CSCI 132:
Basic Data Structures and Algorithms

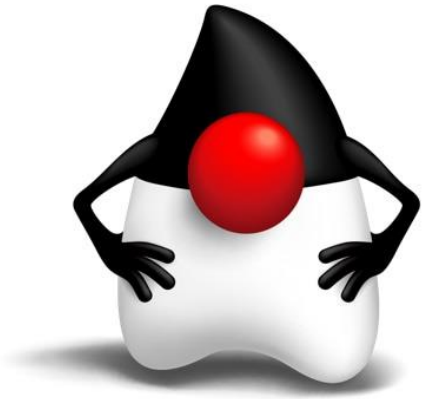
Final Exam Review, Course Conclusion

Reese Pearsall

Spring 2025

Announcements

- Program 5 due **Sunday**
- Final Exam on Monday (5/5) at **2:00 PM – 3:50 PM** in our normal classroom
- Rubber duck screenshot due tonight



Duke

Meatball wishes you good luck on your final exams

The Four Pillars of Object-Oriented Programming (OOP) are the fundamental principles to building modular, re-usable, and maintainable software

- Polymorphism
- Abstraction
- Encapsulation
- Inheritance

Polymorphism is the ability of a class to provide different implementations of a method, depending on the type of object

It's all about being able to reference an object with different types to achieve shared behaviors

"The ability for an object to take many forms"

```
Lion simba = new Lion("African Lion", 400.0, "Africa", 25000, 25);
Bird private = new Bird("Adelie penguin", 10, "Antarctica", 10000000, 15);

simba.makeSound();
private.makeSound();
```

The `makeSound()` method does something different for each object

Polymorphism is the ability of a class to provide different implementations of a method, depending on the type of object

It's all about being able to reference an object with different types to achieve shared behaviors

"The ability for an object to take many forms"

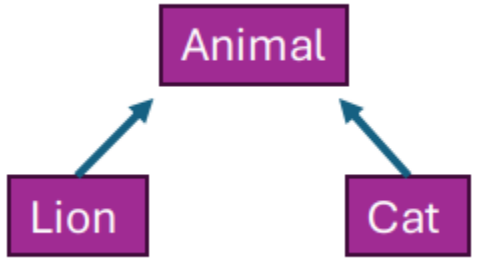
```
Lion simba = new Lion("African Lion", 400.0, "Africa", 25000, 25);
Bird private = new Bird("Adelie penguin", 10, "Antarctica", 10000000, 15);

simba.makeSound();
private.makeSound();
```

We can also treat the `simba` reference variable as an Animal, since Lion inherits from Animal

```
Animal simba = new Lion();
Animal meatball = new Cat();

Animal[] myAnimalArray = {simba, meatball};
```



Polymorphism is the ability of a class to provide different implementations of a method, depending on the type of object

It's all about being able to reference an object with different types to achieve shared behaviors

"The ability for an object to take many forms"

```
LinkedList<String> mylist = new LinkedList<String>();
```

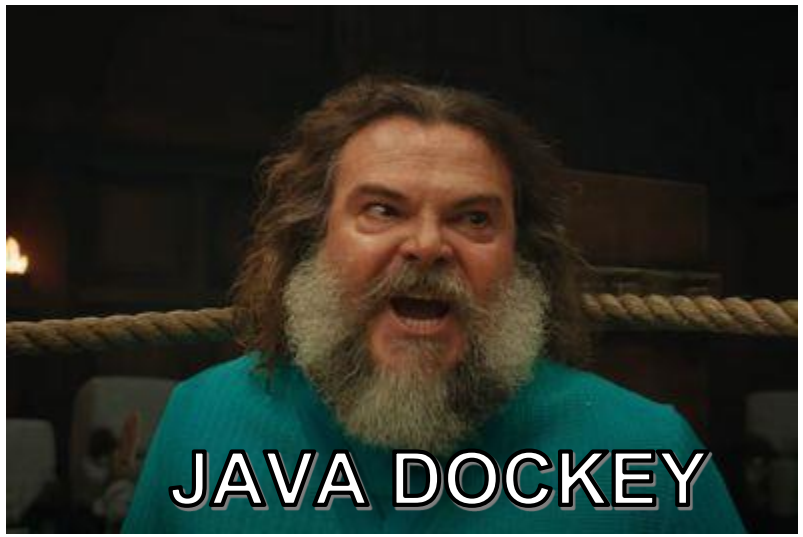
```
List<String> mylist = new LinkedList<String>();
```

