**كلية الهندسة المعلوماتية**



**السنة الخامسة**

**قسم هندسة البرمجيات ونظم المعلومات**

**وظيفة هندسة البرمجيات -3-**

**Design Pattern**

**المهندس**

**روان قرعوني**

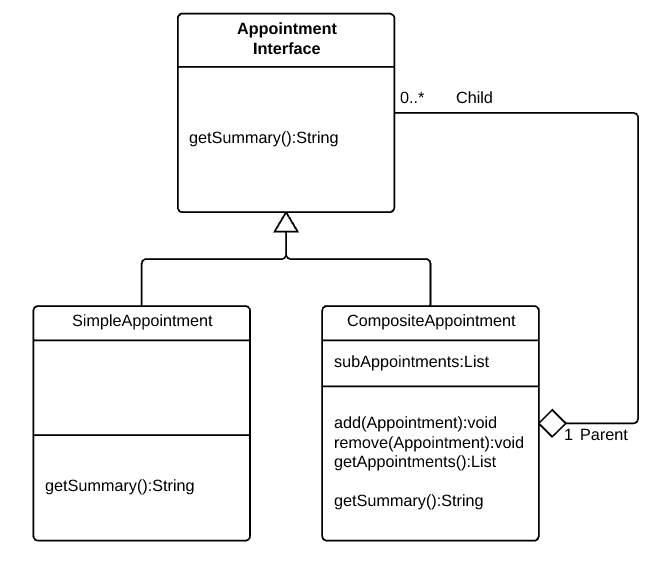
**الطلاب**

**الطلب الأول:**

النموذج المستخدم هو composite design pattern إذ يتكون الموعد من فعالية رئيسية وضمن هذه الفعالية الرئيسية يوجد مواعيد لفعاليات جزئية، وبالتالي نحن بحاجة إلى تمثيل الموعد ببنية هرمية (Hierarchical representations) إذ يمكن للموعد الواحد أن يحوي مواعيد أخرى.

موعد الفعالية الرئيسية لا يختلف من حيث طريقة المعالجة والتعامل عن موعد الفعالية الجزئية وبالتالي نحن بحاجة إلى بنية موحدة للتعامل مع المواعيد الجزئية والمواعيد الرئيسية.

بحسب ما سبق يكون مخطط الصفوف الذي يعبر عن الموعد كما يلي:



الصف المجرد Appointment بحيث يحوي الموعد على توقيت وتاريخ وإجرائية طباعة تفاصيل الموعد.، الصف CompositeAppointment يرث الصف المجرد ويحوي قائمة مواعيد، وهي التي تعبر عن المواعيد الجزئية، الصف SimpleAppointment يرث الصف المجرد.

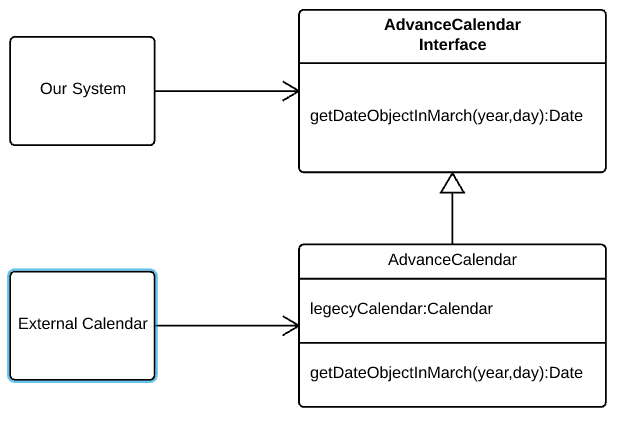
مخطط الصفوف هذا يطابق نموذج التصميم Composite إذ أن الصف Appointment هو Compound والصف CompositeAppointment هو Composite والصف SimpeAppointment هو Leaf.

