

프록시 패턴

이관우

kwlee@hansung.ac.kr

학습 목표

- 원격 프록시에 대한 이해
- 가상 프록시에 대한 이해
- 보호 프록시에 대한 이해

새로운 요구사항



주식회사 Gumball 뽑기 CEO

모든 뽑기 기계의 재
고와 현재 상태를 알
아낼 수 있도록 해
주세요

모니터링 용 코드 만들기

- GumballMachine 클래스에 현재 위치를 지원하기 위한 기능

```
public class GumballMachine {  
    //기타 인스턴스 변수  
    String location;  
  
    public GumballMachine(String location, int count) {  
        //기타 생성자 코드  
        this.location = location;  
    }  
  
    public String getLocation() {  
        return location;  
    }  
    //기타 메소드  
}
```

모니터링 용 코드 만들기

- 뽑기 기계의 위치, 재고, 현재 상태를 출력해 주는 GumballMonitor 클래스

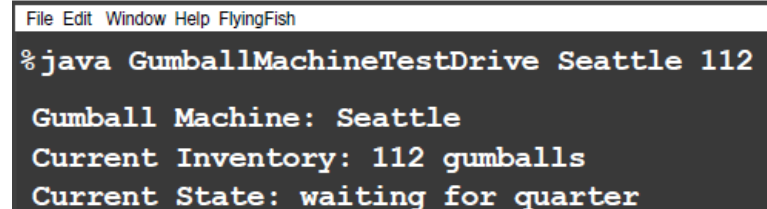
```
public class GumballMonitor {
    GumballMachine machine;

    public GumballMonitor(GumballMachine machine) {
        this.machine = machine;
    }

    public void report() {
        System.out.println("Gumball Machine: " + machine.getLocation());
        System.out.println("Current inventory: " + machine.getCount() + " 개");
        System.out.println("Current state: " + machine.getState());
    }
}
```

모니터링 용 코드 만들기

```
public class GumballMachineTestDrive {  
    public static void main(String[] args) {  
        int count = 0;  
  
        if (args.length < 2) {  
            System.out.println("GumballMachine <name> <inventory>");  
            System.exit(1);  
        }  
  
        try {  
            count = Integer.parseInt(args[1]);  
        } catch (Exception e) {  
            e.printStackTrace();  
            System.exit(1);  
        }  
  
        GumballMachine gumballMachine = new GumballMachine(args[0], count);  
  
        GumballMonitor monitor = new GumballMonitor(gumballMachine);  
        //기타 코드  
        monitor.report();  
    }  
}
```



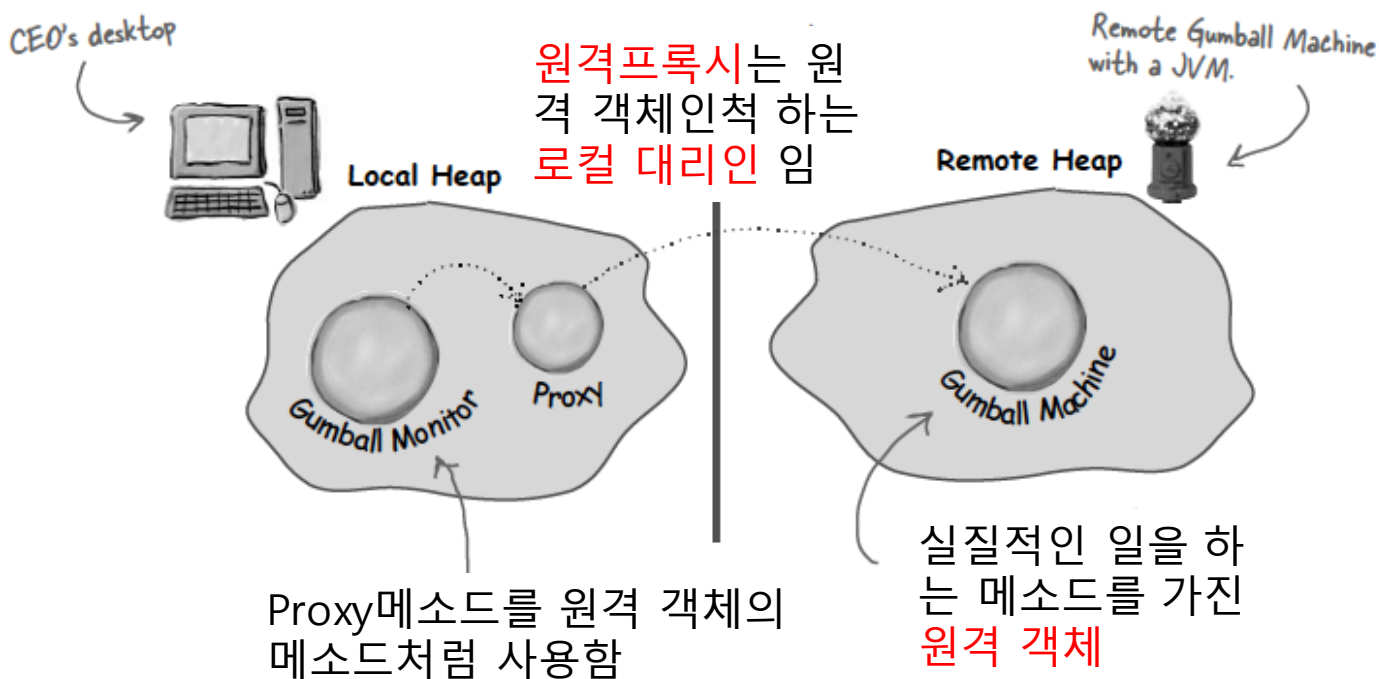
The screenshot shows a terminal window with a menu bar (File, Edit, Window, Help, FlyingFish). The command executed is `% java GumballMachineTestDrive Seattle 112`. The output is:
`Gumball Machine: Seattle`
`Current Inventory: 112 gumballs`
`Current State: waiting for quarter`

...



