Sharing Aspect
With AspectJ

```
import practice.AOP.LoggingAspect.dynamicProxy.CalculatorInterface;
public aspect SharingAspect {
	pointcut methodExecution(): execution(* CalculatorInterface.*(..));
	after() returning(int result): methodExecution() {
		if(result > 10) {
			System.out.println("Sharing result: " + result);
		}
	}
}
```

With DynamicProxy

```
import java.lang.reflect.Method;
public class SharingAspect {
	static final int sharingPoint = 10;

	public static Object invoke(Object target, Method method, Object[] args) throws
Throwable {
		Object result = method.invoke(target, args);
		if(result instanceof Integer && (Integer) result > sharingPoint)
		 System.out.println("Sharing result: " + result);
		return result;
	}
}
import java.lang.reflect.InvocationHandler;
import java.lang.reflect.Method;
public class InvocationHandlerExample implements InvocationHandler {
	final Object target;
	public InvocationHandlerExample(Object target) {
		this.target = target;
	}
	@Override
	public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
		return SharingAspect.invoke(target, method, args);
	}
}
import java.lang.reflect.Proxy;
import practice.AOP.LoggingAspect.Calculator;
import practice.AOP.LoggingAspect.dynamicProxy.CalculatorInterface;
public class SharingAspectProxy {
	public static void main(String[] args) {
	 new SharingAspectProxy().go();
    }
    void go() {
        Calculator calculator = new Calculator();
        CalculatorInterface proxy = (CalculatorInterface) Proxy.newProxyInstance(
                CalculatorInterface.class.getClassLoader(),
                new Class[]{CalculatorInterface.class},
                new InvocationHandlerExample(calculator)
        );
        proxy.add(6, 7);
        proxy.subtract(5, 2);
    }
}
```