Design Patterns

Mediator Pattern

ebru@hacettepe.edu.tr

ebruakcapinarsezer@gmail.com

http://yunus.hacettepe.edu.tr/~ebru/

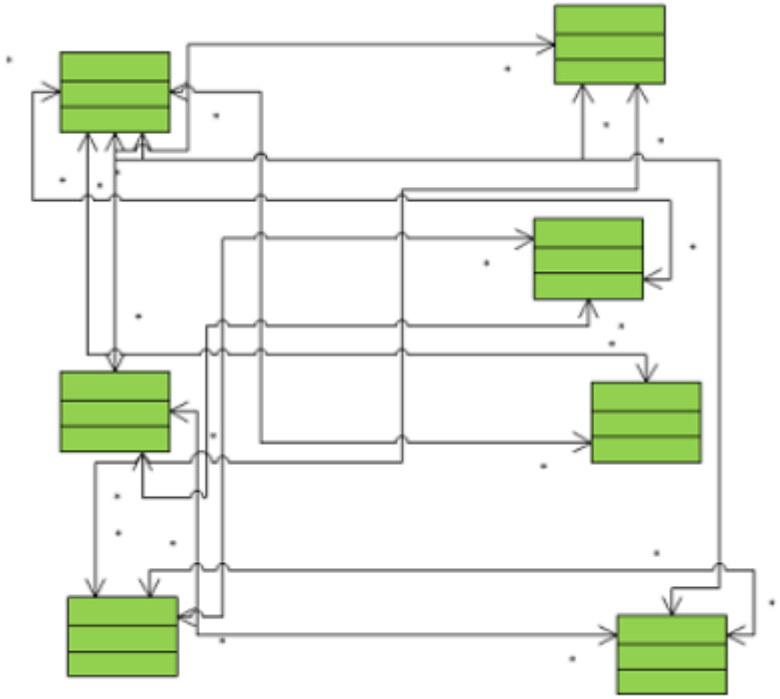
@ebru176

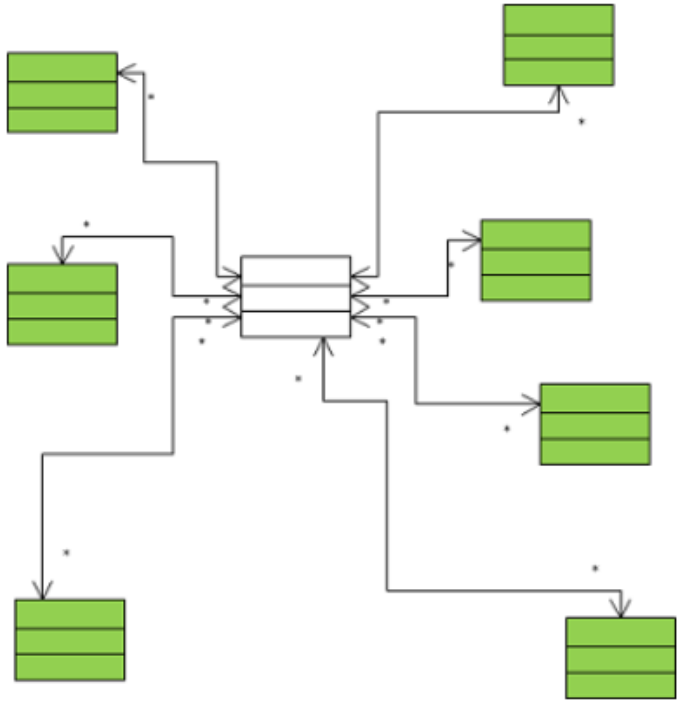
Aralık 2017

What is a mediator

- An object that abstracts inter-workings of two or more objects.
- Acts as communication hub that serve to decouple objects — no need to know about each other, just need to know their Mediator.
- Promotes loose coupling by keeping objects from referencing to each other explicitly

