Component-Based Software Engineering	**Exercise Sheet No. 1**
Dipl.-Inf. Florian Heidenreich	Software Technology Group
Office Hours: Wednesdays, 1000–1100 hours	Institute for Software and Multimedia Technology
INF 2080	Department of Computer Science
`http://st.inf.tu-dresden.de/teaching/cbse`	Technische Universität Dresden
	01062 Dresden

Metamodelling and Metaprogramming

Task 1: Reflection in Java (9 Points)

1a)

Write a program that reads the name of a class from the command line and emits the interface of the class in Java syntax (interface or class, modifiers, constructors, methods, fields, no method bodies).

Apply the program to a set of classes and interfaces as test input. Also use the program on itself.

Hints: You can load a class whose name you know with `java.lang.Class.forName()`. `java.lang.Class` offers a rich interface to allow you to inspect the interface of any class.

1b)

Now write a program that reads a class name and a list of arguments from the command line, and creates an object of that class with the read arguments.

Hints: Treat arguments as strings. A `java.lang.Class` can enumerate its constructors. Choose a constructor with the appropriate parameter count. Then, find the parameter types. To create typed argument objects, call the appropriate constructors that take a string as their only argument. Call dynamic constructors using `java.lang.reflect.Constructor.newInstance()`.

1c)

Extend the solution of the previous task by reading a method name and a list of method arguments. Dynamically call the method on a newly created object.

Use your program to invoke your solution from subtask (a) to print the interface of your solution of this subtask.

Hint: Use `java.lang.reflect.Method.invoke()` to dynamically call methods.

Task 2: Static Metaprogramming with OpenJava

This task looks at static metaprogramming in particular with the OpenJava language extensions to Java.

2a) Download and install OpenJava. Use the package I provided on the exercise website, at least if you're working on a Windows box. The original website for OpenJava is: http://openjava.sourceforge.net/. Work through the tutorial to get a first idea of how OpenJava works.

2b)

The following listing (`DataContainer.oj`) is an OpenJava program that represents a simple data container:

```
package dataaccess;

import javax.swing.JOptionPane;

/**
 * @author sz9
 * @since 01.03.2005
 */
public class DataContainer instantiates metaclasses.PWDProtectedClass {

  private String m_sMyImportantData;

  public DataContainer () {
    super ();

    m_sMyImportantData = "Initial important data.";
  }

  public String getData () {
    return m_sMyImportantData;
  }

  public pwdprotected void setData (String sNewData) {
    m_sMyImportantData = sNewData;
  }

  public static void main (String args[]) {
    DataContainer container = new DataContainer ();
    System.out.println ("Current contents: " + container.getData());

    container.setData (
        JOptionPane.showInputDialog ("Enter data to set",
                                     container.getData())
      );
    System.out.println ("Current contents: " + container.getData());

    container.setData (
        JOptionPane.showInputDialog ("Enter data to set",
                                     container.getData())
      );
    System.out.println ("Current contents: " + container.getData());
    System.exit (0);
  }
}
```

Implement an OpenJava metaclass `PWDProtectedClass` that wraps all methods with the `pwdprotected` modifier in password protection code. The code should use a `javax.swing.JOptionPane` to obtain a password from the user whenever such a method is called. If the user enters the right password, the actual method code should run, otherwise the method returns directly. Once the user has entered the password correctly, he/she should not be asked to enter the password for any calls to the same object again. Calls to other objects, however, still require the password to be entered.

Test your code by using it to translate `DataContainer.oj` and running the resulting program.

Task 3: Dynamic Metaprogramming with DynaMOP

In this task we are going to go all the way and have a look at dynamic metaprogramming. There are a few languages out there, which provide ways of dynamically influencing the structure of a program—for example, MetaML, LISP, or Smalltalk—but they are typically pretty complex and difficult to get used to. So, in order to give you a chance to try out some dynamic metaprogramming without having to learn a complete language, I have devised DynaMOP, a very simple, interpreted, object-oriented language, which essentially cannot do anything but dynamic metaprogramming. I have provided a package with the corresponding sources and some documentation on the exercise web page. Download and install (i.e., unzip into a directory of your choice) the package.

3a)

Implement a simple associative array `Associations`. This class provides one operation `getAssociation (key: String): String`, which returns the string value associated with the given string key. If the key is still unknown, `getAssociation` asks the user to enter a corresponding value and then stores and returns this value.

Here is the `main` operation that you can use to exercise your class:

```
main {
  Associations cache = Associations.new;
  String key = "af";
  while (key != "Schluss") {
    key = System.getStringInput ("Please enter key:");

    String val = cache.getAssociation (key);
    System.println ("Corresponding data:");
    System.println (val);
  };
}
```

Hint: Use method redefinition to store the values.

3b)

Implement a class `Singleton`, which enforces that there can never be more than one instance of the class. How can dynamic metaprogramming help you here?

Hint: Remember that you may redefine a class's `new` operation. The original `new` is defined in `Object`, which is the super class of all classes.

Task 4: Knowledge Metaprogramming (8 Points)

4a) Enumerate and explain several times in system construction.

4b) Explain the difference between static and dynamic metaprogramming.

4c) What is the difference between reflection in Java and dynamic metaprogramming?