CODE SPITZ

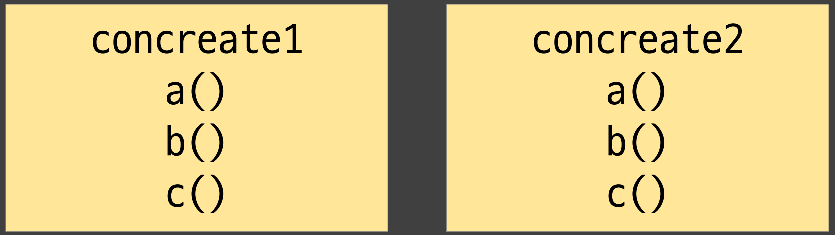
83

OBJECT

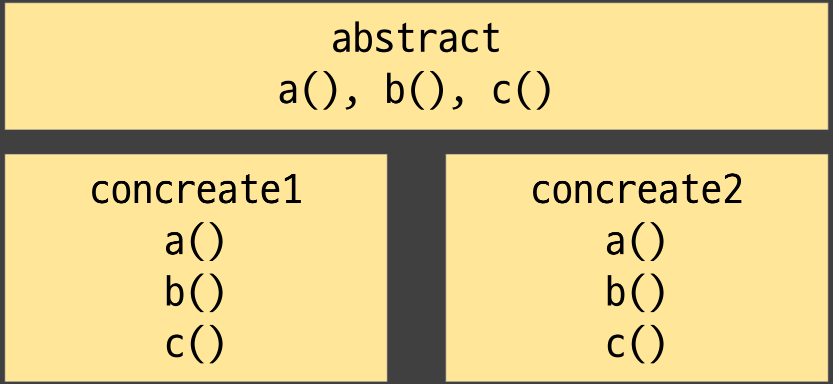
1 2 3 4 5 6

리스코프 치환원칙

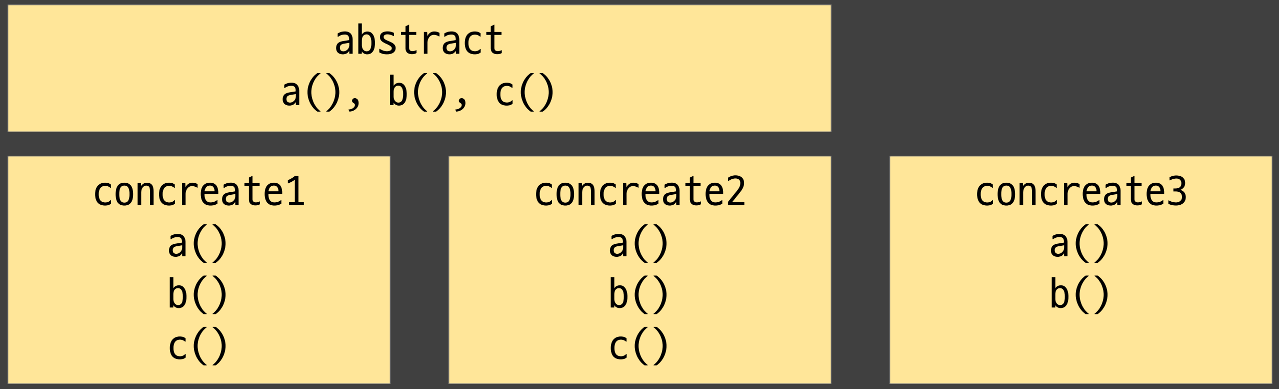
LSP



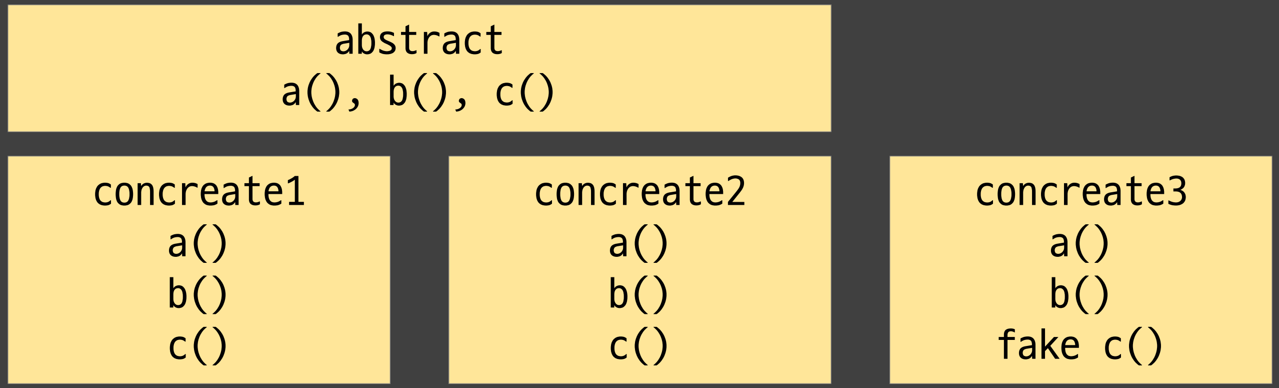
LSP



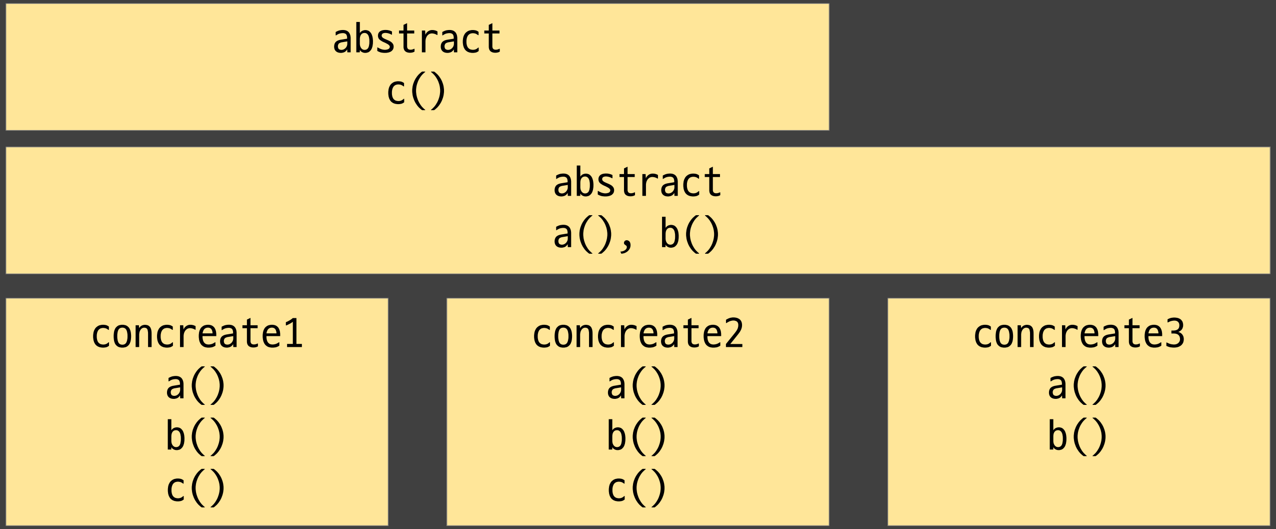
LSP



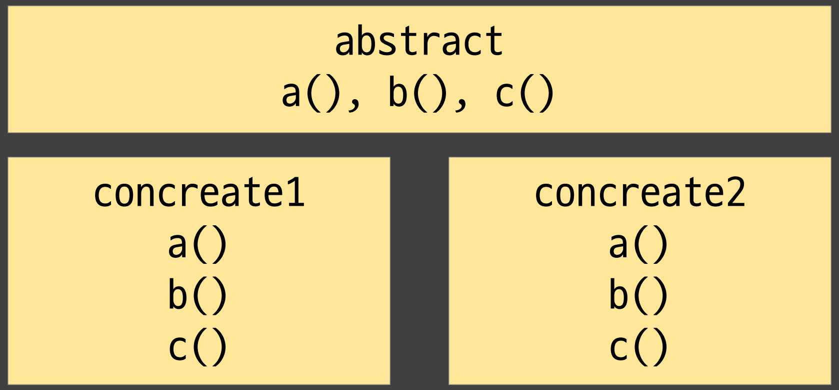
LSP


abstract
a(), b(), c()
concreate1
a()
b()
c()
concreate2
a()
b()
c()
concreate3
a()
b()
fake c()

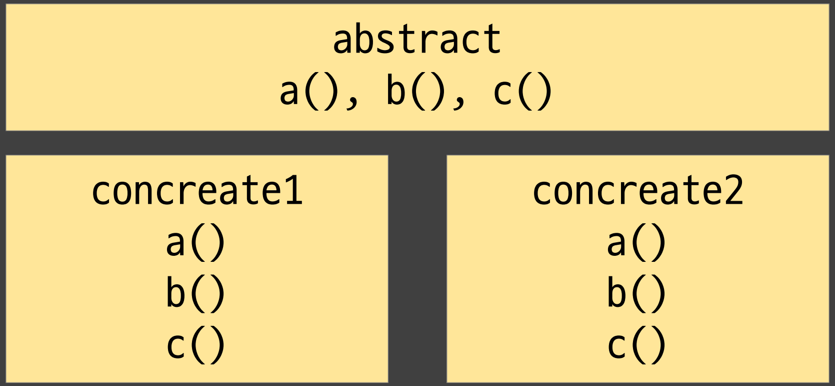
LSP


abstract
c()
abstract
a(), b()
concreate1
a()
b()
c()
concreate2
a()
b()
c()
concreate3
a()
b()

