CODE  SPITZ

83

OBJECT

1 2 3 4 5 6

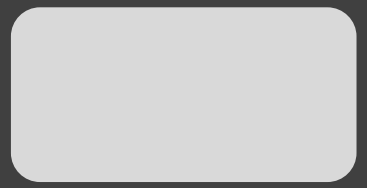
분해

Functional decomposition

Flow chart 기법

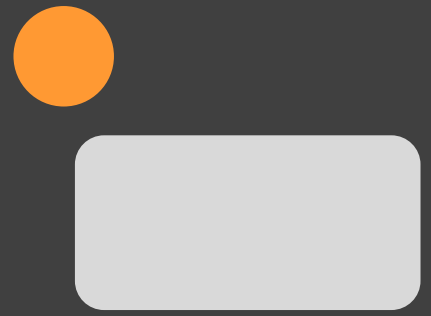
Functional decomposition

Flow chart 기법



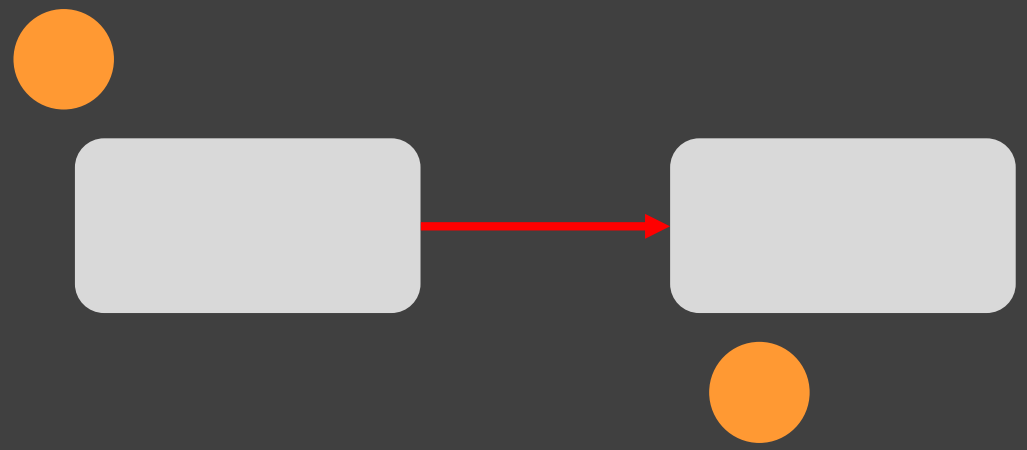
Functional decomposition

Flow chart 기법



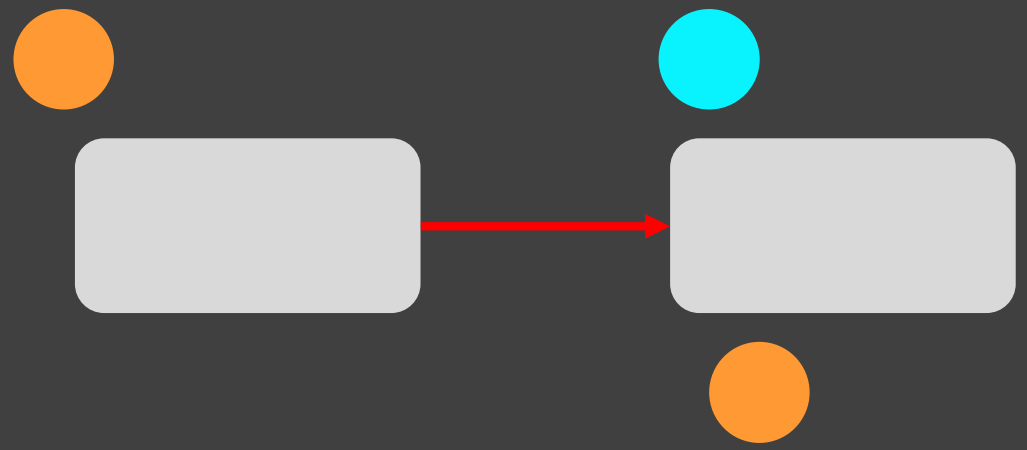
Functional decomposition

Flow chart 기법



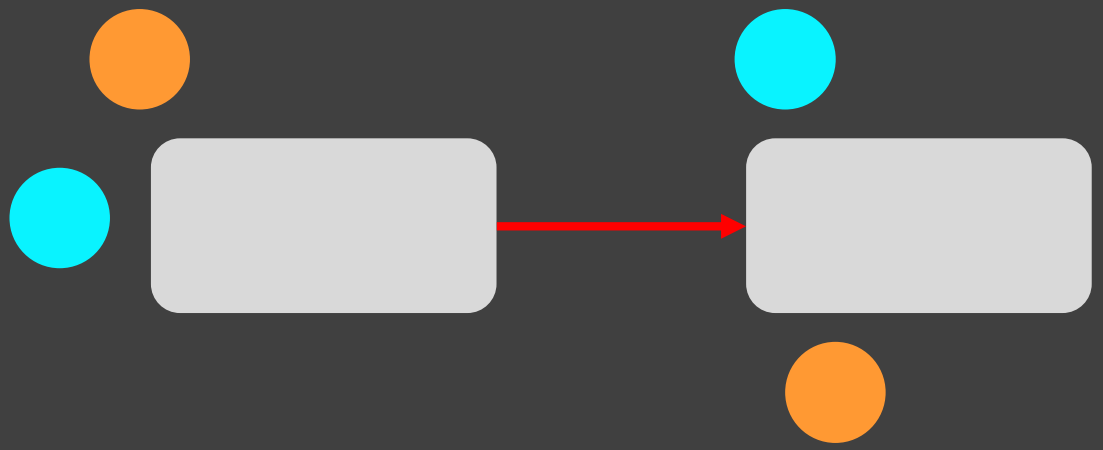
Functional decomposition

Flow chart 기법



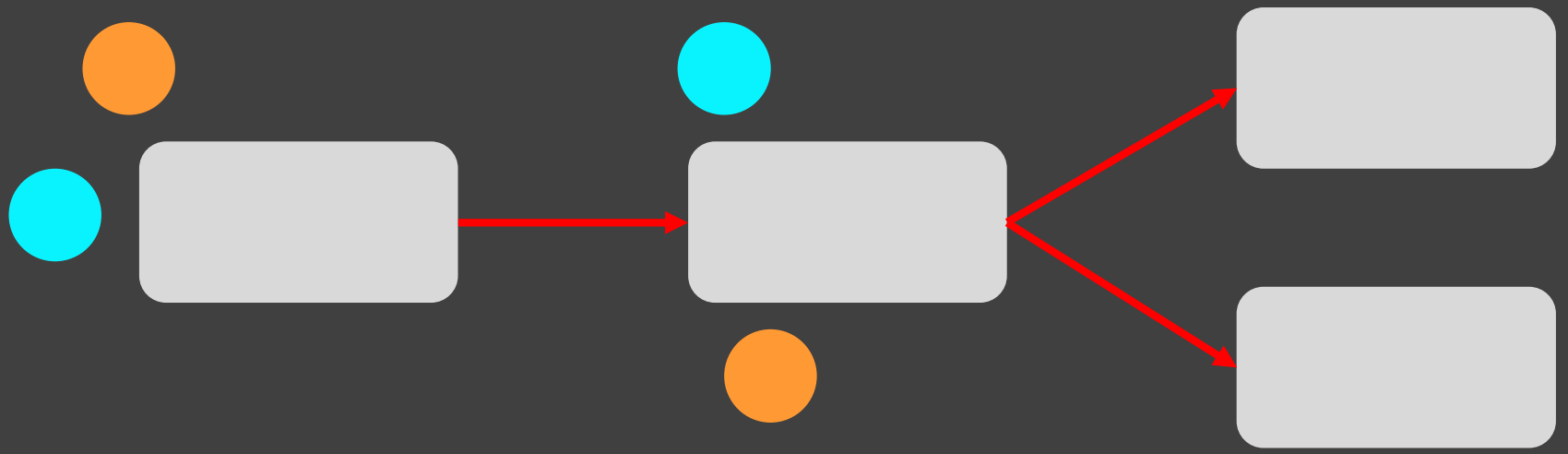
Functional decomposition

Flow chart 기법



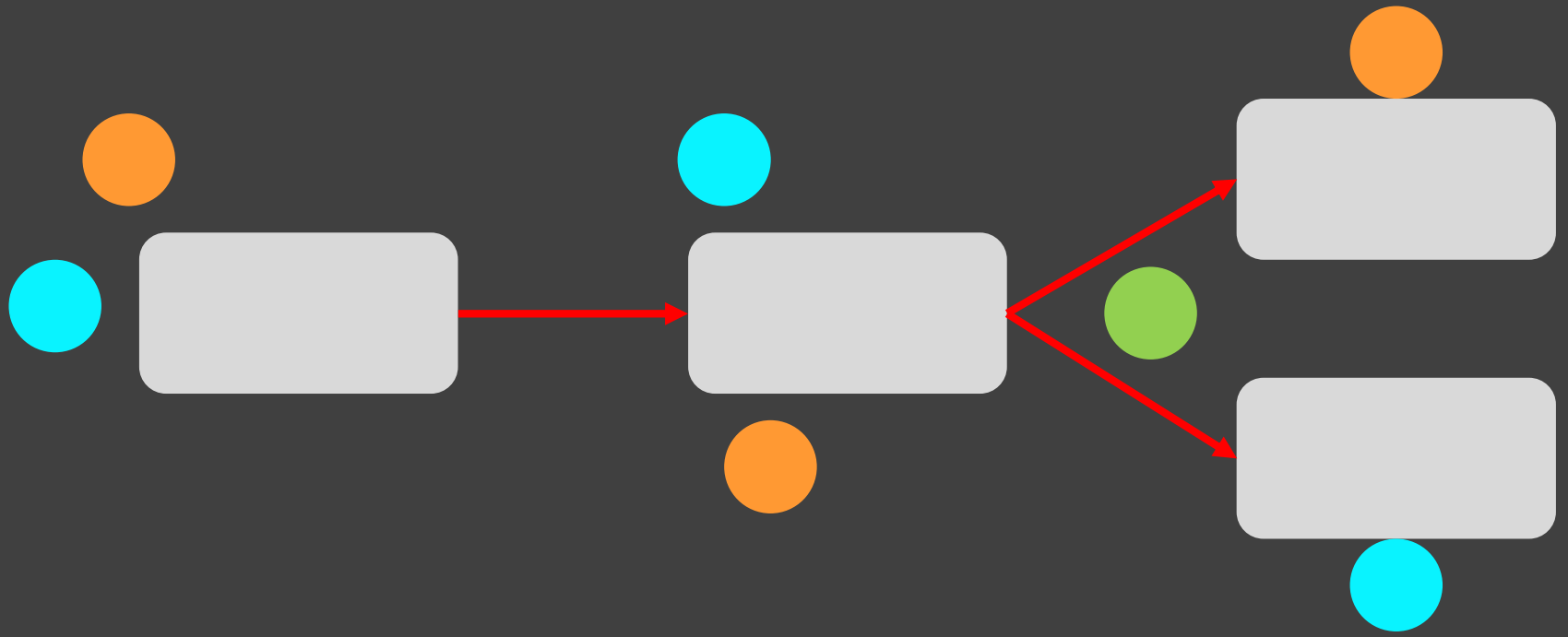
Functional decomposition

Flow chart 기법



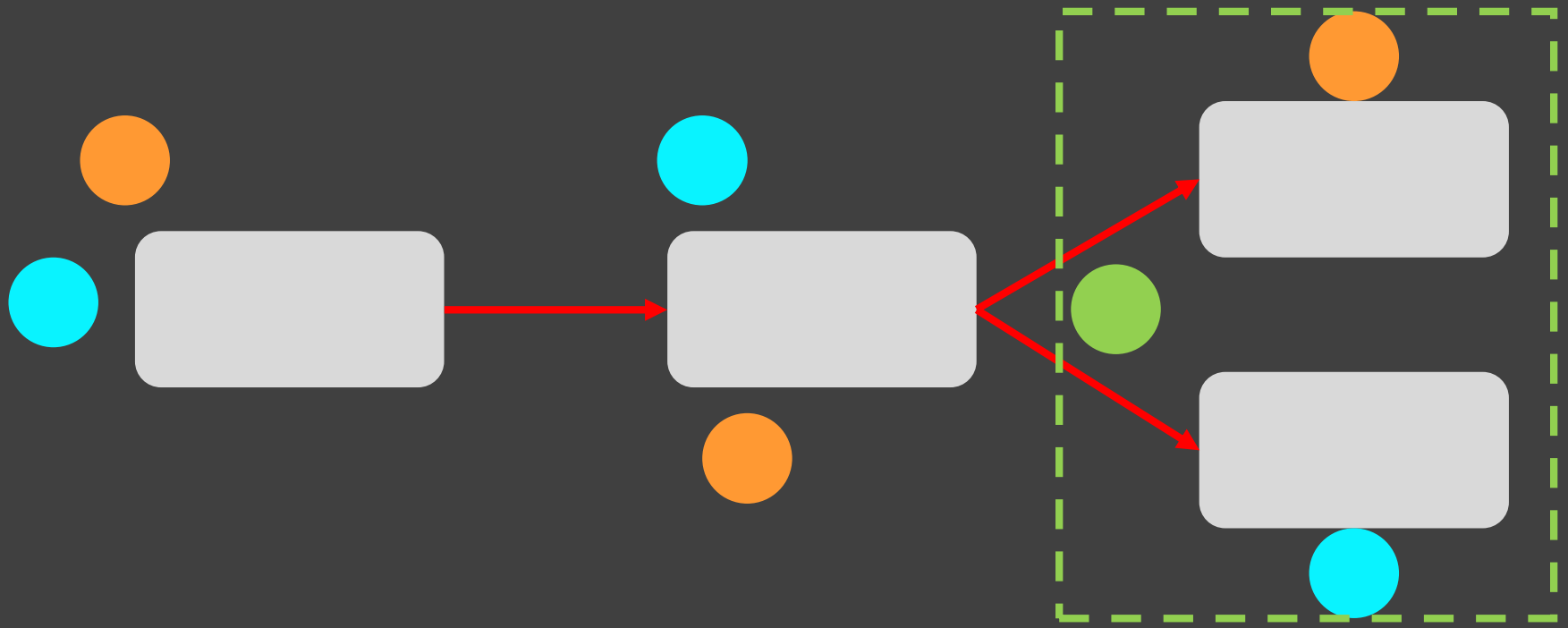
Functional decomposition

Flow chart 기법



