Haven’s Light is Our Guide

Rajshahi University of Engineering and Technology

Department of Computer Science & Engineering

Lab Report

Course Title

Software Engineering Sessional

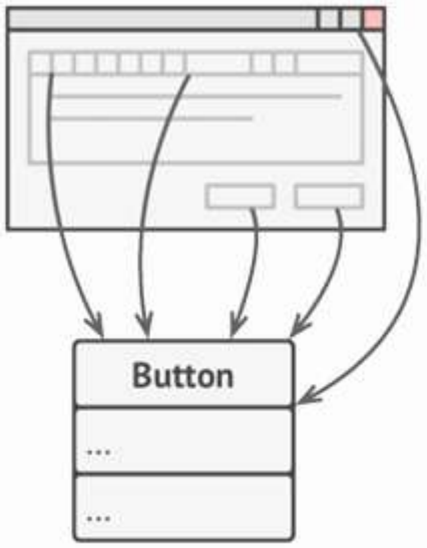
Course No.

CSE – 3206

Submitted by	**Submitted to**
1. Name: Mysha Ahmed Roll: 2003166 2. Name: Ananya Biswas Roll: 2003167 3. Name: Atia Fahmida Roll: 2003168	Farjana Parvin Lecturer, Dept of CSE,RUET

☹ Problem

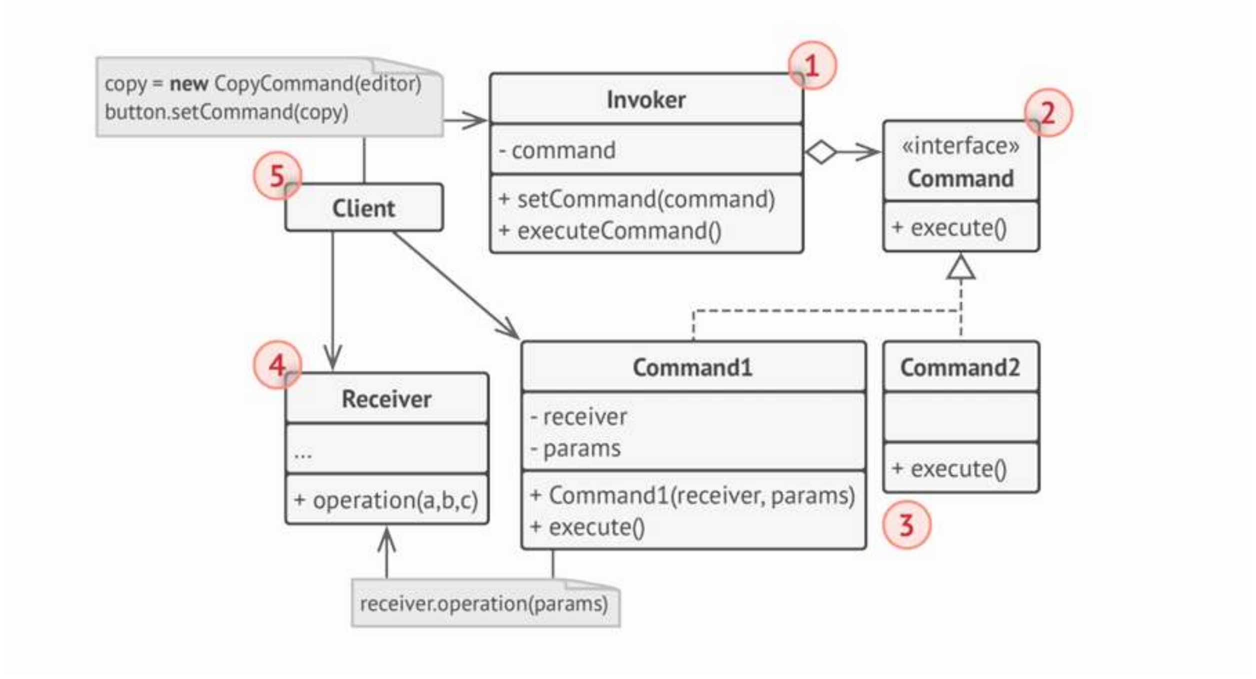
Imagine that you're working on a new text-editor app. Your current task is to create a toolbar with a bunch of buttons for various operations of the editor. You created a very neat `Button` class that can be used for buttons on the toolbar, as well as for generic buttons in various dialogs.



All buttons of the app are derived from the same class.

While all of these buttons look similar, they're all supposed to do different things. Where would you put the code for the various click handlers of these buttons? The simplest solution is to create tons of subclasses for each place where the button is used. These subclasses would contain the code that would have to be executed on a button click.

