# Memento in Java

**Memento** is a behavioral design pattern that allows making snapshots of an object's state and restoring it in future.

The Memento doesn't compromise the internal structure of the object it works with, as well as data kept inside the snapshots.

Learn more about Memento →

## Navigation

Demo

OutputDemo

**Complexity:** ★★★

**Popularity:** ★☆☆

**Usage examples:** The Memento's principle can be achieved using serialization, which is quite common in Java. While it's not the only and the most efficient way to make snapshots of an object's state, it still allows storing state backups while protecting the originator's structure from other objects.

Here are some examples of the pattern in core Java libraries:

- All `java.io.Serializable` implementations can simulate the Memento.
- All `javax.faces.component.StateHolder` implementations.

## Shape editor and complex undo/redo

This graphical editor allows changing the color and position of the shapes on the screen. Any modification can be undone and repeated, though.

The "undo" is based on the collaboration between the Memento and Command patterns. The editor tracks a history of performed commands. Before executing any command, it makes a backup and connects it to the command object. After the execution, it pushes the executed command into history.

When a user requests the undo, the editor fetches a recent command from the history and restores the state from the backup kept inside that command. If the user requests another undo, the editor takes a following command from the history and so on.

Reverted commands are kept in history until the user makes some modifications to the shapes on the screen. This is crucial for redoing undone commands.

**editor**

**editor/Editor.java:** Editor code

```
import refactoring_guru.memento.example.commands.Command;
import refactoring_guru.memento.example.history.History;
import refactoring_guru.memento.example.history.Memento;
import refactoring_guru.memento.example.shapes.CompoundShape;
import refactoring_guru.memento.example.shapes.Shape;

import javax.swing.*;
import java.io.*;
import java.util.Base64;

public class Editor extends JComponent {
    private Canvas canvas;
    private CompoundShape allShapes = new CompoundShape();
    private History history;

    public Editor() {
        canvas = new Canvas(this);
        history = new History();
    }

    public void loadShapes(Shape... shapes) {
        allShapes.clear();
        allShapes.add(shapes);
        canvas.refresh();
    }

    public CompoundShape getShapes() {
        return allShapes;
    }

    public void execute(Command c) {
        history.push(c, new Memento(this));
        c.execute();
    }

    public void undo() {
        if (history.undo())
            canvas.repaint();
    }

    public void redo() {
        if (history.redo())
            canvas.repaint();
    }

    public String backup() {
        try {
            ByteArrayOutputStream baos = new ByteArrayOutputStream();
            ObjectOutputStream oos = new ObjectOutputStream(baos);
```

```
            return Base64.getEncoder().encodeToString(baos.toByteArray());
        } catch (IOException e) {
            return "";
        }
    }

    public void restore(String state) {
        try {
            byte[] data = Base64.getDecoder().decode(state);
            ObjectInputStream ois = new ObjectInputStream(new ByteArrayInputStream(data));
            this.allShapes = (CompoundShape) ois.readObject();
            ois.close();
        } catch (ClassNotFoundException e) {
            System.out.print("ClassNotFoundException occurred.");
        } catch (IOException e) {
            System.out.print("IOException occurred.");
        }
    }
}
```