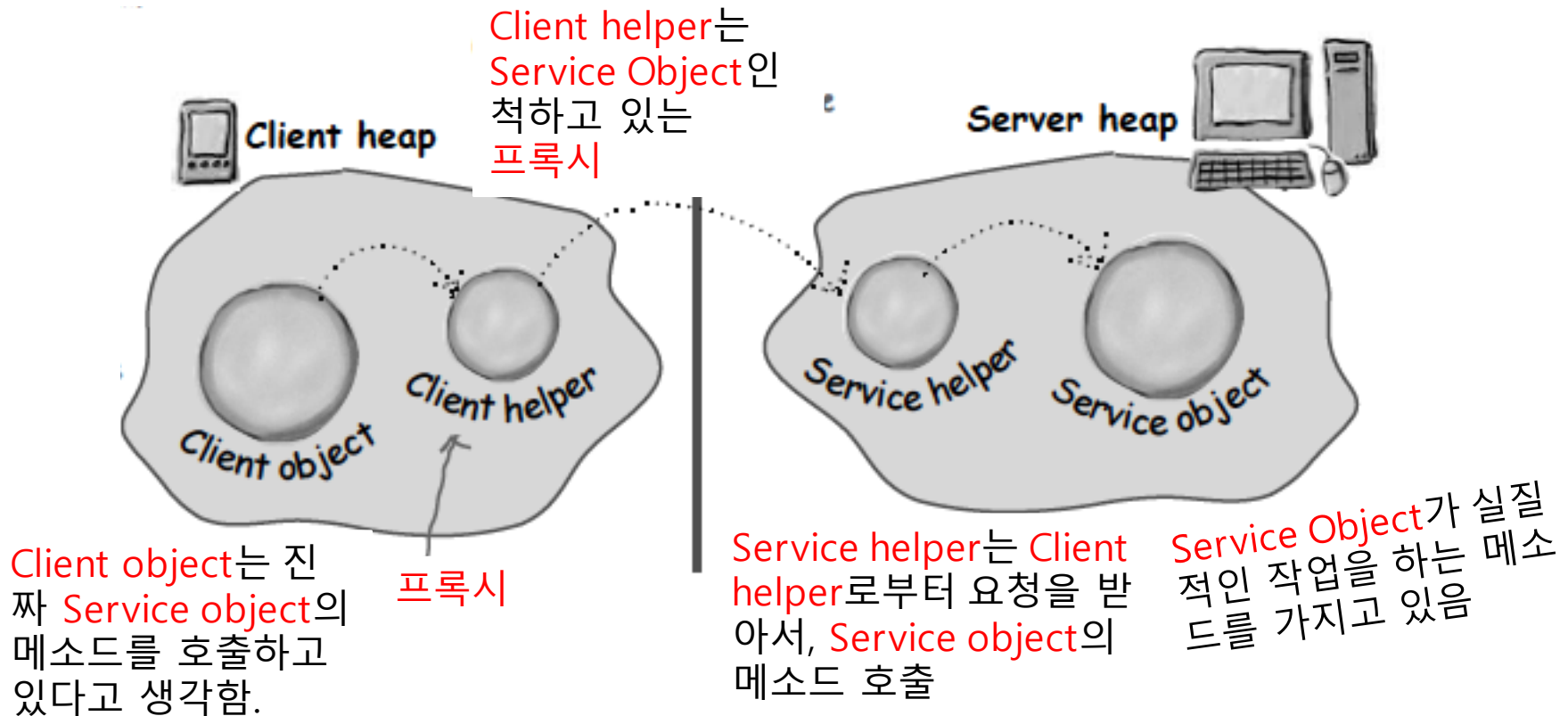
원격 프록시

- 다른 주소 공간(프로그램)에서 실행되고 있는 원격 객체의 메소드 호출이 가능하도록 만들어진 로컬 대리인
- 클라이언트로부터의 메소드 호출을 원격 객체한테 전달해 주기 위해서 네트워크 통신의 세부사항을 처리함



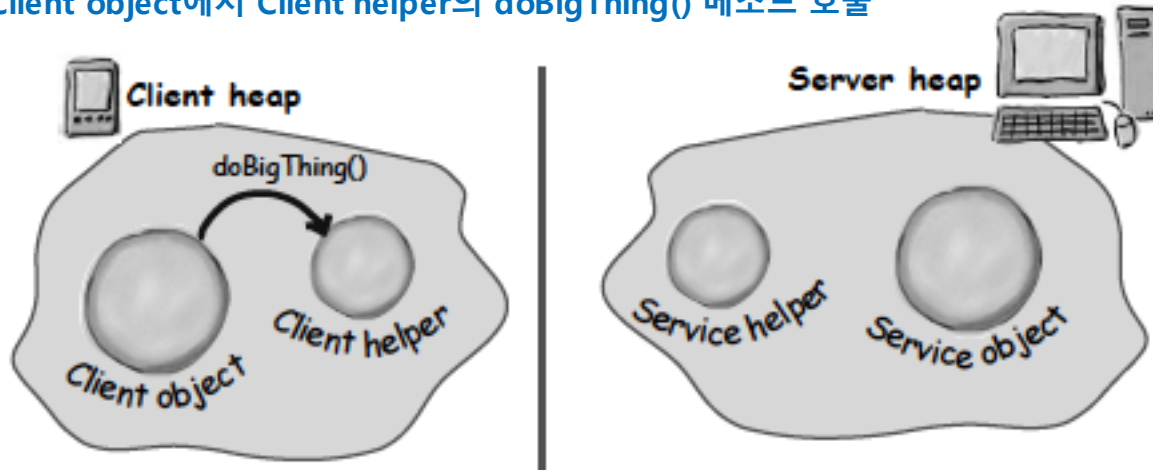
원격 메소드 호출 구현

원격 메소드 호출: Client objects가 원격에 있는 Service object의 메소드를 마치 자신의 로컬 객체인 것 처럼 호출하는 것을 의미

