CS 247 — Software Engineering Principles

Kevin James

Spring 2014

---

1 Object-Orientation

Code which follows **object-oriented** patterns has better affinity between user-defined types and real-world types, can be more easily reused (inheritance, composition, as-is, polymorphism), allows for encapsulation and information hiding, decomposes extremely effectively (and thus seperates concerns), and allows for abstraction through ADTs (Abstract data types), interfaces, etc.

1.1 ADTs

An ADT is a user-defined data type. They are composed with a range of legal values and functions that maniplate variables of a given type. By providing compiler support for these type restrictions, we turn programmer errors into type errors (which are checked by the compiler).

One of the main motivations for designing an ADT is to ensure safety of any client code. Other motivations include evolvability and scalability. ADTs also tend to improve code efficiency by limiting range checks to constructors and mutators.

An ADT constructor initializes the new object to some legal value and throws an error if the passed-in value is illegal. Accessors and mutators provide restricted read/write access to one of the values in the object. It is best practice to ensure legality within each mutator and use `const` references whenever possible.

We use the **outside-in** method fo development: first we determine what the user wants out of it, then we design and implement it.

Function Overloading is syntactic sugar which allows us to define a function with the same name as some other function, so long as this new function has different parameters. It is generally best practice to only overload functions when the purpose of the function is the same for each. For example: `void print(int)` and `void printfloat`.

Another option is to use **default arguments**. Defulat arguments tend to be used when a given argument should be optional. For example: `void print(int, outputStream=cout)`. Default arguments must appear only in the function declaration.

We can also overload operators. For example, we could define the sum of two classes as some other class, some modified version of one class, or even a standard data type of some sort with `MyClass operator+(const MyClass&, const MyClass&)`. Though widely used, this practice should be used sparingly, if at all.

Nonmember functions are critical ADT functions which are declared outside of the class. This leads to ebtter encapsulation and more flexible packaging. Additonally, certain functions are required to be nonmember functions (e.g. `operator>>`). Streaming operators should be nonmember functions so that they can accept a stream as a first operand and thus chain stream operations.

A class can have `private`, `protected`, and `public` data members and functions (in addition to several less-often used flags). It is best practice to use the most secure flag as possible (generally: `private`)