Software Product Line Engineering

Lab class 10 / Assignment 6 + Q/A

Task 1) Terminology

- Aspect-orientation: Modularization of cross-cutting concerns

- **Aspects** are encapsulations of a cross-cutting concern's implementation.
- A **joinpoint** is an event during execution of a program, can serve as a hook for an aspect.
- A **pointcut** is a declarative specification of a set of joinpoints.
- An **advice** is a piece of code that is executed if an associated event (joinpoint) arises.
- An **inter-type declaration** is the declaration of fields, methods, and constructors, which are added to the base implementation by an aspect.
- **Quantification** is the process of selecting several joinpoints via declarative specification (pointcut)

Task 2a-c) Pointcuts in AspectJ

- //every execution of method Client.send(text)

```
pointcut sendExecutionInClientWithParam(String text): execution(void client.Client.send(String)) && args(text);

before(String msgText): sendExecutionInClientWithParam(msgText) {
        System.err.println("sendInClient: " + msgText);
}
```

- //every invocation of a method with name send

```
pointcut allCallsToSend(): call(* *.send(..));

before(): allCallsToSend() {
        System.err.println("Call to send()");
}
```

- //every execution of a public method

```
pointcut allPublicMethodExecutions(): execution(public * *(..));

before(): allPublicMethodExecutions() {
        System.err.println("Public method execution");
}
```

Task 2d-e) Pointcuts in AspectJ

- //every invocation of a method send in class Client not originating in the class Client itself

```
pointcut callsToSendInClientButNotFromClient(): call(public void client.Client.send(..)) && !within(client.Client);

before(): callsToSendInClientButNotFromClient() {
        System.err.println("Call to client.Client.send(..), not from client.Client");
}
```

- //every read-access of the socket on the Server

```
pointcut allReadsOfServerSocket(): get(* server.Server.server);

before(): allReadsOfServerSocket() {
        System.err.println("Read access to server socket");
}
```

Task 2f-g) Pointcuts in AspectJ

- //every instantiation of class Connection

```
pointcut allConnectionInstantiations(): call(server.Connection.new(..));

before(): allConnectionInstantiations() {
        System.err.println("Instantiation of server.Connection");
}
```

- //every method invocation inside the method Server.broadcast() unless the target is an instance of class Connection

```
pointcut allCallsInsideBoradcastExceptToConnection():
        withincode(void server.Server.broadcast(String)) && call(* *(..)) && !target(server.Connection);

before(): allCallsInsideBoradcastExceptToConnection() {
        System.err.println("Call inside server.Server.broadcast, but not to server.Connection");
}
```

Task 3) Advices in AspectJ

```
public aspect Advice {
    after(Foo foo, int i): call(void foo.Foo.firstOperation(int)) && args(i) & this(foo) {
        foo.log("firstOperation executed with parameter " + i);
    }


    around(String parameter, Locker locker): execution(void foo.Foo.main(String, Locker)) && args(parameter, locker) {
        int lockId = locker.lock();
        try {
            proceed(parameter, locker);
        } finally {
            locker.unlock(lockId);
        }

    }
}
```

```
1  public class Foo {
2    void main(String parameter, Locker locker) {
3
4
5        int i = parameter.length();
6        firstOperation(i);
7
8        secondOperation();
9
10
11
12   }
13 }
```

Task 4) Obliviousness Principle

a) What is the obliviousness principle?

The base implementation is unaware of aspects.

b) What are its advantages and disadvantages?

• No preplanning required • Developer do not need to be skilled in aspect-oriented languages/tools	• fragile pointcut problem • aspect-oriented language need to be powerful (increased complexity) • aspects may alter the base implementation execution unexpectedly

c) How can one solve the fragile pointcut problem?

- Convention / Documentation
- Tool support (e.g., Eclipse + AJDT)
- Interfaces between base code and aspects (discarding the obliviousness principle)

Preparation: Implementation Techniques

	Performance	Distributed Development	3rd Party Software	Granularity	Proactive Dev.	Reactive Dev.	Extractive Dev.	Embedded Systems	Required Skills	Scalabilty
Version Control System										
Build Systems										
Preprocessors										
Components / Services										
Framework (White-Box)										
Framework (Black-Box)										
Feature-Oriented Programming										
Aspect-Oriented Programming										