## editor/Canvas.java: Canvas code

```
package refactoring_guru.memento.example.editor;

import refactoring_guru.memento.example.commands.ColorCommand;
import refactoring_guru.memento.example.commands.MoveCommand;
import refactoring_guru.memento.example.shapes.Shape;

import javax.swing.*;
import javax.swing.border.Border;
import java.awt.*;
import java.awt.event.*;
import java.awt.image.BufferedImage;

class Canvas extends java.awt.Canvas {
    private Editor editor;
    private JFrame frame;
    private static final int PADDING = 10;

    Canvas(Editor editor) {
        this.editor = editor;
        createFrame();
        attachKeyboardListeners();
        attachMouseListeners();
        refresh();
    }
```

```
        frame = new JFrame();
        frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        frame.setLocationRelativeTo(null);

        JPanel contentPanel = new JPanel();
        Border padding = BorderFactory.createEmptyBorder(PADDING, PADDING, PADDING, PADDI
        contentPanel.setBorder(padding);
        contentPanel.setLayout(new BoxLayout(contentPanel, BoxLayout.Y_AXIS));
        frame.setContentPane(contentPanel);

        contentPanel.add(new JLabel("Select and drag to move."), BorderLayout.PAGE_END);
        contentPanel.add(new JLabel("Right click to change color."), BorderLayout.PAGE_EN
        contentPanel.add(new JLabel("Undo: Ctrl+Z, Redo: Ctrl+R"), BorderLayout.PAGE_END)
        contentPanel.add(this);
        frame.setVisible(true);
        contentPanel.setBackground(Color.LIGHT_GRAY);
    }

    private void attachKeyboardListeners() {
        addKeyListener(new KeyAdapter() {
            @Override
            public void keyPressed(KeyEvent e) {
                if ((e.getModifiers() & KeyEvent.CTRL_MASK) != 0) {
                    switch (e.getKeyCode()) {
                        case KeyEvent.VK_Z:
                            editor.undo();
                            break;
                        case KeyEvent.VK_R:
                            editor.redo();
                            break;
                    }
                }
            }
        });
    }

    private void attachMouseListeners() {
        MouseAdapter colorizer = new MouseAdapter() {
            @Override
            public void mousePressed(MouseEvent e) {
                if (e.getButton() != MouseEvent.BUTTON3) {
                    return;
                }
                Shape target = editor.getShapes().getChildAt(e.getX(), e.getY());
                if (target != null) {
                    editor.execute(new ColorCommand(editor, new Color((int) (Math.random(
                    repaint();
                }
            }
        };
```

```
MouseAdapter selector = new MouseAdapter() {
    @Override
    public void mousePressed(MouseEvent e) {
        if (e.getButton() != MouseEvent.BUTTON1) {
            return;
        }

        Shape target = editor.getShapes().getChildAt(e.getX(), e.getY());
        boolean ctrl = (e.getModifiers() & ActionEvent.CTRL_MASK) == ActionEvent.

        if (target == null) {
            if (!ctrl) {
                editor.getShapes().unSelect();
            }
        } else {
            if (ctrl) {
                if (target.isSelected()) {
                    target.unSelect();
                } else {
                    target.select();
                }
            } else {
                if (!target.isSelected()) {
                    editor.getShapes().unSelect();
                }
                target.select();
            }
        }
        repaint();
    }
};
addMouseListener(selector);


MouseAdapter dragger = new MouseAdapter() {
    MoveCommand moveCommand;

    @Override
    public void mouseDragged(MouseEvent e) {
        if ((e.getModifiersEx() & MouseEvent.BUTTON1_DOWN_MASK) != MouseEvent.BUT
            return;
        }
        if (moveCommand == null) {
            moveCommand = new MoveCommand(editor);
            moveCommand.start(e.getX(), e.getY());
        }
        moveCommand.move(e.getX(), e.getY());
        repaint();
    }
```

```
                if (e.getButton() != MouseEvent.BUTTON1 || moveCommand == null) {
                    return;
                }
                moveCommand.stop(e.getX(), e.getY());
                editor.execute(moveCommand);
                this.moveCommand = null;
                repaint();
            }
        };
        addMouseListener(dragger);
        addMouseMotionListener(dragger);
    }

    public int getWidth() {
        return editor.getShapes().getX() + editor.getShapes().getWidth() + PADDING;
    }

    public int getHeight() {
        return editor.getShapes().getY() + editor.getShapes().getHeight() + PADDING;
    }

    void refresh() {
        this.setSize(getWidth(), getHeight());
        frame.pack();
    }

    public void update(Graphics g) {
        paint(g);
    }

    public void paint(Graphics graphics) {
        BufferedImage buffer = new BufferedImage(this.getWidth(), this.getHeight(), Buffe
        Graphics2D ig2 = buffer.createGraphics();
        ig2.setBackground(Color.WHITE);
        ig2.clearRect(0, 0, this.getWidth(), this.getHeight());

        editor.getShapes().paint(buffer.getGraphics());

        graphics.drawImage(buffer, 0, 0, null);
    }
}
```

