CSE 373: Introduction, ADTs, Design Decisions, Generics

Michael Lee
Wednesday Jan 3, 2017
Overview

- Michael Lee (mlee42@cs.washington.edu)
  - Currently working on a master’s degree in Computer Science
  - Supervised by Adam Blank

- Office hours (CSE 216)
  - Tuesdays from 1:30 to 3:30
  - Fridays from 4:30 to 6:30
  - Or by appointment
1. About this course
2. Data structures vs abstract data types (ADTs)
3. Generics
4. Administrivia
What are data structures and algorithms?

Data structure: a way of organizing and storing data

Algorithm: a series of precise instructions used to perform a task
What are data structures and algorithms?

Data structures store data

Algorithms do things
Basic techniques for storing and manipulating data

▶ “Expanding arrays”
▶ Nodes and pointers/references
▶ Trees and recursion
Basic techniques for storing and manipulating data

- “Expanding arrays”
- Nodes and pointers/references
- Trees and recursion

How to use pre-made data structures

- Using standard Java collections
  - (Lists, Stacks, Queues, Sets, Maps...)
Basic techniques for storing and manipulating data

▶ “Expanding arrays”
▶ Nodes and pointers/references
▶ Trees and recursion

How to use pre-made data structures

▶ Using standard Java collections
▶ (Lists, Stacks, Queues, Sets, Maps…)

Techniques for organizing code

▶ Refactoring, coding style
▶ Client vs implementer
Content

- Learn new techniques
- Learn how exactly data structures work
- How to precisely analyze algorithms
CSE 373

Content

- Learn new techniques
- Learn how exactly data structures work
- How to precisely analyze algorithms

Core skills

- Design decisions, tradeoffs, and critical thinking
- Abstraction and implementation
- Communication: being able to justify your decisions
Content

- Learn new techniques
- Learn how exactly data structures work
- How to precisely analyze algorithms

Core skills

- Design decisions, tradeoffs, and critical thinking
- Abstraction and implementation
- Communication: being able to justify your decisions

Incidental skills

- Debugging and testing
- Exposure to tools used in industry
Course roadmap

- Week 1: Review of lists, stacks, and queues; misc Java tidbits
- Week 2: How to (precisely!) analyze code
- Week 3-5: Dictionaries (aka Maps) and Sets
- Week 6: Divide and conquer, sorting
- Week 7-9: Graphs and graph algorithms
- Week 10: Other interesting material
Definitions

Abstract Data Type (ADT)
A (mathematical) description of a “thing” with a set of supported operations and how they ought to behave.
Abstract Data Type (ADT)

A (mathematical) description of a "thing" with a set of supported operations and how they ought to behave.
What is a Stack?

A stack stores information in first-in, last-out order (like a deck of cards!)
A stack stores information in first-in, last-out order (like a deck of cards!)

It should support the following operations:

- **push**: add an item to the top of the stack
- **peek**: return (w/o removing) the top of the stack (if not empty)
- **pop**: remove and return the top of the stack (if not empty)
- **size**: return the number of elements in the stack
A stack stores information in first-in, last-out order (like a deck of cards!)

It should support the following operations:

- **push**: add an item to the top of the stack
- **peek**: return (w/o removing) the top of the stack (if not empty)
- **pop**: remove and return the top of the stack (if not empty)
- **size**: return the number of elements in the stack

This is the **Stack ADT**.
What is a Stack?

A stack stores information in first-in, last-out order (like a deck of cards!)

It should support the following operations:

- **push**: add an item to the top of the stack
- **peek**: return (w/o removing) the top of the stack (if not empty)
- **pop**: remove and return the top of the stack (if not empty)
- **size**: return the number of elements in the stack

This is the **Stack ADT**.
### Data structure

A specific way of organizing data and an associated family of algorithms that are used to implement an ADT
How do we implement a stack?

Internal data a stack needs to keep track of:

- **items**: An array containing our data
- **numItems**: An int containing the number of items in the stack

Algorithms:

- **push**:
  - If `numItems == items.length`, create a new array double the length, copy all elements over, and store the new array. Add the new item at the `numItems`-th index and increase `numItems` by one

- **peek**:
  - If `numItems == 0`, crash. Otherwise, return the item at the `numItems`-th index.

- **pop**:
  - Call **peek** and get the item to return. Decrease `numItems` by one.

- **size**:
  - Return `numItems`

This is the ArrayStack data structure. An ArrayStack implements the Stack ADT.
How do we implement a stack?

- Internal data a stack needs to keep track of:

  - **items**: An array containing our data
  - **numItems**: An int containing the number of items in the stack

- Algorithms:
  - **push**:
    - If `numItems == items.length`, create a new array double the length, copy all elements over, and store the new array. Add the new item at the `numItems`-th index and increase `numItems` by one
  - **peek**:
    - If `numItems == 0`, crash. Otherwise, return the item at the `numItems`-th index.
  - **pop**:
    - Call `peek` and get the item to return. Decrease `numItems` by one.
  - **size**:

This is the ArrayStack data structure. An ArrayStack implements the Stack ADT.
How do we implement a stack?

