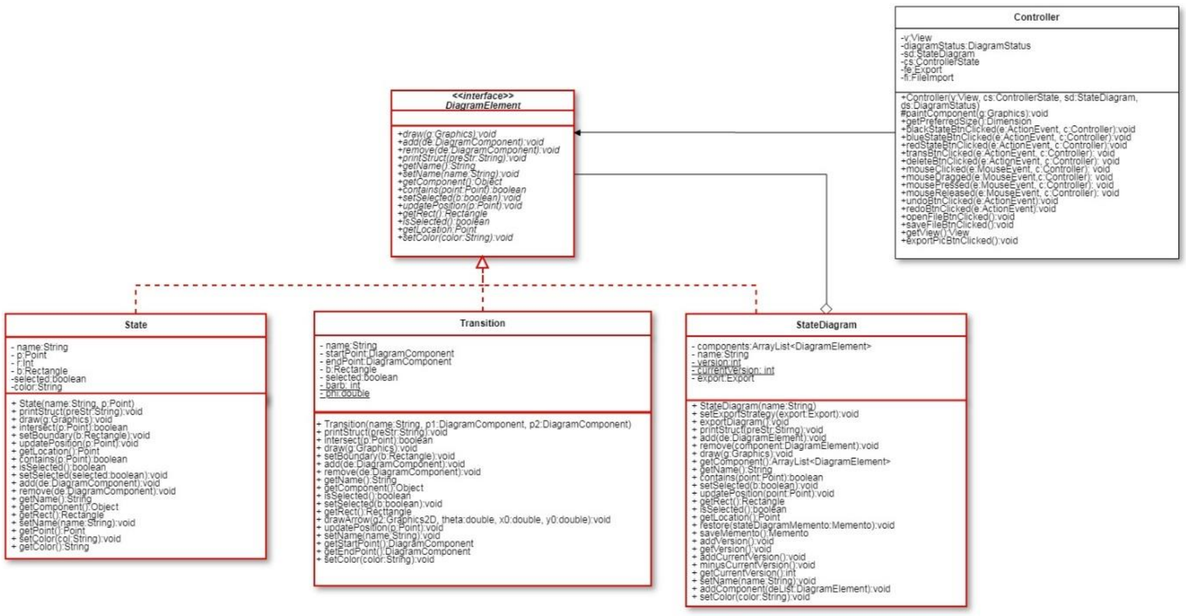
```java
package Mediator;

public class TextArea extends Component {
    public TextArea(Mediator _mediator) {
        super(_mediator);
    }
}
```

ComboBox Class：



```java
package Mediator;

public class ComboBox extends Component {

    public ComboBox(Mediator _mediator) {
        super(_mediator);
    }

    public void changeCurrentSize(FontDialog d) {
        super.mediator.componentsChanged("ComboBox.changeCurrentSize", d);
    }

}
```

- Composite Pattern



Text Book Name:	In our code Name:
Interface Component	DiagramElment
Composite Component	StateDiagram
Primitive Component1	State
Primitive Component2	Transition
Clint	Controller

The user draws small component to consist big component, and combine different component. When we combine more and more component, it will cause structure complex, so we use Composite pattern to build the structure with this situation.

We let State and Transition be a leaf, and DiagramElement will be composite component. Then client can choose StateDiagram directly.If we want to add another object, we just add a class, don’t have to modify the code.

Coding：

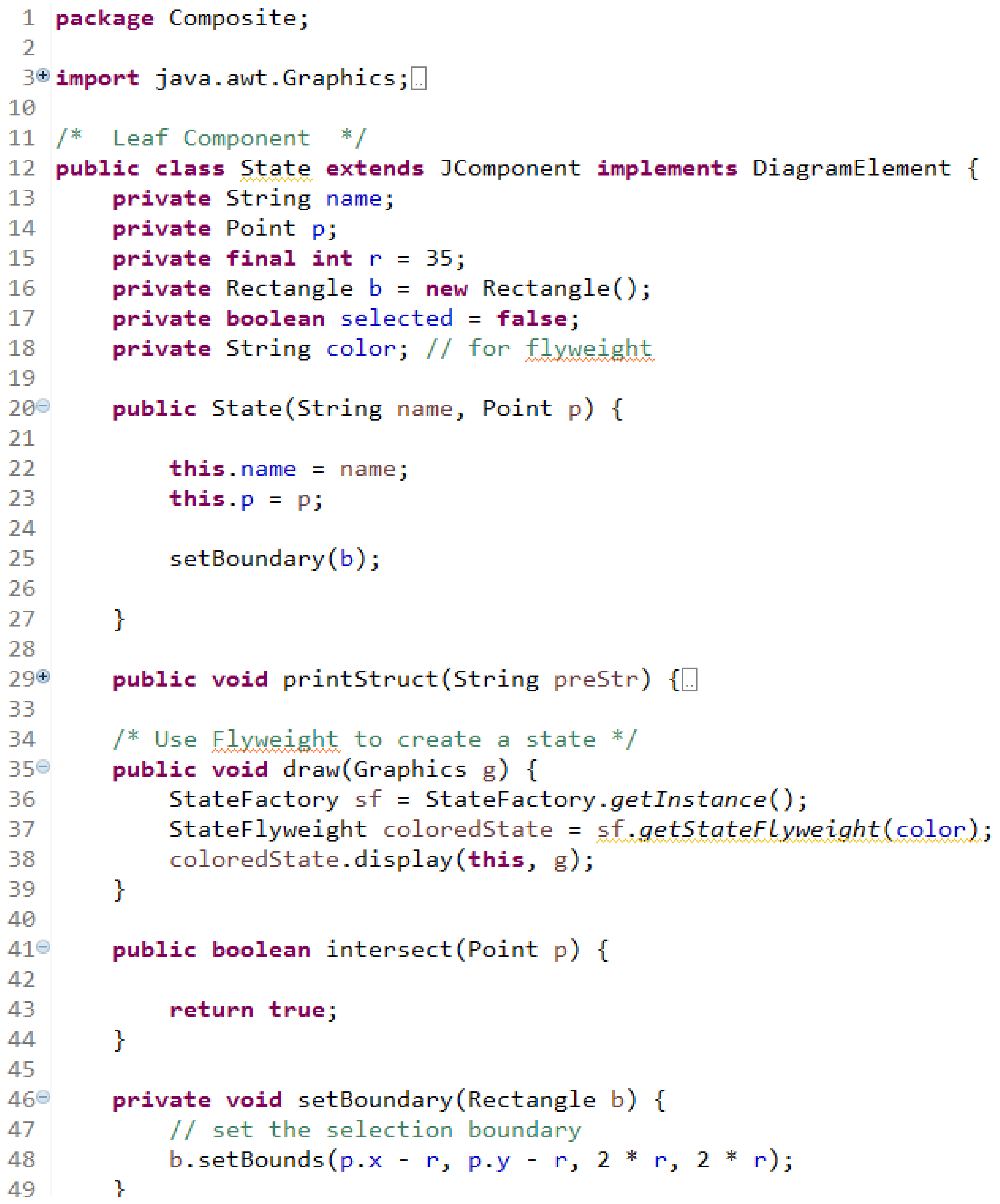
Controller Class：

```
// when this method be called, the draw panel will paint
@Override
protected void paintComponent(Graphics g) {
    super.paintComponent(g);

    System.out.println("Controller : paintComponent() method is working");
    for (DiagramElement d : sd.getComponent()) {
        d.draw(g);
    }
    // print message to check
    sd.printStruct("");
    diagramStatus.printDiagramState();
}
```

DiagramElement Class：

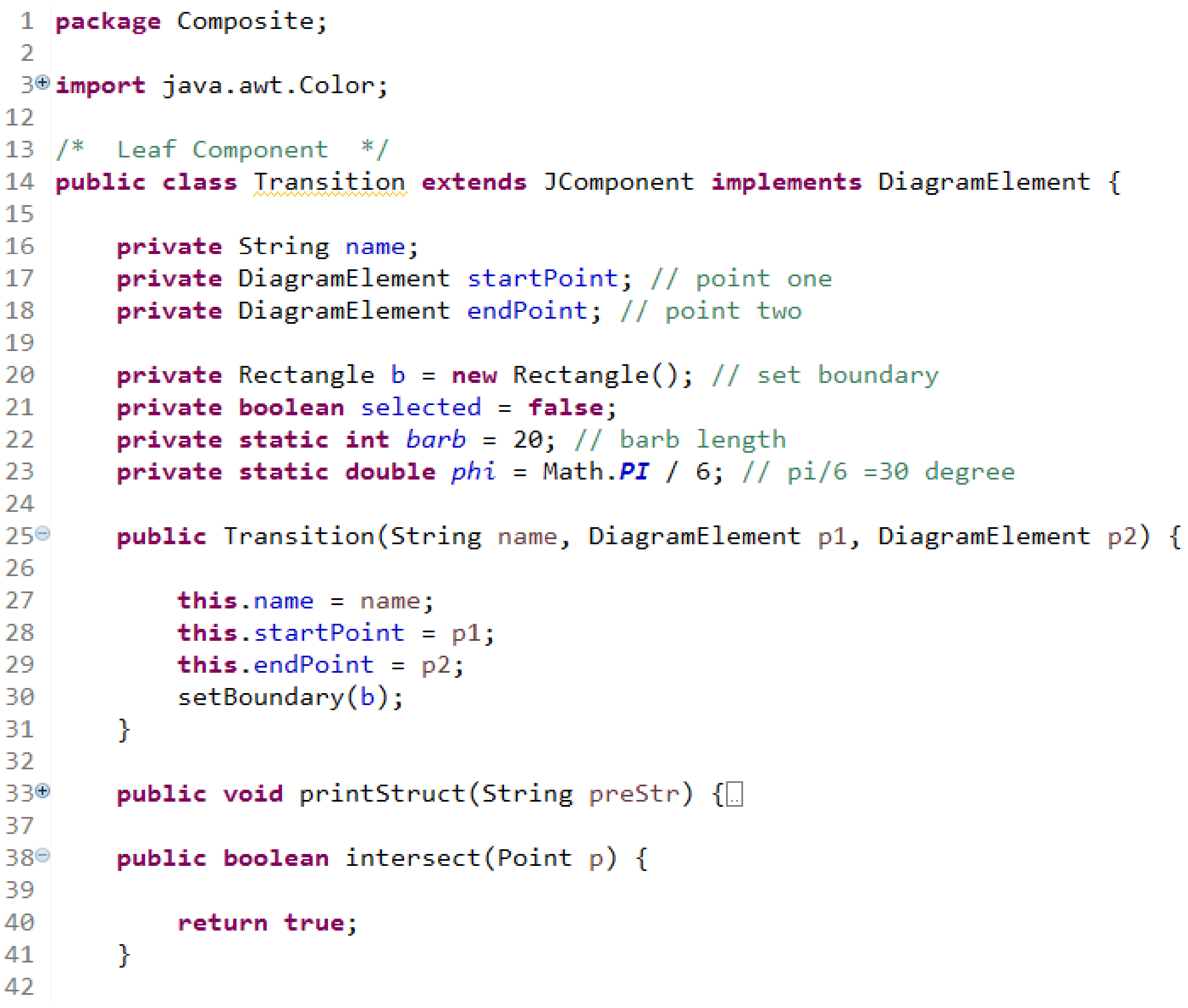
```java
package Composite;

import java.awt.Graphics;

/*  Component  */
public interface DiagramElement {

    public abstract void draw(Graphics g);

    // factory method for create Iterator
    public abstract void add(DiagramElement de);

    public abstract void remove(DiagramElement de);

    public abstract void printStruct(String preStr);

    public abstract String getName();

    public abstract void setName(String name);

    // get object from StateDiagram's ArrayList
    public abstract Object getComponent();

    // this method check the component you clicked
    public abstract boolean contains(Point point);

    public abstract void setSelected(boolean b);

    // update the state's position
    public abstract void updatePosition(Point p);

    public abstract Rectangle getRect();

    public abstract boolean isSelected();

    public abstract Point getLocation();

    public abstract void setColor(String color);

}
```

State Class：

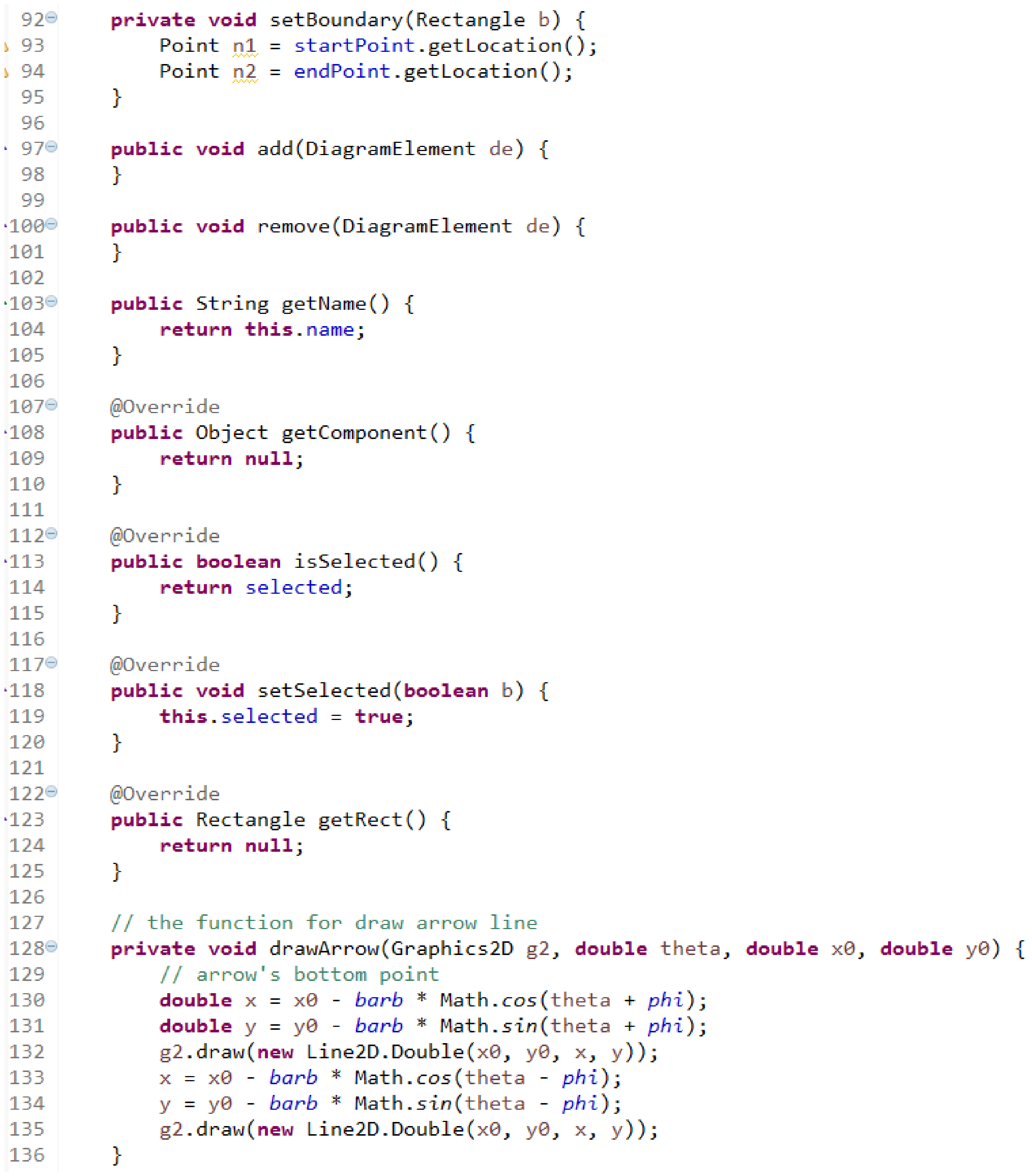


```java
package Composite;

import java.awt.Graphics;

/*  Leaf Component  */
public class State extends JComponent implements DiagramElement {
    private String name;
    private Point p;
    private final int r = 35;
    private Rectangle b = new Rectangle();
    private boolean selected = false;
    private String color; // for flyweight

    public State(String name, Point p) {

        this.name = name;
        this.p = p;

        setBoundary(b);

    }

    public void printStruct(String preStr) {

    /* Use Flyweight to create a state */
    public void draw(Graphics g) {
        StateFactory sf = StateFactory.getInstance();
        StateFlyweight coloredState = sf.getStateFlyweight(color);
        coloredState.display(this, g);
    }

    public boolean intersect(Point p) {

        return true;
    }

    private void setBoundary(Rectangle b) {
        // set the selection boundary
        b.setBounds(p.x - r, p.y - r, 2 * r, 2 * r);
    }
    @Override
    public void updatePosition(Point p) {
        this.p.x = p.x;
        this.p.y = p.y;
        this.setBoundary(this.b);
    }

    public Point getLocation() {
        return p;
    }

    @Override
    public boolean contains(Point p) {
        return b.contains(p);
    }

    /**
     * Return true if this node is selected.
     */
    public boolean isSelected() {
        return selected;
    }

    public void setSelected(boolean selected) {
        this.selected = selected;
    }

    public void add(DiagramElement de) {

    }

    public void remove(DiagramElement de) {

    }

    public String getName() {

        return this.name;
    }

    @Override
    public Object getComponent() {

        return null;
    }

    public Rectangle getRect() {

        return b;
    }

    @Override
    public void setName(String name) {
        this.name = name;

    }

    public Point getPoint() {
        return p;
    }

    public void setColor(String col) {
        this.color = col;
        System.out.println("State : the color is set to " + color);
    }

    public String getColor() {
        System.out.println("State : get the color is " + color);
        return color;
    }
```