|  |
| --- |
| import java.util.ArrayList; |
| import java.util.List; |
|  |
| public class Question1 { |
|  |
| public static void main(String[] args) { |
|  |
| SimpleAppointment SA1 = new SimpleAppointment("Section 1","10:00 AM"); |
| SimpleAppointment SA2 = new SimpleAppointment("Section 2","11:00 AM"); |
| SimpleAppointment SA3 = new SimpleAppointment("Section 3","11:30 AM"); |
|  |
|  |
| CompositeAppointment CA1 = new CompositeAppointment(); |
| CA1.addSubAppointment(SA1); |
| CA1.addSubAppointment(SA2); |
| CA1.addSubAppointment(SA3); |
|  |
|  |
| CompositeAppointment CA2 = new CompositeAppointment(); |
|  |
| SimpleAppointment SA4 = new SimpleAppointment("Stand 23", "5:00 PM"); |
|  |
| CA2.addSubAppointment(CA1); |
| CA2.addSubAppointment(SA4); |
|  |
| System.out.println(CA2.getSummary()); |
| } |
|  |
|  |
|  |
| } |
|  |
| interface Appointment{ |
| public abstract String getSummary(); |
| } |
|  |
|  |
| class CompositeAppointment implements Appointment{ |
| private List<Appointment> subAppointmetns; |
|  |
| public CompositeAppointment(){ |
| subAppointmetns = new ArrayList<>(); |
| } |
|  |
| @Override |
| public String getSummary() { |
| String summary = "Compsosite Appointment \n\t"; |
|  |
| for (Appointment appointment : subAppointmetns) { |
| summary += appointment.getSummary() +"\n\t"; |
| } |
| summary += "\n"; |
| return summary; |
| } |
|  |
| public void addSubAppointment(Appointment appointment){ |
| subAppointmetns.add(appointment); |
| } |
|  |
| public void removeAppointment(Appointment appointment){ |
| subAppointmetns.remove(appointment); |
| } |
|  |
| public List<Appointment> getSubAppointment(){ |
| return subAppointmetns; |
| } |
|  |
| } |
|  |
| class SimpleAppointment implements Appointment{ |
|  |
| private String locaiton; |
| private String time; |
|  |
| public SimpleAppointment(String location,String time){ |
| this.locaiton = location; |
| this.time = time; |
| } |
|  |
| @Override |
| public String getSummary() { |
| return String.format("Simple Appointment { %s , %s }",locaiton,time); |
| } |
|  |
| } |

**الطلب الثاني:**

النموذج المستخدم هو من نوع Adapter Design Pattern لأن المكتبة الخارجية تقدم لنا الوظيفة (Functionally) التي نريدها ولكن الواجهة (Interface) التي تقدمها لا تناسب احتياجاتنا. اخترنا هذا النمط لأنه مرن وعمله هو الربط وتسهيل التواصل مع الأنظمة وليس التعامل مع النظام السابق بطريقة تنفيذ جديدة.

بحسب ما سبق يكون مخطط الصفوف كما يلي:

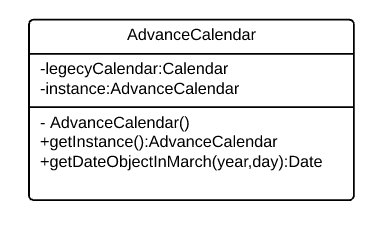


الواجهة AdvanceCalendarInterface تعرف الطريقة التي تناسب احتياجاتنا، الصف AdvanceCalendear طبق الواجهة السابقة وهو Adapter، الواجهة External Calendar هو واجهة المكتبة الخارجية وهي Adaptee.