Before and After Mediator





Classification and Intent

- Classification: Object Behavior
- Encapsulate object-to-object communication
- Keeps objects from knowing about each other directly; this allows us easily change an object's behavior

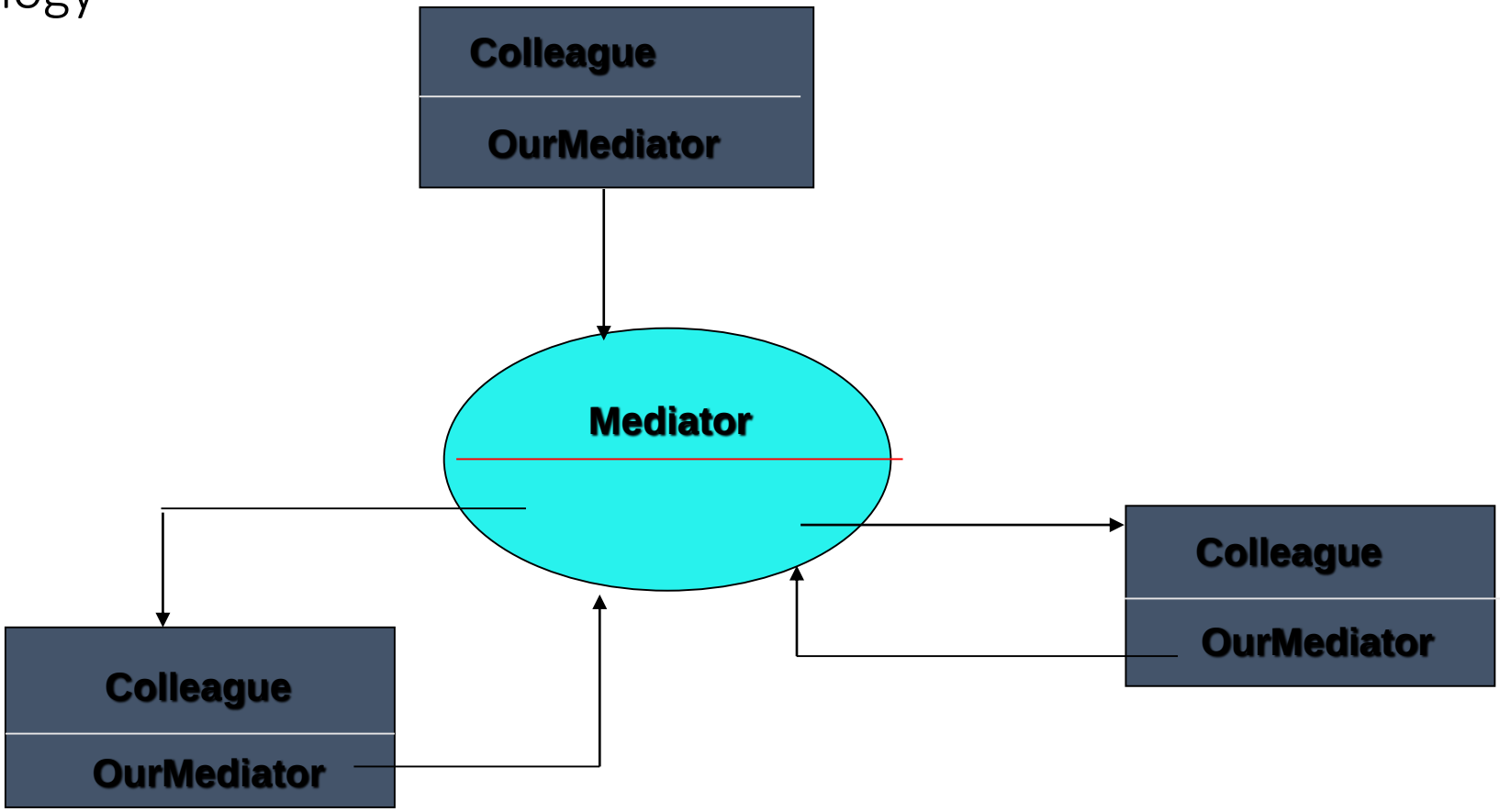
When to use a mediator?