We can reference `mylist` as just a `List`

Polymorphism is the ability of a class to provide different implementations of a method, depending on the type of object

It's all about being able to reference an object with different types to achieve shared behaviors

"The ability for an object to take many forms"




```
LinkedList<String> mylist = new LinkedList<String>();
```

```
List<String> mylist = new LinkedList<String>();
```

We can reference `mylist` as just a `List`

```
public class LinkedList<E>
extends AbstractSequentialList<E>
implements List<E>, Deque<E>, Cloneable, Serializable
```

```
AttributeList x = new AttributeList();
```

```
ArrayList x = new AttributeList();
```

```
AttributeList x = new AttributeList();
```

```
List x = new AttributeList();
```

```
ArrayList x = new AttributeList();
```

```
AttributeList x = new AttributeList();
```

```
AbstractList x = new AttributeList();
```

```
List x = new AttributeList();
```

```
ArrayList x = new AttributeList();
```

```
AttributeList x = new AttributeList();
```

```
Collection x = new AttributeList();
```

```
AbstractList x = new AttributeList();
```

```
List x = new AttributeList();
```

```
ArrayList x = new AttributeList();
```

```
AttributeList x = new AttributeList();
```

```
Cloneable x = new AttributeList();
```

```
Collection x = new AttributeList();
```

```
AbstractList x = new AttributeList();
```

```
List x = new AttributeList();
```

```
ArrayList x = new AttributeList();
```

```
AttributeList x = new AttributeList();
```

```
AbstractCollection x = new AttributeList();
```

```
Cloneable x = new AttributeList();
```

```
Collection x = new AttributeList();
```

```
AbstractList x = new AttributeList();
```

```
List x = new AttributeList();
```

```
ArrayList x = new AttributeList();
```

```
AttributeList x = new AttributeList();
```

```
RandomAccess x = new AttributeList();
```

```
AbstractCollection x = new AttributeList();
```

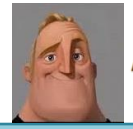
```
Cloneable x = new AttributeList();
```

```
Collection x = new AttributeList();
```

```
AbstractList x = new AttributeList();
```

```
List x = new AttributeList();
```

```
ArrayList x = new AttributeList();
```



```
AttributeList x = new AttributeList();
```

```
RandomAccess x = new AttributeList();
```

```
AbstractCollection x = new AttributeList();
```

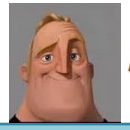
```
Cloneable x = new AttributeList();
```

```
Collection x = new AttributeList();
```

```
AbstractList x = new AttributeList();
```

```
List x = new AttributeList();
```

```
ArrayList x = new AttributeList();
```

