Homework 7, Structural Patterns

Marvin Sevilla

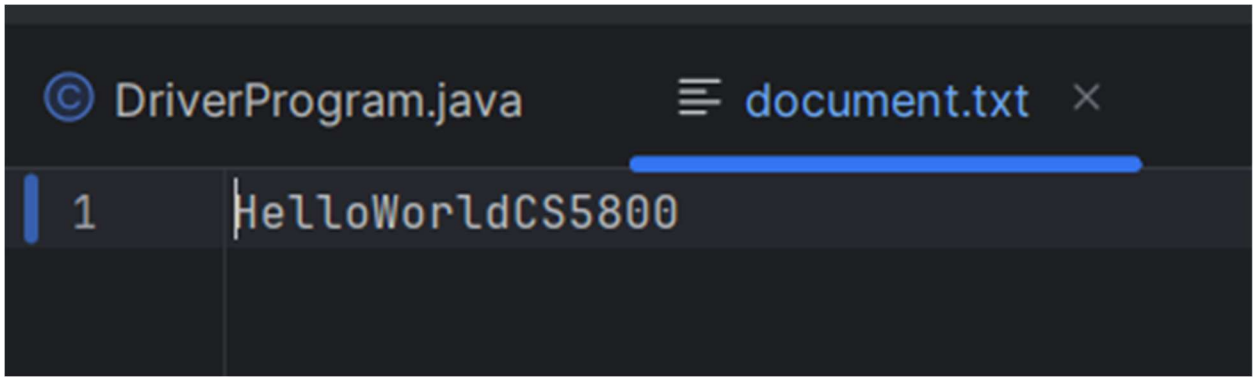
CS 4800.01, Software Engineering

GitHub Link: https://github.com/LoloMarty/2024-/tree/main/CS4800/Homework7

```
HelloWorldCS5800

Char: H
Font: Arial
Color: Red
Size: 12

Char: e
Font: Calibri
Color: Blue
Size: 14

Char: l
Font: Verdana
Color: Black
Size: 16

Char: l
Font: Roboto
Color: White
Size: 12

Char: o
Font: Arial
Color: Red
Size: 12

Char: W
Font: Arial
Color: Red
Size: 12

Char: o
Font: Calibri
Color: Blue
Size: 14
```

```
Char: r
Font: Verdana
Color: Black
Size: 16

Char: l
Font: Roboto
Color: White
Size: 12

Char: d
Font: Arial
Color: Red
Size: 12

Char: C
Font: Arial
Color: Red
Size: 12

Char: S
Font: Calibri
Color: Blue
Size: 14

Char: 5
Font: Verdana
Color: Black
Size: 16

Char: 8
Font: Roboto
Color: White
Size: 12

Char: 0
Font: Arial
Color: Red
Size: 12
```

```
Char: 0
Font: Arial
Color: Red
Size: 12


Process finished with exit code 0
```



Part 1 Source Code

```java
import org.junit.Test;

import static org.junit.Assert.*;

public class CharacterAttributesFactoryTest {

    @Test
    public void getCharacterProperties() {
        String font = "Arial";
        String color = "black";
        int size = 12;

        CharacterAttributes attributes1 = CharacterAttributesFactory.getCharacterProperties(font, color,
        CharacterAttributes attributes2 = CharacterAttributesFactory.getCharacterProperties(font, color,

        assertNotNull(attributes1);
        assertNotNull(attributes2);
        assertEquals(attributes1, attributes2);
    }
}
```

```java
import org.junit.Test;

import static org.junit.Assert.*;

public class CharacterAttributesTest {

    @Test
    public void getFont() {
        String expectedFont = "Arial";
        String color = "Black";
        int size = 12;
        CharacterAttributes attributes = new CharacterAttributes(expectedFont, color, size);

        String actualFont = attributes.getFont();

        assertEquals(expectedFont, actualFont);
    }

    @Test
    public void getColor() {
        CharacterAttributes attributes = new CharacterAttributes("Arial", "Red", 12);

        String expectedColor = "Red";
        String actualColor = attributes.getColor();
        assertEquals(expectedColor, actualColor);
    }

    @Test
    public void getSize() {
        CharacterAttributes attributes = new CharacterAttributes("Arial", "Black", 12);

        int expectedSize = 12;
        int actualSize = attributes.getSize();
        assertEquals(expectedSize, actualSize);
    }
}
```

```
interface CharacterInterface {
        void apply();
}
```

```java
import org.junit.Test;

import static org.junit.Assert.*;

public class CharacterTest {

    @Test
    public void setAttributes() {
        String initialFont = "Arial";
        String initialColor = "Red";
        int initialSize = 12;
        CharacterAttributes initialAttributes = new CharacterAttributes(initialFont, initialColor, initi
        Character character = new Character("A", initialFont, initialColor, initialSize);

        String newFont = "Calibri";
        String newColor = "Blue";
        int newSize = 14;
        CharacterAttributes newAttributes = new CharacterAttributes(newFont, newColor, newSize);

        character.setAttributes(newAttributes);

        assertEquals(newFont, character.getAttributes().getFont());
        assertEquals(newColor, character.getAttributes().getColor());
        assertEquals(newSize, character.getAttributes().getSize());
    }
}
```

```java
import java.util.ArrayList;
import java.io.FileWriter;
import java.io.IOException;
import java.io.FileReader;