|  |
| --- |
| import java.util.Calendar; |
| import java.util.Date; |
|  |
| public class Question2 { |
| public static void main(String[] args) { |
| CalenderInterface calender = new AdvanceCalender(); |
| System.out.println(calender.getDateInMarch(2016, 1)); |
| } |
| } |
|  |
|  |
| interface CalenderInterface{ |
| public Date getDateInMarch(int year,int day); |
| } |
|  |
| class AdvanceCalender implements CalenderInterface{ |
| private Calendar calendar; |
|  |
|  |
| public AdvanceCalender(){ |
| calendar = Calendar.getInstance(); |
| } |
|  |
|  |
| @Override |
| public Date getDateInMarch(int year,int day) { |
| calendar.set(Calendar.YEAR, year); |
| calendar.set(Calendar.MONTH,Calendar.MARCH); |
| calendar.set(Calendar.DAY\_OF\_MONTH,day); |
| return calendar.getTime(); |
| } |
| } |

**الطلب الثالث:**

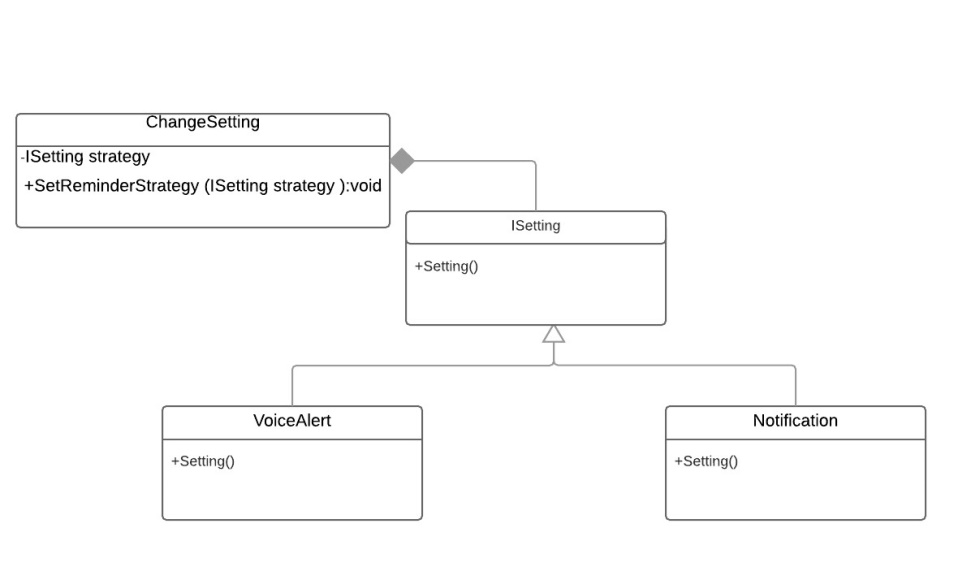
النموذج المستخدم في هذا الطلب هو من نوع Design Patter Singleton فعندما يريد التطبيق أن يتعامل مع التقويم يتم التأكد من وجود نسخة واحد وواحد فقط من التقويم، فالصف يحتوي على باني private ودوماً يرد نفس النسخة عن طريق Static Method وبذلك نضمن وجود نسخة واحدة من التقويم طول فترة حياة البرنامج.



|  |
| --- |
|  |
| import java.util.Date; |
|  |
| public class Question3 { |
| public static void main(String[] args) { |
| CalenderInterface calender = AdvanceCalender2.getInstance(); |
| System.out.println(calender.getDateInMarch(2016, 1)); |
| } |
| } |
|  |
| class AdvanceCalender2 implements CalenderInterface{ |
|  |
| private static AdvanceCalender2 instance; |
| private Calendar calendar; |
| private AdvanceCalender2(){ |
| calendar = Calendar.getInstance(); |
| } |
|  |
| public static AdvanceCalender2 getInstance(){ |
| if (instance == null){ |
| instance = new AdvanceCalender2(); |
| return instance; |
| }else{ |
| return instance; |
| } |
| } |
|  |
| @Override |
| public Date getDateInMarch(int year,int day) { |
| calendar.set(Calendar.YEAR, year); |
| calendar.set(Calendar.MONTH,Calendar.MARCH); |
| calendar.set(Calendar.DAY\_OF\_MONTH,day); |
| return calendar.getTime(); |
| } |
| } |