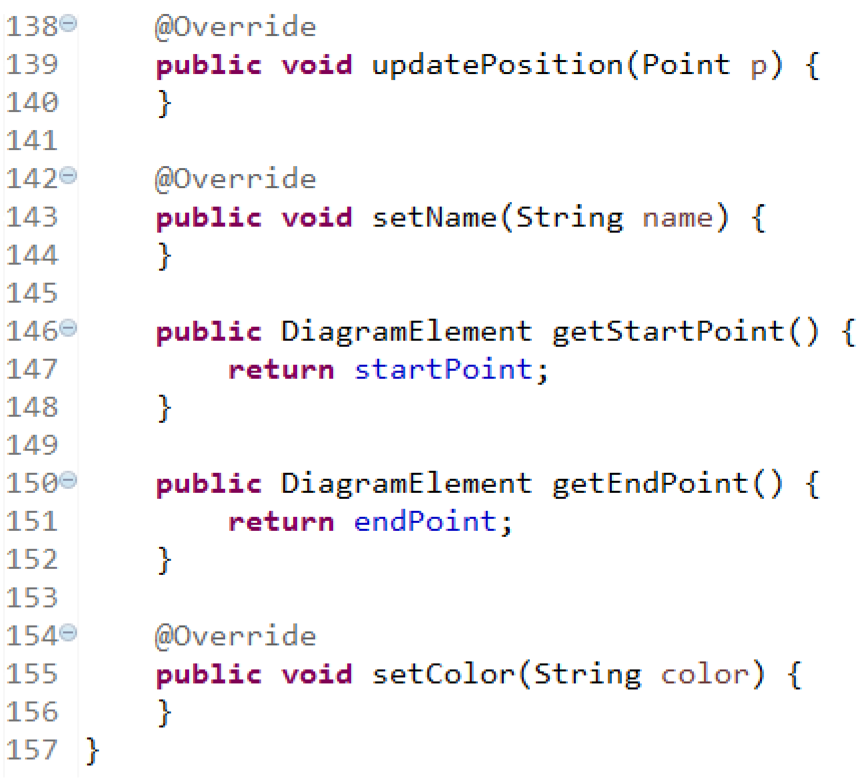
Transition Class：



```java
package Composite;

import java.awt.Color;

/*  Leaf Component  */
public class Transition extends JComponent implements DiagramElement {

    private String name;
    private DiagramElement startPoint; // point one
    private DiagramElement endPoint; // point two

    private Rectangle b = new Rectangle(); // set boundary
    private boolean selected = false;
    private static int barb = 20; // barb length
    private static double phi = Math.PI / 6; // pi/6 =30 degree

    public Transition(String name, DiagramElement p1, DiagramElement p2) {

        this.name = name;
        this.startPoint = p1;
        this.endPoint = p2;
        setBoundary(b);
    }

    public void printStruct(String preStr) {

    public boolean intersect(Point p) {

        return true;
    }

    public void draw(Graphics g) {

        Point n1 = startPoint.getLocation();
        Point n2 = endPoint.getLocation();

        Graphics2D g2 = (Graphics2D) g;
        g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);

        double theta;
        // get the point to calculate the theda
        theta = Math.atan2(n1.y - n2.y, n1.x - n2.x);

        g2.setPaint(Color.blue);

        // when state drag, control the transition
        if (n1.x > n2.x) {

            g2.draw(new Line2D.Double(n2.x + 30, n2.y, n1.x - 30, n1.y));
            drawArrow(g2, theta, n1.x - 30, n1.y);
            // send the theta and a point to draw an arrow
        } else if (n2.x > n1.x) {

            g2.draw(new Line2D.Double(n2.x - 30, n2.y, n1.x + 30, n1.y));
            drawArrow(g2, theta, n1.x + 30, n1.y);

        } else if ((n1.y < n2.y) && (n2.x > n1.x)) {

            g2.draw(new Line2D.Double(n2.x, n2.y - 30, n1.x, n1.y + 30));
            drawArrow(g2, theta, n1.x, n1.y);

        } else if ((n1.y < n2.y) && (n2.x < n1.x)) {
            g2.draw(new Line2D.Double(n2.x, n2.y - 30, n1.x + 30, n1.y + 30));
            drawArrow(g2, theta, n1.x, n1.y);

        }

        int xm = (n1.x + n2.x) / 2;
        int ym = (n1.y + n2.y) / 2;
        g2.drawString(name, xm, ym);// draw the string on middle point

        if (selected) {
            g.setColor(Color.darkGray);
            g.drawRect(b.x, b.y, b.width, b.height);
        }

```

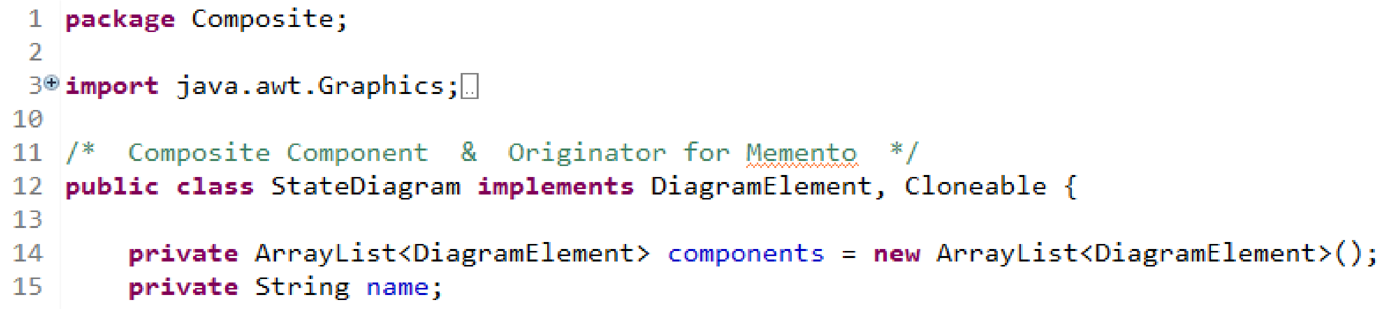


```java
    private void setBoundary(Rectangle b) {
        Point n1 = startPoint.getLocation();
        Point n2 = endPoint.getLocation();
    }

    public void add(DiagramElement de) {
    }

    public void remove(DiagramElement de) {
    }

    public String getName() {
        return this.name;
    }

    @Override
    public Object getComponent() {
        return null;
    }

    @Override
    public boolean isSelected() {
        return selected;
    }

    @Override
    public void setSelected(boolean b) {
        this.selected = true;
    }

    @Override
    public Rectangle getRect() {
        return null;
    }

    // the function for draw arrow line
    private void drawArrow(Graphics2D g2, double theta, double x0, double y0) {
        // arrow's bottom point
        double x = x0 - barb * Math.cos(theta + phi);
        double y = y0 - barb * Math.sin(theta + phi);
        g2.draw(new Line2D.Double(x0, y0, x, y));
        x = x0 - barb * Math.cos(theta - phi);
        y = y0 - barb * Math.sin(theta - phi);
        g2.draw(new Line2D.Double(x0, y0, x, y));
    }
```

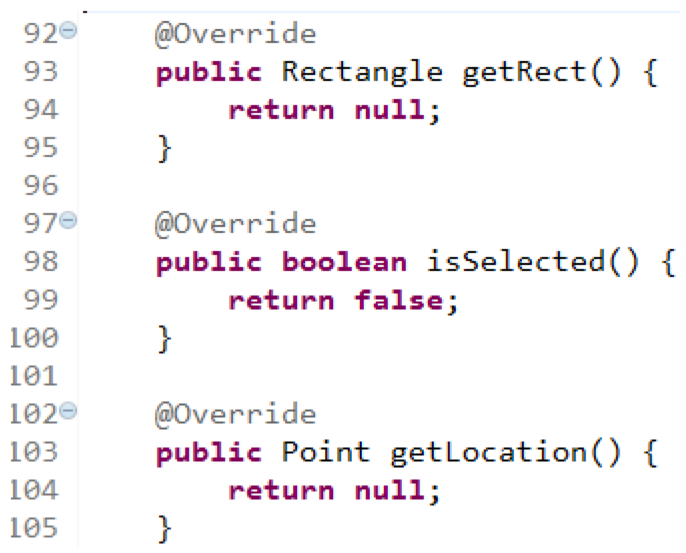


```java
    @Override
    public void updatePosition(Point p) {
    }

    @Override
    public void setName(String name) {
    }

    public DiagramElement getStartPoint() {
        return startPoint;
    }

    public DiagramElement getEndPoint() {
        return endPoint;
    }

    @Override
    public void setColor(String color) {
    }
}
```

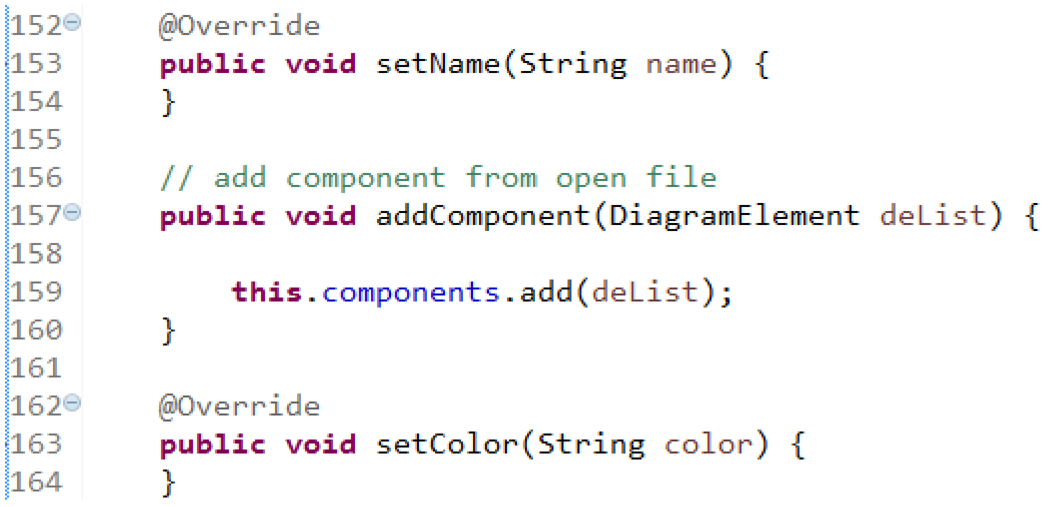
StateDiagram Class：



```java
package Composite;

import java.awt.Graphics;

/*  Composite Component  &  Originator for Memento  */
public class StateDiagram implements DiagramElement, Cloneable {

    private ArrayList<DiagramElement> components = new ArrayList<DiagramElement>();
    private String name;
    public void add(DiagramElement de) {

        this.components.add(de);

    }

    public void remove(DiagramElement component) {
        this.components.remove(component);
    }

    public void draw(Graphics g) {
        for (DiagramElement e : components) {
            e.draw(g);
        }
    }

    public ArrayList<DiagramElement> getComponent() {

        return components;
    }

    // the function which lets StateDiagram set clone Arraylist
    public void setComponent(ArrayList<DiagramElement> deList) {

        this.components = deList;
    }

    public String getName() {

        return this.name;
    }

    @Override
    public boolean contains(Point point) {
        return false;
    }

    @Override
    public void setSelected(boolean b) {
    }

    @Override
    public void updatePosition(Point p) {
    }
```

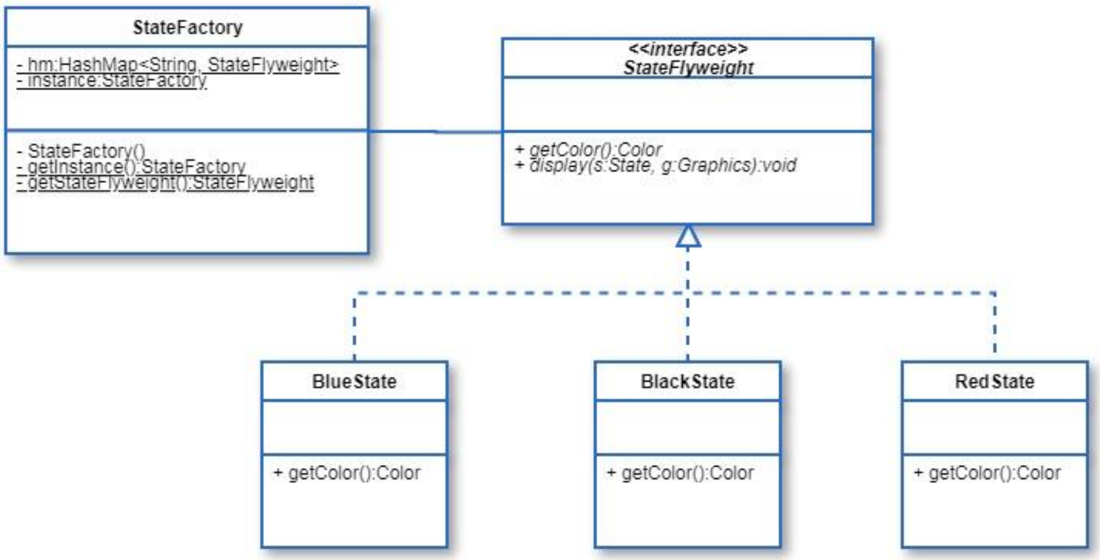


```java
    @Override
    public Rectangle getRect() {
        return null;
    }

    @Override
    public boolean isSelected() {
        return false;
    }

    @Override
    public Point getLocation() {
        return null;
    }
```



```java
    @Override
    public void setName(String name) {
    }

    // add component from open file
    public void addComponent(DiagramElement deList) {

        this.components.add(deList);
    }

    @Override
    public void setColor(String color) {
    }
```

- Flyweight Pattern



Text Book Name:	**In our code Name:**
Flyweight Factory	StateFactory
Interface Flyweight	StateFlyweight
Concrete Flyweight1	BlueState
Concrete Flyweight2	BlackState
Concrete Flyweight3	RedState
Clint	Controller

When the user draw a circle, it may create new object. More circles be created then more objects will come out. If there are too many objects, they may will occupy memory space.

So we use Flyweight pattern in this situation and we can avoid memory blocking.

In this situation, circle is the shared object. FlyweightFactory has black, green and red color. ConcreteFlyweight shows the real color which user use to draw the circle.