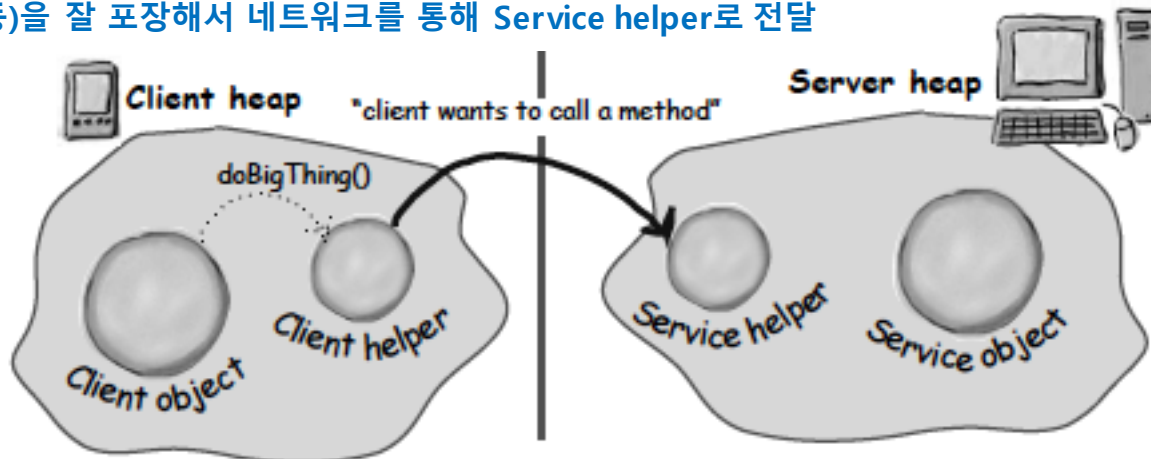


메소드 호출 과정

- 1 Client object에서 Client helper의 doBigThing() 메소드 호출

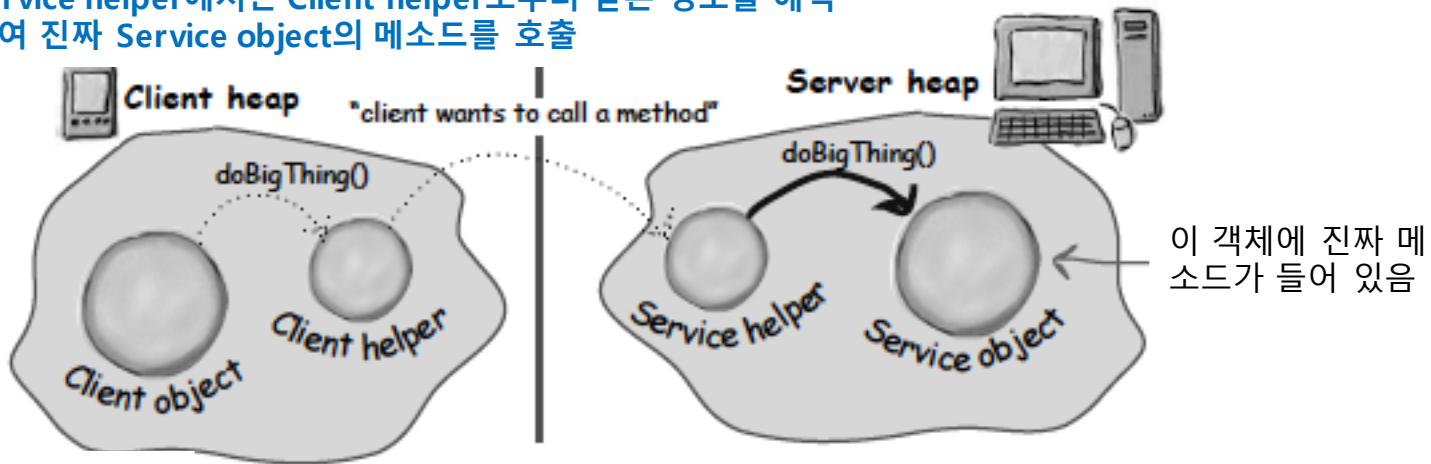


- 2 Client helper는 메소드 호출에 대한 정보(인자, 메소드 이름 등)를 잘 포장해서 네트워크를 통해 Service helper로 전달



