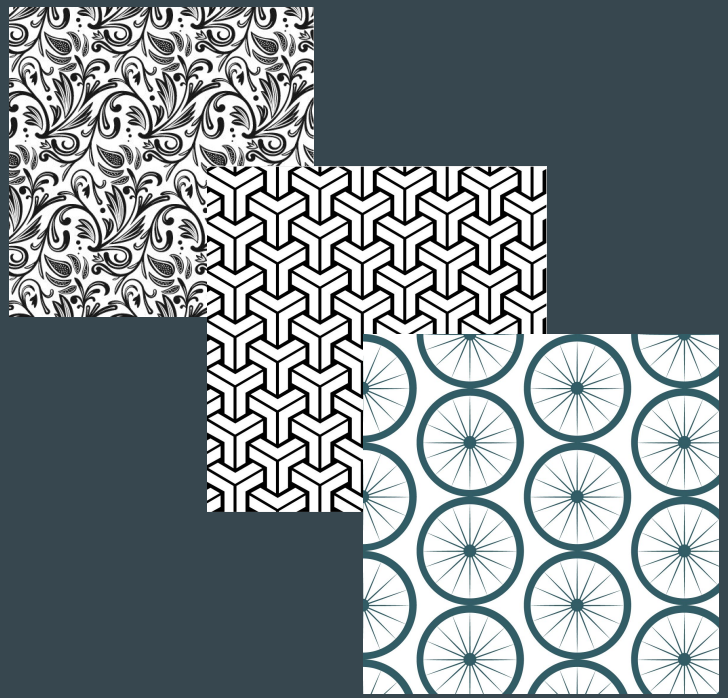
Patrones de Diseño

75.10 - Técnicas de Diseño



Problema

```
public List<T> InvokeAll (List<Callable<T>> tasks ) throws Exception {
   List<T> results = new ArrayList<T>();
   for (Callable<T> task : tasks) {
       results.add(task.call());
   }
   return results;
}
```

Problema

```
public class RandomTask implements Callable<Integer> {

    private Random random = new Random();

    @Override

    public Integer call() throws Exception {

        return random.nextInt();

    }

}
```

Problema

```
public class Calculator {

    public Integer getResult() {

        // do some calculation

        return 5;

    }

}
```

Problema

```
public class StringTask implements Supplier<String> {

    @Override

    public String get() {

        return "Hola";

    }

}
```

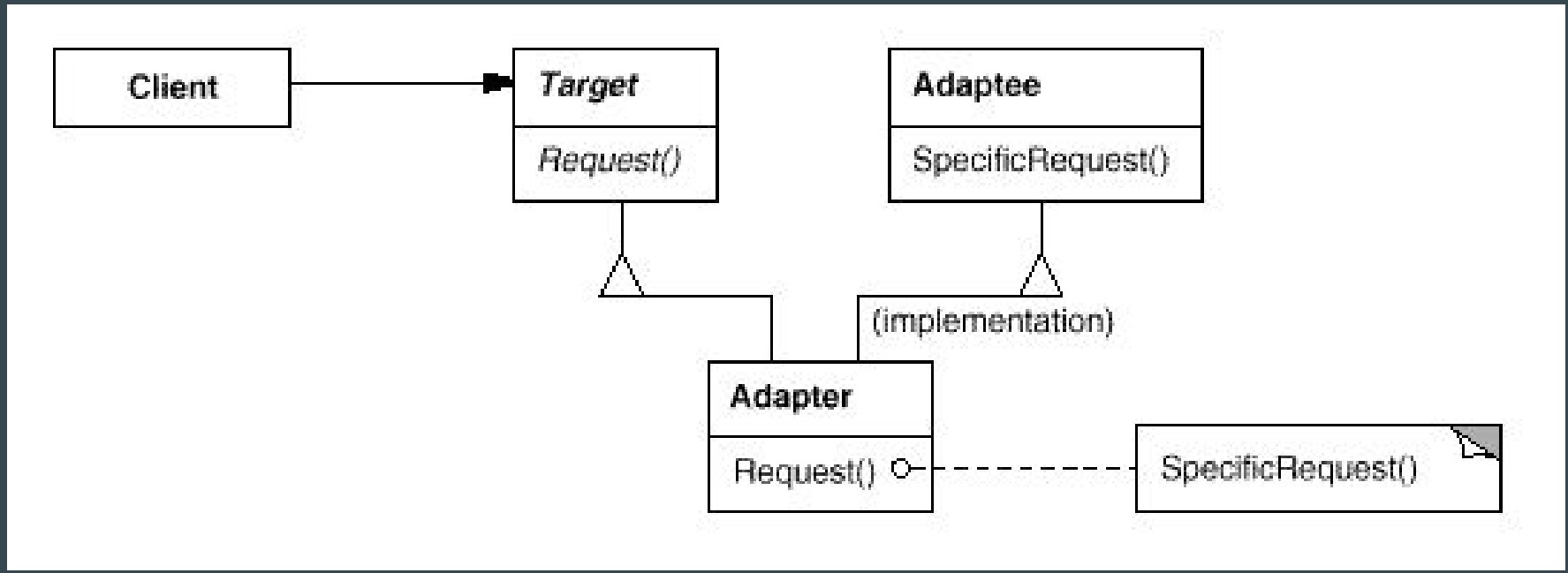
Adapter

- Se quiere utilizar una clase ya existente, pero su interface no coincide con los requerimientos.

