Object-Oriented Design

Abstract Data Types

Outline

- Object-oriented design
- Abstract data types

Object-Oriented Design

Object-Oriented Design

- A design method where *classes* are created for the data types
  - In an object-oriented programming language
    - Java, or C++, or Python, …
- Object variables are *instances* of a class
- A class specifies
  - Data – attributes
  - Behaviour – functions

but not C

i.e. variables that belong to an object

referred to as methods

Object-Oriented Design Principles

- Abstraction
  - Describes the major purpose of an object
  - Its behavior
    - But not details about its implementation
- Encapsulation
- Polymorphism
- Inheritance

What's a stack?

A container that only allows values to be inserted and removed from the top

Object-Oriented Design Principles

- Abstraction
- Encapsulation
  - Internal details of a class should be inaccessible to other classes
    - Also known as information hiding
  - Other classes need rely only on the interface
    - Does not force an implementation
- Polymorphism
- Inheritance

Otherwise, refactoring a class necessitates changing all the modules that use it

Object-Oriented Design Principles

- Abstraction
- Encapsulation
- Polymorphism
  - Behaviour changes based on the type of an object
  - The same interface for two related classes
    - With different behaviour
- Inheritance

Relates to inheritance, not discussed much in 225

Object-Oriented Design Principles

- Abstraction
- Encapsulation
- Polymorphism
- Inheritance
  - One class – the subclass – can inherit methods and attributes from another – the superclass
    - Implementing an abstract base class
    - Extending or specializing an existing class

Modularity

- Classes are often implemented in separate files from other classes
  - So that each class forms a separate module
- Desirable because
  - It increases the reusability of components
    - Can use the class in multiple projects
  - Different teams can work on different classes
- Modules should be *loosely coupled*

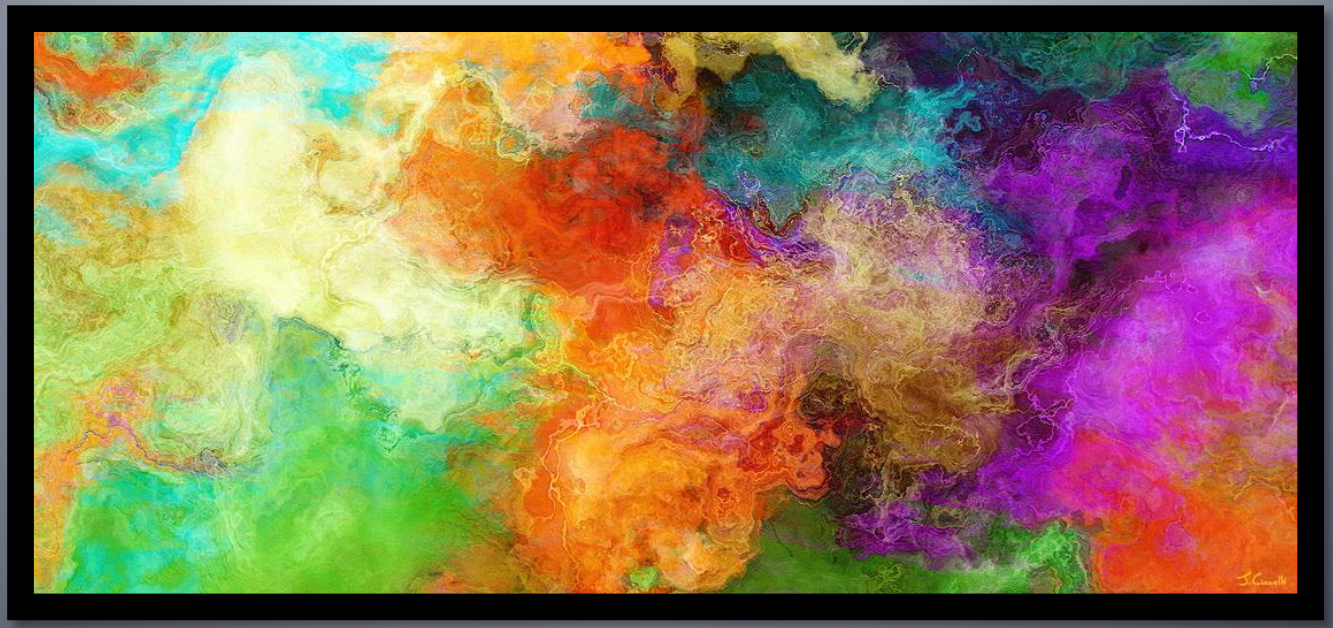
Coupling and Cohesion

*We want to design components that are self-contained: independent and with a single, well-defined purpose**

- Classes should be loosely coupled
  - Independent of each other
  - Should only communicate through their interfaces
    - And not have access to implementation details
- Classes should be highly cohesive
  - Should encapsulte behavior relating to a single task
  - All the functionality should be within the class