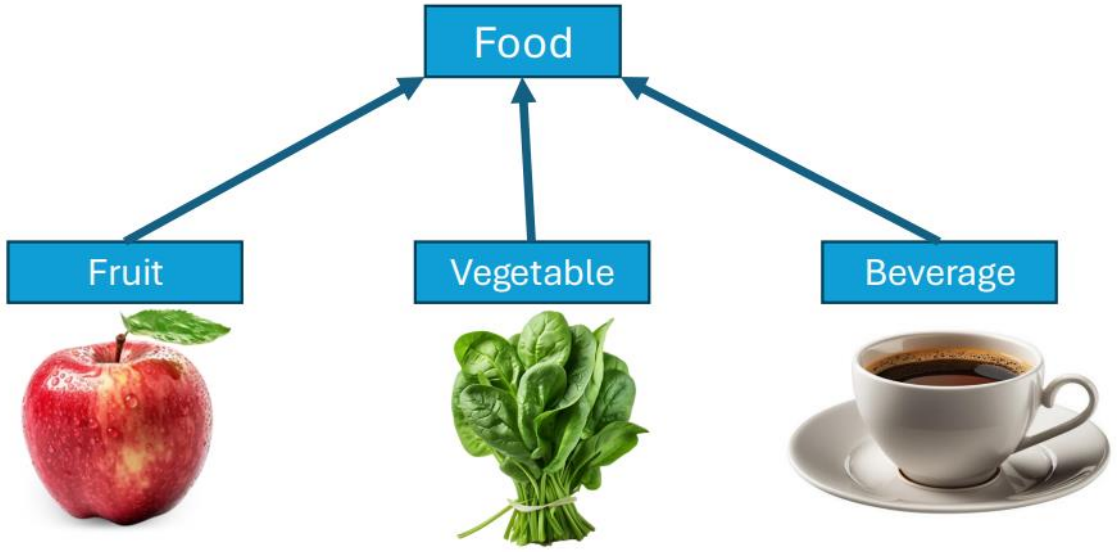


```
AttributeList x = new AttributeList();
```

```
Object x = new AttributeList();
```

Abstraction is the process of hiding certain details and showing only essential information to the user

Abstraction can be achieved with abstract classes and interfaces



simplifying complex systems by breaking them into more manageable parts.

`Food.java`:
Took on name, calories, and price so we didn't have to worry about them in subclasses

```
QueueLinkedList queue = new QueueLinkedList();
```

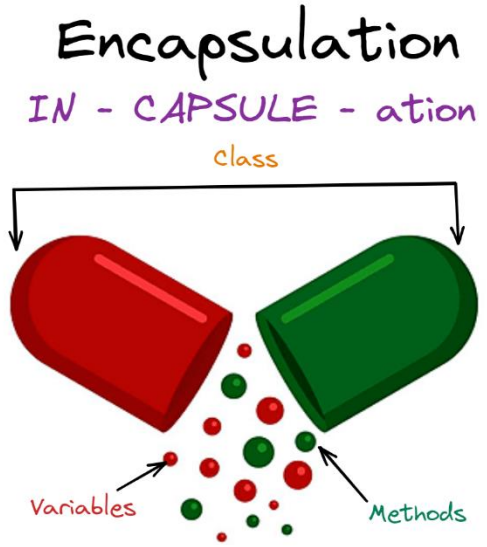
Users don't care about the implementation details of the Queue, so we hide it ("abstract it") behind a class

Encapsulation is the process of wrapping and data together into a single unit

Bundling of data and methods that operates on the data within a single unit, which is called a class

<u>Helpful for code organization</u>

- Where does it make sense to keep things together, and where are responsibilities separate?