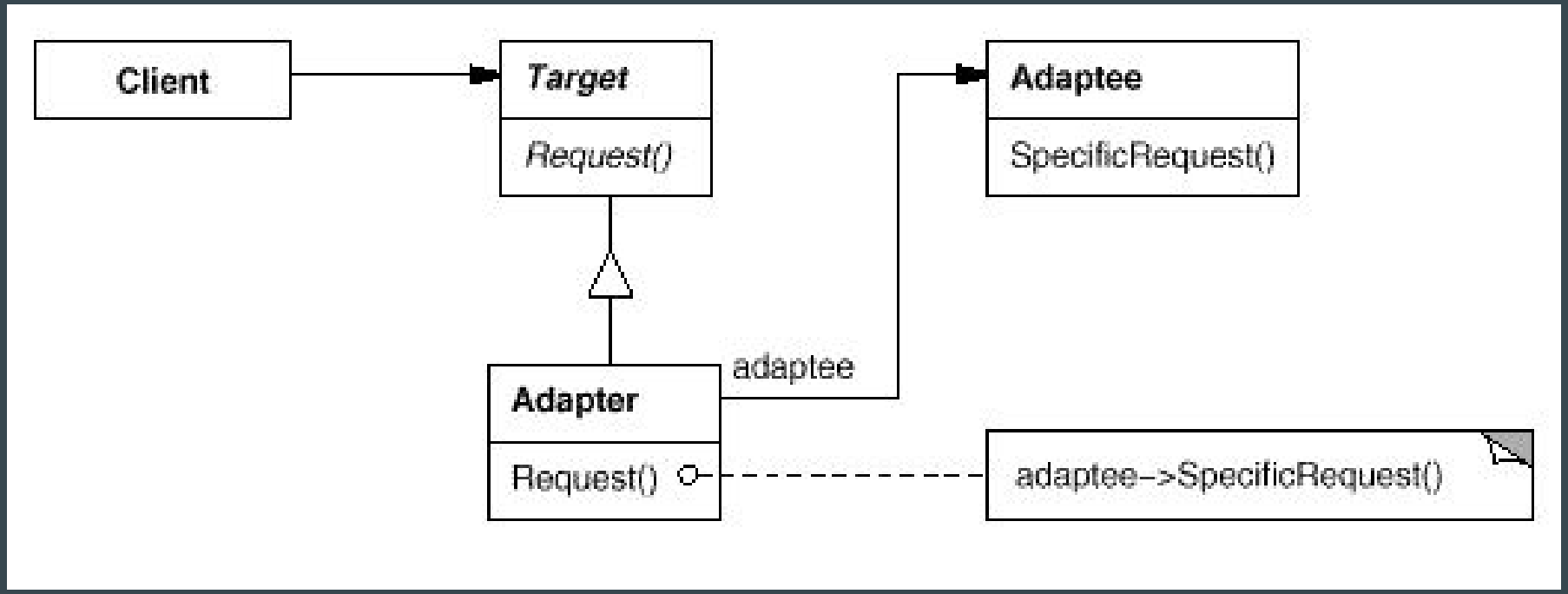
- Se quiere crear clases reusables que cooperen con clases no relacionadas, o con interfaces incompatibles.

- Se necesitan utilizar varias subclases, pero es impráctico adaptarlas a todas (object adapter).

Class Adapter



Object Adapter



Object Adapter

```
public class CalculatorAdapter implements Callable<Integer> {

   private Calculator calculator;

   public CalculatorAdapter(Calculator calculator) { this.calculator = calculator; }

   @Override

   public Integer call() throws Exception {

       return calculator.getResult();

   }

}
```

Class Adapter

```
public class CallableAdapter
    extends StringTask implements Callable<String> {
   @Override
    public String call() throws Exception {
        return this.get();
   }
}
```

Más de una interface

```
public class SuperAdapter<T> implements Supplier<T>, Callable<T> {

   private Factory<T> factory;

   public SuperAdapter(Factory<T> factory) { this.factory = factory; }

   @Override

   public T call() throws Exception { return factory.create(); }

   @Override

   public T get() { return factory.create(); }

}
```

Adapter

- Funcionalidad faltante puede ser implementada en el Adapter.
- Adaptar una aplicación entera.
- Class Adapter vs Object Adapter
- Diferencias entre Adapter, Proxy y Facade.