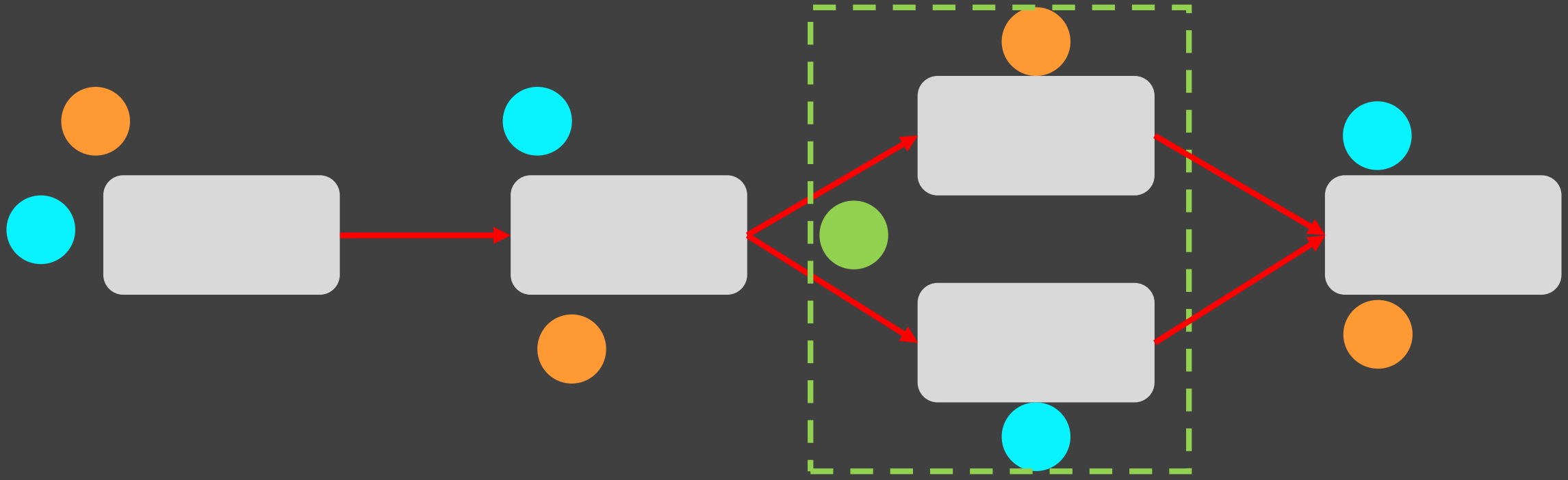
Functional decomposition

Flow chart 기법



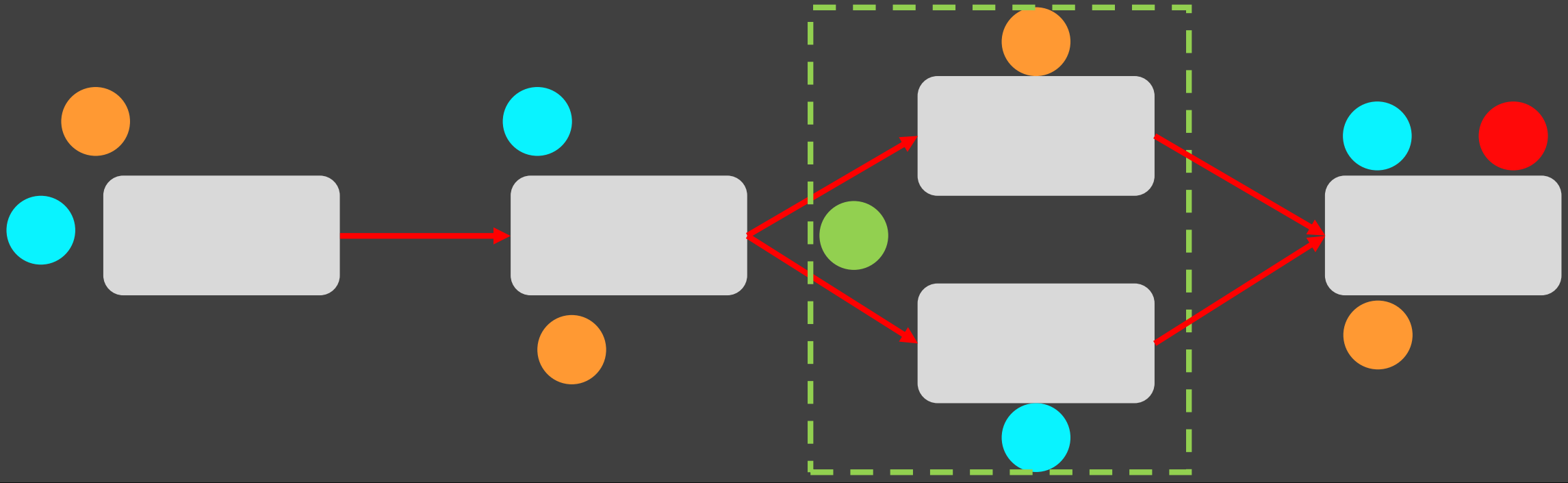
Functional decomposition

Flow chart 기법



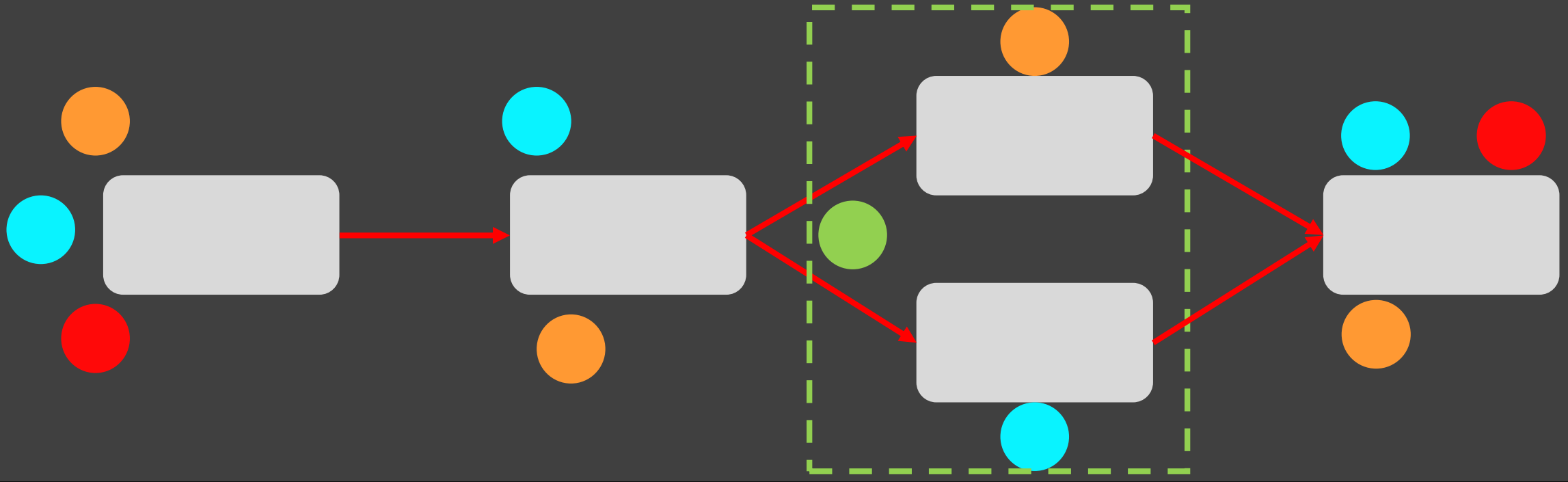
Functional decomposition

Flow chart 기법



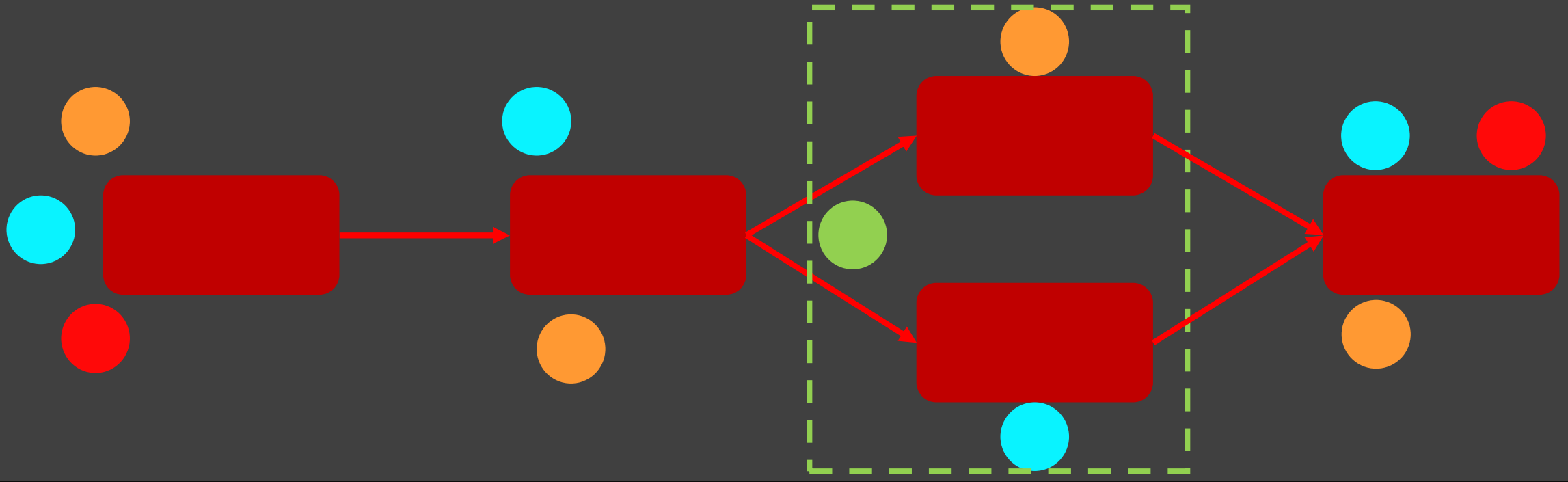
Functional decomposition

Flow chart 기법



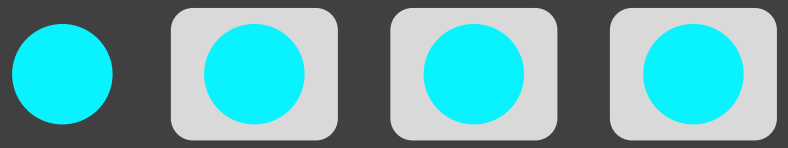
Functional decomposition

Flow chart 기법

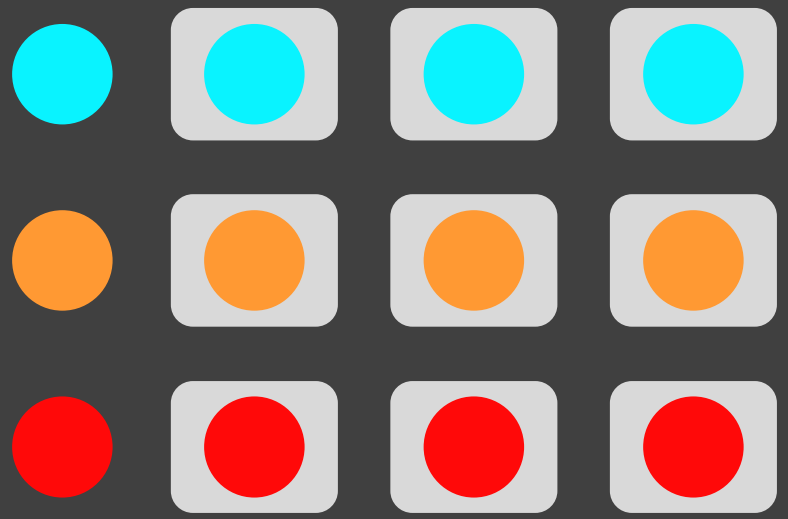


Abstract Data Type

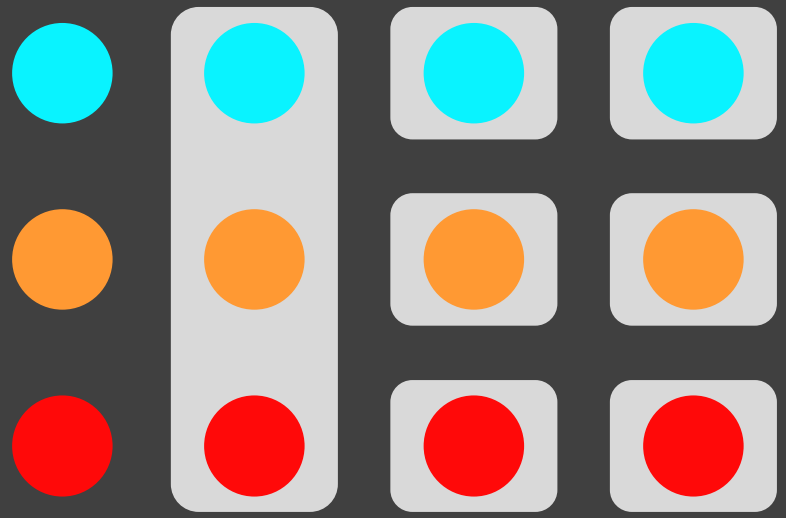
Abstract Data Type



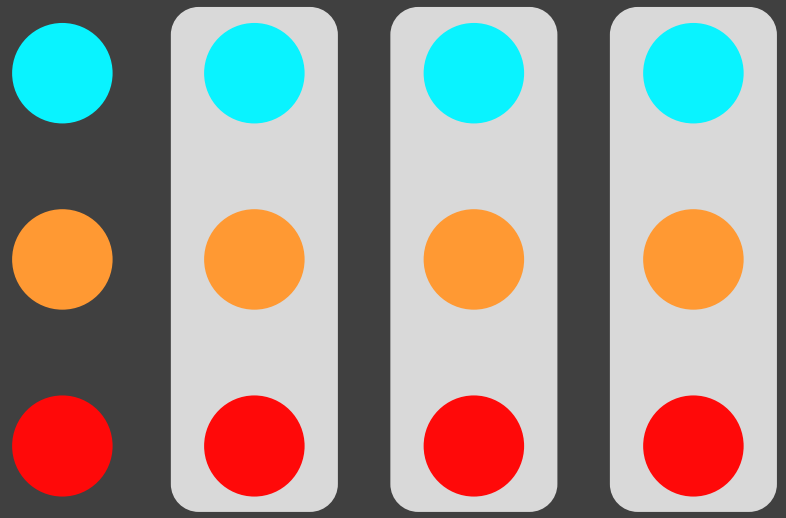
Abstract Data Type



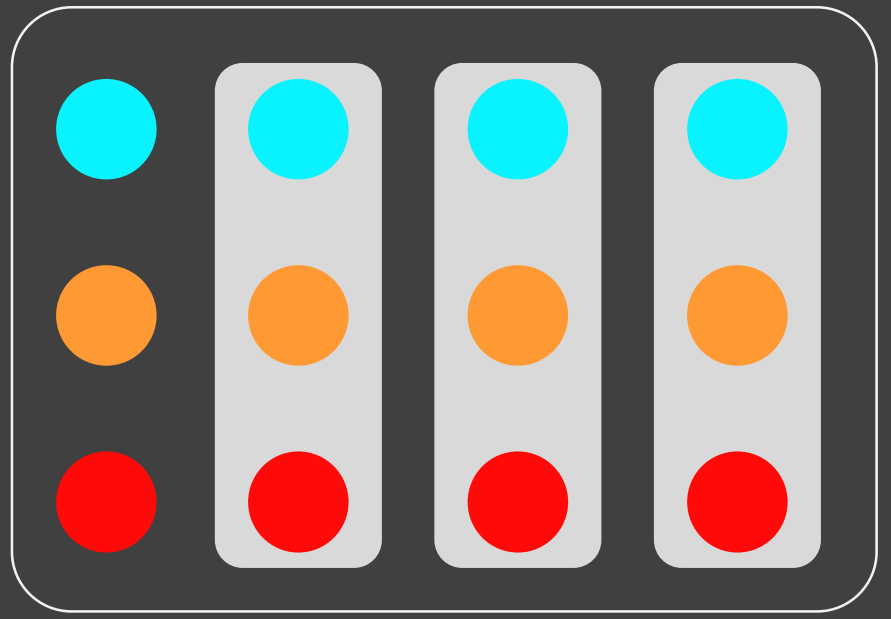
Abstract Data Type



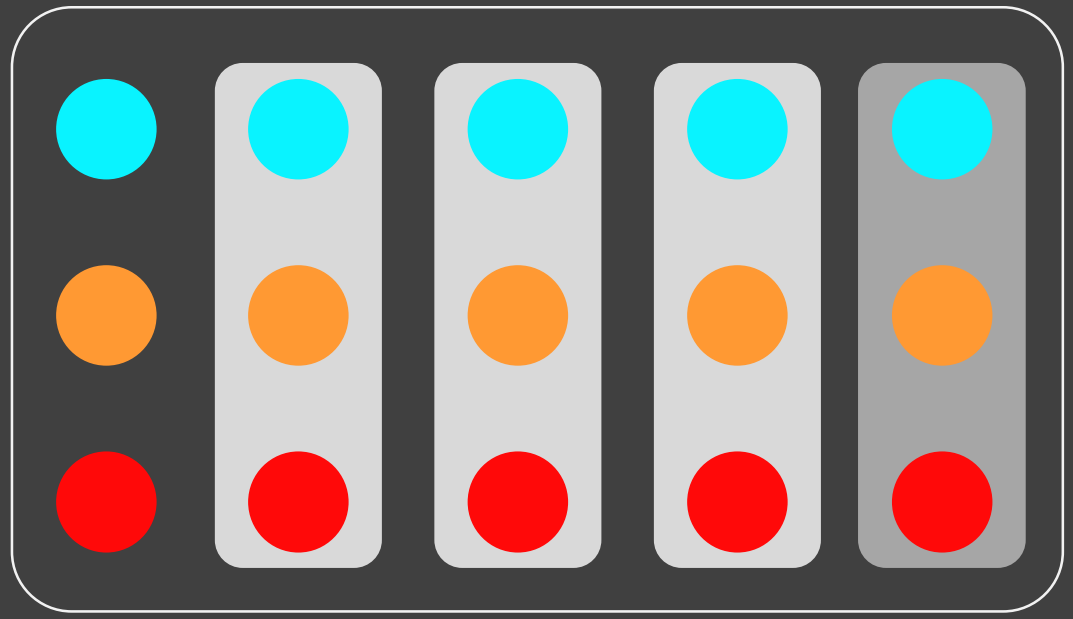
