 American University of Sharjah

College of Engineering

Department of Computer Engineering

Spring 2020

Software Design Lab (COE 312)

Assignment 4

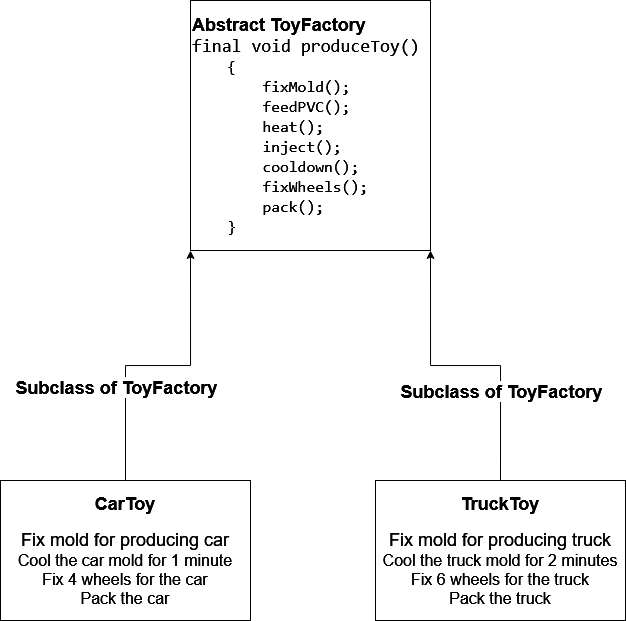
Design Patterns

Miss Hend ElGhazaly

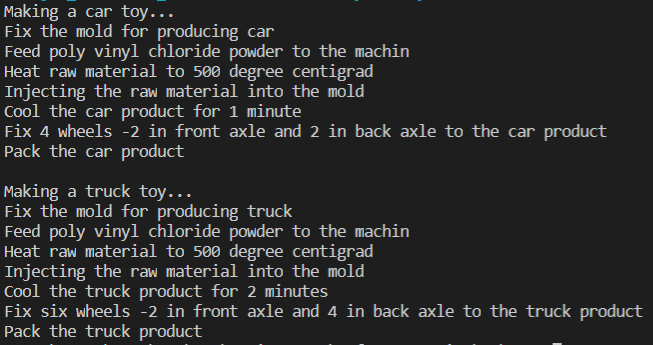
|  |  |
| --- | --- |
| Submission date | 4/5/2020 |
|  |  |

|  |  |
| --- | --- |
| Mohammad Kharoof | 73865 |
| Maya Hiba | 74896 |

**Exercise 1:**

1. The most suitable design pattern is the template design pattern
2. The diagram:
3. Code:

*Output*



*Toyfactory.java*

public abstract class ToyFactory

{

    final void produceToy()

    {

        fixMold();

        feedPVC();

        heat();

        inject();

        cooldown();

        fixWheels();

        pack();

    }

    abstract void fixMold();

    protected void feedPVC()

    {

        System.out.println("Feed poly vinyl chloride powder to the machin");

    }

    protected void heat()

    {

        System.out.println("Heat raw material to 500 degree centigrad");

    }

    protected void inject()

    {

        System.out.println("Injecting the raw material into the mold");

    }

    abstract void cooldown();

    abstract void fixWheels();

    abstract void pack();

}

*CarToy.java*

public void fixMold()

    {

        System.out.println("Fix the mold for producing car");

    }

    public void cooldown()

    {

        System.out.println("Cool the car product for 1 minute");

    }

    public void fixWheels()

    {

        System.out.println("Fix 4 wheels -2 in front axle and 2 in back axle to the car product");

    }

    public void pack()

    {

        System.out.println("Pack the car product");

    }

*TruckToy.java*

public class TruckToy extends ToyFactory

{

    public void fixMold()

    {

        System.out.println("Fix the mold for producing truck");

    }

    public void cooldown()

    {

        System.out.println("Cool the truck product for 2 minutes");

    }

    public void fixWheels()

    {

        System.out.println("Fix six wheels -2 in front axle and 4 in back axle to the truck product");

    }

    public void pack()

    {

        System.out.println("Pack the truck product");

    }

}

*TestToyFactory.java*

public class TestToyFactory

{

    public static void main(String[] args)

    {

        ToyFactory tf;

        System.out.println("Making a car toy...");

        tf = new CarToy();

        tf.produceToy();

        System.out.println("\nMaking a truck toy...");

        tf = new TruckToy();