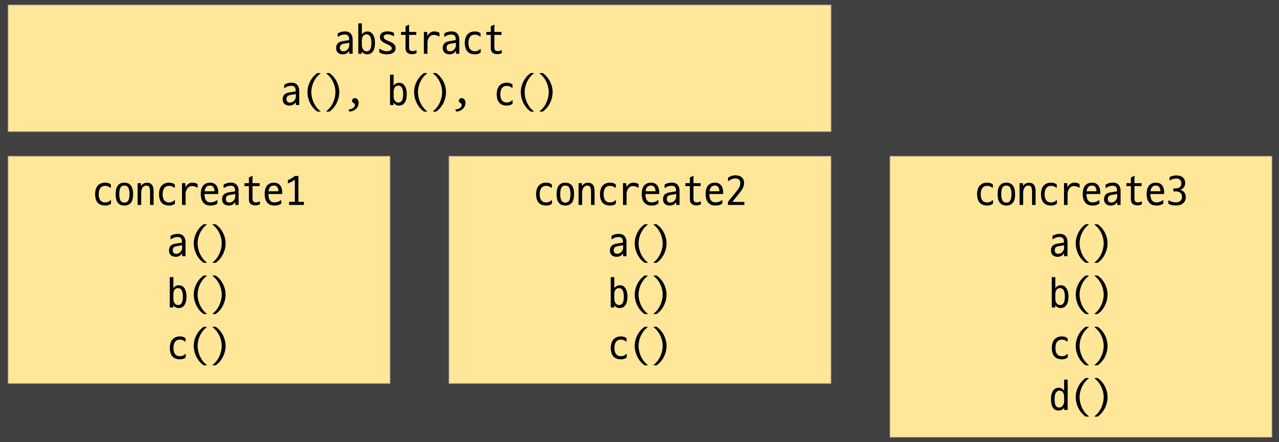
LSP



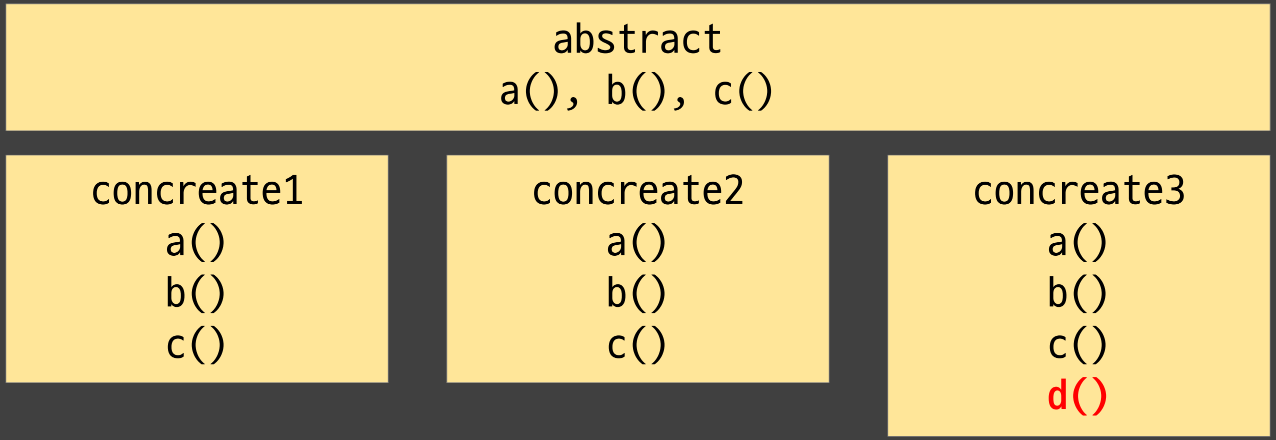
LSP



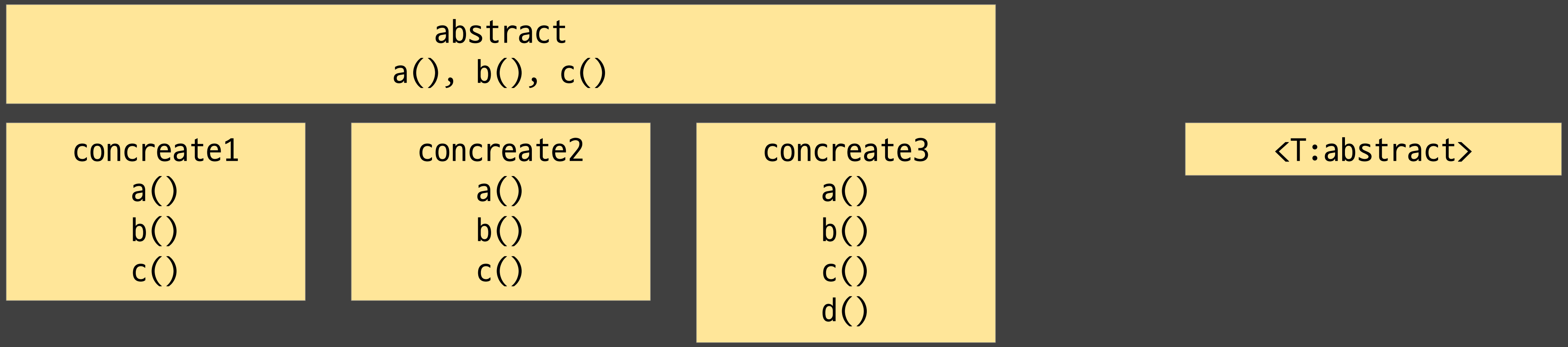
LSP


abstract
a(), b(), c()
concreate1
a()
b()
c()
concreate2
a()
b()
c()
concreate3
a()
b()
c()
d()

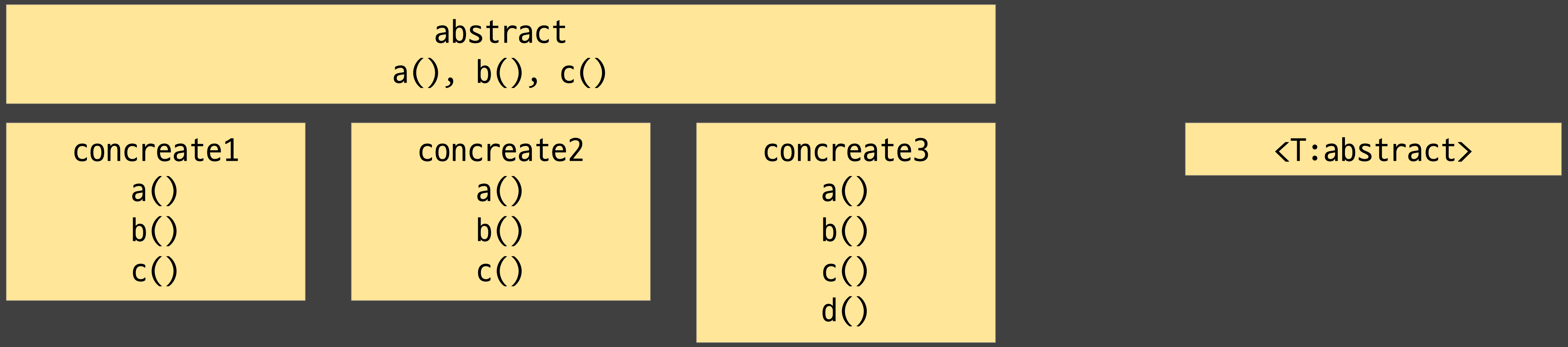
LSP


abstract
a(), b(), c()
concreate1
a()
b()
c()
concreate2
a()
b()
c()
concreate3
a()
b()
c()
d()

LSP

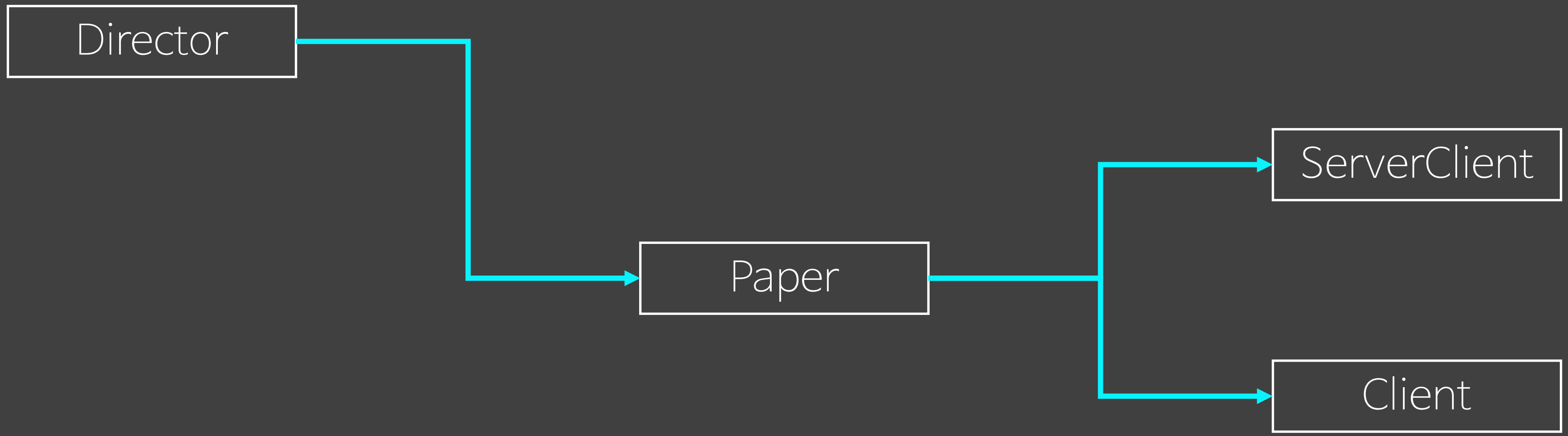

abstract
a(), b(), c()
concreate1
a()
b()
c()
concreate2
a()
b()
c()
concreate3
a()
b()
c()
d()
<T:abstract>

