Java Design Pattern - Proxy

발표자: 한지원

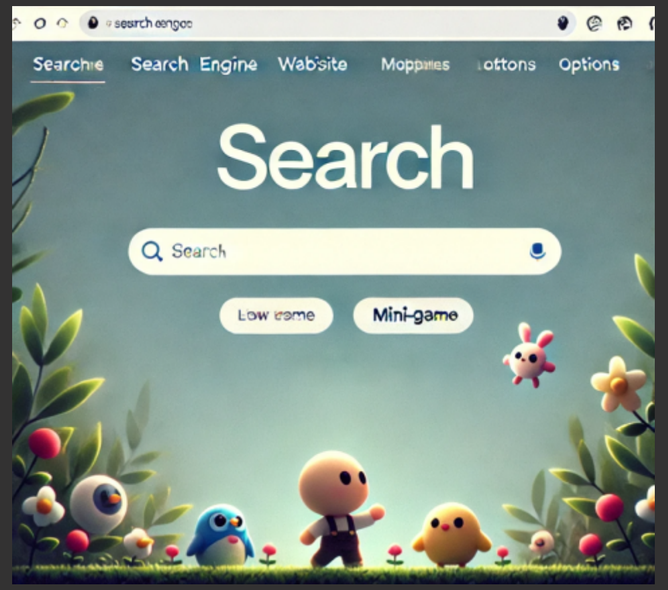
Quiz

```java
@Test  new *
void gameTest(){

    MiniGame miniGame = new MiniGame();
    miniGame.setUserName("쥴리");
    miniGame.welcomeMessage();

    // 사용자 입력에 따라 게임 실행
    String userInput = "start";

    if ("start".equals(userInput)) {
        miniGame.playGame();
    }
}
```



✓ Tests passed: 1 of 1 test – 3 sec 27 ms

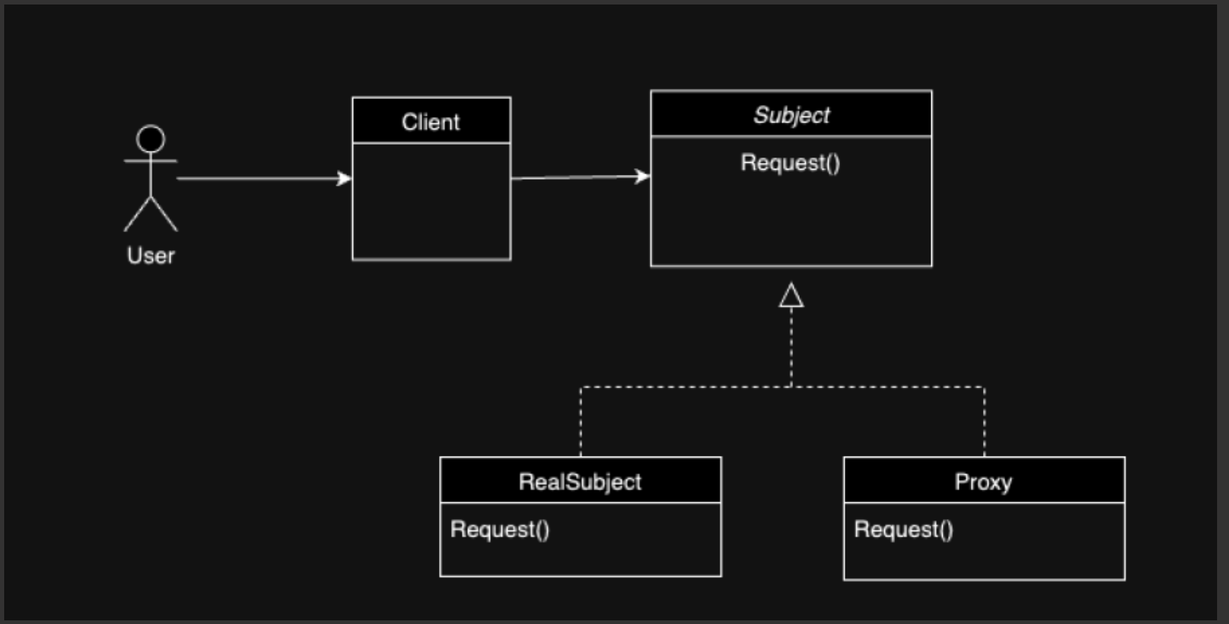
```
MiniGame 초기화 작업이 완료됐습니다.
쥴리님, MiniGame 에 오신 것을 환영합니다 : )
START!!
```

Proxy 패턴이란?

