To Do

Sunday, November 9, 2014 7:02 PM

- Resume review
- Behavioral story prep

Workiva Interview

Thursday, November 6, 2014 9:31 PM

- Overall
    - B+ effort
- Positives
    - Technical answers
    - Rapport with interviewers
- Negatives
    - Be more filtered/slightly less honest with behavioral answers
- Learned
    - Bring some protein bars for the afternoon!

Amazon Info

Friday, November 07, 2014 9:42 AM

Why Amazon?

- Products
  - Amazon Prime
  - Kindle
  - Amazon Prime Video
- Projects that matter
  - I want to be part of something bigger than myself
  - I want my work to make a difference in people's lives
  - I want to work on projects I can be proud of an personally invested in
  - I want to see the impact of my work
- People
  - Smart
  - Talented
  - Motivated
- Create the Future
  - I want to play a small part in created in the future, which I believe Amazon is doing
- Move to Seattle

Leadership Principles

- Customer obsession
- Frugality
- Invent and simplify
- Bias for action
- Hire and develop the best
- Are right, a lot
- Ownership
- Vocally self-critical
- Think big
- Dive deep
- Earn true of others
- Insist on the highest standards
- Deliver results
- Have a backbone

Experience & Projects

Tuesday, November 11, 2014 12:27 PM

Xpanxion

- Overview
    - Currently a software engineer at Xpanxion, a contracting firm
    - Working on an corporate website
    - ASP.NET MVC
    - Got current position because of timing and student loans
- Project
    - ASP.NET MVC framework
    - Bootstrap & AngularJS
    - Corporate website
    - Agile Scrum team
- Why I'd like to leave
    - I don't find the project challenging
    - I don't feel I'm learning, improving, or being pushed to get better
    - No career path
    - Overqualified
- What I would like in a position
    - Work on cool projects that I am passionate about
    - Work with smart, talented, and motivated colleagues
    - Come to work every day to work hard and learn
    - Work on projects that actually matter and that I can be proud of

Microsoft

- Overview
    - Program management intern, summer 2013
    - Chose PM because I wanted to get more well-rounded experience and learn more about software development practices
- Project
    - Cloud deployment analytics project
    - Objective: convert internal logs into usable metrics for team, management, and stackholders
- Solution
    - Logs were uploaded to Microsoft's internal cloud service each night
    - Created PowerView reports for metrics
    - PowerView reports automatically updated with data
- Challenges
    - Finding documentation
        - Dev team needed documentation to implement project
        - Solution: coordinated with COSMOS team
    - Coordinating with Beijing-based team
        - The COSMOS team was located in Beijing
        - They had great information and were willing to help and answer questions, but we could only meet at night
        - Solution: Conference calls around 8 or 9pm Pacific time
    - Dealing with security issues
        - There were compliant issues with moving data "out" of the cage, where the logs were initially located
        - Solution: created user story to modify existing software to also store logs externally,

bypassing the issue

- What I learned/improved
  - Balancing priorities and requests from different stakeholders
  - Prioritizing long vs. short term objectives
- Additional info
  - Skype automated deployment team
  - Product was a series of PowerShell commandlets help automated deployment and updates for Lync Online
  - Product owner for this project, didn't code
  - Why I didn't go back: Air Force service commitment

John Deere

- Overview
  - Software engineering intern, summer 2012
  - Worked on embedded systems engine control project in C
- Project
  - Diesel engines are subject to increasing emissions regulations, requiring more and more resources from engine controller CPU
  - One function of engine controller was tracking engine position in cycle via IR sensor near gear teeth
  - Freescale board also had several co-processors, one of which could be used to track engine position in cycle
  - Objective: prototype software to track engine position on one of the coprocessors
- Previous solution
  - Engine tracking ran on main CPU
  - Interrupt for each gear tooth
  - Routine task using disproportional amount of CPU time
- My solution
  - Started with Freescale sample code and basic documentation
  - Created basic algorithm for tracking cam & crack shaft position
  - Tested with sample test data I created
- Challenges
  - Minimal documentation
  - Obscure system, so not much info available online
  - Overcame these challenges by putting in a lot of time, lots of debugging, and some help from manager and mentor
- What I learned/improved
  - Don't be afraid to ask for help if you're stuck
  - The importance of documentation!
  - Sometimes you just have to put in the time to finish a project
- Additional info
  - On Agile Scrum team, but individual project
  - Fun team, and project, but didn't want to work on tracker software

TreeType

- Overview
  - Onscreen keyboard for windows
  - Designed for people with disabilities, who can't use a physical keyboard
  - Code is 3/4 me
  - Worked on with friend from Microsoft
- Application

  - Users interact with the keyboard via a joystick or other pointing device
  - Keyboard layout designed to minimize movements required for keystrokes
  - User can toggle keyboard on/off to interact with PC normally
- Design
  - Tree-based layout, requiring log(n) movements to reach all n keys
  - Most common keys in the center
  - Cursor snaps back to center after each keystroke
  - Autocorrect located right next to center
  - Keyboard always on top when visible
  - Keyboard can be toggle on/off, so it gets out of the way when user not using it
- Specs
  - Windows 7, 8, and 8.1
  - Screen resolutions of 1024x768+
- Technology
  - WPF
  - UI design
  - Data structures
  - Spec generation
- Challenges/Bugs
  - Mouse hook & virtual keyboard
    - Had to use Win32 APIs, which have limited documentation on MSDN
  - Autocorrect
    - Implemented from scratch using top 10,000 words from Google and a trie
    - Each node of the trie has an array of top 10 words from that node

Route It

- Overview
  - Windows Phone navigation app
  - Reached top 8 of over 40 in company app competition, got to present in from of company leadership
  - Provides custom routing and time estimates based on user's past behavior, current weather, season, and time of day
- Application
  - Records how long user takes to navigate each segment of route
  - This segment, duration, weather, time of day, and date are stored in Azure DB
  - This data is used to improve routing and time estimates for this and other users
- Design
  - MVC
    - Model in Azure
    - Controller in Azure
    - View is actually application
- Challenges
  - Determining if user goes off-route
  - "Big data"
    - Determining when to use specific user data or global data

Study Buddy

- Overview
  - Android Project for school
  - Worked in team of two
  - Used Mercurial for version control
  - Collaborated over skype

- Application
  - Android application to help students find study sessions on campus
  - Users login in via Facebook and create, view, and join study sessions
  - Application showed users study sessions in their local area
- Design
  - MVC model
    - Android app as controller
    - MySQL SB for model
    - CakePHP for controller
  - CakePHP used to create REST API
    - CakePHP is an open source web application framework
    - Written in PHP
    - We could have easily made a mobile website or iPhone app
- Technology
  - Android
  - MySQL
  - cakePHP
  - REST
- Biggest challenge/bug
  - Facebook auth was tricky
  - We worked together, read the documentation, looked at samples, StackOverflow, and trial and error

Resume

Tuesday, November 11, 2014 12:25 PM

HIGHLIGHTS

- Graduated Summa Cum Laude in Computer Engineering from Iowa State in May 2014
- GPA: 3.90 GMAT: 730
- Interned at Microsoft and John Deere
- Software Engineer at Xpanxion, LLC

EXPERIENCE

Xpanxion, LLC. Ames, Iowa May 2014 – Present
Software Engineer

- Developed client-facing web views using HTML, CSS, AJAX, and JQuery for Hospital Corporation of America's custom 'idea management' software, enabling a dynamic and smooth user experience
- Implemented advanced business logic in C#, improved database schema using Microsoft's Entity framework
- Advised team with sprint planning, counseled clients during specification phase to insure on-time delivery of user stories

Microsoft Corporation Redmond, Washington May 2013 – August 2013
Program Management Intern

- Designed a service and dashboard to monitor crucial metrics for an automated deployment system, allowing production team to quickly pinpoint faults
- Negotiated with partner teams to resolve critical dependencies, allowing project to be completed ahead of schedule
- Created an innovative, navigation-based Windows Phone app in C# demonstrating the mapping, location, data mining and cloud services capabilities of the platform; reached the final eight in a company-wide app competition

John Deere, Inc. Waterloo, Iowa May 2012 – August 2012
Software Engineering Intern

- Drove an exploratory embedded systems project allowing John Deere to utilize existing engine control hardware while meeting Final Tier 4 emission standards, creating a multi-million dollar savings opportunity
- Offloaded critical logic from the engine's strained central processor onto a secondary computational unit through analysis of the Freescale platform and extensive refactoring of C code

EDUCATION

Iowa State University, Ames, Iowa August 2009 – May 2014

- Bachelor of Science, Computer Engineering, Summa Cum Laude
- **GPA: 3.90/4.00**
- **GMAT: 730 (96th percentile)**
- Academic Projects
    - Developed an Android application allowing students to locate study sessions on campus, written in Java and utilizing CakePHP for the backend, utilizing Mercurial for team collaboration
    - Designed a 32-bit processor, fully implementing branching, jumping, and ALU operations

NOTABLES

- Programming Languages: primarily Java, C#/.NET, JavaScript/HTML/CSS, with exposure to others
- Designed and developed TreeType, an open-source onscreen keyboard for Windows with accessibility applications
- Air Force ROTC, Vice Wing Commander – ran committee meetings to ensure agreement between cadets and leadership, personally mentored cadets through personal and organizational tribulations
- Private pilot license – 54 flight hours

Questions for Interviewers

Wednesday, November 12, 2014 10:08 AM

Technical

- What is the major function of your team?
- What languages/technologies does your team use?
- What does your team use for version control?
- What development process does your team use?
- How does your team do code reviews?
- What OS does your team develop on?

Amazon

- What do you like most about working for Amazon?
- How would you descript the culture at Amazon?
- What is your typical workday like?
- Do you enjoy living in the Seattle area?