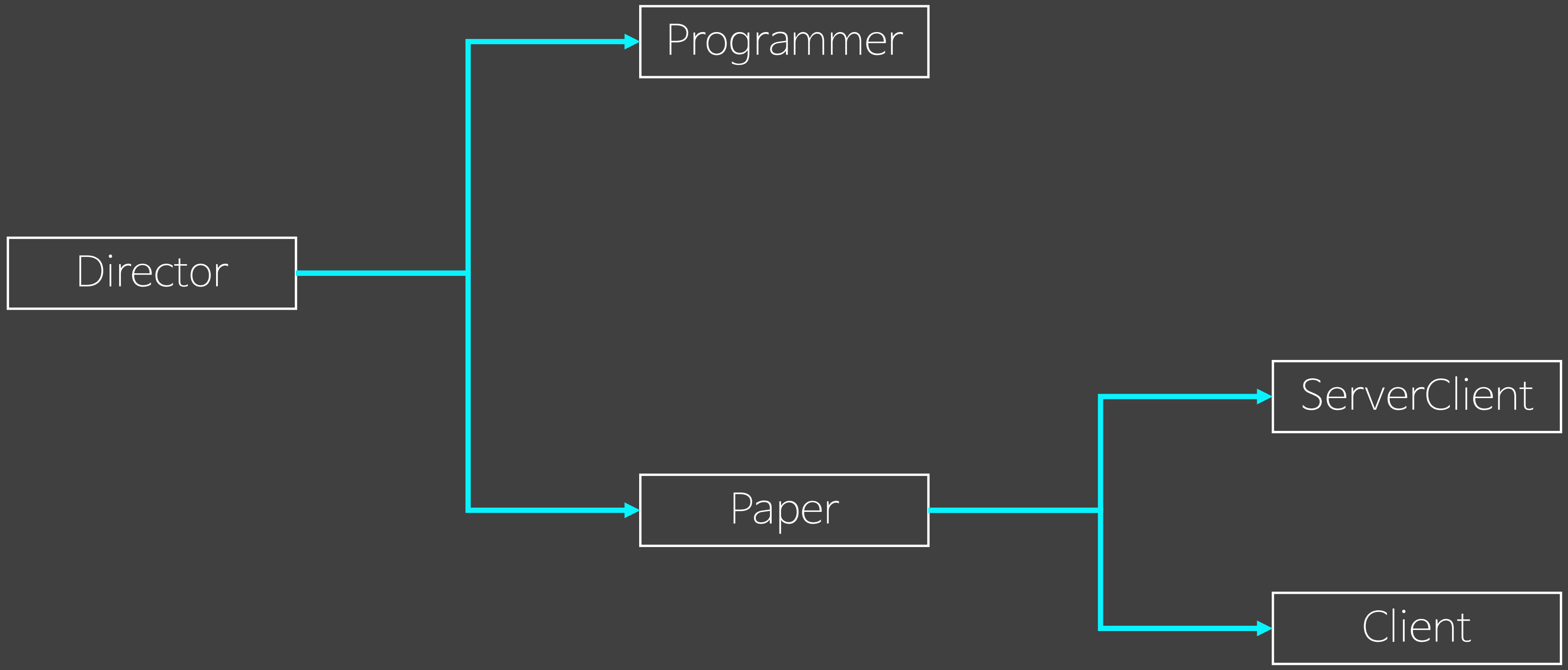
Generic

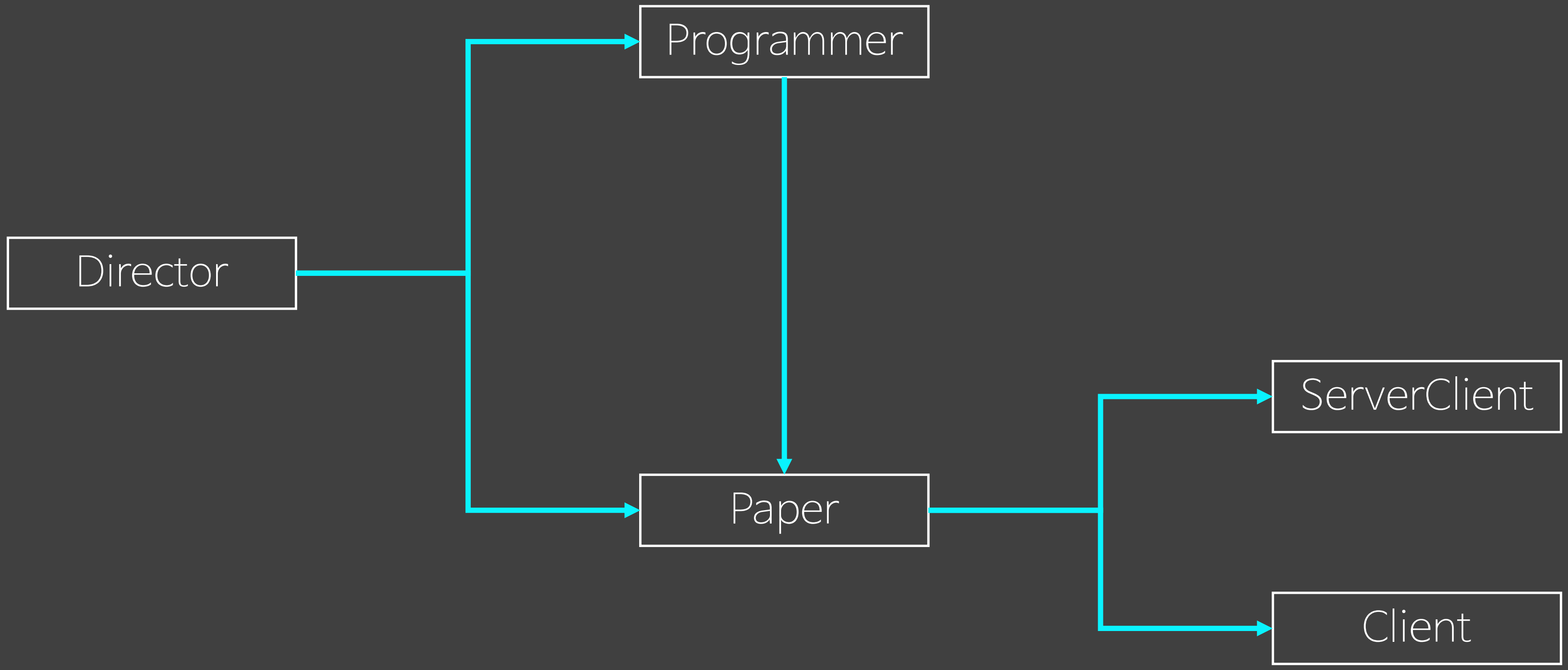

abstract
a(), b(), c()
concreate1
a()
b()
c()
concreate2
a()
b()
c()
concreate3
a()
b()
c()
d()
<T:abstract>

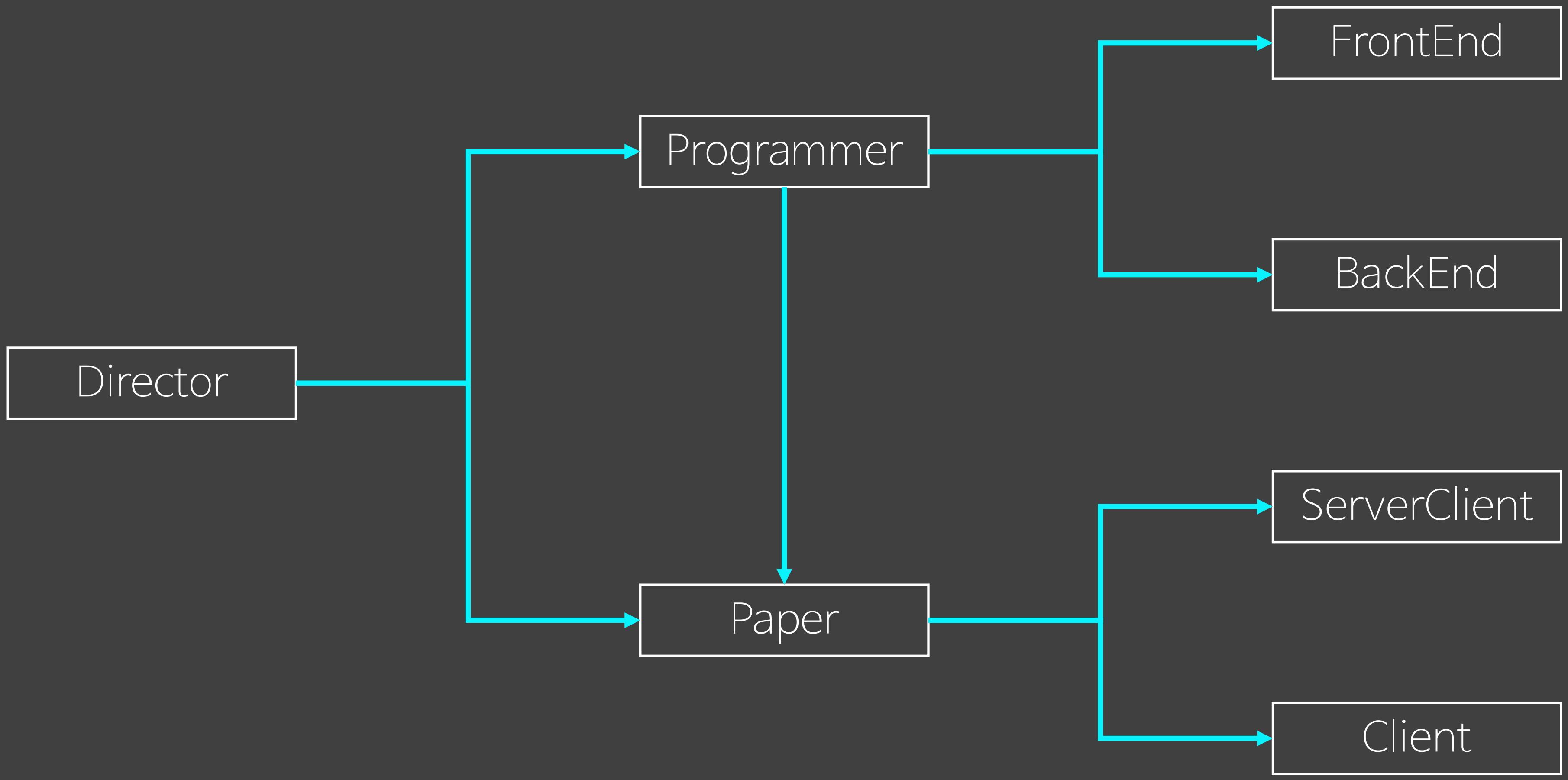
개발자의 세계

Director

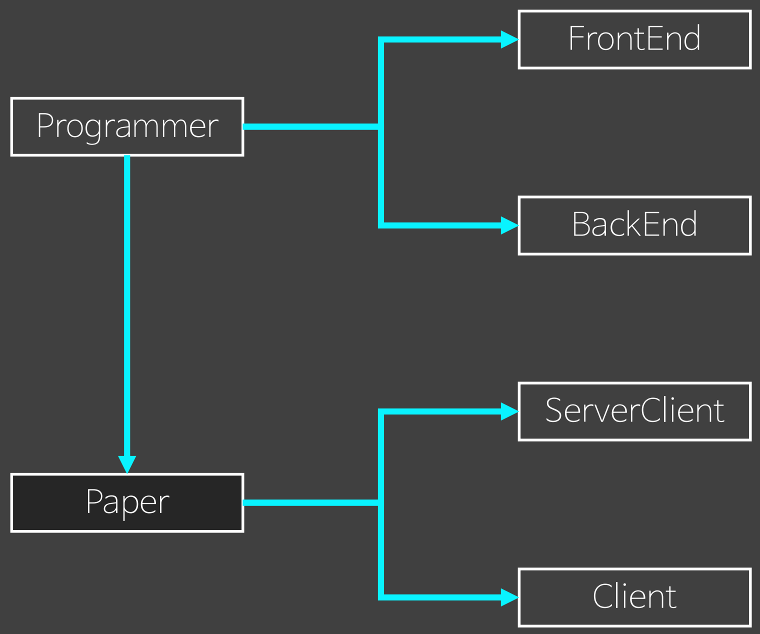

Director
Paper
ServerClient
Client


Programmer
Director
ServerClient
Paper
Client


Director
Programmer
Paper
ServerClient
Client

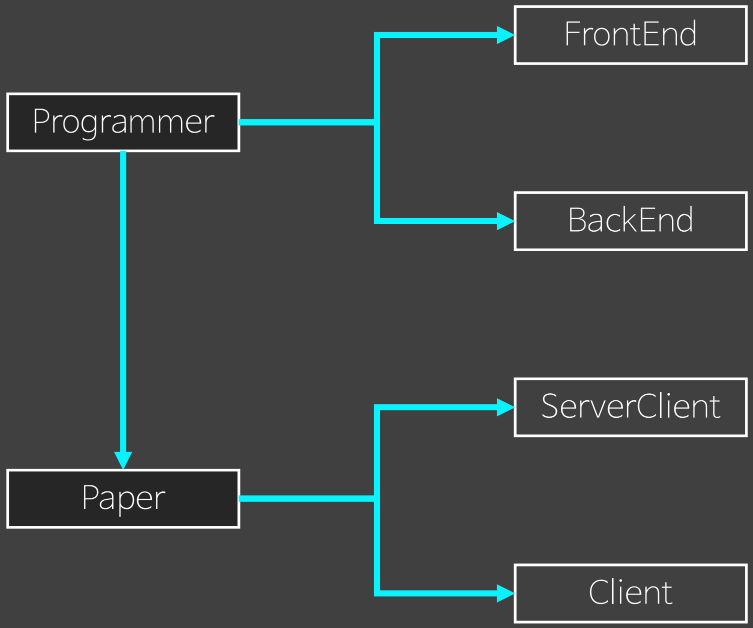

Director
Programmer
Paper
FrontEnd
BackEnd
ServerClient
Client

```java
public interface Paper {}
```

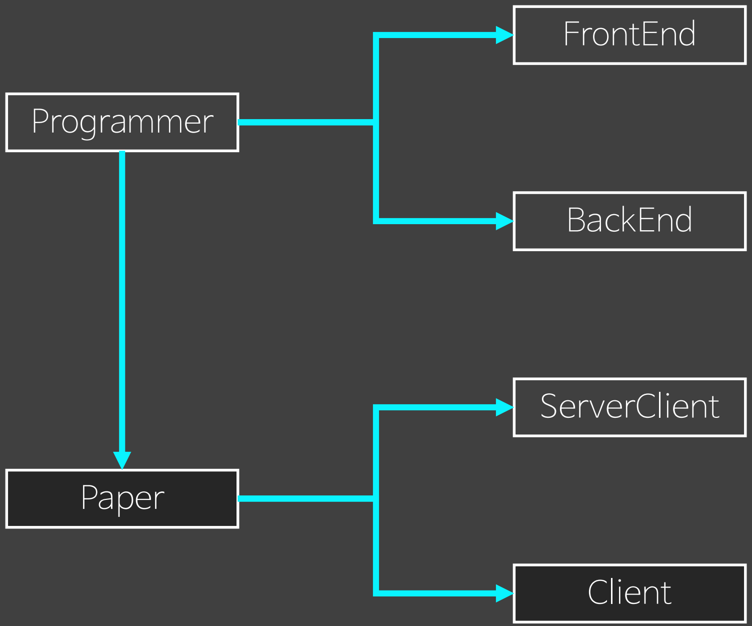


```
public interface Paper {}
```

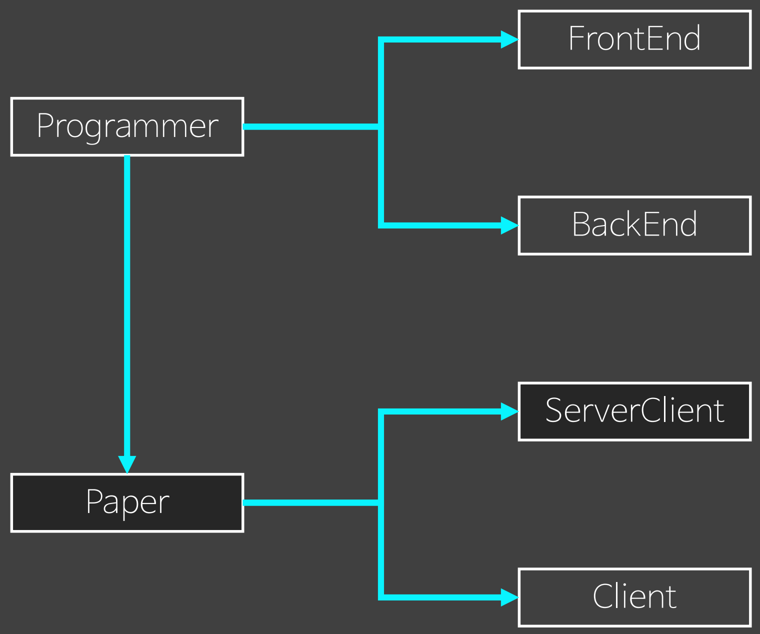
```
public interface Programmer {
    Program makeProgram(Paper paper);
}
```



