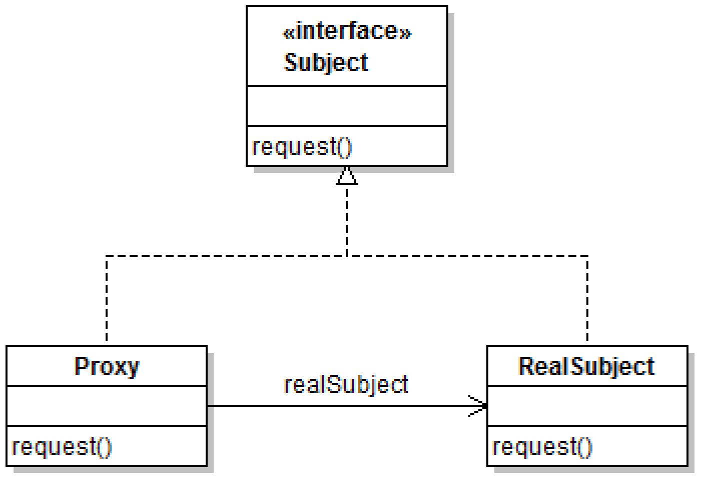
Proxy Pattern

- **Intent**
  - Provide a surrogate or placeholder for another object to control access

- **Structure**



Proxy Pattern

- **Applications**

Remote Proxy	(remote communication)
Virtual Proxy	(creation on demand / lazy evaluation)
CopyOnWrite Proxy	(makes copies only if the copy is modified, i.e. delayed cloning)
Protection Proxy	(access control)
Cache Proxy	(caches already returned results)
Synchronization Proxy	(provides synchronization)
Smart Pointer Proxy	(controls memory management)

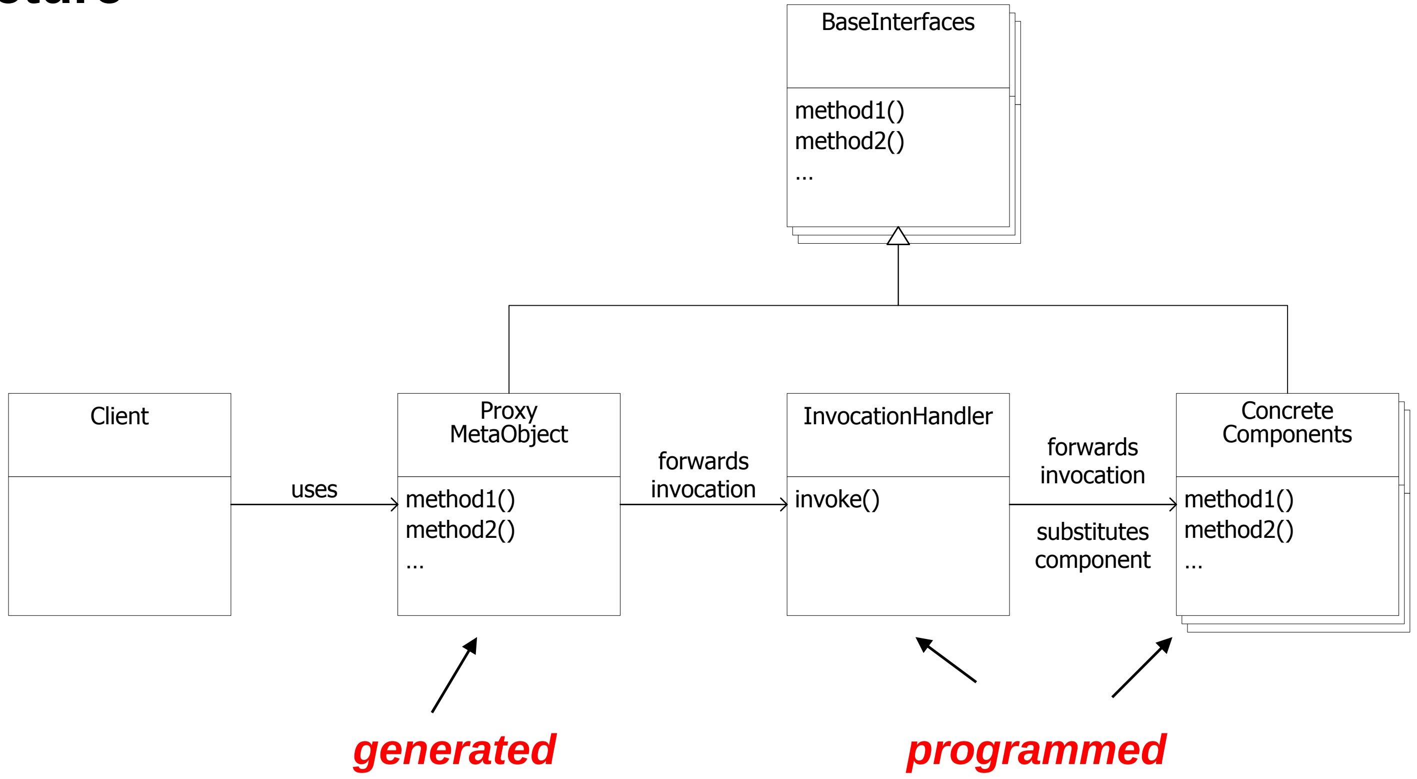
- **Proxy vs Decorator**
  - Proxy is not recursive
  - Proxy controls access, whereas decorator adds responsibilities
  - Proxy implements the same interface as the real object (and does not add additional methods)