Selection Sort

Wednesday, November 5, 2014 7:09 PM

- Crappy O(n^2) sorting algorithm
- Array split in to sorted and unsorted sections
- Unsorted section initially contains 0 elements
- Iterate though unsorted section, find smallest element, place at beginning of unsorted section
- Increment sorted section size
- Repeat until sorted section length == array length

Performance

Case	Complexity
Average	O(n^2)
Best	O(n^2)
Worst	O(n^2)
Space	O(n)

```java
public static void selectionSort(int[] array)
{
    for(int i=0; i<array.length; i++)
    {
        int smallestIndex = 0;
        int smallestValue = Integer.MAX_VALUE;
        for(int j=i; j<array.length; j++)
        {
            if(array[j] < smallestValue)
            {
                smallestIndex = j;
                smallestValue = array[smallestIndex];
            }
        }
        swap(array,i,smallestIndex);
    }
}
private static void swap(int[] array, int first, int second)
{
    int temp = array[first];
    array[first] = array[second];
    array[second] = temp;
}
```

Insertion Sort

Wednesday, November 5, 2014 7:13 PM

- Crappy O(n^2) sorting algorithm

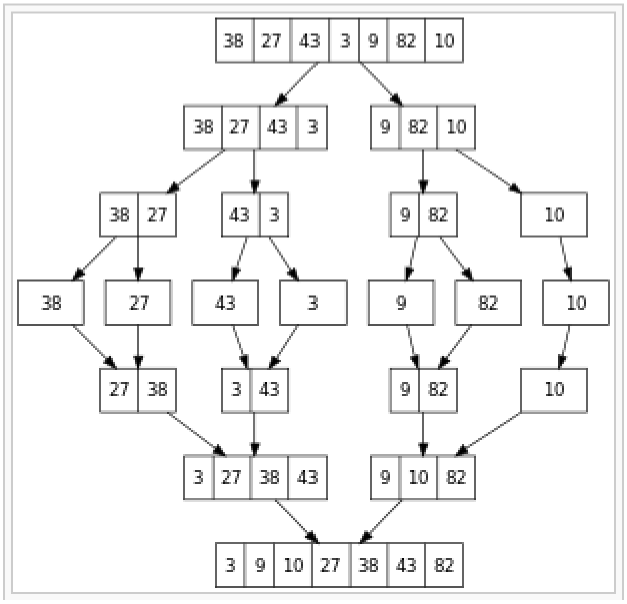
Performance

Case	Complexity
Average	O(n^2)
Best	O(n)
Worst	O(n^2)
Space	O(n)

Merge Sort

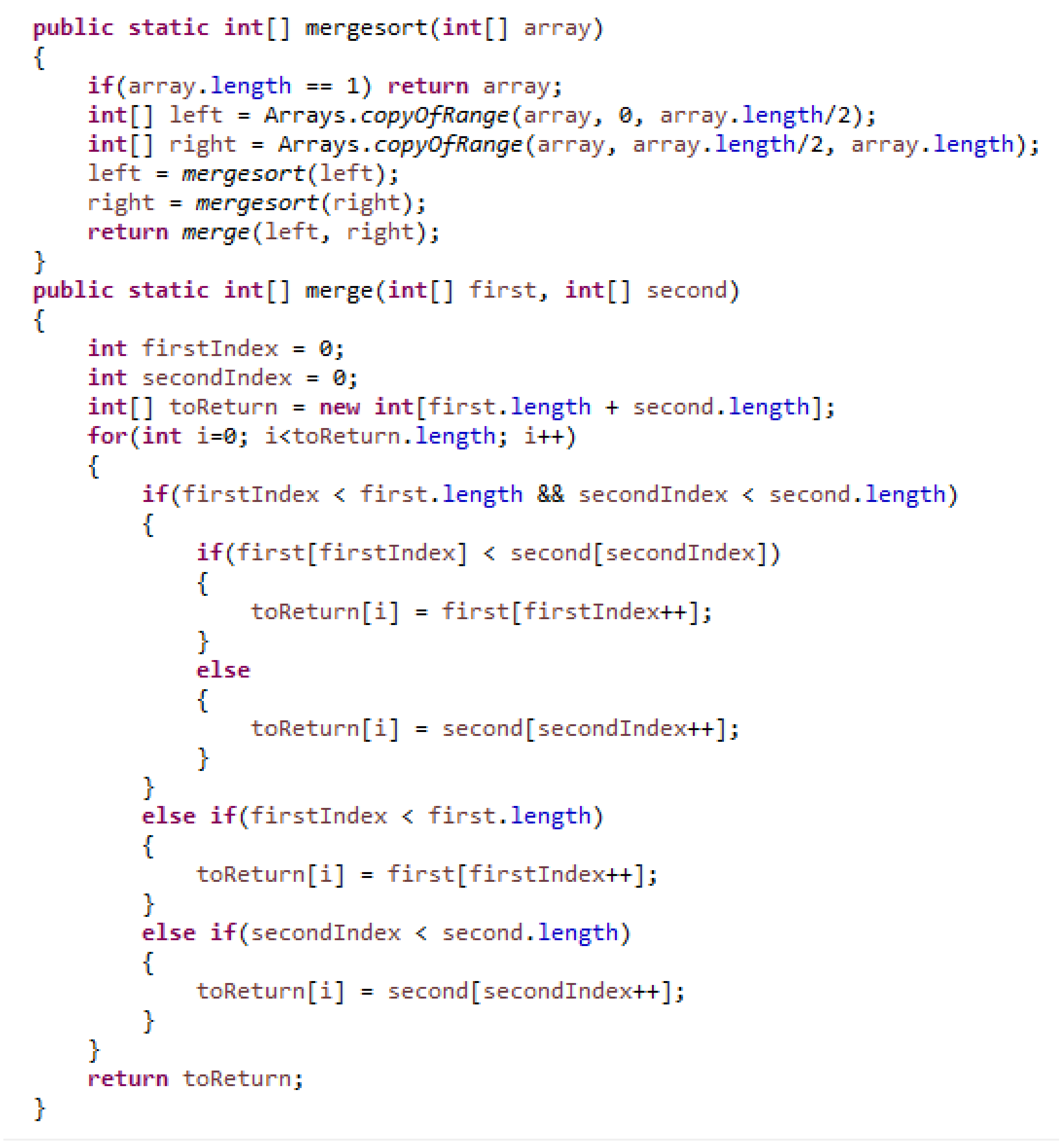
Wednesday, November 5, 2014 7:13 PM

- Generally produces stable sort
- Worst case time complexity is O(nlogn), but most modern libraries use quick sort because of caching



Performance

Case	Complexity
Average	O(nlogn)
Best	O(nlogn)
Worst	O(nlogn)
Space	O(n)



```java
public static int[] mergesort(int[] array)
{
    if(array.length == 1) return array;
    int[] left = Arrays.copyOfRange(array, 0, array.length/2);
    int[] right = Arrays.copyOfRange(array, array.length/2, array.length);
    left = mergesort(left);
    right = mergesort(right);
    return merge(left, right);
}
public static int[] merge(int[] first, int[] second)
{
    int firstIndex = 0;
    int secondIndex = 0;
    int[] toReturn = new int[first.length + second.length];
    for(int i=0; i<toReturn.length; i++)
    {
        if(firstIndex < first.length && secondIndex < second.length)
        {
            if(first[firstIndex] < second[secondIndex])
            {
                toReturn[i] = first[firstIndex++];
            }
            else
            {
                toReturn[i] = second[secondIndex++];
            }
        }
        else if(firstIndex < first.length)
        {
            toReturn[i] = first[firstIndex++];
        }
        else if(secondIndex < second.length)
        {
            toReturn[i] = second[secondIndex++];
        }
    }
    return toReturn;
}
```

Heap Sort

Wednesday, November 5, 2014 7:14 PM

- Advantage over quick sort is that is has a worst case time complexity of O(nlogn)

Performance

Case	Complexity
Average	O(nlogn)
Best	O(nlogn)
Worst	O(nlogn)
Space	O(1)

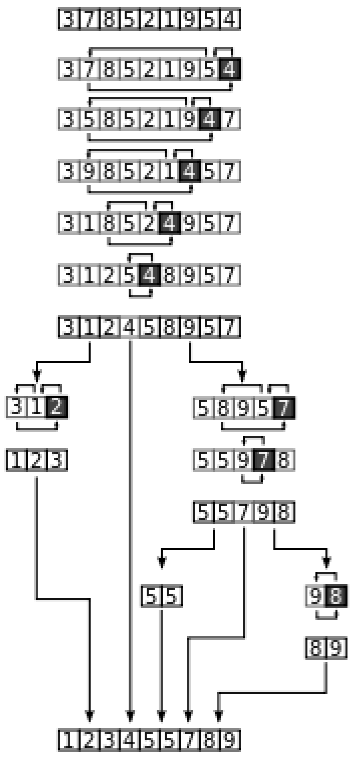
Quick Sort

Wednesday, November 5, 2014 7:14 PM

- Often faster than other O(nlogn) algorithms because its sequential and localized memory reference work well with a caching
- Unstable, so two elements with the same value may be swapped
- How it works:
    - Pick a pivot value
    - Partition array so all elements left of pivot are smaller, all element right of pivot are larger
    - Recursively call quick sort on sub arrays on either side of pivot
- Partition:
    - Pick partition value - pick median, because if you pick left or right most, you will get O(n^2) performance if array is already sorted
    - Move pivot to far right
    - From left to right - 1
        - If array[i] < pivot, swap
        - Increment index to be returned
    - Swap index to be returned and array[right]

Performance

Case	Complexity
Average	O(nlogn)
Best	O(nlogn)
Worst	O(n^2)
Space	O(n)



```java
public static void quicksort(int[] array, int lower, int upper)
{
    //base case to stop recursion
    if(lower < upper)
    {
        int pivot = partition(array, lower, upper);
        quicksort(array, lower, pivot - 1);
        quicksort(array, pivot + 1, upper);
    }
}
private static int partition(int[] array, int lower, int upper)
{
    //median to avoid O(n^2) performance if array is already sorted
    int pivotIndex = (upper+lower)/2;
    int pivotValue = array[pivotIndex];
    swap(array,pivotIndex,upper);
    int storeIndex = lower;
    for(int i=lower; i<upper; i++)
    {
        if(array[i] < pivotValue)
        {
            swap(array,i,storeIndex);
            storeIndex++;
        }
    }
    swap(array,storeIndex,upper);
    return storeIndex;

}
private static void swap(int[] array, int first, int second)
{
    int temp = array[first];
    array[first] = array[second];
    array[second] = temp;
}
```