Abstract Data Type



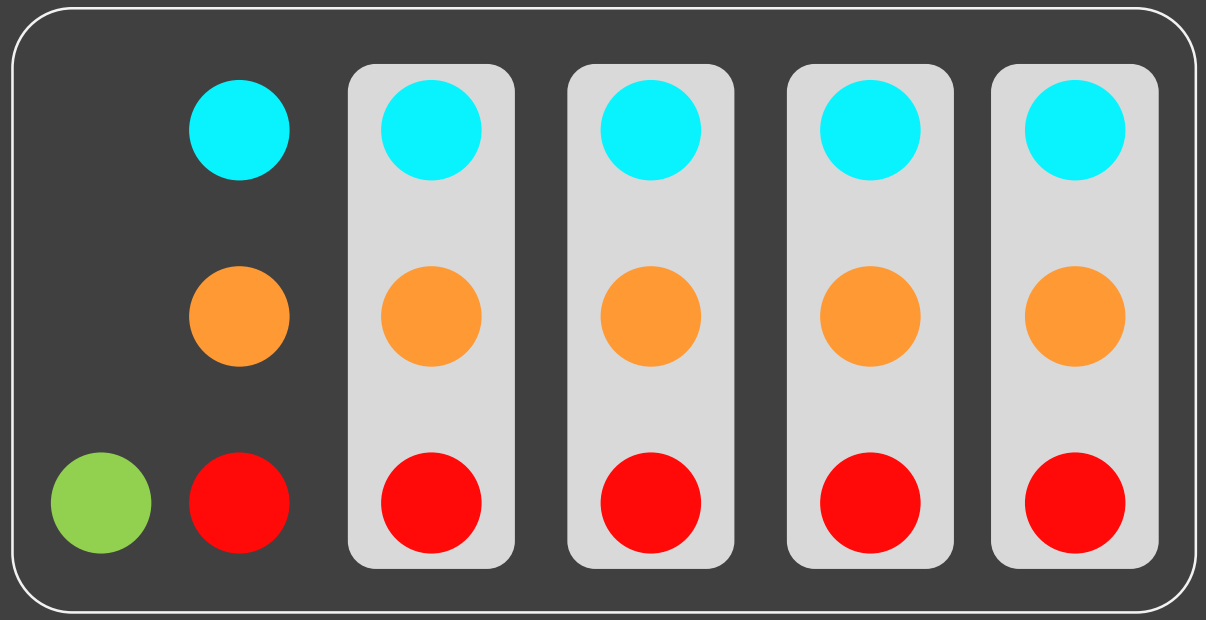
Abstract Data Type



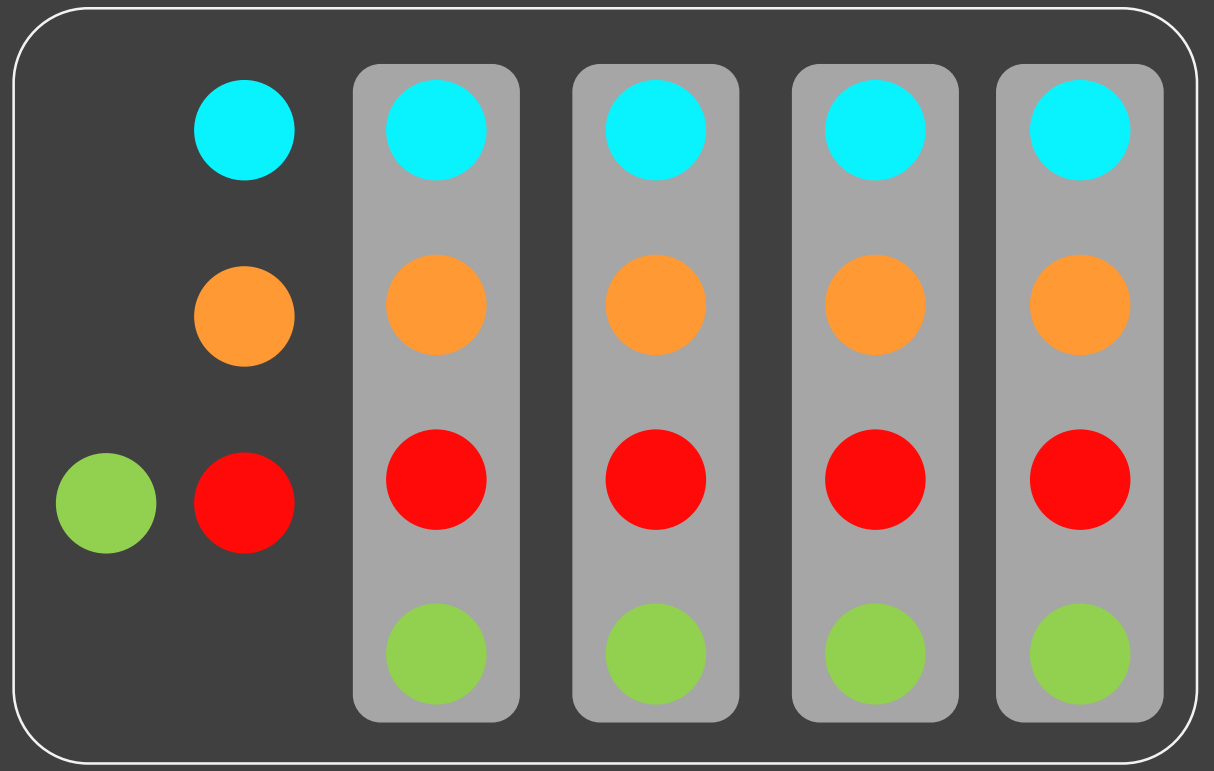
Abstract Data Type



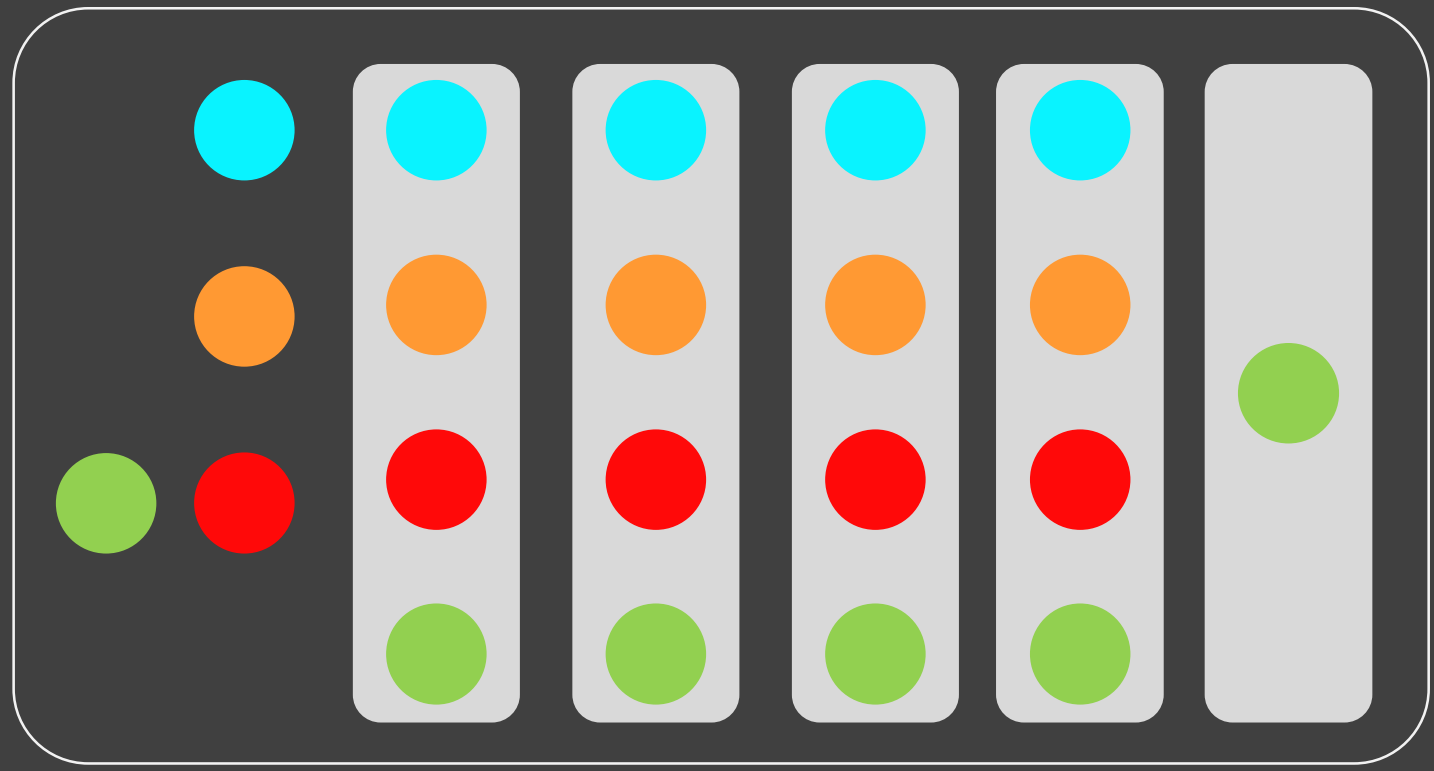
Abstract Data Type



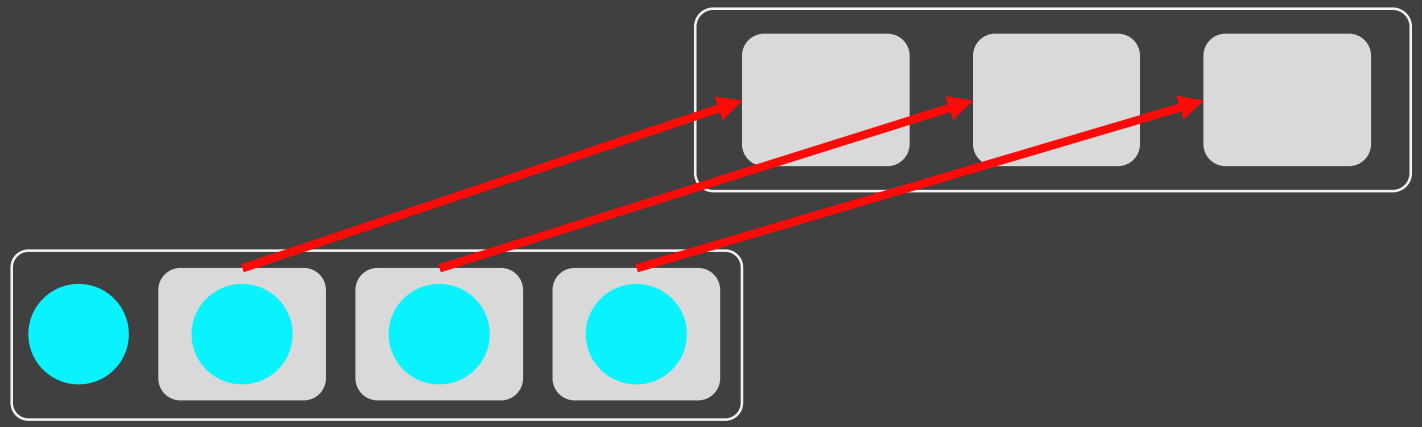
Abstract Data Type



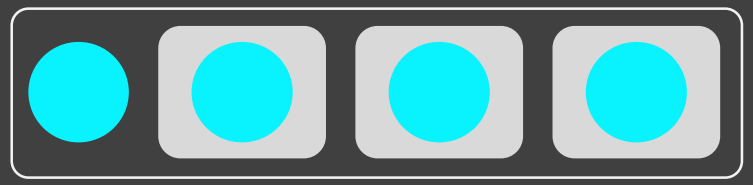
Abstract Data Type

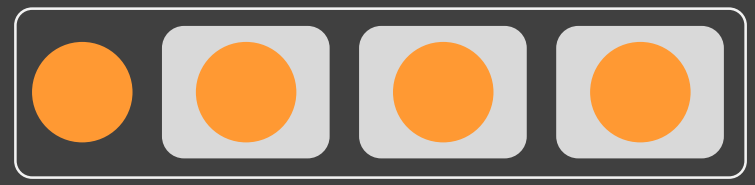


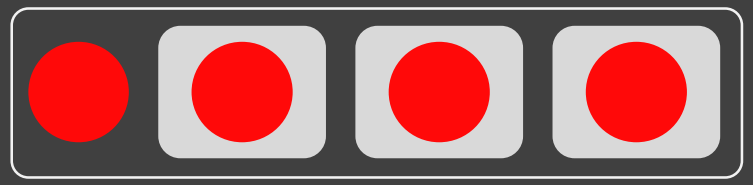
Object Oriented



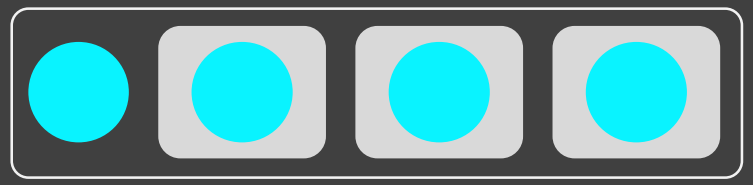
Object Oriented

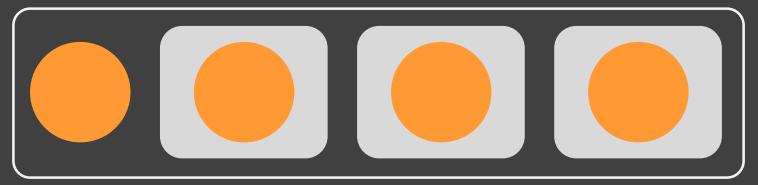


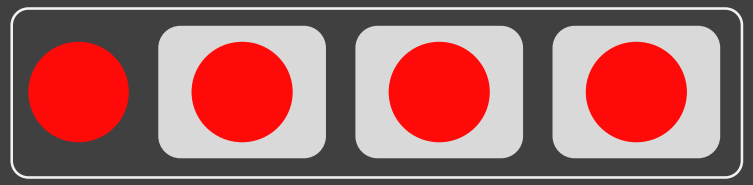


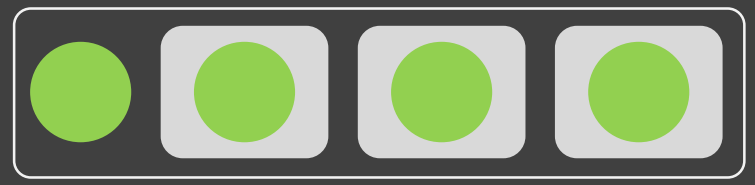


Object Oriented



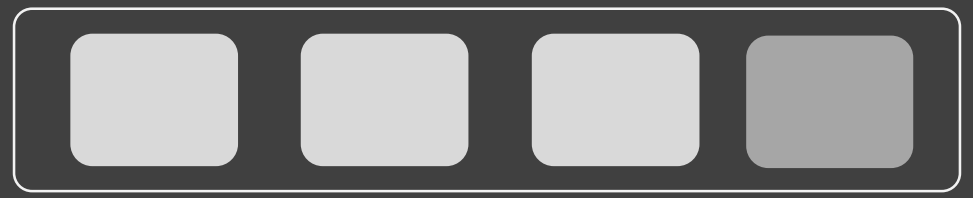


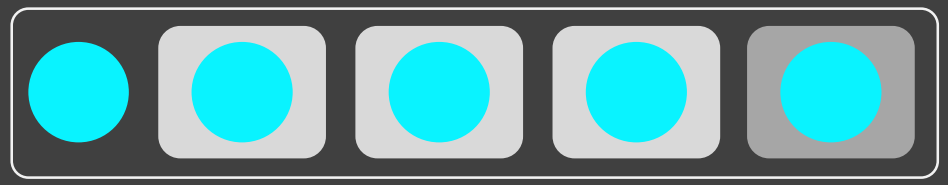