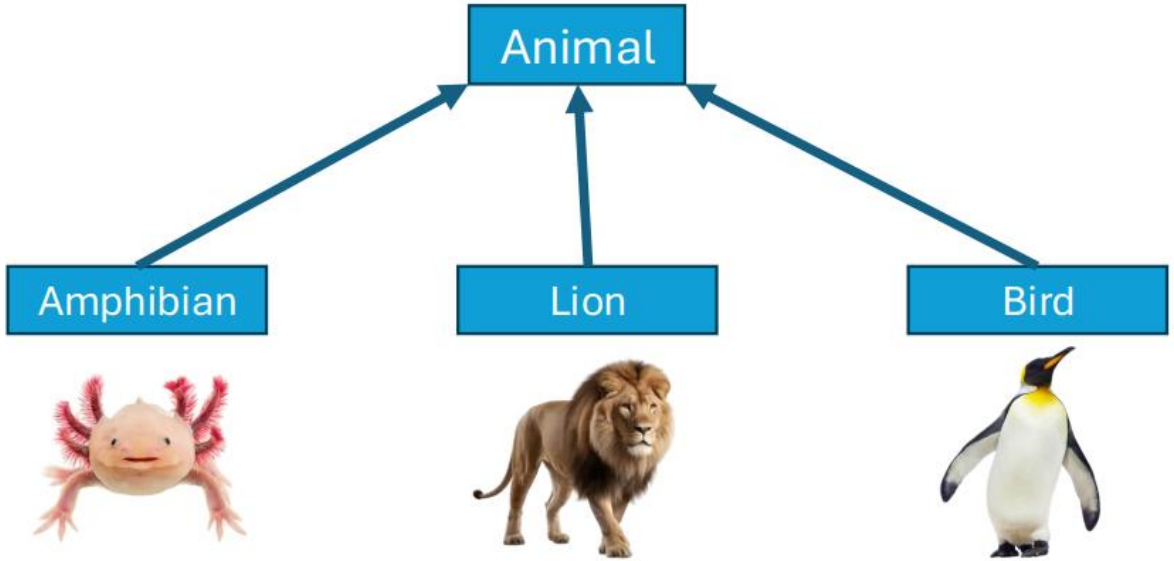


Inheritance is the process of one class inheriting properties and methods from another class

In Java, we have the **extends** keyword

Useful for code reusability

- Reused attributes and methods can be moved to a parent class to reduce redundancy



`final` keyword:
Creates a class that **can not** be extended

```
final class className{
}
```

Final Exam Logistics

In-person

Same format/rules as the midterm exam

- You can use notes, your IDE, lecture recordings, previous assignments, java documentation. No external resources

It will be a D2L quiz

- Bring your laptop

A little bit longer than the Midterm, but you have 2 hours this time ☺

This exam will focus on stuff from the second half of the semester, but some stuff from the first half may appear (cumulative)

