

# Reflection: Proxies

---

**CPSC 501: Advanced Programming Techniques  
Fall 2020**

Jonathan Hudson, Ph.D  
Instructor  
Department of Computer Science  
University of Calgary

Wednesday, August 5, 2020



**UNIVERSITY OF  
CALGARY**

# Intercession via proxy

---

# Dynamic Proxies

---

- A **proxy** is “a person authorized to act on behalf of another”
- In OO programming, a **proxy** is an object that substitutes for another object called the **target**
  - To work, the proxy must implement the target’s entire interface
    - i.e. support all the target’s methods

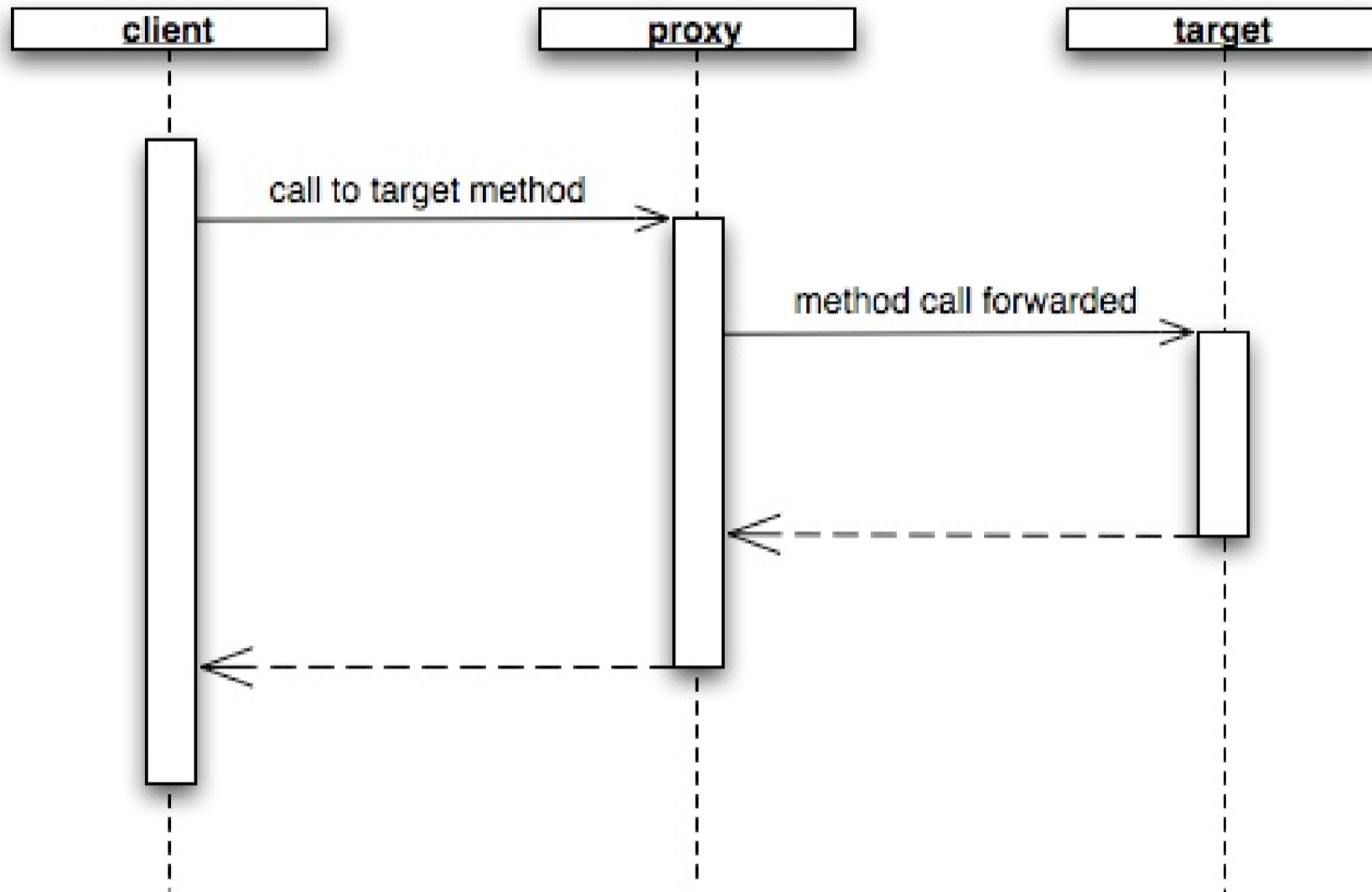
# Dynamic Proxies

---

- The proxy may
  1. Implement its own versions of ***all*** the methods
    - Acts as a ***substitute*** for the target
  2. Or delegate some or all calls it receives to the target
    - Acts as an ***intermediary*** between the caller and the target