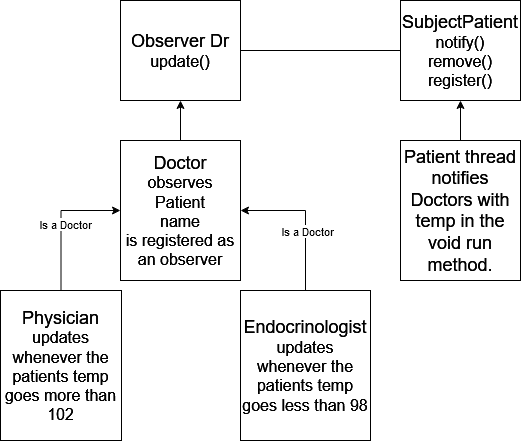
        tf.produceToy();

    }

}

**Exercise 2:**

1. Most suitable design pattern is Observer/Subject design pattern.



1. Code:

*SubjectPatient.java*

public interface SubjectPatient

{

    public void register(ObserverDoctor o);

    public void remove(ObserverDoctor o);

    public void notify(int temp);

}

*Patient.java*

import java.util.ArrayList;

public class Patient implements Runnable, SubjectPatient

{

    private final ArrayList observersList;

    Thread thr;

    int temp;

    public Patient() {

        this.observersList = new ArrayList();

        thr = new Thread(this, "Patient");

        thr.start();

    }

    public Thread getThread() {

        return thr;

    }

    public void register(final ObserverDoctor o) {

        observersList.add(o);

    }

    public void remove(final ObserverDoctor o) {

        final int x = observersList.indexOf(o);

        if (x > -1) {

            observersList.remove(x);

        } else {

            System.out.println("Observer doesnt exist....");

        }

    }

    public void notify(final int temp) {

        for (int i = 0; i < observersList.size(); i++) {

            final ObserverDoctor dummy = (ObserverDoctor) observersList.get(i);

            dummy.updateDr(this, this.temp);

        }

    }

    public void run() {

        while (true) {

            this.temp = (int) (96 + Math.random() \* 10);

            this.notify(this.temp);

            try {

                Thread.sleep(10 \* 1000);

            } catch (final InterruptedException e)

            {

                e.printStackTrace();

            }

        }

    }

}

*ObserverDoctor.java*

public interface ObserverDoctor

{

    void updateDr(Patient patient, int temp);

}

*Doctor.java*

public abstract class Doctor implements ObserverDoctor

{

    String name;

    private SubjectPatient subject;

    Doctor(String n, SubjectPatient s)

    {

        name = "Dr. " + n;

        subject = s;

        subject.register(this);

    }

}

*Physician.java*

import java.util.\*;

public class Physician extends Doctor

{

    Physician(SubjectPatient s)

    {

        super("Physican", s);

    }

    public void updateDr(Patient patient, int temp)

    {

        if(temp > 102)

        {

            System.out.println(patient.getThread().getName() +  " temprature is " + temp + " and was examined by " + this.name + " on " + new Date().toString());

        }

    }

}

*Endocrinologist.java*

import java.util.\*;

public class Endocrinologist extends Doctor

{

    Endocrinologist(SubjectPatient s)

    {

        super("Endocrinologist", s);

    }

    public void updateDr(Patient patient, int temp)

    {

        if(temp < 98)

        {

            System.out.println(patient.getThread().getName() +  " temprature is " + temp + " and was examined by " + this.name + " on " + new Date().toString());

        }

    }

}

*Main.java*

public class Main

{

    public static void main(String[] args)

    {

        Patient p = new Patient();

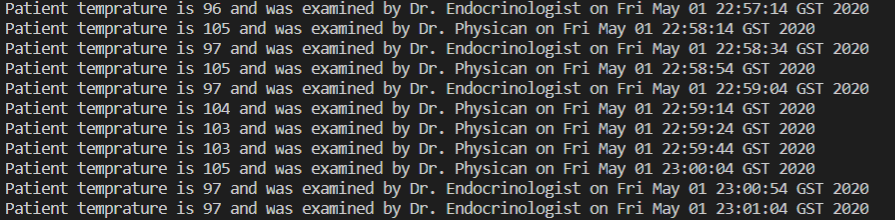
        Physician phys = new Physician(p);

        Endocrinologist end = new Endocrinologist(p);

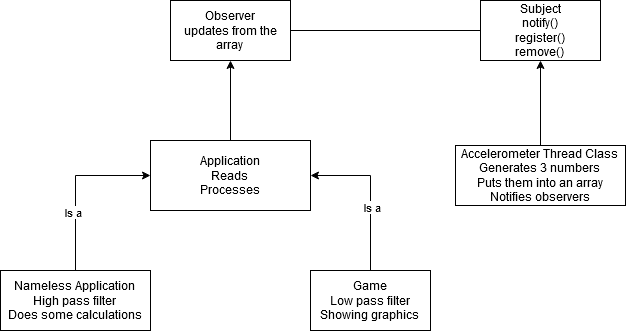
    }

}

*Output*



**Exercise 3:**

1. A combination of template and observer/subject patterns will do the job.
2. 
3. Code:

*ObserverApplication.java*

public interface ObserverApplication

{

    void updateApp(Accelerometer accel, float[] readings);