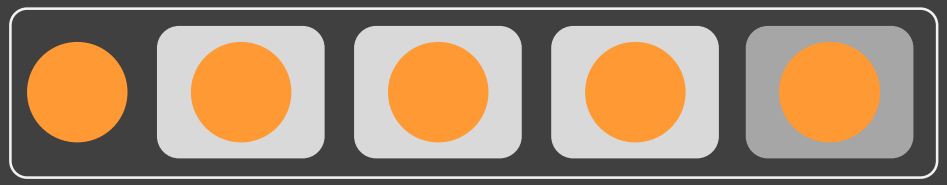


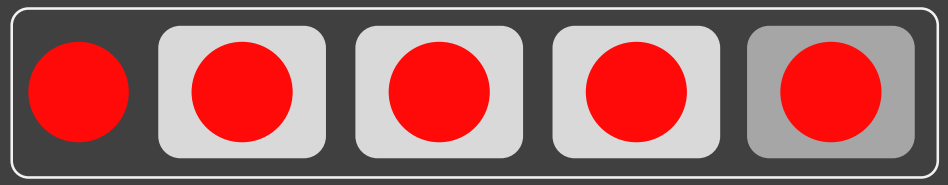
Object Oriented

Object Oriented

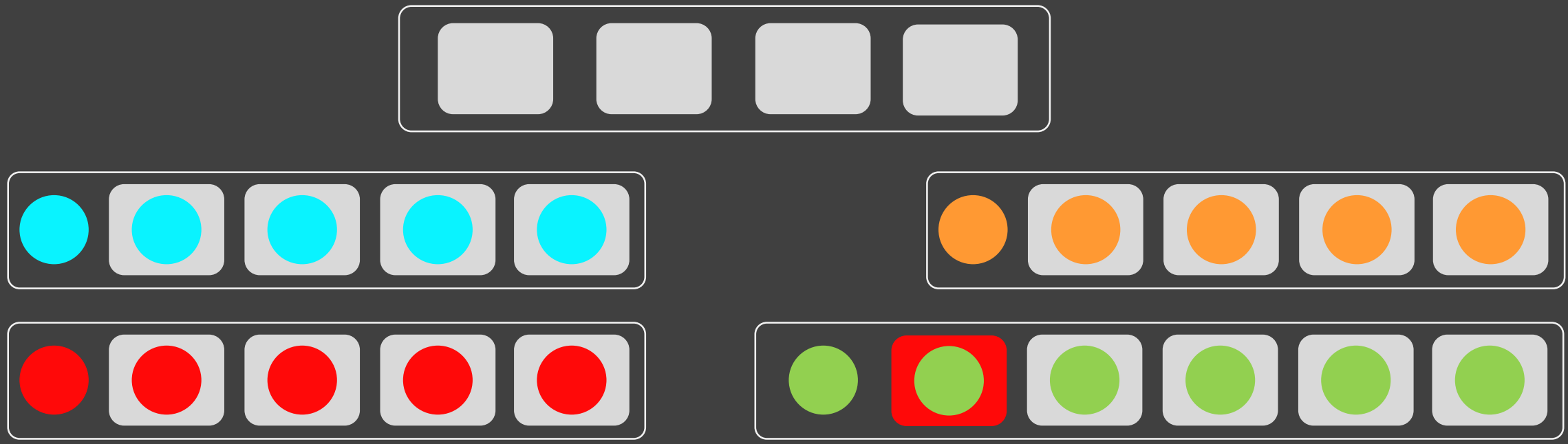




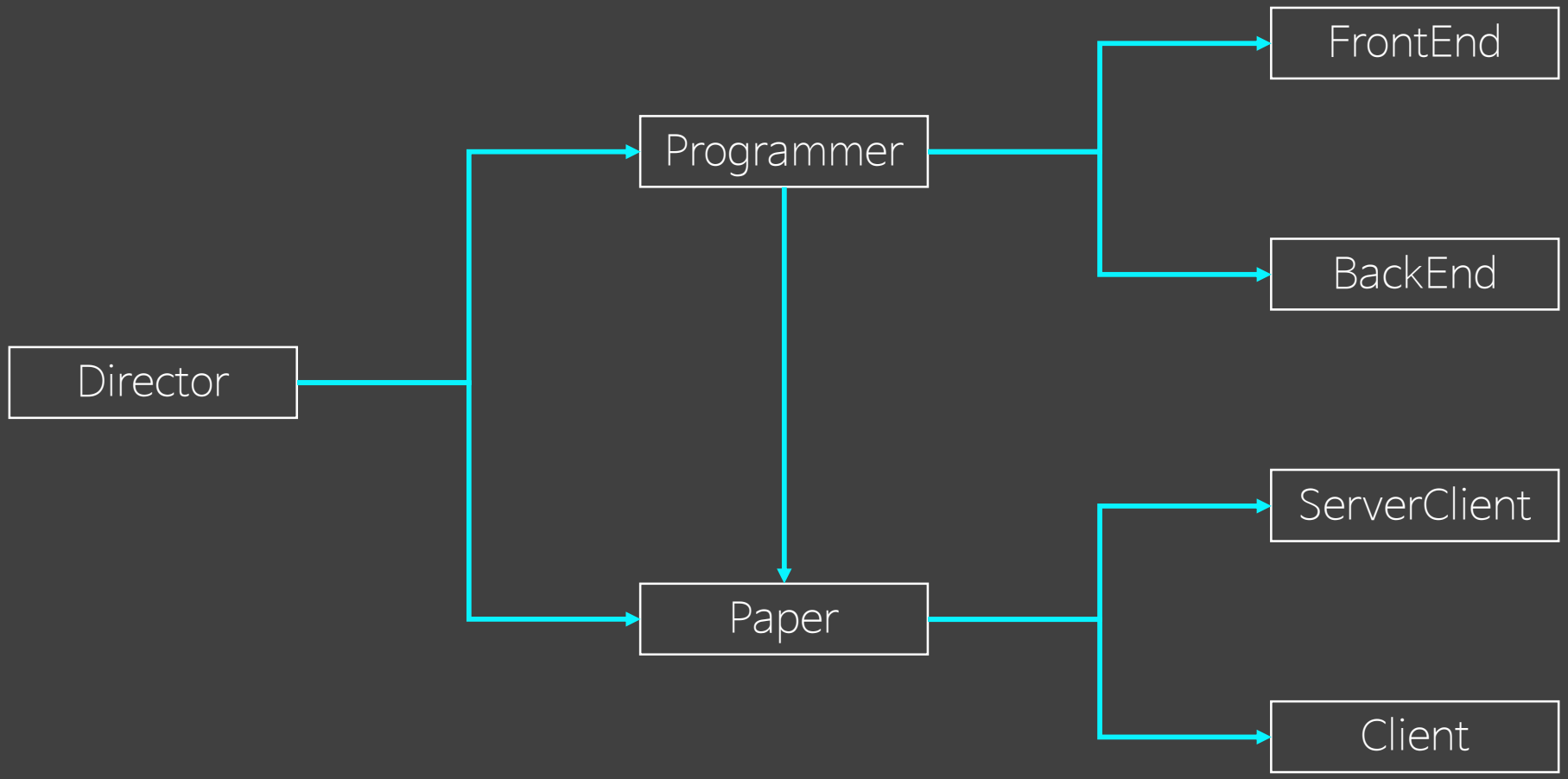




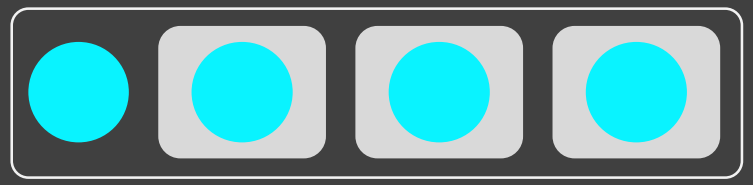
Object Oriented

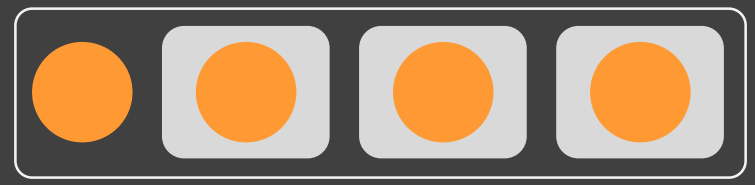


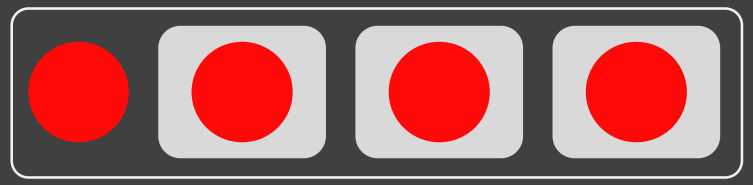
개발자의 세계 ADT


FrontEnd
Programmer
BackEnd
Director
ServerClient
Paper
Client

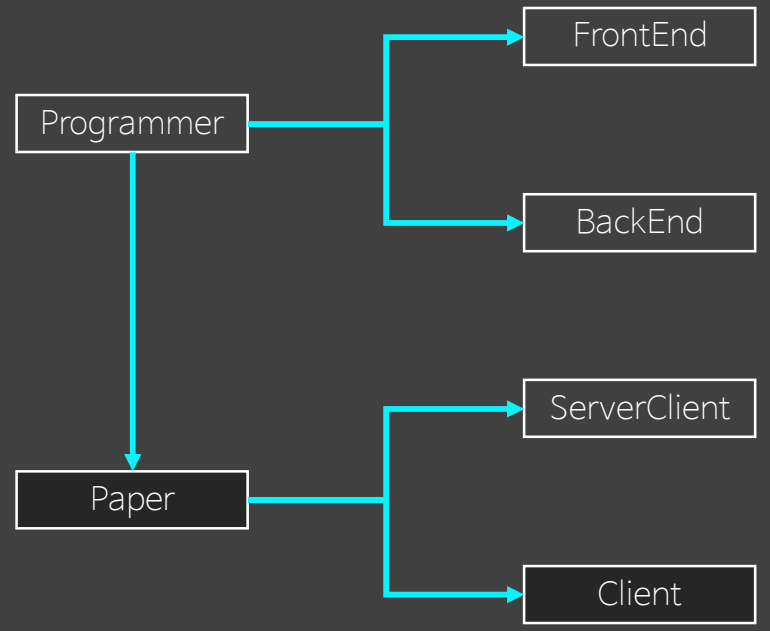
Object Oriented



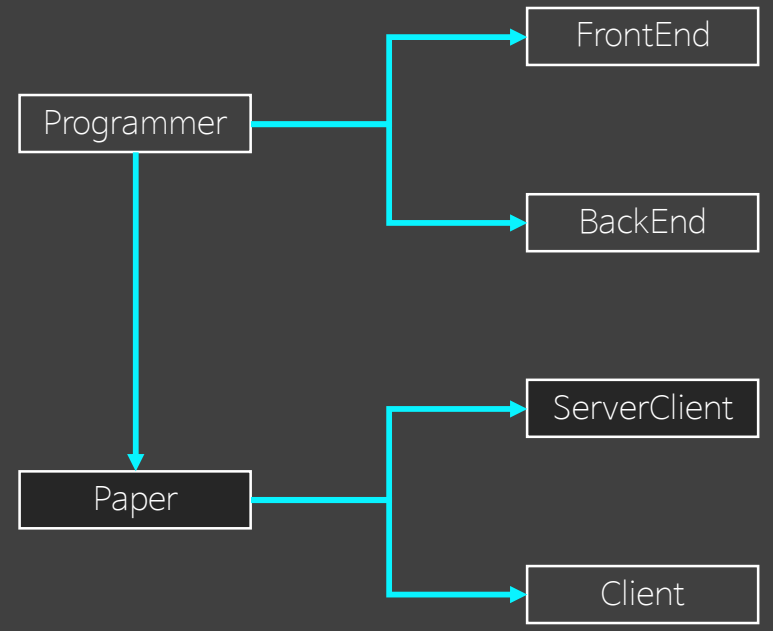




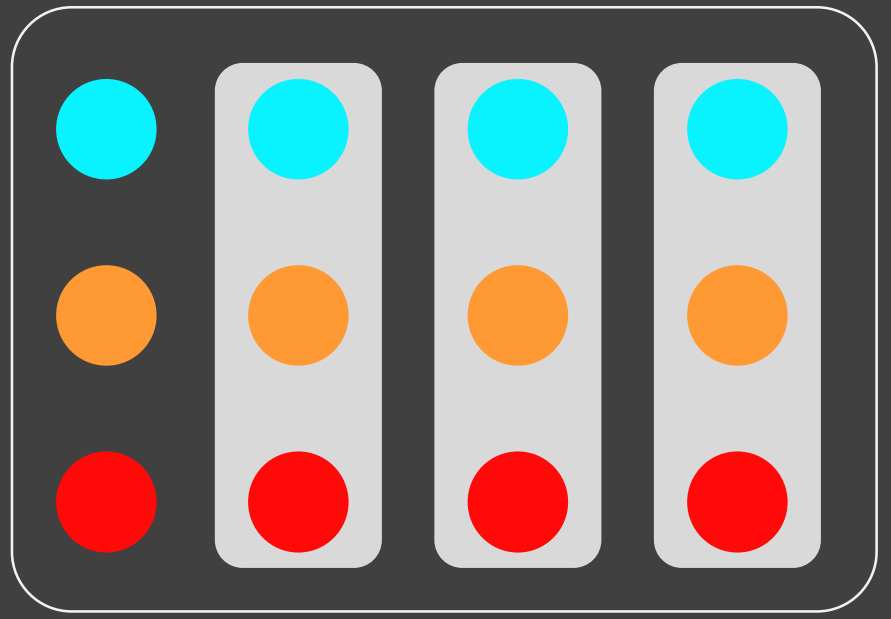
```java
public interface Paper {}
public class Client implements Paper {
    Library library = new Library("vueJS");
