Lecture Inheritance 9/9/19

Monday, September 9, 2019 2:03 PM

Early and Late bindings,

Faculty carol = new Faculty();
Person p = carol;
Sysout(p.toString());

Which version of toString would execute?

Early (Static) binding: The variable p is declared to be of type Person. Therefore we should call the Person toString ()

Late (Dynamic) binding: The object to which p refers was created as a "new faculty"
Therefore we should call the faculty toString()

Java uses Late (Dynamic) binding
Benefits include having custom methods for different types of objects, polymorphism

Polymorphism, - one line of code can do many different things

getClass and instanceof

getClass() - returns a representation of the class of any object

instanceof() - is a relationship between two objects

Ex/

Person bob = new Person()
Person teds = new Student()
Person carol = new Faculty()

Faculty drSmith = new Faculty()

Ex/
bob.getClass() == teds.getClass() -> False because different classes

Bob instanceof Person -> is bob a instance of a person at runtime = True

Teds instanceof Student -> is Ted a instance of a student = True

Ted instanceof Person -> is ted a instance of Person = True

Bob instanceof Student -> False bob not a student

Carol instanceof Person -> True carol is a person

Carol instanceof student -> False carol not student

Upcasting is always safe, casting to a higher class in the inheritance tree
Ex/ casting from student to person
Downcasting can cause issues, you have to cast, Student s = (Student) person;

Lecture Inheritance/Abstract 9/11

Wednesday, September 11, 2019 2:01 PM

WITHOUT ABSTRACT CLASS
Shape[] arr = new Shape[5];
arr[0] = new rectangle
arr[1] = new circle

Can store subclasses in parent class array
All elements in shape[] are different types of shapes

WITH ABSTRACT CLASS

```
Public abstract class Shape{
        Public void color();
        Public abstract void drawMe();
}
```

drawMe method does not need to be implemented by subclass inheriting shape
Color method needs to be implemented because it is not an abstract method
Abstract classes usually have a **"has a"** relationship
A non-abstract method of an abstract class can call a non-abstract method inside the class

Abstract class vs interface
-abstract can create instance member variables
-can implement multiple interfaces, only can extend one class
-inheritance is an **"is a"** relationship
-composition = class with field as a reference to another class
    -EX/ Pet s = new Pet();
    -usually used for classes you create

EX/ parking permit
A parking permit **"is a "** person? No
A parking permit **"has a "** person? yes
So go with composition and abstraction

MULTIPLE INHERITANCE
-Java does not allow multiple inheritance, C++ does but it sucks
-Java can only extend one class, but can implement unlimited classes
    -You can "fake" multiple inheritance this way
    -EX/
        class StudentAthlete extends Student implements Athlete
    -call extends before implements

-final declaration disables overriding for that method or variable

Lecture Exceptions 9/13

Friday, September 13, 2019 2:09 PM

Exception - rare event outside of normal behavior
- usually happens at run-time

EX/ Division by zero, out of memory, access past end of array

To handle exceptions use **Exception Handling**

Exceptions are derived from **Throwable**

Two types of exceptions - **Checked** and **Unchecked**

Any code that can potentially throw an error you should put in a try catch clause

```
try{

}catch(Exception e){

}
```

Can add finally block after try catch
Finally will fun every single time if there is an error or not

If you do not deal with the exception using a try catch then
Can declare a exception in the method

```
Public cat() throws exception{}
```

Checked exception
Try catch block

Uncheck exception
Throws exception in method header

If no throws and no try catch then it is a unchecked exception

Nested Types 9/16

Monday, September 16, 2019 2:06 PM

Top level class can only be public or package-protected

Top level classes
- -declared inside package

Nested types are
- -declared inside class (or method)
- -normally used only in outer (enclosing) class
  - -can have wider visibility

4 different types of nested types

- -**Inner Class**
  - -only applies to classes
  - -will not be static

- -**Local Classes**
  - -class defined in a block of java code

- -**Anonymous class**
  - -Local class without name

- -**Static class**
  - -interface, enum, annotation

Why use nested types?
- -trying to break encapsulation, treated like a member
- -only needed in small parts of the code, localize it to help with abstraction
  - -similar to a helper method concept

Inner classes

Class defined in scope of another class
No static members

Outer and inner classes can access private field

EX/

```
Public class A{
    Private int x;

    Void test(){
        A obj = new A();
        A.x = 2; //accessing private field
    }
}
```