}

*Applications.java*

public abstract class Applications implements ObserverApplication

{

    String name;

    private SubjectAccelerometer subject;

    Applications(String n, SubjectAccelerometer s)

    {

        name = n;

        subject = s;

        subject.register(this);

    }

    final public void app(float[] readings)

    {

        read(readings);

        filter();

        process();

    }

    public void read(float[] readings)

    {

        System.out.print("Reading [");

        for(int i = 0; i < readings.length; i++)

        {

            System.out.print(readings[i]);

            if(i < (readings.length - 1))

                System.out.print(", ");

        }

        System.out.println("]");

    }

    abstract void filter();

    abstract void process();

}

*SubjectAccelerometer.java*

public interface SubjectAccelerometer

{

    public void register(ObserverApplication o);

    public void remove(ObserverApplication o);

    public void notify(float[] temp);

}

*Accelerometer.java*

import java.util.ArrayList;

public class Accelerometer implements Runnable, SubjectAccelerometer {

    private final ArrayList observersList;

    float[] values;

    Thread thr;

    Accelerometer() {

        values = new float[3];

        this.observersList = new ArrayList();

        thr = new Thread(this);

        thr.start();

    }

    public Thread getThread() {

        return thr;

    }

    public void register(final ObserverApplication o) {

        observersList.add(o);

    }

    public void remove(final ObserverApplication o) {

        final int x = observersList.indexOf(o);

        if (x > -1) {

            observersList.remove(x);

        } else {

            System.out.println("Observer doesnt exist....");

        }

    }

    public void notify(final float[] readings)

    {

            for (int i = 0; i < observersList.size(); i++)

            {

                final ObserverApplication dummy = (ObserverApplication) observersList.get(i);

                dummy.updateApp(this, this.values);

            }

    }

    public void run() {

        while (true)

        {

        for (int i = 0; i < values.length; i++)

            values[i] = (float) (Math.random() \* 10);

        this.notify(this.values);

        try {

            Thread.sleep(1000);

        } catch (InterruptedException e) {

            e.printStackTrace();

        }

    }

    }

}

*Game.java*

public class Game extends Applications

{

    Game(SubjectAccelerometer s)

    {

        super("Game", s);

    }

    public void updateApp(Accelerometer accel, float[] readings)

    {

        app(readings);

    }

    public void filter()

    {

        System.out.println(this.name + " is applying low-pass filter.");

    }

    public void process()

    {

        System.out.println(this.name + " is showing graphics.");

    }

}

*NamelessApplication.java*

public class NamelessApplication extends Applications

{

    NamelessApplication (SubjectAccelerometer s)

    {

        super("The application", s);

    }

    public void updateApp(Accelerometer accel, float[] readings)

    {

        app(readings);

    }

    public void filter()

    {

        System.out.println(this.name + " is applying high-pass filter.");

    }

    public void process()

    {

        System.out.println(this.name + " is doing its calculations");

    }

}

*RunApp.java*

public class RunApp

{

    public static void main(String[] args)

    {

        Accelerometer acc = new Accelerometer();

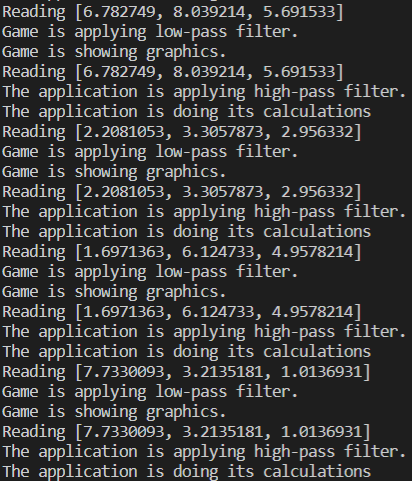
        Game game = new Game(acc);

        NamelessApplication app = new NamelessApplication(acc);

