Computer Science 433
Programming Languages
The College of Saint Rose
Fall 2012

Topic Notes: Encapsulation

Abstraction and *encapsulation* are fundamental concepts in nearly all modern programming languages.

Abstract data types are user-defined (or perhaps language API-defined) data types that

- hide implementation details from users of the types
- provide a limited and restricted set of operations to modify and query the state of instances of the type
- are defined by a single syntactic unit (*e.g.*, a class)

This concept improves reliability, writability, modifiability.

We are all familiar with examples of this, in *e.g.*, Java:

See Example:
`/home/cs433/examples/javageneric`

A C++ example:

See Example:
`/home/cs433/examples/cppvector`

Note that there are different levels of information hiding possible: `public`, `private`, `protected` data members and/or methods.

Most ADTs come equipped with special methods called *constructors*, and in some cases *destructors*, which are called implicitly on creation, deletion, of instances of the ADT, respectively.

In languages without explicit support for encapsulation, such as C, we can still do separate compilation, but there is no guarantee that users of the ADTs will not modify the state of the structure without using provided functions.

See Example:
`/home/cs433/examples/ratios`

Object-oriented programming languages support ADTs (and more). Important terms and concepts:

- *inheritance* – allow classes to be defined in terms of existing ones
- leads to a *class hierarchy*

- a *derived class* or *subclass* inherits from a *parent class* or *superclass*.
- derived classes can add fields and/or methods
- derived classes can *override* an inherited method with one specific to the derived class
- fields can be *class variables* if there is one shared instance for all instances of the class, *instance variables* if there is one per object (in Java, class variables have a `static` qualifier)
- methods can be *class methods* which cannot operate on instance variables, or *instance methods*, which can
- instance variables and methods require an implicit `this` pointer/reference to allow the methods to find the appropriate data for that instance
- *interfaces* are contracts that state that a particular class provides a set of methods, allowing multiple implementations to be used interchangeably as long as the use conforms to the interface
- *abstract classes* are like interfaces, but usually have at least some methods defined, while others need to be provided by a class that is derived from the abstract class
- some languages (*e.g.*, C++) allow *multiple inheritance* – a class that has 2 or more parent classes – this complicates implementation significantly

Excellent examples of Java's OOP constructs can be found at:

On the web: Java Structures web site at
`http://dept.cs.williams.edu/ bailey/JavaStructures/Welcome.html`