Quick Select

Wednesday, November 5, 2014 7:14 PM

- Selection algorithm that can find nth element of array in average of O(n) time
- Based on quick sort partition
- Array doesn't have to be sorted!
- Partition is the same, except the pivot index is a parameter
- Also, your passing in n, the index you are looking for

Performance

Case	Complexity
Average	O(n)
Best	O(n)
Worst	O(n^2)

```java
public static int select(int[] array, int lower, int upper, int n)
{
    //base case, one element array
    if(lower == upper) return array[lower];
    int pivotIndex = (lower + upper) / 2;
    pivotIndex = partition(array,lower,upper,pivotIndex);
    int toReturn = 0;
    if(n == pivotIndex) toReturn = array[n];
    else if(n < pivotIndex) toReturn = select(array, lower, pivotIndex-1, n);
    else if(n > pivotIndex) toReturn = select(array, pivotIndex+1, upper, n);
    return toReturn;
}
private static int partition(int[] array, int lower, int upper, int pivotIndex)
{
    int pivotValue = array[pivotIndex];
    swap(array,pivotIndex,upper);
    int storeIndex = lower;
    for(int i=lower; i<upper; i++)
    {
        if(array[i] < pivotValue)
        {
            swap(array,i,storeIndex);
            storeIndex++;
        }
    }
    swap(array,storeIndex,upper);
    return storeIndex;

}
private static void swap(int[] array, int first, int second)
{
    int temp = array[first];
    array[first] = array[second];
    array[second] = temp;
}
```

Array

Wednesday, November 5, 2014 9:12 PM

- Continuous block of memory
- Can be used to back binary search trees, hashtables, and heaps
- Might require a lot of shifting to insert/delete element in the middle of array
- Could delete by swapping delete element with last, and "reducing" array size by 1
- Might have to dynamically expand array, like ArrayList does

Performance

Operation	Average	Worst
Space	O(n)	O(n)
Lookup	O(1)	O(1)
Insert	O(n)	O(n)
Delete	O(1)	O(n)

Implement heap/BST

- Parent at index I
- Left child at 2i
- Right child at 2i+1

Linked list

Wednesday, November 5, 2014 9:11 PM

Overview

- Each element has a "link" (pointer) to next (and possibly pervious) element
- Can be used to implement lists, stacks, and queues
- Could be doubly linked, circular, or have a "next" pointer

Performance

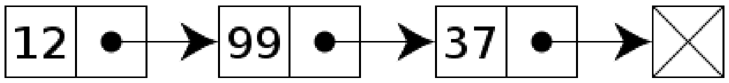
Operation	Average	Worst
Space	O(n)	O(n)
Lookup	O(n)	O(n)
Insert	O(n)	O(n)
Delete	O(n)	O(n)

Advantages

- Can organically expand/contract, so no reallocation like arrays
- Easy insert/delete after you find element
- Great for linear data

Disadvantages

- High overhead due to all the pointers
- Sequential access only, no random access
- O(n) insert and delete, because you have to find correct element



A linked list whose nodes contain two fields: an integer value and a link to the next node. The last node is linked to a terminator used to signify the end of the list.

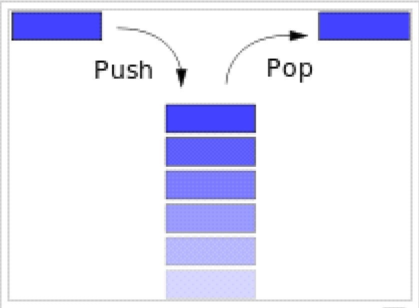
Java implementation here!

Stack and Queue

Wednesday, November 5, 2014 9:12 PM

Stack

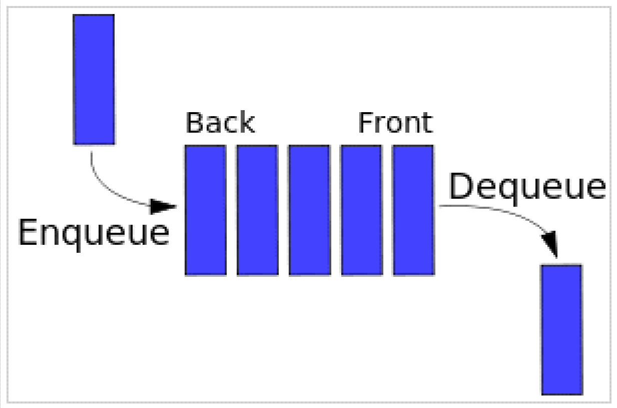
- First in, last out
- Push, pop, peek
- Like call stack or trays in the cafeteria
- Can easily be implemented with linked list
- Stack<T> class in Java



Simple representation of a stack

Queue

- First in, first out
- Enqueue, dequeue
- Like being in line at the bank
- Can easily be implemented with linked list
- Use LinkedList<T> with .add(T t) and .remove()



Representation of a Queue with FIFO (First In First Out) property

Java implementation here!

Hashtable

Wednesday, November 5, 2014 7:10 PM

Overview

- Great for storing and retrieving unordered data in constant time
- Backed by array
- Stores key, value pairs
- Key hashed to hash code % table size to get value address

Performance

Operation	Average	Worst
Space	O(n)	O(n)
Lookup	O(1)	O(n)
Insert	O(1)	O(n)
Delete	O(1)	O(n)

Hash functions

- Obviously need to be deterministic
- You would like a uniform distribution of hash values
- It's important to keep hash function secret, because otherwise someone could maliciously deliberately cause hash collusions

Collisions

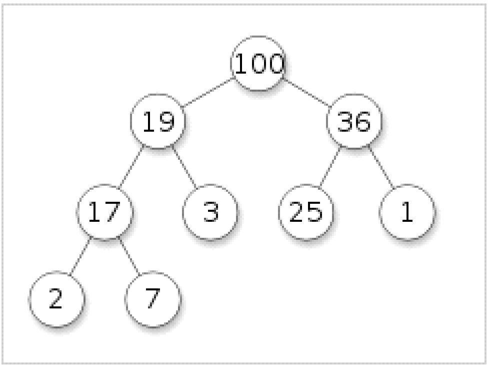
- A collision occurs when two keys hash to the same bucket
- There are two ways to handle this: open addressing and chaining
- For chaining, each bucket is a linked list
- Worst case scenario, all keys hash to the same bucket, and lookup is O(n)

Load factor & Resizing

- Load factor = number of elements / size of table
    - Load factor is also the average length of chain
- Want to keep load factor low for best lookup performance
- Load factor above predetermined constant, say 0.75, triggers resize
- Array is doubled
- Insert operation is atomized O(1) operation, if you divide the O(n) resize operation by total inserts

Heap

Wednesday, November 5, 2014 9:11 PM



Example of a complete binary max-heap with node keys being integers from 1 to 100

Overview

- Binary tree with largest/smallest element as root
- Complete binary tree
- No implied order between siblings
- Probably backed by array
- Good way to implement priority queue

Organization

- Parent index: i
- Left child index: 2i
- Right child index: 2i+1

Operations

- Heapify
    - Turn array into min or max heap
- Insert
    - Insert new element at last position
    - Percolate up
- Delete min/max
    - Remove root
    - Place last element at root
    - Percolate down

Java implementation here!

Binary search tree

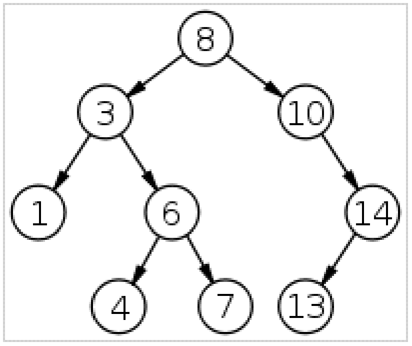
Wednesday, November 5, 2014 9:11 PM

Overview

- Dynamic data structure
- All nodes in left child smaller than parent, all nodes in right child larger than parent
- A balanced tree allows for O(logn) performance for loopup, insert, and delete
- **Duplicate nodes are not allowed!**
- If BST is backed by array, index i's left child is at 2i+1, and its right child is at 2i+2

Performance

Operation	Average	Worst
Space	O(n)	O(n)
Lookup	O(logn)	O(n)
Insert	O(logn)	O(n)
Delete	O(logn)	O(n)



A binary search tree of size 9 and depth 3, with root 8 and leaves 1, 4, 7 and 13

