ENSF 619-Fall 2020

Ziad Chemali & Lotfi Hasni

Lab-7

November 6,2020

Exercise: A

i) Code

a) DoubleArrayListSubject.java

```
package exerciseA;
import java.util.ArrayList;

/*
 * DoubleArrayListSubject.java
 * Lab:7-Exercise A
 * Completed by: Ziad Chemali and Lotfi Hasni
 * Submission DAte: November 6 ,2020
 */
public class DoubleArrayListSubject implements Subject {
        public ArrayList<Double> data;
        private ArrayList<Observer> observers;
public DoubleArrayListSubject() {
data=new ArrayList<Double>();
observers=new ArrayList<Observer>();
}

public void addData(Double num) {
        this.data.add(num);
        notifAll();
}
public void setData(Double num,int index) {
        data.set(index, num);
        notifAll();
}

public void populate(double[] arr) {
        for(int i=0;i<arr.length;i++) {
                data.add(arr[i]);
        }
        notifAll();

}

public void display() {

        if(data.size()==0)
                System.out.println("List is Empty");
        else {
                for(int i=0;i<data.size();i++)
                {
                        System.out.print(data.get(i)+"  ");
                }
        }
}

@Override
public void addObserver(Observer observer) {
this.observers.add(observer);
}
```

```java
@Override
public void remove(Observer observer) {
        this.observers.remove(observer);
}

@Override
public void notifAll() {

        for(int i=0;i<observers.size();i++) {
                observers.get(i).update(data);
        }
}}
```

b) FiveRowsTable_Observer.java

```java
package exerciseA;

import java.util.ArrayList;

/*
 * FiveRowsTable_Observer.java
 * Lab:7-Exercise A
 * Completed by: Ziad Chemali and Lotfi Hasni
 * Submission DAte: November 6 ,2020
 */
public class FiveRowsTable_Observer implements Observer {
        ArrayList<Double> array;
        private DoubleArrayListSubject mydata;

        public FiveRowsTable_Observer(DoubleArrayListSubject mydata) {
                this.mydata = mydata;
                this.array = this.mydata.data;
                this.mydata.addObserver(this);
                display();
        }

        @Override
        public void update(ArrayList<Double> array) {
                this.array = array;
                display();
        }

        @Override
        public void display() {
                System.out.println("\nNotification to Five-Rows Table
Observer: Data Changed:");
                int row = 0;
                String temp;
                while (row < 5) {
                        temp = "";
                        for (int i = row; i < array.size(); i += 5) {
                                temp += array.get(i) + " ";
```

```
                        }
                        System.out.println(temp);
                        row++;
                }

        }
}
```

c) Observer.java

```
package exerciseA;

import java.util.ArrayList;
/*
 * Observer.java
 * Lab:7-Exercise A
 * Completed by: Ziad Chemali and Lotfi Hasni
 * Submission DAte: November 6 ,2020
 */
public interface Observer {
public void update(ArrayList<Double> array);
public void display();
}
```

d) ObserverPatternController.java

```
package exerciseA;

/* ObserverPatternController.java
 *  ENSF 619 - Lab 7-ExerciseA
 * M. Moussavi
 * Submission Date: November 6 ,2020
 */

public class ObserverPatternController {
        public static void main(String[] s) {
                double[] arr = { 10, 20, 33, 44, 50, 30, 60, 70, 80, 10, 11,
23, 34, 55 };
                System.out.println("Creating object mydata with an empty list
-- no data:");
                DoubleArrayListSubject mydata = new DoubleArrayListSubject();
                System.out.println("Expected to print: Empty List ...");
                mydata.display();
                mydata.populate(arr);
                System.out.println("mydata object is populated with: 10, 20,
33, 44, 50, 30, 60, 70, 80, 10, 11, 23, 34, 55 ");
                System.out.print("Now, creating three observer objects: ht,
vt, and hl ");
                System.out.println("\nwhich are immediately notified of
existing data with different views.");
                ThreeColumnTable_Observer ht = new
ThreeColumnTable_Observer(mydata);
                FiveRowsTable_Observer vt = new
FiveRowsTable_Observer(mydata);
```

```
                OneRow_Observer hl = new OneRow_Observer(mydata);
                System.out.println("\n\nChanging the third value from 33, to
66 -- (All views must show this change):");
                mydata.setData(66.0, 2);
                System.out.println("\n\nAdding a new value to the end of the
list -- (All views must show this change)");
                mydata.addData(1000.0);
                System.out.println("\n\nNow removing two observers from the
list:");
                mydata.remove(ht);
                mydata.remove(vt);
                System.out.println("Only the remained observer (One Row ), is
notified.");
                mydata.addData(2000.0);
                System.out.println("\n\nNow removing the last observer from
the list:");
                mydata.remove(hl);
                System.out.println("\nAdding a new value the end of the
list:");
                mydata.addData(3000.0);
                System.out.println("Since there is no observer -- nothing is
displayed ...");
                System.out.print("\nNow, creating a new Three-Column observer
that will be notified of existing data:");
                ht = new ThreeColumnTable_Observer(mydata);
        }
}
```