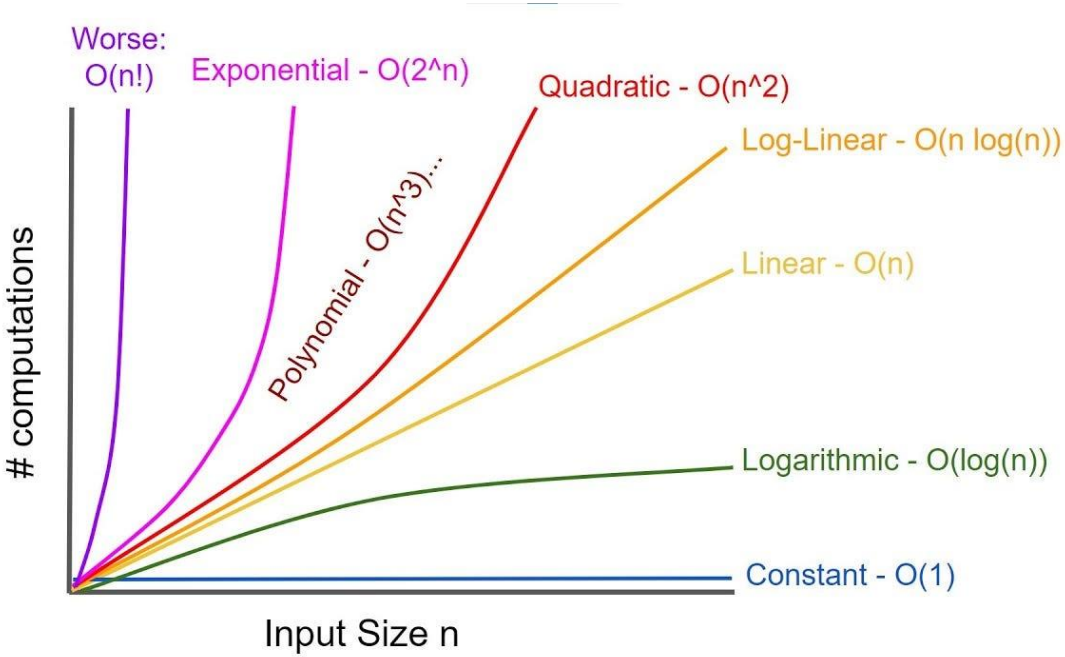
Types of questions

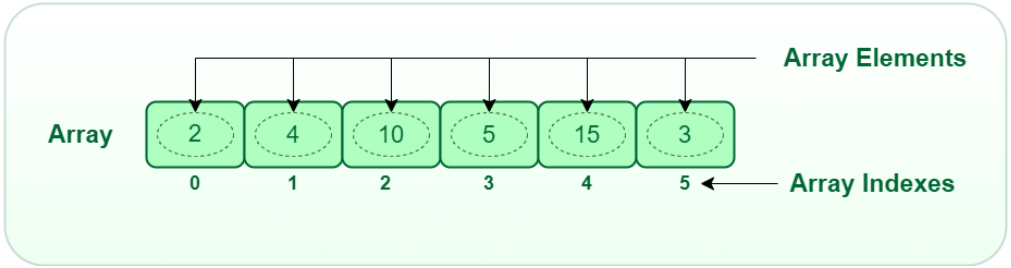
- Multiple Choice
- True/False
- Matching

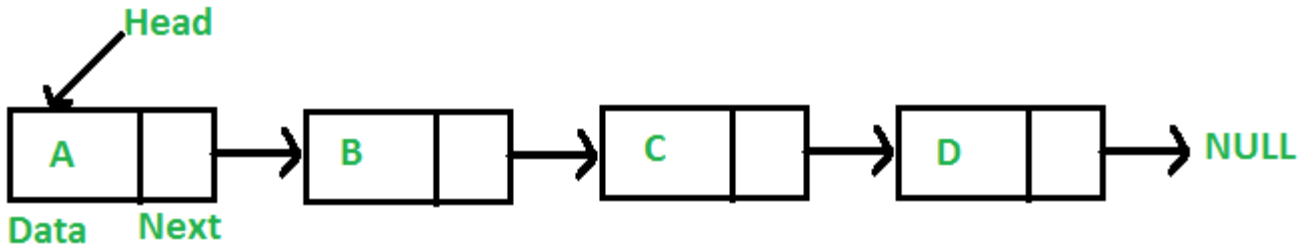
You won't have to write code, but you will need to look at code given to you

Things from the first half of the semester

- Arrays, Array Lists
- Linked Lists
- Running time analysis, Big-O notation
- Basic Java class structure