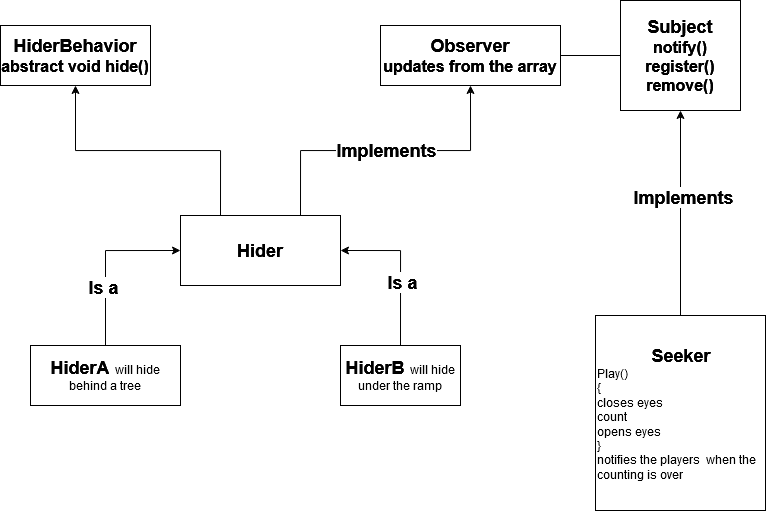
    }

}

*Output*



Exercise 4:

1. Template, Observer and Strategy design patterns are needed.
2. 
3. The code:

*HidingBehavior.java*

public abstract class HidingBehavior

{

    abstract void hide();

}

*ObserverHiders.java*

public interface ObserverHiders

{

    void updateHiders(Seeker seeker, int x);

}

*SubjectSeeker.java*

public interface SubjectSeeker

{

    public void registerHider(ObserverHiders o);

    public void removeHider(ObserverHiders o);

    public void notifyHider(int x);

}

*Seeker.java*

import java.util.ArrayList;

import java.util.Random;

public class Seeker implements Runnable, SubjectSeeker

{

    private final ArrayList observersList;

    Thread thr;

    String name;

    Seeker() {

        this.observersList = new ArrayList();

        name = "Seeker";

        thr = new Thread(this, name);

        thr.start();

    }

    public String getName()

    {

        return this.name;

    }

    final void play() {

        closeEyes();

        count();

        open();

    }

    public void closeEyes() {

        System.out.println(name + " closes their eyes");

    }

    public void count()

    {

        System.out.println(name + " counts to 10..9..8..7..6..5..4..3..2..1");

        try {

            Thread.sleep(3 \* 1000);

        }

        catch (InterruptedException e)

        {

            e.printStackTrace();

        }

    }

    public void open()

    {

        System.out.println(name + " opens their eyes");

    }

    public Thread getThread() {

        return thr;

    }

    public void registerHider(final ObserverHiders o) {

        observersList.add(o);

    }

    public void removeHider(final ObserverHiders o) {

        final int x = observersList.indexOf(o);

        if (x > -1) {

            observersList.remove(x);

        } else {

            System.out.println("Observer doesnt exist....");

        }

    }

    public void notifyHider(int x)

    {

            for (int i = 0; i < observersList.size(); i++)

            {

                final ObserverHiders dummy = (ObserverHiders) observersList.get(i);

                dummy.updateHiders(this, x);

            }

    }

    public void run()

    {

        play();

        Random rnd = new Random();

        int x = rnd.nextInt(2);

        this.notifyHider(x);

    }

}

*Hider.java*

public abstract class Hider extends HidingBehavior implements ObserverHiders

{

    String name;

    private SubjectSeeker subject;

    public Hider(String n, SubjectSeeker s)

    {

        subject = s;

        name = "Hider " + n;

        subject.registerHider(this);

    }

}

*HiderB.java*

public class HiderB extends Hider

{

    public HiderB(String n, SubjectSeeker s)

    {

        super(n,s);

    }

    public void hide()

    {

        System.out.println(name + " hid under a ramp");

    }

    public void updateHiders(Seeker s, int x)

    {

        System.out.println(name + " saw the seeker and runs");

        if (x == 1)

        {

            try

            {

                Thread.sleep(5000);

            }

            catch (InterruptedException e)

            {

                e.printStackTrace();

            }

            System.out.println(s.getName() + " catches " + name);

            System.out.println(name + " lost");

        }

    }

}

*HiderA.java*

public class HiderA extends Hider {

    public HiderA(String n, SubjectSeeker s) {

        super(n, s);

    }

    public void hide() {

        System.out.println(name + " hid behind a tree");

    }

    public void updateHiders(Seeker s, int x) {

        System.out.println(name + " saw the seeker and runs");

        if (x == 0)

        {

            try

            {

                Thread.sleep(5000);

            }

            catch (InterruptedException e)

            {

                e.printStackTrace();

            }

            System.out.println(s.getName() + " catches " + name);

            System.out.println(name + " lost");

        }

    }

}

*MainGame.java*

public class MainGame {

    public static void main(String[] args) throws InterruptedException

    {

        Seeker s = new Seeker();

        Hider p1 = new HiderA("Maya", s);

        Hider p2 = new HiderB("Kharoof", s);

        System.out.println("Game started");

        p1.hide();

        p2.hide();

    }

}

*Output*