*The Pragmatic Programmer, Hunt and Thomas

Abstract Data Types



Abstract Data Types

- A collection of data
  - Describes ***what*** data is stored but ***not how*** it is stored
- Set of operations on the data
  - Describes precisely ***what*** effect the operations have on the data but
  - Does ***not*** specify ***how*** operations are carried out
- An ADT is not an actual (*concrete*) structure

e.g. a Stack

Concrete Data Type

- The term *concrete data type* is usually used in contrast with an ADT
- An ADT is a collection of data and a set of operations on the data
- A concrete data type is an *implementation* of an ADT using a *data structure*
  - A construct that is defined in a programming language to store a collection of data

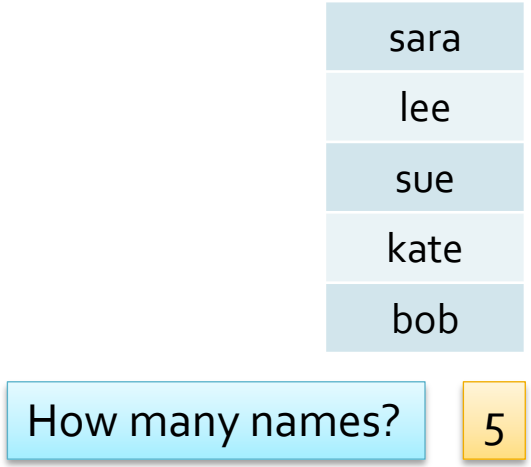
e.g. a Stack implemented with an array

ADT Operations

- Mutators
  - Often known as *setters*
  - Operations that change the contents of an ADT usually subdivided into
    - Adding data to a data collection and
    - Removing data from a collection
  - Different ADTs allow data to be added and removed at different locations and in different ways
- Accessors
- Constructors
- Other

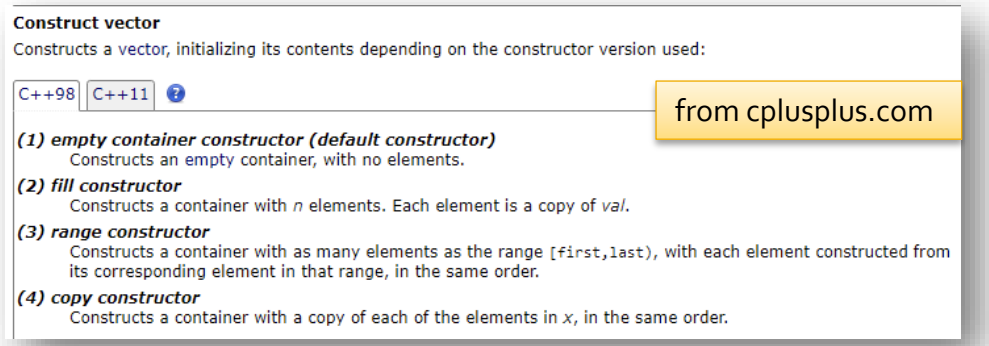
ADT Operators

- Mutators
- Accessors
  - Often known as *getters*
  - Retrieve data from the collection
    - e.g. the item at the top of the stack
  - Ask questions about the data collection
    - Is it full?
    - How many items are stored?
    - …
- Constructors
- Other



ADT Operators

- Mutators
- Accessors
- Constructors
  - Create an ADT
    - Empty
    - Initialized with data
    - A copy of an existing object
- Other



Construct vector

Constructs a vector, initializing its contents depending on the constructor version used:

C++98 C++11

from cplusplus.com

(1) empty container constructor (default constructor)
Constructs an empty container, with no elements.

(2) fill constructor
Constructs a container with *n* elements. Each element is a copy of *val*.

(3) range constructor
Constructs a container with as many elements as the range `[first,last)`, with each element constructed from its corresponding element in that range, in the same order.

(4) copy constructor
Constructs a container with a copy of each of the elements in *x*, in the same order.

aka a *clone*

Information Hiding

- Information related to how storage is implemented should be hidden
- An ADT's operations can be used in the design of other modules or applications
  - Other modules do not need to know the *implementation* of the ADT operations
  - Which allows implementation of operations to be changed without affecting other modules
- Different languages handle information hiding in different ways

Specification of ADT Operations

- Operations should be specified in detail without describing implementation issues
  - In C++ an implementation of an ADT is divided into header (*.h*) and implementation (*.cpp*) files
- The *header file* contains the **class** definition which only includes method prototypes
  - Occasionally there are exceptions to this
- The *implementation file* contains the method definition

C++ Classes – Private vs Public

- The public methods of a class define its *interface*
  - The way in which it communicates with the rest of the application and in which it can be acted on
  - Any method that does not need to be called from *outside* the class should be made private
- Class attributes should usually be made private
  - They form part of the class implementation and should remain hidden
  - And protected from inappropriate changes

Objects can access private attributes (or methods) of *other* objects of the same class

C++ Classes and Dynamic Memory

- If a class allocates space in dynamic memory using *new* it should provide
  - A destructor that deallocates dynamic memory by calling *delete*
  - A *copy constructor* that makes a deep copy
    - That is called when objects are passed by value
  - An *overloaded assignment operator* that makes a deep copy
    - That is called when objects are returned (by value)