جامعة دمشق
كلية الهندسة المعلوماتية
السنة الخامسة
قسم هندسة البرمجيات ونظم المعلومات
الفصل الأول

هندسة برمجيّات /3/

تطبيق لجلب خادمة إلى المنزل

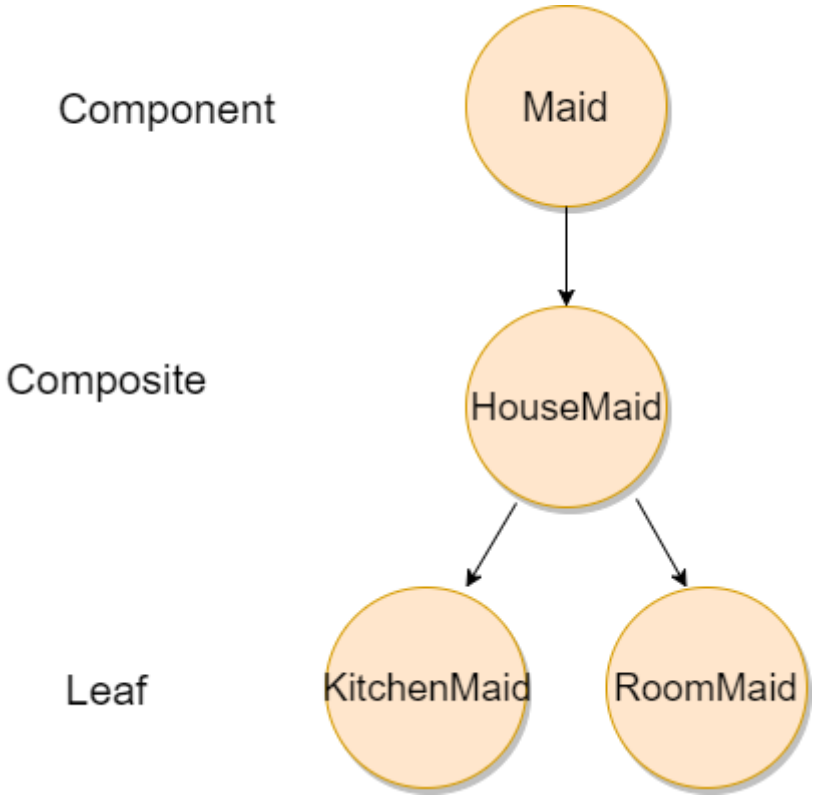
إعداد الطالبة

رولا الرهبان

إشراف المهندس

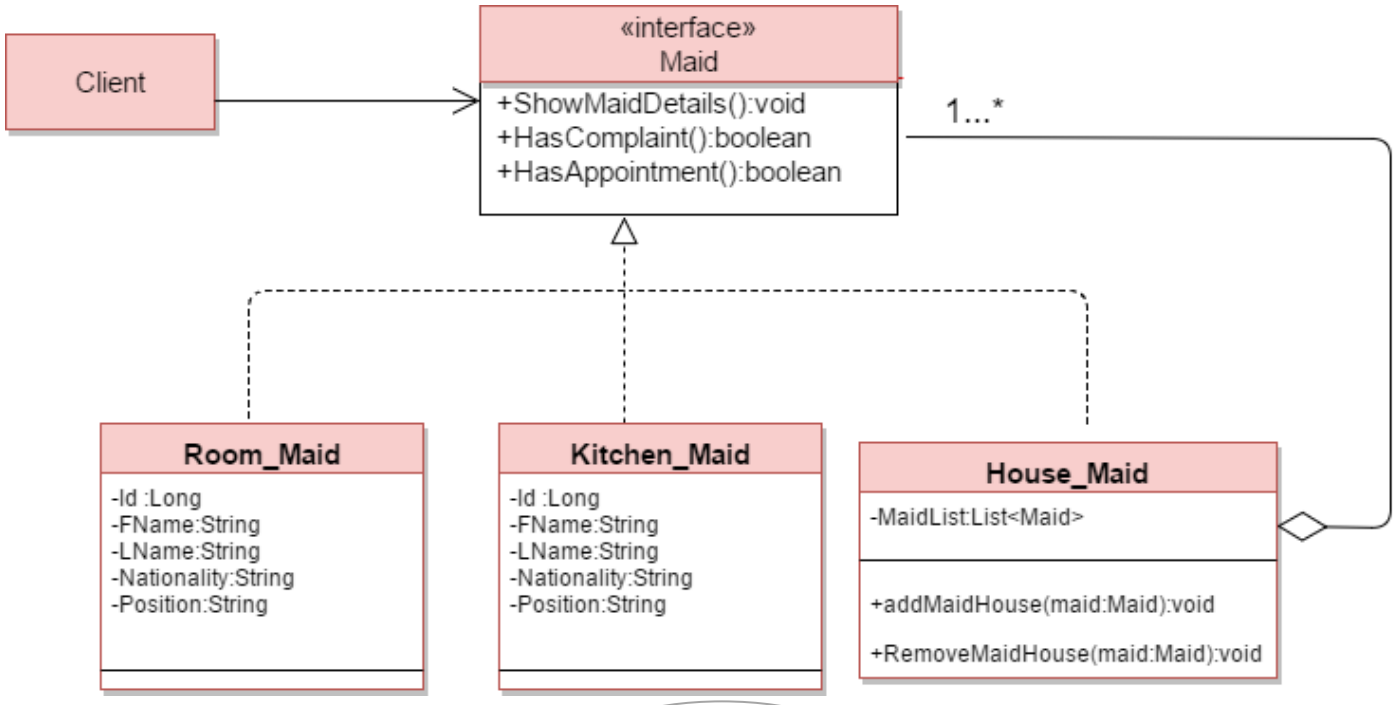
روان قرعوني

- باعتبار composite design pattern يصف كيفية حل مشكلات التصميم المتكررة ويمثل التسلسل الهرمي للجزء الكامل بحيث يمكن للclients التعامل مع الكائنات بشكل موحد فيمكن استخدامه لتصنيف الخادمات في تطبيقنا على شكل tree كالتالي ..



ال Design pattern المستخدم في هذا الطلب هو Composite ونوعه structural

Class Diagram



Code with Java:

Component interface

```java
public interface Maid {
    public void ShowMaidDetails();
    public boolean HasComplaint();
    public boolean HasAppointment();
}
```

Composite Class

```java
public class HouseMaid implements Maid {

    private List<Maid> MaidList ;
     @Override
    public void ShowMaidDetails() {
        for(Maid maid:MaidList)
        {
            maid.ShowMaidDetails();
        }
    }
     public void addMaidHouse(Maid maid)
    {
        MaidList.add(maid);
    }

    public void removeMaidHouse(Maid maid)
    {
        MaidList.remove(maid);
    }

    @Override
    public boolean HasComplaint() {
      return false;
    }

    @Override
    public boolean HasAppointment() {
        return false;
    }
```

```java
public class KitchenMaid implements Maid {

    private long Id;
    private String FName;
    private String LName;
    private String Nationality;
    private String Position;

    public KitchenMaid(long Id, String FName, String LName, String Nationality) {
        this.Id = Id;
        this.FName = FName;
        this.LName = LName;
        this.Nationality = Nationality;
        this.Position = "Kitchen";
    }

    @Override
    public void ShowMaidDetails() {
        System.out.println(Id + " " + FName + " " + LName + " " + Nationality + " " + Position);
    }

    public long getId() {
        return Id;
    }

    public void setId(long Id) {
        this.Id = Id;
    }

    public String getFName() {
        return FName;
```

```java
public class RoomMaid implements Maid {
    private long Id;
    private String FName;
    private String LName;
    private String Nationality;
    private String Position;

    public RoomMaid(long Id, String FName, String LName, String Nationality) {
        this.Id = Id;
        this.FName = FName;
        this.LName = LName;
        this.Nationality = Nationality;
        this.Position = "Rooms";
    }

    public long getId() {
        return Id;
    }

    public void setId(long Id) {
        this.Id = Id;
    }

    public String getFName() {
        return FName;
    }

    public void setFName(String FName) {
        this.FName = FName;
    }
```

