CS 520: Object-Oriented Programming
Instructor: Andrew Black
Name: Ovidiu Mura
Submission Date: November 16, 2009; Fall/2009
Homework: Assignment 4
(Due Day: November 9, 2009 @ 16:00)

1. In section 1, cook writes that "objects and ADTs are fundamentally different". Summarize, in your own words, the difference.

Answer:

The object and ADTs are different even if they are both forms of data abstraction. The term *data abstraction* is described as a concept or mechanism hides the implementation details of data. Objects and ADTs are concepts which hide the implementation details of data. William R. Cook presents in his paper[1] a simple example, the representation of integer set. There are two standard approaches to describe sets abstractly: *algebras* relate to ADT and *characteristic function* relates the object.

Representing an integer set through *algebra* approach, a collection of abstract values and operations to manipulate the values. The *characteristic function* approach to represent an integer set maps a domain of values to a Boolean value, which tells us if the respective value is included in the set.

2. Is Smalltalk's implementation of SmallInteger an object or an ADT? Explain why.

Answer:

The Smalltalk's implementation of SmallInteger is an ADT. ADT is characterized by a public name, a hidden representation, and operations to create, combine, and observe values of the abstraction.

> Smalltalk system does include a primitive integer type, implemented as an ADT for efficiency. The primitive type is wrapped in high-level objects, which communicate with each to other through an ingenious interface to perform coercions and implement both fixed and infinite precision arithmetic (On Understanding Data Abstraction, Revisited, William R. Cook, University of Texas at Austin).

The SmallInteger uses for each basic operation (*, -, +, <, >, <=, >=, =, etc.) the primitive operation of abstract basic types, ADT from Smalltalk. For example, the multiplication operation calls the <primitive: 9>.

[1] On Understanding Data Abstraction, Revisited, William R. Cook, University of Texas at Austin

```
* aNumber
        "Primitive. Multiply the receiver by the argument and answer with the
        result if it is a SmallInteger. Fail if the argument or the result is not a
        SmallInteger. Essential. No Lookup. See Object documentation
        whatIsAPrimitive."

        <primitive: 9>
        ^ super * aNumber
```

The primitive denotes an invocation of a virtual machine primitive. Any code following the primitive is executed only if the primitive fails. The same syntax is also used for method annotations (Pharo by Example, p.51).

The code, which follows the primitive call, runs only if the primitive operation fails. This reflects that SmallInteger is a second level abstraction data type.

Smalltalk doesn't allow the user to create an object using the operator *new*. Even the structure is object oriented the implementation, I consider to be an ADT. In the implementation of the methods are used complex operations.

The principles from section 2.6 (*On Understanding Data Abstraction, Revisited,* William R. Cook, University of Texas at Austin), are relevant:

- The SmallInteger work like built in types
- SmallInteger have sound proof techniques
- ADT is implemented efficient
- SmallInteger has a fundamental model based on existential types
- SmallInteger has a solid connection to mathematics.

3. Is Smalltalk's implementation of Boolean an object or an ADT? Explain why.

Answer:

The Smalltalk's implementation of Boolean is an object. The Smalltalk is an object-oriented language, in which Boolean is a subclass of the Object class. The Boolean class inherits all of the methods from the Object class. The Boolean uses procedural abstraction for information hiding, not type abstraction.

There are two Boolean classes in Smalltalk: True class and False class. Both classes are derived from Boolean class. Using the two objects True and False, this is the only way to implement data abstraction in Smalltalk. The objects use simple operations.

4. In section 2.4 Cook writes “the following program is illegal” [in ML]:

```
fun f(a: set, b: set2) = union(a, b)
```

   Explain why it is illegal.

Answer:

There is not allowed the *union* operation to combine two different abstract data types, *set* and *set2.* The *union* operation is defined *union(set s1, set s2).* Allowing multiple implementations for an abstraction is possible in ML but is not allowed to interoperate.

There is not possible to write an operation to *union* two different kind of ADT. First, the both types need to be the same abstract type, and than to make the *union* operation.

5. Explain why Objects do not support what Cook calls “complex operations” (called “binary operations” in the literature).

Answer:

Objects do not support “complex operations” because the objects (in object-oriented design) are not allowed to inspect the other object representation.

Cook defines the “complex operations” as an operation which inspects multiple representations. Objects and ADTs are two forms of data abstraction.
The objects support just simple operations which inspect a single representation (a single form of ADT, which is its own type).

The complex operations are operations which inspects multiple representations (multiple forms of ADTs). ADT has complex operations unlike objects which have operations on the same form of data abstraction (which is the object itself representation).