    Language language = new Language("kotlinJS");
    Programmer programmer;
    public void setProgrammer(Programmer programmer){
        this.programmer = programmer;
    }
}
```



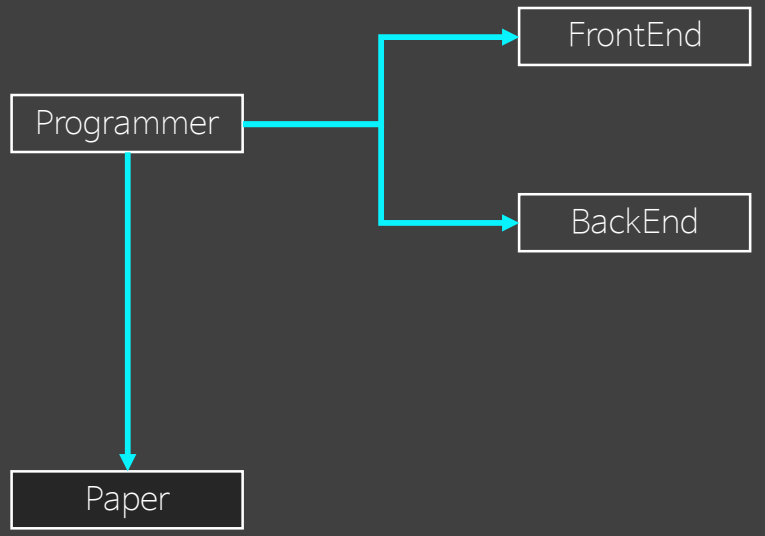
```java
public interface Paper {}

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;
    }
}
```



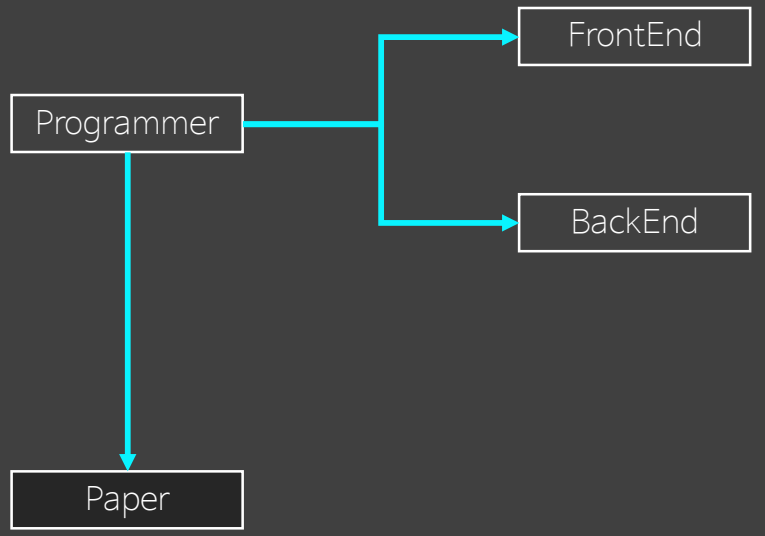
Abstract Data Type



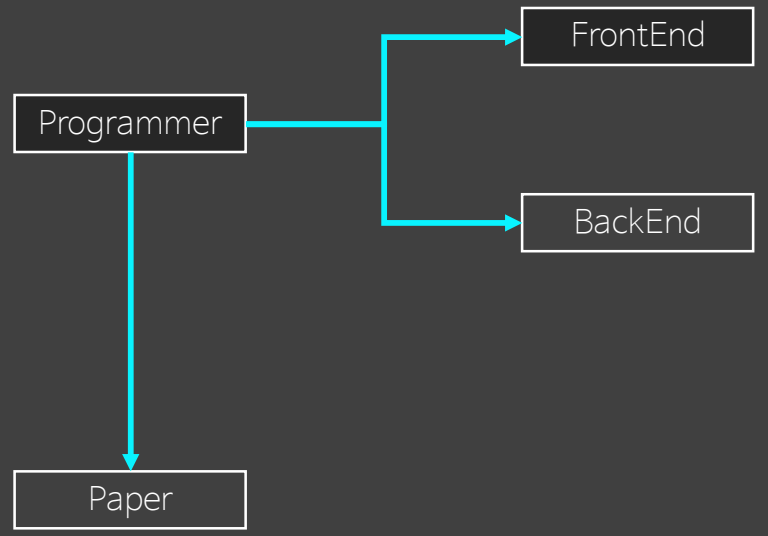
```java
public class Paper{
    public Paper(boolean isClient){this.isClient = isClient;}
    public final boolean isClient;
    Library library = new Library("vueJS");
    Language language = new Language("kotlinJS");
    Programmer programmer;
    Server server = new Server("test");