다른 객체에 대한 접근을 제어하기 위한 대리자 역할을 하는
객체를 두는 디자인 패턴

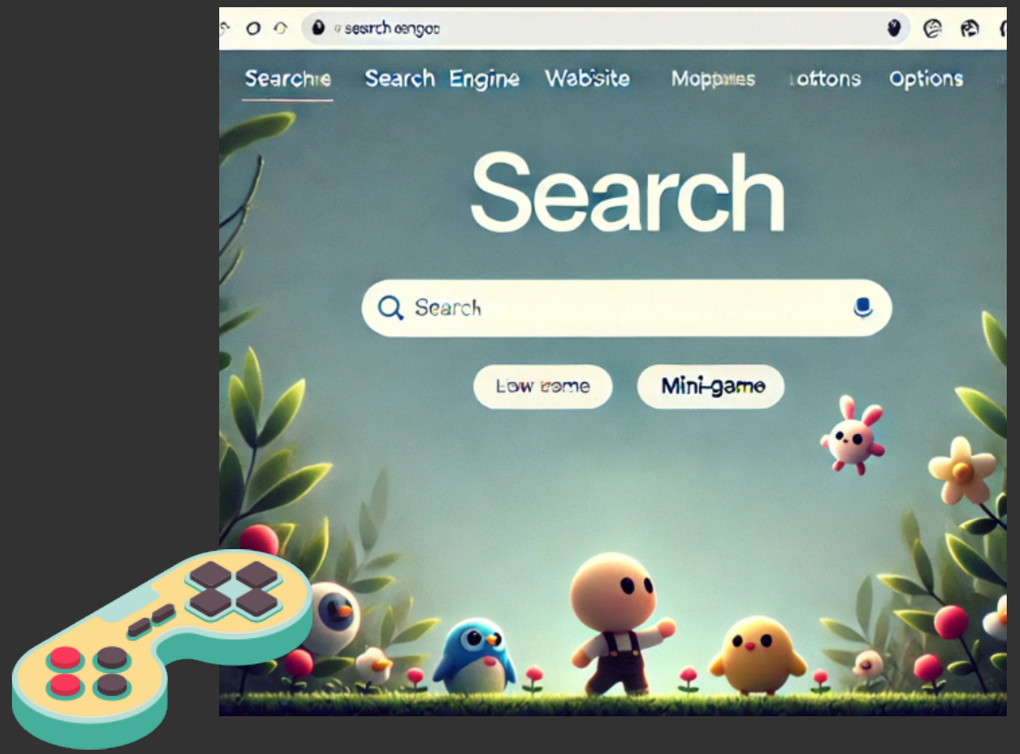
실제로 그 객체를 사용할 때까지
객체 생성과 초기화에 들어가는 **비용 및 시간을 지연**시키는 방법