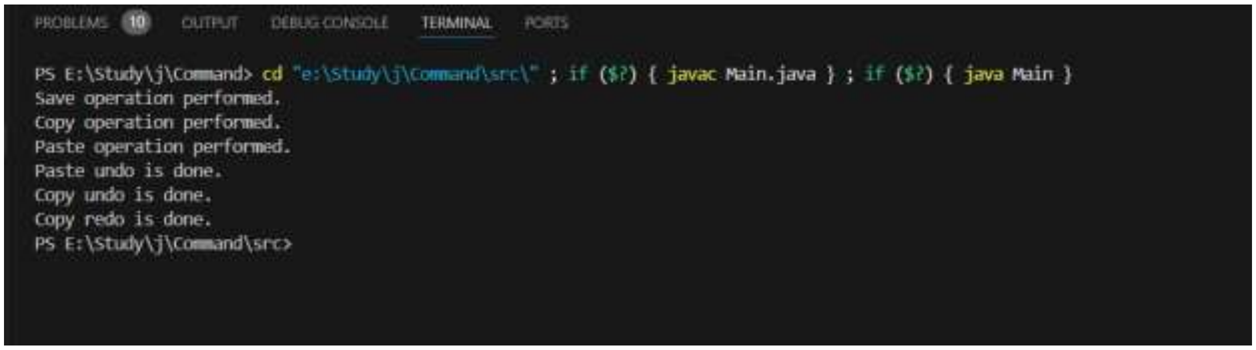
UML Diagram:


copy = **new** CopyCommand(editor)
button.setCommand(copy)
1
Invoker
- command
+ setCommand(command)
+ executeCommand()
2
«interface»
Command
+ execute()
5
Client
4
Receiver
...
+ operation(a,b,c)
Command1
- receiver
- params
+ Command1(receiver, params)
+ execute()
Command2
+ execute()
3
receiver.operation(params)

Code:

```java
//Abstract Class

public interface Command {

    public void execute();
    public void undo();
    public void redo();
}

//Copy Child Class

public class CopyCommand implements Command{
    TextEditor te;
    CopyCommand(TextEditor te){
        this.te=te;
    }
    @Override
    public void execute() {
        te.copy();
    }

    @Override
    public void undo() {
        te.copy_undo();
    }

    @Override
    public void redo() {
        te.copy_redo();
    }
}

//Paste Child

public class PasteCommand implements Command {
    TextEditor te;
    PasteCommand(TextEditor te){
        this.te=te;
    }
    @Override
    public void execute() {
        te.paste();
    }
    @Override
    public void undo() {
        te.paste_undo();
    }

    @Override
    public void redo() {
        te.paste_redo();
    }
}

//Save Child

public class SaveCommand implements Command {
    TextEditor te;
    SaveCommand(TextEditor te){
        this.te=te;
    }
    @Override
    public void execute() {
        te.save();
    }
    @Override
    public void undo() {
        te.save_undo();
    }

    @Override
    public void redo() {
        te.save_redo();
    }
}

//Invoker

import java.util.Stack;

public class Control {
    Stack<Command> undoHistory=new Stack<>();
    Stack<Command> redoHistory=new Stack<>();
    Command command;
    Control(){}

    public void setCommand(Command command){
        this.command=command;
    }

    public void pressButton(){
        command.execute();
        undoHistory.add(command);
    }

    public void undo(){
        if(!undoHistory.isEmpty())
        {
            Command lastCommand=undoHistory.pop();
            redoHistory.add(lastCommand);
            lastCommand.undo();

        }
        else{
            System.out.println("Nothing to undo");
        }
    }

    public void redo(){
        if(!redoHistory.isEmpty())
        {
            Command lastCommand=redoHistory.pop();
            undoHistory.add(lastCommand);
            lastCommand.redo();
        }
        else{
            System.out.println("Nothing to redo");
        }
    }
}

//Reciever

public class TextEditor {
    public void copy(){
        System.out.println("Copy operation performed.");
    }
    public void copy_undo(){
        System.out.println("Copy undo is done.");
    }
    public void copy_redo(){
        System.out.println("Copy redo is done.");
    }
    public void paste(){
        System.out.println("Paste operation performed.");
    }
    public void paste_undo(){
        System.out.println("Paste undo is done.");
    }
    public void paste_redo(){
        System.out.println("Paste redo is done.");
    }
    public void save(){
        System.out.println("Save operation performed.");
    }
    public void save_undo(){
        System.out.println("Save undo is done.");
    }
    public void save_redo(){
        System.out.println("Paste redo is done.");
    }
}

//Main Class
public class Main {
    public static void main(String[] args){
        TextEditor te=new TextEditor();

        Control control=new Control();

        control.setCommand(new SaveCommand(te));
        control.pressButton();
        control.setCommand(new CopyCommand(te));
        control.pressButton();
        control.setCommand(new PasteCommand(te));
        control.pressButton();

        //undo the last operation
        control.undo();
        control.undo();

        //redo the last operation
        control.redo();
    }
}
```