import java.io.BufferedReader;

public class CharString {
    ArrayList<Character> string;
    String fileName;

    public CharString() {
        string = new ArrayList<Character>();
        fileName = "document.txt"; // Name of the file to write to
    }

    public void save(String givenCharacter, String givenFont, String givenColor, int givenSize) {
        string.add(new Character(givenCharacter, givenFont, givenColor, givenSize));

        try {
            FileWriter writer = new FileWriter(fileName);
            writer.write(this.buildString());
            writer.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public String buildString() {
        String builtString = "";
        for (Character character : this.string) {
            builtString += character.getHeldCharacter();
        }

        return builtString;
    }

    public void load() {
        try {
            FileReader reader = new FileReader(fileName);
            BufferedReader bufferedReader = new BufferedReader(reader);

            String line;
            while ((line = bufferedReader.readLine()) != null) {
                System.out.println("\n" + line + "\n");
            }

            bufferedReader.close();
        } catch (IOException e) {
            System.out.println("An error occurred while reading the file.");
            e.printStackTrace();
        }

        for (Character character : this.string) {
```

```
54              System.out.printf("Char: %s\n", character.getHeldCharacter());
55              character.printCharacaterAttributes();
56              System.out.println("\n");
57          }
58      }
59  }
```

```java
import org.junit.Test;
import static org.junit.Assert.*;
import java.io.File;

public class CharStringTest {

    @Test
    public void save() {
        CharString charString = new CharString();

        charString.save("A", "Arial", "Black", 12);

        File file = new File("document.txt");
        assertTrue(file.exists());

        String expectedContent = "A";
        assertEquals(expectedContent, charString.buildString());
    }

    @Test
    public void buildString() {
        CharString charString = new CharString();
        charString.save("A", "Arial", "Black", 12);
        charString.save("B", "Times New Roman", "Red", 14);

        assertEquals("AB", charString.buildString());
    }
}
```

```java
public class Disk {
    static CharString document;

    public static CharString getDocument() {
        if (document == null) {
            document = new CharString();
        }

        return document;
    }
}
```

```java
import org.junit.Test;

import static org.junit.Assert.*;

public class DiskTest {

    @Test
    public void getDocument() {
        CharString expected = Disk.getDocument();
        CharString actual = Disk.getDocument();

        assertNotNull(actual);
        assertEquals(expected, actual);
    }
}
```

```java
public class DriverProgram {
    public static void main(String[] args) {
        CharString document = Disk.getDocument();

        document.save("H", "Arial", "Red", 12);
        document.save("e", "Calibri", "Blue", 14);
        document.save("l", "Verdana", "Black", 16);
        document.save("l", "Roboto", "White", 12);
        document.save("o", "Arial", "Red", 12);
        document.save("W", "Arial", "Red", 12);
        document.save("o", "Calibri", "Blue", 14);
        document.save("r", "Verdana", "Black", 16);
        document.save("l", "Roboto", "White", 12);
        document.save("d", "Arial", "Red", 12);
        document.save("C", "Arial", "Red", 12);
        document.save("S", "Calibri", "Blue", 14);
        document.save("5", "Verdana", "Black", 16);