Advantages

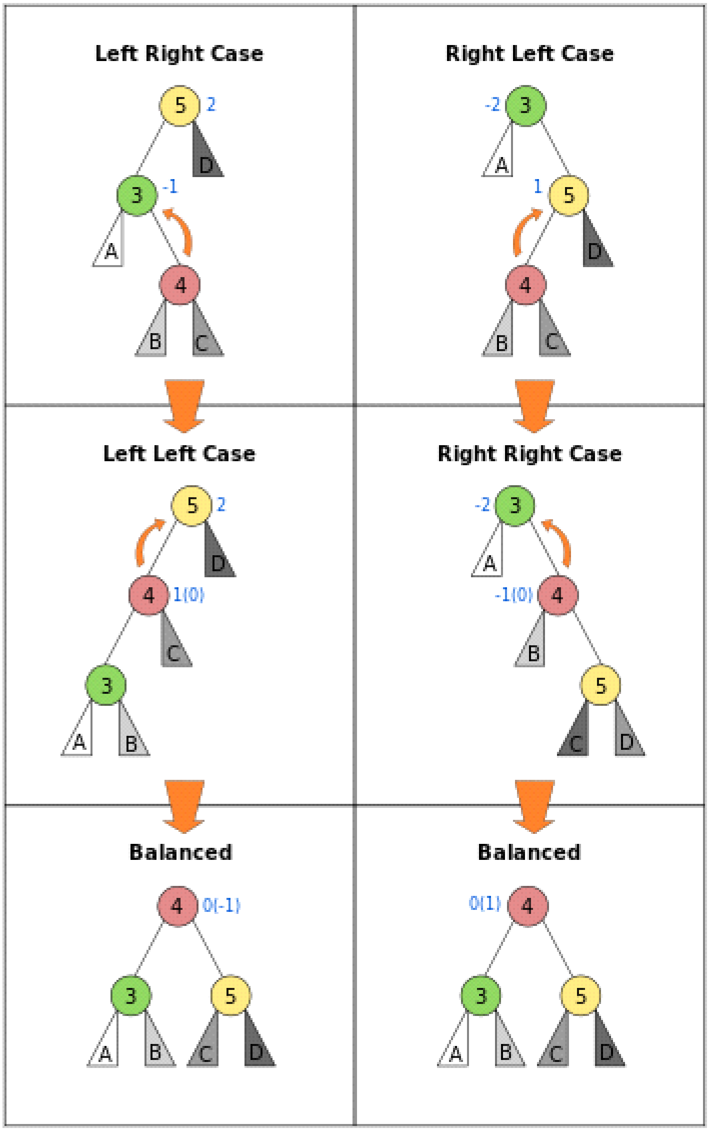
- O(logn) lookup, insert, and delete when balanced
- Simple implementation
- Dynamic data structure

Disadvantages

- You have to keep in balanced
- Lookup and insert take longer than hashtables - O(logn) vs. O(n)

AVL trees

- The height of sub trees must be <= 1
- Balance factor = left sub tree height - right sub tree height
- Delete node: take largest node of deleted node's left child, and put it where deleted node use to be



Pictorial description of how rotations rebalance an AVL tree. The numbered circles represent the nodes being rebalanced. The lettered triangles represent subtrees which are themselves balanced AVL trees. A blue number next to a node denotes possible balance factors (those in parentheses occurring only in case of deletion).

Searches

Sunday, November 9, 2014 12:34 PM

```
public class Node
{
    public int value;
    public Node left;
    public Node right;
}
```

Breadth-First Search

Note: this is for a binary tree. For a graph, you need to keep track of visited nodes

```
public static void BFS(Node root)
{
    LinkedList<Node> queue = new LinkedList<Node>();
    queue.add(root);
    while(!queue.isEmpty())
    {
        Node current = queue.remove();
        System.out.println(current.value);
        if(current.left != null) queue.add(current.left);
        if(current.right != null) queue.add(current.right);
    }

}
```

Depth-First Search

Note: this is an in-order transversal of a binary tree. There is also pre-order and post-order
Also, you can implement DFS iteratively with a stack.

```
public static void DFS(Node current)
{
    if(current.left != null) DFS(current.left);
    System.out.println(current.value);
    if(current.right != null) DFS(current.right);
}
```

B tree

Wednesday, November 5, 2014 9:13 PM

Trie

Wednesday, November 5, 2014 9:13 PM

Trees

Wednesday, November 5, 2014 9:22 PM

Graphs

Wednesday, November 5, 2014 9:22 PM

Service Oriented Architecture

Monday, November 10, 2014 10:30 AM

Overview

- Software architecture design pattern based on distinct pieces of software providing application functionality as a service to other applications, independent of vender, product, or technology
- Service is self-contains unit of functionality
    - Self-contained
    - "black box" to consumers of service
    - May be composed of other services
- REST is good example
    - GET, PUT, POST, DELETE

Map Reduce

Friday, November 07, 2014 10:14 AM

Overview

- Basically divide-and-conquer in parallel
- Problem is sub-divided and passed to worker nodes
- Workers find solution for their data subsets (Map)
- Results are sorted by key
- Works combine sub-solutions (Map)
- Map(k1,v1)->list(k2,v2)
- Reduce(k2,list(v2))->list(v3)
- Map and reduce in parallel

Example

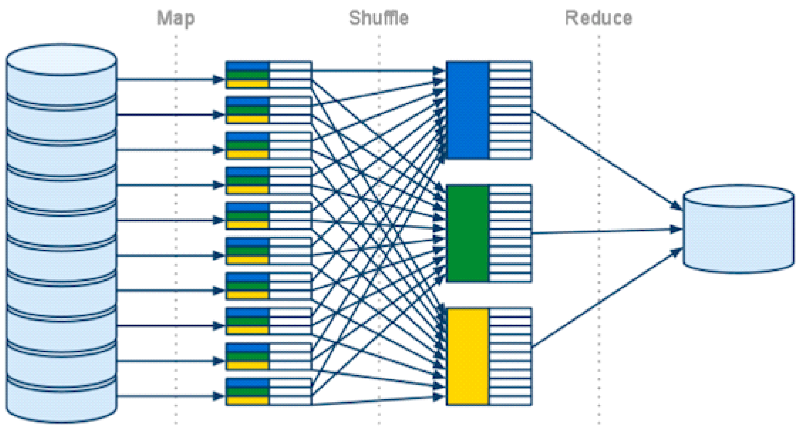
- Let's say we have a huge list of (key, value) pairs, key being a city, and the value being the recorded temperature
- We want to find the highest recorded temperature in the each city in the list
- Master node divides list into sub-lists, which are passed to worker nodes
- Worker nodes may further sub-divide sub-lists
- Each worker node finds the highest recorded temperature for each city in its sub-list Map(city, temperature)->list(city, highest recorded temp)
- During the shuffle step, the results of the map operation are sorted by key, in this case city
- Nodes combine lists to find highest recorded temperature for each city

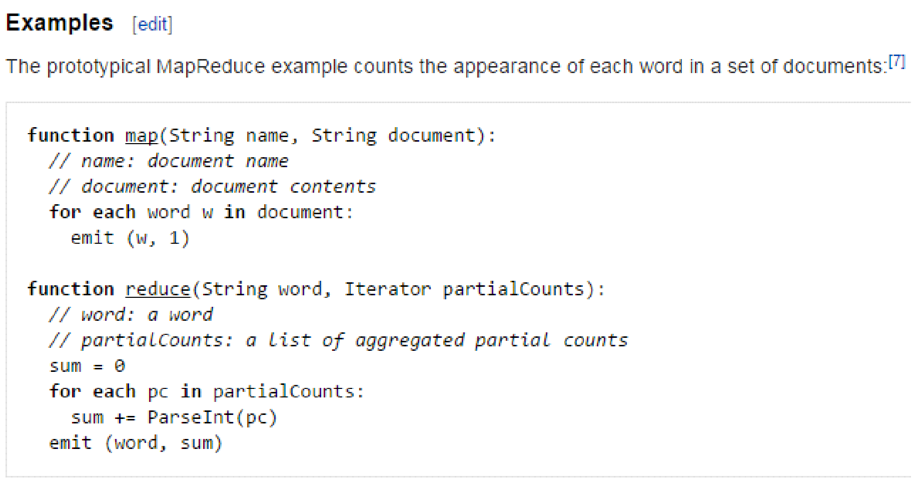
How it works

- **Map** - each worker node applies map() function to local data and writes output to temporary storage
- **Shuffle** - worker nodes redistribute data based on output keys of map() function, so all data belonging to one key goes to the same worker node
- **Reduce** - worker nodes process each group of output data in parallel

Hadoop

- Open source framework for distributed storage and distributed data processing
- Hadoop Distributed File System splits files into blocks (64 or 128MB) and distributes them amongst the nodes
  - Duplicate data for reliability
- Hadoop Map/Reduce is used to process data
  - Leverages data locality, so each node only processes data that is stored on it





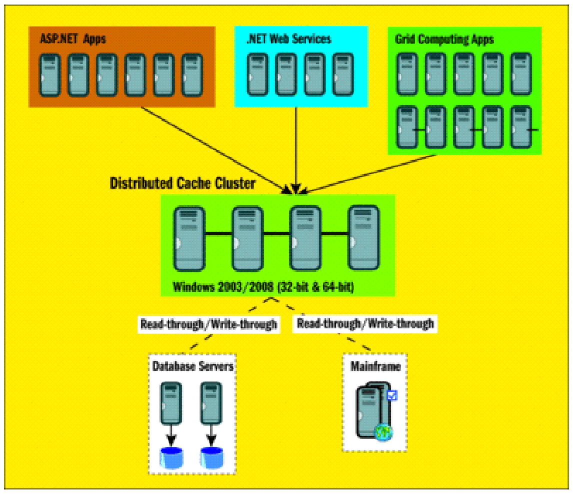
Examples [edit]

The prototypical MapReduce example counts the appearance of each word in a set of documents:[7]

```
function map(String name, String document):
  // name: document name
  // document: document contents
  for each word w in document:
    emit (w, 1)

function reduce(String word, Iterator partialCounts):
  // word: a word
  // partialCounts: a list of aggregated partial counts
  sum = 0
  for each pc in partialCounts:
    sum += ParseInt(pc)
  emit (word, sum)
```

Distributed Caching

Friday, November 07, 2014 10:14 AM


ASP.NET Apps
.NET Web Services
Grid Computing Apps
Distributed Cache Cluster
Windows 2003/2008 (32-bit & 64-bit)
Read-through/Write-through
Read-through/Write-through
Database Servers
Mainframe

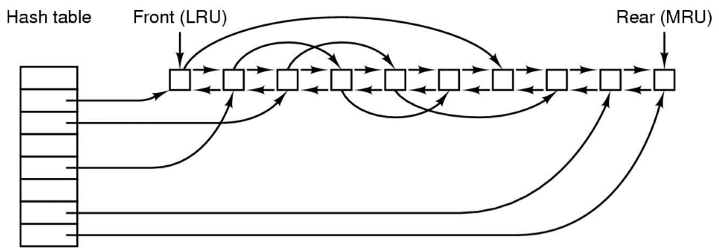
Overview

- Similar to local cache, but spans multiple machines
- Distributed cache cluster sits between resource (i.e. DB) and services requesting data
- Logically looks like one big cache
- Cache data probably resides in machines memory, not disk (bro, it's a cache, it has to be fast)