e) OneRow_Observer .java

```
package exerciseA;

import java.util.ArrayList;
/*
 * OneRow_Observer .java
 * Lab:7-Exercise A
 * Completed by: Ziad Chemali and Lotfi Hasni
 * Submission DAte: November 6 ,2020
 */
public class OneRow_Observer implements Observer{

        private DoubleArrayListSubject mydata;
        private ArrayList<Double> array;

        public OneRow_Observer(DoubleArrayListSubject mydata) {

                this.mydata=mydata;
                this.array=this.mydata.data;
                this.mydata.addObserver(this);
                display();
        }

        @Override
        public void update(ArrayList<Double> array) {

                this.array=array;
```

```
                display();
        }

        @Override
        public void display() {

                System.out.println("\nNotification to One-Row Observer: Data
Changed:");
                System.out.println(array.toString());

        }

}
```

f) Subject.java

```
package exerciseA;
/*
 * Subject.java
 * Lab:7-Exercise A
 * Completed by: Ziad Chemali and Lotfi Hasni
 * Submission Date: November 6 ,2020
 */
public interface Subject {
public void addObserver(Observer observer);
public void remove(Observer observer) ;
public void notifAll();

}
```

g) ThreeColumnTable_Observer.java

```
package exerciseA;

import java.util.ArrayList;
/*
 * ThreeColumnTable_Observer.java
 * Lab:7 Exercise A
 * Completed by: Ziad Chemali and Lotfi Hasni
 * Submission DAte: November 6 ,2020
 */
public class ThreeColumnTable_Observer implements Observer {
ArrayList<Double> array;
private DoubleArrayListSubject mydata;

public ThreeColumnTable_Observer(DoubleArrayListSubject mydata) {

        this.mydata=mydata;
        this.array=this.mydata.data;
        this.mydata.addObserver(this);
        display();
        }

        @Override
```

```java
        public void update(ArrayList<Double> array) {
                this.array=array;
                display();
        }

        @Override
        public void display() {

                System.out.println("\nNotification to Three-Column Table
Observer: Data Changed:");
                int col=0;
           for(int i=0;i<array.size();i++) {
                   if(col<3)
                   System.out.print(array.get(i)+" ");
                   col++;
                   if(col==3)
                   { col=0;
                   System.out.println();
                   }

           }
        }

}
```

