LoggingAspect

With JavaRefection

Supponiamo di avere una classe Calculator che contiene metodi per eseguire operazioni matematiche di base
e vogliamo aggiungere il logging per tracciare l'esecuzione di questi metodi senza dover modificare direttamente il codice della classe Calculator.

```
public class Calculator {
	public int add(int a, int b) {
		return a + b;
	}
	public int subtract(int a, int b) {
		return a - b;
	}
}
// contiene la logica per registrare i messaggi di log
public aspect LoggingAspect {
	private MyLogger logger = new MyLogger();

	pointcut methodExecution(): execution(* Calculator.*(..));
	before(): methodExecution() {
	 String methodName = thisJoinPoint.getSignature().getName();
	 Object[] args = thisJoinPoint.getArgs();
	logger.info("calling method (" + methodName + ") with arguments (" + argsToString(args)
	+ ")");
    }

	after() returning(Object result): methodExecution() {
		String methodName = thisJoinPoint.getSignature().getName();
		logger.info("method (" + methodName + ") returned: " + result);
	}

	after() throwing(Throwable e): methodExecution() {
		String methodName = thisJoinPoint.getSignature().getName();
		logger.warning("method (" + methodName + ") threw an exeption: " +
		e.getMessage());
	}
	class MyLogger {
		    public void info(String message) {
		        System.out.println("[INFO] " + message);
		    }
		    public void warning(String message) {
		        System.out.println("[WARNING] " + message);
		    }
		    public void error(String message) {
		        System.out.println("[ERROR] " + message);
		    }
	}
	String argsToString(Object[] args) {
	 StringBuilder sb = new StringBuilder();
	 for (Object arg : args) {
	     sb.append(arg).append(", ");
	 }
	 if (sb.length() > 0) {
	     sb.setLength(sb.length() - 2); // Remove the last comma and space
	 }
	 return sb.toString();
    }
}
```

With DynamicProxy

```
public interface CalculatorInterface {
	public int add(int a, int b);
	public int subtract(int a, int b);
}
public class Calculator implements CalculatorInterface {
	@Override
	public int add(int a, int b) {return a + b;}
	@Override
	public int subtract(int a, int b) {return a - b;}
}
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 {
		String methodName = method.getName();
		System.out.println("calling (" + methodName + ") with arguments (" +
argsToString(args) + ")");
		Object result;
		try {
			result = method.invoke(target, args);
			System.out.println("method (" + methodName + ") returned: " + result);
		} catch(Throwable e) {
			System.out.println("method (" + methodName + ") threw an exception: " +
e.getMessage());
			throw e;
		}
		return result;
	}
	private String argsToString(Object[] args) {
		if (args == null || args.length == 0) {return "";}
	 StringBuilder sb = new StringBuilder();
	 for (Object arg : args) {sb.append(arg).append(", ");}
	 if (sb.length() > 0) {sb.setLength(sb.length() - 2);}
	 return sb.toString();
	}
}
import java.lang.reflect.Proxy;
public class LoggingAspect {
	public static void main(String[] args) {
		new LoggingAspect().go();
	}
	void go() {
		// oggetto target
		CalculatorInterface target = new Calculator();
		// handler
		InvocationHandlerExample handler = new InvocationHandlerExample(target);
		// proxy
		CalculatorInterface proxy = (CalculatorInterface) Proxy.newProxyInstance(
		CalculatorInterface.class.getClassLoader(), new
		Class[]{CalculatorInterface.class}, handler);
		// usare il proxy
		proxy.add(2, 4);
		proxy.subtract(2, 4);
	}
}
```