Writing Policies

- Write-through - write is done synchronously to both the cache and backing storages
- Write-back - initially, writing is only done to cache. Write to backing storage is postponed until cache block is about to be removed
  - More efficient
  - Issues if there is a problem with backing storage

Structure of the Cache

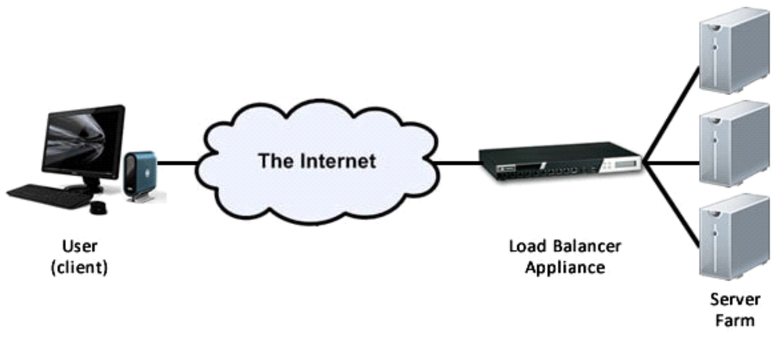


Caching Strategies

- Most efficient caching algorithm would be to always discard the data that will not be needed for the longest. There is no way to know this in advance in real life
- Least Recently Used
- Random Replacement

Load Balancing

Friday, November 07, 2014 10:14 AM



Overview

- Load balancing distributes workloads across multiple computing resources
- Objectives:
  - Optimize resource use
  - Maximize throughput
  - Minimize response time
  - Avoid overloading any single resource

Example

- A load balancer could be a software program that listens to a specific port, to which external clients connect
- The load balancer forwards request to one of the backend servers
- Backend server replies to load balancer, which replies to the client
- This also abstracts the backend servers from the client

General

Wednesday, November 5, 2014 7:10 PM

Lengths

- Array.length
- ArrayList.size()
- String.length()

Enums

```
public enum Day {
        SUNDAY,
        MONDAY,
        TUESDAY,
        WEDNESDAY,
        THURSDAY,
        FRIDAY,
        SATURDAY
}
```

How Java Works

- Java is compiled to Java bytecode which is ran on the Java Virtual Machine
- Memory management is handled by the garbage collector
    - When the garbage collector runs, it determines which objects have no references and frees their memory
- Java has a single process with multiple threads

Tips

Saturday, November 8, 2014 11:42 AM

- Dude, if you're just using a hash table for like <Integer, Boolean>, you could just use an array! It's initialized to 0, so used an array instead of a hash table unless the counts matter
- Never check floating point equality with ==
- Make sure to validate input and think of edge cases
- Basically every object has .equals() and .hashCode()
- Use LinkedList with .add() and .remove() for a queue!
- There is an actual stack class

Primitive Types

Wednesday, November 5, 2014 9:40 PM

Type	Discription
byte	1 byte int
short	2 byte int
int	4 byte int
long	8 byte int
float	32-bit floating point
double	64-bit floating point
boolean	True/false
char	16-bit unicode character

- Strings are objects!
- All objects initialize to null
- All primitives initialize as expected

Autoboxing/Unboxing

- Java compiler automatically converts primitive types to their associated objects and vise visa
- For example, int to Integer

String to primitive

- int - Integer.parseInt(String str)

Bitwise Operations

Wednesday, November 5, 2014 9:22 PM

Java Bitwise Operators

- ~ bitwise NOT
- & bitwise AND
- | bitwise OR
- ^ bitwise XOR
- << signed left shift
- >> signed right shift
- **>>> unsigned right shift**

Other Bases

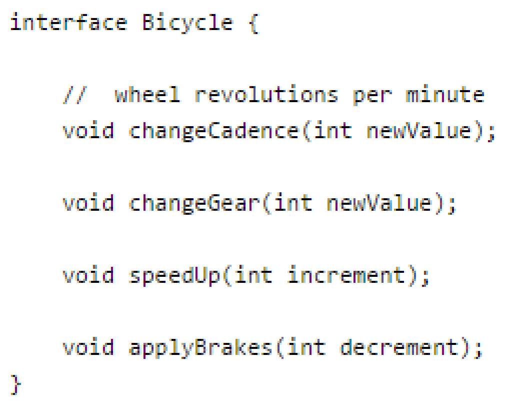
- Hex: 0x0000
- Binary: 0b0000

Inheritance

Wednesday, November 5, 2014 9:46 PM

Interface

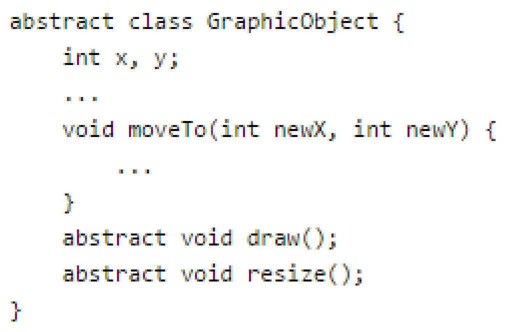
- Interfaces are like adjectives - they specify abilities, like comparable
- And implement multiple interfaces
- Interface - contract interface, class x that implements interface y will contain all of y's methods



```
interface Bicycle {

    //  wheel revolutions per minute
    void changeCadence(int newValue);

    void changeGear(int newValue);

    void speedUp(int increment);

    void applyBrakes(int decrement);
}
```

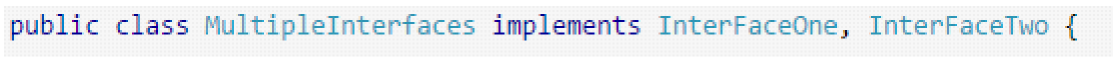
Abstract class

- Cannot instantiate an abstract class
- Super classes
  - Use @Override in subclasses
  - super(parameters) to class super class contractor
  - super.method() to class super class method
  - Final methods cannot be overridden by sub-classes
- Final methods cannot be overridden
- You can extend normal, non-abstract classes



```
abstract class GraphicObject {
    int x, y;
    ...
    void moveTo(int newX, int newY) {
        ...
    }
    abstract void draw();
    abstract void resize();
}
```

Implementing multiple interfaces



```
public class MultipleInterfaces implements InterFaceOne, InterFaceTwo {
```

Generics

Wednesday, November 5, 2014 9:45 PM

```
public class Node<T> {

    private T data;

    public Node(T data) { this.data = data; }

    public void setData(T data) {
        System.out.println("Node.setData");
        this.data = data;
    }
}
```

```
public static void addNumbers(List<? super Integer> list) {
    for (int i = 1; i <= 10; i++) {
        list.add(i);
    }
}
```

The upper bounded wildcard, `<? extends Foo>`, where `Foo` is any type, matches `Foo` and any subtype of `Foo`. The `process` method can access the list elements as type `Foo`:

```
public static void process(List<? extends Foo> list) {
    for (Foo elem : list) {
        // ...
    }
}
```

Arrays Class

Wednesday, November 5, 2014 7:46 PM

Useful Methods:

- int binarySearch(T[] array, Object key, int from, int to)
  - **Array must already have been sorted!**
  - Returns index of specified element or negative if not found
  - Range optional
- void sort(T[] array, int from, int to)
  - Sorts array
  - Range optional
- String toString(T[])
  - Returns String representing the array
- boolean equals(T[] a, T[] b)
  - Returns true if arrays have same type, number of elements, the same elements, and all elements are in the same order

String Class

Wednesday, November 5, 2014 9:27 PM

Useful methods

- char chatAt(int index)
- int compareTo(String str)
  - Returns 0 if strings are equal
- boolean contains(CharSequence)
  - Returns true if string contains CharSequence
- String concat(String str)
  - Return this String contactinated with str
- String trim()
  - Returns copy of this string with leading and trailing whitespace removed
- String[] split(String regex)
  - Returns string array of this string separated by each regex instance
- String substring(int begin, int end)
  - Return copy of this sting from begin (inclusive) to end (exclusive)
- int length()

Tips

- If you want to reverse a string:
  - StringBuilder sb = new StringBuilder(str);
  - sb.reverse();
  - sb.toString();

StringBuilder Class

Wednesday, November 5, 2014 9:18 PM

Constructor example

- StringBuilder sb = new StringBuilder()

Useful methods

- String toString()
- append(String str)
    - Add str to the end of the string
- insert(int offset, String str)
    - Inserts str starting at offset
- reverse()

Java 8

Thursday, November 06, 2014 10:00 AM

- Java 8 just came out in March!