Coding：

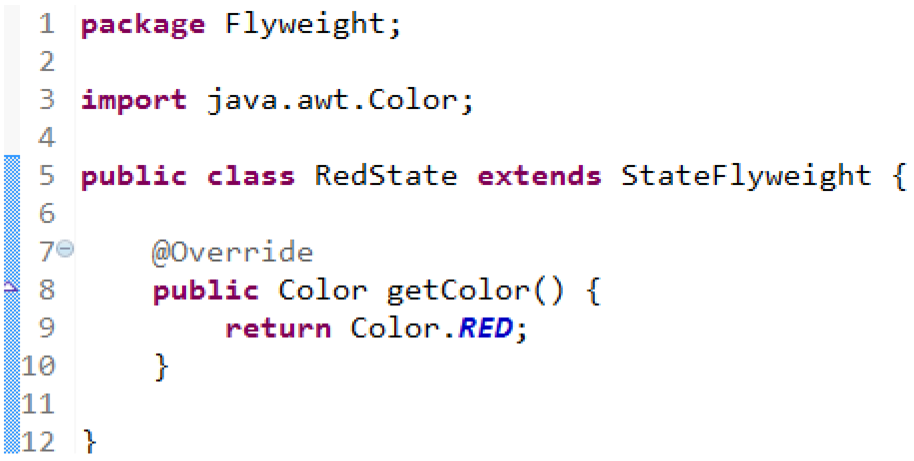
StateFactory Class：

```java
package Flyweight;

import java.util.HashMap;

/* singleton */
public class StateFactory {
    private static HashMap<String, StateFlyweight> hm;
    private static StateFactory instance = new StateFactory();

    private StateFactory() {
        hm = new HashMap<String, StateFlyweight>();
        StateFlyweight blackState, blueState, redState;

        blackState = new BlackState();
        hm.put("black", blackState);
        blueState = new BlueState();
        hm.put("blue", blueState);
        redState = new RedState();
        hm.put("red", redState);
    }

    public static StateFactory getInstance() {
        return instance;
    }

    public static StateFlyweight getStateFlyweight(String key) {
        return hm.get(key);
    }
}
```

StateFlyweight Class：

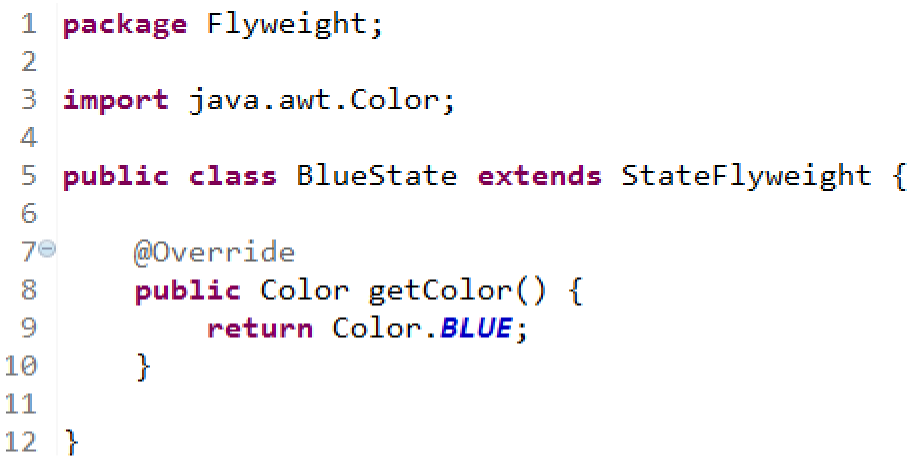
```java
package Flyweight;

import java.awt.Color;

public abstract class StateFlyweight {
    public abstract Color getColor();

    public void display(State s, Graphics g) {
        System.out.println("StateFlyweight color is :" + s.getColor());

        g.setColor(this.getColor());
        g.drawOval(s.getRect().x, s.getRect().y, s.getRect().width, s.getRect().height);
        g.drawString(s.getName(), s.getRect().x + 20, s.getRect().y + 35);

        // if the state be selected, display a selection
        if (s.isSelected()) {
            g.setColor(Color.darkGray);
            g.drawRect(s.getRect().x, s.getRect().y, s.getRect().width, s.getRect().height);
        }

    }
}
```

RedState Class：



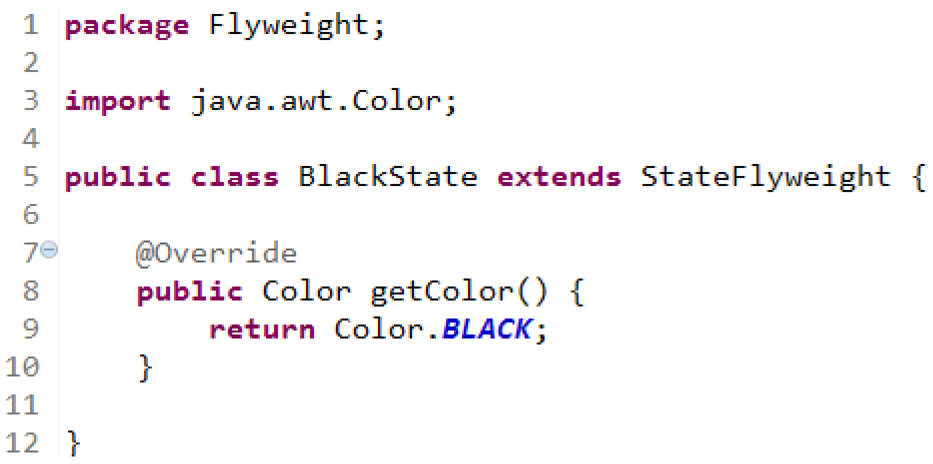
```java
package Flyweight;

import java.awt.Color;

public class RedState extends StateFlyweight {

    @Override
    public Color getColor() {
        return Color.RED;
    }

}
```

BlueState Class：



```java
package Flyweight;

import java.awt.Color;

public class BlueState extends StateFlyweight {

    @Override
    public Color getColor() {
        return Color.BLUE;
    }

}
```

BlackState Class：

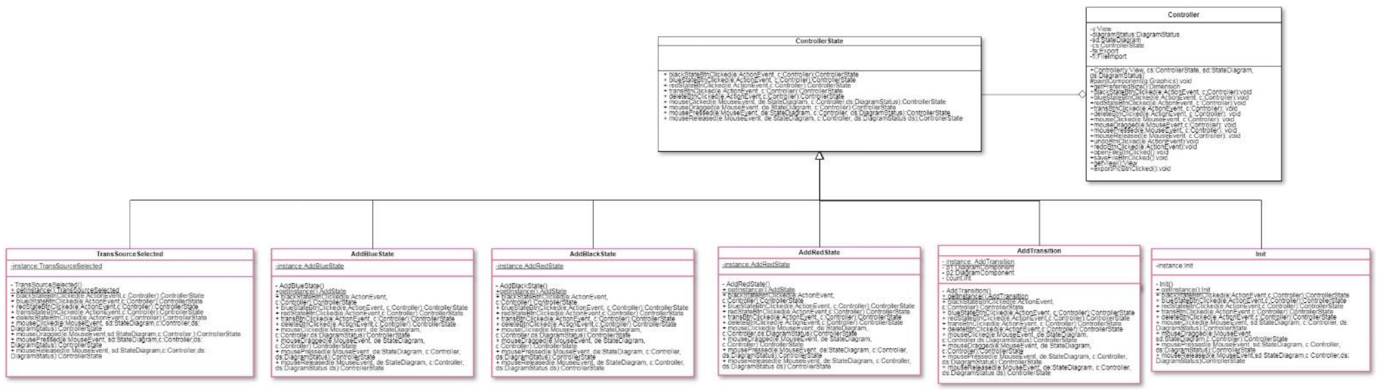


```java
package Flyweight;

import java.awt.Color;

public class BlackState extends StateFlyweight {

    @Override
    public Color getColor() {
        return Color.BLACK;
    }

}
```

State Class：

```
 private String color; // for flyweight


  /* Use Flyweight to create a state */
  public void draw(Graphics g) {
       StateFactory sf = StateFactory.getInstance();
       StateFlyweight coloredState = sf.getStateFlyweight(color);
       coloredState.display(this, g);
  }


 public void setColor(String col) {
      this.color = col;
      System.out.println("State : the color is set to " + color);
 }

 public String getColor() {
      System.out.println("State : get the color is " + color);
      return color;
 }
```

- Singleton Pattern



Text Book Name:	In our code Name:
Singleton	AddBlueState、AddBlackState、AddRedState、AddTransition、TransSourceSelected、Init

State Diagram Editor needs to record the system’s state. There are four kinds of states: AddState, AddTrans, TransSourceSelected, Init. So we choose to use StatePattern to record the state.

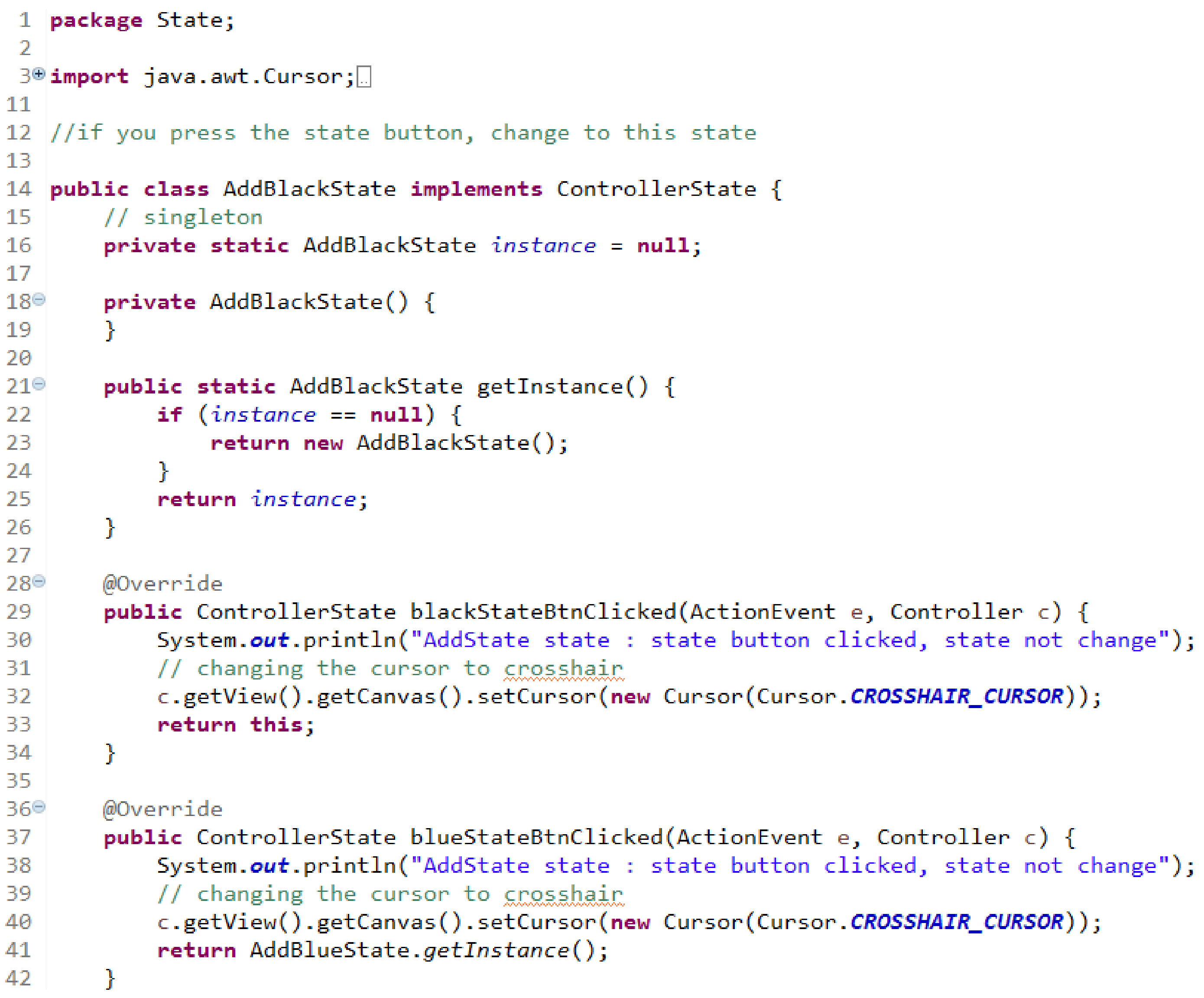
For example, when user start to use Editor, the state is Init. When the user click on BlackState Button, the state will change into AddState. If the user draw a black circle to the panel, state will change into BlackState and turn back to Init.

All the state is independent, so we use Singleton Pattern, this pattern can make sure there is just only one state at the same time.

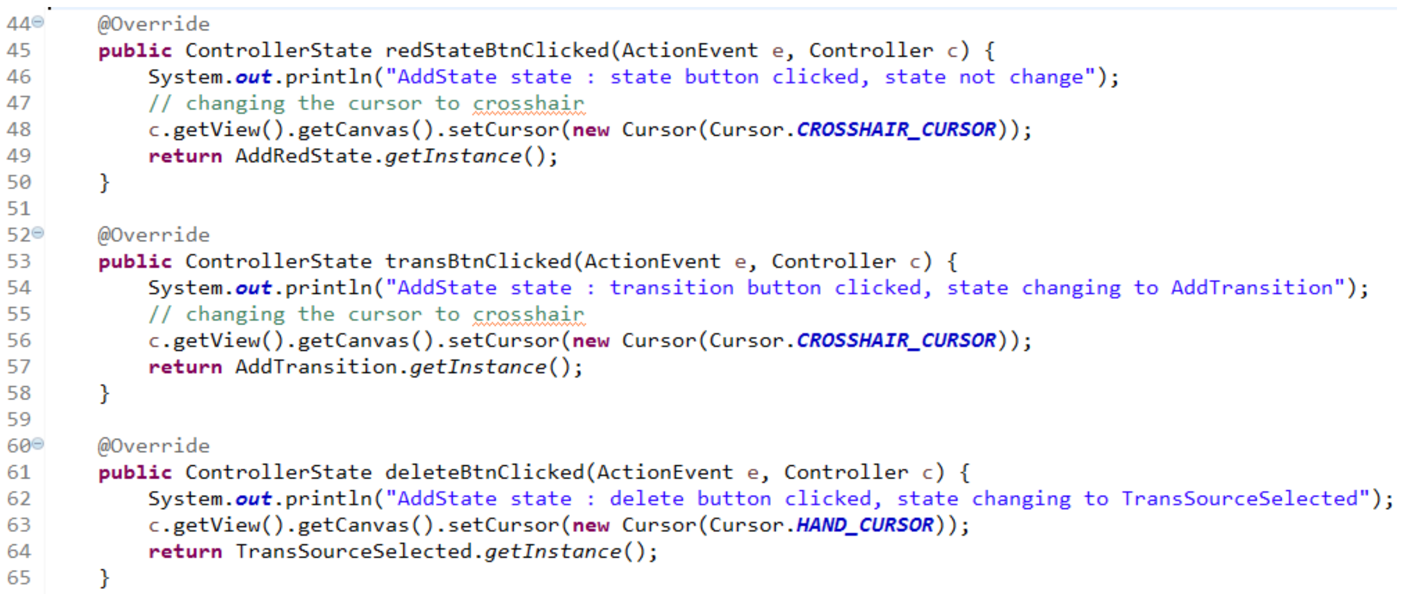
For example, when user start to use Editor, the state is Init. When the user click on BlackState Button, the state will change into AddState. If the user draw a black circle to the panel, state will change into BlackState and turn back to Init. In this example, if system has a lot of same state, it will make the system error. So we set Singleton to every state.

