**Course: ENSF 614–Fall2021**

**Lab #: Lab 7**

**Student Names: Graydon Hall, Jared Kraus**

**Submission Date: 2021-10-25**

# Exercise A

## Output:

Graphical user interface, application, Word

Description automatically generated

## Source Code

|  |
| --- |
| */\* File Name: Text.java  \* Lab # and Assignment #: Lab #7  \* Lab section: 1  \* Completed by: Graydon Hall and Jared Kraus  \* Submission Date: 2021-11-22s  \*/  package* ExB;  *import* java.awt.\*;  *public class* Text *implements* Component {  *int* x;  *int* y;  String text;   *public* Text(String text, *int* x, *int* y) {  *this*.x = x;  *this*.y = y;  *this*.text = text;  }   *@Override  public void* draw(Graphics g) {  *var* oldColor = g.getColor();  Color dark\_green = *new* Color(0,102,0);  g.setColor(dark\_green);  g.drawString(text, x, y);  g.setColor(oldColor);  } } |
| */\* File Name: BorderDecorator.java  \* Lab # and Assignment #: Lab #7  \* Lab section: 1  \* Completed by: Graydon Hall and Jared Kraus  \* Submission Date: 2021-11-22  \*/  package* ExB;  *import* java.awt.\*;  *public class* BorderDecorator *extends* Decorator {   *public* BorderDecorator(Component component, *int* x, *int* y, *int* width, *int* height) {  *super*(component, x, y, width, height);  }   *@Override  public void* draw(Graphics g) {  component.draw(g);  Graphics2D g2d = (Graphics2D) g;  Stroke dashed = *new* BasicStroke(3, BasicStroke.CAP\_BUTT, BasicStroke.JOIN\_BEVEL, 0, *new float*[]{9}, 0);  g2d.setStroke(dashed);  g2d.drawRect(x, y, width, height);  } } |
| */\* File Name: ColouredFrameDecorator.java  \* Lab # and Assignment #: Lab #7  \* Lab section: 1  \* Completed by: Graydon Hall and Jared Kraus  \* Submission Date: 2021-11-22  \*/  package* ExB;  *import* java.awt.\*;  *public class* ColouredFrameDecorator *extends* Decorator {  *private int* thickness;   *public* ColouredFrameDecorator(Component component, *int* x, *int* y, *int* width, *int* height, *int* t) {  *super*(component, x, y, width, height);  *this*.thickness = t;  }   *@Override  public void* draw(Graphics g) {  component.draw(g);  Graphics2D g2d = (Graphics2D) g;  *var* oldStroke = g2d.getStroke();  *var* oldColour = g2d.getColor();  g2d.setStroke(*new* BasicStroke(thickness));  g2d.setColor(Color.RED);  g2d.drawRect(x, y, width, height);  g2d.setStroke(oldStroke);  g2d.setColor(oldColour);  } } |
| */\* File Name: ColouredGlassDecorator.java  \* Lab # and Assignment #: Lab #7  \* Lab section: 1  \* Completed by: Graydon Hall and Jared Kraus  \* Submission Date: 2021-11-22  \*/  package* ExB;  *import* java.awt.\*;  *public class* ColouredGlassDecorator *extends* Decorator {   *public* ColouredGlassDecorator(Component component, *int* x, *int* y, *int* width, *int* height) {  *super*(component, x, y, width, height);  }   *@Override  public void* draw(Graphics g) {  component.draw(g);  Graphics2D g2d = (Graphics2D) g;  *var* oldColour = g2d.getColor();  *var* oldComposite = g2d.getComposite();  g2d.setColor(Color.GREEN);  g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC\_OVER, 1 \* 0.1f));  g2d.fillRect(25, 25, 110, 110);  g2d.setColor(oldColour);  g2d.setComposite(oldComposite);  } } |
| */\* File Name: Component.java  \* Lab # and Assignment #: Lab #7  \* Lab section: 1  \* Completed by: Graydon Hall and Jared Kraus  \* Submission Date: 2021-11-22  \*/  package* ExB;  *import* java.awt.\*;  *public interface* Component {  *public void* draw(Graphics g); } |
| */\* File Name: Decorator.java  \* Lab # and Assignment #: Lab #7  \* Lab section: 1  \* Completed by: Graydon Hall and Jared Kraus  \* Submission Date: 2021-11-22  \*/  package* ExB;  *public abstract class* Decorator *implements* Component {   Component component;  *int* x;  *int* y;  *int* width;  *int* height;    *public* Decorator(Component component, *int* x, *int* y, *int* width, *int* height) {  *this*.component = component;  *this*.x = x;  *this*.y = y;  *this*.width = width;  *this*.height = height;  }    } |
| */\* File Name: Text.java  \* Lab # and Assignment #: Lab #7  \* Lab section: 1  \* Completed by: Graydon Hall and Jared Kraus  \* Submission Date: 2021-11-22s  \*/  package* ExB;  *import* java.awt.\*;  *public class* Text *implements* Component {  *private int* x;  *private int* y;  *private* String text;   *public* Text(String text, *int* x, *int* y) {  *this*.x = x;  *this*.y = y;  *this*.text = text;  }   *@Override  public void* draw(Graphics g) {  *var* oldColor = g.getColor();  Color dark\_green = *new* Color(0,102,0);  g.setColor(dark\_green);  g.drawString(text, x, y);  g.setColor(oldColor);  } } |