메소드 호출 과정

- 3 Service helper에서는 Client helper로부터 받은 정보를 해석하여 진짜 Service object의 메소드를 호출

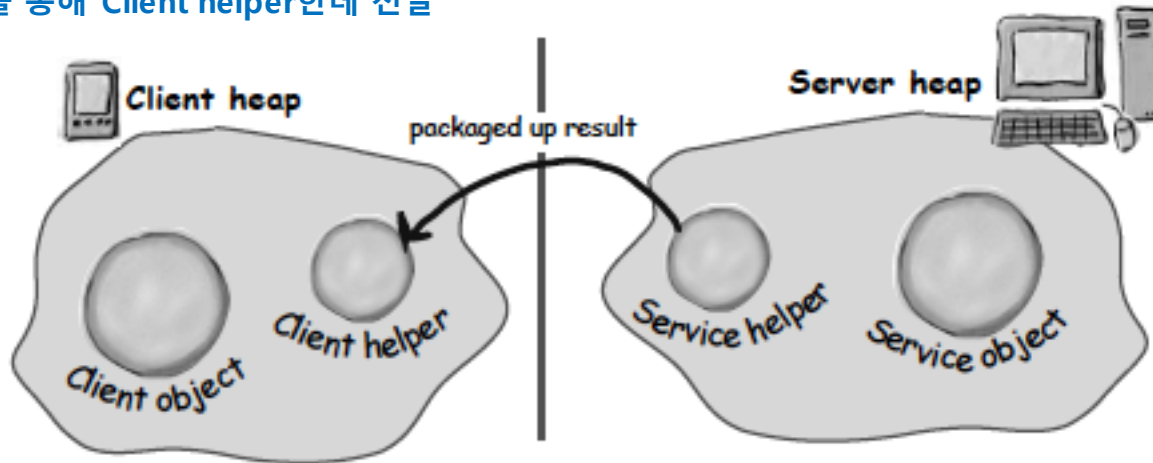


- 4 Service object의 메소드가 호출되고, 메소드 실행이 끝나면 Service helper에 어떤 결과가 리턴됨

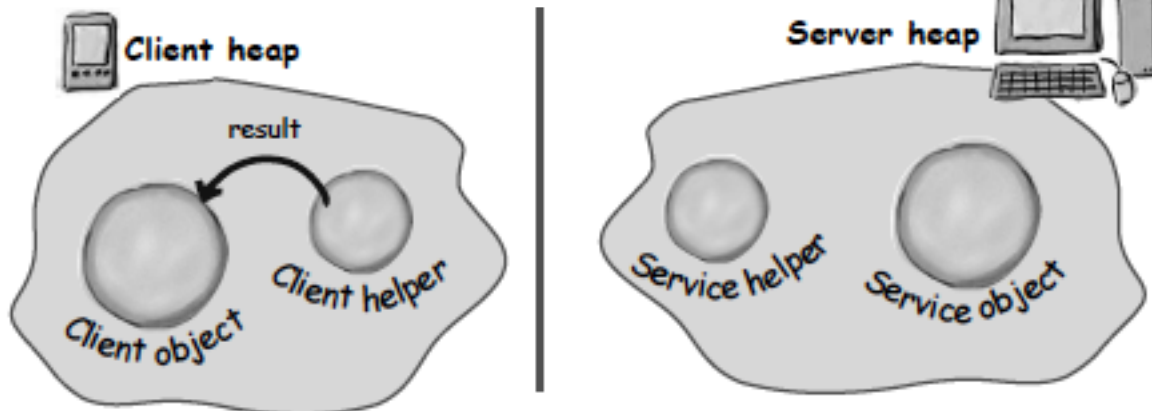


메소드 호출 과정

- 5 Service helper에서 호출 결과로 리턴된 정보를 포장해서 네트워크를 통해 Client helper한테 전달

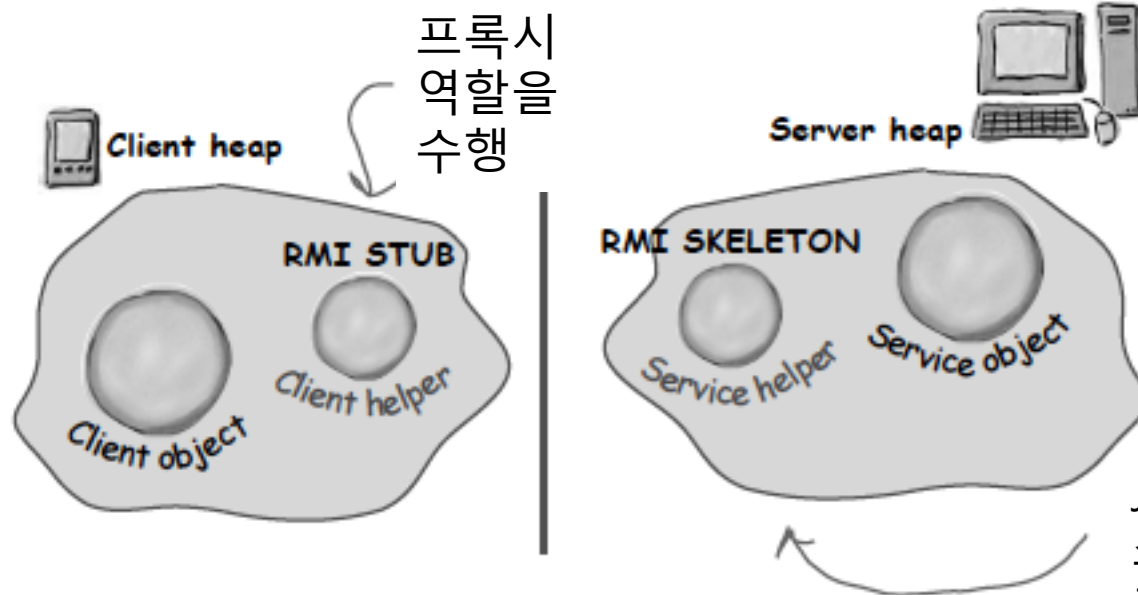


- 6 Client helper에서 리턴된 값을 해석하여 Client object한테 리턴합니다. Client object 입장에서는 메소드 호출이 어디로 전달되었었는지, 어디에서 왔는지 전혀 알 수 없습니다.