# Dynamic Proxies

---

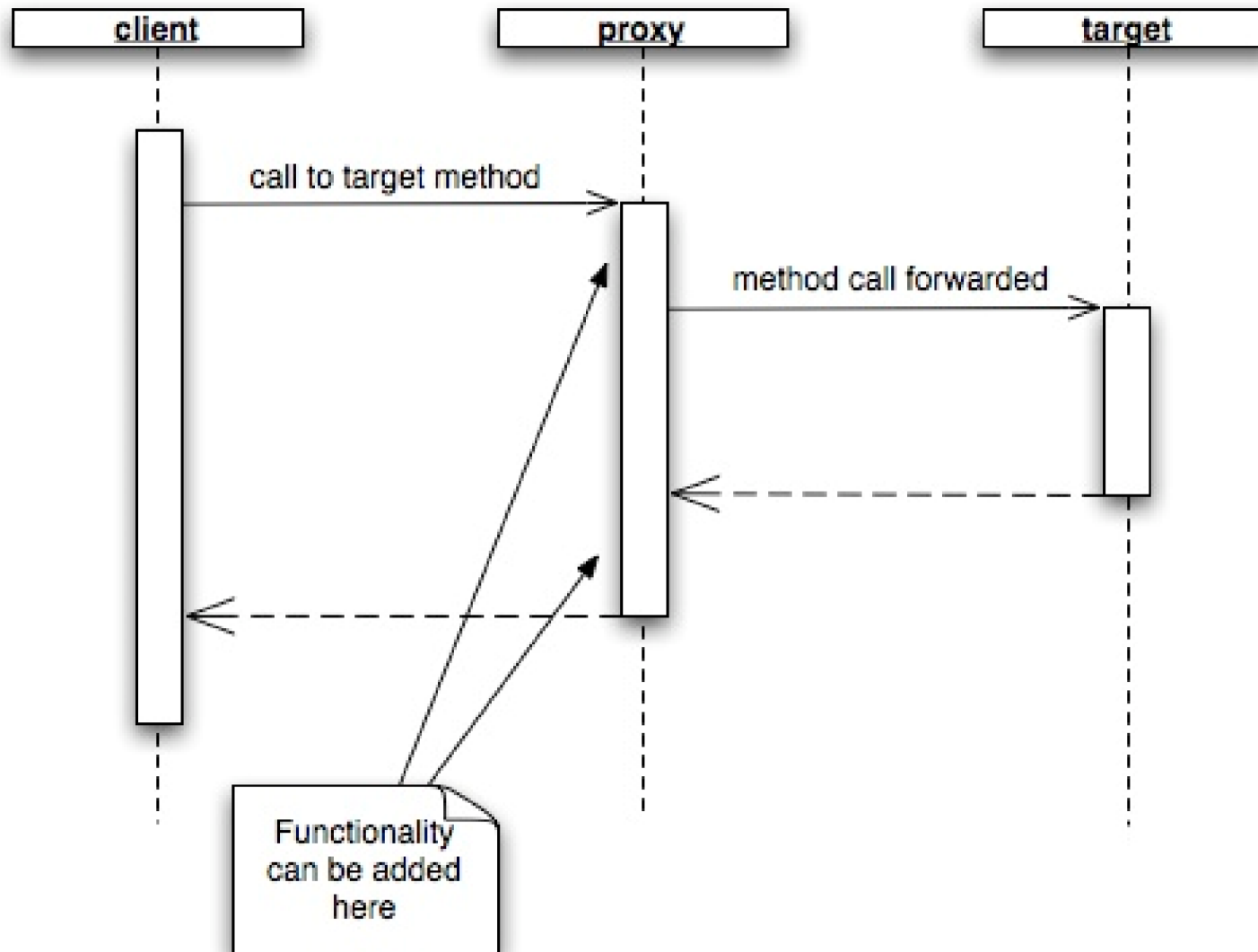


# Dynamic Proxies

---

- If an intermediary, a proxy may add functionality before and after forwarding the method to the target
  - Allows you to add behavior to objects

# Dynamic Proxies



# Dynamic Proxies

---

- Method invocation **intercession** is the ability to intercept method calls reflectively
  - Not supported by Java directly
    - (Java Aspects one way of doing intercession)
  - But is approximated using **dynamic proxies**
    - Proxies can be created at runtime as needed



