

Unit 03: Interfaces and Abstract Data Types (ADTs)

Anthony Estey

CSC 115: Fundamentals of Programming II

University of Victoria

Unit 03 Overview

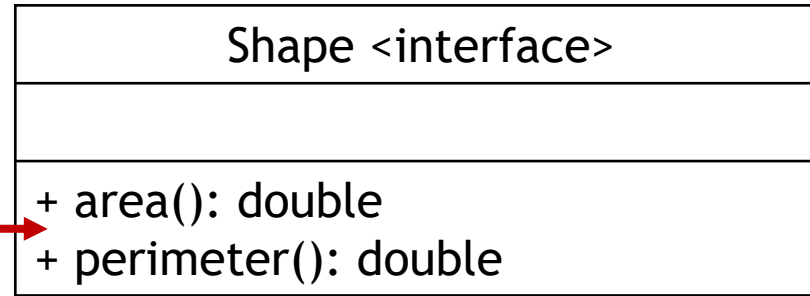
- ▶ Related Reading:
 - ▶ Textbook Chapter 4
- ▶ Learning Objectives: (You should be able to...)
 - ▶ Be able to write an interface and implement an interface
 - ▶ understand the concept of an abstract data type

Interfaces in Java

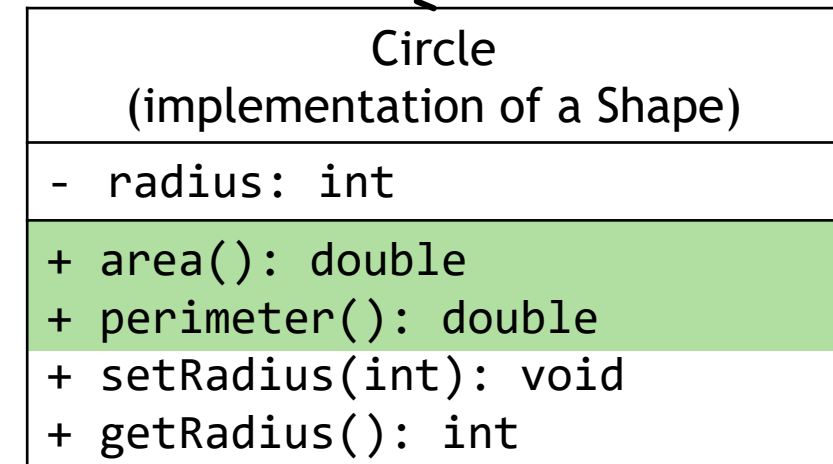
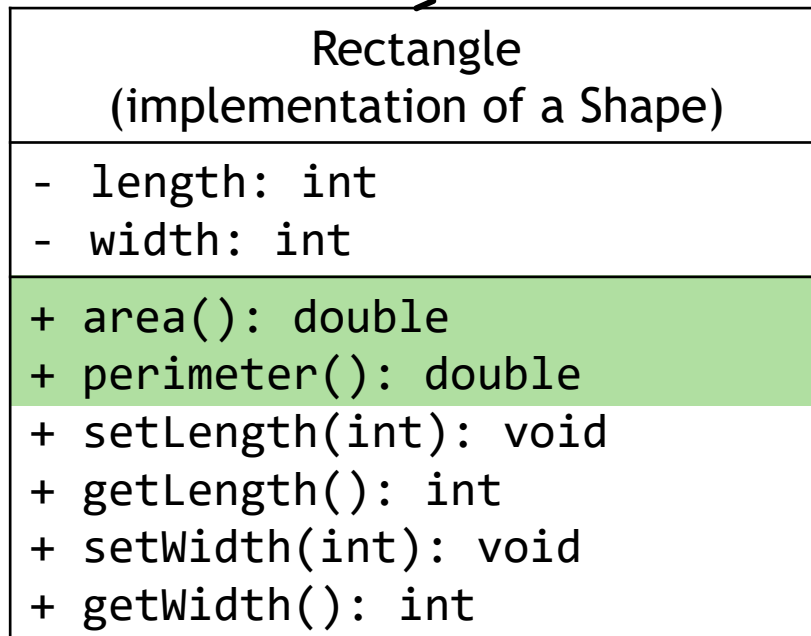
- ▶ A Java interface specifies methods and constants but supplies no implementation details
- ▶ Can be used to specify some common behavior that may be useful over many different types of objects
- ▶ We can think of an interface like a **contract**. The person writing the interface (presenting the contract) says, *“I want these features”*
- ▶ The developer implementing the interface (fulfills the contract), replies, *“Okay, I will build those features into the solution”*

