Prof. Dr. A. Poetzsch-Heffter
Dipl.-Inform. Kathrin Geilmann

University of Kaiserslautern
Department of Computer Science
Software Technology Group

Advanced Aspects of Object-Oriented Programming (SS 2011)

Practice Sheet 3 (Hints and Comments)

Exercise 1 Introspection and Reflection

a) See enclosed source. Note, that the source does not consider enums and does not try to get access to private fields (which is sometimes possible, depending on the security manager). Furthermore, it does not handle arrays, neither as fields nor as toplevel object.

b) It's not, due to value of private fields, static fields, etc.

c) Serialization API: Pro: can store complete objects, including private fields, can be tuned for each class by implementing special methods. Cons: needs the marker interface serializable implemented by all objects it should serialize, writes binary representations, that are dependent on the source, thus code changes can make the stored files unreadable.

   XMLEncoder / XMLDecoder: Stores only the public parts of objects into an xml representation. It has similar restrictions and features as our solution. It was implemented for JavaBeans, but can be used for any objects, that fulfill certain properties, like having a default constructor, having getter and setter methods for fields.

   Java Persistence API : Stores objects into a database. The interfaces are included in the Java API, the implementation has to be provided by third party code. Popular implementations are hibernate and eclipselink. The programmer defines a mapping between an object and a relational datastore and then the implementation provides him with classes and objects that allow to retrieve objects in a very flexible way. It is even possible to query the store like a database.

   Our solution: pro: can be used for any object, the only constraint is, that it has to provide a default constructor, human readable file format. Cons: does not store everything.

Exercise 2 Required and Provided Interfaces

a) Provided interface for classes

   - *outside java.io*: all public methods, constructors, fields, inner classes, etc of `ObjectOutputStream` and its superclasses.
   - *inside java.io*: like for classes outside java.io + protected fields, methods and constructors + fields, methods, constructors with the default modifier.
   - *subclasses of ObjectOutputStream inside java.io* like all other classes inside java.io
   - *subclasses of ObjectOutputStream outside java.io* like all other classes outside java.io + protected field, methods, constructors, etc.

b) To find the required interface of a class, you have to look at the code (documentation, specification) to find out, which objects you can / have to pass to the class in order to work with it. In this case, the OOS requires an OutputStream (parameter to the constructor).

Exercise 3 Structural vs. Nominal Typing

a) Advantages:

   - Very flexible, if an object should be used somewhere, it just has to have the required set of features.
   - Very easy to extend with new types, just factor out commonalities.

   The advantages are at the same time the disadvantages, because the programmer has no direct control over the subtype relation, such that unrelated classes can be subtypes of each other, if by chance they have same methods and instance variables.

b) `Employee` is a subtype of `Student`, so they can register for exams, which is certainly not intended. This problem is an example for subtyping by accident, which is the price to pay for the flexibility of the type system.