# Exercise B

## Output

Graphical user interface, application

Description automatically generated

## Source Code

|  |
| --- |
| */\* File Name: DemoDecoratorPattern.java  \* Lab # and Assignment #: Lab #7  \* Lab section: 1  \* Completed by: Graydon Hall and Jared Kraus  \* Submission Date: 2021-11-22  \*/  package* ExB;  *import* javax.swing.\*; *import* java.awt.\*;  *public class* DemoDecoratorPattern *extends* JPanel {  Component t;   *public* DemoDecoratorPattern(){  t = *new* Text("Hello World", 60, 80);  }   *public void* paintComponent(Graphics g){  *int* fontSize = 10;  g.setFont(*new* Font("TimesRoman", Font.PLAIN, fontSize));  *// GlassFrameDecorator info: x = 25, y = 25, width = 110, and height = 110* t = *new* ColouredGlassDecorator(*new* ColouredFrameDecorator(  *new* BorderDecorator(t, 30, 30, 100, 100), 25, 25, 110, 110, 10), 25, 25,  110, 110);  t.draw(g);  }    *public static void* main(String[] args) {   DemoDecoratorPattern panel = *new* DemoDecoratorPattern();  JFrame frame = *new* JFrame("Learning Decorator Pattern");  frame.getContentPane().add(panel);  frame.setSize(400,400);  frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  frame.setLocationRelativeTo(*null*);  frame.setVisible(*true*);  } } |
| */\* File Name: BorderDecorator.java  \* Lab # and Assignment #: Lab #7  \* Lab section: 1  \* Completed by: Graydon Hall and Jared Kraus  \* Submission Date: 2021-11-22  \*/  package* ExB;  *import* java.awt.\*;  *public class* BorderDecorator *extends* Decorator {   *public* BorderDecorator(Component component, *int* x, *int* y, *int* width, *int* height) {  *super*(component, x, y, width, height);  }   *@Override  public void* draw(Graphics g) {  component.draw(g);  Graphics2D g2d = (Graphics2D) g;  Stroke dashed = *new* BasicStroke(3, BasicStroke.CAP\_BUTT, BasicStroke.JOIN\_BEVEL, 0, *new float*[]{9}, 0);  g2d.setStroke(dashed);  g2d.drawRect(x, y, width, height);  } } |
| */\* File Name: ColouredFrameDecorator.java  \* Lab # and Assignment #: Lab #7  \* Lab section: 1  \* Completed by: Graydon Hall and Jared Kraus  \* Submission Date: 2021-11-22  \*/  package* ExB;  *import* java.awt.\*;  *public class* ColouredFrameDecorator *extends* Decorator {  *private int* thickness;   *public* ColouredFrameDecorator(Component component, *int* x, *int* y, *int* width, *int* height, *int* t) {  *super*(component, x, y, width, height);  *this*.thickness = t;  }   *@Override  public void* draw(Graphics g) {  component.draw(g);  Graphics2D g2d = (Graphics2D) g;  *var* oldStroke = g2d.getStroke();  *var* oldColour = g2d.getColor();  g2d.setStroke(*new* BasicStroke(thickness));  g2d.setColor(Color.RED);  g2d.drawRect(x, y, width, height);  g2d.setStroke(oldStroke);  g2d.setColor(oldColour);  } } |