자바 RMI (Remote Method Invocation)

- RMI에서 Client helper는 **RMI STUB**, Service helper는 **RMI SKELETON**



Java 1.2부터는 스켈레톤 객체가 직접적으로 필요하지 않음

Java RMI로 원격 서비스 만들기

- **첫번째 단계:** 원격 인터페이스 만들기
 - 클라이언트에서 원격으로 호출할 수 있는 메소드 정의
- **두번째 단계:** 서비스 구현 클래스 만들기
 - 실제 작업을 처리하는 클래스
 - 원격 인터페이스를 실제 구현한 클래스
- **세번째 단계:** RMI 레지스트리 실행
- **네번째 단계:** 원격 서비스 시작
 - 서비스 인스턴스를 만들고, 이 인스턴스를 RMI 레지스트리에 등록

예제 프로젝트 링크

<https://github.com/kwanulee/PatternExample/tree/master/proxy/javarmi>

첫번째 단계: 원격 인터페이스 만들기

```
import java.rmi.Remote;  
import java.rmi.RemoteException;
```

```
public interface MyRemote extends Remote{
```

```
    public String sayHello() throws RemoteException;  
}
```

Java.rmi.Remote 확장



원격 메소드의 인자 및
리턴값은 반드시 원시
형식 또는 Serializable
형식으로 선언

모든 메소드를
RemoteException을 던
지는 메소드로 선언

두번째 단계: 서비스 구현 클래스 만들기

```
public class MyRemoteImpl extends UnicastRemoteObject  
                        implements MyRemote{
```

원격 객체 기능 상속

```
    protected MyRemoteImpl() throws RemoteException {}
```

```
    public String sayHello() throws RemoteException {  
        return "Server says, 'Hey'";  
    }
```