Coding：

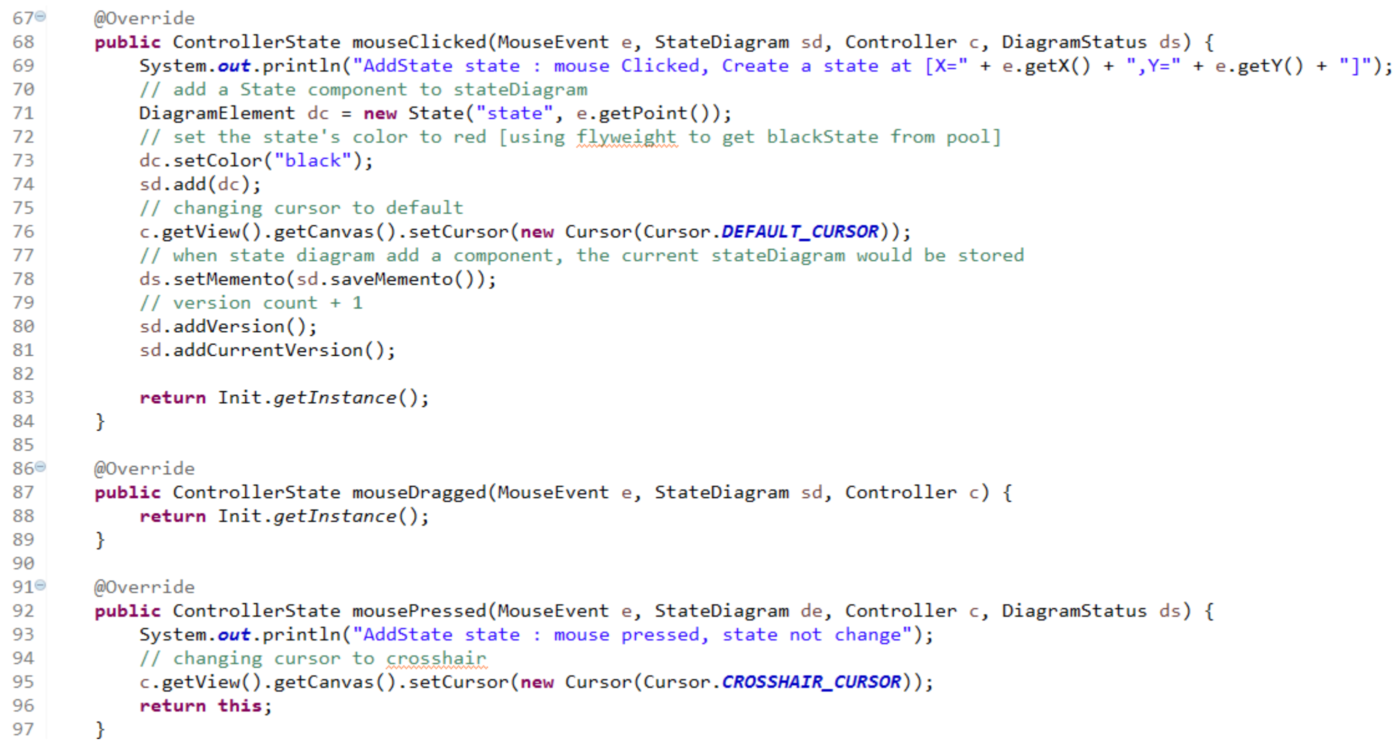
AddBlackState Class：



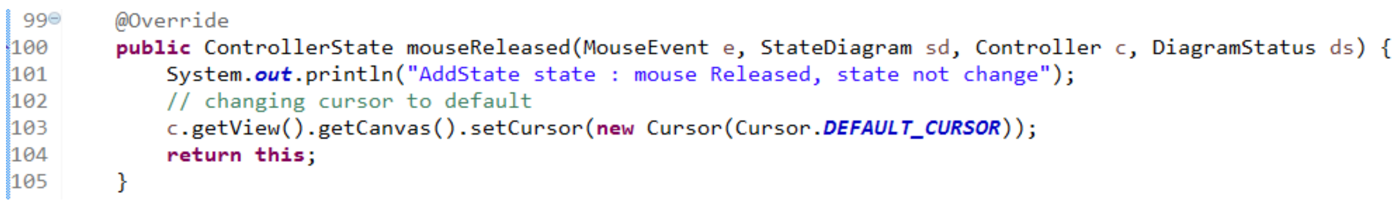
```java
package State;

import java.awt.Cursor;

//if you press the state button, change to this state

public class AddBlackState implements ControllerState {
    // singleton
    private static AddBlackState instance = null;

    private AddBlackState() {
    }

    public static AddBlackState getInstance() {
        if (instance == null) {
            return new AddBlackState();
        }
        return instance;
    }

    @Override
    public ControllerState blackStateBtnClicked(ActionEvent e, Controller c) {
        System.out.println("AddState state : state button clicked, state not change");
        // changing the cursor to crosshair
        c.getView().getCanvas().setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR));
        return this;
    }

    @Override
    public ControllerState blueStateBtnClicked(ActionEvent e, Controller c) {
        System.out.println("AddState state : state button clicked, state not change");
        // changing the cursor to crosshair
        c.getView().getCanvas().setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR));
        return AddBlueState.getInstance();
    }
```



```java
    @Override
    public ControllerState redStateBtnClicked(ActionEvent e, Controller c) {
        System.out.println("AddState state : state button clicked, state not change");
        // changing the cursor to crosshair
        c.getView().getCanvas().setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR));
        return AddRedState.getInstance();
    }

    @Override
    public ControllerState transBtnClicked(ActionEvent e, Controller c) {
        System.out.println("AddState state : transition button clicked, state changing to AddTransition");
        // changing the cursor to crosshair
        c.getView().getCanvas().setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR));
        return AddTransition.getInstance();
    }

    @Override
    public ControllerState deleteBtnClicked(ActionEvent e, Controller c) {
        System.out.println("AddState state : delete button clicked, state changing to TransSourceSelected");
        c.getView().getCanvas().setCursor(new Cursor(Cursor.HAND_CURSOR));
        return TransSourceSelected.getInstance();
    }
```

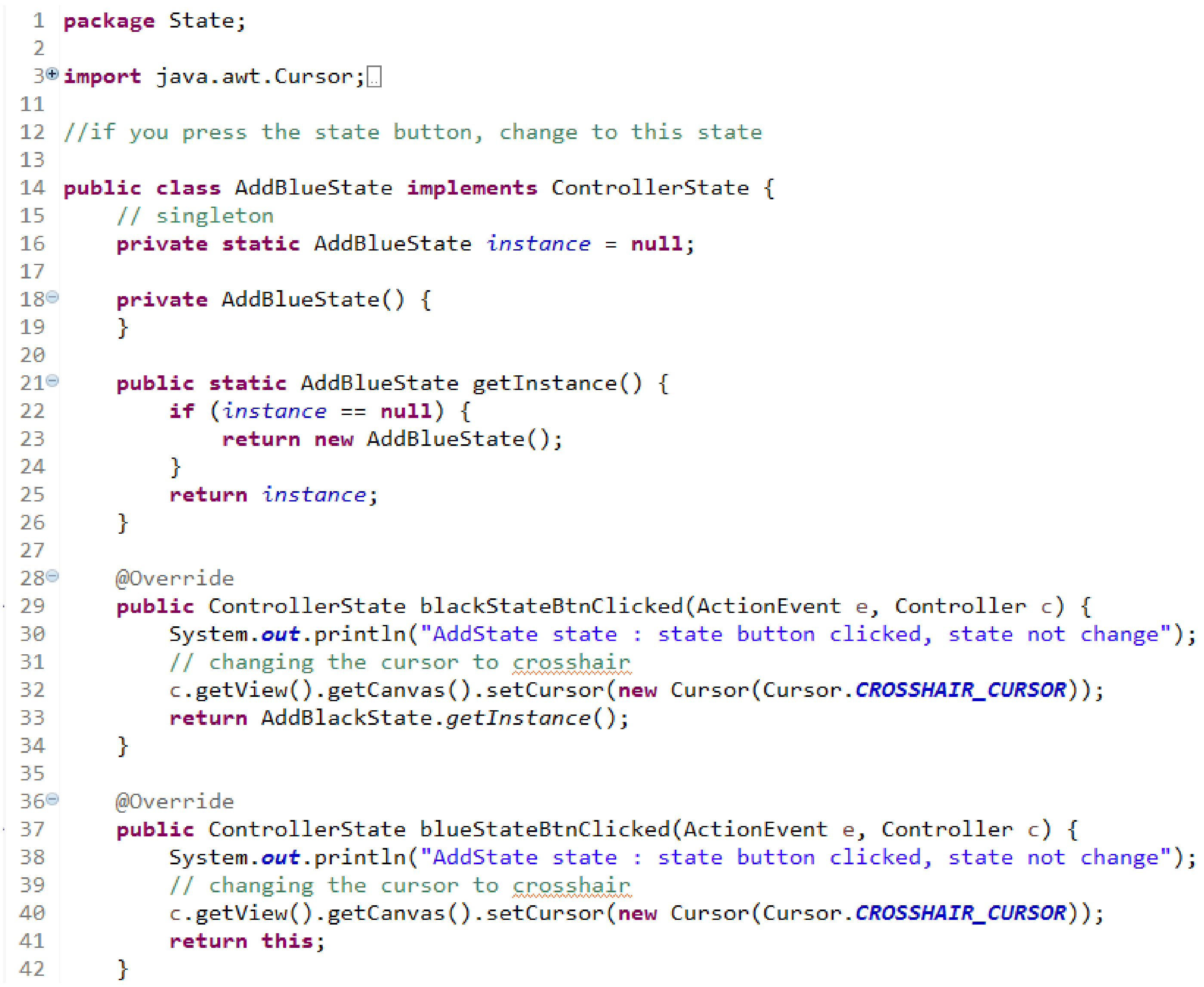


```java
@Override
public ControllerState mouseClicked(MouseEvent e, StateDiagram sd, Controller c, DiagramStatus ds) {
    System.out.println("AddState state : mouse Clicked, Create a state at [X=" + e.getX() + ",Y=" + e.getY() + "]");
    // add a State component to stateDiagram
    DiagramElement dc = new State("state", e.getPoint());
    // set the state's color to red [using flyweight to get blackState from pool]
    dc.setColor("black");
    sd.add(dc);
    // changing cursor to default
    c.getView().getCanvas().setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
    // when state diagram add a component, the current stateDiagram would be stored
    ds.setMemento(sd.saveMemento());
    // version count + 1
    sd.addVersion();
    sd.addCurrentVersion();

    return Init.getInstance();
}

@Override
public ControllerState mouseDragged(MouseEvent e, StateDiagram sd, Controller c) {
    return Init.getInstance();
}

@Override
public ControllerState mousePressed(MouseEvent e, StateDiagram de, Controller c, DiagramStatus ds) {
    System.out.println("AddState state : mouse pressed, state not change");
    // changing cursor to crosshair
    c.getView().getCanvas().setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR));
    return this;
}
```

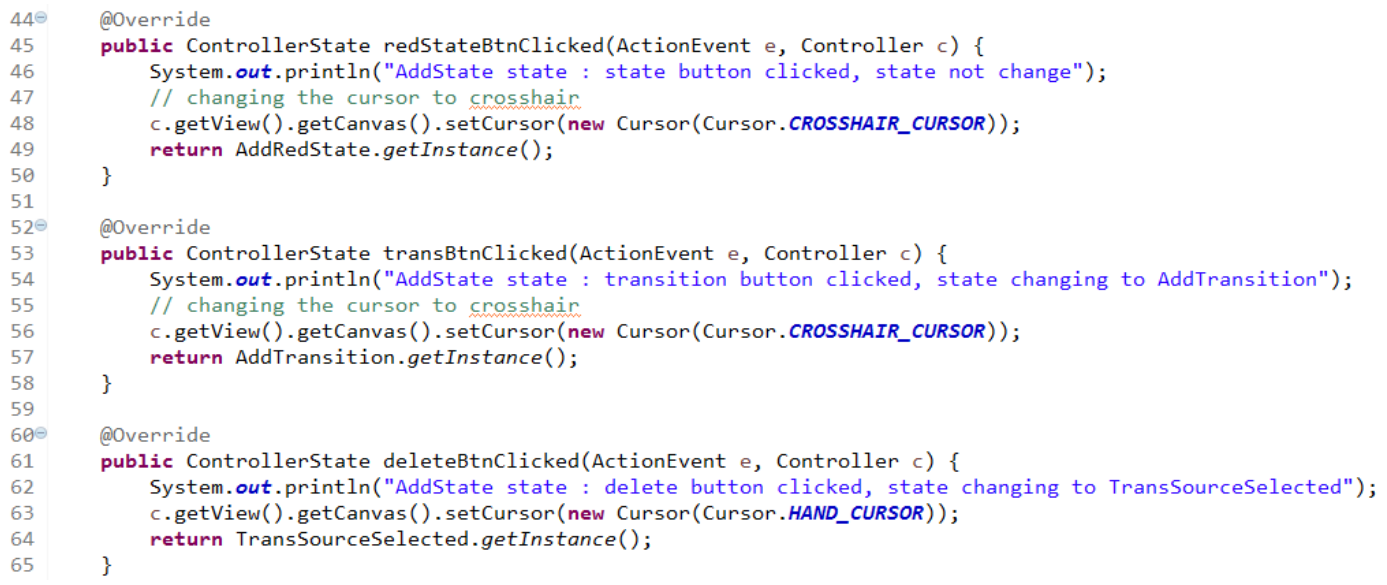


```java
@Override
public ControllerState mouseReleased(MouseEvent e, StateDiagram sd, Controller c, DiagramStatus ds) {
    System.out.println("AddState state : mouse Released, state not change");
    // changing cursor to default
    c.getView().getCanvas().setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
    return this;
}
```

