**Exam 1 Review Topics**

Exam 1 covers all material to date in this course. This includes chapters 1 through 8 (excluding

6) and 10.1 from your textbook. Major topics to review include:

Object Oriented Programming Ideas and Motivation

Why use OO(Object Oriented) Programming?

* + - Objects can also be reused within an across applications. ...
* It makes software easier to maintain. ...
* Reuse also enables faster development.

**Inheritance** is a powerful feature in **object oriented programming**. It refers to defining a new class with little or no modification to an existing class. The new class is called derived (or child) class and the one from which it **inherits** is called the base (or parent) class.

**Aggregation** is a way of composing different abstractions together in defining a class. For **example**, a car class can be defined to contain other classes such as engine class, seat class, wheels class etc. ...

Encapsulation. In an object oriented python program, you can restrict access to methods and variables. This can prevent the data from being modified by accident and is known as Encapsulation.

We call the part of a program where a variable is accessible its **scope**,

The duration for which the variable exists its **lifetime**.

A **variable** which is defined in the main body of a file is called a **global variable**.

It will be **visible** throughout the file, and also inside any file which imports that file.

Software Engineering

What are Stages: Requirements through Maintenance

* Requirement gathering and analysis. (what’s the problem)
* Design.
* Implementation or coding.
* Testing.
* Deployment.
* Maintenance.

ADTs and Abstraction

ADT is an Abstract Data Type –

Abstract means you need to interact with it but don’t need to define all the details for the user. It is called “**abstract**” because it gives an implementation-independent view. The process of providing only the essentials and hiding the details is known as **abstraction**.

**Motivation** (why ADTs) *Interchangeability of Parts*: Different implementations of an **abstract data type** may have different performance characteristics. With **abstract data types**, it becomes easier for each part of a program to use an implementation of its **data types** that will be more efficient for that particular part of the program. For example Animal type can have Cats and birds and store different characteristics for each type.

Walls and mirrors is a book.

walls" of the title refer to the abstract data type (ADT) which has a wall between its public interface and private implementation.

The "mirrors" of the title refer to recursion. The idea is of looking at a reflection in two mirrors placed in opposition to one another, so a repeated image is reflected smaller and smaller in them.

**Interfaces** Java **Interface** is a way to specify an **Abstract Data Type** (ADT). You can declare a class as **abstract** when it contains zero or more **abstract** methods or when an **interface** is implemented to a class where not all methods are not implemented.

**Relationship with data structures**

Static and Dynamic/Reference Arrays

What is the difference? **Static array** contents are initialized to the default fixed size of the **array** element type. **Dynamic arrays** are initialized to having 0 elements. Associative **arrays** are initialized to having 0 elements

How do we choose? **Static Data** structure has fixed memory size. whereas in **Dynamic Data** Structure, the size can be randomly updated during run time which may be considered efficient with respect to memory complexity of the code. ... Unlike **static data** structures, **dynamic data** structures are flexible.

**Dynamic arrays benefit** from many of the **advantages** of **arrays**, including good locality of reference and data cache utilization, compactness (low memory use), and random access. They usually have only a small fixed additional overhead for storing information about the size and capacity

How do we use? For dynamic arrays Key Features of **Dynamic Array** Add Element: Add element at the end if the **array** size is not enough then extend the size of the **array** and add an element at the end of the original **array** as well as given index. Doing all that copying takes O(n) time, where n is the number of elements in our **array**.

Java Collections Framework

Advantages of use

Reduces programming effort: By providing useful data structures and algorithms, the **Collections Framework** frees you to concentrate on the important parts of your program rather than on the low-level "plumbing" required to make it work **Java's collection** classes provides a higher level interface than arrays. Arrays have a fixed size

Measuring Algorithmic Efficiency

Big O versus Benchmarking Big O isa manner in which to measure efficiency. Fortunately, Big-O analysis gives us just that. The concept of Big O is based on the time it takes for an algorithm to run based on the size of the dataset.

BenchMarking is based on actual

we can use the Benchmark module provided to us by the Ruby Standard Library actual running

Comparing and selecting data structures

Integer, Float, Boolean, Char etc, all are **data structures**. They are known as Primitive **Data Structures**. ... Some example of Abstract **Data Structure** are : Linked List. Tree. arraylist

Specific Data Structures/ADTs

Linked Lists, Doubly Linked Lists, Circular Linked Lists, Stacks (Postfix/Infix), Queues,

Priority Queues, Circular Queues, Trees, BSTs

Underlying data structures

How to choose?

Binary trees

How are they organized?

**Example** of a Perfect **binary tree** is ancestors in the family. Keep a person at root, parents as children, parents of parents as their children. A **binary tree** is balanced if the height of the **tree** is O(Log n) where n is the number of nodes.

Terms: leaf, root, child, height, etc.

The depth of a node is the number of edges from the root to the node.

The height of a node is the number of edges from the node to the deepest leaf.

The height of a tree is a height of the root.

A full binary tree.is a binary tree in which each node has exactly zero or two children

A child of a tree is a lower branch

In-order, pre-order, post-order traversals **Study what results would be in each way below**

* In **preorder** each parent node is visited before (**pre**) its children.
* In **inorder** each parent node is visited **in** between its children.
* In **postorder** each parent node is visited after (**post**) its children.

What is the difference between a binary tree and a BST?

[**Binary Tree**](http://en.wikipedia.org/wiki/Binary_tree) is a specialized form of tree with two child (left child and right Child). It is simply representation of data in Tree structure

[**Binary Search Tree (BST)**](http://en.wikipedia.org/wiki/Binary_search_tree) is a special type of Binary Tree that follows following condition:

1. left child node is smaller than its parent Node
2. right child node is greater than its parent Node

What types of applications may use binary trees or BSTs?

**Applications of Binary tree:**

* Implementing routing table in router.
* Data compression code.
* Implementation of Expression parsers and expression solvers.
* To solve database problem such as indexing.
* Expression evaluation.

**Binary search trees support** everything you can get from a sorted array: efficient search, in-order forward/backwards traversal from any given element, predecessor /successor element search, and max /min queries, with the added benefit of efficient inserts and deletes. With a self-balancing binary search tree (BST), all of the above run in logarithmic time.

What shape of BST (binary search Tree) is desirable? In general, it is preferable for a BST to be as shallow as possible. This keeps the average cost of a BST operation low.

Huffman Encoding

**Huffman Coding** (also known as **Huffman Encoding**) is a algorithm for doing data compression and it forms the basic idea behind file compression. This post talks about fixed length and variable length **encoding**, uniquely decodable codes, prefix rules and construction of **Huffman** Tree.

Javadoc/Programming by Contract **Programming "by contract**" is essentially a means of allowing programmers to verify execution of their methods does not corrupt the state of their data structures and so on. It is about what your code does not how it does it, some like to say