Abstraction

- *Abstraction*: ignoring details in order to reason about larger, more complex systems
  - Example: When planning with a map, you think about *moving* from *place* to *place*, ignoring things like taking steps, avoiding obstacles, looking around and breathing
  - Example: When planning a night out, you think about *getting places*, *meeting up, having dinner* and such, ignoring things like getting in the car, turning the key, navigating, recognizing people, ordering, chewing, etc...
- If you don't *abstract* away details, even simple problems are way too complex to figure out!

Abstraction

- Computers don't automatically (unconsciously) abstract things like people do
  - Programs have to specify every little detail
  - Even simple problems are too complex to effectively design, implement, debug and maintain
  - Good code design is about good abstraction
  - *Object-oriented design* with classes is one (dominant) approach

Abstraction: container class

- *Containers* are a vital abstraction in program design
  - Simply, a class that holds more than one of something
  - An *array* is a very no-frills container
- Specific containers have specific capabilities:
  - Phone book
  - Todo list
  - Class roster
  - Calendar
  - Email inbox
  - Deck of cards

Abstraction: container class

- Specific containers have specific capabilities:
  - Phone book
  - Todo list
  - Class roster
  - Calendar
  - Email inbox
  - The line at the DMV
  - Deck of cards
- Based on those examples, what are the general-purpose (abstract) capabilities of a container?

Specifying an element in a container

- A number of actions require that you specify which thing in the container (*element*) to act on
  - Retrieve, delete
- You can specify by *value*:
  - E.g. Bob, the ace of clubs, student 34123422, Jim's phone number
- Or by *position*:
  - E.g. the first person in line, the top card, the third seat
- You can also *iterate*:
  - Act on each element in the container, one at a time

Exercise: container class

- Based on those capabilities, write a main function that tests a hypothetical class called *Container* that acts as a container for some class called *ClassX*
  - *Encapsulation* is a mechanism for implementing abstraction
  - In C++, this is done by making data members private, and using public methods to manipulate them
  - The capabilities you came up with should be public methods of the class, not data members!
    - What information does each method require (input parameters)?
    - What information does each method give back (return value or reference parameter)?

Exercise: container class

- Now, write a class definition for *Container* that can satisfy your test program
  - What private data members do you need?

All about the memory

- What happens when we add an object to our container?
  - We make a *copy* in a different place in memory
- This has a lot of implications
  - Make sure you can keep track of where everything is in memory!