- **Internal data a stack needs to keep track of:**
  - items: An array containing our data
  - numItems: An int containing the number of items in the stack

- **Algorithms:**
  - **push**: If numItems == items.length, create a new array double the length, copy all elements over, and store the new array. Add the new item at the numItems-th index and increase numItems by one.
  - **peek**: If numItems == 0, crash. Otherwise, return the item at the numItems-th index.
  - **pop**: Call peek and get the item to return. Decrease numItems by one.
  - **size**: Return numItems
How do we implement a stack?

- **Internal data a stack needs to keep track of:**
  - *items*: An array containing our data
  - *numItems*: An int containing the number of items in the stack

- **Algorithms:**
  - **push**: If `numItems == items.length`, create a new array double the length, copy all elements over, and store the new array. Add the new item at the `numItems`-th index and increase `numItems` by one.
  - **peek**: If `numItems == 0`, crash. Otherwise, return the item at the `numItems`-th index.
  - **pop**: Call `peek` and get the item to return. Decrease `numItems` by one.
  - **size**: Return `numItems`

This is the **ArrayStack data structure**. An ArrayStack implements the Stack ADT.
**Implementation of a data structure**

Is a *specific* implementation in a *specific* language

AKA a *concrete data structure* (CSE 373-specific term)
How do we implement a stack in Java?

```java
public class ArrayStack<T> {
    private T[] items;
    private int numItems;

    // Constructor omitted for space

    public void push(T item) {
        if (this.numItems == this.items.length) {
            T[] newItems = new T[this.items.length * 2];
            this.copyTo(this.items, newItems, this.items.length);
            this.items = newItems;
        }
        this.items[this.numItems] = item;
        this.numItems += 1;
    }

    private void copyTo(T[] src, T[] dst, int amount) {
        for (int i = 0; i < amount; i++) {
            dst[i] = src[i];
        }
    }
}
```
public class ArrayStack<T> {
    private T[] items;
    private int numItems;

    // Constructor omitted for space

    public void push(T item) {
        if (this.numItems == this.items.length) {
            T[] newItems = new T[this.items.length * 2];
            this.copyTo(this.items, newItems, this.items.length);
            this.items = newItems;
        }
        this.items[this.numItems] = item;
        this.numItems += 1;
    }

    private void copyTo(T[] src, T[] dst, int amount) {
        for (int i = 0; i < amount; i++) {
            dst[i] = src[i];
        }
    }
}
How do we implement a stack in Java?

```java
public T peek() {
    if (this.numItems == 0) {
        throw new IllegalStateException();
    }
    return this.items[this.numItems];
}

public T pop() {
    T out = this.peek();
    this.numItems -= 1;
    return out;
}

public int size() {
    return this.numItems;
}
```
How do we implement a stack in Java?

```java
public T peek() {
    if (this.numItems == 0) {
        throw new IllegalStateException();
    }
    return this.items[this.numItems];
}

public T pop() {
    T out = this.peek();
    this.numItems -= 1;
    return out;
}

public int size() {
    return this.numItems;
}
}
```

This is a **concrete implementation of ArrayStack in Java**.
What is this thing?

```java
public class ArrayStack<T> {
    private T[] items;
    private int numItems;

    public void push(T item) { ... }
    // ...
}
```
Previously, in CSE 143, if we wanted a stack of ints:

```java
class ArrayIntStack {
    private int[] items;
    private int numItems;

    public void push(int item) {
        // ...
    }
    // ...
}
```

If we wanted a stack of Strings:

```java
class ArrayStringStack {
    private String[] items;
    private int numItems;

    public void push(String item) {
        // ...
    }
    // ...
}
```

Rinse and repeat for each type.
Previously:

```java
public class ArrayStringStack {
    private String[] items;
    private int numItems;

    public void push(String item) { ... }
    // ...
}
```
Previously:

```java
public class ArrayStringStack {
    private String[] items;
    private int numItems;

    public void push(String item) {
        // ...
    }
}
```

Using generics:

```java
public class ArrayStack<T> {
    private T[] items;
    private int numItems;

    public void push(T item) {
        // ...
    }
}
```
Previously:

```java
public class ArrayStringStack {
    private String[] items;
    private int numItems;

    public void push(String item) { ... }
    // ...
}
```

Using generics:

```java
public class ArrayStack<T> {
    private T[] items;
    private int numItems;

    public void push(T item) { ... }
    // ...
}
```