In Main Method

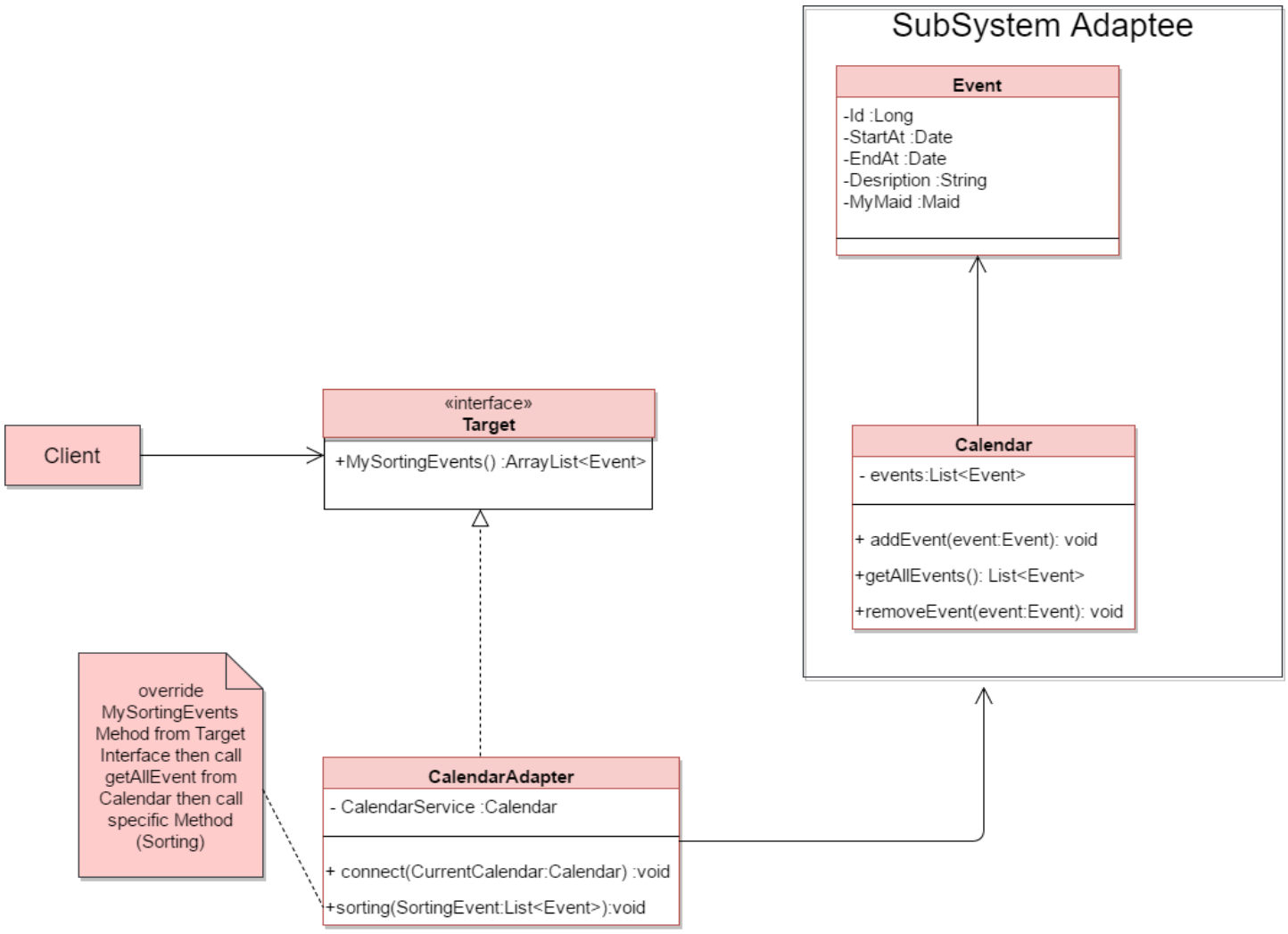
```
19      public static void main(String[] args) {
20          // TODO code application logic here
21          //1
22          Maid room1=new RoomMaid(1, "Roula", "Rohban", "Syrian");
23          Maid room2=new RoomMaid(2, "Rahaf", "Diab", "Syrian");
24          Maid kitchen1=new KitchenMaid(3, "Tamara", "marie", "Syrian");
25         HouseMaid House=new HouseMaid();
26          House.addMaidHouse(room1);
27          House.addMaidHouse(room2);
28          House.addMaidHouse(kitchen1);
29          House.ShowMaidDetails();
30
```