Anonymous class syntax

-does not need a name for the class

Ex/

```
Public Iterator test(){
        Return new Iterator(){
                Public has next ……

        }
}
```

Team Class 9/18

Wednesday, September 18, 2019 1:59 PM

Anonymous class

```
Public Iterator test(){
  return new Iterator(){
        Public boolean hasNext(){
        }
  };
}
```

Static class

Nested class can access elements outside of nested class but inside of outerclass

```
Public class outerClass(){
        Int y = 5;
        Public static class nestedExample(){
                Int x;
                Void test(){
                        Y=50;
                }
        }
}
```

Can access it from outerclass

```
        outerClass.nestedExample test = new outerClass.nestedExample();
        Test.test();
```

Cloning

- -cloneable interface
- -cannot clone every type of object
- -is a checked exception, need try catch

```
@Override
Public Mouse clone(){
        Mouse obj = null;

        try{
                obj = (Mouse) super.clone();
        }catch(exception e){
                Sysout(e)
        }
}
```

Lambda Expressions 9/23

Monday, September 23, 2019 1:57 PM

Lambda expressions are concise approaches to define anonymous class instances
Only works in **Function interfaces**
-parameters are inferred by the compiler
-parentheses can be dropped

Function interfaces only have one abstract method

EX/

```
Interface Task{
	Public int compute(int x);
}

Interface atest{
	Public int compute();
}


Interface processor{
	Public float increase(int x, float y);
}


Main{
	Task t = new Task(){
		Public int computer(int x){
			Return x + x;
		}
	};
	Sysout(t.compute(1));

	//only works with function interfaces
	Task t = (int x) -> {
		Return x + x;
	};
	Sysout(t.compute(1));

	Task t = (x) -> x + x;
	Sysout(t.compute(1));

	atest t = () -> x + x;
	Sysout(t.compute());

	Processor t = (int x, float y) -> x * y;
	Sysout(t.increase(10, 5));


}
```

Clone method

-if you override you need to implement cloneable
-**Marker interface** = no abstract methods, creates "is a" relationship

Garbage, generic, initblocks 9/25

Wednesday, September 25, 2019 2:00 PM

Java does automatic garbage collection
-reclaims unused memory for future use
-some languages do not do this automatically

Destructor in other languages
-method with name finalize()
-last method to run

Initialization Block type
-code execute when each object is created
-**runs before constructor**

Static initialization block
-code executed when class loaded

Static {A = 1;} //static initialization block
{A = 2;} //initialization block

Generic Programming
-defining constructs that can be used with different data types

EX/ ArrayList<E>

Valid types
- Class
- Interface

Invalid
- Primitive types
- Wrappers

Creating generic class
- Ex/

```
Public class test<T>{
    Private T value;
    Public test(T v){value = v;}
    Public T getVal(){return value}
}
```

Generic arrays are tricky
Ex/
Valid
- T[] data = (T[]) new Object[4];

Invalid
- T[] data = new T[5];

Generics 9/30

Monday, September 30, 2019 2:05 PM

Accept multiple/all methods for class or methods
-can have generic methods in non-generic class
-**Bounded** generics limit what data types can be passed/returned

Unbounded example

```
Public class generic<T>{
        Private T value;
        Public gen(T t){

        }
}
```

Bounded example

```
Public static <T extends Comparable<T>> boolean test(T value){

}
```

Generics and subtyping

```
ArrayList<String> strL = new ArrayList<String>();
ArrayList<Object> objL  = strL; //illegal this does not work
```

Because if you put an obj element that is in the String class
And then you try to call the obj method from the string element
It will error out
-the compiler stops this from being allowed

Ex/

```
ArrayList<String> strL = new ArrayList<String>();
ArrayList<Object> objL  = strL; //illegal this does not work
objL.add(1);
String s = objL.get(1); //error but you will not get this far using ArrayList
```

Also will not work:

```
ArrayList<parentclass> obj  = new ArrayList<subclass>();
```

Arrays will allow this to happen

```
Parent[] array = new subclass[5];

Subclass[0] = new parent(); //will not work, compile error because the compiler knows
```

Ex/

```
Fruit[] arr = tropicalfruitarray;
Fuit[0] = new fruit(); //will not work, gen exception at runtime
//compiler does not know any better
```

Wildcard
Collection whos element type matches anything
Ex/ ArrayList<?>

Can also be bounded
Ex/
ArrayList< ? extends shape>
ArrayList< ? super shape>

Extends- will take anything under shape, <=
Super- will take anything above shape, >=

