

Program Development

4.1 Objects

Object-oriented programming rests on the three basic principles of *encapsulation*:

- *Abstraction*: ignore the details
- *Modularization*: break into pieces
- *Information hiding*: separate the implementation and the function

OOP uses the idea of classes. A *class* is a structure which houses data together with operations that act on that data. We strive for *loose coupling*: each class is largely independent and communicates with other classes via a small well-defined interface. We strive for *cohesion*: each class performs one and only one task (for readability, reuse).

We strive for *responsibility-driven design*: each class should be responsible for its own data. You should ask yourself: What does the class need to know? What does it do?

The power of OOP also comes from two further principles which we will discuss later:

- *Inheritance*: classes inherit properties from other classes
- *Polymorphism*: there are multiple implementations of methods and the correct one is executed

4.2 Literate Programming

Good programming requires extensive comments and documentation. At the very least:

explain the purpose of each instance variable, and for each method explain its purpose, parameters, returns, where applicable.

You should also strive for a consistent layout and for expressive variable names. For a class, one might list the functions, constructors and public fields, and for each method explains what it does together with pre-conditions, post-conditions, the meaning of the parameters, exceptions that may be thrown and other things.

UML is an extensive language for modeling programs especially those for an object-oriented programming language. It is a system of diagrams designed to capture objects, interaction between objects, and organization of objects, and then some.

4.3 Testing

One needs to test extensively. Look at the *boundary* values: make sure it handles the smallest or largest value the program must work for, and suitably rejects the value just out of range. Add *watches* or *debug statements* so that you know what is happening at all times. Especially look at the empty case, or the 0 input.