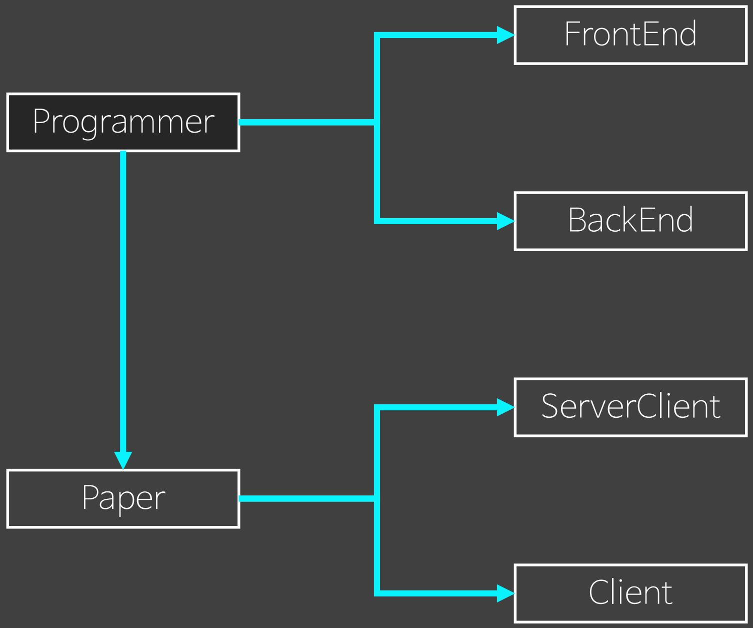
```java
public interface Paper {}

public class Client implements Paper {
    Library library = new Library("vueJS");
    Language language = new Language("kotlinJS");
    Programmer programmer;
    public void setProgrammer(Programmer programmer){
        this.programmer = programmer;
    }
}
```



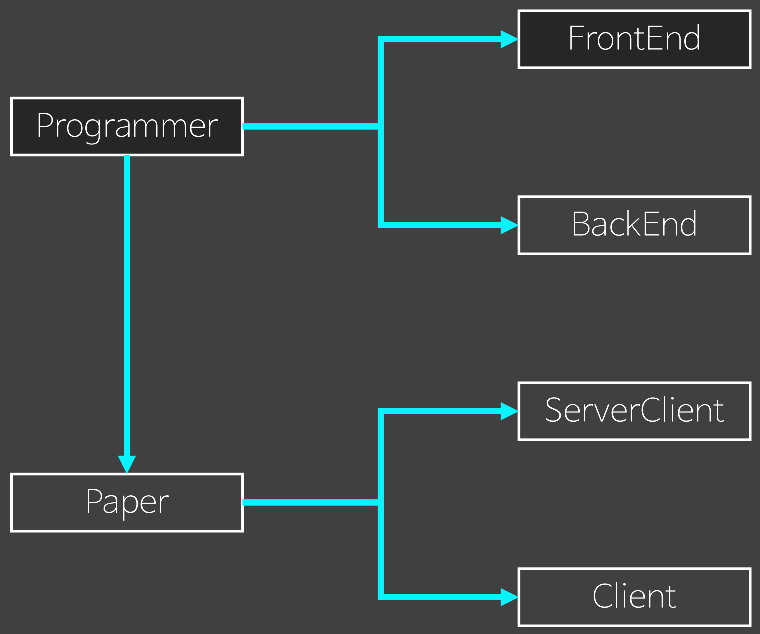
```java
public interface Paper {}

public class ServerClient implements Paper {
    Server server = new Server("test");
    Language backEndLanguage = new Language("java");
    Language frontEndLanguage = new Language("kotlinJS");
    private Programmer backEndProgrammer;
    private Programmer frontEndProgrammer;
    public void setBackEndProgrammer(Programmer programmer){
        backEndProgrammer = programmer;
    }
    public void setFrontEndProgrammer(Programmer programmer){
        frontEndProgrammer = programmer;
    }
}
```


$$\begin{tikzcd}
 & \text{FrontEnd} \\
\text{Programmer} \arrow[ur] \arrow[dr] \arrow[ddd] & \\
 & \text{BackEnd} \\
 & \text{ServerClient} \\
\text{Paper} \arrow[ur] \arrow[dr] & \\
 & \text{Client}
\end{tikzcd}$$

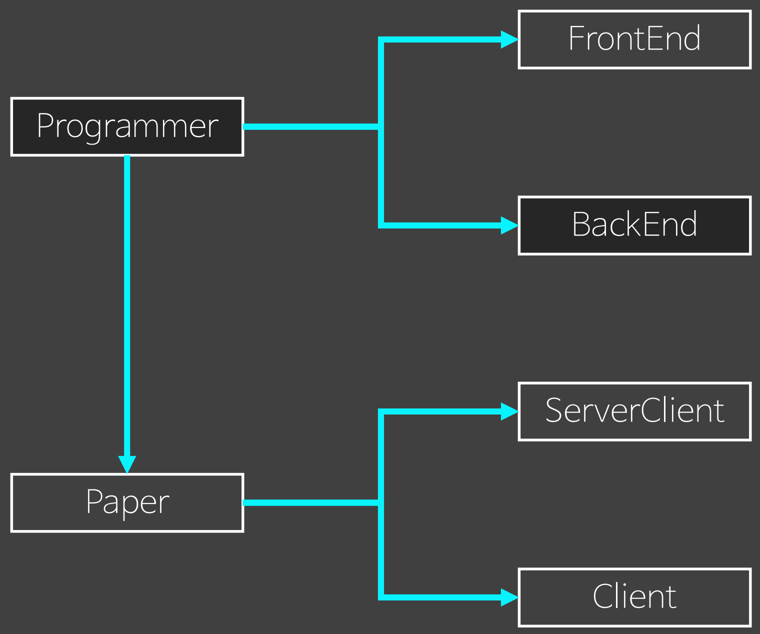
```java
public interface Programmer {
    Program makeProgram(Paper paper);
}
```



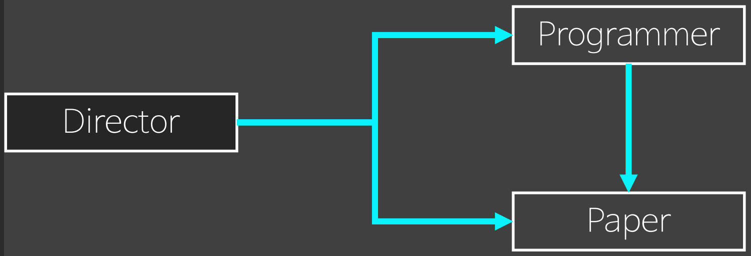
```
public interface Programmer {
    Program makeProgram(Paper paper);
}

public class FrontEnd implements Programmer{
    private Language language;
    private Library library;
    @Override
    public Program makeProgram(Paper paper){
        if(paper instanceof Client){
            Client pb = (Client)paper;
            language = pb.language;
            library = pb.library;
        }
        return makeFrontEndProgram();
    }
    private Program makeFrontEndProgram(){return new Program();}
}
```



```java
public interface Programmer {
    Program makeProgram(Paper paper);
}

public class BackEnd implements Programmer{
    private Server server;
    private Language language;
    @Override
    public Program makeProgram(Paper paper){
        if(paper instanceof ServerClient){
            ServerClient pa = (ServerClient)paper;
            this.server = pa.server;
            this.language = pa.backEndLanguage;
        }
        return makeBackEndProgram();
    }
    private Program makeBackEndProgram(){return new Program();}
}
```

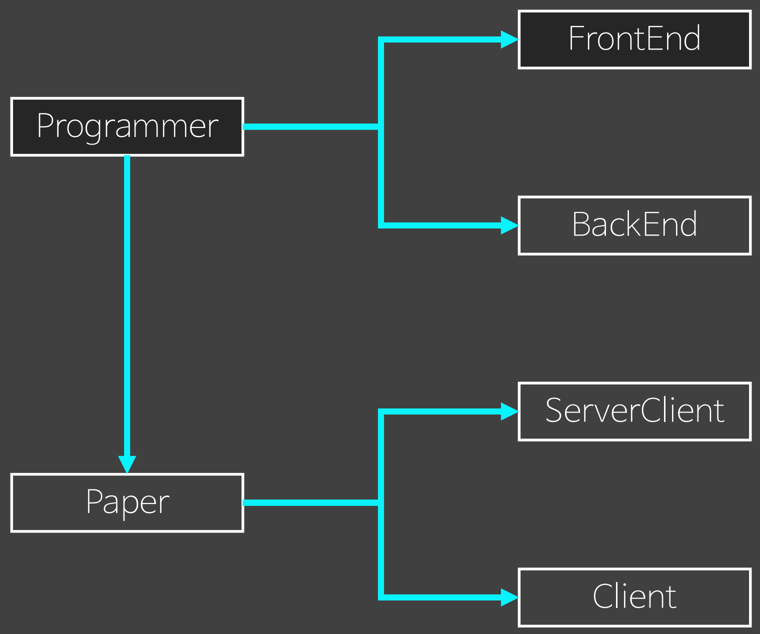


```java
public class Director{
    private Map<String, Paper> projects = new HashMap<>();
    public void addProject(String name, Paper paper){projects.put(name, paper);}
    public void runProject(String name){
        if(!projects.containsKey(name)) throw new RuntimeException("no project");
        Paper paper = projects.get(name);
        if(paper instanceof ServerClient){
            ServerClient project = (ServerClient)paper;
            Programmer frontEnd = new FrontEnd(), backEnd = new BackEnd();
            project.setFrontEndProgrammer(frontEnd);
            project.setBackEndProgrammer(backEnd);
            Program client = frontEnd.makeProgram(project);
            Program server = backEnd.makeProgram(project);
            deploy(name, client, server);
        }else if(paper instanceof Client){
            Client project = (Client)paper;
            Programmer frontEnd = new FrontEnd();
            project.setProgrammer(frontEnd);
            deploy(name, frontEnd.makeProgram(project));
        }
    }
    private void deploy(String projectName, Program...programs){}
}
```