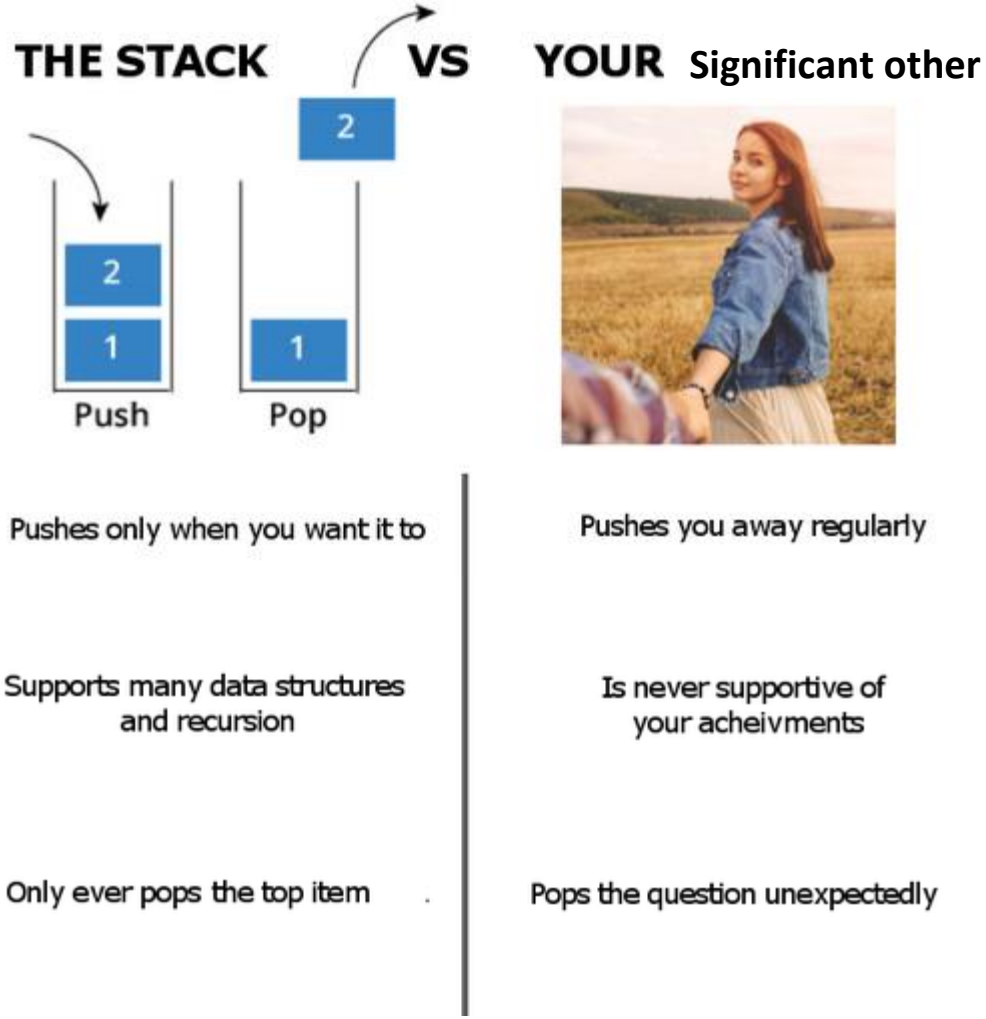






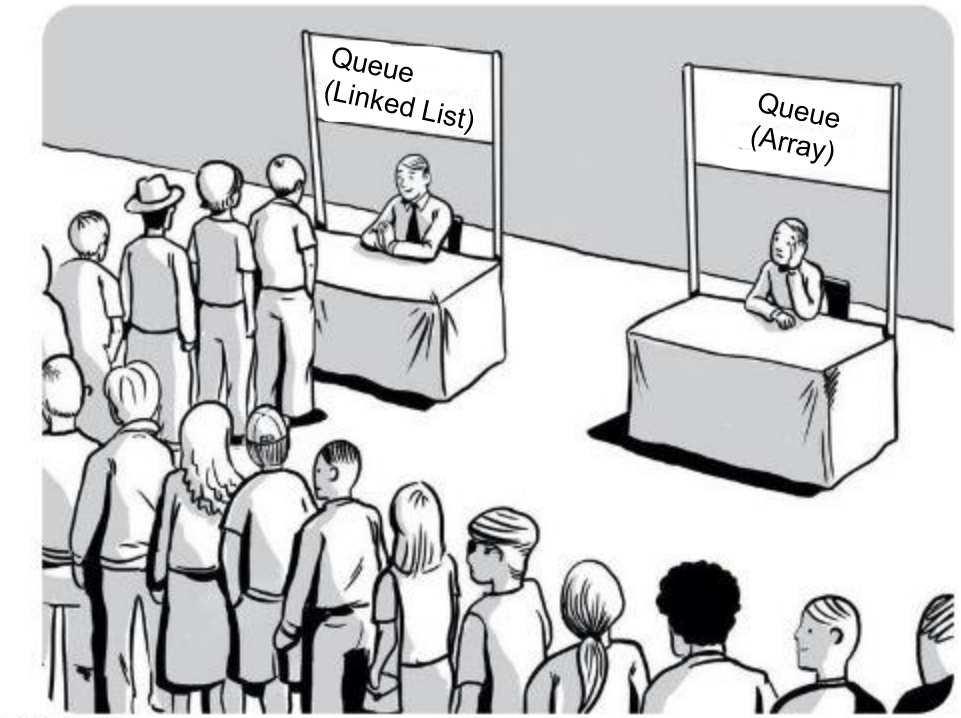
Stacks

- Be able to understand basic stack methods (push pop peek)
- Given code that utilizes a stack, be able to visualize and illustrate the contents of a stack
- Know the running time of stack operations