**history**

**history/History.java:** History stores commands and mementos

```
import refactoring_guru.memento.example.commands.Command;

import java.util.ArrayList;
import java.util.List;

public class History {
    private List<Pair> history = new ArrayList<Pair>();
    private int virtualSize = 0;

    private class Pair {
        Command command;
        Memento memento;
        Pair(Command c, Memento m) {
            command = c;
            memento = m;
        }

        private Command getCommand() {
            return command;
        }

        private Memento getMemento() {
            return memento;
        }
    }

    public void push(Command c, Memento m) {
        if (virtualSize != history.size() && virtualSize > 0) {
            history = history.subList(0, virtualSize - 1);
        }
        history.add(new Pair(c, m));
        virtualSize = history.size();
    }

    public boolean undo() {
        Pair pair = getUndo();
        if (pair == null) {
            return false;
        }
        System.out.println("Undoing: " + pair.getCommand().getName());
        pair.getMemento().restore();
        return true;
    }

    public boolean redo() {
        Pair pair = getRedo();
        if (pair == null) {
            return false;
        }
```

```
            pair.getCommand().execute();
            return true;
        }

        private Pair getUndo() {
            if (virtualSize == 0) {
                return null;
            }
            virtualSize = Math.max(0, virtualSize - 1);
            return history.get(virtualSize);
        }

        private Pair getRedo() {
            if (virtualSize == history.size()) {
                return null;
            }
            virtualSize = Math.min(history.size(), virtualSize + 1);
            return history.get(virtualSize - 1);
        }
    }
```

### history/Memento.java: Memento class

```
package refactoring_guru.memento.example.history;

import refactoring_guru.memento.example.editor.Editor;

public class Memento {
    private String backup;
    private Editor editor;

    public Memento(Editor editor) {
        this.editor = editor;
        this.backup = editor.backup();
    }

    public void restore() {
        editor.restore(backup);
    }
}
```

## commands

### commands/Command.java: Base command class

```
public interface Command {
    String getName();
    void execute();
}
```

### commands/ColorCommand.java: Changes color of selected shape

```
package refactoring_guru.memento.example.commands;

import refactoring_guru.memento.example.editor.Editor;
import refactoring_guru.memento.example.shapes.Shape;

import java.awt.*;

public class ColorCommand implements Command {
    private Editor editor;
    private Color color;

    public ColorCommand(Editor editor, Color color) {
        this.editor = editor;
        this.color = color;
    }

    @Override
    public String getName() {
        return "Colorize: " + color.toString();
    }

    @Override
    public void execute() {
        for (Shape child : editor.getShapes().getSelected()) {
            child.setColor(color);
        }
    }
}
```

### commands/MoveCommand.java: Moves selected shape

```
package refactoring_guru.memento.example.commands;

import refactoring_guru.memento.example.editor.Editor;
import refactoring_guru.memento.example.shapes.Shape;
```

```
    private Editor editor;
    private int startX, startY;
    private int endX, endY;

    public MoveCommand(Editor editor) {
        this.editor = editor;
    }

    @Override
    public String getName() {
        return "Move by X:" + (endX - startX) + " Y:" + (endY - startY);
    }

    public void start(int x, int y) {
        startX = x;
        startY = y;
        for (Shape child : editor.getShapes().getSelected()) {
            child.drag();
        }
    }

    public void move(int x, int y) {
        for (Shape child : editor.getShapes().getSelected()) {
            child.moveTo(x - startX, y - startY);
        }
    }

    public void stop(int x, int y) {
        endX = x;
        endY = y;
        for (Shape child : editor.getShapes().getSelected()) {
            child.drop();
        }
    }

    @Override
    public void execute() {
        for (Shape child : editor.getShapes().getSelected()) {
            child.moveBy(endX - startX, endY - startY);
        }
    }
}
```

**shapes:** Various shapes

**shapes/Shape.java**

```
import java.awt.*;
import java.io.Serializable;

public interface Shape extends Serializable {
    int getX();
    int getY();
    int getWidth();
    int getHeight();
    void drag();
    void drop();
    void moveTo(int x, int y);
    void moveBy(int x, int y);
    boolean isInsideBounds(int x, int y);
    Color getColor();
    void setColor(Color color);
    void select();
    void unSelect();
    boolean isSelected();
    void paint(Graphics graphics);
}
```

### shapes/BaseShape.java

```
package refactoring_guru.memento.example.shapes;

import java.awt.*;

public abstract class BaseShape implements Shape {
    int x, y;
    private int dx = 0, dy = 0;
    private Color color;
    private boolean selected = false;

    BaseShape(int x, int y, Color color) {
        this.x = x;
        this.y = y;
        this.color = color;
    }

    @Override
    public int getX() {
        return x;
    }

    @Override
    public int getY() {
```

```
@Override
public int getWidth() {
    return 0;
}

@Override
public int getHeight() {
    return 0;
}

@Override
public void drag() {
    dx = x;
    dy = y;
}

@Override
public void moveTo(int x, int y) {
    this.x = dx + x;
    this.y = dy + y;
}

@Override
public void moveBy(int x, int y) {
    this.x += x;
    this.y += y;
}

@Override
public void drop() {
    this.x = dx;
    this.y = dy;
}

@Override
public boolean isInsideBounds(int x, int y) {
    return x > getX() && x < (getX() + getWidth()) &&
            y > getY() && y < (getY() + getHeight());
}

@Override
public Color getColor() {
    return color;
}

@Override
public void setColor(Color color) {
    this.color = color;
}
```

```
    public void select() {
        selected = true;
    }

    @Override
    public void unSelect() {
        selected = false;
    }

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

    void enableSelectionStyle(Graphics graphics) {
        graphics.setColor(Color.LIGHT_GRAY);

        Graphics2D g2 = (Graphics2D) graphics;
        float[] dash1 = {2.0f};
        g2.setStroke(new BasicStroke(1.0f,
                BasicStroke.CAP_BUTT,
                BasicStroke.JOIN_MITER,
                2.0f, dash1, 0.0f));
    }

    void disableSelectionStyle(Graphics graphics) {
        graphics.setColor(color);
        Graphics2D g2 = (Graphics2D) graphics;
        g2.setStroke(new BasicStroke());
    }

    @Override
    public void paint(Graphics graphics) {
        if (isSelected()) {
            enableSelectionStyle(graphics);
        }
        else {
            disableSelectionStyle(graphics);
        }

        // ...
    }
}
```

**shapes/Circle.java**

```
import java.awt.*;

public class Circle extends BaseShape {
    private int radius;

    public Circle(int x, int y, int radius, Color color) {
        super(x, y, color);
        this.radius = radius;
    }

    @Override
    public int getWidth() {
        return radius * 2;
    }

    @Override
    public int getHeight() {
        return radius * 2;
    }

    @Override
    public void paint(Graphics graphics) {
        super.paint(graphics);
        graphics.drawOval(x, y, getWidth() - 1, getHeight() - 1);
    }
}
```

### shapes/Dot.java

```
package refactoring_guru.memento.example.shapes;

import java.awt.*;

public class Dot extends BaseShape {
    private final int DOT_SIZE = 3;

    public Dot(int x, int y, Color color) {
        super(x, y, color);
    }

    @Override
    public int getWidth() {
        return DOT_SIZE;
    }

    @Override
```

```
    }

    @Override
    public void paint(Graphics graphics) {
        super.paint(graphics);
        graphics.fillRect(x - 1, y - 1, getWidth(), getHeight());
    }
}
```

