Step one: creating Invocation Handlers

We know we need to write two invocation handlers, one for the owner and one for the non-owner. But what are invocation handlers? Here’s the way to think about them: when a method call is made on the proxy, the proxy forwards that call to your invocation handler; but we do by calling the invocation handler’s corresponding method. So, what does it call? Have a look at the InvocationHandler interface:

There’s only one method, invoke(), and no matter what methods get called on the proxy, the invoke() method is what gets called on the handler. Let’s see how this works:

1. Let’s say the setHotOrNotRating() method is called on the proxy.

```java
proxy.setHotOrNotRating(9);
```

2. The proxy then turns around and calls invoke() on the InvocationHandler.

```java
invoke(Object proxy, Method method, Object[] args)
```

3. The handler decides what it should do with the request and possibly forwards it on to the RealSubject. How does the handler decide? We’ll find out next.

Creating Invocation Handlers continued...

When invoke() is called by the proxy, how do you know what to do with the call? Typically, you’ll examine the method that was called on the proxy and make decisions based on the method’s name and possibly its arguments. Let’s implement the OwnerInvocationHandler to see how this works:

```java
import java.lang.reflect.*;
public class OwnerInvocationHandler implements InvocationHandler {
    private PersonBean person;
    public OwnerInvocationHandler(PersonBean person) {
        this.person = person;
    }
    public Object invoke(Object proxy, Method method, Object[] args) throws IllegalAccessException {
        try {
            if (method.getName().startsWith("get")) {
                return method.invoke(person, args);
            } else if (method.getName().equals("setHotOrNotRating")) {
                throw new IllegalAccessException();
            } else if (method.getName().startsWith("set")) {
                return method.invoke(person, args);
            } else {
                MethodInvocationException a = (MethodInvocationException) e.printStackTrace();
                return null;
            }
        } catch (IllegalAccessException e) {
            throw e;
        } catch (InvocationTargetException e) {
            throw e;
        }
    }
}
```

If any other method is called, we’re just going to return null rather than fake a chance.