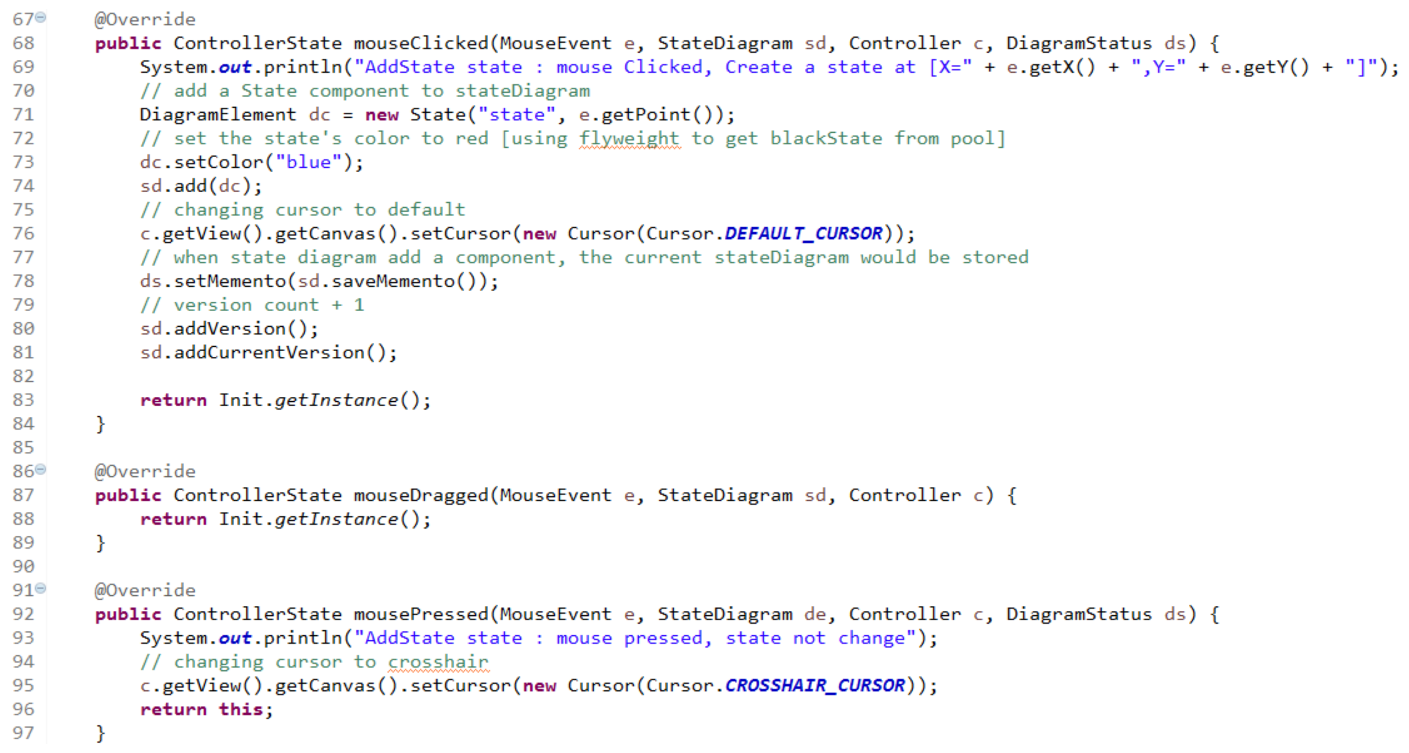
AddBlueState Class：



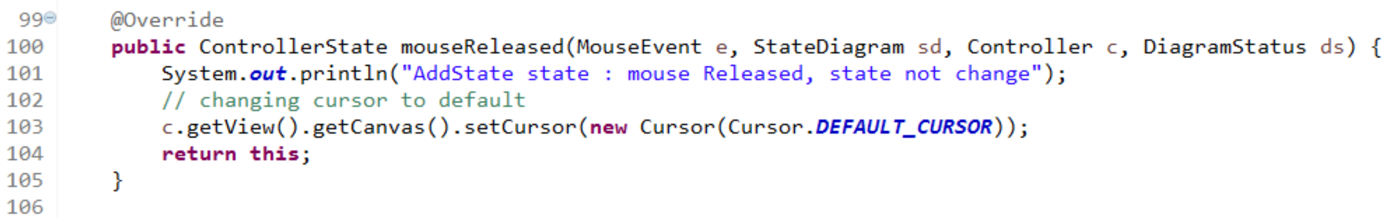
```java
package State;

import java.awt.Cursor;

//if you press the state button, change to this state

public class AddBlueState implements ControllerState {
    // singleton
    private static AddBlueState instance = null;

    private AddBlueState() {
    }

    public static AddBlueState getInstance() {
        if (instance == null) {
            return new AddBlueState();
        }
        return instance;
    }

    @Override
    public ControllerState blackStateBtnClicked(ActionEvent e, Controller c) {
        System.out.println("AddState state : state button clicked, state not change");
        // changing the cursor to crosshair
        c.getView().getCanvas().setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR));
        return AddBlackState.getInstance();
    }

    @Override
    public ControllerState blueStateBtnClicked(ActionEvent e, Controller c) {
        System.out.println("AddState state : state button clicked, state not change");
        // changing the cursor to crosshair
        c.getView().getCanvas().setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR));
        return this;
    }
```



```java
    @Override
    public ControllerState redStateBtnClicked(ActionEvent e, Controller c) {
        System.out.println("AddState state : state button clicked, state not change");
        // changing the cursor to crosshair
        c.getView().getCanvas().setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR));
        return AddRedState.getInstance();
    }

    @Override
    public ControllerState transBtnClicked(ActionEvent e, Controller c) {
        System.out.println("AddState state : transition button clicked, state changing to AddTransition");
        // changing the cursor to crosshair
        c.getView().getCanvas().setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR));
        return AddTransition.getInstance();
    }

    @Override
    public ControllerState deleteBtnClicked(ActionEvent e, Controller c) {
        System.out.println("AddState state : delete button clicked, state changing to TransSourceSelected");
        c.getView().getCanvas().setCursor(new Cursor(Cursor.HAND_CURSOR));
        return TransSourceSelected.getInstance();
    }
```

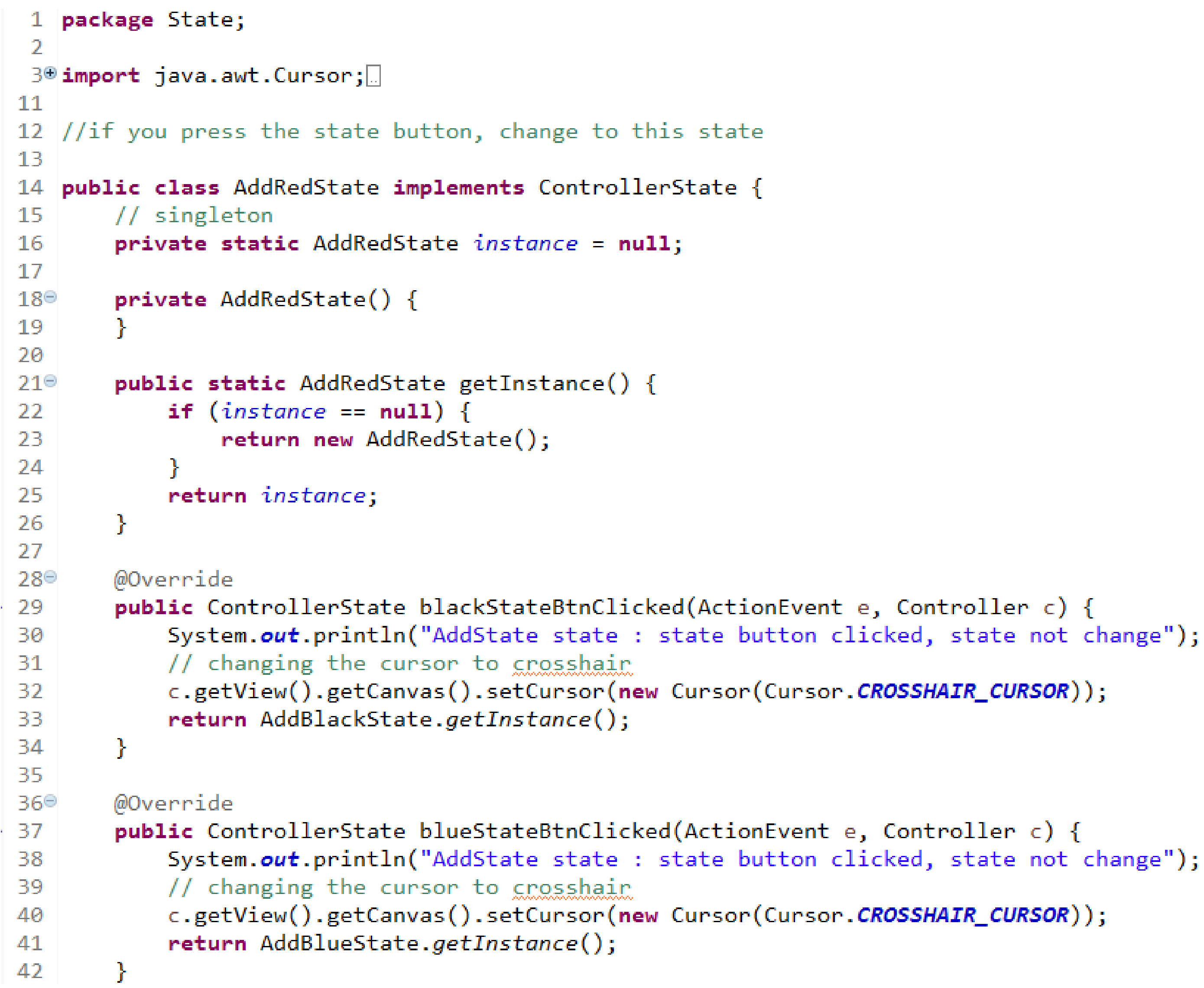


```java
    @Override
    public ControllerState mouseClicked(MouseEvent e, StateDiagram sd, Controller c, DiagramStatus ds) {
        System.out.println("AddState state : mouse Clicked, Create a state at [X=" + e.getX() + ",Y=" + e.getY() + "]");
        // add a State component to stateDiagram
        DiagramElement dc = new State("state", e.getPoint());
        // set the state's color to red [using flyweight to get blackState from pool]
        dc.setColor("blue");
        sd.add(dc);
        // changing cursor to default
        c.getView().getCanvas().setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
        // when state diagram add a component, the current stateDiagram would be stored
        ds.setMemento(sd.saveMemento());
        // version count + 1
        sd.addVersion();
        sd.addCurrentVersion();

        return Init.getInstance();
    }

    @Override
    public ControllerState mouseDragged(MouseEvent e, StateDiagram sd, Controller c) {
        return Init.getInstance();
    }

    @Override
    public ControllerState mousePressed(MouseEvent e, StateDiagram de, Controller c, DiagramStatus ds) {
        System.out.println("AddState state : mouse pressed, state not change");
        // changing cursor to crosshair
        c.getView().getCanvas().setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR));
        return this;
    }
```

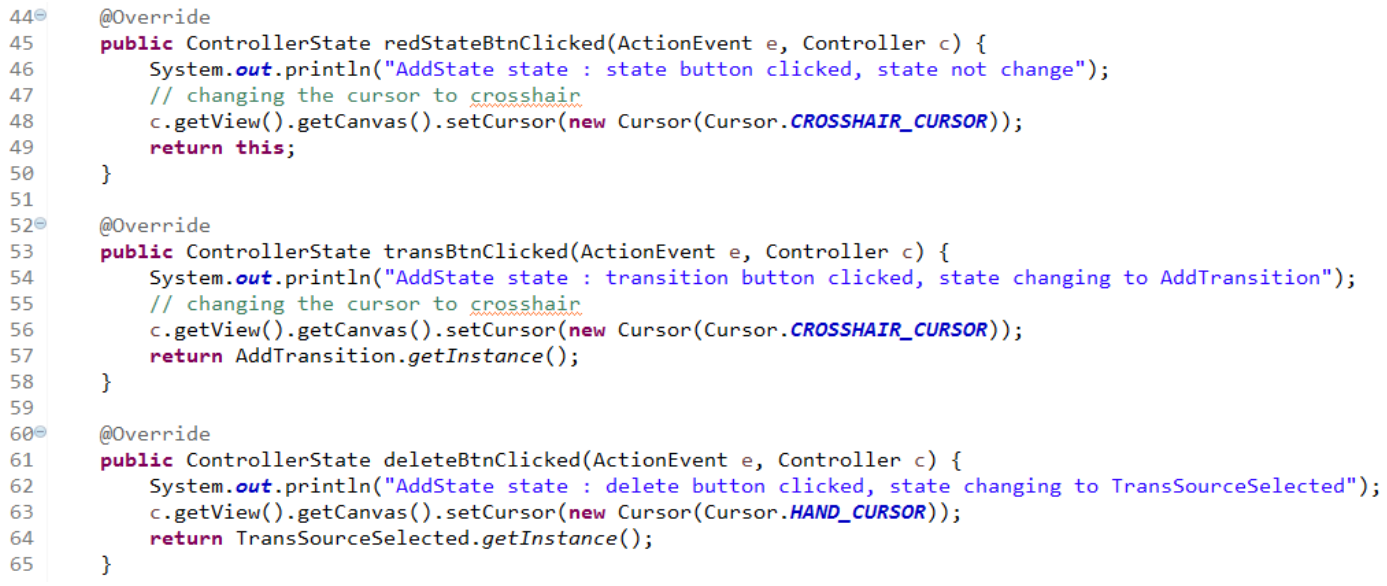


```java
    @Override
    public ControllerState mouseReleased(MouseEvent e, StateDiagram sd, Controller c, DiagramStatus ds) {
        System.out.println("AddState state : mouse Released, state not change");
        // changing cursor to default
        c.getView().getCanvas().setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
        return this;
    }

```

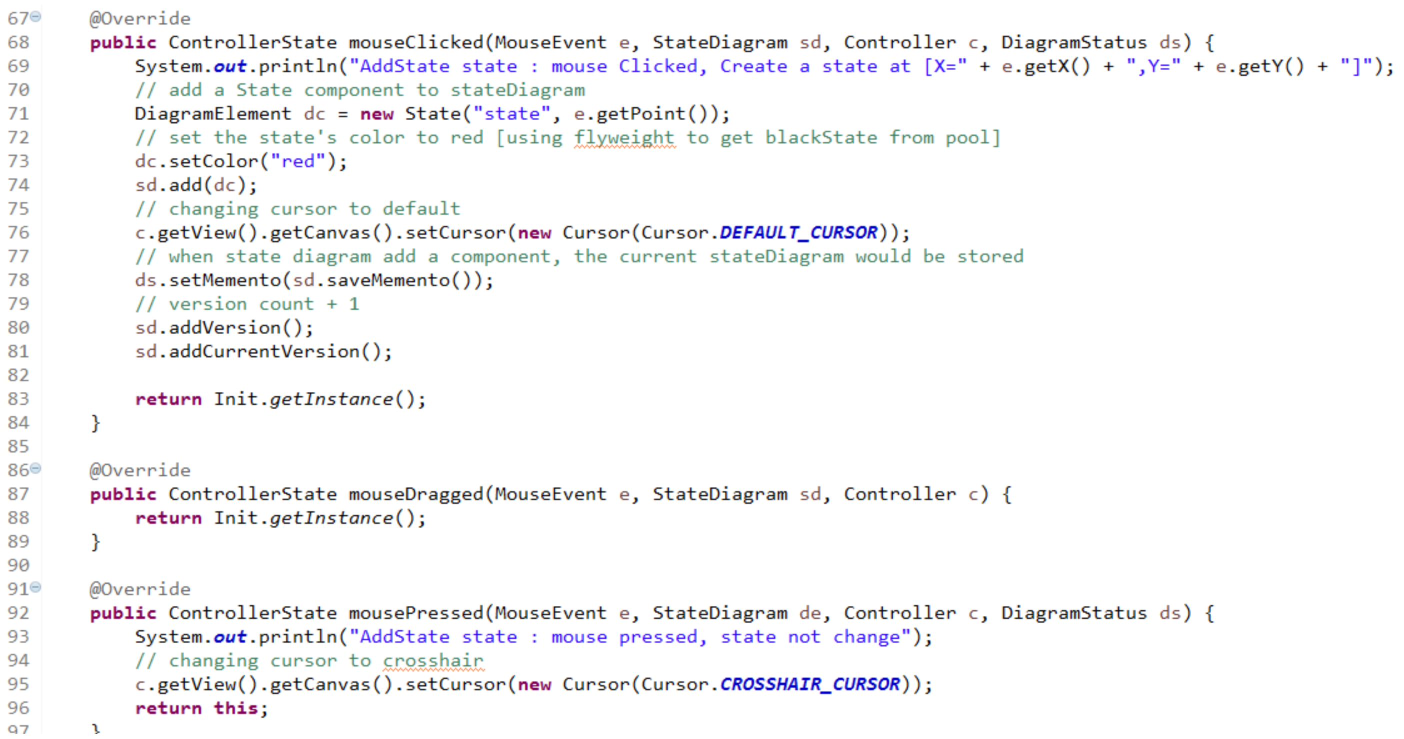
AddRedState Class：



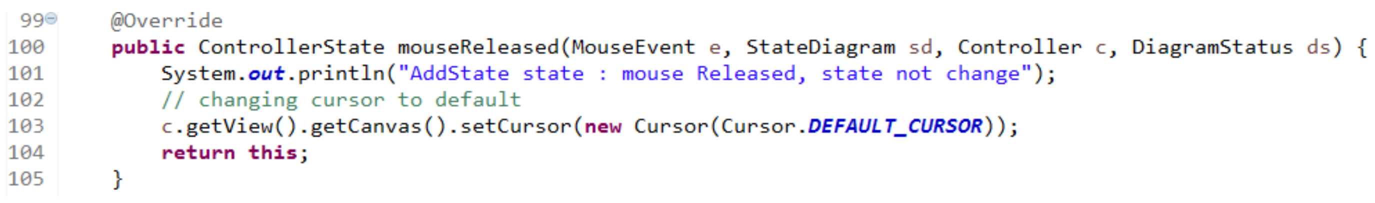
```java
package State;

import java.awt.Cursor;

//if you press the state button, change to this state

public class AddRedState implements ControllerState {
    // singleton
    private static AddRedState instance = null;

    private AddRedState() {
    }

    public static AddRedState getInstance() {
        if (instance == null) {
            return new AddRedState();
        }
        return instance;
    }

    @Override
    public ControllerState blackStateBtnClicked(ActionEvent e, Controller c) {
        System.out.println("AddState state : state button clicked, state not change");
        // changing the cursor to crosshair
        c.getView().getCanvas().setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR));
        return AddBlackState.getInstance();
    }

    @Override
    public ControllerState blueStateBtnClicked(ActionEvent e, Controller c) {
        System.out.println("AddState state : state button clicked, state not change");
        // changing the cursor to crosshair
        c.getView().getCanvas().setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR));
        return AddBlueState.getInstance();
    }
```



```java
    @Override
    public ControllerState redStateBtnClicked(ActionEvent e, Controller c) {
        System.out.println("AddState state : state button clicked, state not change");
        // changing the cursor to crosshair
        c.getView().getCanvas().setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR));
        return this;
    }

    @Override
    public ControllerState transBtnClicked(ActionEvent e, Controller c) {
        System.out.println("AddState state : transition button clicked, state changing to AddTransition");
        // changing the cursor to crosshair
        c.getView().getCanvas().setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR));
        return AddTransition.getInstance();
    }

    @Override
    public ControllerState deleteBtnClicked(ActionEvent e, Controller c) {
        System.out.println("AddState state : delete button clicked, state changing to TransSourceSelected");
        c.getView().getCanvas().setCursor(new Cursor(Cursor.HAND_CURSOR));
        return TransSourceSelected.getInstance();
    }
```