- When one or more objects must interact with several different objects.
- When centralized control is desired
- When simple object need to communicate in complex ways.
- When you want to reuse an object that frequently interacts with other objects

Motivation

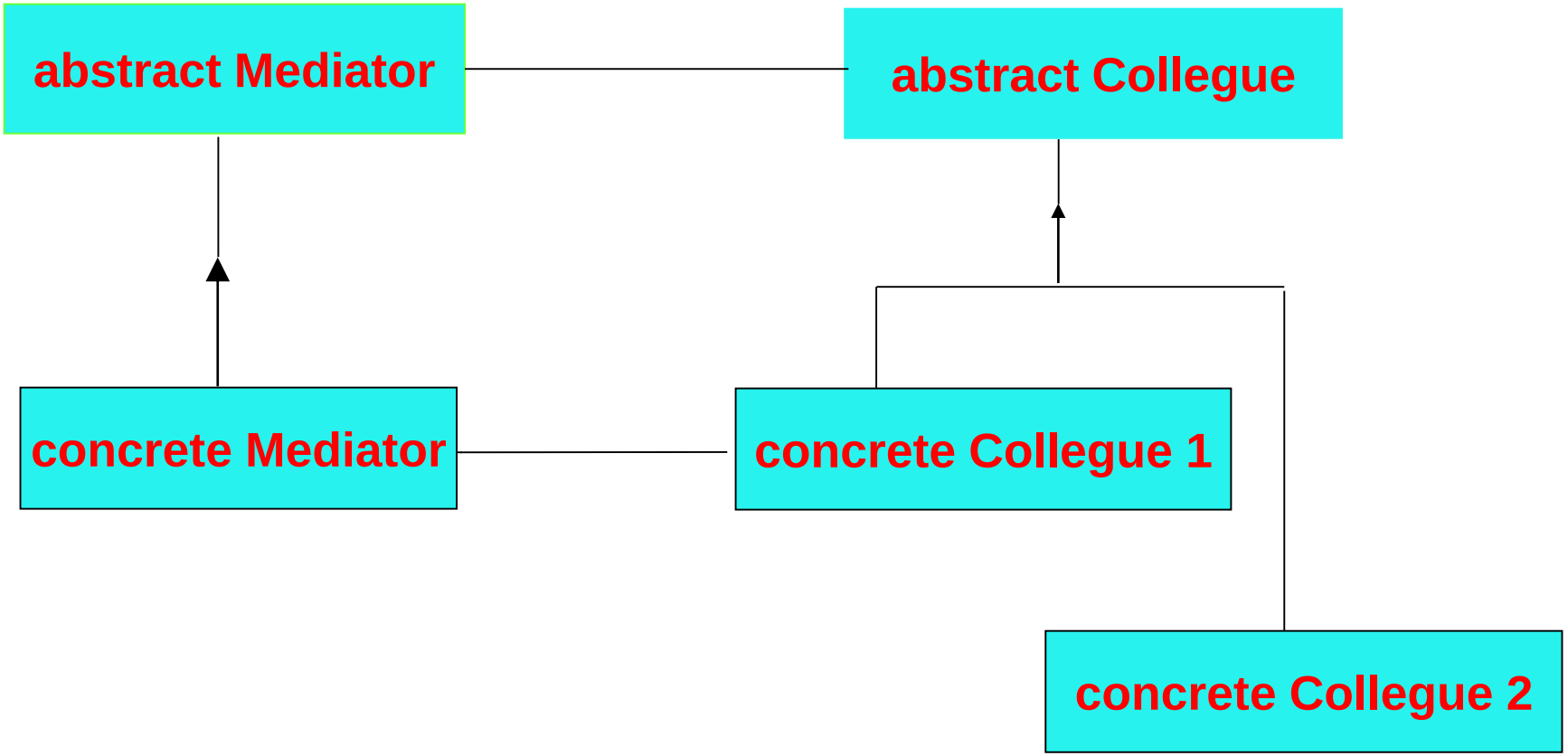
- OO-design allows for more reusable and more elegant programs, when objects need to refer to each other, this elegance is often lost.
- By consolidating all interaction in a single class, we can regain elegance and reusability