ii) Output

```
Creating object mydata with an empty list -- no data:
Expected to print: Empty List ...
List is Empty
mydata object is populated with: 10, 20, 33, 44, 50, 30, 60, 70, 80, 10, 11,
23, 34, 55
Now, creating three observer objects: ht, vt, and hl
which are immediately notified of existing data with different views.

Notification to Three-Column Table Observer: Data Changed:
10.0 20.0 33.0
44.0 50.0 30.0
60.0 70.0 80.0
10.0 11.0 23.0
34.0 55.0
Notification to Five-Rows Table Observer: Data Changed:
10.0 30.0 11.0
20.0 60.0 23.0
33.0 70.0 34.0
44.0 80.0 55.0
```

```
50.0 10.0

Notification to One-Row Observer: Data Changed:
[10.0, 20.0, 33.0, 44.0, 50.0, 30.0, 60.0, 70.0, 80.0, 10.0, 11.0, 23.0,
34.0, 55.0]


Changing the third value from 33, to 66 -- (All views must show this change):

Notification to Three-Column Table Observer: Data Changed:
10.0 20.0 66.0
44.0 50.0 30.0
60.0 70.0 80.0
10.0 11.0 23.0
34.0 55.0
Notification to Five-Rows Table Observer: Data Changed:
10.0 30.0 11.0
20.0 60.0 23.0
66.0 70.0 34.0
44.0 80.0 55.0
50.0 10.0

Notification to One-Row Observer: Data Changed:
[10.0, 20.0, 66.0, 44.0, 50.0, 30.0, 60.0, 70.0, 80.0, 10.0, 11.0, 23.0,
34.0, 55.0]


Adding a new value to the end of the list -- (All views must show this
change)

Notification to Three-Column Table Observer: Data Changed:
10.0 20.0 66.0
44.0 50.0 30.0
60.0 70.0 80.0
10.0 11.0 23.0
34.0 55.0 1000.0

Notification to Five-Rows Table Observer: Data Changed:
10.0 30.0 11.0
20.0 60.0 23.0
66.0 70.0 34.0
44.0 80.0 55.0
50.0 10.0 1000.0

Notification to One-Row Observer: Data Changed:
[10.0, 20.0, 66.0, 44.0, 50.0, 30.0, 60.0, 70.0, 80.0, 10.0, 11.0, 23.0,
34.0, 55.0, 1000.0]


Now removing two observers from the list:
Only the remained observer (One Row ), is notified.

Notification to One-Row Observer: Data Changed:
[10.0, 20.0, 66.0, 44.0, 50.0, 30.0, 60.0, 70.0, 80.0, 10.0, 11.0, 23.0,
34.0, 55.0, 1000.0, 2000.0]
```

```
Now removing the last observer from the list:

Adding a new value the end of the list:
Since there is no observer -- nothing is displayed ...

Now, creating a new Three-Column observer that will be notified of existing
data:
Notification to Three-Column Table Observer: Data Changed:
10.0 20.0 66.0
44.0 50.0 30.0
60.0 70.0 80.0
10.0 11.0 23.0
34.0 55.0 1000.0
2000.0 3000.0
```

Exercise: B & C

i) Code

a) BorderDecorator.java

```
package exerciseB;

import java.awt.BasicStroke;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Rectangle;
/*
 * BorderDecorator.java
 * Lab:7-Exercise B & C
 * Completed by: Ziad Chemali and Lotfi Hasni
 * Submission Date: November 6 ,2020
 */
public class BorderDecorator extends Decorator{

        public BorderDecorator(Component t, int x, int y, int width, int
height) {
                cmp=t;
                this.x=x;
                this.y=y;
                this.width=width;
                this.height=height;
        }

        @Override
        public void draw(Graphics g) {
                cmp.draw(g);
        Graphics2D g2 = (Graphics2D) g;
        float dash[] = { 10.0f };
    g2.setStroke(new BasicStroke(3.0f, BasicStroke.CAP_BUTT,
         BasicStroke.JOIN_MITER, 10.0f, dash, 0.0f));

        Rectangle r=new Rectangle(x,y,width, height);
        g2.draw(r);
        }
```

```
}
```

b) ColouredFrameDecorator.java

```
package exerciseB;

import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Rectangle;

/*
 * ColouredFrameDecorator.java
 * Lab:7-Exercise B & C
 * Completed by: Ziad Chemali and Lotfi Hasni
 * Submission DAte: November 6 ,2020
 */
public class ColouredFrameDecorator extends Decorator {
protected int thickness;
        public ColouredFrameDecorator(Component t, int x, int y, int width,
int height, int thickness) {
                cmp=t;
                this.x=x;
                this.y=y;
                this.width=width;
                this.height=height;
                this.thickness=thickness;
        }

        @Override
        public void draw(Graphics g) {
                cmp.draw(g);
                Graphics2D g2 = (Graphics2D) g;
                g2.setStroke(new BasicStroke(thickness));
                g2.setColor(Color.red);
                Rectangle r=new Rectangle(x,y,width, height);
                g2.draw(r);
        }



}
```

c) ColouredGlassDecorator.java

```
package exerciseB;

import java.awt.AlphaComposite;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
```

```
/*
 * ColouredGlassDecorator.java
 * Lab:7-Exercise B & C
 * Completed by: Ziad Chemali and Lotfi Hasni
 * Submission DAte: November 6 ,2020
 */
public class ColouredGlassDecorator extends Decorator {

	public ColouredGlassDecorator(Component t, int x, int y, int width,
int height) {
		cmp=t;
		this.x=x;
		this.y=y;
		this.height=height;
		this.width=width;

	}

	@Override
	public void draw(Graphics g) {
		cmp.draw(g);
		Graphics2D g2d = (Graphics2D) g;
		g2d.setColor(Color.green);

	g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER, 1
* 0.1f));
		g2d.fillRect(25, 25, 110, 110);

	}

}
```