수퍼클래스 생성자에서 예외를 발생시킬 수 있으므로, 서브클래스 생성자에서도 추가

```
    public static void main(String[] args) {  
        try{  
            MyRemote service = new MyRemoteImpl();  
            Naming.rebind("RemoteHello", service);  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```

클라이언트는 등록된 이름을 바탕으로 레지스트리를 검색

RMI에서는 service에 해당하는 STUB를 레지스트리에 등록

세 번째, 네 번째 단계

- 세 번째 단계: RMI 레지스트리 실행

- 터미널을 띄워서 프로젝트 폴더(javarmi)의 bin 위치로 디렉토리 변경
- rmiregistry 실행

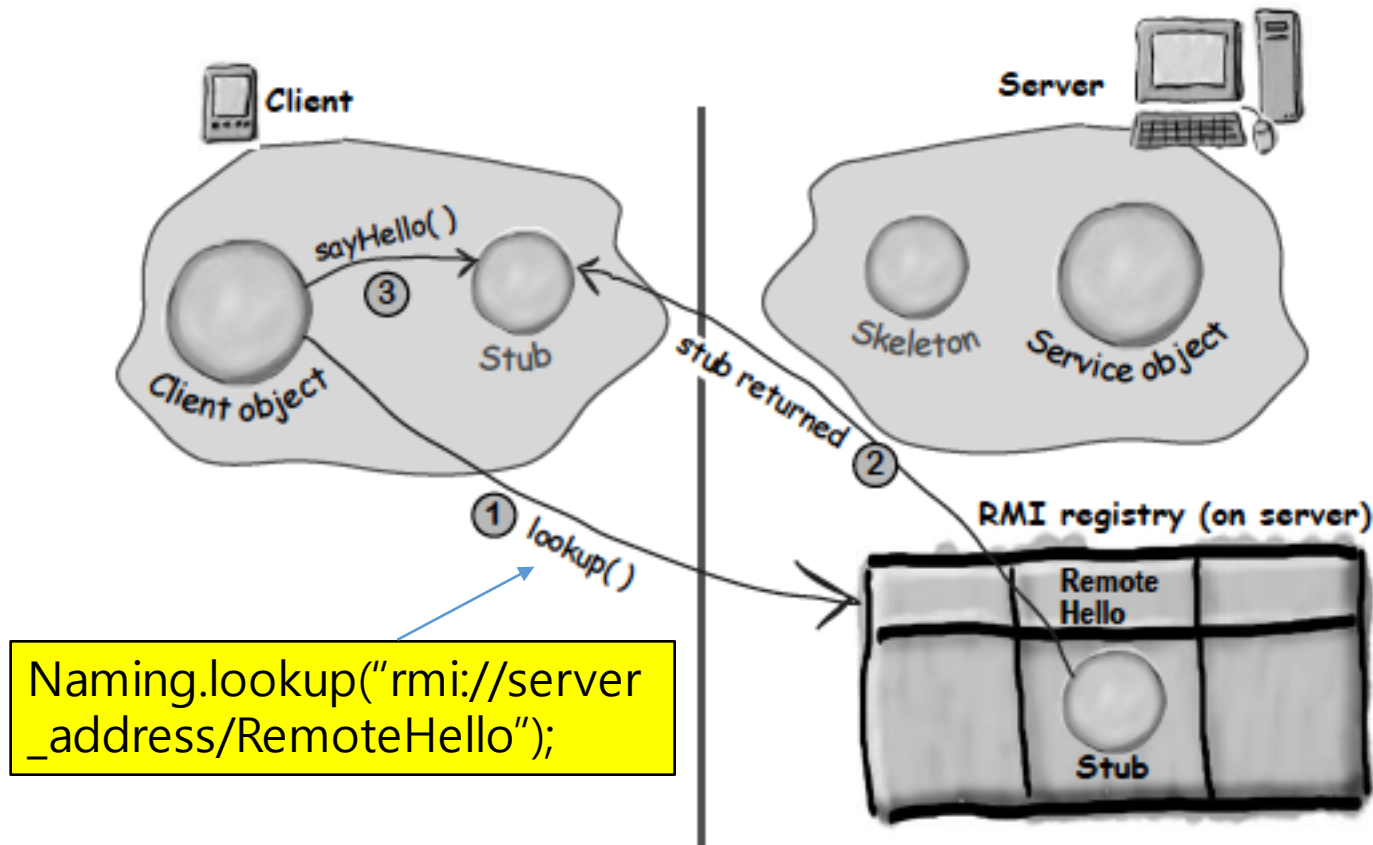
```
[(base) kwanwooui-MacBook-Air-2:bin kwanwoo$ pwd  
/Users/kwanwoo/Downloads/javarmi/bin  
[(base) kwanwooui-MacBook-Air-2:bin kwanwoo$ rmiregistry  
□
```

- 네 번째 단계: 원격 서비스 시작

- 다른 터미널을 열고 프로젝트 폴더의 bin 위치로 디렉토리 변경
- java hansung/designpatterns/proxy/MyRemoteImpl 실행

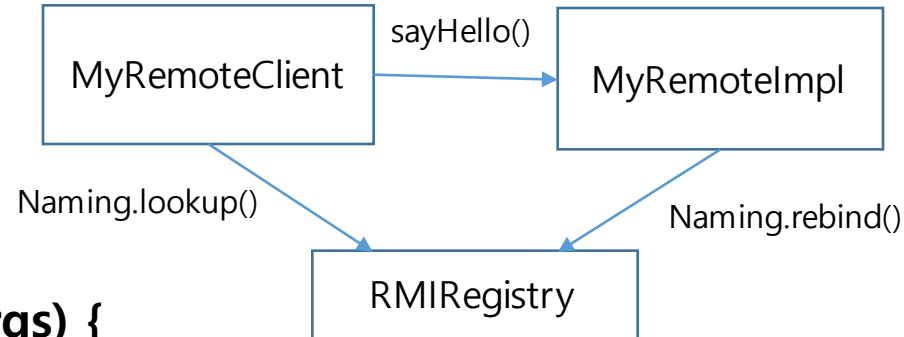
```
(base) kwanwooui-MacBook-Air-2:bin kwanwoo$ java hansung/designpatterns/proxy/MyRemoteImpl  
□
```

클라이언트에서는 어떻게 스텐브 객체를 가져올까요?



클라이언트 코드

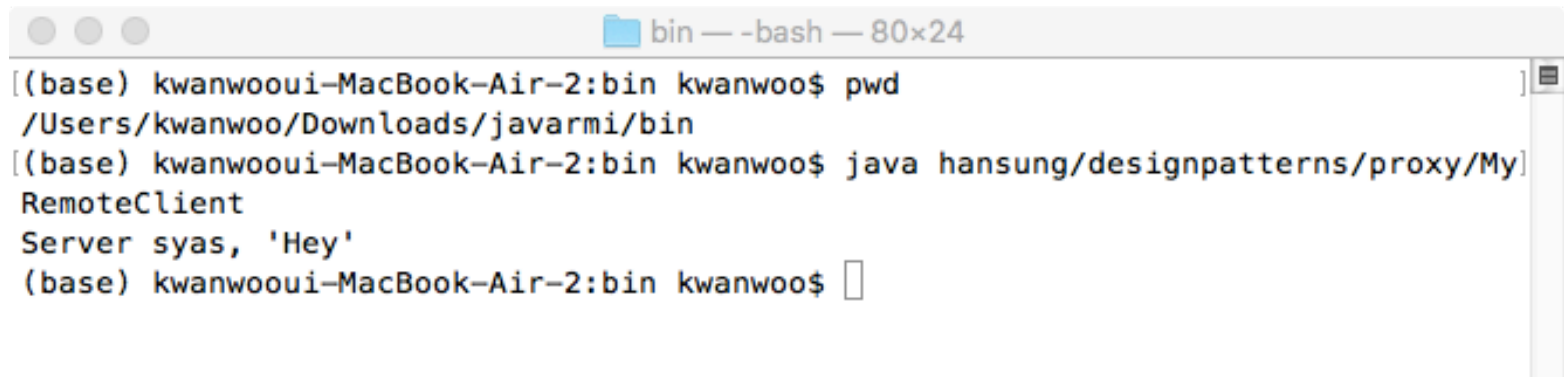
```
public class MyRemoteClient {  
    public static void main(String[] args) {  
        new MyRemoteClient().go();  
    }  
  