    Language backEndLanguage = new Language("java");
    Language frontEndLanguage = new Language("kotlinJS");
    private Programmer backEndProgrammer;
    private Programmer frontEndProgrammer;
    public void setBackEndProgrammer(Programmer programmer){
        if(!isClient) backEndProgrammer = programmer;
    }
    public void setFrontEndProgrammer(Programmer programmer){
        if(!isClient) frontEndProgrammer = programmer;
    }
    public void setProgrammer(Programmer programmer){
        if(isClient) this.programmer = programmer;
    }
}
```



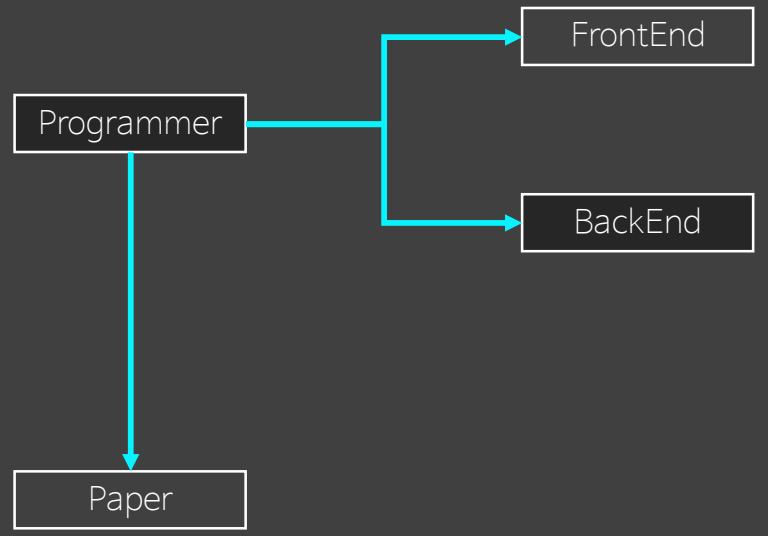
```java
public class Paper{
    public Paper(boolean isClient){this.isClient = isClient;}
    public final boolean isClient;
    Library library = new Library("vueJS");
    Language language = new Language("kotlinJS");
    Programmer programmer;
    Server server = new Server("test");
    Language backEndLanguage = new Language("java");
    Language frontEndLanguage = new Language("kotlinJS");
    private Programmer backEndProgrammer;
    private Programmer frontEndProgrammer;
    public void setBackEndProgrammer(Programmer programmer){
        if(!isClient) backEndProgrammer = programmer;
    }
    public void setFrontEndProgrammer(Programmer programmer){
        if(!isClient) frontEndProgrammer = programmer;
    }
    public void setProgrammer(Programmer programmer){
        if(isClient) this.programmer = programmer;
    }
}
```