**الطلب الرابع:**

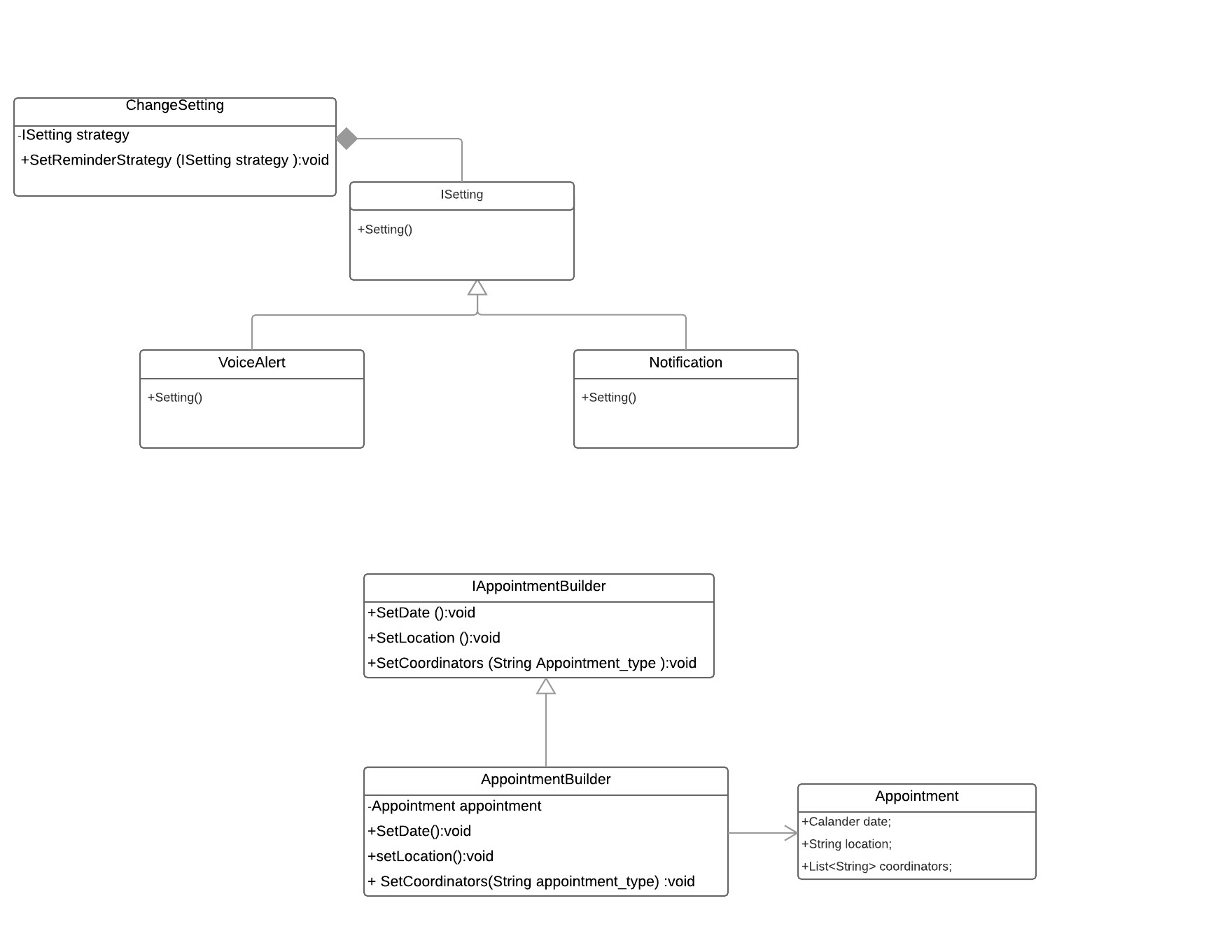
النموذج المستخدم هو من نوع Strategy Design Pattern بما أنه يوجد لدينا عدة طرق للتنبيه (إشعارات أو تنبيه صوتي) وبهدف السماح بتغيير طريقة التذكير من إعدادات التطبيق يكون كل طريقة تنبيه هي صف فمثلاً لدينا الصفان VoiceAlertو Notification وهما صفان يرثان من الواجهة ISetting التي تحوي طريقة التنبيه. وعند تغيير طريقة التنبيه من التطبيق يتم استدعاء الصف ChangeSetting الذي بدوره يعمل على تغيير طريقة التنبيه.