    public void go() {  
        try {  
            MyRemote service = (MyRemote)  
                Naming.lookup("rmi://localhost/RemoteHello");  
            String s = service.sayHello();  
            System.out.println(s);  
        } catch (Exception ex) {  
            ex.printStackTrace();  
        }  
    }  
}
```



서버 Ip 주소 및 호스트 이름,
여기서는 localhost 사용

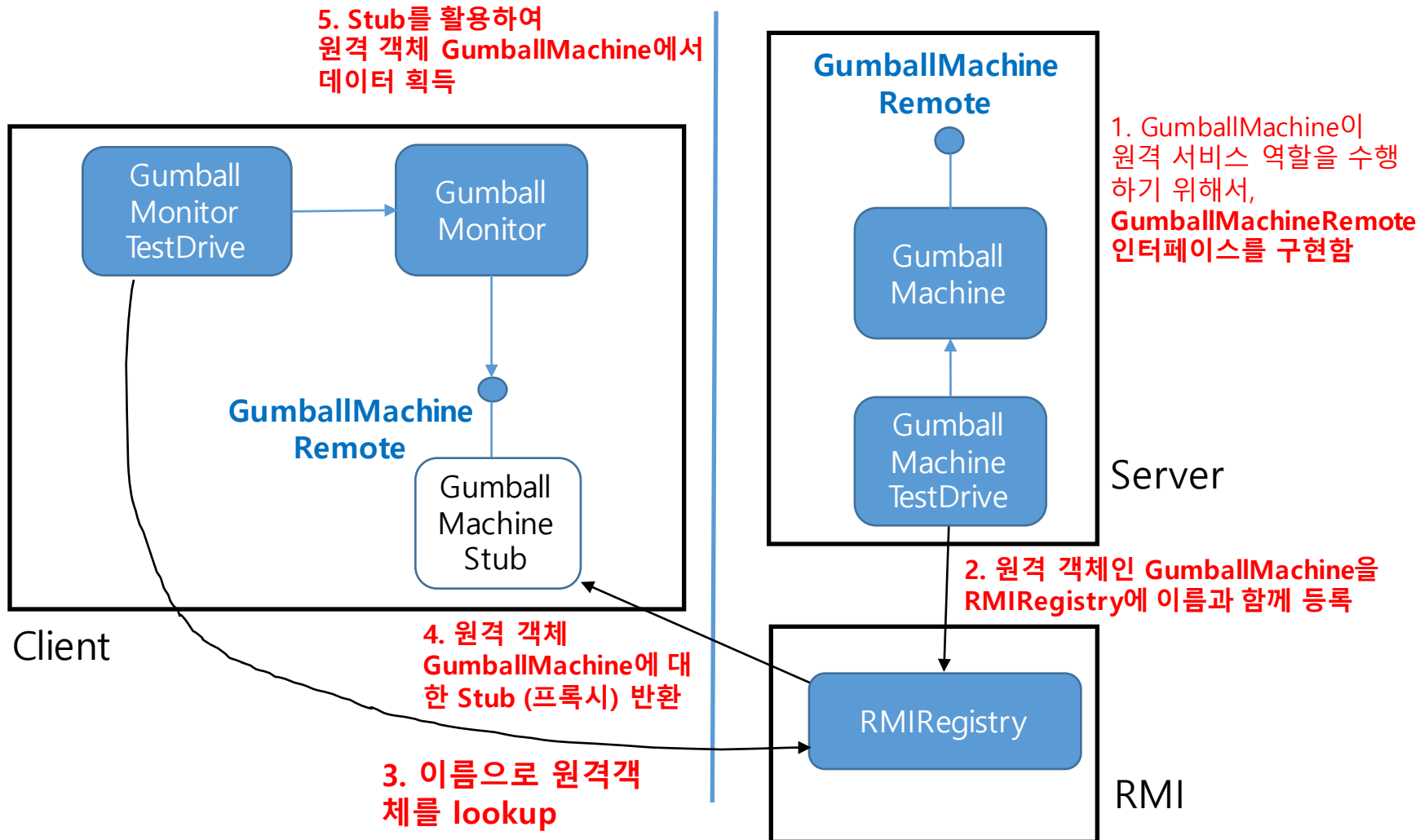
클라이언트 코드 실행

- 터미널을 새로 띄워서 프로젝트 폴더(javarmi)의 bin 위치로 디렉토리 변경
- *java hansung/designpatterns/proxy/MyRemoteClient* 실행



```
bin — -bash — 80x24
[(base) kwanwooui-MacBook-Air-2:bin kwanwoo$ pwd
/Users/kwanwoo/Downloads/javarmi/bin
[(base) kwanwooui-MacBook-Air-2:bin kwanwoo$ java hansung/designpatterns/proxy/MyRemoteClient
Server says, 'Hey'
(base) kwanwooui-MacBook-Air-2:bin kwanwoo$
```

GumballMachine의 원격 모니터링



GumballMachine 용 원격 인터페이스

```
import java.rmi.*;
```

```
public interface GumballMachineRemote extends Remote {  
    public int getCount() throws RemoteException;  
    public String getLocation() throws RemoteException;  
    public State getState() throws RemoteException;  
}
```

↑
네트워크를 통해 전송할 수 있도록 직렬화

```
public interface State extends Serializable {  
    public void insertQuarter();  
    public void ejectQuarter();  
    public void turnCrank();  
    public void dispense();  
}
```

GumballMachine 의 수정

```
public class GumballMachine
```

```
    extends UnicastRemoteObject  
    implements GumballMachineRemote
```

원격 객체 기능 상속

```
{  
    // 인스턴스 변수
```

원격 인터페이스 구현

```
    public GumballMachine(String location, int numberGumballs)  
        throws RemoteException {
```

```
        // 생성자 코드
```

```
    public int getCount() throws RemoteException { return count; }
```

```
    public State getState() throws RemoteException { return state; }
```

```
    public String getLocation() throws RemoteException {  
        return location; }
```

```
    // 기타 메소드
```

```
}
```

RMI 레지스트리 등록

```
public class GumballMachineTestDrive {  
  
    public static void main(String[] args) {  
        GumballMachineRemote gumballMachine = null;  
        int count;  
        ...  
        try {  
            count = Integer.parseInt(args[1]);  
            gumballMachine = new GumballMachine(args[0], count);  
            Naming.rebind("gumballmachine", gumballMachine);  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```

서버측 실행 절차

1. 터미널에서 "gumball\wbin" 으로 디렉토리 이동
2. "rmiregistry" 실행
3. 새로운 터미널에서 다음 명령 입력하여 실행
"java hansung/designpatterns/proxy/GumballMachineTestDrive localhost 300"