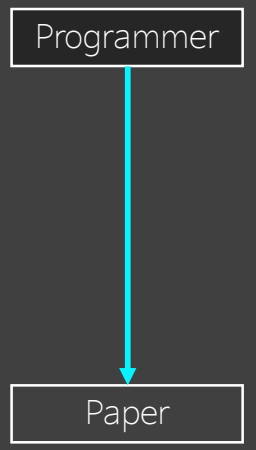
```java
public interface Programmer {
    Program makeProgram(Paper paper);
}

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();}
}
```



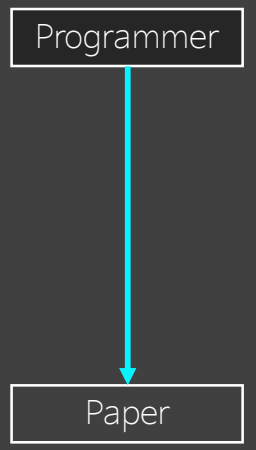
```java
public interface Programmer {
    Program makeProgram(Paper paper);
}

public class 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();}
}
```



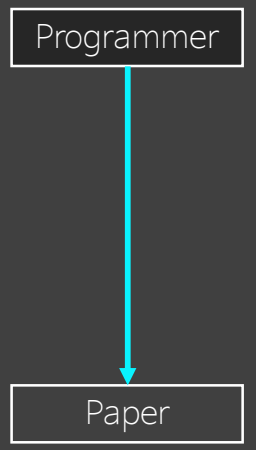
```java
public class Programmer {
    public Programmer(Boolean isFrontEnd){this.isFrontEnd = isFrontEnd;}
    public final boolean isFrontEnd;
    private Language frontLanguage;
    private Library frontLibrary;
    private Server server;
    private Language backEndLanguage;
    public Program makeProgram(Paper paper) {
        if(isFrontEnd){
            frontLanguage = paper.getFrontEndLanguage();
            frontLibrary = paper.getFrontEndLibrary();
        }else{
            this.server = paper.getServer();
            this.backEndLanguage = paper.getBackEndLanguage();
        }
        return isFrontEnd ? makeFrontEndProgram() : makeBackEndProgram();
    }
    private Program makeFrontEndProgram(){return new Program();}
    private Program makeBackEndProgram(){return new Program();}
}
```



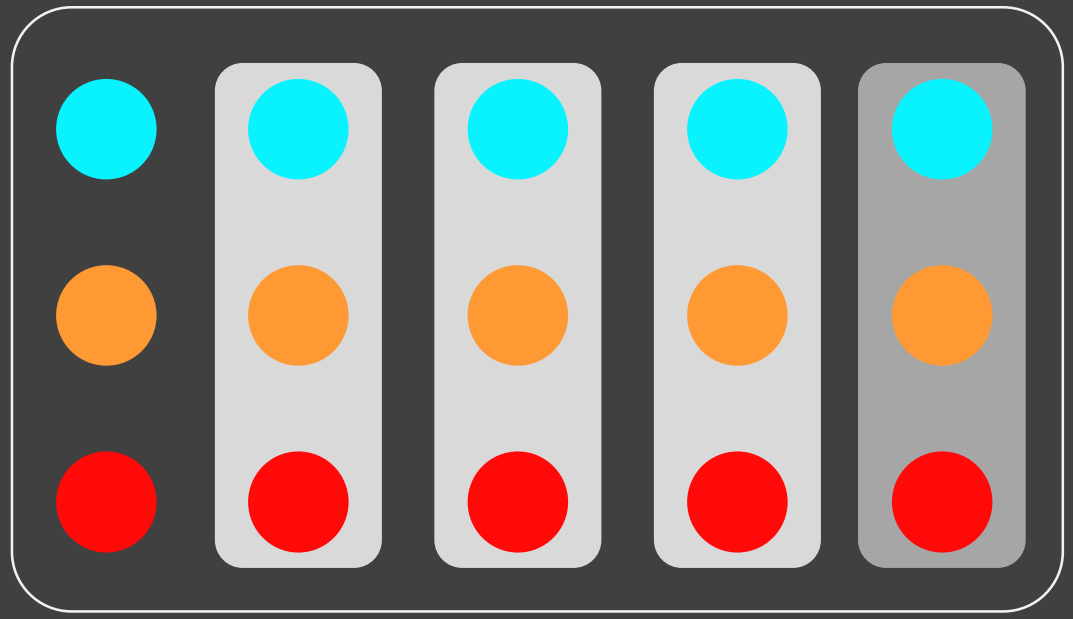
```java
public class Programmer {
    public Programmer(Boolean isFrontEnd){this.isFrontEnd = isFrontEnd;}
    public final boolean isFrontEnd;
    private Language frontLanguage;
    private Library frontLibrary;
    private Server server;
    private Language backEndLanguage;
    public Program makeProgram(Paper paper) {
        if(isFrontEnd){
            frontLanguage = paper.getFrontEndLanguage();
            frontLibrary = paper.getFrontEndLibrary();
        }else{
            this.server = paper.getServer();
            this.backEndLanguage = paper.getBackEndLanguage();
        }
        return isFrontEnd ? makeFrontEndProgram() : makeBackEndProgram();
    }
    private Program makeFrontEndProgram(){return new Program();}
    private Program makeBackEndProgram(){return new Program();}
}
```



```java
public class Programmer {
    public Programmer(Boolean isFrontEnd){this.isFrontEnd = isFrontEnd;}
    public final boolean isFrontEnd;
    private Language frontLanguage;
    private Library frontLibrary;
    private Server server;
    private Language backEndLanguage;
    public Program makeProgram(Paper paper) {
        if(isFrontEnd){
            frontLanguage = paper.getFrontEndLanguage();
            frontLibrary = paper.getFrontEndLibrary();
        }else{
            this.server = paper.getServer();
            this.backEndLanguage = paper.getBackEndLanguage();
        }
        return isFrontEnd ? makeFrontEndProgram() : makeBackEndProgram();
    }
    private Program makeFrontEndProgram(){return new Program();}
    private Program makeBackEndProgram(){return new Program();}
}
```

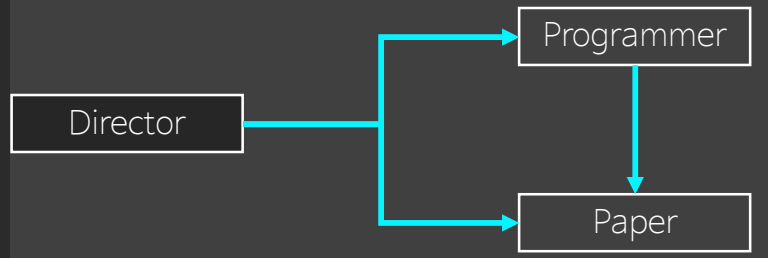


Abstract Data Type

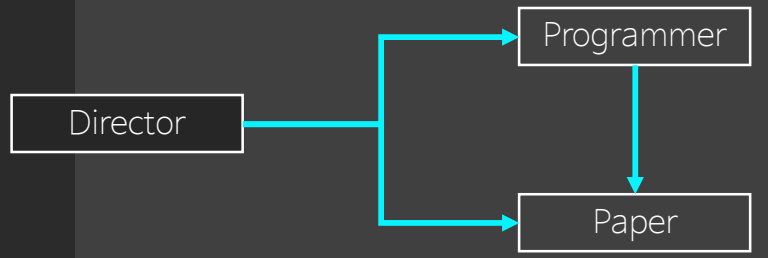


```
public class Paper{
  public Paper(boolean isClient){this.isClient = isClient;}
  public final boolean isClient;
  Library library = new Library("vueJS");
  Language language = new Language("kotlinJS");
  Programmer programmer;
  Server server = new Server("test");
  Language backEndLanguage = new Language("java");
  Language frontEndLanguage = new Language("kotlinJS");
  private Programmer backEndProgrammer;
  private Programmer frontEndProgrammer;
  public void setBackEndProgrammer(Programmer programmer){if(!isClient) backEndProgrammer = programmer;}
  public void setFrontEndProgrammer(Programmer programmer){if(!isClient) frontEndProgrammer = programmer;}
  public void setProgrammer(Programmer programmer){if(isClient) this.programmer = programmer;}
  public Language getFrontEndLanguage(){return isClient ? language : frontEndLanguage;}
  public Library getFrontEndLibrary(){return isClient ? library : null;}
  public Server getServer(){return isClient ? null : server;}
  public Language getBackEndLanguage(){return isClient ? null : backEndLanguage;}
}
```

```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 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.isClient){
        Programmer frontEnd = new Programmer(true), backEnd = new Programmer(false);
        paper.setFrontEndProgrammer(frontEnd);
        paper.setBackEndProgrammer(backEnd);
        Program client = frontEnd.makeProgram(paper);
        Program server = backEnd.makeProgram(paper);
        deploy(name, client, server);
    }else{
        Programmer frontEnd = new Programmer(true);
        paper.setProgrammer(frontEnd);
        deploy(name, frontEnd.makeProgram(paper));
    }
  }
  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.isClient){
        Programmer frontEnd = new Programmer(true), backEnd = new Programmer(false);
        paper.setFrontEndProgrammer(frontEnd);
        paper.setBackEndProgrammer(backEnd);
        Program client = frontEnd.makeProgram(paper);
        Program server = backEnd.makeProgram(paper);
        deploy(name, client, server);
    }else{
        Programmer frontEnd = new Programmer(true);
        paper.setProgrammer(frontEnd);
        deploy(name, frontEnd.makeProgram(paper));
    }
  }
  private void deploy(String projectName, Program... programs){}
}
```