d) Component.java

```
package exerciseB;

import java.awt.Graphics;
/*
 * Component.java
 * Lab:7-Exercise B & C
 * Completed by: Ziad Chemali and Lotfi Hasni
 * Submission DAte: November 6 ,2020
 */
public interface Component {
public void draw(Graphics g);
}
```

e) Decorator.java

```
package exerciseB;
/*
 * Decorator.java
 * Lab:7-Exercise B & C
 * Completed by: Ziad Chemali and Lotfi Hasni
```

```
 * Submission Date: November 6 ,2020
 */
public  abstract class Decorator implements Component{
protected Component cmp;
protected int x,y,width,height;
}
```

f) DemoDecoratorPattern.java

```
package exerciseB;

import java.awt.Font;
import java.awt.Graphics;
import javax.swing.JFrame;
import javax.swing.JPanel;
/*
 * DemoDecoratorPattern.java
 * Lab:7-Exercise B & C
 * Completed by: Ziad Chemali and Lotfi Hasni
 * Submission DAte: November 6 ,2020
 */
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));

            // Now lets decorate t with BorderDecorator: x = 30, y = 30, width
= 100, and height 100
            t = new BorderDecorator(t, 30, 30, 100, 100);

            // Now lets add a ColouredFrameDecorator with x = 25, y = 25,
width = 110, height = 110,
                // and thickness = 10.
            t = new ColouredFrameDecorator(t, 25, 25, 110, 110, 10);

            //Exercise c
            t=new ColouredGlassDecorator(t,25,25,110,110);
            // Now lets draw the product on the screen
            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);
        }
```

```
}
```

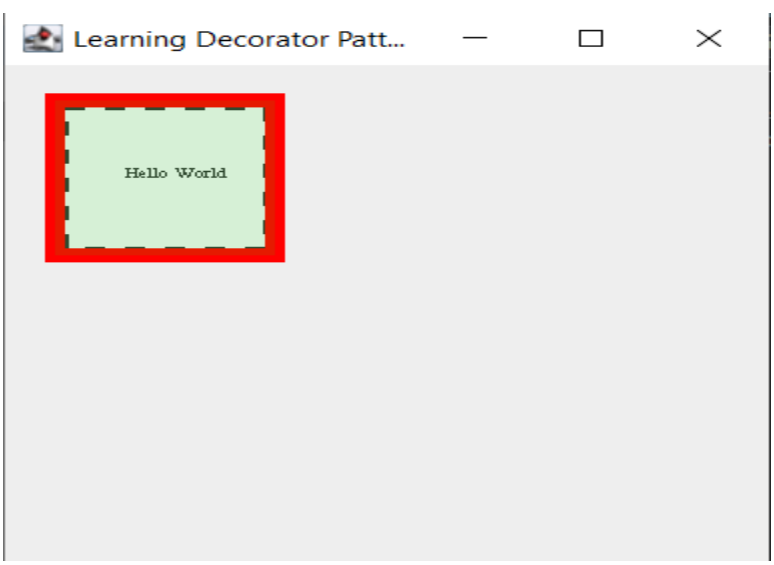
g) Text.java

```
package exerciseB;

import java.awt.Graphics;
/*
 * Text.java
 * Lab:7-Exercise B & C
 * Completed by: Ziad Chemali and Lotfi Hasni
 * Submission DAte: November 6 ,2020
 */
public class Text implements Component {
        private String text;
        int x,y;
public Text(String text, int x, int y) {
        this.text=text;
        this.x=x;
        this.y=y;
}
@Override
public void draw(Graphics g) {

         g.drawString(text, x, y);
}
}
```

ii) Output



Exercise: D

i) Code

a) Header files

1) Client_A.h

```
/*
* Client_A.h
* Lab 7, Exercise D
* By: Ziad Chemali & Lotfi Hasni
* Submission: November 6, 2020
*/
#ifndef ClientA_h
#define ClientA_h
#include"LoginServer.h"
class Client_A {
private:
	LoginServer* instance;
public:
	void add(string username, string password);
	User* validate(string username, string password);
	Client_A();

};
#endif
```