# Dynamic Proxies

---

- Proxies are useful when you need to add similar behavior to many different classes
  - E.g. Adding tracing code to all method calls
- Without proxies, you could create non-tracing and tracing classes
  - Or subclasses of a common superclass
  - You arrange to instantiate the needed class at runtime

# Dynamic Proxies

---

- Without proxies, you could create non-tracing and tracing classes
  - **Drawbacks:**
    - **Tedious: must be done for all methods of all classes**
    - **Error prone: may overlook a method**
    - **Fragile: if a method is added or changed, then the traced class must also be added/changed**
- Dynamic proxies allows the added behavior to be implemented in a single class

# Intercession via Java proxy

---

# Dynamic Proxies

---

- `java.lang.reflect.Proxy`
  - Can create a proxy class (a class object) that implements a set of proxied interfaces with:  
**`static Class getProxyClass(ClassLoader loader, Class[] interfaces)`**

# Dynamic Proxies

---

- Proxy classes have a constructor that takes an InvocationHandler parameter
- You create a proxy instance with the constructor and newInstance()
- E.g.

```
Proxy cl = Proxy.getProxyClass(MyInterface.getClassLoader(),
                               Class[] {MyInterface.class});
Constructor c = cl.getConstructor(new Class[] {InvocationHandler.class});
Object proxy = c.newInstance(new Object[] {new MyIH(target)});
```

# Dynamic Proxies

---

- Or with a single call:

```
Object proxy = Proxy.newProxyInstance(MyInterface.getClassLoader(),  
                                     new Class[]{MyInterface.class},  
                                     new MyIH(target));
```

- Use **isProxyClass()** to see if a class object represents a proxy class
  - E.g.  
**if (Proxy.isProxyClass(obj.getClass()))**

...

# Dynamic Proxies

---

- The proxy instance delegates handling of methods to its invocation handler
  - Is an object that implements the **InvocationHandler** interface
    - Must implement the **invoke()** method
    - Must also keep a reference to the target object
    - E.g.

# Dynamic Proxies

---

```
public class MyIH implements InvocationHandler {  
  
    private Object target;  
  
    public MyIH(Object obj) {  
        target = obj;  
    }  
  
    public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {  
        return method.invoke(target, args);  
    }  
}
```