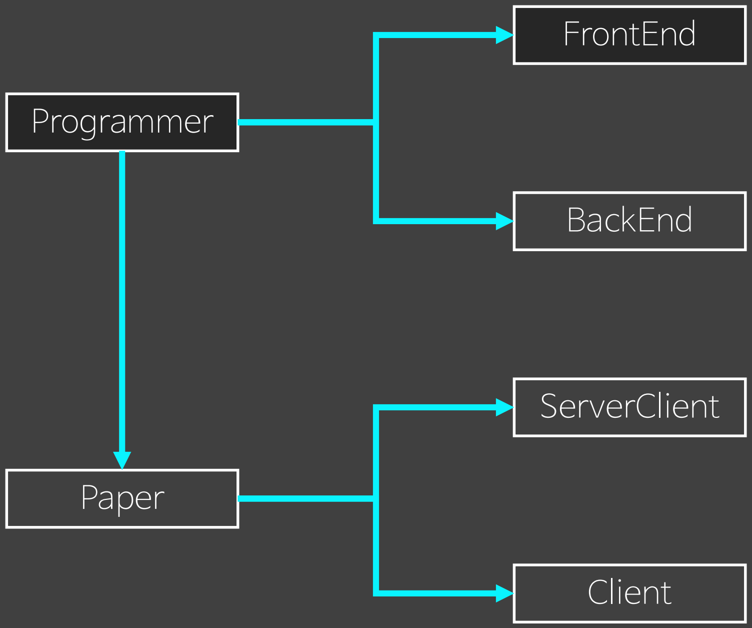
LSP위반 시행착오

```java
public interface Programmer {
    Program makeProgram(Paper paper);
}

public class FrontEnd implements Programmer{
    private Language language;
    private Library library;
    @Override
    public Program makeProgram(Paper paper){
        if(paper instanceof Client){
            Client pb = (Client)paper;
            language = pb.language;
            library = pb.library;
        }
        return makeFrontEndProgram();
    }
    private Program makeFrontEndProgram(){return new Program();}
}
```

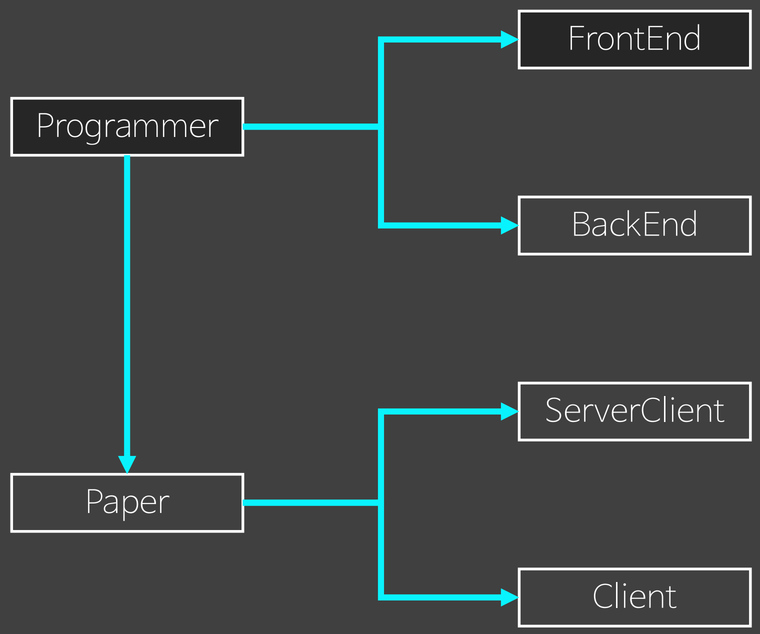


```java
public interface Programmer {
    Program makeProgram(Paper paper);
}

public class FrontEnd implements Programmer{
    private Language language;
    private Library library;
    @Override
    public Program makeProgram(Paper paper){
        if(paper instanceof Client){
            Client pb = (Client)paper;
            language = pb.language;
            library = pb.library;
        }
        return makeFrontEndProgram();
    }
    private Program makeFrontEndProgram(){return new Program();}
}
```

묻지 말고 시켜라
헐리웃 원칙
Tell, Don't Ask

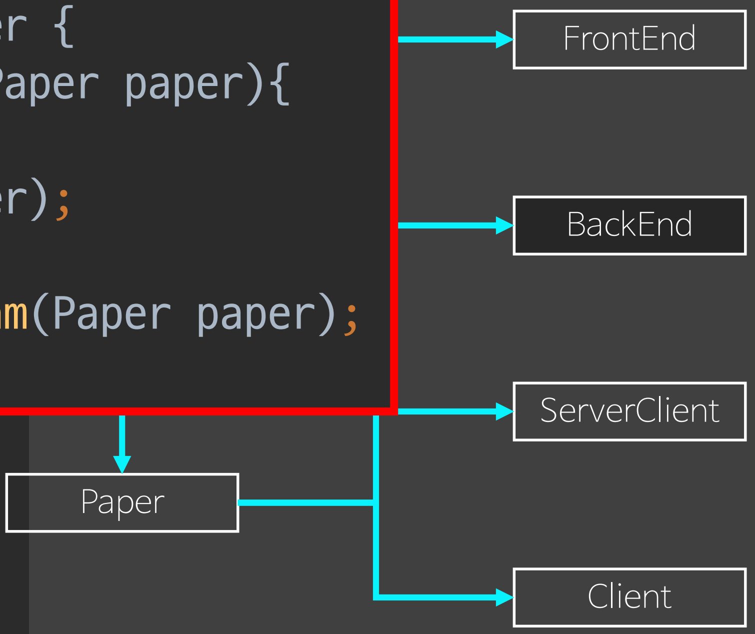


```java
public interface Programmer {
    Program makeProgram(Paper paper);
}

public class FrontEnd implements Programmer {
    private Language language;
    private Library library;
    @Override
    public Program makeProgram(Paper paper){
        paper.setData(this);
        return makeFrontEndProgram();
    }
    void setLanguage(Language language){this.language = language;}
    void setLibrary(Library library){this.library = library;}
    private Program makeFrontEndProgram(){return new Program();}
}
```

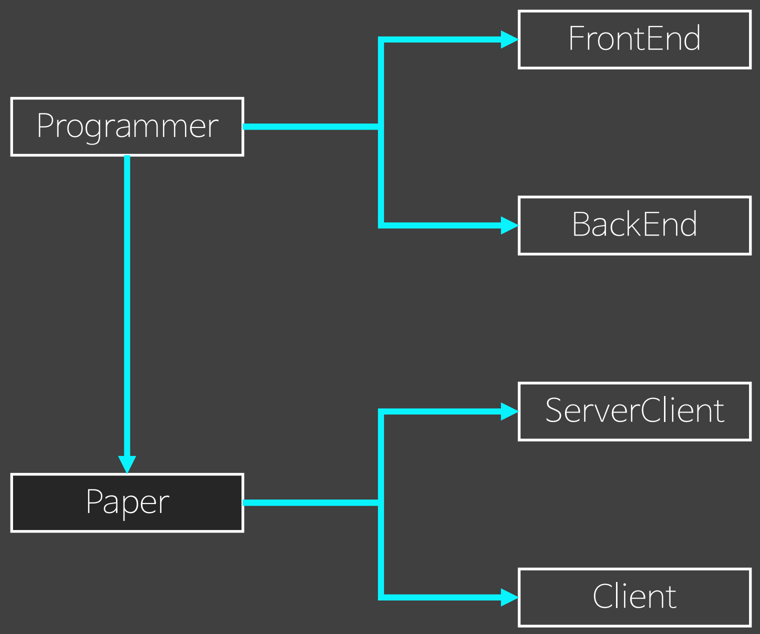

$$\begin{tikzcd}
 & FrontEnd \\
Programmer \arrow[ur] \arrow[dr] \arrow[ddd] & \\
 & BackEnd \\
 & ServerClient \\
Paper \arrow[ur] \arrow[dr] & \\
 & Client
\end{tikzcd}$$

```java
public interface Programmer {
    Program makeProgram(Paper paper);
}

public class BackEnd implements Programmer
    private Server server;
    private Language language;
    @Override
    public Program makeProgram(Paper paper){
        paper.setData(this);
        return makeBackEndProgram();
    }
    public void setServer(Server server){this.server = server;}
    public void setLanguage(Language language){this.language = language;}
    private Program makeBackEndProgram(){
        return new Program();
    }
}
```

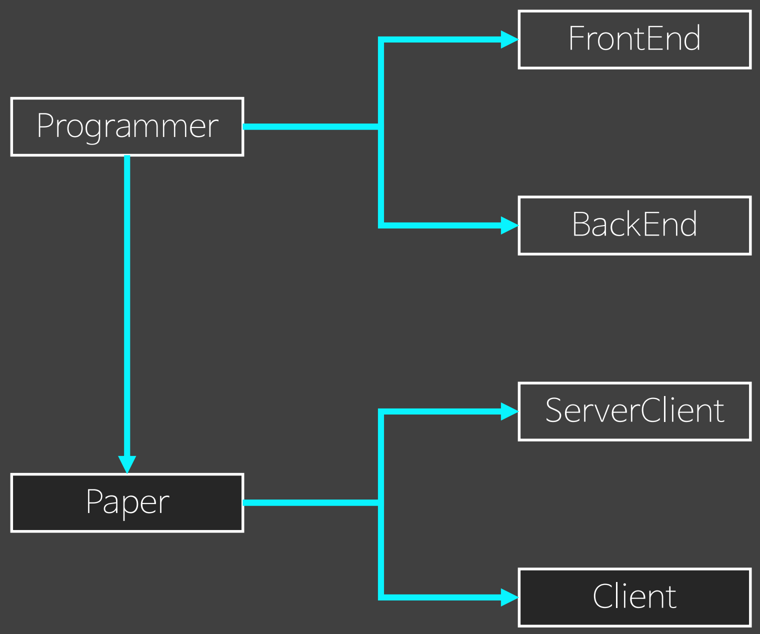
```java
public abstract class Programmer {
    public Program getProgram(Paper paper){
        paper.setData(this);
        return makeProgram(paper);
    }
    abstract Program makeProgram(Paper paper);
}
```



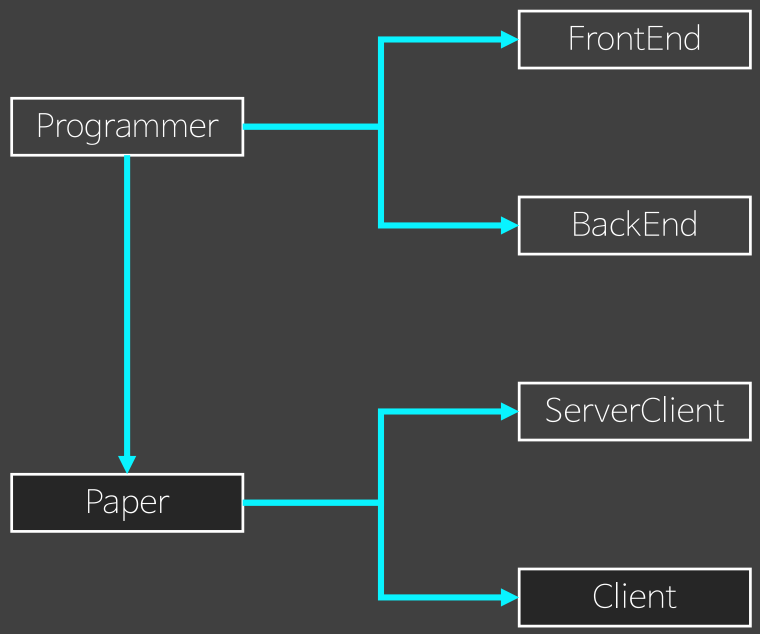
```java
public interface Paper {
    void setData(Programmer programmer);
}
```



