## 5. 中介者（Mediator）

### Intent

集中相关对象之间复杂的沟通和控制方式。

### Class Diagram

* Mediator：中介者，定义一个接口用于与各同事（Colleague）对象通信。
* Colleague：同事，相关对象

### Implementation

Alarm（闹钟）、CoffeePot（咖啡壶）、Calendar（日历）、Sprinkler（喷头）是一组相关的对象，在某个对象的事件产生时需要去操作其它对象，形成了下面这种依赖结构：

使用中介者模式可以将复杂的依赖结构变成星形结构：

public abstract class Colleague {  
 public abstract void onEvent(Mediator mediator);  
}

public class Alarm extends Colleague {  
  
 @Override  
 public void onEvent(Mediator mediator) {  
 mediator.doEvent("alarm");  
 }  
  
 public void doAlarm() {  
 System.out.println("doAlarm()");  
 }  
}

public class CoffeePot extends Colleague {  
 @Override  
 public void onEvent(Mediator mediator) {  
 mediator.doEvent("coffeePot");  
 }  
  
 public void doCoffeePot() {  
 System.out.println("doCoffeePot()");  
 }  
}

public class Calender extends Colleague {  
 @Override  
 public void onEvent(Mediator mediator) {  
 mediator.doEvent("calender");  
 }  
  
 public void doCalender() {  
 System.out.println("doCalender()");  
 }  
}

public class Sprinkler extends Colleague {  
 @Override  
 public void onEvent(Mediator mediator) {  
 mediator.doEvent("sprinkler");  
 }  
  
 public void doSprinkler() {  
 System.out.println("doSprinkler()");  
 }  
}

public abstract class Mediator {  
 public abstract void doEvent(String eventType);  
}

public class ConcreteMediator extends Mediator {  
 private Alarm alarm;  
 private CoffeePot coffeePot;  
 private Calender calender;  
 private Sprinkler sprinkler;  
  
 public ConcreteMediator(Alarm alarm, CoffeePot coffeePot, Calender calender, Sprinkler sprinkler) {  
 this.alarm = alarm;  
 this.coffeePot = coffeePot;  
 this.calender = calender;  
 this.sprinkler = sprinkler;  
 }  
  
 @Override  
 public void doEvent(String eventType) {  
 switch (eventType) {  
 case "alarm":  
 doAlarmEvent();  
 break;  
 case "coffeePot":  
 doCoffeePotEvent();  
 break;  
 case "calender":  
 doCalenderEvent();  
 break;  
 default:  
 doSprinklerEvent();  
 }  
 }  
  
 public void doAlarmEvent() {  
 alarm.doAlarm();  
 coffeePot.doCoffeePot();  
 calender.doCalender();  
 sprinkler.doSprinkler();  
 }  
  
 public void doCoffeePotEvent() {  
 // ...  
 }  
  
 public void doCalenderEvent() {  
 // ...  
 }  
  
 public void doSprinklerEvent() {  
 // ...  
 }  
}

public class Client {  
 public static void main(String[] args) {  
 Alarm alarm = new Alarm();  
 CoffeePot coffeePot = new CoffeePot();  
 Calender calender = new Calender();  
 Sprinkler sprinkler = new Sprinkler();  
 Mediator mediator = new ConcreteMediator(alarm, coffeePot, calender, sprinkler);  
 // 闹钟事件到达，调用中介者就可以操作相关对象  
 alarm.onEvent(mediator);  
 }  
}

doAlarm()  
doCoffeePot()  
doCalender()  
doSprinkler()

### JDK

* All scheduleXXX() methods of [java.util.Timer](http://docs.oracle.com/javase/8/docs/api/java/util/Timer.html)
* [java.util.concurrent.Executor#execute()](http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/Executor.html#execute-java.lang.Runnable-)
* submit() and invokeXXX() methods of [java.util.concurrent.ExecutorService](http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ExecutorService.html)
* scheduleXXX() methods of [java.util.concurrent.ScheduledExecutorService](http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ScheduledExecutorService.html)
* [java.lang.reflect.Method#invoke()](http://docs.oracle.com/javase/8/docs/api/java/lang/reflect/Method.html#invoke-java.lang.Object-java.lang.Object...-)