Queues

- Be able to understand basic queue methods (enqueue dequeue peek)
- Given code that utilizes a queue, be able to visualize and illustrate the contents of a stack
- Know the running time of queue operations
- Know what a Priority Queue is



Recursion

- Given a basic recursion function, derive the output and number of recursive calls made
- Understand how to calculate the running time of a recursive algorithm
- Understand limitations/benefits of recursion

Sorting

- Bubble sort, selection sort, merge sort, quick sort
- Be able to describe/illustrate the steps of these sorting algorithms
- Know the running time for each sorting algorithm
- Know which ones are efficient/not efficient
- I may ask you a question about some of the obscure sorting algorithms we discussed

Interviewer: Asks me a sorting algorithm

Nervous me:

Searching

- Understand the differences between linear search and binary search
- Understand the running times of those algorithms
- Be able to look at code for linear search and binary search and understand what is happening

Special Topics

- Java Generics
- Software Testing
- OOP Principles

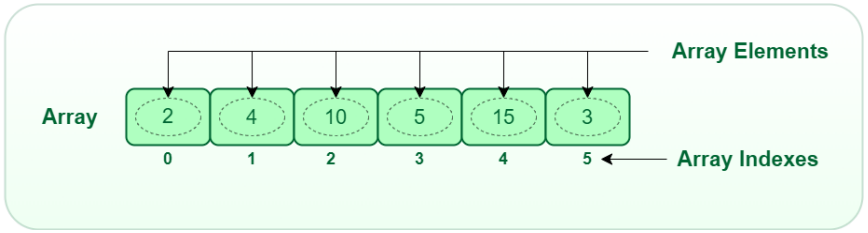
Final Exam Study Guide

Course Goals

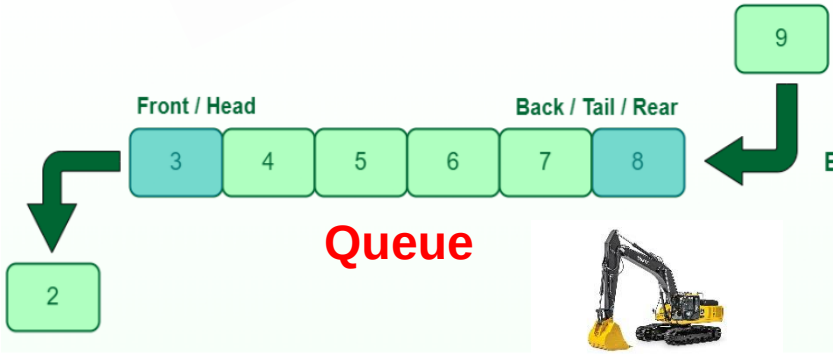
- Design and Implement programs of simple and moderate complexity in Java
- Explain the concept of an ADT
- Understand and implement basic data structures: Linked lists, stacks, and queues
- Given a simple algorithm, determine the time complexity using Big-O notation
- Understand basic searching and sorting algorithms and their runtime
- Understand how recursion works, be able to analyze recursion runtime, and be able to implement recursion in a program
- Be able to debug programs and become an independent problem solver