Important new features

- Lambda expressions (basically anonymous functions
- New Streaming API
- New Date/Time API

Networking

Thursday, November 06, 2014 10:12 AM

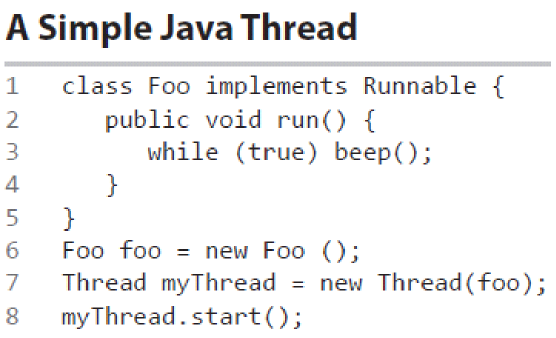
- URL object
- URLConnection connection = urlObject.openConnection()
- BufferReader in = new BufferReader(new InputStreamReader(connection.getInputStream()));

Concurrency

Thursday, November 06, 2014 10:12 AM

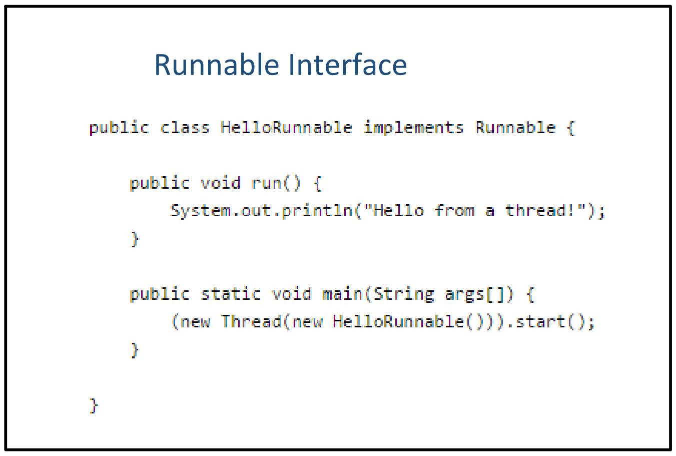
Overview

- Use Runnable interface
  - Implements runnable
  - Void run()
  - Public static void main(String args[])
- API
  - Thread.sleep(ms)
  - t.join()
    - Wait for thread t to terminate
- Synchronized methods
  - public synchronized void increment()
  - If one thread is executing a synchronized method, any other thread that tries to execute it will block
- Synchronized block
  - synchronized(object) { //this part is synchronized }
  - Synchronized statements must specify object providing lock
- Immutable objects' states can't change after they are created



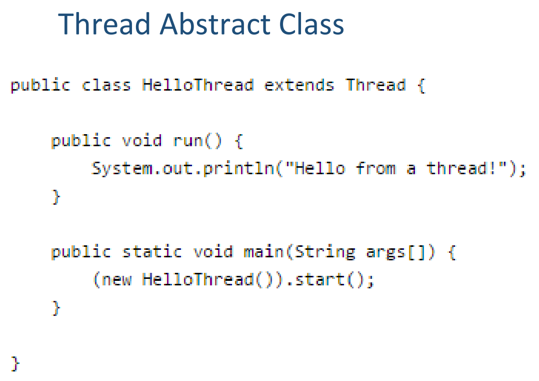
A Simple Java Thread

```
class Foo implements Runnable {
   public void run() {
      while (true) beep();
   }
}
Foo foo = new Foo ();
Thread myThread = new Thread(foo);
myThread.start();
```



Runnable Interface

```
public class HelloRunnable implements Runnable {

    public void run() {
        System.out.println("Hello from a thread!");
    }

    public static void main(String args[]) {
        (new Thread(new HelloRunnable())).start();
    }

}
```

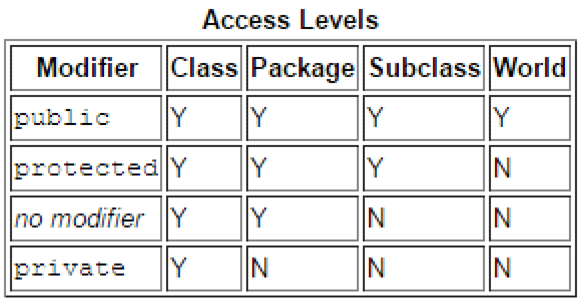


Thread Abstract Class

```
public class HelloThread extends Thread {

    public void run() {
        System.out.println("Hello from a thread!");
    }

    public static void main(String args[]) {
        (new HelloThread()).start();
    }

}
```

Access Levels

Sunday, November 9, 2014 12:36 PM



Access Levels

Modifier	Class	Package	Subclass	World
`public`	Y	Y	Y	Y
`protected`	Y	Y	Y	N
no modifier	Y	Y	N	N
`private`	Y	N	N	N

Comparable vs. Comparator

Wednesday, November 12, 2014 2:31 PM

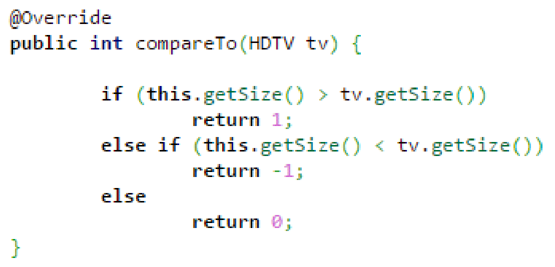
Comparable

- Interface
- Int compareTo() method

Comparator

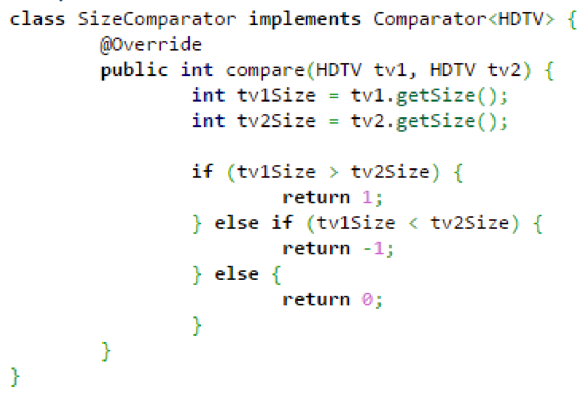
- Interface
- Use when you don't want the class itself to do the comparison, or you just one to compare certain attributes of the objects
- Use external class to compare

Comparable



```java
@Override
public int compareTo(HDTV tv) {

        if (this.getSize() > tv.getSize())
                return 1;
        else if (this.getSize() < tv.getSize())
                return -1;
        else
                return 0;
}
```

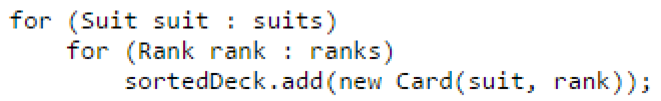
Comparator



```java
class SizeComparator implements Comparator<HDTV> {
        @Override
        public int compare(HDTV tv1, HDTV tv2) {
                int tv1Size = tv1.getSize();
                int tv2Size = tv2.getSize();

                if (tv1Size > tv2Size) {
                        return 1;
                } else if (tv1Size < tv2Size) {
                        return -1;
                } else {
                        return 0;
                }
        }
}
```

For each

Wednesday, November 12, 2014 3:34 PM



```
for (Suit suit : suits)
    for (Rank rank : ranks)
        sortedDeck.add(new Card(suit, rank));
```

Proof by induction

Wednesday, November 5, 2014 7:22 PM

- Base case
    - Proof that algorithm holds for base case (probably 0 or 1)
- Inductive step
    - Proof that if algorithm holds for n, it also holds for n+1
- Therefore, algorithm is true for all n

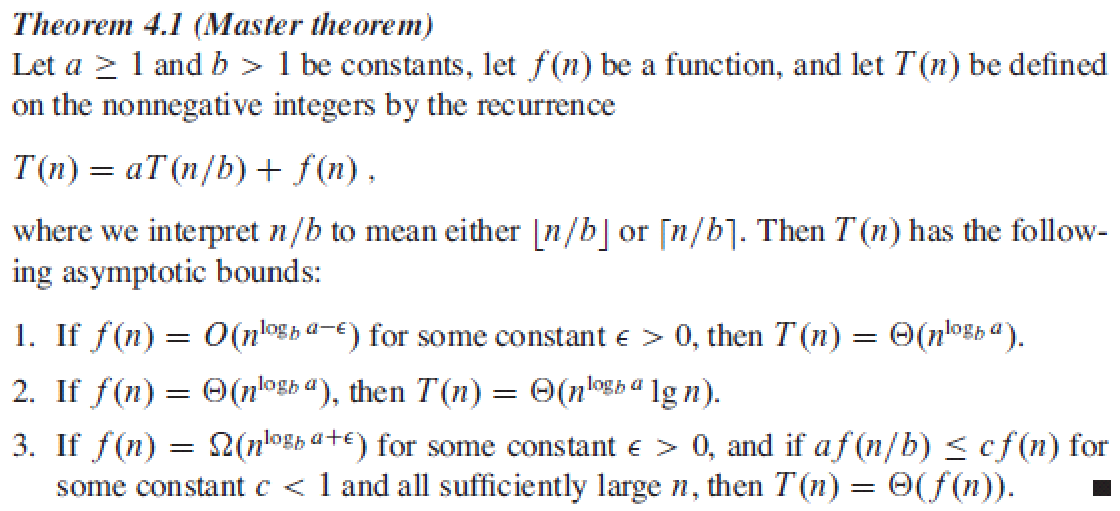
- # Dominance Relations

Wednesday, November 5, 2014 9:49 PM

- Constance functions - f(n)=1
  - Examples:
    - Cost of adding 2 numbers
    - Printing a string
  - There is no dependence on the parameter n
- Logarithmic functions - f(n)=log(n)
  - Example:
    - Binary search
  - Functions grow slowly with n
- Linear functions - f(n) = n
  - Examples:
    - The cost of looking at each item once in an n-element array, maybe to ID biggest, smallest, or average value
- Super linear functions - f(n)=n*log(n)
  - Examples:
    - Quicksort
    - Merge sort
  - Grow slightly faster than linear functions
- Quadratic functions - f(n)=n^2
  - Cost of looking at most or all pairs of items in an n-element universe
  - Examples:
    - Insertion sort
    - Selection sort
- Exponential functions - f(n)=c^n
- Factorial functions - f(n) = n!
  - Example:
    - Generating all permutations or orderings of n items
- $1 < \log n < n < n \log n < n^2 < 2^n < n!$

Master theorem

Saturday, October 12, 2013 3:31 PM

- Recursion:
  - Divide problem into sub problems that are smaller instances of the same problem
  - Conquer the sub problems by solving them recursively
  - Combine the solutions to the sub problems into the solution for the original problem
- Eventually, sub problems bottom out to base case
- Recurrence - an equation or inequality that describes a function in terms of its value on smaller inputs
- Example: recurrence of merge sort (worst case)
  - $T(n) = \theta(1)\ if\ n = 1$
  - $T(n) = 2T\left(\frac{n}{2}\right) + \theta(n)\ if\ n > 1$
- If you divide problem into unequal sizes (like 2/3 and 1/3) recurrence would look like this:
  - $T(n) = T\left(\frac{2n}{3}\right) + T\left(\frac{n}{3}\right) + \theta(n)$
- Remember that $\log_4 3$ is the power 4 needs to be raised to to get 3
- Memorize the following:
- 

  Theorem 4.1 (Master theorem)
  Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence

  $$T(n) = aT(n/b) + f(n) ,$$

  where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:

  1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
  2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
  3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

  - n=size of problem
  - a=number of sub problems in recursion
  - n/b is the size of each sub problem
- Mater theorem problem steps
  - Set up recurrence relation
  - Decide it master theorem applies
  - Decide which case to use
- Pro tip:
  - $constant < \log(n) < n < n * \log(n) < n^k < k^n < n!$

P and NP

Thursday, November 06, 2014 10:13 AM

P

- Decision problems that can be determined in polynomial time
- O(n^x) or less
- These are problems we can actually calculate exact solutions for

NP

- Decision problems that can be verified by not determined in polynomial time
- O(x^n), O(n!)