الطلب الثاني

- لدي class Calendar يحوي تابع لعرض جميع الeventsل clientما , نريد استخدام هذه الخدمة مع التعديل عليها بحيث يتم ترتيب المواعيد قبل عرضهم للclient , لذلك نلجأ إلى Adapter Design Pattern الذي يوّفر واجهة متوافقة للربط بين الcalendar Service والTarget الذي يطلبه الزبون

الDesign Pattern المستخدم في هذا الطلب هو Adapter Design pattern ونوعه Structural

Class Diagram



Code With Java :

Target Interface

```
 */
public interface Target {
    public ArrayList<Event> MySortingEvents();

}
```

Implement The Target interface with the Adapter Class

```
public class CalendarAdapter implements Target{
    private Calendar CalendarService;
    public void connect(Calendar currentCalendar)
    {
        this.CalendarService=currentCalendar;
    }
    @Override
    public ArrayList<Event> MySortingEvents() {
        //get events without sorting from Calendar library
        ArrayList<Event> events=(ArrayList<Event>) CalendarService.getAllEvents();
        //then sorting my events
        sorting(events);
        //then return the sorting events
        return events;
    }

    public void sorting(ArrayList<Event> SortingEvents)
    {
```

SubSystem Adaptee

```
public class Calendar {

    private List<Event> events = new ArrayList<Event>();

    List<Event> getAllEvents() {
        return events;
    }

    public void removeEvent(Event event) {
        events.remove(event);
    }

    public void addEvent(Event event) {
        events.add(event);
    }
```

```
public class Event {
    private long Id;
    private Date StartAt;
    private Date EndAt;
    private String Desription;
    private Maid MyMaid;

    public Event(long Id, Date StartAt, Date EndAt, String Desription, Maid MyMaid) {
        this.Id = Id;
        this.StartAt = StartAt;
        this.EndAt = EndAt;
        this.Desription = Desription;
        this.MyMaid = MyMaid;
    }

    @Override
    public String toString() {
        return Id + " "+ StartAt +" "+ EndAt + " " + Desription + " " + MyMaid ;
    }
```

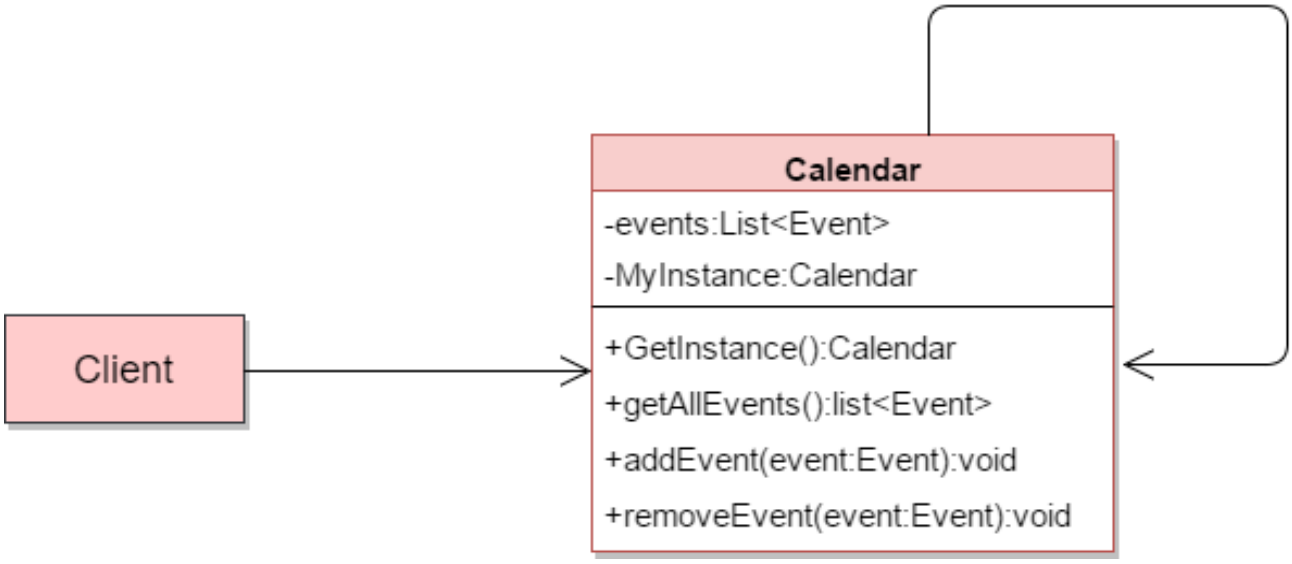
```java
//2
Target T=new CalendarAdapter();
List<Event> myEvents=T.MySortingEvents();
for(Event event:myEvents)
{
    System.out.println(event);
}
```