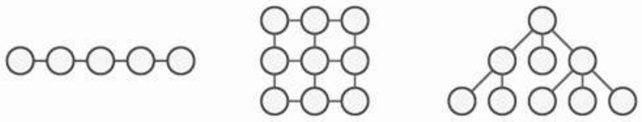
Output:



```
PS E:\Study\j\Command> cd "e:\Study\j\Command\src\" ; if ($?) { javac Main.java } ; if ($?) { java Main }
Save operation performed.
Copy operation performed.
Paste operation performed.
Paste undo is done.
Copy undo is done.
Copy redo is done.
PS E:\Study\j\Command\src>
```

☹ Problem

Collections are one of the most used data types in programming. Nonetheless, a collection is just a container for a group of objects.



Various types of collections.

Most collections store their elements in simple lists. However, some of them are based on stacks, trees, graphs and other complex data structures.

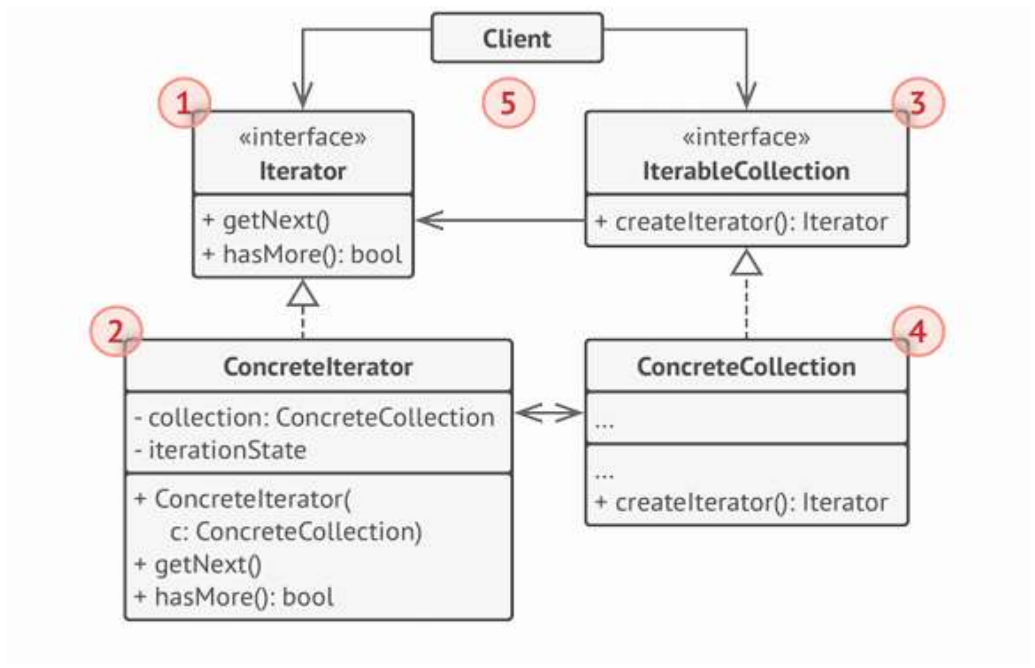
But no matter how a collection is structured, it must provide some way of accessing its elements so that other code can use these elements. There should be a way to go through each element of the collection without accessing the same elements over and over.

This may sound like an easy job if you have a collection based on a list. You just loop over all of the elements. But how do you sequentially traverse elements of a complex data structure, such as a tree? For example, one day you might be just fine

☺ Solution

The main idea of the Iterator pattern is to extract the traversal behavior of a collection into a separate object called an *iterator*.

UML Diagram:

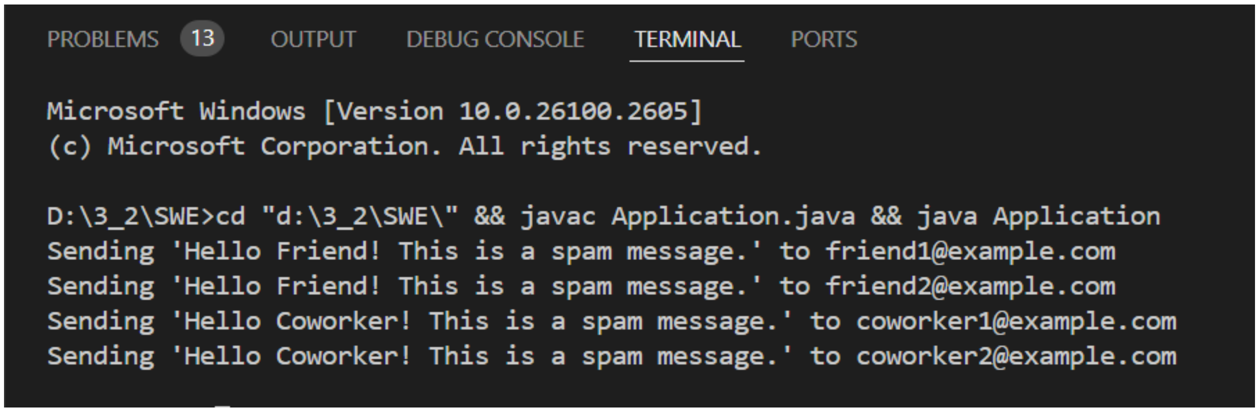


Code:

```
interface SocialNetwork {
    ProfileIterator createFriendsIterator(String profileId);
    ProfileIterator createCoworkersIterator(String profileId);
 }
 // Profile class
 class Profile {
    private String email;

    public Profile(String email) {
        this.email = email;
    }
    public String getEmail() {
        return email;
    }
 }
 // ProfileIterator interface
 interface ProfileIterator {
    boolean hasMore();
    Profile getNext();
 }
 // FacebookIterator class (Concrete Iterator)
 class FacebookIterator implements ProfileIterator {
    private Facebook facebook;
    private String profileId, type;
    private int currentPosition = 0;
    private Profile[] cache;
    public FacebookIterator(Facebook facebook, String profileId, String type) {
        this.facebook = facebook;
        this.profileId = profileId;
        this.type = type;
        this.lazyInit();
    }
    private void lazyInit() {
        if (cache == null) {
            cache = facebook.socialGraphRequest(profileId, type);
        }
    }
    @Override
    public boolean hasMore() {
        return currentPosition < cache.length;
    }
    @Override
    public Profile getNext() {
        if (this.hasMore()) {
            return cache[currentPosition++];
        }
        return null;
    }
 }
 // Facebook class (Concrete Collection)
 class Facebook implements SocialNetwork {
    @Override
    public ProfileIterator createFriendsIterator(String profileId) {
        return new FacebookIterator(this, profileId, "friends");
    }
    @Override
    public ProfileIterator createCoworkersIterator(String profileId) {
        return new FacebookIterator(this, profileId, "coworkers");
    }
    public Profile[] socialGraphRequest(String profileId, String type) {
        // Simulate social graph request: Returning dummy data
        if (type.equals("friends")) {
            return new Profile[]{new Profile("friend1@example.com"), new
Profile("friend2@example.com")};
        } else if (type.equals("coworkers")) {
            return new Profile[]{new Profile("coworker1@example.com"), new
Profile("coworker2@example.com")};
        }
        return new Profile[]{};
    }
 }
 // SocialSpammer class (Client)
 class SocialSpammer {
    public void send(ProfileIterator iterator, String message) {
        while (iterator.hasMore()) {
            Profile profile = iterator.getNext();
            System.out.println("Sending '" + message + "' to " + profile.getEmail());
        }
    }
 }
// Application class
```

```java
public class Application {
    private SocialNetwork network;
    private SocialSpammer spammer;
    public Application(SocialNetwork network) {
        this.network = network;
        this.spammer = new SocialSpammer();
    }
    public void sendSpamToFriends(String profileId) {
        ProfileIterator iterator = network.createFriendsIterator(profileId);
        spammer.send(iterator, "Hello Friend! This is a spam message.");
    }
    public void sendSpamToCoworkers(String profileId) {
        ProfileIterator iterator = network.createCoworkersIterator(profileId);
        spammer.send(iterator, "Hello Coworker! This is a spam message.");
    }
    public static void main(String[] args) {
        // Create the Facebook social network
        SocialNetwork facebook = new Facebook();
        // Initialize the application
        Application app = new Application(facebook);
        // Simulate spamming to friends and coworkers
        String userId = "user123";
        app.sendSpamToFriends(userId);
        app.sendSpamToCoworkers(userId);
    }
}
```

Output:



```
Microsoft Windows [Version 10.0.26100.2605]
(c) Microsoft Corporation. All rights reserved.

D:\3_2\SWE>cd "d:\3_2\SWE\" && javac Application.java && java Application
Sending 'Hello Friend! This is a spam message.' to friend1@example.com
Sending 'Hello Friend! This is a spam message.' to friend2@example.com
Sending 'Hello Coworker! This is a spam message.' to coworker1@example.com
Sending 'Hello Coworker! This is a spam message.' to coworker2@example.com
```