|  |
| --- |
| public class Question4 { |
| public static void main(String[] args) { |
| Client client = new Client(); |
|  |
| client.SetReminderStrategy(new VoiceAlert()); |
|  |
| client.SetReminderStrategy(new Notification()); |
| } |
| } |
|  |
|  |
| interface ISetting |
| { |
| String setting(); |
| } |
|  |
| //strategies |
| //Strategy 1:Voice Alter |
| class VoiceAlert implements ISetting |
| { |
| @Override |
| public String setting() |
| { |
| return "Voice alert"; |
| } |
| } |
|  |
| //Strategy 2: Application Notification |
| class Notification implements ISetting |
| { |
| @Override |
| public String setting() |
| { |
| return "Notification"; |
| } |
| } |
|  |
|  |
| class ChangeSetting |
| { |
| private ISetting strategy; |
|  |
| //Change the strategy |
| public void SetReminderStrategy(ISetting strategy){ |
| this.strategy = strategy; |
| } |
| } |

**الطلب الخامس:**

النموذج المستخدم هو من نوع Builder Design Pattern، إذ أنَّ إنشاء الموعد يتم على عدة خطوات/مرحل وأحد المراحل تتغير تبعاً لنوع الموعد لذلك نقوم بخلق صف موعد Appointment يحوي صفات ذلك الموعد كما نقوم بخلق واجهة IAppointmentBuilder تحوي المهام الرئيسية لإنشاء أي موعد أما عن الصف AppointmentBuilder فيقوم بتنجيز هذه المهام (إذ أنه يرث الواجهة IAppointmentBuilder وينجّز كافة توابعها)مع الأخذ بعين الاعتبار نوع الموعد عند القيام بعملية التنجيز في المهام التي تتغير تبعاً لنوع الموعد.



|  |
| --- |
| import java.util.ArrayList; |
| import java.util.Calendar; |
| import java.util.List; |
|  |
|  |
| public class Question5 { |
| public static void main(String[] args) { |
|  |
| AppointmentBuilder builder = new AppointmentBuilder(); |
| builder.SetDate(); |
| builder.SetLocation(); |
| builder.SetCoordinators("event"); |
| } |
|  |
| } |
|  |
| class Appointment{ |
| public Calendar date; |
| public String location; |
| public List<String> coordinators; |
| } |
| interface IAppointmentBuilder |
| { |
| // The appointment creation consist of many phases: set the date, location, coordinators |
| void SetDate(); |
| void SetLocation(); |
|  |
| //some phases change due to the type of the appointment.. so we pass the appointment type to these phases |
| void SetCoordinators(String Appointment\_type); |
| } |
|  |
| class AppointmentBuilder implements IAppointmentBuilder{ |
|  |
| private Appointment appointment; |
|  |
| public AppointmentBuilder(){ |
| this.appointment=new Appointment(); |
| } |
|  |
| @Override |
| public void SetDate() { |
| appointment.date=Calendar.getInstance(); |
| } |
|  |
| @Override |
| public void SetLocation(){ |
| appointment.location="Hamra"; |
| } |
|  |
| @Override |
| public void SetCoordinators(String appointment\_type) { |
| List<String> coords; |
| if(appointment\_type.equals("event")){ |
| coords =new ArrayList<>(2); |
| coords.add("Alaa"); |
| coords.add("hania"); |
| }else if(appointment\_type.equals("conference")){ |
| coords =new ArrayList<>(3); |
| coords.add("Mohammed"); |
| coords.add("Basel"); |
| coords.add("Omar"); |
| } |
| } |
| } |