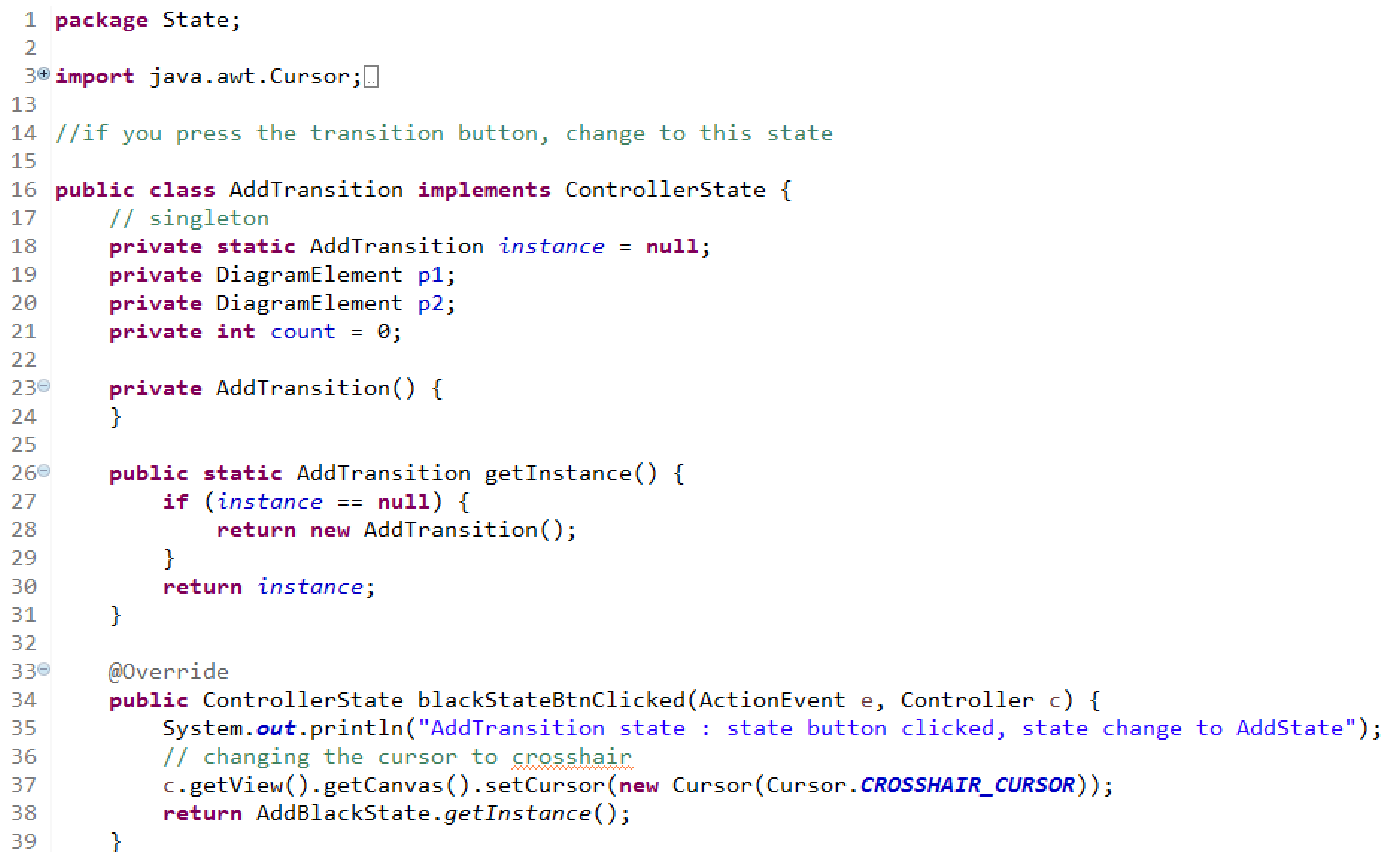


```java
    @Override
    public ControllerState mouseClicked(MouseEvent e, StateDiagram sd, Controller c, DiagramStatus ds) {
        System.out.println("AddState state : mouse Clicked, Create a state at [X=" + e.getX() + ",Y=" + e.getY() + "]");
        // add a State component to stateDiagram
        DiagramElement dc = new State("state", e.getPoint());
        // set the state's color to red [using flyweight to get blackState from pool]
        dc.setColor("red");
        sd.add(dc);
        // changing cursor to default
        c.getView().getCanvas().setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
        // when state diagram add a component, the current stateDiagram would be stored
        ds.setMemento(sd.saveMemento());
        // version count + 1
        sd.addVersion();
        sd.addCurrentVersion();

        return Init.getInstance();
    }

    @Override
    public ControllerState mouseDragged(MouseEvent e, StateDiagram sd, Controller c) {
        return Init.getInstance();
    }

    @Override
    public ControllerState mousePressed(MouseEvent e, StateDiagram de, Controller c, DiagramStatus ds) {
        System.out.println("AddState state : mouse pressed, state not change");
        // changing cursor to crosshair
        c.getView().getCanvas().setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR));
        return this;
    }
```

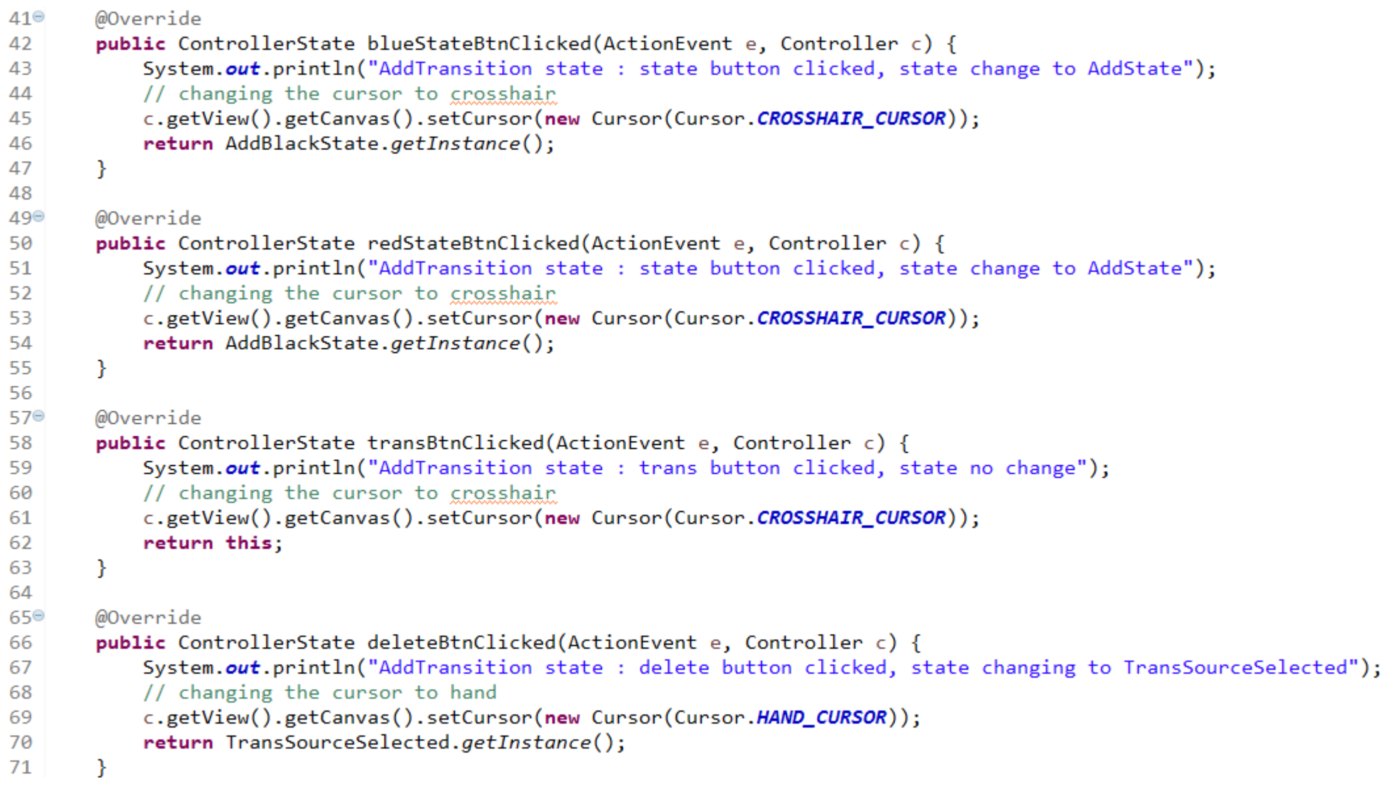


```java
    @Override
    public ControllerState mouseReleased(MouseEvent e, StateDiagram sd, Controller c, DiagramStatus ds) {
        System.out.println("AddState state : mouse Released, state not change");
        // changing cursor to default
        c.getView().getCanvas().setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
        return this;
    }
```

