CSCI 136: Data Structures and Advanced Programming
Lecture 5
Abstraction / Generics
Instructor: Dan Barowy Williams

Topics

• Abstraction
• Generics
• Vectors

Your to-dos

1. Lab 1, due Tuesday 9/20 by 10pm.
2. Read before Wed: Bailey, all of Ch 2. Suggestion: read actively.

Active reading

Code provided in the book/in class is there for a reason: to teach you a lesson!

Retype—don’t copy and paste—that code into your editor, compile it, and run it.

In the process, you will notice detail you wouldn’t otherwise. Programming is all about the detail.

I did not invent this idea. One famous practitioner…
Active reading

"I would advise you to read with a pen in your hand, and enter in a little book short hints of what you find [...] for this will be the best method of imprinting such particulars in your memory.”
—Benjamin Franklin

This is sometimes called “learning the hard way.”

Thing is, it's actually one of the easiest ways to learn.

Remember: learning feels inefficient.

The purpose of a class:

To “abstract away” implementation detail.

Abstraction

Abstraction is the process of removing irrelevant detail so that a problem contains only necessary information.
Think of a class as having two sides.

**The outside:** A class should represent one concept, and the class’s methods should support working with that one concept.

**E.g., Nim:** Represents a Nim game board.

You can ask it to:
- set up a new game (the constructor),
- print out its board (displayBoard),
- check whether a move is valid (isValidMove),
- check whether the game is over (isGameOver),
- prompt the user to take a turn (takeATurn),
- etc.

Think of a class as having two sides.

**The inside:** A class should contain whatever code is necessary to implement that one concept and nothing else.

**E.g., Nim:** Represents a Nim game board.

Stores:
- int[] of piles
- int representing the current player
- etc.

Ensures:
- Board is initialized correctly (new)
- Board is represented naturally to a user (displayBoard)
- Moves are valid (via isValidMove when taking a turn).
- etc.
Hiding data inside a class is called: **encapsulation**

Classes can **encapsulate** other classes!

Suppose we wanted to write a function that reverses an **array of ints**.

Let’s try to do that together.
Suppose we wanted to write one function that reverses an array of any type.

Let’s try to modify our program.

Problem: Java has no (type safe) way to express the idea of “any array.”

However, there is an alternative…

Generic types

A generic type is a placeholder (a type variable) for a type to be specified later. Generic types permit the creation of common algorithms and data structures (e.g., a generic sequence), thus reducing code duplication. Generics allow for data type abstraction.

In this class, we will focus on use. Later, we will revisit how to make your own (i.e., definition)
Vector\langle E \rangle \text{ is a sequence of any type } E

The Vector\langle E \rangle class itself handles growing its internal array if space is insufficient.

Vector \text{ is a generic class; it works with any type.}

The type parameter you use must be a class type.

Primitives (like int) do not work. Use “boxed” types instead.

Suppose we wanted to write one function that reverses any array.

Let’s try again.
Recap & Next Class

Today:

• Abstraction
• Generics
• Vector

Next class:

• How Java computes things