        document.save("8", "Roboto", "White", 12);
        document.save("0", "Arial", "Red", 12);
        document.save("0", "Arial", "Red", 12);

        document.load();

    }
}
```

```
1  public class Character {
2      String heldCharacter;
3      CharacterAttributes attributes;
4
5      public Character(String givenCharacter, String givenFont, String givenColor, int givenSize) {
6          this.heldCharacter = givenCharacter;
7
8          CharacterAttributes fetchedAttributes = CharacterAttributesFactory.getCharacterProperties(givenF
9                  givenColor, givenSize);
10
11         if (fetchedAttributes != null) {
12             this.attributes = fetchedAttributes;
13         } else {
14             this.attributes = new CharacterAttributes(givenFont, givenColor, givenSize);
15         }
16     }
17
18     public void printCharacaterAttributes() {
19         this.attributes.apply();
20     }
21
22     public CharacterAttributes getAttributes() {
23         return attributes;
24     }
25
26     public void setAttributes(CharacterAttributes attributes) {
27         this.attributes = attributes;
28     }
29
30     public String getHeldCharacter() {
31         return heldCharacter;
32     }
33 }
```

```java
public class CharacterAttributes implements CharacterInterface {
    private final String font;
    private final String color;
    private final int size;

    public CharacterAttributes(String givenFont, String givenColor, int givenSize) {
        this.font = givenFont;
        this.color = givenColor;
        this.size = givenSize;
    }

    public void apply() {
        System.out.printf("Font: %s\nColor: %s\nSize: %d", this.font, this.color, this.size);
    }

    public String getFont() {
        return font;
    }

    public String getColor() {
        return color;
    }

    public int getSize() {
        return size;
    }
}
```

```
public class DriverProgram {
    public static void main(String[] args) {
        CharString document = Disk.getDocument();

        document.save("H", "Arial", "Red", 12);
        document.save("e", "Calibri", "Blue", 14);
        document.save("l", "Verdana", "Black", 16);
        document.save("l", "Roboto", "White", 12);
        document.save("o", "Arial", "Red", 12);
        document.save("W", "Arial", "Red", 12);
        document.save("o", "Calibri", "Blue", 14);
        document.save("r", "Verdana", "Black", 16);
        document.save("l", "Roboto", "White", 12);
        document.save("d", "Arial", "Red", 12);
        document.save("C", "Arial", "Red", 12);
        document.save("S", "Calibri", "Blue", 14);
        document.save("5", "Verdana", "Black", 16);
        document.save("8", "Roboto", "White", 12);
        document.save("0", "Arial", "Red", 12);
        document.save("0", "Arial", "Red", 12);

        document.loadimport java.util.HashMap;
import java.util.Map;

public class CharacterAttributesFactory {
    private static Map<String, CharacterAttributes> propertiesMap = new HashMap<>();

    public static CharacterAttributes getCharacterProperties(String font, String color, int size) {
        String key = font + color + size;
        if (!propertiesMap.containsKey(key)) {
            propertiesMap.put(key, new CharacterAttributes(font, color, size));
        }
        return propertiesMap.get(key);
    }

}
();

    }
}
```

```
Song not cached, fetching from server...


Name: Paint It Blue
Album: Cowboy Jams
Artist: Charley Crockett
SongID: 1
Duration (s): 120



Name: Paint It Blue
Album: Cowboy Jams
Artist: Charley Crockett
SongID: 1
Duration (s): 120

Song not cached, fetching from server...