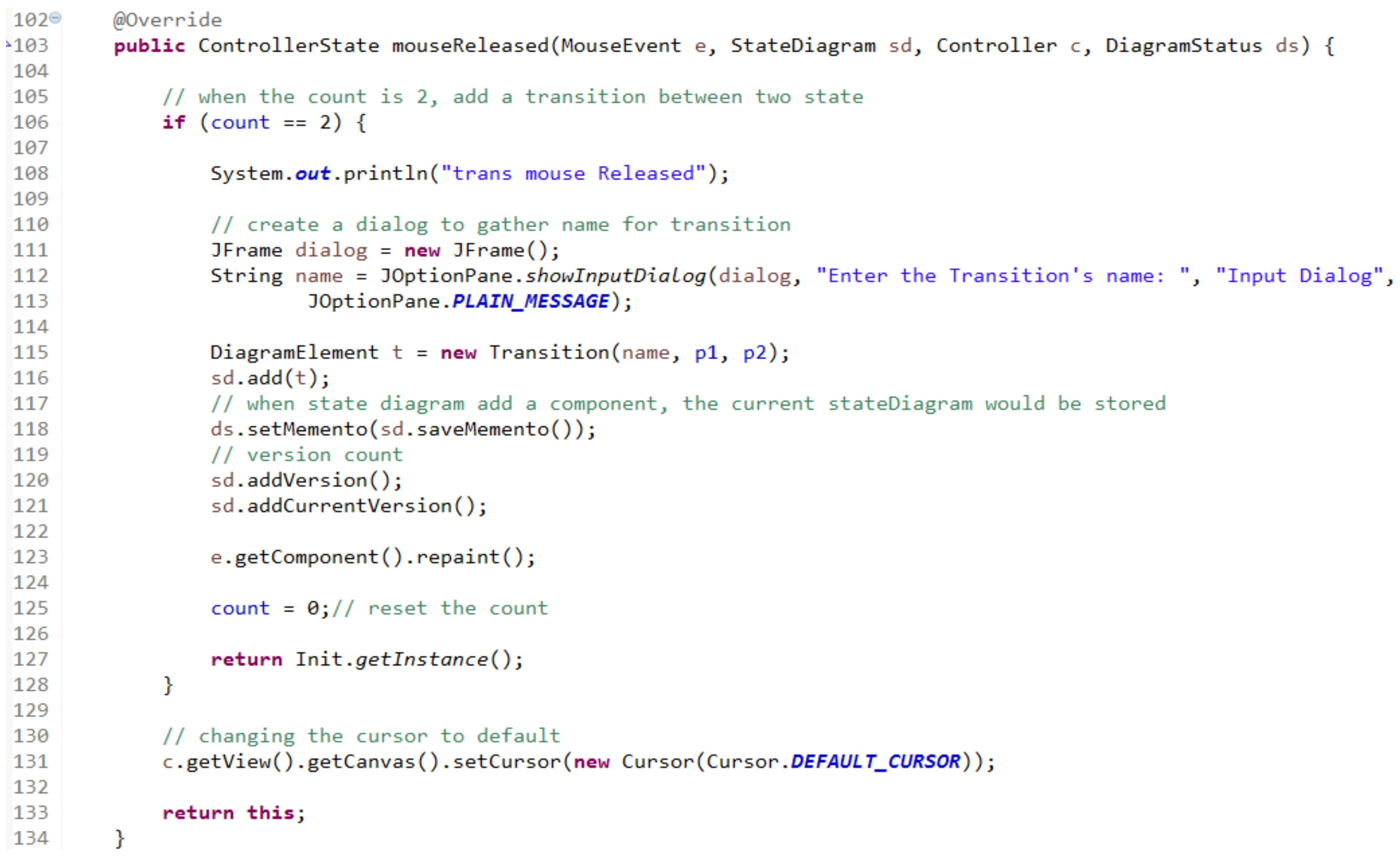
AddTransition Class：



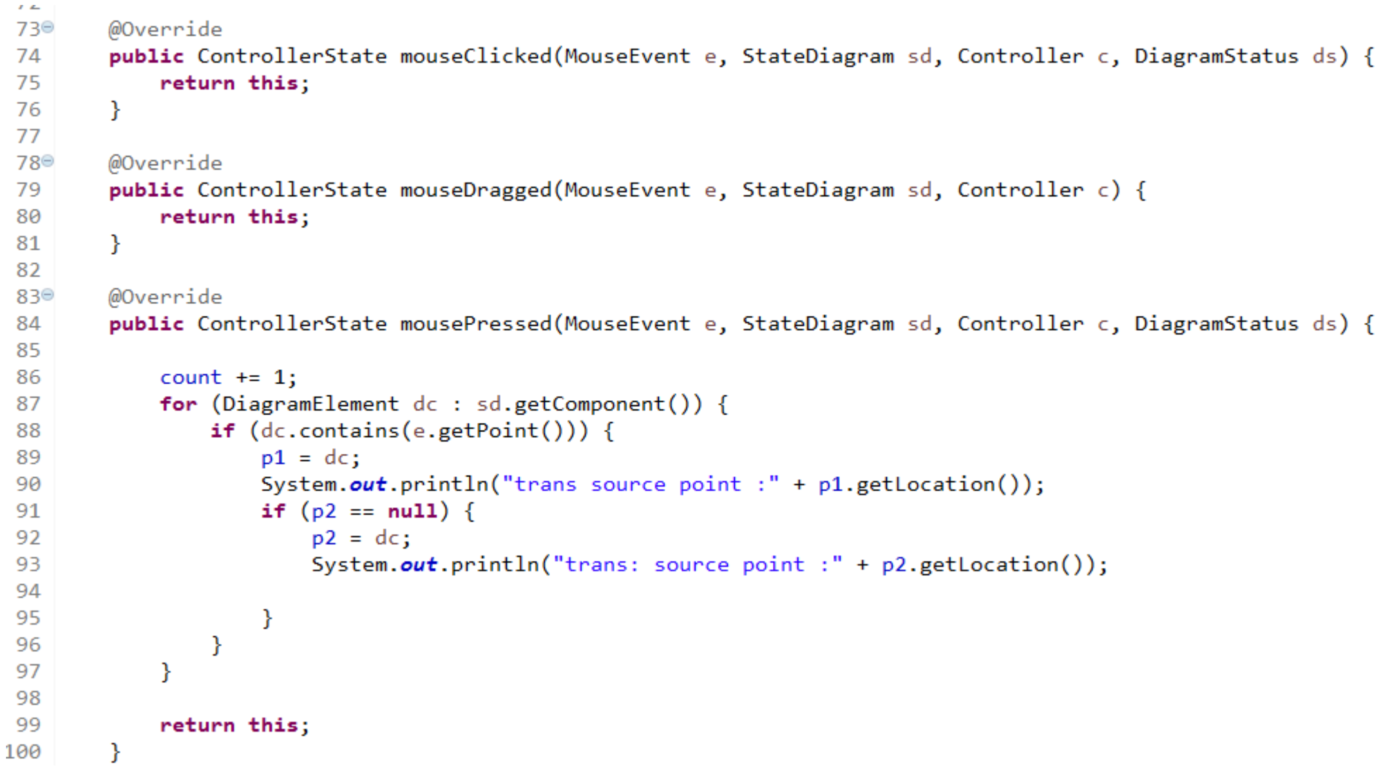
```java
package State;

import java.awt.Cursor;

//if you press the transition button, change to this state

public class AddTransition implements ControllerState {
    // singleton
    private static AddTransition instance = null;
    private DiagramElement p1;
    private DiagramElement p2;
    private int count = 0;

    private AddTransition() {
    }

    public static AddTransition getInstance() {
        if (instance == null) {
            return new AddTransition();
        }
        return instance;
    }

    @Override
    public ControllerState blackStateBtnClicked(ActionEvent e, Controller c) {
        System.out.println("AddTransition state : state button clicked, state change to AddState");
        // changing the cursor to crosshair
        c.getView().getCanvas().setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR));
        return AddBlackState.getInstance();
    }
```



```java
    @Override
    public ControllerState blueStateBtnClicked(ActionEvent e, Controller c) {
        System.out.println("AddTransition state : state button clicked, state change to AddState");
        // changing the cursor to crosshair
        c.getView().getCanvas().setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR));
        return AddBlackState.getInstance();
    }

    @Override
    public ControllerState redStateBtnClicked(ActionEvent e, Controller c) {
        System.out.println("AddTransition state : state button clicked, state change to AddState");
        // changing the cursor to crosshair
        c.getView().getCanvas().setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR));
        return AddBlackState.getInstance();
    }

    @Override
    public ControllerState transBtnClicked(ActionEvent e, Controller c) {
        System.out.println("AddTransition state : trans button clicked, state no change");
        // changing the cursor to crosshair
        c.getView().getCanvas().setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR));
        return this;
    }

    @Override
    public ControllerState deleteBtnClicked(ActionEvent e, Controller c) {
        System.out.println("AddTransition state : delete button clicked, state changing to TransSourceSelected");
        // changing the cursor to hand
        c.getView().getCanvas().setCursor(new Cursor(Cursor.HAND_CURSOR));
        return TransSourceSelected.getInstance();
    }
```

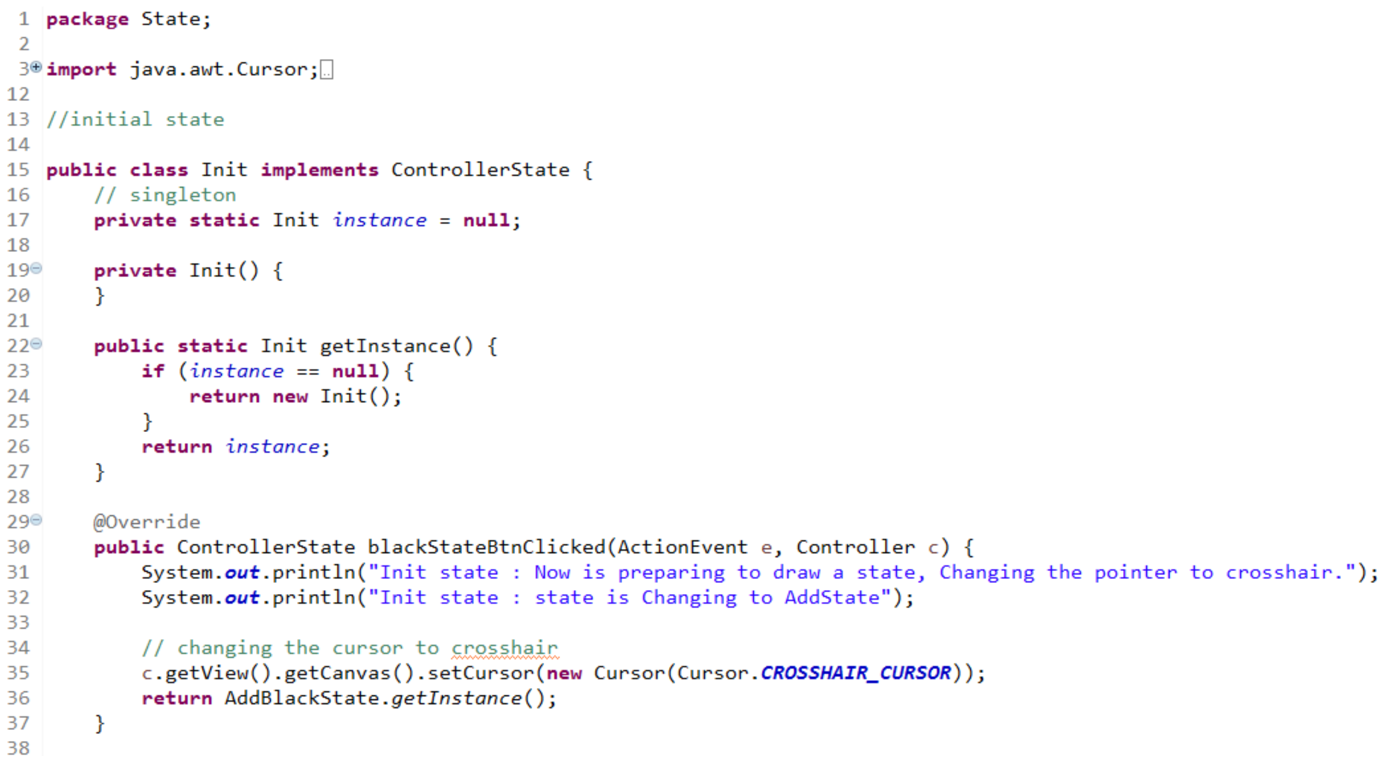


```
    @Override
    public ControllerState mouseReleased(MouseEvent e, StateDiagram sd, Controller c, DiagramStatus ds) {

        // when the count is 2, add a transition between two state
        if (count == 2) {

            System.out.println("trans mouse Released");

            // create a dialog to gather name for transition
            JFrame dialog = new JFrame();
            String name = JOptionPane.showInputDialog(dialog, "Enter the Transition's name: ", "Input Dialog",
                    JOptionPane.PLAIN_MESSAGE);

            DiagramElement t = new Transition(name, p1, p2);
            sd.add(t);
            // when state diagram add a component, the current stateDiagram would be stored
            ds.setMemento(sd.saveMemento());
            // version count
            sd.addVersion();
            sd.addCurrentVersion();

            e.getComponent().repaint();

            count = 0;// reset the count

            return Init.getInstance();
        }

        // changing the cursor to default
        c.getView().getCanvas().setCursor(new Cursor(Cursor.DEFAULT_CURSOR));

        return this;
    }
```

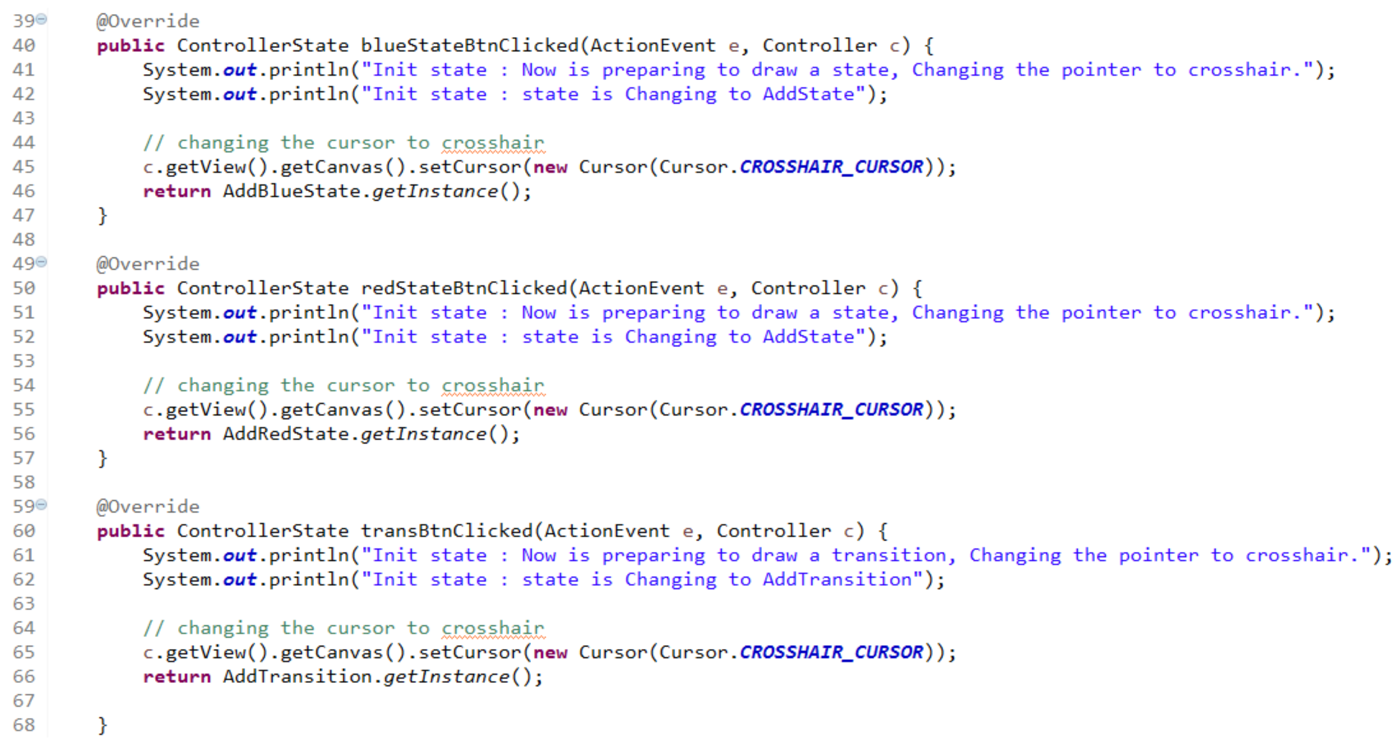


```
    @Override
    public ControllerState mouseClicked(MouseEvent e, StateDiagram sd, Controller c, DiagramStatus ds) {
        return this;
    }

    @Override
    public ControllerState mouseDragged(MouseEvent e, StateDiagram sd, Controller c) {
        return this;
    }

    @Override
    public ControllerState mousePressed(MouseEvent e, StateDiagram sd, Controller c, DiagramStatus ds) {

        count += 1;
        for (DiagramElement dc : sd.getComponent()) {
            if (dc.contains(e.getPoint())) {
                p1 = dc;
                System.out.println("trans source point :" + p1.getLocation());
                if (p2 == null) {
                    p2 = dc;
                    System.out.println("trans: source point :" + p2.getLocation());

                }
            }
        }

        return this;
    }
```

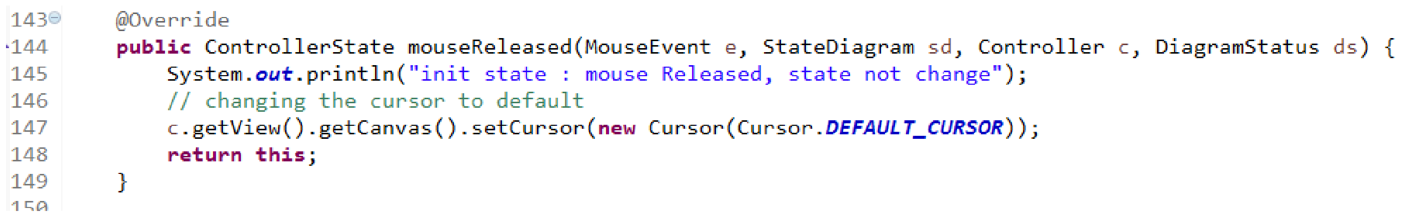
Init Class：



```java
package State;

import java.awt.Cursor;

//initial state

public class Init implements ControllerState {
    // singleton
    private static Init instance = null;

    private Init() {
    }

    public static Init getInstance() {
        if (instance == null) {
            return new Init();
        }
        return instance;
    }

    @Override
    public ControllerState blackStateBtnClicked(ActionEvent e, Controller c) {
        System.out.println("Init state : Now is preparing to draw a state, Changing the pointer to crosshair.");
        System.out.println("Init state : state is Changing to AddState");

        // changing the cursor to crosshair
        c.getView().getCanvas().setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR));
        return AddBlackState.getInstance();
    }