| */\* File Name: ColouredGlassDecorator.java  \* Lab # and Assignment #: Lab #7  \* Lab section: 1  \* Completed by: Graydon Hall and Jared Kraus  \* Submission Date: 2021-11-22  \*/  package* ExB;  *import* java.awt.\*;  *public class* ColouredGlassDecorator *extends* Decorator {   *public* ColouredGlassDecorator(Component component, *int* x, *int* y, *int* width, *int* height) {  *super*(component, x, y, width, height);  }   *@Override  public void* draw(Graphics g) {  component.draw(g);  Graphics2D g2d = (Graphics2D) g;  *var* oldColour = g2d.getColor();  *var* oldComposite = g2d.getComposite();  g2d.setColor(Color.GREEN);  g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC\_OVER, 1 \* 0.1f));  g2d.fillRect(25, 25, 110, 110);  g2d.setColor(oldColour);  g2d.setComposite(oldComposite);  } } |
| */\* File Name: Component.java  \* Lab # and Assignment #: Lab #7  \* Lab section: 1  \* Completed by: Graydon Hall and Jared Kraus  \* Submission Date: 2021-11-22  \*/  package* ExB;  *import* java.awt.\*;  *public interface* Component {  *public void* draw(Graphics g); } |
| */\* File Name: Decorator.java  \* Lab # and Assignment #: Lab #7  \* Lab section: 1  \* Completed by: Graydon Hall and Jared Kraus  \* Submission Date: 2021-11-22  \*/  package* ExB;  *public abstract class* Decorator *implements* Component {   Component component;  *int* x;  *int* y;  *int* width;  *int* height;    *public* Decorator(Component component, *int* x, *int* y, *int* width, *int* height) {  *this*.component = component;  *this*.x = x;  *this*.y = y;  *this*.width = width;  *this*.height = height;  } } |
| */\* File Name: Text.java  \* Lab # and Assignment #: Lab #7  \* Lab section: 1  \* Completed by: Graydon Hall and Jared Kraus  \* Submission Date: 2021-11-22s  \*/  package* ExB;  *import* java.awt.\*;  *public class* Text *implements* Component {  *int* x;  *int* y;  String text;   *public* Text(String text, *int* x, *int* y) {  *this*.x = x;  *this*.y = y;  *this*.text = text;  }   *@Override  public void* draw(Graphics g) {  *var* oldColor = g.getColor();  Color dark\_green = *new* Color(0,102,0);  g.setColor(dark\_green);  g.drawString(text, x, y);  g.setColor(oldColor);  } } |