In this class, we’ll keep things simple/handle the messiness for you
Overloading

- Link to overload form available this **Friday**
- Other registration questions? Email cse373@cs.washington.edu
- Forwards emails to the CSE advisors
- Note: I have no control over course enrollment
Projects and Homework

Policies

- Mix of **partner projects** and solo **written homework**
- Three late days (lose 20% per day if no late days left)
- No submissions accepted after 2 days
- All assignments due at 11:30pm

Grades

- 15%: Written assignments
- 45%: Partner projects
- 20%: Midterm
- 20%: Final

See syllabus for more details
## Projects and Homework

### Policies
- Mix of **partner projects** and solo **written homework**
- Three late days (lose 20% per day if no late days left)
- No submissions accepted after 2 days
- All assignments due at 11:30pm

### Grades
- 15%: Written assignments
- 45%: Partner projects
- 20%: Midterm
- 20%: Final

See syllabus for more details
### Policies regarding sharing work

1. Showing other students your code or written work is **not** ok.
2. Do not publicly publish your projects or homework (we want to reuse these assignments).
## Academic honesty

### Policies regarding sharing work

1. Showing other students your code or written work is **not** ok.
2. Do not publicly publish your projects or homework (we want to reuse these assignments).

### Policies on discussion and collaboration

1. Discussing ideas on a high level **is ok**.
2. Rule-of-thumb: If you’re taking notes/taking photos during group discussions, you’re over-sharing.
## Getting help

### Course staff

- Piazza (Q&A forum)
- Office hours (see course website)
### Getting help

<table>
<thead>
<tr>
<th>Course staff</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ Piazza (Q&amp;A forum)</td>
</tr>
<tr>
<td>▶ Office hours (see course website)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ Lecture slides (posted after class)</td>
</tr>
<tr>
<td>▶ Panopto videos (posted after some delay)</td>
</tr>
<tr>
<td>▶ “Resources” section on course website</td>
</tr>
<tr>
<td>▶ Optional textbook: <em>Data Structures and Algorithms Analysis in Java</em>, 3rd edition, Weiss</td>
</tr>
</tbody>
</table>
Course survey, due Friday, Jan 5 at 11:30pm

Link: https://goo.gl/KNuQL1

(Link also available on course website)
Project 1

Full spec will be posted on class website later today

- Deliverables:
  - Implement a doubly-linked list and a dictionary (aka a map)
  - Implement a graphing calculator
  - Do a writeup
  - Extra credit: extend your calculator and implement a programming language

This is a partner project:
- Fri, Jan 5, 11:30pm: find a partner, fill out form
- If you really want to work solo, email me by tonight and explain why

Timeline: two week project
- Wed, Jan 10, 11:30pm: part 1 due
- Wed, Jan 17, 11:30pm: part 2 due
Project 1

Full spec will be posted on class website later today

- Deliverables:
  - Implement a doubly-linked list and a dictionary (aka a map)
  - Implement a graphing calculator
  - Do a writeup
  - Extra credit: extend your calculator and implement a programming language

- This is a partner project:
  - Fri, Jan 5, 11:30pm: find a partner, fill out form
  - If you really want to work solo, email me by tonight and explain why
Project 1

Full spec will be posted on class website later today

- Deliverables:
  - Implement a doubly-linked list and a dictionary (aka a map)
  - Implement a graphing calculator
  - Do a writeup
  - Extra credit: extend your calculator and implement a programming language

- This is a partner project:
  - Fri, Jan 5, 11:30pm: find a partner, fill out form
  - If you really want to work solo, email me by tonight and explain why

- Timeline: two week project
  - Wed, Jan 10, 11:30pm: part 1 due
  - Wed, Jan 17, 11:30pm: part 2 due
Summary: ADTs and data structures

Abstract Data Type (ADT)

- A (mathematical) description of a "thing" with a set of supported operations and how they ought to behave
## Summary: ADTs and data structures

<table>
<thead>
<tr>
<th>Abstract Data Type (ADT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ A (mathematical) description of a &quot;thing&quot; with a set of supported operations and how they ought to behave</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>▶ A specific bundle of data and family of algorithms that implements an ADT</td>
</tr>
</tbody>
</table>
## Summary: ADTs and data structures

### Abstract Data Type (ADT)
- A (mathematical) description of a "thing" with a set of supported operations and how they ought to behave

### Data structure
- A specific bundle of data and family of algorithms that implements an ADT

### Implementation of a data structure
- Is a *specific* implementation in a *specific* language
- AKA a *concrete data structure* (CSE 373-specific term)
Summary: TODO list

Today:

▶ Skim through syllabus
▶ Make sure you’re signed up on Piazza

By Friday, Jan 5:

▶ Course survey
▶ Look at project 1 spec and finish the setup instructions
▶ Find a partner and fill out form

By Wednesday, Jan 10:

▶ Project 1 part 1 due

By Wednesday, Jan 17:

▶ Project 1 part 2 due