**كلية الهندسة المعلوماتية**



**السنة الخامسة**

**قسم هندسة البرمجيات ونظم المعلومات**

**وظيفة هندسة البرمجيات -3-**

**Design Pattern**

**المهندس**

**روان قرعوني**

**الطلاب**

**آلاء الحموي**

**هانئة المالكي**

**محمد غانم**

**عمر السابق**

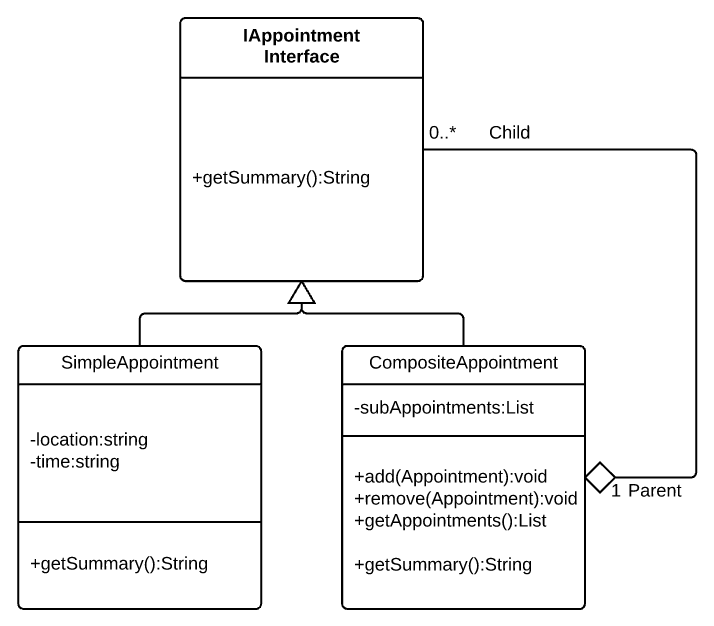
**محمد باسل الشمالي**

**الطلب الأول:**

نموذج التصميم المستخدم هو composite design pattern. لأن الموعد يتكون 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.

كما أن 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:



الواجهة البرمجية IAppointment تحدد سلوك الموعد أو جزء منه بتعريفها إجرائية تعيد تفاصيل الموعد، الصف CompositeAppointment يحقق الواجهة السابقة ويحوي قائمة مواعيد وهي التي تعبر عن المواعيد الجزئية، الصف SimpleAppointment يحقق الواجهة السابقة ويحوي على واصفتان تعبران عن موقع وتوقيت الموعد.

مخطط الصفوف هذا يطابق نموذج التصميم Composite إذ أن الواجهة IAppointment هي Compound والصف CompositeAppointment هو Composite والصف SimpeAppointment هو Leaf.

**كود الطلب الأول:**

|  |
| --- |
| **interface** IAppointment{  **public** **abstract** String getSummary();  }  **class** CompositeAppointment **implements** IAppointment{  **private** List<IAppointment> subAppointmetns;    **public** CompositeAppointment(){  subAppointmetns = **new** ArrayList<>();  }  @Override  **public** String getSummary() {  String summary = "Compsosite Appointment \n\t";  **for** (IAppointment appointment : subAppointmetns) {  summary += appointment.getSummary() +"\n\t";  }  summary += "\n";  **return** summary;  }    **public** **void** addSubAppointment(IAppointment appointment){  subAppointmetns.add(appointment);  }    **public** **void** removeAppointment(IAppointment appointment){  subAppointmetns.remove(appointment);  }    **public** List<IAppointment> getSubAppointment(){  **return** subAppointmetns;  }  }  **class** SimpleAppointment **implements** IAppointment{  **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);  }  } |