# Exercise C

## Output

Text

Description automatically generated

## Source Code

|  |
| --- |
| */\* File Name: main.cpp*  *\* Lab # and Assignment #: Lab #7*  *\* Lab section: 1*  *\* Completed by: Graydon Hall and Jared Kraus*  *\* Submission Date: 2021-11-22*  *\*/*  #include "Client\_A.hpp"  #include "Client\_B.hpp"  #include "User.hpp"  #include <iostream>  using namespace std;  int main() {      Client\_A ca;      cout << "Created a new Client\_A object called ca ..." << endl;        cout << "adding two usernames, Jack and Judy, by client ca ..." << endl;      ca.add("Jack", "apple5000");      ca.add("Judy", "orange$1234");        Client\_B cb;      cout << "Created a new Client\_B object called cb ... " << endl;      cout << "Adding two usernames called Jim and Josh, by client cb ..." << endl;      cb.add("Jim", "brooks$2017");      cb.add("Josh", "mypass2000");        cout << "Now adding another username called Jim by client ca.\n";      cout << "It must be avoided because a similar username already exits ..." << endl;      ca.add("Jim", "brooks$2017");      cout << "Another attempt to add username called Jim, but this time by client cb,\n";      cout << "with a different password\n";      cout << "It must be avoided again ..." << endl;      cb.add("Jim", "br$2017");        cout << "Now client cb validates existence of username Jack and his password: " << endl;      if( User \*u = cb.validate("Jack", "apple5000"))          cout << "Found: username: " << u->username << " and the password is: " << u->password <<  endl;      else          cout << "Username or password NOT found" << endl;      cout << "Now client ca validates existence of username Jack with a wrong password. " << endl;      if( User \*u = ca.validate("Jack", "apple4000"))          cout << "Found: username is: " << u->username << " and password is: " << u->password <<  endl;      else          cout << "Username or password NOT found" << endl;        cout << "Trying to make a new Client\_A object which is a copy of client ca:" << endl;      Client\_A ca2 = ca;      cout << "Adding a usernames called Tim by client ca2 ..." << endl;      cb.add("Tim", "blue\_sky");      cout << "Make a new Client\_A object called ca3:" << endl;      Client\_A ca3;      cout << "Make ca3 a copy of ca2:" << endl;      ca3 = ca2;      cout << "Now client ca3 validates existence of username Tim and his password: " << endl;      if( User \*u = ca3.validate("Tim", "blue\_sky"))          cout << "Found: username: " << u->username << " and the password is: " << u->password <<  endl;      else          cout << " Tim NOT found" << endl;  #if 0      cout << "Lets now make a couple of objects of LoginServer by main funciton:" << endl;      LoginServer x;      LoginServer y = x;      x = y;      cout << "Now LoginServer x validates existence of username Tim and his password: " << endl;      if( User \*u = y.validate("Tim", "blue\_sky"))          cout << "Found: username: " << u->username << " and the password is: " << u->password <<  endl;      else          cout << "Tim NOT found" << endl;  #endif        return 0;  } |
| */\* File Name: User.hpp*  *\* Lab # and Assignment #: Lab #7*  *\* Lab section: 1*  *\* Completed by: Graydon Hall and Jared Kraus*  *\* Submission Date: 2021-11-22*  *\*/*  #ifndef USER  #define USER  #include <string>  using namespace std;  struct User  {      string username;      string password;  };  #endif |
| */\* File Name: LoginServer.hpp*  *\* Lab # and Assignment #: Lab #7*  *\* Lab section: 1*  *\* Completed by: Graydon Hall and Jared Kraus*  *\* Submission Date: 2021-11-22*  *\*/*  #ifndef LOGINSERVER  #define LOGINSERVER  #include <string>  #include <vector>  #include "User.hpp"  using namespace std;  class LoginServer{  private:  static LoginServer\* instance;      vector<User> users;      LoginServer(){}      LoginServer(const LoginServer& src);      LoginServer& operator=(LoginServer&rhs);  public:      static LoginServer\* getInstance();      void add(string username, string password);      User\* validate(string username, string password);  };  #endif |