NP Hard

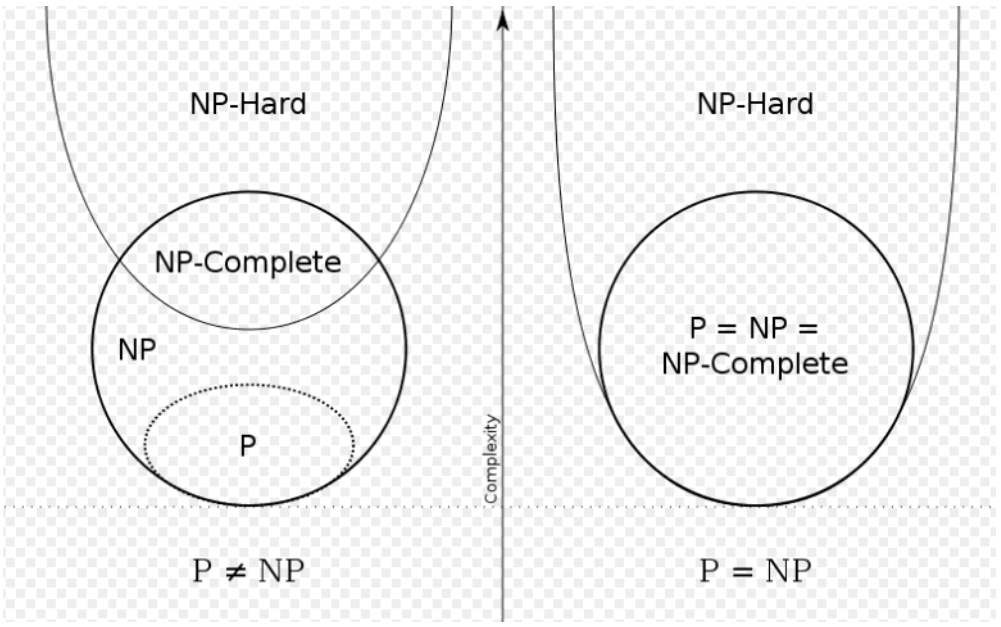
- Problem x is NP hard if every NP problem can be reduced to x in polynomial time
- "At least as challenging as the most challenging NP problem"

NP Complete

- $NP - complete = NP \cap NP - hard$
- Problems that are both NP and NP hard

P=NP?

- We don't know, but I suspect P != NP
- If we found a polynomial solution to a NP-hard problem, all NP problems could be reduced to it
- This would break most encryption



Relational vs. Non-relational

Wednesday, November 5, 2014 9:21 PM

- Relational DB - SQL
    - Overview
        - Tables, foreign keys link tables
        - One-to-one, many-to-one, and many-to-many relationships
        - Database schema is determined in advance
        - Process data when it is added to the DB
        - Great for datasets that aren't likely to expand quickly
    - Advantages
        - Data is always consistent (so like, financial information)
        - Familiarity
        - Good for real-time analysis, since data is already structured correctly
    - Disadvantages
        - Schema has to be determined in advance
        - Data has to conform to schema, so you could lose some data
        - Hard to scale
- Non-Relational DB
    - Overview
        - No explicit or pre-determined structure
        - Process data during query
        - Designed to excel in speed and volume
        - Could be:
            - Graphs
            - Documents
            - Key-value
            - Distributed
            - Hierarchical
    - Advantages
        - Don't have to determine schema in advance
        - Data can be used in different ways in different applications
        - Performance
        - Scalability
    - Disadvantages
        - Data isn't necessarily consistent in the DB all the time

DynoamoDB

Monday, November 10, 2014 10:01 AM

- Amazon Web Service NoSQL DB service
- Supports document and key-value data models
- Fully managed and scalible

SimpleDB

Friday, November 07, 2014 10:13 AM

- Basically a simpler and less scalable version of DynamoDB
- Automatically indexes all item attributes for query flexibility at the cost of performance and scale

Unix

Friday, November 07, 2014 11:21 AM

Kernel & User Space

- Kernel mode - all instructions allowed
- User model - some instructions forbidden
- System calls are interface between user space programs and kernel
    - Switches CPU from user to kernel mode
    - Examples: IO, files

Processes & Threads

Wednesday, November 5, 2014 9:21 PM

Kernel vs. User level threads

- User space threads
  - Kernel not away that program has threads
  - IO will block entire process
  - Allows you to use threading on OS that doesn't support threading
  - Threads & thread table managed in user space
- Kernel space threads
  - Thread managed by kernel
  - IO doesn't block entire process

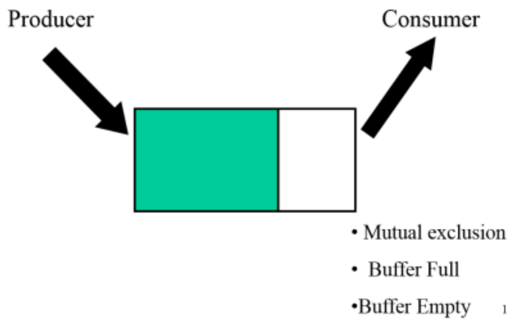
Process vs. Thread

	Process	Thread
Creation	Expensive	Cheap
Context Switch	Expensive	Cheap
Stack registers program counter State	Own	Own
Address space Global variables Open files	Own	Shared

Synchronization

Friday, November 07, 2014 10:15 AM

- Mutex - lock on a resource
- Semaphore - record of how many units of a particular resource are available, and forced threads/processes to wait if none available
    - Producer-consumer problem
- Deadlock - each of two tasks is waiting for a lock that the other task holds



Memory Management

Friday, November 07, 2014 10:15 AM

Stack & Heap

- Stack
  - Automatically allocated and deallocated
  - Local variables and parameters
  - Used to implement function calls
- Heap
  - Manually allocated and deallocated
  - Dynamic sized data structures
  - Used for data whose size is hard to predict
  - Heap is garbage-collected in Java
  - Malloc() and free()

Address Space

- Protects processes from one another
- Protects OS from user processes
- Each process has its own address space
  - Address spaces don't overlap

Virtual Memory

Paging

Paging

Friday, November 07, 2014 10:15 AM

Paging

- Divide address space into fixed-sized pages
- Internal fragmentation within page
- Page fault - if requesting page isn't in memory

Segmentation

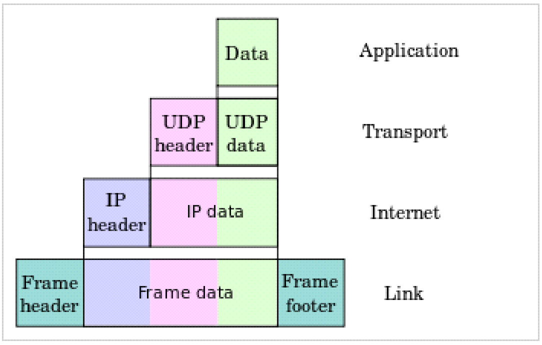
- Divide address space into variable-sized segments
- External fragmentation on disk

Internet Stack

Friday, November 07, 2014 10:16 AM

Layers

- Application
  - Treats lower levels as black boxes
  - Examples:
    - DNS, FTP, HTTP, SSL, POP
- Transport
  - TCP - Transmission Control Protocol
    - Reliable, order and error-checked
    - Application protocol examples:
      - HTTP, FTP, POP
    - Maintains connection
  - UDP - User Datagram Protocol
    - Connectionless
    - Low overhead, not ordered, no acknowledgement that packets have been received
    - Good for streaming media, where speed is more important than perfect accuracy
    - DNS uses UDP
- Internet
  - Responsible for sending packets across network boundaries
  - Delivers packets from source host to destination based on IP address in packet headers
  - Stateless
  - Routing
  - End-to-end principal - network infrastructure is considered inherently unreliable
    - Only provides best-effort delivery
  - IP
    - IPv4 had 32-bit addresses, i.e.: 172.16.254.1
    - IPv6 has 64-bit addresses, i.e.: 2001:0DB8:AC10:FE01:: (2nd half omitted)
- Link
  - Methods and standards that operate only between adjacent network nodes
  - IEEE 802.x
  - Ethernet, Wifi