Name: Free Bird
Album: Patriotic Jams
Artist: Lynrd Skynyrd
SongID: 6
Duration (s): 600


Process finished with exit code 0
```

Part 2 Source Code

```java
import org.junit.Test;

import java.util.LinkedList;
import java.util.List;

import static org.junit.Assert.*;

public class ServerTest {

    @Test
    public void addSong() {
        Server server = Server.getInstance();
        server.addSong("Title1", "Artist1", "Album1", 1, 180);

        Song searchedSong = server.searchById(1);
        assertNotNull(searchedSong);
        assertEquals("Title1", searchedSong.getTitle());
        assertEquals("Artist1", searchedSong.getArtist());
        assertEquals("Album1", searchedSong.getAlbum());
        assertEquals(180, searchedSong.getDuration());

        List<Song> songsByTitle = server.searchByTitle("Title1");
        assertNotNull(songsByTitle);
        assertEquals(1, songsByTitle.size());
        assertEquals("Title1", songsByTitle.get(0).getTitle());

        List<Song> songsByAlbum = server.searchByAlbum("Album1");
        assertNotNull(songsByAlbum);
        assertEquals(1, songsByAlbum.size());
        assertEquals("Album1", songsByAlbum.get(0).getAlbum());
    }

    @Test
    public void searchById() {
        Server server = Server.getInstance();

        Song testSong = new Song("Test Song", "Test Artist", "Test Album", -1, 120);
        server.addSong("Test Song", "Test Artist", "Test Album", -1, 120);

        Song result = server.searchById(-1);

        assertEquals(testSong.getTitle(), result.getTitle());
    }

    @Test
    public void searchByTitle() {
        Server server = Server.getInstance();

        List<Song> testSong = new LinkedList<Song>();
        testSong.addFirst(new Song("Test Song", "Test Artist", "Test Album", 1, 120));
        server.addSong("Test Song", "Test Artist", "Test Album", 1, 120);

        List<Song> result = server.searchByTitle("Test Song");
```

```
54
55          assertEquals(testSong.getFirst().getTitle(), result.getFirst().getTitle());
56      }
57
58      @Test
59      public void searchByAlbum() {
60          Server server = Server.getInstance();
61
62          List<Song> testSong = new LinkedList<Song>();
63          testSong.addFirst(new Song("Test Song", "Test Artist", "Test Album", 1, 120));
64          server.addSong("Test Song", "Test Artist", "Test Album", 1, 120);
65
66          List<Song> result = server.searchByAlbum("Test Album");
67
68          assertEquals(testSong.getFirst().getTitle(), result.getFirst().getTitle());
69      }
70
71      @Test
72      public void getInstance() {
73          Server instance1 = Server.getInstance();
74          Server instance2 = Server.getInstance();
75
76          assertSame(instance1, instance2);
77
78          assertNotNull(instance1);
79          assertNotNull(instance2);
80      }
81  }
```

```java
import java.util.List;

public interface SongService {
    Song searchById(Integer songID);
    List<Song> searchByTitle(String title);
    List<Song> searchByAlbum(String album);
}
```

```java
public class Song {
    private final String title;
    private final String artist;
    private final String album;
    private final int id;
    private final int duration;

    public Song(String givenTitle, String givenArtist, String givenAlbum, int givenID, int givenDuration
        this.title = givenTitle;
        this.artist = givenArtist;
        this.album = givenAlbum;
        this.id = givenID;
        this.duration = givenDuration;
    }

    public String getTitle() {
        return title;
    }

    public String getArtist() {
        return artist;
    }

    public String getAlbum() {
        return album;
    }

    public int getId() {
        return id;
    }

    public int getDuration() {
        return duration;
    }

}
```

```java
import java.util.ArrayList;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;

public class Server implements SongService {
    private static Server instance;
    private static HashMap<String, List<Song>> titleHashmap;
    private static HashMap<String, List<Song>> albumHashmap;
    private static HashMap<String, Song> idHashmap;

    private final int waitTime = 3000;

    private Server() {
        titleHashmap = new HashMap<String, List<Song>>();
        albumHashmap = new HashMap<String, List<Song>>();
        idHashmap = new HashMap<String, Song>();
    }

    public void addSong(String title, String artist, String album, int id, int Duration) {
        if (titleHashmap == null) {