| */\* File Name: LoginServer.cpp*  *\* Lab # and Assignment #: Lab #7*  *\* Lab section: 1*  *\* Completed by: Graydon Hall and Jared Kraus*  *\* Submission Date: 2021-11-22*  *\*/*  #include "User.hpp"  #include "LoginServer.hpp"  #include <iostream>  using namespace std;  *// define static class variable*  LoginServer\* LoginServer::instance = nullptr;  LoginServer\* LoginServer::getInstance(){      if(instance==nullptr){          instance = new LoginServer();      }  *// note that instance is a pointer to a LoginServer*      return instance;  }  void LoginServer::add(string username, string password){  *// first check user does not already exist*      User u;      for(int i=0; i < users.size(); i++){          u = users.at(i);          if (u.username == username ){              cout<<"Error: Unable to add " << username << " because a user with that name already exists.\n";              return;          }      }  *// if we get here, then means user with same username not found in database*      User user;      user.password = password;      user.username=username;      users.push\_back(user);      cout<< username << " sucessfully added!\n";    }  User\* LoginServer::validate(string username, string password){  *// check if a user exists in the database with this username AND password*      User u;      for(int i=0; i < users.size(); i++){          u = users.at(i);          if (u.username == username & u.password == password){              return &(users.at(i));          }      }  *// user was not found*      cout << "User not found\n";      return NULL;  }  LoginServer::LoginServer(const LoginServer& src){      if(this !=&src){          for(int i=0; i < src.users.size(); i++){              this->users.push\_back(src.users.at(i));              this->instance=src.instance;          }      }  }  LoginServer& LoginServer::operator=(LoginServer&rhs){      if(this !=&rhs){          for(int i=0; i < rhs.users.size(); i++){              this->users.push\_back(rhs.users.at(i));              this->instance=rhs.instance;          }      }  } |
| */\* File Name: Client\_A.hpp*  *\* Lab # and Assignment #: Lab #7*  *\* Lab section: 1*  *\* Completed by: Graydon Hall and Jared Kraus*  *\* Submission Date: 2021-11-22*  *\*/*  #ifndef CLIENTA  #define CLIENTA  #include <string>  #include <vector>  #include "LoginServer.hpp"  using namespace std;  class Client\_A{  private:      LoginServer\* instance;  public:      void add(string username, string password);      User\* validate(string username, string password);      Client\_A();  };  #endif |
| */\* File Name: Client\_A.cpp*  *\* Lab # and Assignment #: Lab #7*  *\* Lab section: 1*  *\* Completed by: Graydon Hall and Jared Kraus*  *\* Submission Date: 2021-11-22*  *\*/*  #include <string>  #include <vector>  #include "LoginServer.hpp"  #include "Client\_A.hpp"  using namespace std;  Client\_A::Client\_A(){      instance = LoginServer::getInstance();  }  void Client\_A::add(string username, string password){      instance->add(username, password);  }  User\* Client\_A::validate(string username, string password){      return instance->validate(username, password);  } |
| */\* File Name: Client\_B.hpp*  *\* Lab # and Assignment #: Lab #7*  *\* Lab section: 1*  *\* Completed by: Graydon Hall and Jared Kraus*  *\* Submission Date: 2021-11-22*  *\*/*  #ifndef CLIENTB  #define CLIENTB  #include <string>  #include <vector>  #include "LoginServer.hpp"  using namespace std;  class Client\_B{  private:      LoginServer\* instance;  public:      void add(string username, string password);      User\* validate(string username, string password);      Client\_B();  };  #endif |
| */\* File Name: Client\_B.cpp*  *\* Lab # and Assignment #: Lab #7*  *\* Lab section: 1*  *\* Completed by: Graydon Hall and Jared Kraus*  *\* Submission Date: 2021-11-22*  *\*/*  #include <string>  #include <vector>  #include "LoginServer.hpp"  #include "Client\_B.hpp"  using namespace std;  Client\_B::Client\_B(){      instance = LoginServer::getInstance();  }  void Client\_B::add(string username, string password){      instance->add(username, password);  }  User\* Client\_B::validate(string username, string password){      return instance->validate(username, password);  } |

## Exercise C, part II Discussion

1. **Does your program allow creating objects of LoginServer?**   
   No it does not
2. **If no, why?**  
   The reason we are not allowed to do this is because of the Singleton design pattern we implemented. The singleton principal is a design pattern which ensures we can only have one instance for a class, which in this case is the LoginServer class. For this design pattern, we make the constructor, copy constructor, and assignment operator for LoginServer private, so that they cannot be publicly accessed. This way, the only way a LoginServer can be created is by using the static getInstance() method, which will only instantiate a new LoginServer object if one does not currently exist.