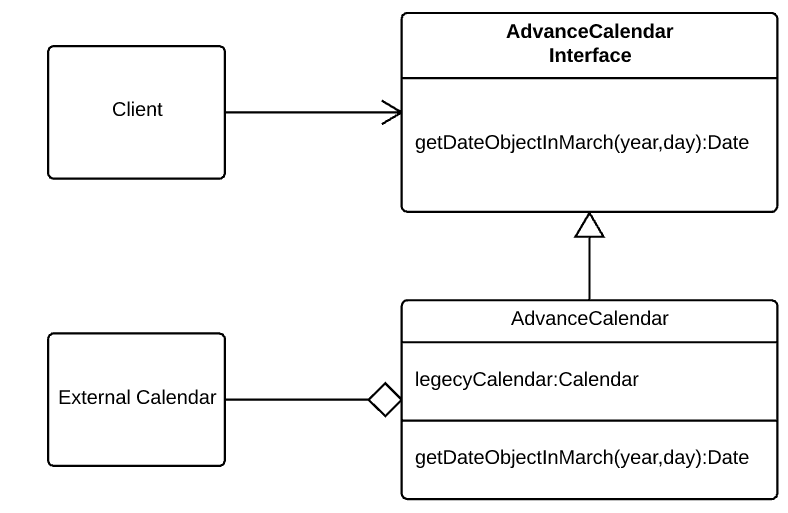
**إختبار الطلب الأول:**

|  |
| --- |
| **الكود:**  **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());  }  } |
| **الخرج:**  Compsosite Appointment  Compsosite Appointment  Simple Appointment { Section 1 , 10:00 AM }  Simple Appointment { Section 2 , 11:00 AM }  Simple Appointment { Section 3 , 11:30 AM }  Simple Appointment { Stand 23 , 5:00 PM } |

**الطلب الثاني:**

نموذج التصميم 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 عبر الإجرائية التي تعيد كائن من صف تاريخ، الصف AdvanceCalendear طبق الواجهة السابقة وهو Adapterويحو على كائن من صف المكتبة الخارجية يستخدمه في عملية الحصول على التاريخ المطلوب، الواجهة External Calendar هو واجهة المكتبة الخارجية وهي Adaptee.

**كود الطلب الثاني:**

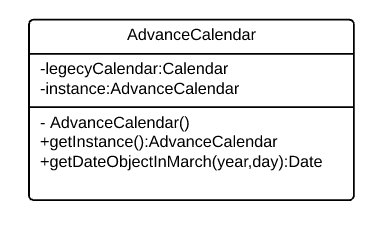
|  |
| --- |
|  |
| **interface** CalenderInterface{  **public** Date getDateInMarch(**int** year,**int** day);  }  **class** AdvanceCalender **implements** CalenderInterface{  **private** Calendar calendar;    **public** AdvanceCalender(){  calendar = Calendar.*getInstance*();  }      @Override  **public** 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();  }  } |
|  |
|  |

**إختبار الطلب الثاني:**

|  |
| --- |
| **الكود:**  **public** **class** Question2 {  **public** **static** **void** main(String[] args) {  CalenderInterface calender = **new** AdvanceCalender();  System.***out***.println(calender.getDateInMarch(2016, 1));  }  } |
| **الخرج:**  Tue Mar 01 22:46:33 EET 2016 |

**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.



**كود الطلب الثالث:**

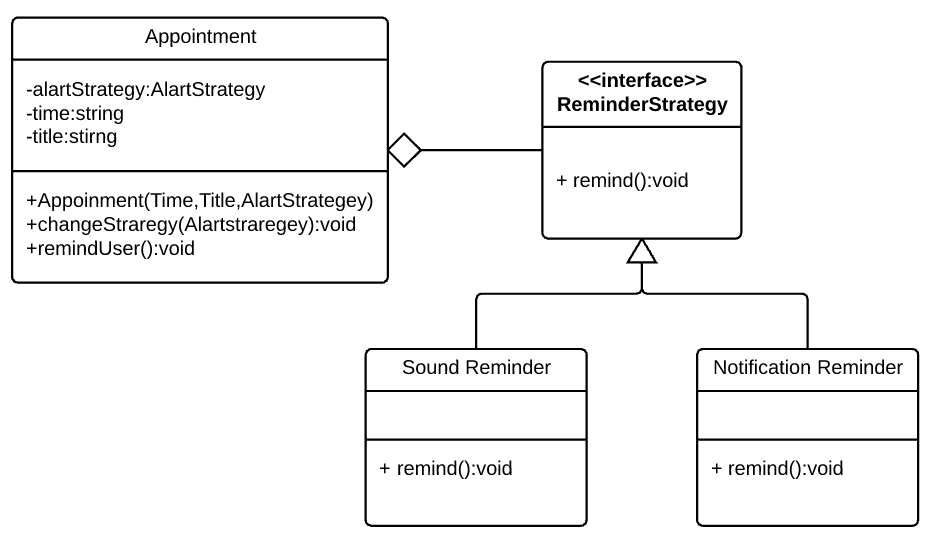
|  |
| --- |
|  |
| **class** SingltonAdvanceCalender **implements** CalenderInterface{    **private** **static** SingltonAdvanceCalender *instance*;  **private** Calendar calendar;  **private** SingltonAdvanceCalender(){  calendar = Calendar.*getInstance*();  }    **public** **static** SingltonAdvanceCalender getInstance(){  **if** (*instance* == **null**){  System.***out***.println("Create instance for the first time");  *instance* = **new** SingltonAdvanceCalender();  **return** *instance*;  }**else**{  System.***out***.println("Reuse the existing instance");  **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();  }  } |
|  |
|  |