Mediator: a system of at least 3 objects messaging in a star topology



Provides a common connection point, centralized (subclassable) behavior and behavior mgmt, all with a common interface

Structure


abstract Mediator
abstract Collegue
concrete Mediator
concrete Collegue 1
concrete Collegue 2

Participant--Abstract Mediator

Define the Collegue-to-Mediator interface

Participant--Concrete Mediator

- Derivated from Abstract Mediator
- Aware of and knows the purpose of all concrete Collegues
- Receives messages from a colleague & sends necessary commands to other colleagues

Participant--Abstract Colleague

- Define the Mediator-to- Colleague interface
- Knows about the mediator, but none of its colleagues

Participant--Concrete Colleagues

- Derived from abstract Colleague
- Each Concrete Colleague knows its own behavior on a small scale, but NOT on a large scale.

Mediator & the Rest of the World

Related Design Pattern

- Observer: the Mediator class may be implemented using an Observer
- Façade: is similar to a Mediator, but with one-way communication from the Façade to its subsystem classes.

Conclusion

- A mediator is an objcet-behavior design pattern.
- Use a Mediator when simple objects interact in complex ways
- The Mediator pattern consists of two types of objects: the Mediator and its Colleagues
- All object-to-object communication is encapsulated in the Mediator
- Mediator allows for greater reusability, and generally more elegant, readable code.

Landing problem

```java
package com.javapapers.designpattern.mediator;

public interface IATCMediator {

        public void registerRunway(Runway runway);

        public void registerFlight(Flight flight);

        public boolean isLandingOk();

        public void setLandingStatus(boolean status);
}
```

```java
package com.javapapers.designpattern.mediator;

public class ATCMediator implements IATCMediator {
        private Flight flight;
        private Runway runway;
        public boolean land;

        public void registerRunway(Runway runway) {
                this.runway = runway;
        }

        public void registerFlight(Flight flight) {
                this.flight = flight;
        }

        public boolean isLandingOk() {
                return land;
        }

        @Override
        public void setLandingStatus(boolean status) {
                land = status;

        }
}
```

```java
package com.javapapers.designpattern.mediator;

public interface Command {
        void land();
}
```

```java
package com.javapapers.designpattern.mediator;

public class MediatorDesignPattern {
        public static void main(String args[]) {

                IATCMediator atcMediator = new ATCMediator();
                Flight sparrow101 = new Flight(atcMediator);
                Runway mainRunway = new Runway(atcMediator);
                atcMediator.registerFlight(sparrow101);
                atcMediator.registerRunway(mainRunway);
                sparrow101.getReady();
                mainRunway.land();
                sparrow101.land();
        }
}
```

```java
package com.javapapers.designpattern.mediator;

public class Flight implements Command {
        private IATCMediator atcMediator;

        public Flight(IATCMediator atcMediator) {
                this.atcMediator = atcMediator;
        }

        public void land() {
                if (atcMediator.isLandingOk()) {
                        System.out.println("Landing done....");
                        atcMediator.setLandingStatus(true);
                } else
                        System.out.println("Will wait to land....");
        }

        public void getReady() {
                System.out.println("Getting ready...");
        }
```

```java
age com.javapapers.designpattern.mediator;

ic class Runway implements Command {
        private IATCMediator atcMediator;

        public Runway(IATCMediator atcMediator) {
                this.atcMediator = atcMediator;
                atcMediator.setLandingStatus(true);
        }

        @Override
        public void land() {
                System.out.println("Landing permission granted...");
                atcMediator.setLandingStatus(true);
        }

}
```