Takeaways

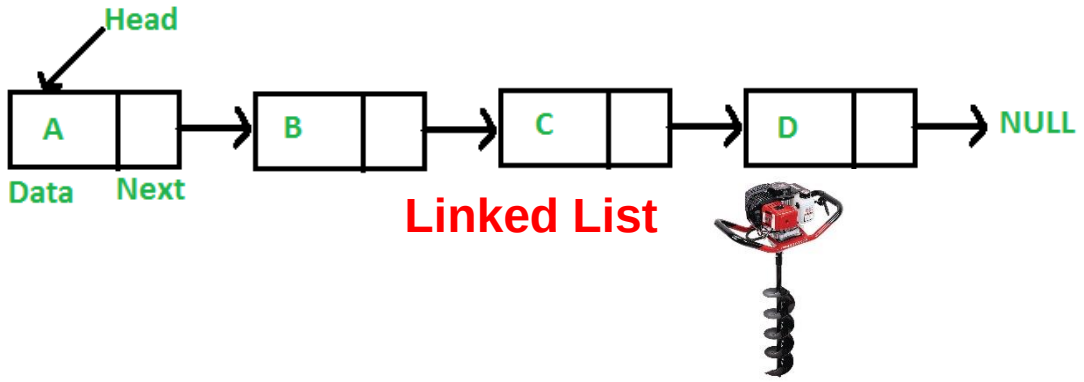
We have different data structures that handle data differently. There are **tradeoffs** between using these data structures



Arrays

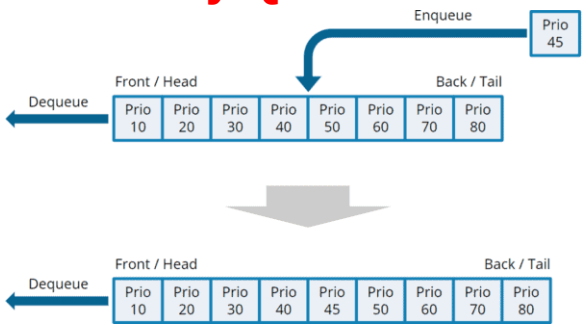


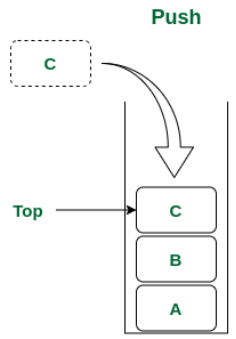
Queue

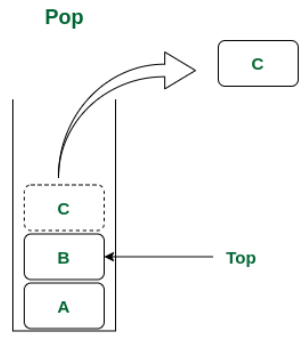


Linked List

Priority Queue







Stack

Given a problem, you should be able to identify a good candidate for a data structure and provide a justification

Takeaways

- There be many different types of algorithms. Every algorithm has a running time, which is important to be aware of
- The algorithm you select is important. It can be the difference between your program finishing in 1 second, or you program *never finishing at all*
- The data structure you select is important. When deciding which data structure to use, you should have a reason to back it up
- We have methods for measuring the efficiency of some algorithm (big-O notation), but they are not perfect.
- When you write an algorithm, you should be able to broadly describe the effectiveness and efficiency of it
- Sorting is a very valuable operation to do on a dataset

My Goals for you

Get you comfortable with writing basic Java programs

Give you a good toolset that can help you solve a variety of problems (Data Structures)

Give you techniques and methods for solving a variety of problems (Algorithms)

Give you the skills to analyze the algorithms that you write (Big-O notation)

Learn how to sort things (shoutout to bogo sort)

Thank You!

This class has been fun to teach for us to teach. I understand that there were certain parts that were not very exciting, but I appreciate you being in this class and participating

I hope you enjoyed this class, and I hope the stuff you learned will be helpful in your career/future classes

If I can be of assistance to you for anything in the future (reference, advising, support), please let me know!

I will see most of you again next semester ☺

Connect with me on LinkedIn!

Reese Pearsall (He/Him)
Instructor at Montana State University
Bozeman, Montana, United States · Contact info

3 months of no java coding