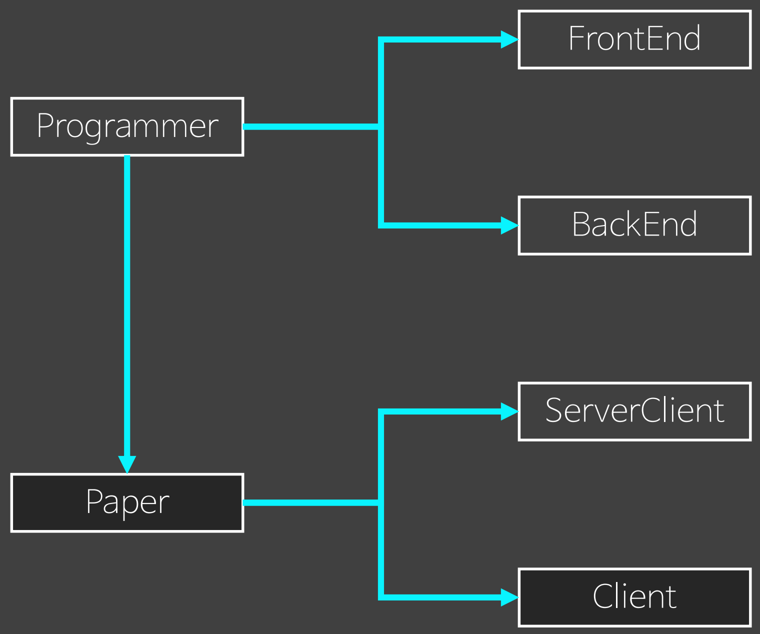
```java
public interface Paper {
    void setData(Programmer programmer);
}

public class Client implements Paper {
    Library library = new Library("vueJS");
    Language language = new Language("kotlinJS");
    Programmer programmer;
    public void setProgrammer(Programmer programmer){
        this.programmer = programmer;
    }
    @Override
    public void setData(Programmer programmer) {
        if(programmer instanceof FrontEnd){
            FrontEnd frontEnd = (FrontEnd)programmer;
            frontEnd.setLibrary(library);
            frontEnd.setLanguage(language);
        }
    }
}
```



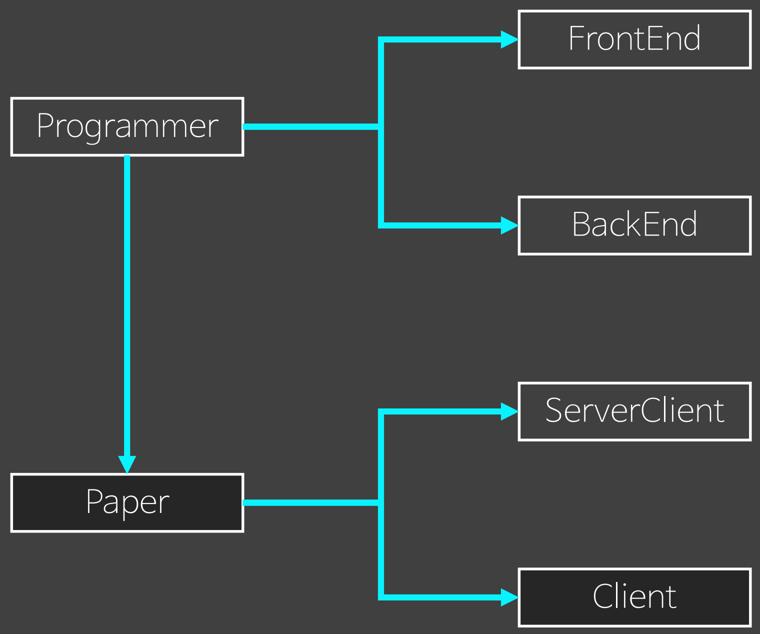
```java
public class ServerClient implements Paper {
    Server server = new Server("test");
    Language backEndLanguage = new Language("java");
    Language frontEndLanguage = new Language("kotlinJS");
    private Programmer backEndProgrammer;
    private Programmer frontEndProgrammer;
    public void setBackEndProgrammer(Programmer programmer){
        backEndProgrammer = programmer;
    }
    public void setFrontEndProgrammer(Programmer programmer){
        frontEndProgrammer = programmer;
    }
    @Override
    public void setData(Programmer programmer) {
        if(programmer instanceof FrontEnd){
            FrontEnd frontEnd = (FrontEnd)programmer;
            frontEnd.setLanguage(frontEndLanguage);
        }else if(programmer instanceof BackEnd){
            BackEnd backEnd = (BackEnd)programmer;
            backEnd.setLanguage(backEndLanguage);
            backEnd.setServer(server);
        }
    }
}
```


$$\begin{tikzcd}
 & \text{FrontEnd} \\
\text{Programmer} \arrow[ur] \arrow[dr] \arrow[dd] & \\
 & \text{BackEnd} \\
 & \text{ServerClient} \\
\text{Paper} \arrow[ur] \arrow[dr] & \\
 & \text{Client}
\end{tikzcd}$$

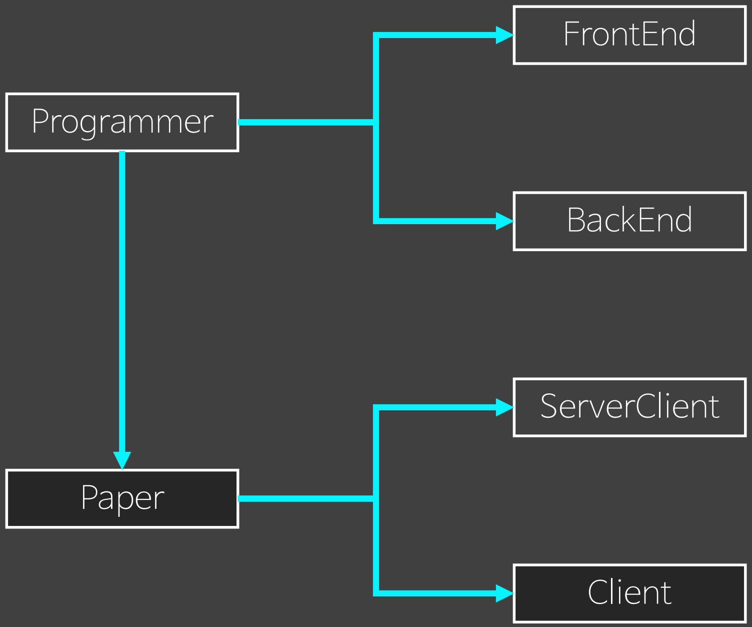
```java
public interface Paper {
    void setData(Programmer programmer);
}

public class Client implements Paper {
    Library library = new Library("vueJS");
    Language language = new Language("kotlinJS");
    Programmer programmer;
    public void setProgrammer(Programmer programmer){
        this.programmer = programmer;
    }
    @Override
    public void setData(Programmer programmer) {
        if(programmer instanceof FrontEnd){
            FrontEnd frontEnd = (FrontEnd)programmer;
            frontEnd.setLibrary(library);
            frontEnd.setLanguage(language);
        }
    }
}
```



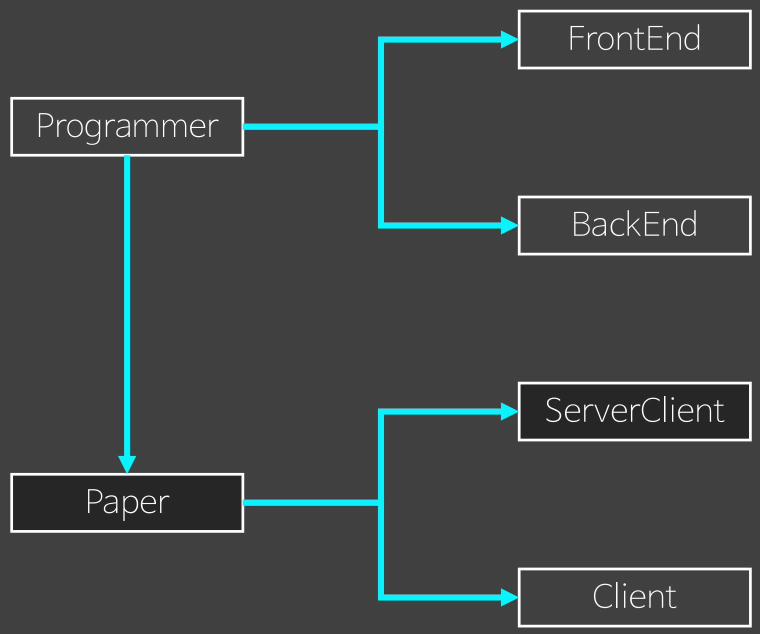
```java
public interface Paper<T extends Programmer> {
    void setData(T programmer);
}

public class Client implements Paper {
    Library library = new Library("vueJS");
    Language language = new Language("kotlinJS");
    Programmer programmer;
    public void setProgrammer(Programmer programmer){
        this.programmer = programmer;
    }
    @Override
    public void setData(Programmer programmer) {
        if(programmer instanceof FrontEnd){
            FrontEnd frontEnd = (FrontEnd)programmer;
            frontEnd.setLibrary(library);
            frontEnd.setLanguage(language);
        }
    }
}
```



```java
public interface Paper<T extends Programmer> {
    void setData(T programmer);
}
public class Client implements Paper<FrontEnd> {
    Library library = new Library("vueJS");
    Language language = new Language("kotlinJS");
    FrontEnd programmer;
    public void setProgrammer(FrontEnd programmer){
        this.programmer = programmer;
    }
    @Override
    public void setData(FrontEnd programmer) {
        programmer.setLibrary(library);
        programmer.setLanguage(language);
    }
}
```

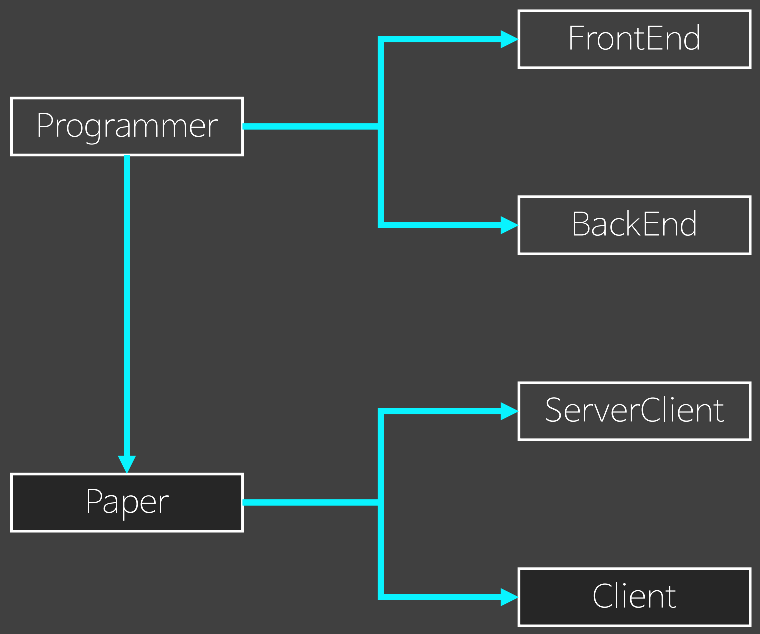


```java
public class ServerClient implements Paper<FrontEnd? or BackEnd?> {
    Server server = new Server("test");
    Language backEndLanguage = new Language("java");
    Language frontEndLanguage = new Language("kotlinJS");
    private Programmer backEndProgrammer;
    private Programmer frontEndProgrammer;
    public void setBackEndProgrammer(Programmer programmer){
        backEndProgrammer = programmer;
    }
    public void setFrontEndProgrammer(Programmer programmer){
        frontEndProgrammer = programmer;
    }
    @Override
    public void setData(Programmer programmer) {
        if(programmer instanceof FrontEnd){
            FrontEnd frontEnd = (FrontEnd)programmer;
            frontEnd.setLanguage(frontEndLanguage);
        }else if(programmer instanceof BackEnd){
            BackEnd backEnd = (BackEnd)programmer;
            backEnd.setLanguage(backEndLanguage);
            backEnd.setServer(server);
        }
    }
}
```