الطلب الثالث

- نريد التعامل مع نسخة واحدة من الcalendar طول فترة حياة البرنامج وبالتالي نموذج التصميم المناسب هو Singleton Design Pattern , بحيث سيكون لدينا Object واحد من class Calendar على مستوى الApllication ويكون الوصول للUnique Object بشكل Globally

الDesign Pattern المستخدم في هذا الطلب هو Singleton Design pattern ونوعه Creational

Class Diagram



Code with Java

Implement Singleton class

```
15  public class Calendar {
16
17      private List<Event> events ;
18      private static Calendar MyInstance = null;
19
20      private Calendar() {
21      }
22
23      public static Calendar GetInstance() {
24          if (MyInstance == null) {
25              MyInstance = new Calendar();
26          }
27          return MyInstance;
28      }
29
30      List<Event> getAllEvents() {...3 lines }
33
34      public void removeEvent(Event event) {...3 lines }
37
38      public void addEvent(Event event) {...3 lines }
41
42  }
43
```

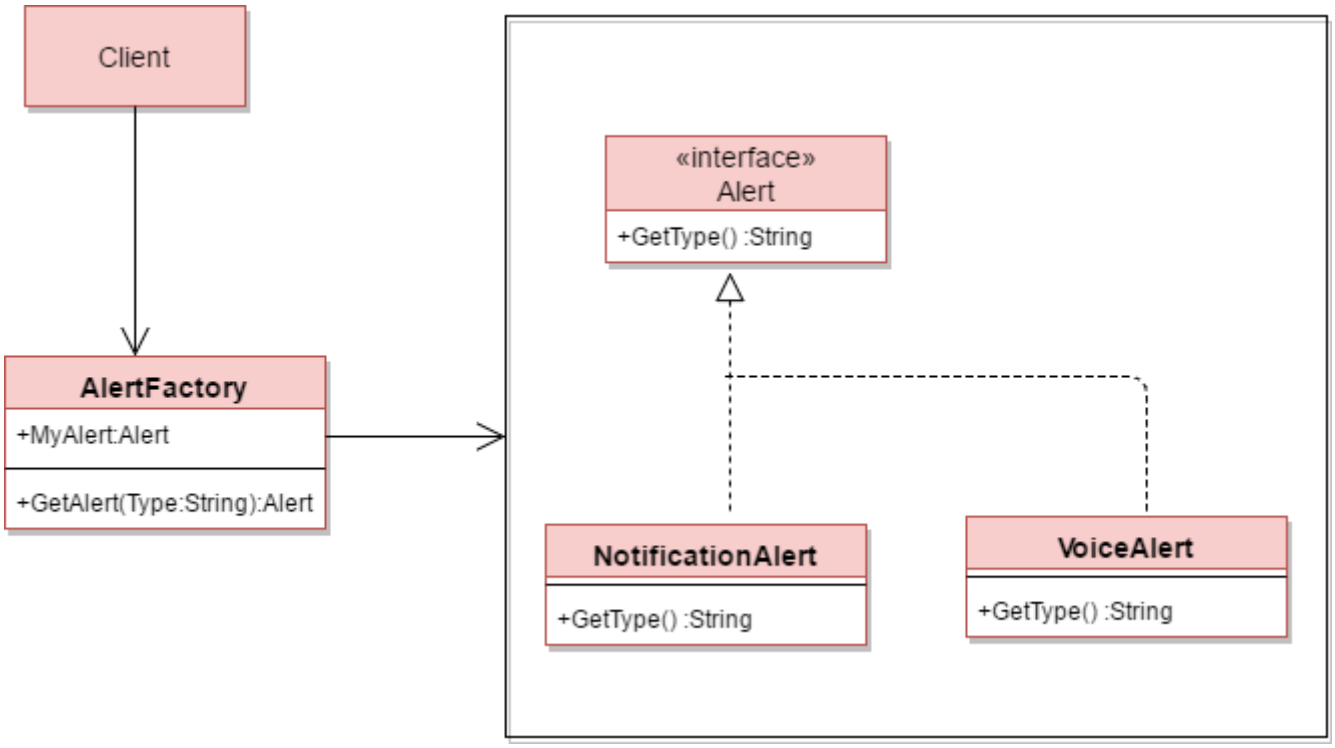
Call the Unique instance of Calendar

```
41
42          //3
43
44          Calendar Unique = Calendar.GetInstance();
```