Proxy 패턴의 구성

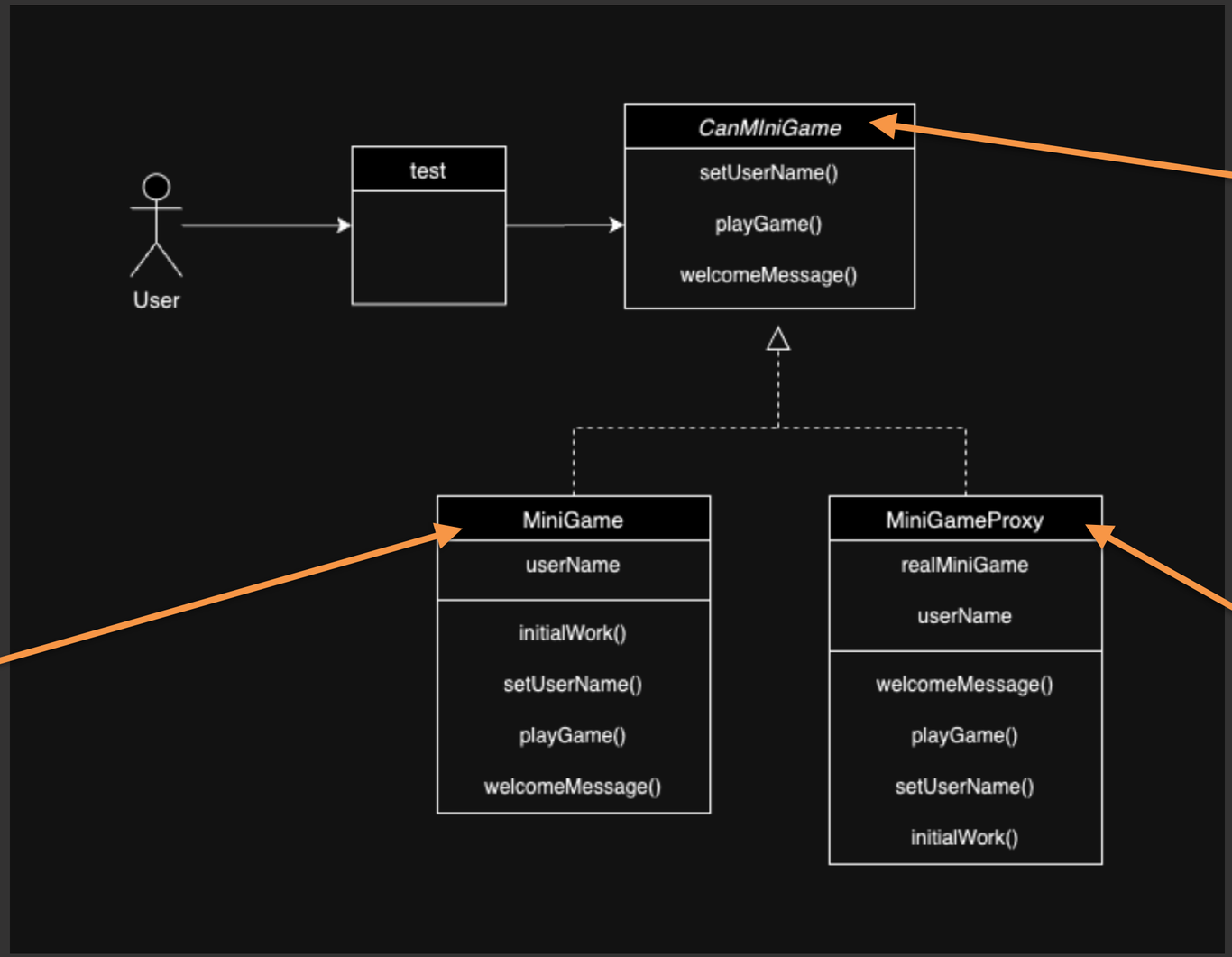


- Proxy: 실제로 참조할 대상에 대한 참조자 관리, Subject와 동일한 인터페이스를 제공하여 실제 대상을 대체할 수 있어야 함. 실제 대상에 대한 접근을 제어하고 실제 대상의 생성과 삭제를 책임
- Subject: RealSubject와 Proxy에 공통 인터페이스를 정의 -> RealSubject가 요청되는 곳에 Proxy를 사용할 수 있도록
- **RealSubject**: 프록시가 대표하는 실제 객체

Proxy 패턴의 예제


Searchie
Search Engine
Website
Mopplies
Lottons
Options
Search
Search
Low game
Mini-game

Proxy 패턴의 예제


CanMIniGame
setUserName()
playGame()
welcomeMessage()
test
User
MiniGame
userName
initialWork()
setUserName()
playGame()
welcomeMessage()
MiniGameProxy
realMiniGame
userName
welcomeMessage()
playGame()
setUserName()
initialWork()
Subject
RealSubject
Proxy

Proxy 패턴의 예제 - MiniGame

```java
public class MiniGame implements CanMiniGame {  4 usages  jiwonhan

    private String userName;  3 usages

    public MiniGame() { initialWork(); }
    public MiniGame(String userName) {...}

    private void initialWork() {  2 usages  jiwonhan
        try {
            Thread.sleep( millis: 3000);
            System.out.println("MiniGame 초기화 작업이 완료됐습니다.");
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
            e.printStackTrace();
        }
    }
```

오래 걸리는 초기 작업 ..

Proxy 패턴의 예제 - CanMiniGame

```
public interface CanMiniGame {  3 usages

    void setUserName(String userName);

    void welcomeMessage();  3 usages  2 ir

    void playGame();  3 usages  2 implemen
}
```

Proxy 객체에서도 실제 객체를 동일하게 이용할 수 있도록 작성

Proxy 패턴의 예제 - Proxy

```java
public class MiniGameProxy implements CanMiniGame {

    private MiniGame realMiniGame;  8 usages

    private String userName;  4 usages

    public MiniGameProxy() {  1 usage  new *
        realMiniGame = null; // 객체 생성을 지연
        userName = "null";
    }

    @Override  3 usages  new *
    public void setUserName(String userName) {...}

    @Override  3 usages  new *
    public void welcomeMessage() {...}

    @Override  3 usages  new *
    public void playGame() {
        if (realMiniGame == null){
            realMiniGame = new MiniGame(userName);
        }
        realMiniGame.playGame();
    }
}
```

```java
private MiniGame realMiniGame;  8 usages
```

✅ MiniGame 참조를 관리

```java
public MiniGameProxy() {  1 usage  new *
    realMiniGame = null; // 객체 생성을 지연
    userName = "null";
}
```

✅ 객체 생성 지연

Proxy 패턴의 예제 - Proxy

```java
public class MiniGameProxy implements CanMiniGame {

    private MiniGame realMiniGame;  8 usages

    private String userName;  4 usages

    public MiniGameProxy() {  1 usage  new *
        realMiniGame = null; // 객체 생성을 지연
        userName = "null";
    }

    @Override  3 usages  new *
    public void setUserName(String userName) {...}

    @Override  3 usages  new *
    public void welcomeMessage() {...}

    @Override  3 usages  new *
    public void playGame() {
        if (realMiniGame == null){
            realMiniGame = new MiniGame(userName);
        }
        realMiniGame.playGame();
    }
}
```

```java
@Override  3 usages  new *
public void welcomeMessage() {
    if (realMiniGame == null){
        System.out.println(userName+ "님, MiniGame 에 오신 것을 환영합니다 : ) ");
    }else{
        realMiniGame.welcomeMessage();
    }
}
```

proxy가 대리할 수 있는 역할

```java
@Override  3 usages  new *
public void playGame() {
    if (realMiniGame == null){
        realMiniGame = new MiniGame(userName);
    }
    realMiniGame.playGame();
}
```

proxy가 대리할 수 없는 역할

Proxy 패턴의 예제 To-be

```java
@Test  new *
void miniGameTestProxy(){
    CanMiniGame miniGame = new MiniGameProxy();
    miniGame.setUserName("쥴리"); // 실제 객체는 생성되지 않았다.
    miniGame.welcomeMessage();

    // 사용자 입력에 따라 게임 실행
    String userInput = "start";

    if ("start".equals(userInput)) {
        miniGame.playGame();
    }
}
```

✓ Tests passed: 1 of 1 test – 3 sec 21 ms

```
쥴리님, MiniGame 에 오신 것을 환영합니다 : )
MiniGame 초기화 작업이 완료됐습니다.
START!!
```

Proxy 패턴의 예제 To-be

Proxy 미사용

✓ Tests passed: 1 of 1 test – 3 sec 26 ms

```
MiniGame 초기화 작업이 완료됐습니다.
쥴리님, MiniGame 에 오신 것을 환영합니다 : )
```

VS

Proxy 사용

✓ Tests passed: 1 of 1 test – 7 ms

```
쥴리님, MiniGame 에 오신 것을 환영합니다 : )
```

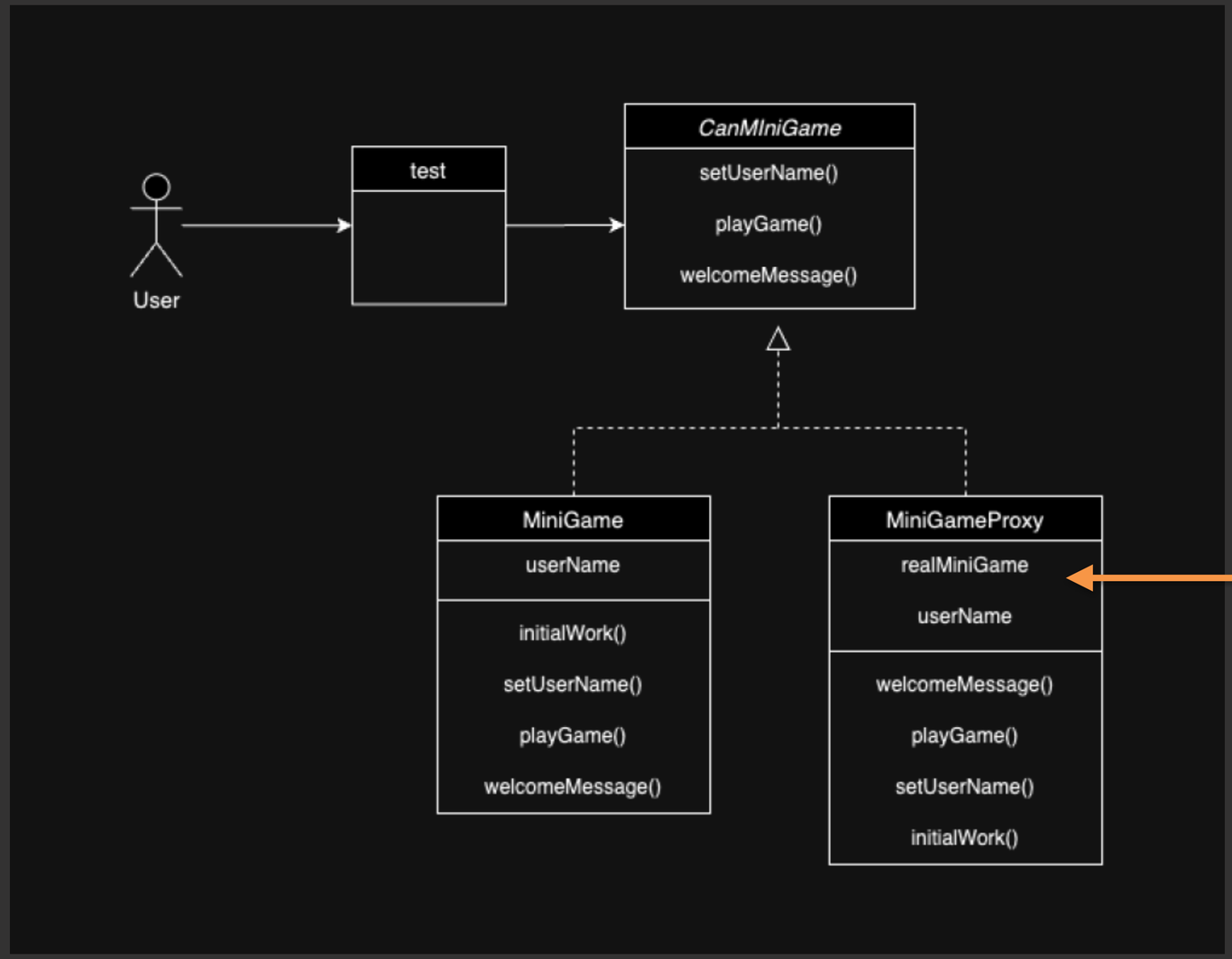
만약 RealSubject가 여러 개라면?

PuzzleMiniGame

BubbleMiniGame

MarioMiniGame

Proxy 패턴의 예제


User
test
CanMIniGame
setUserName()
playGame()
welcomeMessage()
MiniGame
userName
initialWork()
setUserName()
playGame()
welcomeMessage()
MiniGameProxy
realMiniGame
userName
welcomeMessage()
playGame()
setUserName()
initialWork()
MiniGame 타입

만약 RealSubject가 여러 개라면?

"프록시 객체가 항상 실제 객체의 존재를 알 필요는 없다"

만약 RealSubject가 여러 개라면?

```
public class MiniGameProxy implements CanMiniGame {  2 usages

//    private MiniGame realMiniGame;
    private CanMiniGame realMiniGame;  9 usages
    private String className; // Real MiniGame 객체의 클래스 이름
    private String userName;  4 usages
```

```
public MiniGameProxy(String className) {  2 usa
    this.realMiniGame = null; // 객체 생성을 지연
    this.className = className;
    this.userName = "null";
}
```

- ✅ 참조 타입을 CanMiniGame으로!
- ✅ className을 변수로 추가
- ✅ reflextion 기능을 사용해 인스턴스 생성

```
realMiniGame = (CanMiniGame) Class.forName( className: "com.example.demo.proxy."+ className).getDeclaredConstructor().newInstance();
```

만약 RealSubject가 여러 개라면?

```java
@Test  new *
void proxyTestWhenVariousMiniGame(){
    CanMiniGame miniGame = new MiniGameProxy( className: "PuzzleMiniGame");
    miniGame.setUserName("쥴리"); // 실제 객체는 생성되지 않았다.
    miniGame.welcomeMessage();
    miniGame.playGame();

    CanMiniGame miniGame2 = new MiniGameProxy( className: "BubbleMiniGame");
    miniGame2.setUserName("나띠"); // 실제 객체는 생성되지 않았다.
    miniGame2.welcomeMessage();
    miniGame2.playGame();

    CanMiniGame miniGame3 = new MiniGameProxy( className: "MarioMiniGame");
    miniGame3.setUserName("벨"); // 실제 객체는 생성되지 않았다.
    miniGame3.welcomeMessage();
    miniGame3.playGame();
}
```

```
✓ Tests passed: 1 of 1 test – 21 sec 537 ms

쥴리님, PuzzleMiniGame 에 오신 것을 환영합니다 : )
Puzzle 미니 게임 초기화 작업이 완료됐습니다.
Puzzle 미니 게임을 시작합니다.
나띠님, BubbleMiniGame 에 오신 것을 환영합니다 : )
Bubble 미니 게임 초기화 작업이 완료됐습니다.
Bubble 미니 게임을 시작합니다.
벨님, MarioMiniGame 에 오신 것을 환영합니다 : )
Mario 미니 게임 초기화 작업은 조금 더 오래 걸립니다. 조금만 기다려 주세요!
Mario 미니 게임 초기화 작업이 완료됐습니다.
Mario 미니 게임을 시작합니다.
```

Proxy 패턴 핵심 Point

프록시 객체는 항상 **실제 객체에 대한 참조를 유지**해야 한다.

실제 객체는 **프록시 객체의 존재를 몰라야 한다.**

프록시 객체가 **항상 실제 객체의 존재를 알 필요는 없다.**

프록시 객체는 간단한 기능을 제공을 통해, **실제 객체를 생성하기 전에**도 사용하는 해당 객체가 **이미 존재하는 것처럼 보이게 할 수 있다.**

References

- GoF의 디자인 패턴
- Java 언어로 배우는 디자인 패턴 입문