            titleHashmap = new HashMap<String, List<Song>>();
        }
        if (albumHashmap == null) {
            albumHashmap = new HashMap<String, List<Song>>();
        }
        if (idHashmap == null) {
            idHashmap = new HashMap<String, Song>();
        }

        Song songToAdd = new Song(title, artist, album, id, Duration);

        if (titleHashmap.get(title) == null) {
            titleHashmap.put(title, new LinkedList<Song>());
        }
        titleHashmap.get(title).addFirst(songToAdd);

        if (albumHashmap.get(album) == null) {
            albumHashmap.put(album, new LinkedList<Song>());
        }
        albumHashmap.get(album).addFirst(songToAdd);

        if (idHashmap.get(Integer.toString(id)) == null) {
            idHashmap.put(Integer.toString(id), songToAdd);
        }
    }

    public Song searchById(Integer songID) {
        try {
            Thread.sleep(waitTime);
        } catch (Exception e) {
            e.printStackTrace();
        }
```

```
54
55          return idHashmap.get(Integer.toString(songID));
56      }
57
58      public List<Song> searchByTitle(String title) {
59          try {
60              Thread.sleep(waitTime);
61          } catch (Exception e) {
62              e.printStackTrace();
63          }
64
65          return titleHashmap.get(title);
66      }
67
68      public List<Song> searchByAlbum(String album) {
69          try {
70              Thread.sleep(waitTime);
71          } catch (Exception e) {
72              e.printStackTrace();
73          }
74
75          return albumHashmap.get(album);
76      }
77
78      public static Server getInstance() {
79          if (instance == null) {
80              instance = new Server();
81          }
82
83          return instance;
84      }
85  }
```

```java
import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;

public class ProxyServer implements SongService {
    private static ProxyServer proxyServer;
    private static HashMap<String, List<Song>> titleHashmap;
    private static HashMap<String, List<Song>> albumHashmap;
    private static HashMap<String, Song> idHashmap;
    private Song songResult;
    private List<Song> listSongResult;

    private ProxyServer() {
        titleHashmap = new HashMap<String, List<Song>>();
        albumHashmap = new HashMap<String, List<Song>>();
        idHashmap = new HashMap<String, Song>();
    }

    public Song searchById(Integer songID) {
        this.songResult = idHashmap.get(Integer.toString(songID));

        if (songResult == null) {
            System.out.println("Song not cached, fetching from server...");
            Song retrievedSong = Server.getInstance().searchById(songID);
            idHashmap.put(Integer.toString(songID), retrievedSong);
        }

        return idHashmap.get(Integer.toString(songID));
    }

    public List<Song> searchByTitle(String title) {
        this.listSongResult = titleHashmap.get(title);

        if (listSongResult == null) {
            System.out.println("Song not cached, fetching from server...");
            List<Song> retrievedSong = Server.getInstance().searchByTitle(title);
            titleHashmap.put(title, retrievedSong);
        }

        return titleHashmap.get(title);
    }

    public List<Song> searchByAlbum(String album) {
        this.listSongResult = albumHashmap.get(album);

        if (listSongResult == null) {
            System.out.println("Song not cached, fetching from server...");
            List<Song> retrievedSong = Server.getInstance().searchByAlbum(album);
            albumHashmap.put(album, retrievedSong);
        }

        return albumHashmap.get(album);
    }
```

```
54
55      public static ProxyServer getInstance() {
56          if (proxyServer == null) {
57              proxyServer = new ProxyServer();
58          }
59
60          return proxyServer;
61      }
62
63  }
```

```java
import java.sql.Driver;
import java.util.List;

public class DriverProgram {
    public static void printSongInfo(Song song) {
        System.out.println();
        System.out.printf("\nName: %s", song.getTitle());
        System.out.printf("\nAlbum: %s", song.getAlbum());
        System.out.printf("\nArtist: %s", song.getArtist());
        System.out.printf("\nSongID: %d", song.getId());
        System.out.printf("\nDuration (s): %d\n\n", song.getDuration());
    }

    public static void printSongListInfo(List<Song> listOfSongs) {
        for (Song song : listOfSongs) {
            printSongInfo(song);
        }
    }

    public static void main(String[] args) {