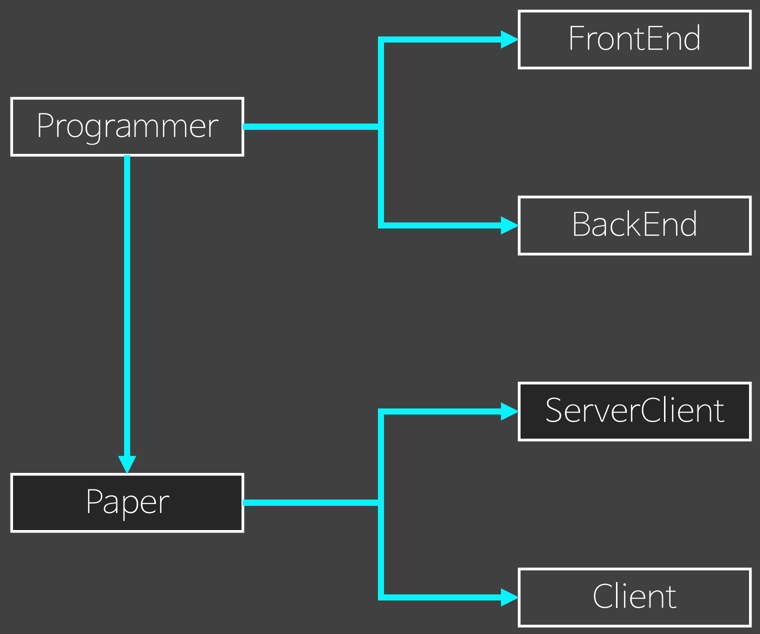


OCP와 제네릭을 통한 해결

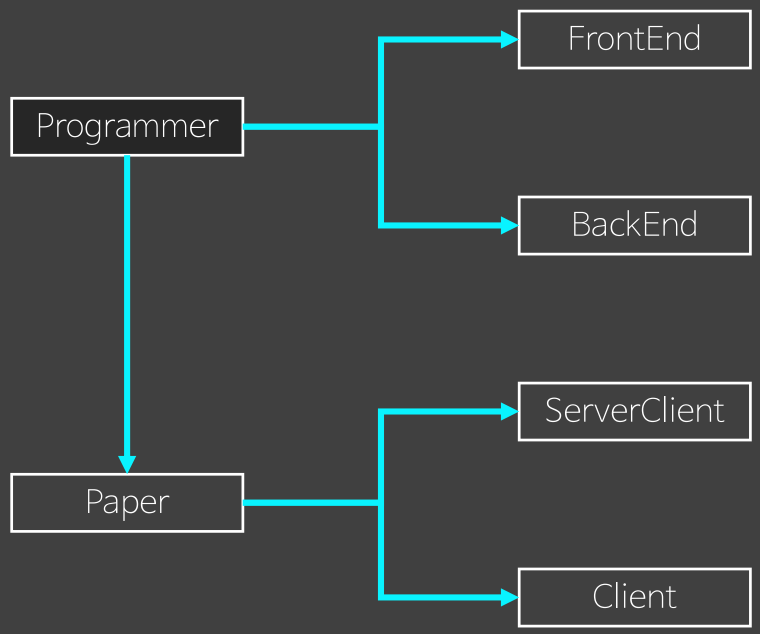
```java
public interface Paper{}

public class Client implements Paper {
    Library library = new Library("vueJS");
    Language language = new Language("kotlinJS");
    Programmer programmer;
    public void setProgrammer(Programmer programmer){
        this.programmer = programmer;
    }
}
```



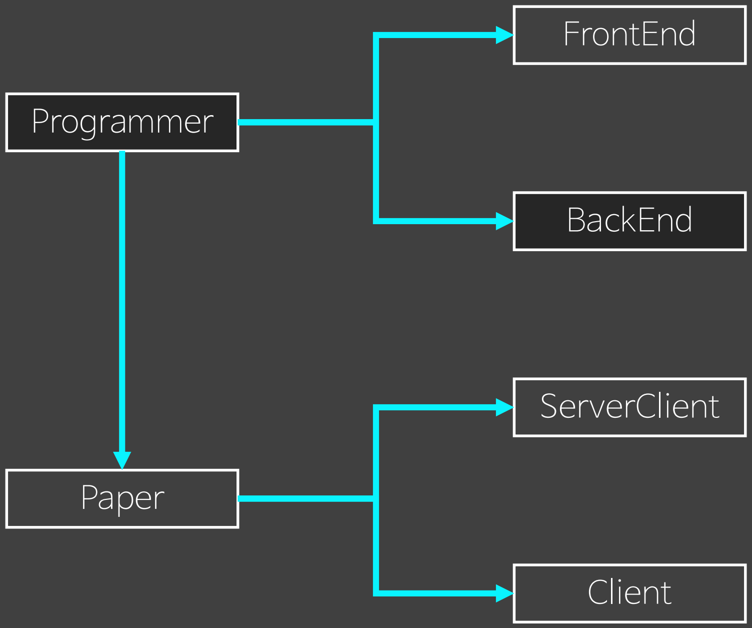
```java
public interface Paper{}

public class ServerClient implements Paper{
    Server server = new Server("test");
    Language backEndLanguage = new Language("java");
    Language frontEndLanguage = new Language("kotlinJS");
    private Programmer backEndProgrammer;
    private Programmer frontEndProgrammer;
    public void setBackEndProgrammer(Programmer programmer){
        backEndProgrammer = programmer;
    }
    public void setFrontEndProgrammer(Programmer programmer){
        frontEndProgrammer = programmer;
    }
}
```


$$\begin{tikzcd}
 & \text{FrontEnd} \\
\text{Programmer} \arrow[ur] \arrow[dr] \arrow[ddd] & \\
 & \text{BackEnd} \\
 & \text{ServerClient} \\
\text{Paper} \arrow[ur] \arrow[dr] & \\
 & \text{Client}
\end{tikzcd}$$

```java
public abstract class Programmer<T extends Paper>{
    public Program getProgram(T paper){
        setData(paper);
        return makeProgram();
    }
    abstract void setData(T paper);
    abstract Program makeProgram();
}
```

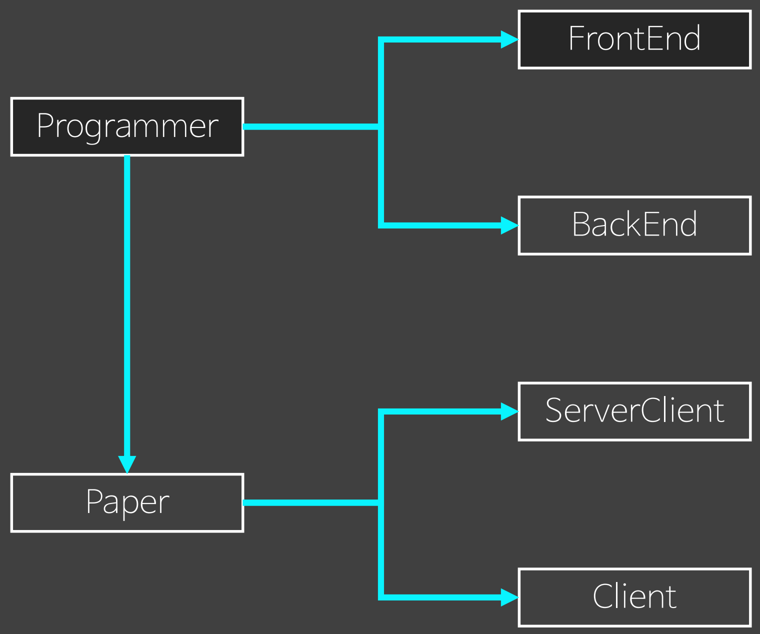


```
public abstract class Programmer<T extends Paper>{
    public Program getProgram(T paper){
        setData(paper);
        return makeProgram();
    }
    abstract void setData(T paper);
    abstract Program makeProgram();
}
public abstract class BackEnd<T extends Paper> extends Programmer<T> {
    protected Server server;
    protected Language language;
    @Override
    protected Program makeProgram(){
        return new Program();
    }
}
```



```
public abstract class Programmer<T extends Paper>{
    public Program getProgram(T paper){
        setData(paper);
        return makeProgram();
    }
    abstract void setData(T paper);
    abstract Program makeProgram();
}
```

```
public abstract class FrontEnd<T extends Paper> extends Programmer<T>{
    protected Language language;
    protected Library library;
    @Override
    protected Program makeProgram(){
        return new Program();
    }
}
```



클라이언트의 변화

```java
public class Director{
    private Map<String, Paper> projects = new HashMap<>();
    public void addProject(String name, Paper paper){projects.put(name, paper);}
    public void runProject(String name){
        if(!projects.containsKey(name)) throw new RuntimeException("no project");
        Paper paper = projects.get(name);
        if(paper instanceof ServerClient){
            ServerClient project = (ServerClient)paper;
            Programmer frontEnd = new FrontEnd(), backEnd = new BackEnd();
            project.setFrontEndProgrammer(frontEnd);
            project.setBackEndProgrammer(backEnd);
            Program client = frontEnd.makeProgram(project);
            Program server = backEnd.makeProgram(project);
            deploy(name, client, server);
        }else if(paper instanceof Client){
            Client project = (Client)paper;
            Programmer frontEnd = new FrontEnd();
            project.setProgrammer(frontEnd);
            deploy(name, frontEnd.makeProgram(project));
        }
    }
    private void deploy(String projectName, Program...programs){}
}
```

```java
public abstract class Programmer<T extends Paper>{
    public Program getProgram(T paper){
        setData(paper);
        return makeProgram();
    }
    abstract void setData(T paper);
    abstract protected Program makeProgram();
}
```

```java
if(paper instanceof ServerClient){
    ServerClient project = (ServerClient)paper;
    Programmer frontEnd = new FrontEnd<ServerClient>(){
        @Override
        void setData(ServerClient paper) {
            language = paper.frontEndLanguage;
        }
    };
    Programmer backEnd = new BackEnd<ServerClient>(){
        @Override
        void setData(ServerClient paper) {
            server = paper.server;
            language = paper.backEndLanguage;
        }
    };
    project.setFrontEndProgrammer(frontEnd);
    project.setBackEndProgrammer(backEnd);
    Program client = frontEnd.getProgram(project);
    Program server = backEnd.getProgram(project);
    deploy(name, client, server);
}
```

```java
public abstract class Programmer<T extends Paper>{
    public Program getProgram(T paper){
        setData(paper);
        return makeProgram();
    }
    abstract void setData(T paper);
    abstract protected Program makeProgram();
}
```

```java
if(paper instanceof ServerClient){
    ServerClient project = (ServerClient)paper;
    Programmer frontEnd = new FrontEnd<ServerClient>(){
        @Override
        void setData(ServerClient paper) {
            language = paper.frontEndLanguage;
        }
    };
    Programmer backEnd = new BackEnd<ServerClient>(){
        @Override
        void setData(ServerClient paper) {
            server = paper.server;
            language = paper.backEndLanguage;
        }
    };
    project.setFrontEndProgrammer(frontEnd);
    project.setBackEndProgrammer(backEnd);
    Program client = frontEnd.getProgram(project);
    Program server = backEnd.getProgram(project);
    deploy(name, client, server);
}
```

```java
public abstract class Programmer<T extends Paper>{
    public Program getProgram(T paper){
        setData(paper);
        return makeProgram();
    }
    abstract void setData(T paper);
    abstract protected Program makeProgram();
}
```

```java
}else if(paper instanceof Client){
    Client project = (Client)paper;
    FrontEnd frontEnd = new FrontEnd<Client>(){
        @Override
        void setData(Client paper) {
            library = paper.library;
            language = paper.language;
        }
    };
    project.setProgrammer(frontEnd);
    deploy(name, frontEnd.getProgram(project));
}
```

```java
public abstract class Programmer<T extends Paper>{
    public Program getProgram(T paper){
        setData(paper);
        return makeProgram();
    }
    abstract void setData(T paper);
    abstract protected Program makeProgram();
}
```

```java
}else if(paper instanceof Client){
    Client project = (Client)paper;
    FrontEnd frontEnd = new FrontEnd<Client>(){
        @Override
        void setData(Client paper) {
            library = paper.library;
            language = paper.language;
        }
    };
    project.setProgrammer(frontEnd);
    deploy(name, frontEnd.getProgram(project));
}
```

```java
public abstract class Programmer<T extends Paper>{
    public Program getProgram(T paper){
        setData(paper);
        return makeProgram();
    }
    abstract void setData(T paper);
    abstract protected Program makeProgram();
}
```

```java
public class Director{
    private Map<String, Paper> projects = new HashMap<>();
    public void addProject(String name, Paper paper){projects.put(name, paper);}
    public void runProject(String name){
        if(!projects.containsKey(name)) throw new RuntimeException("no project");
        Paper paper = projects.get(name);
        if(paper instanceof ServerClient){
            ServerClient project = (ServerClient)paper;
            Programmer frontEnd = new FrontEnd<ServerClient>(){
                @Override
                void setData(ServerClient paper) {...}
            };
            Programmer backEnd = new BackEnd<ServerClient>(){
                @Override
                void setData(ServerClient paper) {...}
            };
            project....
            deploy(name, frontEnd.getProgram(project), backEnd.getProgram(project));
        }else if(paper instanceof Client){
            Client project = (Client)paper;
            FrontEnd frontEnd = new FrontEnd<Client>(){
                @Override
                void setData(Client paper) {...}
            };
            project.setProgrammer(frontEnd);
            deploy(name, frontEnd.getProgram(project));
        }
    }
    private void deploy(String projectName, Program...programs){}
}
```

```java
public class Director{
    private Map<String, Paper> projects = new HashMap<>();
    public void addProject(String name, Paper paper){projects.put(name, paper);}
    public void runProject(String name){
        if(!projects.containsKey(name)) throw new RuntimeException("no project");
        Paper paper = projects.get(name);
        if(paper instanceof ServerClient){
            ServerClient project = (ServerClient)paper;
            Programmer frontEnd = new FrontEnd<ServerClient>(){
                @Override
                void setData(ServerClient paper) {...}
            };
            Programmer backEnd = new BackEnd<ServerClient>(){
                @Override
                void setData(ServerClient paper) {...}
            };
            project....
            deploy(name, frontEnd.getProgram(project), backEnd.getProgram(project));
        }else if(paper instanceof Client){
            Client project = (Client)paper;
            FrontEnd frontEnd = new FrontEnd<Client>(){
                @Override
                void setData(Client paper) {...}
            };
            project.setProgrammer(frontEnd);
            deploy(name, frontEnd.getProgram(project));
        }
    }
    private void deploy(String projectName, Program...programs){}
}
```

