# Proxy Examples::

MemoProxy.java

```
/*
 *
 * @author guthrie
*
/
package proxy.memo;

import java.util.Collections;
import java.util.HashMap;
import java.util.Map;

public class MemoProxy implements Application {
	Application app;
	Map<Integer,Integer> cache =
		Collections.synchronizedMap(new HashMap<Integer,Integer>());    // <T, R>

	MemoProxy ( Application ap) {
		app = ap;
	}

	//  (pre/delegate/post) all mingled, not functor-ready!
	public int compute(int x) {
		Integer value = cache.get(x);      // see if this argument seen before;
		if (value==null) {
			System.out.println("Compute(" + x + ")");
			value = app.compute(x);
			cache.put(x,value);
		}
		return value;
	}
	// ------------------------------------------------
	public static void main(String[] args) {
		// First; direct usage...
		Application ap = new Applic();
		System.out.println(ap.compute(2));

		// now memoized...
		ap = new MemoProxy(ap);
		System.out.println("Now Optimized...");
		System.out.println(ap.compute(2));		// $$$ = same!  (P2I)
		System.out.println(ap.compute(3));		// $$$
		System.out.println(ap.compute(3));		// $0 !
	}
}

// ================================================
interface Application {
	int compute( int x);			// <T, R>
}

class Applic implements Application {
	public int compute ( int x ) {
```

```
                return (2*x);
        }
}
```

## GenMemoProxy.java

```
/*
 *
 * @author guthrie
 * Memoization proxy, w/ Generics
*
*  Combines Generics with Proxy – creates a very general memoization capability
 */
package proxy.memo.v1;

import java.util.Collections;
import java.util.HashMap;
import java.util.Map;

public class GenMemoProxy<R,A> implements Application<R,A> {
	Application<R,A> app;
	Map<A,R> cache =
		Collections.synchronizedMap(new HashMap<A,R>());

	GenMemoProxy ( Application<R,A> ap) {
		app = ap;
	}

	public R compute(A x) {
		R value = cache.get(x);  // see if this argument seen before;
		if (value==null) {
			System.out.println("Compute(" + x + ")");
			value = app.compute(x);
			cache.put(x,value);
		}
		return value;
	}
	// ---------------------------------------------------
	public static void main(String[] args) {
		// First; direct usage...
		Application<Integer,Integer> ap = new Applic();
		System.out.println(ap.compute(2));

		// now memoized...
		ap = new GenMemoProxy<Integer,Integer>(ap);
		System.out.println("Now Optimized...");
		System.out.println(ap.compute(2));
		System.out.println(ap.compute(3));
		System.out.println(ap.compute(3));

		// Another function...
		//     now memoized...
		Application<Integer,String> sap = new StringApplic();
		sap = new GenMemoProxy<Integer,String>(sap);
		System.out.println("Now Optimized...");
		System.out.println(sap.compute("Hello"));
		System.out.println(sap.compute("World"));
		System.out.println(sap.compute("World"));
	}
}
```

```
// ================================================
//          <Return, Argument>
interface Application<R,A> {
        R compute( A x);
}

class Applic implements Application<Integer,Integer> {
        public Integer compute ( Integer x ) {
                return (2*x);
        }
}

class StringApplic implements Application<Integer,String> {
        public Integer compute ( String s ) {
                return (s.length());
        }

}
```

## DynProxy.java

```
/*
*
 * From: Sun, Reflection, Dynamic Proxy Classes
 *
 * http://java.sun.com/j2se/1.3/docs/guide/reflection/proxy.html
*
 */
package proxy;
import java.lang.reflect.*;

class myException extends Exception {}

interface Foo {
        void bar(Object obj); // throws myException
}

class FooImpl implements Foo {
        public void bar(Object obj) { // throws myException
                // ...
        System.out.println("  - Inside method ");
        }
}

// ------------------------------------------------------
public class dynProxy
   implements java.lang.reflect.InvocationHandler {

        private Object obj;

        public static <T> T newInstance(T obj) {
        return (T) java.lang.reflect.Proxy.newProxyInstance(
                obj.getClass().getClassLoader(),
                obj.getClass().getInterfaces(),
                new dynProxy(obj));
        }

        private dynProxy(Object obj) {
        this.obj = obj;
        }

        public Object invoke(Object proxy, Method m, Object[] args)
        throws Throwable
        {
                Object result;
        try {
                System.out.println("before method " + m.getName());
                result = m.invoke(obj, args);
                } catch (InvocationTargetException e) {
                throw e.getTargetException();
                } catch (Exception e) {
                throw new RuntimeException("unexpected invocation exception: " +
                                        e.getMessage());
        } finally {
                System.out.println("after method " + m.getName());
```

```
        }
        return result;
        }

        // ----------------------------------------------------

        public static void main(String args[]) {
                // Old ...
                System.out.println("Original:: ");
                Foo foo = new FooImpl();
                foo.bar(null);

                // New ...
                System.out.println("\nProxied:: ");
                foo =  dynProxy.newInstance(foo);
                foo.bar(null);
        }
}
```

## DynGenProxy.java

```
/*
 *  Creating a proxy using JDK tools...
 *  * From:
 *    http://www.javaworld.com/javaworld/
 *          jw-02-2002/jw-0222-designpatterns_p.html
 *
 *    GRG: Modified to be generalized over a functor.
 */

package proxy.DynProxy;

import java.lang.reflect.InvocationHandler;
import java.lang.reflect.Method;
import java.lang.reflect.Proxy;

//-----------------------------------------------------------
interface Thing {
   public void doSomething();
}

class aThing implements Thing {
   public void doSomething() {
            System.out.println("Inside Method aThing.doSomething()");
   }
}
// -----------------------------------------------------------
interface Functor<T> {
           void pre (T arg);
           void post(T arg);
}

class myFunctor implements Functor<String> {
           public void pre (String s) {
                    System.out.println("Before Calling " + s);
           }
           public void post(String s) {
                    System.out.println("After  Calling "  + s);
           }
}

//-----------------------------------------------------------
class myInvocationHandler implements InvocationHandler {
           private Object target = null;
           Functor<String> f;

    public myInvocationHandler(Object s, Functor<String> fun) {
      target = s;
            f = fun;
   }
   public Object invoke(Object proxy, Method m, Object[] args){
            Object result = null;

            f.pre(m.getName());
```

```
            try {
                    result = m.invoke(target, args);
            }
            catch(Exception ex) {
                    System.exit(1);
            }

            f.post(m.getName());

            return result;
    }
}

class dynProxy {
        static Object makeProxy (Thing t, Functor<String> f ) {
                return Proxy.newProxyInstance(
                                t.getClass().getClassLoader(),
                                t.getClass().getInterfaces(),
                                new myInvocationHandler(t,f));
        }
}

//-------------------------------------------------------
// Test it.

public class dynGenProxy {
   public static void main(String args[]) {

            Thing t = new aThing();

            // First just try it directly...
            t.doSomething();

            // ----------------- Now Proxy it. ----------------------
            // Create Functor
            Functor<String> f = new myFunctor();

            // Create Proxy
            Thing p = (Thing)dynProxy.makeProxy(t,f);

            p.doSomething();
    }
}
```