الطلب الرابع

سنلجأ في هذا الطلب إلى Factory Method Design pattern لتفويض مسؤولية خلق غرض من (تنبيه) سواء كان (إشعار , صوت) لكلاس آخر هو AlertFactory وهو Design Pattern نوعه Creational .

Class Diagram



Code With Java :

Alert Interface :

```java
public interface Alert {
    public String GetType();
}
```

NotificationAlert Class :

```java
public class NotificationAlert implements Alert{

    @Override
    public String GetType() {
        return "Notification";
    }

}
```

VoiceAlert Class :

```java
public class VoiceAlert implements Alert{

    @Override
    public String GetType() {
        return "Voice";
    }

}
```

AlertFactory Method :

```java
public class AlertFactory {

    Alert MyAlert;

    public Alert GetAlert(String Type) {
        if (Type.equals("Notifaction")) {
            MyAlert = new NotificationAlert();
        } else if (Type.equals("Voice")) {
            MyAlert = new VoiceAlert();
        }
        return MyAlert;
    }
}
```

IN Main Method :

```java
    //4

    AlertFactory alert=new AlertFactory();
    Alert MyAlert= alert.GetAlert("Voice");
    //MyAlert.GetType(); // print Voice
```

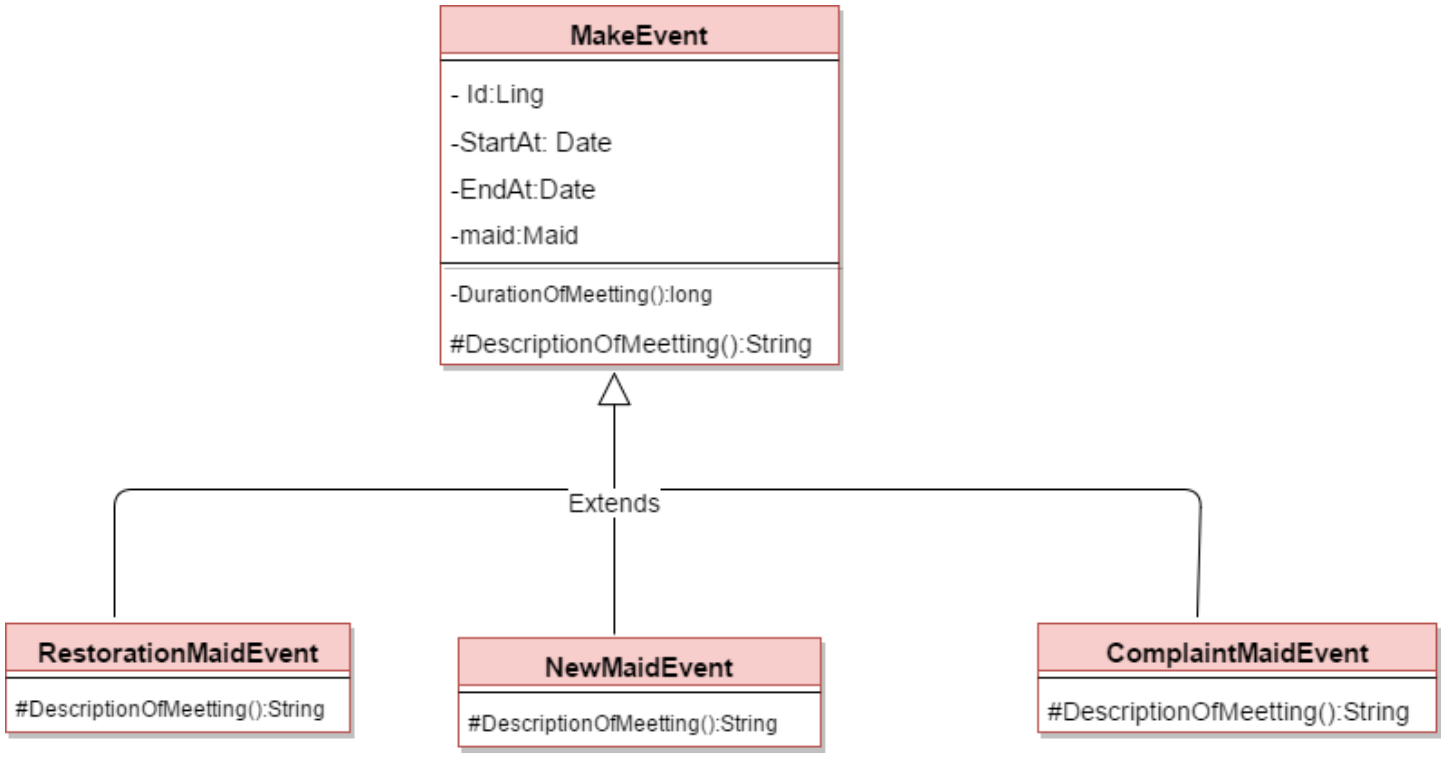
الطلب الخامس

- باعتبار أن الBehavioral Design Pattern تركز على الطرق التي تتعاون بها الكائنات الفردية لتحقيق هدف مشترك , حيث يتم ترتيب الخطوات بنفس الطريق مع وجود اختلاف في التنفيذ في بعض الخطوات , يكون ال Design pattern المناسب لهذا الطلب هو template design pattern (فهو يعتبر تطبيق عملي للتعميم والوراثة)

ففي مثالنا الهدف المشترك هو إنشاء موعد , فيه توابع (خطوات) مشتركة بين (خادمة جديدة , شكوى على خادمة , إعادة خادمة) فالخطوات الثابتة لدينا هي حساب زمن الموعد DurationOfMeetting وتحديد زمن بداية المقابلة وزمن نهاية المقابلة والخادمة المعنيّة , أما الخطوة التي تختلف بين الأنواع الثلاث للموعد التي ذكرناها سابقا نجعلها protected abstract , في مثالنا هي (DescriptionOfMeetting) أي سبب الاجتماع