Interface Shape Example

We must be able to calculate the perimeter and area of all shapes

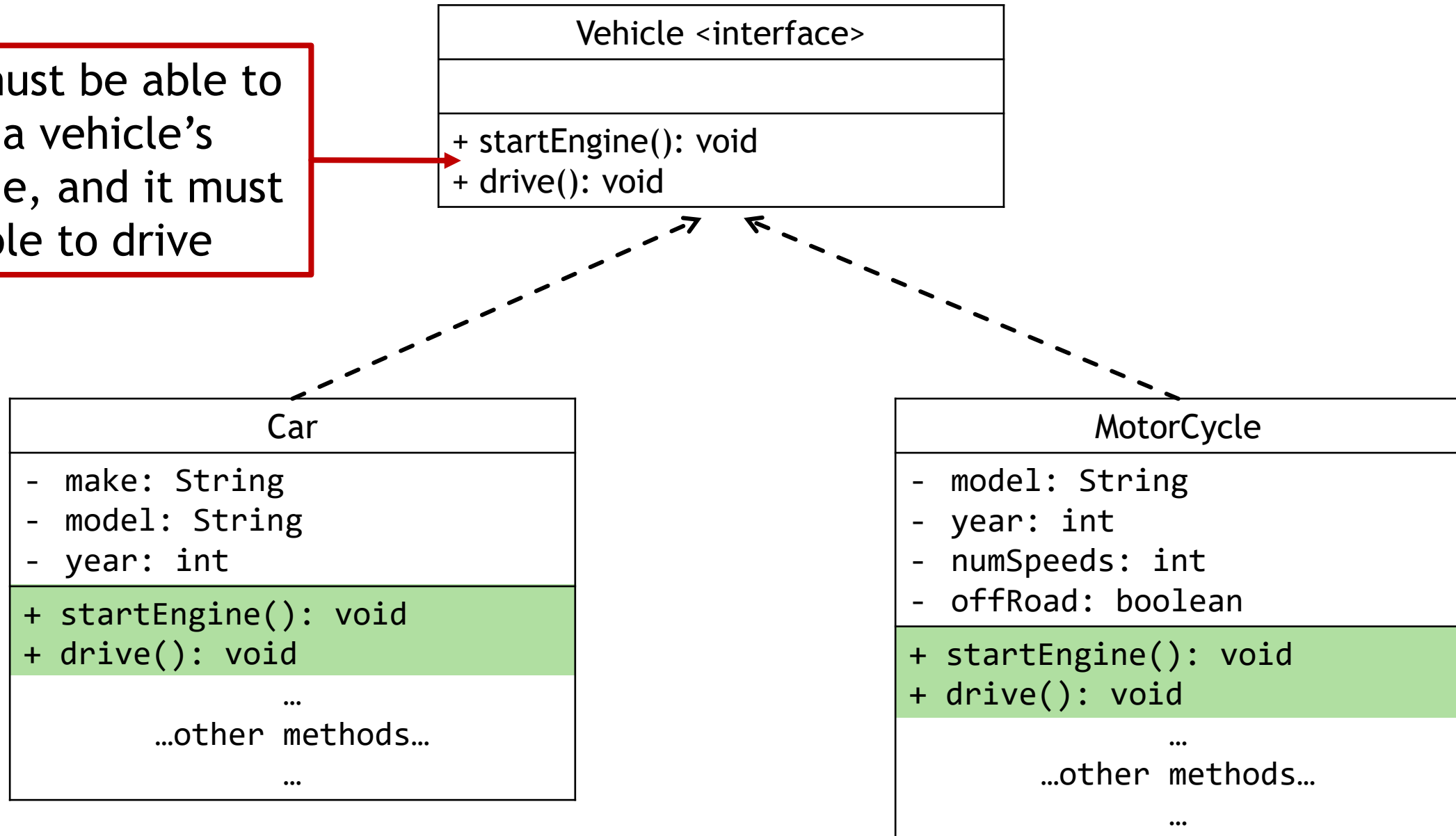


Both implementations fulfill the contract (we can get the shape's perimeter and area)



Interface Vehicle Example

We must be able to start a vehicle's engine, and it must be able to drive



Writing code that uses an Interface

► General structure:

```
public interface name {  
    public returnType method1();  
    public returnType method2();  
}
```

► Specific example:

```
public interface Shape {  
    public double area();  
    public double perimeter();  
}
```

Our classes begin with:

```
public class name
```

Interfaces instead have:

```
public interface name
```

The methods are only signatures
(they don't have a body)

An interface doesn't do anything;
it specifies desired behaviors

An interface typically does
provide documentation for each
method specified (comments)

Interface Code Example

► General structure:

```
public class name implements interfaceName {  
    public returnType method1() {  
        statements;  
    }  
}
```

Classes that implement an interface specify which interface they implement (which contract they fulfill)

► Specific example:

```
public class Circle implements Shape {  
    ...  
    public double area() {  
        return Math.PI * radius * radius;  
    }  
}
```

The methods do have bodies, as they provide an implementation of the required operations.

Documentation / Comments are okay!

```
1 public interface Shape{
2
3     /*
4      * Purpose: calculates the area of this Shape
5      * Parameters: none
6      * Returns: double - the area of the shape
7      */
8     double area();
9
10    /*
11     * Purpose: calculates the perimeter of this Shape
12     * Parameters: none
13     * Returns: double - the perimeter of the shape
14     */
15     double perimeter();
16 }
```

- My example didn't include documentation, but it should be used to clarify the desired behaviour

Interfaces - Why?

- ▶ What are the reasons we might want to use an Interface?
- ▶ Being able to guarantee a program (or suite of programs) all include certain features that work in a certain way is very useful

What is an interface?

These objects implement the interface `IPowerPlug`



So they can be used with `PowerSocket` objects



Interfaces - Why?

- ▶ Also think about it from a software development perspective:
- ▶ Interfaces only contain desired behaviours, no code
 - ▶ This is the perfect medium for which clients and developers can communicate
 - ▶ Clients can request behaviours and/or operations about their desired product; developers can discuss these requests, until an agreement is made
 - ▶ Clients are not programmers, they are not interested in implementation details. They only need to see the interface (contract).
 - ▶ Developers then provide an implementation based on the contract (the clients will *use* the end product, but don't ever have to see the underlying code)

Data abstraction

- ▶ We have now seen that we can create interfaces in Java
- ▶ Interfaces provide required methods, but omit the details of **fields** and *how* the methods are implemented
 - ▶ So far, we have seen that interfaces can be used to create types for data that supports multiple variants (different shapes or different vehicles)
- ▶ The separation of what something does (specification) and how it does it (implementation) is a fundamental concept in engineering!

Data structures

- ▶ A storage structure for data
 - ▶ We will explore different types of data structures throughout this course (and explore even more in CSC 225)
- ▶ A way of *storing, accessing, organizing, and manipulating* data using a set of well-define operations
 - ▶ Wikipedia definition: a collection of data values, the relationships among them, and the functions or operations that can be applied to the data
- ▶ The only data structure we have used so far is an array
 - ▶ there are other data structures we can use to perform the same operations on our collection of data, which are implement in different ways
 - ▶ ... and have different speeds and memory requirements

Two important pieces

- ▶ Interfaces and data structures are two very important pieces:
 - ▶ Interfaces allow us to specify operations
 - ▶ Data structures allow us to organize, access, and manipulate data

Abstract Data Types (ADTs)

- ▶ An ADT is composed of:
 - ▶ A description of what data is stored (but not how the data is stored)
 - ▶ A set of operations on that data (but not how the operations are implemented)
- ▶ Specifications of an ADT indicate
 - ▶ What the ADT operations do (*interface*)
- ▶ Implementations of an ADT include choosing a particular data structure
 - ▶ *how* is the data stored, accessed, organized, etc. (*data structure*)

Example

- ▶ ADT Dictionary:
- ▶ *What* does it do?
 - ▶ Stores a pair of strings, representing the word and definition (data)
 - ▶ Operations:
 - ▶ insert(word, definition)
 - ▶ delete(word)
 - ▶ find(word)
- ▶ We use a data structure to implement an ADT
 - ▶ this is where the *how* comes in

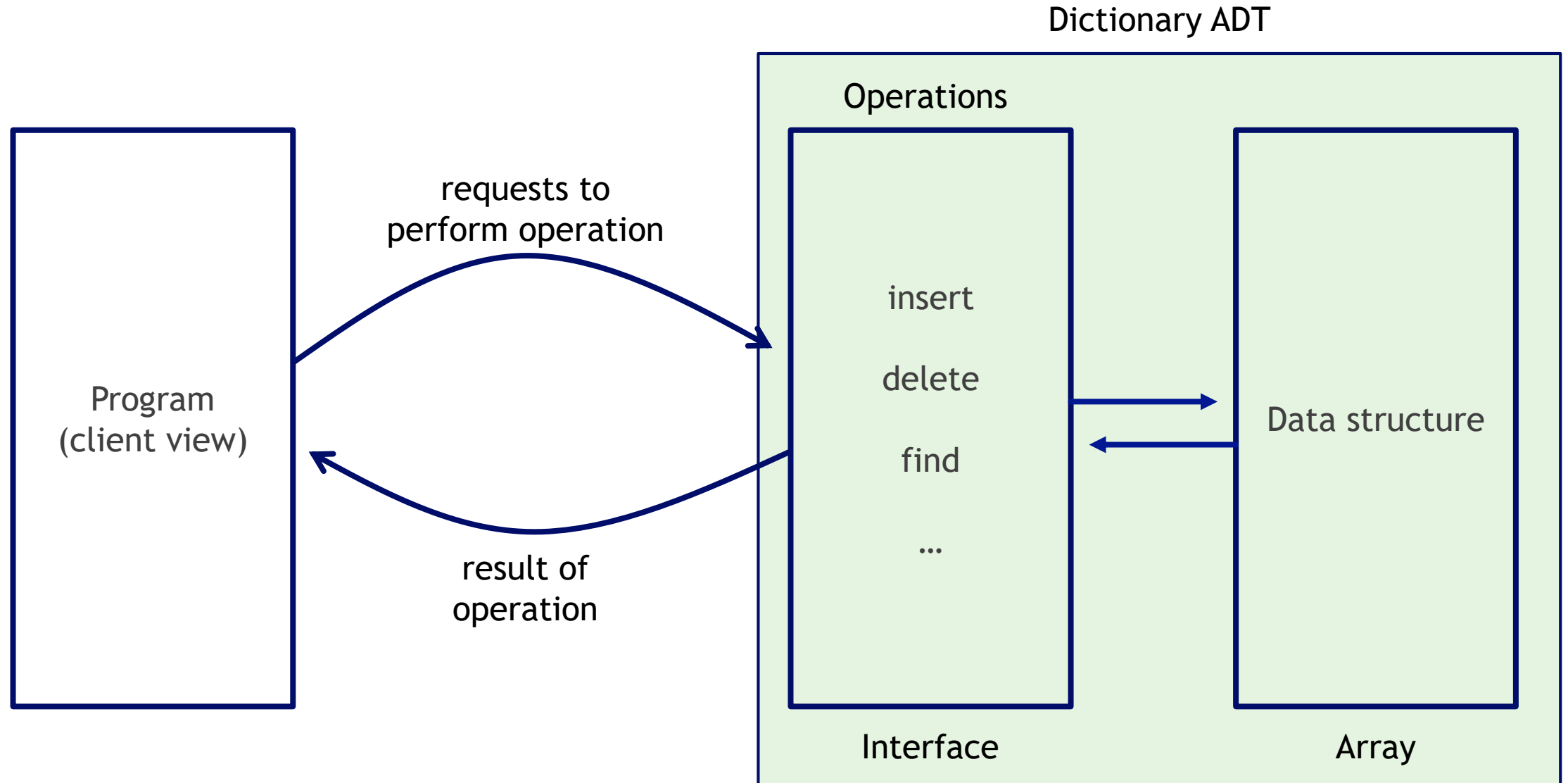
We also know the effect operations have on the data.

If we ***delete*** a word from the dictionary, a subsequent ***find*** operation should fail.

Client vs Programmer

- ▶ Clients know how to use something
 - ▶ *What* operations are available and *what* they do
- ▶ Programmers must decide *how* to implement the operations
- ▶ Their choices may be influenced by a number of things:
 - ▶ execution speed
 - ▶ memory requirements
 - ▶ maintenance (debugging, scalability, etc.)
- ▶ Dictionary example:
 - ▶ Clients/users: add new words to dictionary, look up words to see definitions
 - ▶ Programmer: determine *how* data is stored; *how* operations are implemented

Implementing the Dictionary ADT

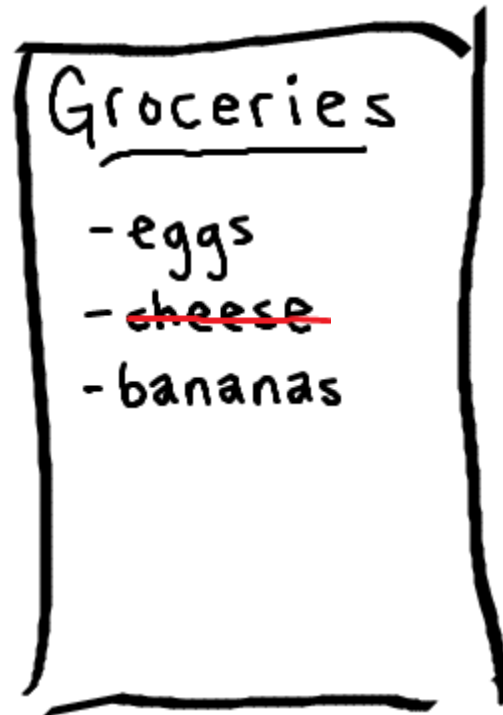


ADT Example

- ▶ Assume you wanted to create something that allowed someone to:
 - ▶ Keep track of what groceries they needed to buy
 - ▶ Maintain information about all of their contacts
 - ▶ Record all of the courses they have completed as they progress through their undergraduate degree program
- ▶ Can these be generalized into a common set of required features?

The Notion of a List

- ▶ A **list** allows us one to manage a collection of items
 - ▶ Elements can be inserted and removed in *any* order
 - ▶ Any element can be accessed at any given time by their position in the list



Another ADT Example: List

- ▶ ADT List Operations:
 - ▶ Create an empty list
 - ▶ Determine whether a list is empty
 - ▶ Determine the number of items in a list
 - ▶ Add an item at a given position in a list
 - ▶ Remove the item at a given position in a list
 - ▶ Get the item at a given position in a list
 - ▶ Remove all items from a list
- ▶ Items are referenced by their position in a list:
 - ▶ (1st, 2nd, 3rd, etc)

The ADT List

- ▶ Specifications of the operations (interface)
 - ▶ Define the ‘*contract*’ for the ADT list
 - ▶ Include the operations a list must be able to perform
 - ▶ Do not specify how to store the list or how to perform the operations
- ▶ Remember the important takeaway:
 - ▶ The operations can be used in an application without knowledge of how the operations are implemented

The ADT List

- ▶ Programmer Implements a list using a data structure
- ▶ So far the only data structure we have seen is an array:
 - ▶ Each item in the list is stored in an array
 - ▶ Items positions are identified by their index within the array (so the k th item will be stored at index $k-1$ (since arrays are 0-indexed))
- ▶ We will work on this implementation today!
- ▶ But next week we will see there are other data structures we can use to implement the list ADT (and other ADTs as well)