

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);
    }
}

```

Proxy Examples:

Private – Do not Distribute

```
}  
}
```

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<>(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<>(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);  
    }  
}
```