

Advanced Aspects of Object-Oriented Programming (SS 2009)

Practice Sheet 2

Date of Issue: 28.04.09
Deadline: 04.05.09
(until 10 a.m. as PDF via E-Mail)

Exercise 1 Prototyping

In this exercise, we will encode the class-based approach to object-orientation using prototyping. First of all, have a look at the wikipedia page about prototype-based programming: (http://en.wikipedia.org/wiki/Prototype-based_programming).

In pure prototyping there are no visible pointers or links to the original prototype from which an object is cloned. The prototype object is copied exactly, but given a different name (or reference). Behavior and attributes are simply duplicated as-is.

We want to implement the University Administration System from the lecture using prototyping with a class-based approach. As we want to do this in the (familiar) Java language, which offers no direct support for prototyping, we present a small library to provide prototyping facilities. Furthermore, we restrict the library user from using certain Java features. In fact, you are only allowed to use pre-defined (Java) types. However, it is allowed to create anonymous classes which are direct subtypes of the `Function` class. You are not allowed to use reflection.

```
public final class ProtoObj {
    /** Initial object */
    public static final ProtoObj INIT_PROTO = new ProtoObj();

    /** Tests whether the object has the given property */
    public final boolean hasProperty(String name) { ... }

    /** Set the property with name propertyName to the given value.
     * If no such property exists, it is created. */
    public final void setProperty(String propertyName, Object value) { ... }

    /** Gets the value of the property with name propertyName */
    public final Object getProperty(String propertyName) { ... }

    /** Gets the value of the (function) property with name functionName.
     * This is just a convenience method to avoid casting. */
    public final Function getFunction(String functionName) { ... }

    /** clone this object (copy all properties) */
    public final ProtoObj clone() { ... }
}

/** This class represents a function. To create a new function, override the run method */
public abstract class Function {
    public abstract void run(ProtoObj self, Object... args);
}
```

- a) To get familiar with the prototype-based programming approach, you can look at the following example (similar to the one in the lecture):

```
ProtoObj vehicle = ProtoObj.INIT_PROTO.clone();
vehicle.setProperty("name", "_");
ProtoObj sportsCar = vehicle.clone();
sportsCar.setProperty("driveToWork", new Function() {
    @Override
    public void run(ProtoObj self, Object... args) {
        System.out.println(self.getProperty("name") + "_drives_to_work");
    }
});
ProtoObj porsche911 = sportsCar.clone();
porsche911.setProperty("name", "Bobs_Car");

// call the driveToWork method on the porsche911 object
porsche911.getFunction("driveToWork").run(porsche911);
```

- b) Implement the University Administration System in the spirit of the provided Java sources (UAS.zip). You should use the following template to implement the system (e.g. implement the methods `call` and `newObject`).

```

static ProtoObj PERSON_TYPE = ...
static ProtoObj STUDENT_TYPE = ...
static ProtoObj PROFESSOR_TYPE = ...
static ProtoObj ASSISTENT_TYPE = ...

public static void main(String... args) {
    ProtoObj michi = newObject(PERSON_TYPE, "Michi");
    ProtoObj john = newObject(STUDENT_TYPE, "John", 12345);
    ProtoObj einstein = newObject(PROFESSOR_TYPE, "Prof. Einstein", 402, "AGL Relativ");
    ProtoObj peter = newObject(ASSISTENT_TYPE, "Peter", true);

    ProtoObj[] people = {john, michi, einstein, peter};

    for (ProtoObj p : people) {
        call(p, "print");
    }
}

public static void call(ProtoObj self, String functionName, Object... args) { ... }

public static ProtoObj newObject(ProtoObj type, Object... args) { ... }

```

Exercise 2 Reflection and Annotations

The Proxy class (`java.lang.reflect.Proxy`) allows to intercept method calls.

- Inform yourself about the Proxy class using the JDK documentation. Is it a *normal* JDK class? Also inform yourself about Java annotations (<http://java.sun.com/docs/books/tutorial/java/java00/annotations.html>).
- Develop a generic wrapper using the Proxy class, which allows, for a certain object, upon invocation of one of its methods, to check annotated method parameters for non-nullness. An example of using the wrapper could look as follows:

```

public interface Example {
    public void print(@NonNull String s);
}

...

Example e = new Example() {
    public void print(String s) {
        System.out.println(s);
    }
};
e = (Example) Wrapper.wrap(e);
e.print("Hello"); // prints out "Hello"
e.print(null); // should throw an exception

```

The annotation type declaration is given as follows (does not need to be modified).

```

import java.lang.annotation.*;

@Target(ElementType.PARAMETER)
@Inherited
@Retention(RetentionPolicy.RUNTIME)
public @interface NonNull {}

```

Listing 1: NonNull.java

- For which types and methods does the presented technique work?