### shapes/Rectangle.java

```
package refactoring_guru.memento.example.shapes;

import java.awt.*;

public class Rectangle extends BaseShape {
    private int width;
    private int height;

    public Rectangle(int x, int y, int width, int height, Color color) {
        super(x, y, color);
        this.width = width;
        this.height = height;
    }

    @Override
    public int getWidth() {
        return width;
    }

    @Override
    public int getHeight() {
        return height;
    }

    @Override
    public void paint(Graphics graphics) {
        super.paint(graphics);
        graphics.drawRect(x, y, getWidth() - 1, getHeight() - 1);
    }
}
```

### shapes/CompoundShape.java

```
import java.awt.*;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class CompoundShape extends BaseShape {
    private List<Shape> children = new ArrayList<>();

    public CompoundShape(Shape... components) {
        super(0, 0, Color.BLACK);
        add(components);
    }

    public void add(Shape component) {
        children.add(component);
    }

    public void add(Shape... components) {
        children.addAll(Arrays.asList(components));
    }

    public void remove(Shape child) {
        children.remove(child);
    }

    public void remove(Shape... components) {
        children.removeAll(Arrays.asList(components));
    }

    public void clear() {
        children.clear();
    }

    @Override
    public int getX() {
        if (children.size() == 0) {
            return 0;
        }
        int x = children.get(0).getX();
        for (Shape child : children) {
            if (child.getX() < x) {
                x = child.getX();
            }
        }
        return x;
    }

    @Override
    public int getY() {
```

```
        }
        int y = children.get(0).getY();
        for (Shape child : children) {
            if (child.getY() < y) {
                y = child.getY();
            }
        }
        return y;
    }

    @Override
    public int getWidth() {
        int maxWidth = 0;
        int x = getX();
        for (Shape child : children) {
            int childsRelativeX = child.getX() - x;
            int childWidth = childsRelativeX + child.getWidth();
            if (childWidth > maxWidth) {
                maxWidth = childWidth;
            }
        }
        return maxWidth;
    }

    @Override
    public int getHeight() {
        int maxHeight = 0;
        int y = getY();
        for (Shape child : children) {
            int childsRelativeY = child.getY() - y;
            int childHeight = childsRelativeY + child.getHeight();
            if (childHeight > maxHeight) {
                maxHeight = childHeight;
            }
        }
        return maxHeight;
    }

    @Override
    public void drag() {
        for (Shape child : children) {
            child.drag();
        }
    }

    @Override
    public void drop() {
        for (Shape child : children) {
            child.drop();
        }
```

```
    @Override
    public void moveTo(int x, int y) {
        for (Shape child : children) {
            child.moveTo(x, y);
        }
    }

    @Override
    public void moveBy(int x, int y) {
        for (Shape child : children) {
            child.moveBy(x, y);
        }
    }

    @Override
    public boolean isInsideBounds(int x, int y) {
        for (Shape child : children) {
            if (child.isInsideBounds(x, y)) {
                return true;
            }
        }
        return false;
    }

    @Override
    public void setColor(Color color) {
        super.setColor(color);
        for (Shape child : children) {
            child.setColor(color);
        }
    }

    @Override
    public void unSelect() {
        super.unSelect();
        for (Shape child : children) {
            child.unSelect();
        }
    }

    public Shape getChildAt(int x, int y) {
        for (Shape child : children) {
            if (child.isInsideBounds(x, y)) {
                return child;
            }
        }
        return null;
    }

    public boolean selectChildAt(int x, int y) {
```

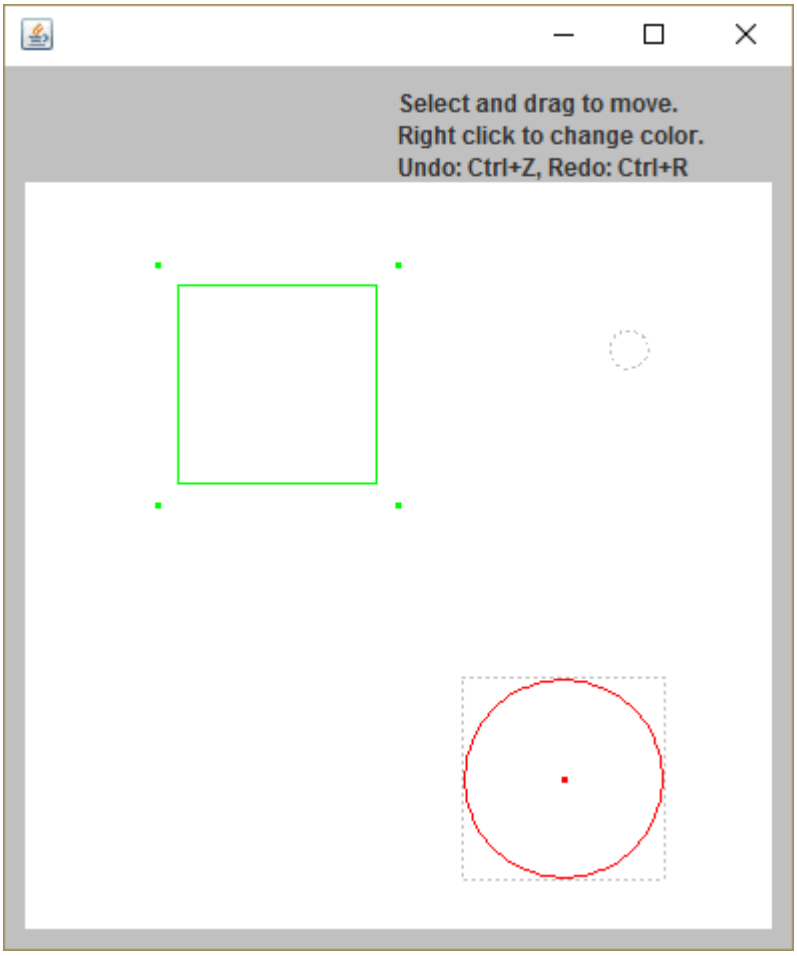
```
            child.select();
            return true;
        }
        return false;
    }

    public List<Shape> getSelected() {
        List<Shape> selected = new ArrayList<>();
        for (Shape child : children) {
            if (child.isSelected()) {
                selected.add(child);
            }
        }
        return selected;
    }

    @Override
    public void paint(Graphics graphics) {
        if (isSelected()) {
            enableSelectionStyle(graphics);
            graphics.drawRect(getX() - 1, getY() - 1, getWidth() + 1, getHeight() + 1);
            disableSelectionStyle(graphics);
        }

        for (Shape child : children) {
            child.paint(graphics);
        }
    }
}
```

### Demo.java: Initialization code

```
package refactoring_guru.memento.example;

import refactoring_guru.memento.example.editor.Editor;
import refactoring_guru.memento.example.shapes.Circle;
import refactoring_guru.memento.example.shapes.CompoundShape;
import refactoring_guru.memento.example.shapes.Dot;
import refactoring_guru.memento.example.shapes.Rectangle;

import java.awt.*;

public class Demo {
    public static void main(String[] args) {
        Editor editor = new Editor();
        editor.loadShapes(
```

```
                new CompoundShape(
                        new Circle(110, 110, 50, Color.RED),
                        new Dot(160, 160, Color.RED)
                ),

                new CompoundShape(
                        new Rectangle(250, 250, 100, 100, Color.GREEN),
                        new Dot(240, 240, Color.GREEN),
                        new Dot(240, 360, Color.GREEN),
                        new Dot(360, 360, Color.GREEN),
                        new Dot(360, 240, Color.GREEN)
                )
        );
    }
}
```

### OutputDemo.png: Screenshot




READ NEXT