GumballMonitor 클라이언트

```
public class GumballMonitor {  
    GumballMachineRemote machine;
```

GumballMachine 원격 인터페이스 사용

```
    public GumballMonitor(GumballMachineRemote machine) {  
        this.machine = machine;  
    }  
  
    public void report() {  
        try {  
            System.out.println("Gumball Machine: " + machine.getLocation());  
            System.out.println("Current inventory: " + machine.getCount() + "  
                                gumballs");  
            System.out.println("Current state: " + machine.getState());  
        } catch (RemoteException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

메소드의 수행이 네트워크 오퍼레이션을 포함하므로, RemoteException 예외를 처리할 수 있어야 함

<https://github.com/kwanulee/PatternExample/tree/master/proxy/gumball>

GumballMonitor 테스트

```
public class GumballMonitorTestDrive {  
  
    public static void main(String[] args) {  
        String location;  
        if (args.length > 0){  
            location = "rmi://" + args[0] + "/gumballmachine";  
  
            GumballMonitor monitor=null;  
            try {  
                GumballMachineRemote machine =  
                    (GumballMachineRemote) Naming.lookup(location);  
                monitor = new GumballMonitor(machine);  
                System.out.println(monitor);  
            } catch (Exception e) {  
                e.printStackTrace();  
            }  
            monitor.report();  
        }  
    }  
}
```

모니터링할 위치

프록시, 스텝

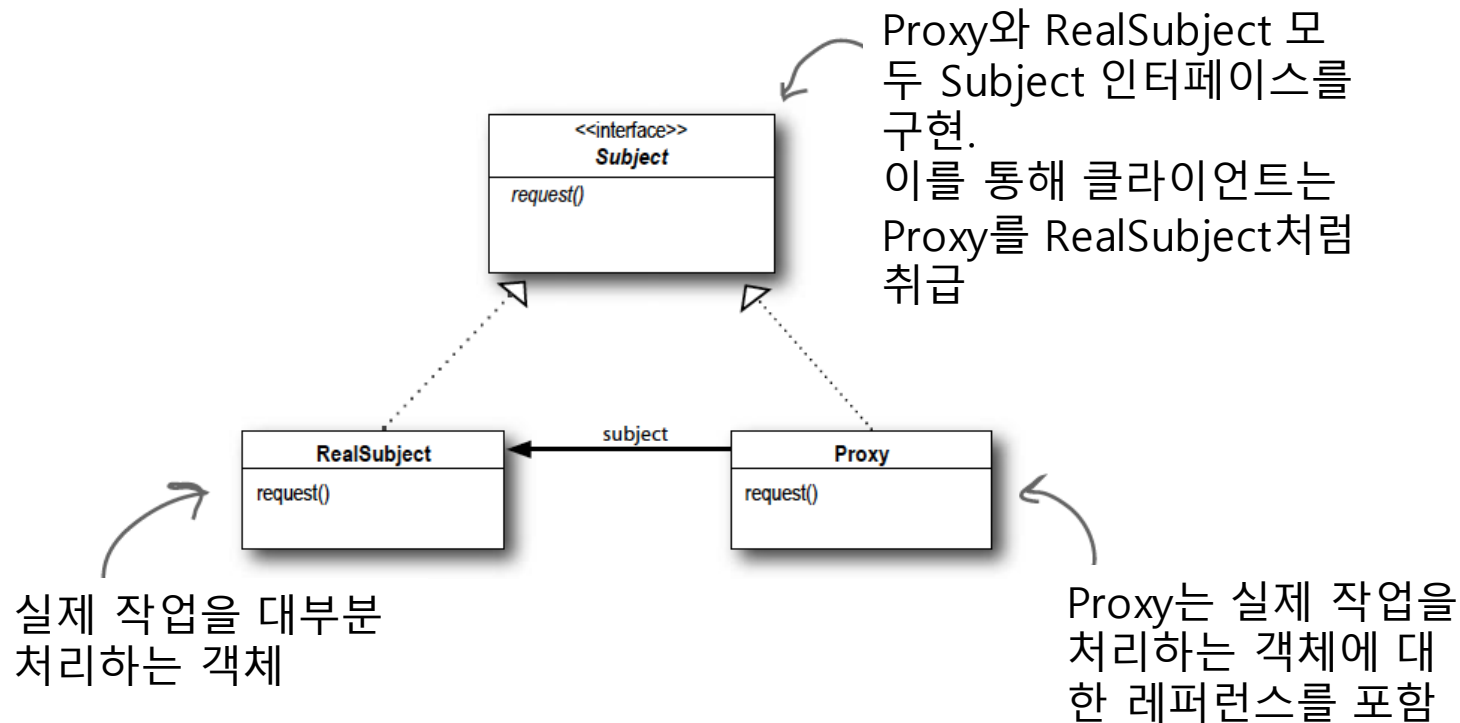
클라이언트

1. 새로운 터미널에서 "gumballwbin" 으로 디렉토리 이동
2. "java hansung/designpatterns/proxy/GumballMonitorTestDrive localhost" 실행

프록시 (Proxy) 패턴

- 정의

- 어떤 객체에 대한 접근을 제어하기 위한 용도로 대리인이나 대변인에 해당하는 객체를 제공하는 패턴



핵심 정리

- 프록시 패턴을 이용하면 어떤 객체에 대한 대변인을 내세워서 클라이언트의 접근을 제어할 수 있습니다.
- 원격 프록시는 클라이언트와 원격 객체 사이의 데이터 전달을 관리해 줍니다.