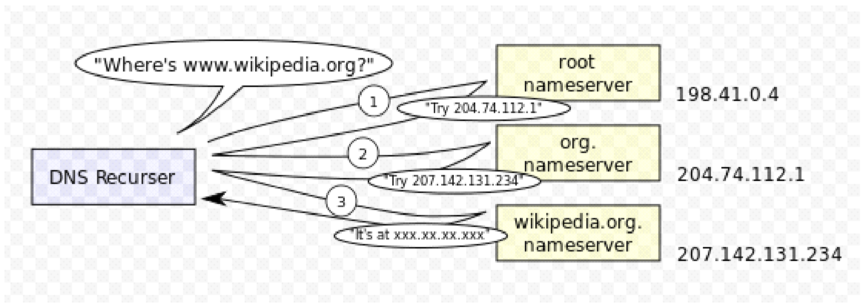


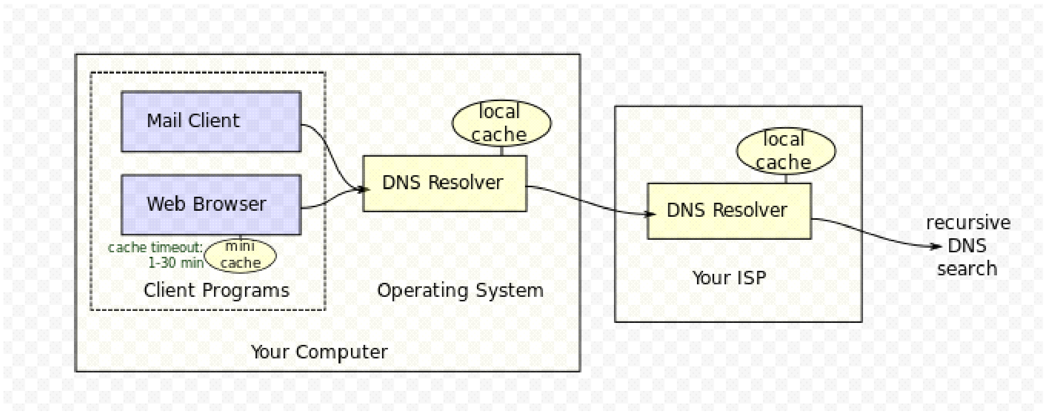
Encapsulation of application data descending through the layers described in RFC 1122

DNS

Friday, November 07, 2014 10:16 AM

- Maps domain names to IP addresses
- DNS resolver is client side of DNS
  - Responsible for initiating and sequencing the queries that ultimately lead to full resolution of IP address
  - Probably has a cache containing recent lookups





Sockets & Ports

Friday, November 07, 2014 10:16 AM

Sockets

- Network socket is an endpoint of inter-process communications
- Socket address is the combination of an IP address and port number
  - Port tells socket which process or thread to deliver packet to
- Socket types
  - Datagram socket - connectionless, uses UDP
  - Stream socket - connection-oriented, uses TCP
  - Raw socket - bypasses transport layer, packet headers are made available to application

Ports

- Ports uniquely identify different applications or processes running on the same computer
- Port numbers are 16 bits
- Some common ports
  - 20 - FTP
  - 22 - SSH
  - 53 - DNS
  - 80 - HTTP
- For TCP, only one application and be on a port at a time
- For UDP, multiple applications can subscribe to same port

TLS/SSL

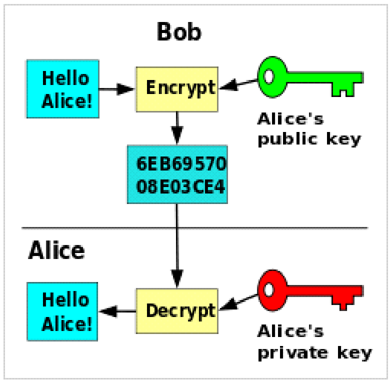
Friday, November 07, 2014 10:26 AM

Overview

- TLS - Transport Layer Security
- SSL - Secure Sockets Layer
- Cryptographic protocols designed to provide communication security over the internet
- Uses public key encryption
- SSL certificates are important to prevent man-in-the-middle attacks

Initial SSL Handshake

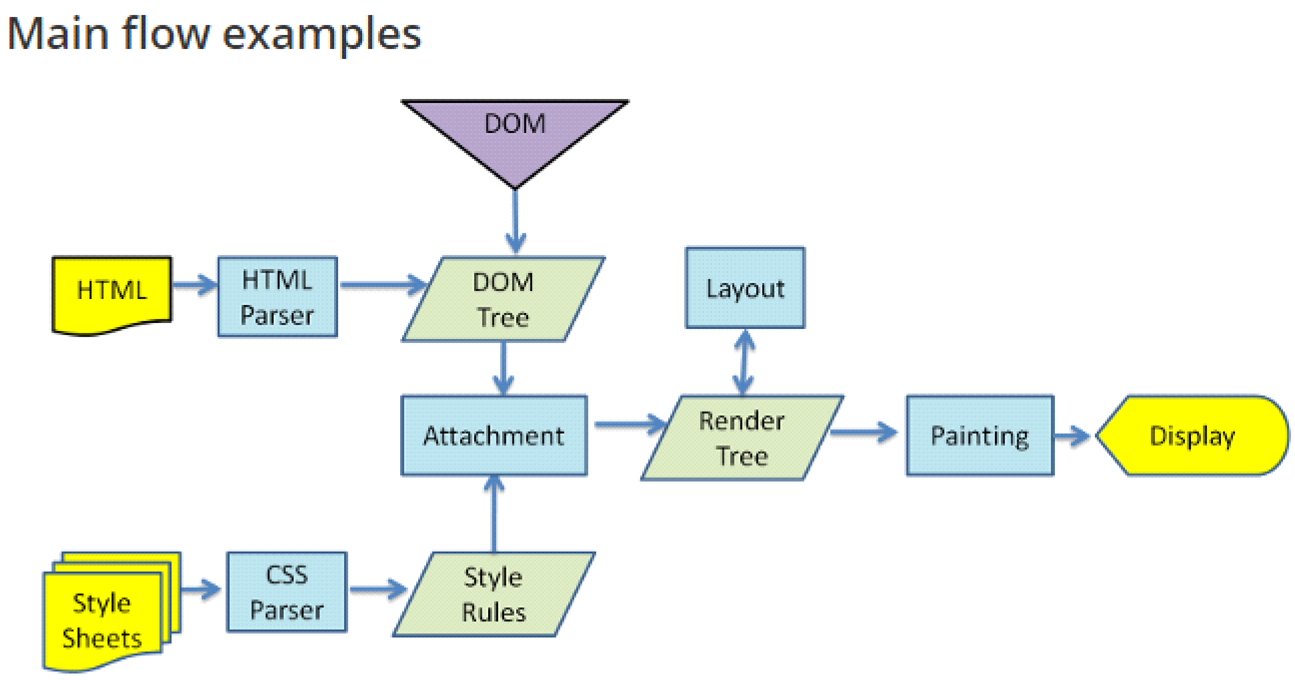
- Client makes initial contact with sever
- Server sends client its SSL certificate
- Client authenticates certificate
- Client creates pre-master secret, encrypts it with server's public key, and sends secret to server
- Server decrypts pre-master secret with private key, and generates master secret
- Both client and server us master secret to generate symmetric session keys to encrypt and decrypt further communications



In an asymmetric key encryption scheme, anyone can encrypt messages using the public key, but only the holder of the paired private key can decrypt. Security depends on the secrecy of the private key.

Web Browser Basics

Friday, November 07, 2014 11:13 AM



Vending Machine Example

Wednesday, November 12, 2014 2:59 PM

- Person
    - Fields
        - Coins
        - Items
    - Methods
    - Subclasses
        - Customer
        - ServiceTech
- Coin
    - Fields
        - Value
    - Methods
        - .getValue()
    - Subclasses
        - Penny
        - Nickle
        - Dime
        - Quarter
- Vending Machine
    - Fields
    - Methods
        - .getInventory()
        - .getCurrentBalance()
        - .insertCoins()
        - .despenseItem()
        - .balanceBack()
        - .addInventory()
        - .openAndAuth(ServiceTech serviceTech)
        - .close()
- Product
    - Fields
        - Value
        - Description
    - Methods
        - .getValue()
        - .getDescription()

```
public class Coin
{
        public enum CoinType
        {
                PENNY,
                NICKLE,
                DIME,
                QUARTER
        }
        private int value;
        private CoinType type;
        public Coin(CoinType coin)
        {
                this.type=type;
                If(type==PENNY) value = 1;
                else if(type==NICKLE) value = 5;
                else if(type==DIME) value = 10;
                else value = 25;
        }
        public int getValue() { return value; }
        public CoinType getType() { return CoinType; }
}
```

```
public class VendingMachine
{
        private Hashtable<String,Stack<Product>> inventory;
        private int currentBalance;
        private Hashtable<Coin.CoinType,Stack<Coins>> coins;
        private ArrayList<Coin> balance;
        private int intBalance;
        public VendingMachine()
        {
                inventory = new Hashtable<Coin.CoinType,Stack<Product>>();
                currentBalance = 0;
                coins = new Hashtable<String,Stack<Coin>>();
                balance = new ArrayList<Coin>();
        }
        public boolean insertCoins(ArrayList<Coin> coins)
        {
                for(Coin c : coins)
                {
                        currentBalance += c.getValue();
                }
                If(balance.size() == 0) balance = coins;
                else balance.addAll(coin);
        }
        public Product despenseItem(String description)
        {
                Product toReturn;
                If(inventory.getKey(description).size() > 1
                        && currentBalance >= inventory.getKey(description).elementAt(0))
                {
                        toReturn = inventory.getKey(description).elementAt(0);
                        currentBalance -= toReturn.getValue();

                }
        }
}
```

```
public class Product
{
        private int value;
        private String description;

        public Product(String description, int value)
        {
                this.value = value;
                this.description = description;
        }
        public int getValue() { return value; }
        public String getDescription() { return description; }
}
```

```
public class Person
{
        public enum PersonType
        {
                CUSTOMER,
                SERVICETECH
        }
        private Hashtable<Coin.CoinType,ArrayList<Coin>> coins;
        private Hashtable<String,ArrayList<Product>> products;
        private String name;
        private PersonType type;
        public Person(String name PersonType type)
        {
                this.name = name;
                This.type = type;
                coins = new HashTable<Coin.CoinType,ArrayList<Coin>>();
                products = new Hashtable<String,ArrayList<Product>>();
        }
}
```

Depth-First Search

Sunday, August 16, 2015 9:15 PM

-

Breadth-First Search

Sunday, August 16, 2015 9:18 PM

Dijkstra's Algorithm

Sunday, August 16, 2015 9:18 PM