Design Pattern المستخدم في هذا الطلب هو Template وهو من نوع Behavioral

Class Diagram



Code with Java:

كلاس إنشاء موعد :

```java
public abstract class Event {

    private long Id;
    private Date StartAt;
    private Date EndAt;
    private Maid MyMaid;

    public Event(long Id, Date StartAt, Date EndAt, Maid MyMaid) {
        this.Id = Id;
        this.StartAt = StartAt;
        this.EndAt = EndAt;
        this.MyMaid = MyMaid;
    }

    ///Step which difference between Child
    protected abstract String DescriptionOfMeetting();

    /////constante in each child
    private long DurationOfMeetting() {
        return EndAt.getTime() - StartAt.getTime();

    }

    public long getId() {
        return Id;
    }

    public void setId(long Id) {
```

كلاس موعد خادمة جديدة :

```java
public class NewMaidEvent extends Event{

    public NewMaidEvent(long Id, Date StartAt, Date EndAt, Maid MyMaid) {
        super(Id, StartAt, EndAt, MyMaid);
    }

    @Override
    protected String DescriptionOfMeetting() {
        return "For Add " + MyMaid.toString();
    }
}
```

كلاس موعد إعادة خادمة :

```java
public class RestorationMaidEvent extends Event{

    public RestorationMaidEvent(long Id, Date StartAt, Date EndAt, Maid MyMaid) {
        super(Id, StartAt, EndAt,  MyMaid);
    }

    @Override
    protected String DescriptionOfMeetting() {
        return "For Restoration" + MyMaid.toString();
    }
}
```

كلاس موعد تسجيل شكوى على خادمة :

```java
public class ComplaintMaidEvent extends Event{

    public ComplaintMaidEvent(long Id, Date StartAt, Date EndAt, Maid MyMaid) {
        super(Id, StartAt, EndAt, MyMaid);
    }

    @Override
    protected String DescriptionOfMeetting() {
        return " Register a complaint to" + MyMaid.toString();
    }

}
```

يكون الclass Diagram للتطبيق كاملاً: