

Recap of Object Oriented Programming Fundamentals

Objects

The active elements of an OO program. Objects are of a definite type (their class, and possibly some other interface) and have two parts: what they know (attributes) and what they can do (behavior). They occupy memory, they get work done, they have a unique id.

Classes

The templates from which objects can be instantiated. The definition of the class determines what objects of that class can know and do. Classes themselves are passive (compared to objects at least, and if you ignore class members).

Encapsulation

The idea that an object encapsulates knowledge and behavior. We control what outsiders may see with access control. Generally, the internal state of an object is hidden from other objects. The algorithms used in the definition of the class methods are hidden by the class.

Includes the idea of containment as an essential relationship between classes.

Inheritance

Classes may be related to each other in an inheritance hierarchy. Top level, abstract classes tend not to be instantiated into active, living, breathing objects. They serve to gather together the common attributes and behavior which their concrete subclasses inherit.

Messaging

Messages are what gets work done in an OO system. There are four parts to a message: a receiver object, a method name, optional method arguments, and an optional return value.

Identity

Objects must have a unique identity, knowledge of which is required to send an object a message. Pointers give us aliases for the unique names of objects, and confuse the issue of identity.

Polymorphism

The idea that the code that is executed as the result of a message being sent depends on the class of the object that receives the message. Objects of different classes can react differently to being sent the same method in a message.

Type

What determines the capabilities or behavior of a thing, such as an object. Distinct from, but often confused with, class. Type is equivalent to the interface of a class. Programming to types, not classes, maintains flexibility.