Dynamic Proxy Class

- **Java Dynamic Proxy Classes**
  - Class that implements a list of interfaces
    - Created at runtime (using `java.lang.reflect.Proxy`)
    - Interfaces to be implemented are specified at runtime
  - Each proxy instance has an associated invocation handler object
    - Responsible to execute the methods
    - A method invocation on a proxy instance through one of its proxy interfaces will be dispatched to the invoke() method of the instance's invocation handler

Dynamic Proxy Class

- **Structure**



Dynamic Proxy Class

- **Participants**
  - Proxy interface
    - An interface that is implemented by a proxy class
  - Dynamic proxy
    - Instance of a dynamic proxy class
    - Dynamic proxy class implements a list of interfaces specified at runtime when dynamic proxy instance is generated
  - Invocation Handler Object
    - Each proxy instance has an associated invocation handler object which implements the interface *InvocationHandler*
    - A method invocation on a proxy instance trough one of its proxy interfaces will be dispatched to the invoke() method of the instance's invocation handler
  - Concrete Object
    - Object which is used in invocation handler

Dynamic Proxy Class

- **Invocation Handler**

```
public interface InvocationHandler {
 /*
  * @param proxy  the proxy instance on that the method was invoked
  * @param method the Method instance corresponding to the
  *              interface method invoked on the proxy instance.
  * @param args   an array of objects containing the values of the
  *              arguments passed in the method invocation on the
  *              proxy  (or null)
  */
  public Object invoke(Object proxy, Method method, Object[] args)
        throws Throwable;
}
```

Dynamic Proxy Class: Example

- **Example: Logger**

```
public class LoggingHandler implements InvocationHandler {
   private Object target;
   public LoggingHandler(Object t) { this.target=t; }

   public Object invoke(Object proxy, Method m, Object[] args)
                                              throws Throwable {
      System.out.println(">> "+m.getName());
      return m.invoke(target, args);
   }
}
```

- **Creation**

```
obj = (AnyInterface)Proxy.newProxyInstance(
                AnyInterface.class.getClassLoader(),
                new Class[] { AnyInterface.class },
                new LoggingHandler(target));
```

Dynamic Proxy Class

- **Can be used with ANY interface**

```
String s = "Hello"; // String implements CharSequence
CharSequence obj = (CharSequence) Proxy.newProxyInstance (
   String.class.getClassLoader(),
   new Class[] { CharSequence.class },
   new LoggingHandler(s)
);
obj.length();
obj.charAt(0);
```

```
>> length()
>> charAt(0)
```

```
Collection c = (Collection)Proxy.newProxyInstance (
   Collection.class.getClassLoader(),
   new Class[] { Collection.class },
   new LoggingHandler(new HashSet())
);
c.size();
c.add("text");
```

```
>> size()
>> add(text)
```