

# Modularity and Abstraction

Feb 23

An **abstract data type** is a collection of objects and operations that present well defined abstract properties to the users, meanwhile hiding the way they are represented in terms of lower level data representation.

I.e. allowing the user to make use of a data type without telling them how its implemented.  
E.g. a stack.

The idea is to separate behavior (public) from implementation (private).

The interface is a contract between the user and the implementer outlining what is expected from the class.

## Priority Queue

Specification:

- PriorityQueue()
  - Constructor: creates an empty queue.
- size()
  - Returns int value outlining how many items in the queue
- insert(ComparisonKey o)
  - Inserts a comparable object onto the queue.
- remove()
  - Returns highest priority object object in the queue.
  - Returns type ComparisonKey

In order to use this idea, we need to be able to compare objects.

- Some are easy, e.g. int, String, double.
- We need to guarantee that an objects class has a method to compare the objects.

Solution:

Use an interface that guarantees that the object is comparable. (i.e. an interface with a compareTo method in it). Every class that wants to store objects on the priority queue must implement this interface.

[See the ComparisonKey interface code ☺](#)

This comparison key guarantees that any object that gets put onto the stack is comparable and therefor able to be prioritized.

[See PQ examples ☺](#)