This second lecture is an introduction to AspectJ, with specific focus on the language constructs. In the subsequent lecture 3 we shall further study how to write monitors using AspectJ.

**Installation**

1. Install AspectJ on your preferred system. AspectJ works well in Eclipse for those who want to work with Eclipse.

**Reading**


2. Study AspectJ documentation (it is online and also part of the downloaded system). See course website for details.

**Assignment**

Consider the Java API `java.net.URLConnection` on the Java doc website:

http://java.sun.com/j2se/1.5.0/docs/api/

This is a package for accessing URLs. The following example program accesses the course website, using the `URLConnection` package, and prints its output:

```java
import java.io.*;
import java.net.*;

public class Test {
    public static void main(String[] args) {
        try {
            URL url =
```
new URL("http://www.runtime-verification.org/course");
URLConnection connection = url.openConnection();
DataInputStream dis =
    new DataInputStream(connection.getInputStream());
String inputline;
while((inputline = dis.readLine()) != null)
    System.out.println(inputline);
connection.setAllowUserInteraction(false);
//dis.close();
}catch(Exception e) {}
}
}

The main method first creates a URL object, then opens a connection on that URL, gets
the input stream and reads the source, line by line. The program contains two errors:
(i) it calls the URLConnection.setAllowUserInteraction(boolean) function after
the connection has been read from, and (ii) the inputstream is not closed before program
termination (see commented statement). The documentation of the API requests a
policy (not very clearly stated) for using a URLConnection, specifically referring to
the first kind of error. Write an aspect that checks the following policy:

1. it is not allowed to call a set-method on a URLConnection object after a get-
   method has been called or after the connect method has been called.
2. an input stream that has been created from a URLConnection object using the
getInputStream method should be closed before the main program terminates.

Assume that the program is single-threaded, hence only thread is the main program
executing the main method, as in the test program.

Update:
The second sub-task is difficult to specify and you can ignore it. The reason it is com-
plicated is that the input stream can be passed to another constructor, DataInputStream
in this specific example, and that this in principle can be repeated. The final call of a
close method can be on the last derived input stream.

An alternative simpler non-obligatory (since it is presented so late) sub-task is the fol-

Consider the situation where an InputStream has been created from a
URLConnection object using the getInputStream method. Consider that a
DataInputStream subsequently is generated from this InputStream with a
call of the constructor DataInputStream(InputStream). Then this resulting
DataInputStream should be closed before the main program terminates.