        Server server = Server.getInstance();
        ProxyServer proxyServer = ProxyServer.getInstance();

        server.addSong("Paint It Blue", "Charley Crockett", "Cowboy Jams", 1, 120);
        server.addSong("Separate Ways", "Journey", "80s Hits", 2, 240);
        server.addSong("Life is a Highway", "Pixar", "Cars Movie OST", 3, 120);
        server.addSong("21", "Sam Hunt", "Sad Country", 4, 120);
        server.addSong("Paint It Black", "The Animals", "Vietnam War Music", 5, 240);
        server.addSong("Free Bird", "Lynrd Skynyrd", "Patriotic Jams", 6, 600);

        List<Song> returnedSongs = proxyServer.searchByTitle("Paint It Blue");
        // Slow Retrieval
        DriverProgram.printSongListInfo(returnedSongs);

        returnedSongs = proxyServer.searchByTitle("Paint It Blue");
        // Fast Retrieval (Cached)
        DriverProgram.printSongListInfo(returnedSongs);

        Song returnedSong = proxyServer.searchById(6);
        // Fast Retrieval (Cached)
        DriverProgram.printSongInfo(returnedSong);

    }
}
```

```java
import static org.junit.Assert.*;
import java.util.*;
public class ProxyServerTest {

    @org.junit.Test
    public void searchById() {
        ProxyServer proxyServer = ProxyServer.getInstance();
        Server server = Server.getInstance();

        Song testSong = new Song("Test Song", "Test Artist", "Test Album", 1, 120);
        server.addSong("Test Song", "Test Artist", "Test Album", 1, 120);

        Song result = proxyServer.searchById(1);

        assertEquals(testSong.getTitle(), result.getTitle());
    }

    @org.junit.Test
    public void searchByTitle() {
        ProxyServer proxyServer = ProxyServer.getInstance();
        Server server = Server.getInstance();


        List<Song> testSong = new LinkedList<Song>();
        testSong.addFirst(new Song("Test Song", "Test Artist", "Test Album", 1, 120));
        server.addSong("Test Song", "Test Artist", "Test Album", 1, 120);

        List<Song> result = proxyServer.searchByTitle("Test Song");

        assertEquals(testSong.getFirst().getTitle(), result.getFirst().getTitle());
    }

    @org.junit.Test
    public void searchByAlbum() {
        ProxyServer proxyServer = ProxyServer.getInstance();
        Server server = Server.getInstance();

        List<Song> testSong = new LinkedList<Song>();
        testSong.addFirst(new Song("Test Song", "Test Artist", "Test Album", 1, 120));
        server.addSong("Test Song", "Test Artist", "Test Album", 1, 120);

        List<Song> result = proxyServer.searchByAlbum("Test Album");

        assertEquals(testSong.getFirst().getTitle(), result.getFirst().getTitle());
    }

    @org.junit.Test
    public void getInstance() {
        ProxyServer instance1 = ProxyServer.getInstance();
        ProxyServer instance2 = ProxyServer.getInstance();

        assertSame(instance1, instance2);

```

```
54          assertNotNull(instance1);
55          assertNotNull(instance2);
56      }
57  }
```

```java
import org.junit.Test;

import static org.junit.Assert.*;

public class SongTest {

    @Test
    public void getTitle() {
        String givenTitle = "Bohemian Rhapsody";
        String givenArtist = "Queen";
        String givenAlbum = "A Night at the Opera";
        int givenID = 1;
        int givenDuration = 355;

        Song song = new Song(givenTitle, givenArtist, givenAlbum, givenID, givenDuration);

        assertEquals(givenTitle, song.getTitle());
    }

    @Test
    public void getArtist() {
        String expectedArtist = "Ed Sheeran";
        Song song = new Song("Shape of You", expectedArtist, "÷", 1, 234);

        String actualArtist = song.getArtist();

        assertEquals(expectedArtist, actualArtist);
    }

    @Test
    public void getAlbum() {
        Song song = new Song("Title", "Artist", "Album", 1, 180);
        assertEquals("Album", song.getAlbum());
    }

    @Test
    public void getId() {
        String givenTitle = "Song Title";
        String givenArtist = "Artist Name";
        String givenAlbum = "Album Title";
        int givenID = 123;
        int givenDuration = 240;

        Song song = new Song(givenTitle, givenArtist, givenAlbum, givenID, givenDuration);

        assertEquals(givenID, song.getId());
    }

    @Test
    public void getDuration() {
        int expectedDuration = 240;

        Song song = new Song("Title", "Artist", "Album", 1, expectedDuration);
```

```
54            assertEquals(expectedDuration, song.getDuration());
55        }
56    }
```