OCP위반

```java
public class Director{
    private Map<String, Paper> projects = new HashMap<>();
    public void addProject(String name, Paper paper){projects.put(name, paper);}
    public void runProject(String name){
        if(!projects.containsKey(name)) throw new RuntimeException("no project");
        Paper paper = projects.get(name);
        if(paper instanceof ServerClient){
            ServerClient project = (ServerClient)paper;
            Programmer frontEnd = new FrontEnd<ServerClient>(){
                @Override
                void setData(ServerClient paper) {...}
            };
            Programmer backEnd = new BackEnd<ServerClient>(){
                @Override
                void setData(ServerClient paper) {...}
            };
            project....
            deploy(name, frontEnd.getProgram(project), backEnd.getProgram(project));
        }else if(paper instanceof Client){
            Client project = (Client)paper;
            FrontEnd frontEnd = new FrontEnd<Client>(){
                @Override
                void setData(Client paper) {...}
            };
            project.setProgrammer(frontEnd);
            deploy(name, frontEnd.getProgram(project));
        }
    }
    private void deploy(String projectName, Program...programs){}
}
```

OCP위반 $\longrightarrow$

1. 클라이언트로
2. 경우의 수만큼 생성

```java
public class Director{
    private Map<String, Paper> projects = new HashMap<>();
    public void addProject(String name, Paper paper){projects.put(name, paper);}
    public void runProject(String name){
        if(!projects.containsKey(name)) throw new RuntimeException("no project");
        Paper paper = projects.get(name);
        if(paper instanceof ServerClient){
            ServerClient project = (ServerClient)paper;
            Programmer frontEnd = new FrontEnd<ServerClient>(){
                @Override
                void setData(ServerClient paper) {...}
            };
            Programmer backEnd = new BackEnd<ServerClient>(){
                @Override
                void setData(ServerClient paper) {...}
            };
            project....
            deploy(name, frontEnd.getProgram(project), backEnd.getProgram(project));
        }else if(paper instanceof Client){
            Client project = (Client)paper;
            FrontEnd frontEnd = new FrontEnd<Client>(){
                @Override
                void setData(Client paper) {...}
            };
            project.setProgrammer(frontEnd);
            deploy(name, frontEnd.getProgram(project));
        }
    }
    private void deploy(String projectName, Program...programs){}
}
```

OCP위반 $\longrightarrow$

1. 클라이언트로
2. 경우의 수만큼 생성

```java
public class Director{
    private Map<String, Paper> projects = new HashMap<>();
    public void addProject(String name, Paper paper){projects.put(name, paper);}
    public void runProject(String name){
        if(!projects.containsKey(name)) throw new RuntimeException("no project");
        Paper paper = projects.get(name);
        if(paper instanceof ServerClient){
            ServerClient project = (ServerClient)paper;
            Programmer frontEnd = new FrontEnd<ServerClient>(){
                @Override
                void setData(ServerClient paper) {...}
            };
            Programmer backEnd = new BackEnd<ServerClient>(){
                @Override
                void setData(ServerClient paper) {...}
            };
            project....
            deploy(name, frontEnd.getProgram(project), backEnd.getProgram(project));
        }else if(paper instanceof Client){
            Client project = (Client)paper;
            FrontEnd frontEnd = new FrontEnd<Client>(){
                @Override
                void setData(Client paper) {...}
            };
            project.setProgrammer(frontEnd);
            deploy(name, frontEnd.getProgram(project));
        }
    }
    private void deploy(String projectName, Program...programs){}
}
```

OCP위반 →

1. 클라이언트로
2. 경우의 수만큼 생성

↑

추상화를 통해

```java
public interface Paper{
    Program[] run();
}
```

```java
public interface Paper{
    Program[] run();
}
```

```java
public abstract class ServerClient implements Paper{
    Server server = new Server("test");
    Language backEndLanguage = new Language("java");
    Language frontEndLanguage = new Language("kotlinJS");
    private Programmer backEndProgrammer;
    private Programmer frontEndProgrammer;
    public void setBackEndProgrammer(Programmer programmer){
        backEndProgrammer = programmer;
    }
    public void setFrontEndProgrammer(Programmer programmer){
        frontEndProgrammer = programmer;
    }
}
```

```java
public abstract class Client implements Paper{
    Library library = new Library("vueJS");
    Language language = new Language("kotlinJS");
    FrontEnd programmer;
    public void setProgrammer(FrontEnd programmer){
        this.programmer = programmer;
    }
}
```

```java
public class Director{
    private Map<String, Paper> projects = new HashMap<>();
    public void addProject(String name, Paper paper){projects.put(name, paper);}
    public void runProject(String name){
        if(!projects.containsKey(name)) throw new RuntimeException("no project");
        deploy(name, projects.get(name).run());
    }
    private void deploy(String projectName, Program...programs){}
}
```

```java
public class Director{
    private Map<String, Paper> projects = new HashMap<>();
    public void addProject(String name, Paper paper){projects.put(name, paper);}
    public void runProject(String name){
        if(!projects.containsKey(name)) throw new RuntimeException("no project");
        deploy(name, projects.get(name).run());
    }
    private void deploy(String projectName, Program...programs){}
}
```

```java
public class Main {
    public static void main(String[] args){
        Director director = new Director();
        director.addProject("여행사A 프론트개편", new Client() {
            @Override
            public Program[] run() {
                FrontEnd frontEnd = new FrontEnd<Client>(){
                    @Override
                    void setData(Client paper) {
                        library = paper.library;
                        language = paper.language;
                    }
                };
                setProgrammer(frontEnd);
                return new Program[]{frontEnd.getProgram(this)};
            }
        });

        director.runProject("여행사A 프론트개편");
```

```java
public class Main {
    public static void main(String[] args){
        Director director = new Director();
        director.addProject("여행사A 프론트개편", new Client() {
            @Override
            public Program[] run() {
                FrontEnd frontEnd = new FrontEnd<Client>(){
                    @Override
                    void setData(Client paper) {
                        library = paper.library;
                        language = paper.language;
                    }
                };
                setProgrammer(frontEnd);
                return new Program[]{frontEnd.getProgram(this)};
            }
        });
        director.runProject("여행사A 프론트개편");
```

```java
public class Main {
    public static void main(String[] args){
        Director director = new Director();
        director.addProject("여행사A 프론트개편", new Client() {
            @Override
            public Program[] run() {
                FrontEnd frontEnd = new FrontEnd<Client>(){
                    @Override
                    void setData(Client paper) {
                        library = paper.library;
                        language = paper.language;
                    }
                };
                programmer = frontEnd;
                return new Program[]{frontEnd.getProgram(this)};
            }
        });

        director.runProject("여행사A 프론트개편");
```

```java
public class Main {
    public static void main(String[] args){
        Director director = new Director();
        director.addProject("여행사A 프론트개편", new Client() {
            @Override
            public Program[] run() {
                FrontEnd frontEnd = new FrontEnd<Client>(){
                    @Override
                    void setData(Client paper) {
                        library = paper.library;
                        language = paper.language
                    }
                };
                programmer = frontEnd;
                return new Program[]{frontEnd.get
            }
        });

        director.runProject("여행사A 프론트개편");
```

```java
public abstract class Client implements Paper{
    Library library = new Library("vueJS");
    Language language = new Language("kotlinJS");
    protected Programmer programmer;
    ~~public void setProgrammer(Programmer programmer){~~
        ~~this.programmer = programmer;~~
    ~~}~~
}
```

```java
director.addProject("xx은행 리뉴얼", new ServerClient() {
    @Override
    public Program[] run() {
        Programmer frontEnd = new FrontEnd<ServerClient>(){
            @Override void setData(ServerClient paper){language = paper.frontEndLanguage;}
        };
        Programmer backEnd = new BackEnd<ServerClient>(){
            @Override void setData(ServerClient paper) {
                server = paper.server;
                language = paper.backEndLanguage;
            }
        };
        frontEndProgrammer = frontEnd;
        backEndProgrammer = backEnd;
        return new Program[]{frontEnd.getProgram(this), backEnd.getProgram(this)};
    }
});
director.runProject("xx은행 리뉴얼");
director.addProject("xx은행 리뉴얼", new ServerClient() {
    @Override
    public Program[] run() {
        Programmer frontEnd = new FrontEn
            @Override void setData(Server
        };
        Programmer backEnd = new BackEnd<
            @Override void setData(Server
                server = paper.server;
                language = paper.backEndL
            }
        };
        frontEndProgrammer = frontEnd;
        backEndProgrammer = backEnd;
        return new Program[]{frontEnd.get
    }
});
director.runProject("xx은행 리뉴얼");
```

```java
public abstract class ServerClient implements Paper{
    Server server = new Server("test");
    Language backEndLanguage = new Language("java");
    Language frontEndLanguage = new Language("kotlinJS");
    protected Programmer backEndProgrammer;
    protected Programmer frontEndProgrammer;
    public void setBackEndProgrammer(Programmer programmer){
        backEndProgrammer = programmer;
    }
    public void setFrontEndProgrammer(Programmer programmer){
        frontEndProgrammer = programmer;
    }
}
```