2) Client_B.h

```
/*
* Client_B.h
* Lab 7, Exercise D
* By: Ziad Chemali & Lotfi Hasni
* Submission: November 6, 2020
*/
#ifndef ClientB_h
#define ClientB_h
#include"LoginServer.h"
class Client_B {
private:
	LoginServer* instance;
public:
	void add(string username, string password);
	User* validate(string username, string password);
	Client_B();

};
#endif
```

3) LoginServer.h

```
/*
* LoginServer.h
* Lab 7, Exercise D
* By: Ziad Chemali & Lotfi Hasni
* Submission: November 6, 2020
*/
#ifndef LoginServer_h
#define LoginSErver_h
#include <string>
#include <vector>
```

```
#include<string.h>
using namespace std;

struct User
{
	string username;
	string password;

};

class LoginServer {

public:


	static LoginServer* getInstance();
	void add(string username, string password);
	User* validate(string username, string password);
    ~LoginServer();
private:
	vector<User> users;
	static LoginServer* instance;
	LoginServer();
	};
#endif // !LoginServer_h
```

b) cpp files

1) Client_A.cpp

```
/*
* Client_A.cpp
* Lab 7, Exercise D
* By: Ziad Chemali & Lotfi Hasni
* Submission: November 6, 2020
*/
#include "Client_A.h"

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);
}

Client_A::Client_A()
{
	instance = LoginServer::getInstance();
}
```

2) Client_B.cpp

```
/*
* Client_B.cpp
* Lab 7, Exercise D
```

```
* By: Ziad Chemali & Lotfi Hasni
* Submission: November 6, 2020
*/

#include "Client_B.h"

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);
}

Client_B::Client_B()
{
	instance =LoginServer::getInstance();
}
```

3) LoginServer.cpp

```
/*
* LoginServer.cpp
* Lab 7, Exercise D
* By: Ziad Chemali & Lotfi Hasni
* Submission: November 6, 2020
*/


#include "LoginServer.h"
#include<iostream>
LoginServer::LoginServer()
{
	}




LoginServer* LoginServer::getInstance()
{
	if (instance == nullptr)
		instance = new LoginServer();
	return instance;
}

void LoginServer::add(string username, string password)
{

	bool check = true;
	for (int i = 0;i < users.size();i++)
	{
		User temp = users[i];
```

```
		if (username == temp.username || password == temp.password) {
			std::cout << "Error "<<username << " already exists" << endl;
			return;
		}
	}
	User add_to_list = { username, password };
	users.push_back(add_to_list);
}

User* LoginServer::validate(string username, string password)
{
	for (int i = 0;i < users.size();i++)
	{
		User temp = users[i];

		if (username == temp.username && password == temp.password) {
			return &users[i];
		}

	}

	return nullptr;
}
LoginServer::~LoginServer()
{
	cout << "Deleting" << endl;
	delete instance;
}
LoginServer* LoginServer::instance = new LoginServer();
```

4) main.cpp

```
/*
* main.cpp
* Lab 7, Exercise D
* By: Ziad Chemali & Lotfi Hasni
* Submission: November 6, 2020
*/

#include "Client_A.h"
#include "Client_B.h"

#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;
}
```

ii) Output:

Microsoft Visual Studio Debug Console

```
Created a new Client_A object called ca ...
adding two usernames, Jack and Judy, by client ca ...
Created a new Client_B object called cb ...
Adding two usernames called Jim and Josh, by client cb ...
Now adding another username called Jim by client ca.
It must be avoided because a similar username already exits ...
Error Jim already exists
Another attempt to add username called Jim, but this time by client cb,
with a different password
It must be avoided again ...
Error Jim already exists
Now client cb validates existence of username Jack and his password:
Found: username: Jack and the password is: apple5000
Now client ca validates existence of username Jack with a wrong password.
Username or password NOT found
Trying to make a new Client_A object which is a copy of client ca:
Adding a usernames called Tim by client ca2 ...
Make a new Client_A object called ca3:
Make ca3 a copy of ca2:
Now client ca3 validates existence of username Tim and his password:
Found: username: Tim and the password is: blue_sky
```

iii) Question answer:

Program does not allow creating an object of LoginServer, because singleton patten is responsible of creating only **single** object. This singleton constructor is **private** and the object can be accessed by getInstance() function

<u>Solution:</u>

LoginServer* x=LoginServer::getInstance();

And now we can access the same singleton object