**إختبار الطلب الثالث:**

|  |
| --- |
| **الكود:**  **public** **class** Question3 {  **public** **static** **void** main(String[] args) {  CalenderInterface calender = SingltonAdvanceCalender.*getInstance*();  System.***out***.println(calender.getDateInMarch(2016, 1)+"\n");  calender = SingltonAdvanceCalender.*getInstance*();  System.***out***.println(calender.getDateInMarch(2016, 3)+"\n");  calender = SingltonAdvanceCalender.*getInstance*();  System.***out***.println(calender.getDateInMarch(2016, 7)+"\n");  calender = SingltonAdvanceCalender.*getInstance*();  System.***out***.println(calender.getDateInMarch(2016, 23)+"\n");  }  } |
| **الخرج:**  Create instance for the first time  Tue Mar 01 23:02:43 EET 2016  Reuse the existing instance  Thu Mar 03 23:02:43 EET 2016  Reuse the existing instance  Mon Mar 07 23:02:43 EET 2016  Reuse the existing instance  Wed Mar 23 23:02:43 EET 2016 |

**الطلب الرابع:**

نموذج التصميم المستخدم هو من نوع Straregy Design Pattern، لأنه 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 Sound Reminder و Notification Reminder وهما صفان يرثان من الواجهة ReminderStrategy التي تعرف طريقة التنبيه. وعند تغيير طريقة التنبيه من التطبيق يتم استدعاء إجرائية ChangeStrategy على الموعد.



**كود الطلب الرابع:**

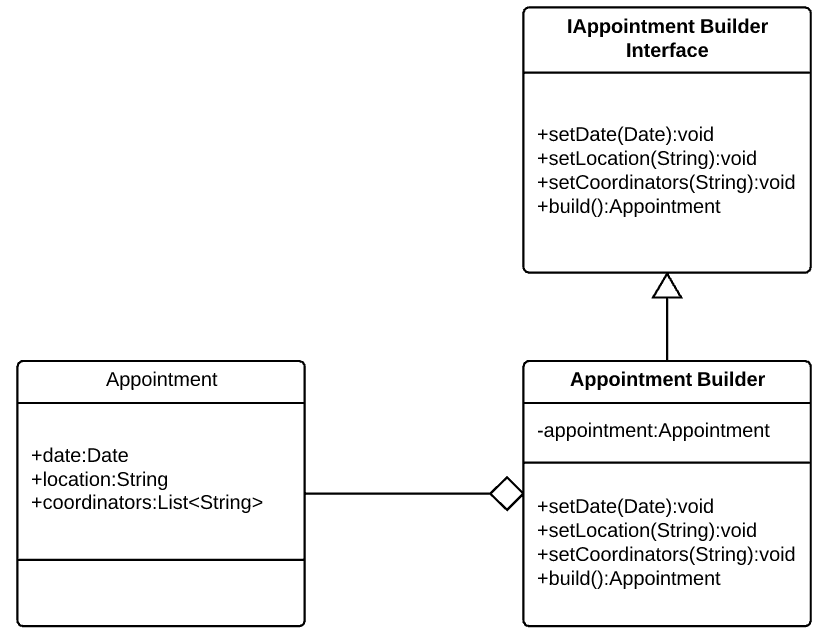
|  |
| --- |
|  |
| **class** Appointment{  **private** String title;  **private** String time;  **private** RemindingStrategy remindingStrategy;    **public** Appointment(String title,String time,RemindingStrategy rs){  **this**.time = time;  **this**.title = title;  **this**.remindingStrategy = rs;  }    **public** **void** changeReminderStrategy(RemindingStrategy remindingStrategy){  **this**.remindingStrategy = remindingStrategy;  }    **public** **void** reminderUser(){  System.***out***.print(remindingStrategy.remind());  System.***out***.println(" - "+title + " - " +time);  }  }  **interface** RemindingStrategy  {  **public** String remind();  }  **class** VoiceReminder **implements** RemindingStrategy  {  @Override  **public** String remind()  {  **return** "Voice Reminder";  }  }    **class** NotificationReminder **implements** RemindingStrategy  {  @Override  **public** String remind()  {  **return** "Notification Reminder";  }  } |
|  |
|  |