```

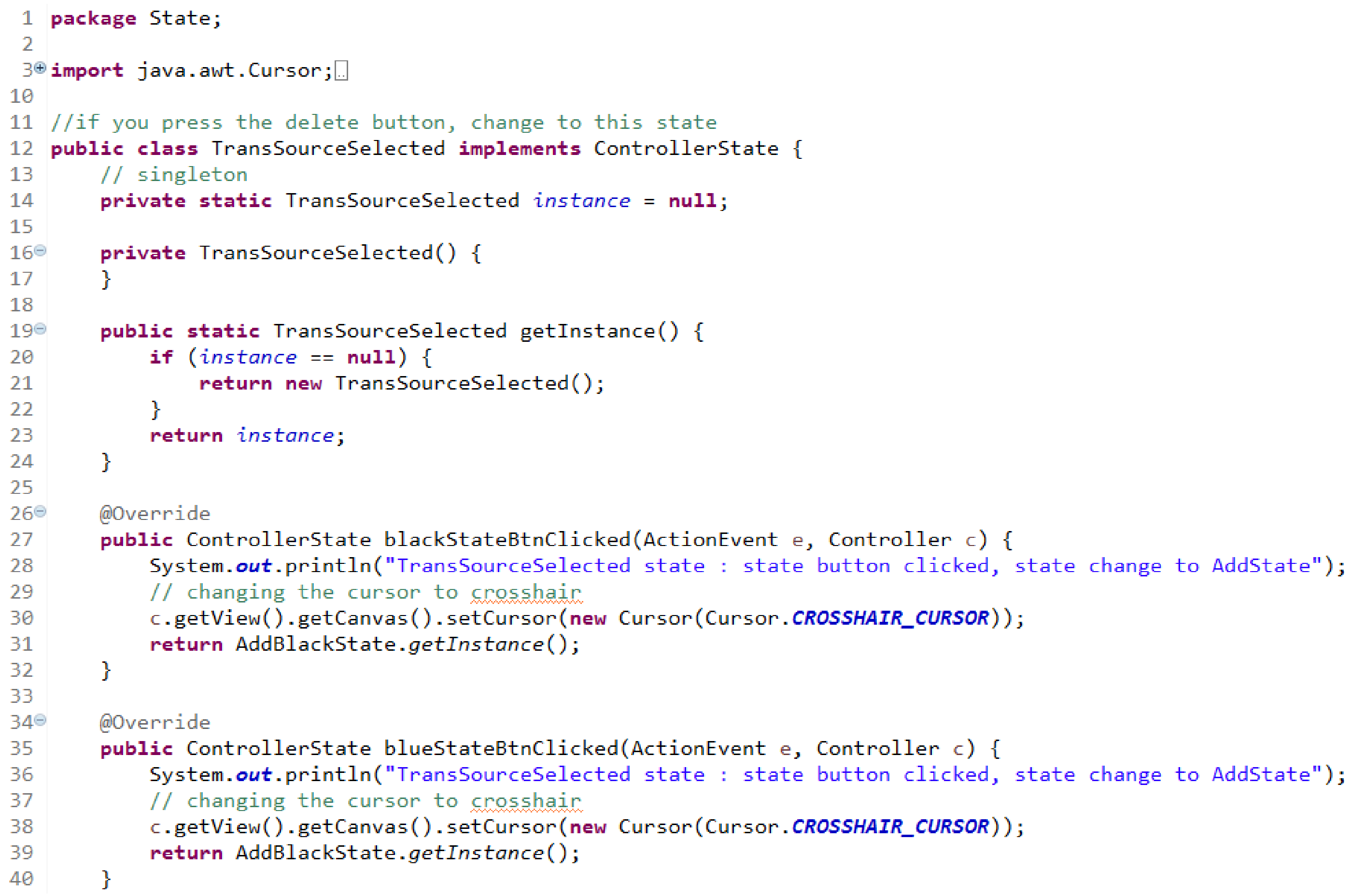


```java
    @Override
    public ControllerState blueStateBtnClicked(ActionEvent e, Controller c) {
        System.out.println("Init state : Now is preparing to draw a state, Changing the pointer to crosshair.");
        System.out.println("Init state : state is Changing to AddState");

        // changing the cursor to crosshair
        c.getView().getCanvas().setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR));
        return AddBlueState.getInstance();
    }

    @Override
    public ControllerState redStateBtnClicked(ActionEvent e, Controller c) {
        System.out.println("Init state : Now is preparing to draw a state, Changing the pointer to crosshair.");
        System.out.println("Init state : state is Changing to AddState");

        // changing the cursor to crosshair
        c.getView().getCanvas().setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR));
        return AddRedState.getInstance();
    }

    @Override
    public ControllerState transBtnClicked(ActionEvent e, Controller c) {
        System.out.println("Init state : Now is preparing to draw a transition, Changing the pointer to crosshair.");
        System.out.println("Init state : state is Changing to AddTransition");

        // changing the cursor to crosshair
        c.getView().getCanvas().setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR));
        return AddTransition.getInstance();

    }
    @Override
    public ControllerState deleteBtnClicked(ActionEvent e, Controller c) {
        System.out.println("Init state : state is Changing to TransSourceSelected");
        c.getView().getCanvas().setCursor(new Cursor(Cursor.HAND_CURSOR));
        return TransSourceSelected.getInstance();

    }

    @Override
    public ControllerState mouseClicked(MouseEvent e, StateDiagram sd, Controller c, DiagramStatus ds) {
        System.out.println("init state : mouse clicked, state not change");

        for (DiagramElement dc : sd.getComponent()) {
            // if pointer inside the component's range, then pop up a dialog to edit State
            // name
            if (dc.contains(e.getPoint())) {

                // changing the cursor to default
                c.getView().getCanvas().setCursor(new Cursor(Cursor.DEFAULT_CURSOR));

                JFrame dialog = new JFrame();
                String name = JOptionPane.showInputDialog(dialog, "Enter the State's name: ", "Input Dialog",
                        JOptionPane.PLAIN_MESSAGE);

                if (name == null) {
                    dc.setName("name");
                } else {
                    dc.setName(name);
                }

            }
        }

        return this;
    }
    @Override
    public ControllerState mouseDragged(MouseEvent e, StateDiagram sd, Controller c) {
        System.out.println("init state : mouse Pressed, state not change");

        // when state be dragged, update the position for state
        for (DiagramElement d : sd.getComponent()) {
            // pointer need to inside the state, and state is be selected
            // pre-condition : to avoid dragged all the component at same time
            if (d.contains(e.getPoint()) && d.isSelected() == true) {
                // changing the cursor to Move
                c.getView().getCanvas().setCursor(new Cursor(Cursor.MOVE_CURSOR));
                d.updatePosition(e.getPoint());
                e.getComponent().repaint();
            }
        }

        return this;
    }

    @Override
    public ControllerState mousePressed(MouseEvent e, StateDiagram sd, Controller c, DiagramStatus ds) {
        System.out.println("init state : mouse Pressed, state not change");
        for (DiagramElement d : sd.getComponent()) {
            // if state be pressed, display the being selected component
            if (d.contains(e.getPoint())) {
                // changing the cursor to hand_mode
                c.getView().getCanvas().setCursor(new Cursor(Cursor.HAND_CURSOR));
                d.setSelected(true);
            } else {
                // changing the cursor to default
                c.getView().getCanvas().setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
                d.setSelected(false);
            }
        }
        return this;
    }
```

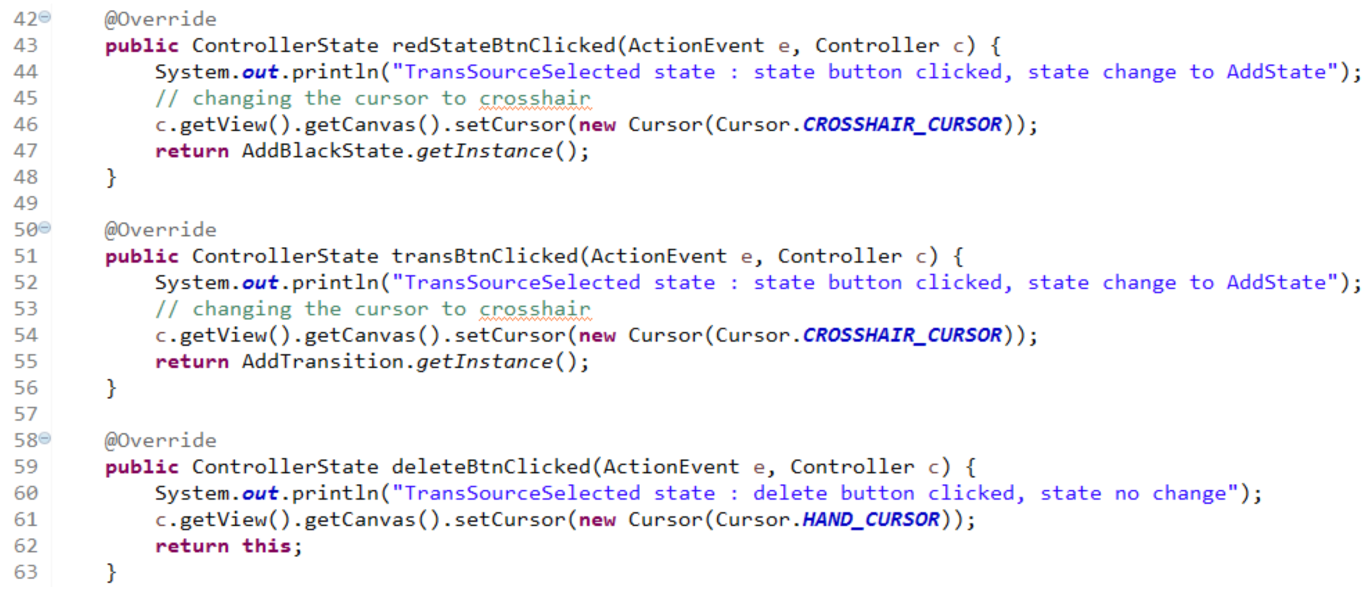


```java
    @Override
    public ControllerState mouseReleased(MouseEvent e, StateDiagram sd, Controller c, DiagramStatus ds) {
        System.out.println("init state : mouse Released, state not change");
        // changing the cursor to default
        c.getView().getCanvas().setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
        return this;
    }
```

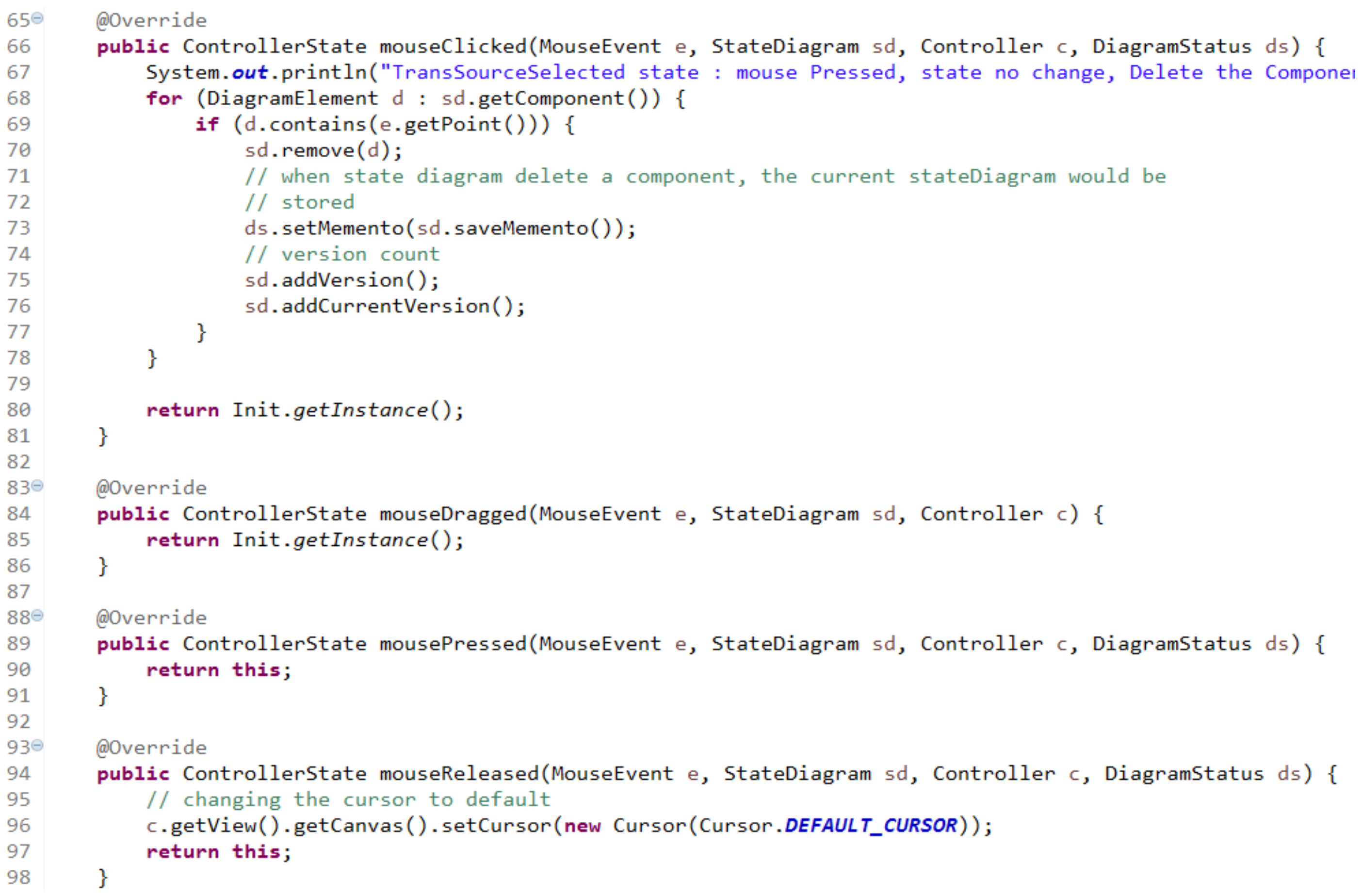
TransSourceSelected Class：



```java
package State;

import java.awt.Cursor;

//if you press the delete button, change to this state
public class TransSourceSelected implements ControllerState {
    // singleton
    private static TransSourceSelected instance = null;

    private TransSourceSelected() {
    }

    public static TransSourceSelected getInstance() {
        if (instance == null) {
            return new TransSourceSelected();
        }
        return instance;
    }

    @Override
    public ControllerState blackStateBtnClicked(ActionEvent e, Controller c) {
        System.out.println("TransSourceSelected state : state button clicked, state change to AddState");
        // changing the cursor to crosshair
        c.getView().getCanvas().setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR));
        return AddBlackState.getInstance();
    }

    @Override
    public ControllerState blueStateBtnClicked(ActionEvent e, Controller c) {
        System.out.println("TransSourceSelected state : state button clicked, state change to AddState");
        // changing the cursor to crosshair
        c.getView().getCanvas().setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR));
        return AddBlackState.getInstance();
    }
```

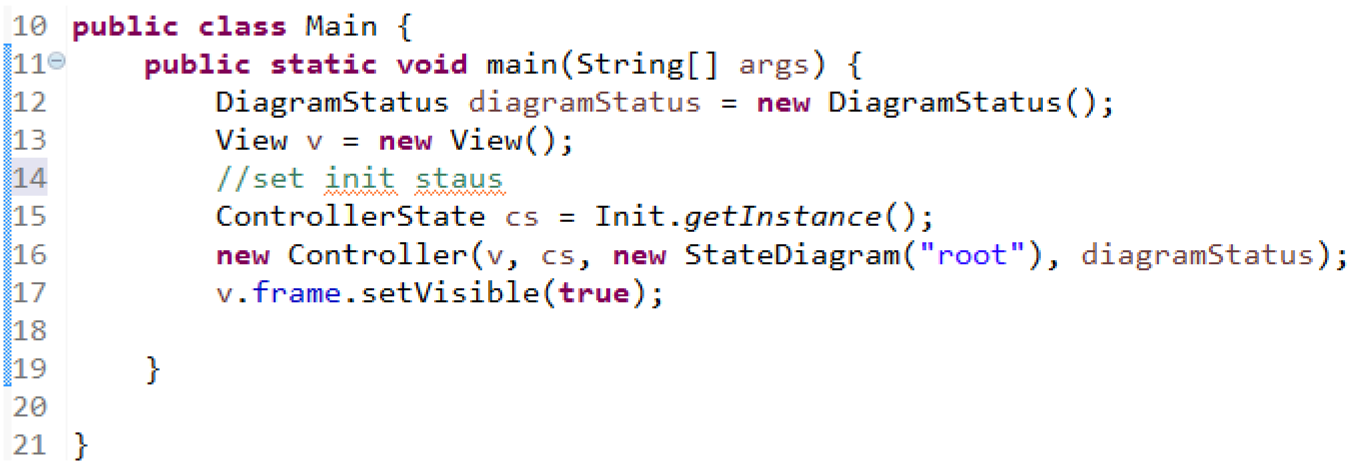


```java
    @Override
    public ControllerState redStateBtnClicked(ActionEvent e, Controller c) {
        System.out.println("TransSourceSelected state : state button clicked, state change to AddState");
        // changing the cursor to crosshair
        c.getView().getCanvas().setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR));
        return AddBlackState.getInstance();
    }

    @Override
    public ControllerState transBtnClicked(ActionEvent e, Controller c) {
        System.out.println("TransSourceSelected state : state button clicked, state change to AddState");
        // changing the cursor to crosshair
        c.getView().getCanvas().setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR));
        return AddTransition.getInstance();
    }

    @Override
    public ControllerState deleteBtnClicked(ActionEvent e, Controller c) {
        System.out.println("TransSourceSelected state : delete button clicked, state no change");
        c.getView().getCanvas().setCursor(new Cursor(Cursor.HAND_CURSOR));
        return this;
    }
```



```java
    @Override
    public ControllerState mouseClicked(MouseEvent e, StateDiagram sd, Controller c, DiagramStatus ds) {
        System.out.println("TransSourceSelected state : mouse Pressed, state no change, Delete the Compone
        for (DiagramElement d : sd.getComponent()) {
            if (d.contains(e.getPoint())) {
                sd.remove(d);
                // when state diagram delete a component, the current stateDiagram would be
                // stored
                ds.setMemento(sd.saveMemento());
                // version count
                sd.addVersion();
                sd.addCurrentVersion();
            }
        }

        return Init.getInstance();
    }

    @Override
    public ControllerState mouseDragged(MouseEvent e, StateDiagram sd, Controller c) {
        return Init.getInstance();
    }

    @Override
    public ControllerState mousePressed(MouseEvent e, StateDiagram sd, Controller c, DiagramStatus ds) {
        return this;
    }

    @Override
    public ControllerState mouseReleased(MouseEvent e, StateDiagram sd, Controller c, DiagramStatus ds) {
        // changing the cursor to default
        c.getView().getCanvas().setCursor(new Cursor(Cursor.DEFAULT_CURSOR));
        return this;
    }
```

Main：



```java
public class Main {
    public static void main(String[] args) {
        DiagramStatus diagramStatus = new DiagramStatus();
        View v = new View();
        //set init staus
        ControllerState cs = Init.getInstance();
        new Controller(v, cs, new StateDiagram("root"), diagramStatus);
        v.frame.setVisible(true);

    }

}
```