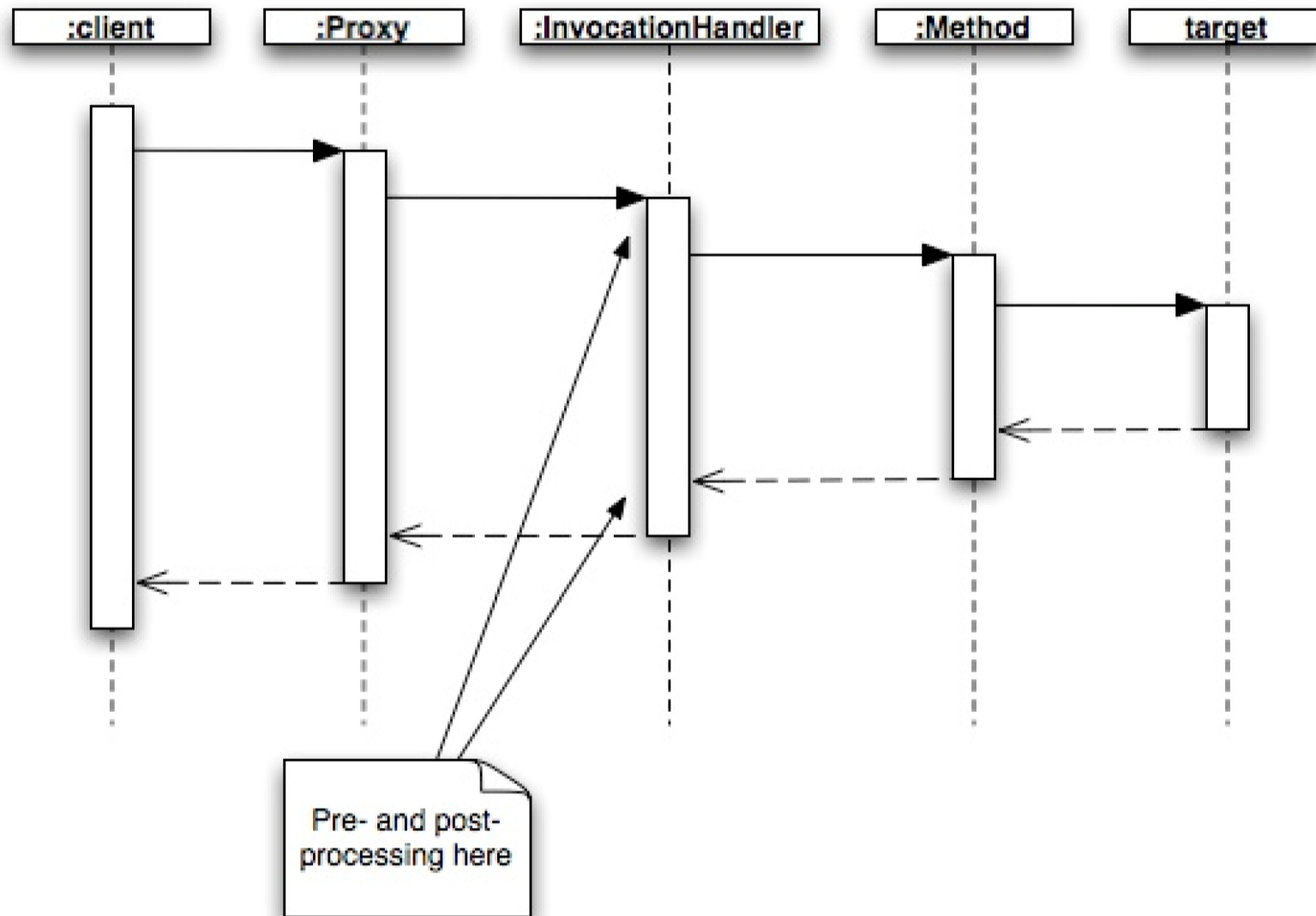
# Dynamic Proxies

---

- The above **invoke()** method merely forwards the message to the target
  - Could add pre- and post-processing
  - E.g.

```
public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {  
    Object result = null;  
    // preprocessing here  
    result = method.invoke(target, args);  
    // postprocessing here  
    return result;  
}
```

# Dynamic Proxies



# Example

---

# Example – MyInterface.java

---

- Example: tracing proxy
  - Target interface

```
public interface MyInterface {  
    public void print();  
    public void display();  
}
```

# Example – MyClass.java

---

- Target class

```
public class MyClass implements MyInterface {  
  
    public void print() {  
        System.out.println("Hello, world!");  
    }  
  
    public void display() {  
        System.out.println("Goodbye, cruel world!");  
    }  
  
}
```

# Example – TracingIH.java

---

- Invocation handler

```
public class TracingIH implements InvocationHandler {  
  
    public static Object createProxy(Object obj) {  
        return Proxy.newProxyInstance(  
            obj.getClass().getClassLoader(),  
            obj.getClass().getInterfaces(),  
            new TracingIH(obj));  
    }  
  
    private Object target;  
  
    private TracingIH(Object obj) {  
        target = obj;  
    }  
}
```

## Example – TracingIH.java (cont'd)

---

```
public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
    Object result = null;
    try {
        System.out.println(method.getName() + "() begins");
        result = method.invoke(target, args);
    } catch (InvocationTargetException e) {
        System.out.println(method.getName() + " throws " + e.getCause());
        throw e.getCause();
    }
    System.out.println(method.getName() + "() returns\n");
    return result;
}
}
```

# Example – Test.java

---

- Client code

```
public class Test {  
  
    public static void main(String[] args) {  
        MyInterface obj = new MyClass();  
        if (args.length > 0 && args[0].equals("trace")) {  
            obj = (MyInterface) TracingIH.createProxy(obj);  
        }  
        obj.print();  
        obj.display();  
    }  
}
```



# Example – no proxy

---

- Sample run: java Test

```
run:
```

```
Hello, world!
```

```
Goodbye, cruel world!
```

```
BUILD SUCCESSFUL (total time: 0 seconds)
```

# Example – proxy enabled

---

- Sample run: java Test trace

```
run:
```

```
print() begins
```

```
Hello, world!
```

```
print() returns
```

```
display() begins
```

```
Goodbye, cruel world!
```

```
display() returns
```

```
BUILD SUCCESSFUL (total time: 0 seconds)
```

# Readings

---

- Forman & Forman Chapter 4
- Java API: `java.lang.reflect.Proxy`

# Onward to ... Java Aspects.

---

Jonathan Hudson  
[jwhudson@ucalgary.ca](mailto:jwhudson@ucalgary.ca)  
<https://pages.cpsc.ucalgary.ca/~hudsonj/>



UNIVERSITY OF  
CALGARY