**إختبار الطلب الرابع:**

|  |
| --- |
| **الكود:**  **public** **class** Question4 {  **public** **static** **void** main(String[] args) {  Appointment appointment = **new** Appointment(  "Design patter mini-project",  "10:00",  **new** NotificationReminder());  appointment.reminderUser();  System.***out***.println("User change reminding method from setting");  appointment.changeReminderStrategy(**new** VoiceReminder());  appointment.reminderUser();  }  } |
| **الخرج:**  Notification Reminder - Do design patter mini-project - 10:00  User change reminding method from setting  Voice Reminder - Do design patter mini-project - 10:00 |

**الطلب الخامس:**

نموذج التصميم 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.



**كود الطلب الخامس:**

|  |
| --- |
|  |
| **class** Appointment{  **public** Date date;  **public** String location;  **public** List<String> coordinators;    @Override  **public** String toString(){  String string = "Date : " + date;  string += "\nLocation :" + location;  string += "\nCoordinators :" + coordinators;  **return** string;  }  }  **interface** IAppointmentBuilder  {  **void** SetDate(Date date);  **void** SetLocation(String locatoin);  **void** SetCoordinators(String Appointment\_type);  Appointment Build();  }  **class** AppointmentBuilder **implements** IAppointmentBuilder{  **private** Appointmentappointment;    **public** AppointmentBuilder(){  **this**.appointment = **new** Appointment();  }    @Override  **public** Appointment Build() {  **return** appointment;  }    @Override  **public** **void** SetDate(Date date) {  appointment.date=date;  }    @Override  **public** **void** SetLocation(String location){  appointment.location=location;  }  @Override  **public** **void** SetCoordinators(String appointment\_type) {  appointment.coordinators = **new** ArrayList<>();  **if**(appointment\_type.equals("event")){  appointment.coordinators.add("Alaa");  appointment.coordinators.add("Hania");  }**else** **if**(appointment\_type.equals("conference")){  appointment.coordinators.add("Mohammed");  appointment.coordinators.add("Basel");  appointment.coordinators.add("Omar");  }  }  } |
|  |
|  |

**إختبار الطلب الخامس:**

|  |
| --- |
| **الكود:**  **public** **class** Question5 {  **public** **static** **void** main(String[] args) {    AppointmentBuilder builder = **new** AppointmentBuilder();  builder.SetDate(Calendar.*getInstance*().getTime());  builder.SetLocation("Hamra");  builder.SetCoordinators("event");      Appointment appointment = builder.Build();  System.***out***.println(appointment.toString());    builder.SetCoordinators("conference");  appointment = builder.Build();  System.***out***.println(appointment.toString());  }  } |
| **الخرج:**  Date : Sun Dec 04 00:42:51 EET 2016  Location :Hamra  Coordinators :[Alaa, Hania]  Date : Sun Dec 04 00:42:51 EET 2016  Location :Hamra  Coordinators :[Mohammed, Basel, Omar] |