Will create a warning if you add a class
Reading is okay, writing is tricky for extends
Ex/
<? extends B> arr
Arr.add(new C()); //throws warning

Note: null can always be added except prim types ex/ arr.add(null)

Super is tricky to read, but not writing
Ex/
<? Super B>arr
A test = arr.get(0); //throws warning

Ex/
<? Extends compiter> cl
Cl.add(new computer);
//cannot do because the arraylist could be a subclass of computer
//so the compiler does not let you do it

Algorithms 10/2

Wednesday, October 2, 2019 2:02 PM

Efficiency

-amount of time and space used

Measuring efficiency

-benchmarking
    -measure time and space needed
-asymptotic analysis

Benchmarking

-advantages
    -precise information for given configuration
-disadvantages
    -affected by configuration
    -data sets (often to small)
    -hardware/software
    -biased inputs
    -does not measure **intrinsic** efficiency

Asymptotic analysis

-calculate time as function
-remove constant factors
-remove low order terms

-big o notation

    -dominates efficiency for larger sizes
    -language, compiler, hardware irrelevant

-big o counts number of operations performed

Linked List 10/2

Wednesday, October 2, 2019 2:33 PM

Array vs Linked List
- -Array
  - -Advantages
    - -Can access position of any element O(1)
    - -efficient use of space
  - -Disadvantages
    - -expensive to grow/shrink array
    - -expensive to insert/remove O(n)
    - -tricky to insert/remove elements
- -LinkedList
  - -contains nodes that contain data, and a reference to the next node
  - -if the node is at the end of the list then it points to a null reference
  - -can be used with generics to hold multiple data types in one list
    - -singly vs doubly linked list,
    - -singly points to next, doubly points to next and prev node

Linked list 10/7

Monday, October 7, 2019 2:02 PM

Doubly linked list

Each node contains reference to prev and next

Ex/

```
Class Node{
    Object data;
    Node next;
    Node prev;
}
```

Circularly linked list

Last node linked to first node

'round robin'

Restricted Abstractions

Restricting the operators to make it simpler to use

Types

- -stack
- -queue
- -dequeue

Stack

-Elements removed in opposite order of insertion

-Last in, first out (LIFO)

-Stack operations

- -push = add element (to top)
- -pop = remove element (from top)

To create from linked list

Only allow access to the end of the list

Queue Recursion 10/9

Wednesday, October 9, 2019 2:00 PM

Queue
- -elements removed in order of insertion
- -first-in, first-out (FIFO)
- -only access to elements at the beginning/end of the list
  - -only add to end
  - -only remove from front
- -queue operations
  - -enqueue = add element (to back)
  - -dequeue = remove element (from front)
- -implementations
  - -linked list implementation
    - -add to tail (back) of list
    - -remove from head (front) of list
  - -circular array implementation

Dequeue
- -double ended queue
- -can remove from end and insert from front as well
- -can also add to end and remove from front

Recursion

Memory organization
- -3 main areas
  - -call stack = makes possible method execution
    - -makes recursion possible
  - -heap = where objects are created
  - -static area

Recursion
- -a procedure that calls itself
- -approach
  - -if
    - -problem is simple solve it directly
  - -else
    - -simplify problem into smaller instances
    - -solve smaller instances using an algorithm

Ex/ Factorial example

```
Int factorial(int n){
        If(n == 0){
                Return 1;
        }
        Return n * factorial(n-1);
}


//factorial(3)
//3 * factorial(2)
// 3 * 2 * factorial(1)
//3 * 2 * 1 * factorial(0)
//3 * 2 * 1 * 1
//returns 6
```

Recursion vs Iteration

- -iterative algorithm
  - -may be more efficient

Hashing 10/14

Monday, October 14, 2019 2:01 PM

Map in Java = Dictionary in Python
"Lookup Table"
Key -> Value pairs

Hashing
- -technique for storing key-value entries
- -ideally can result in O(1)
- -array is called a hash table

Hash function
- -takes key, generates hash index of value

Typical hash function
1. transforms key into hash code
2. Compresses hash code so it lies in the table
   i. Uses modulus to compress hash code
   ii. Fits hash code to fit in bound of table

Collision = two search keys map to same entry in hash table

Good hash function
- -fast to compute
- -minimizes collisions
  - -using a function to distribute values uniformly
  - -reduces probability of collisions

Hash Codes
- -**String**
  - -by adding Unicode values
  - -better approach
    - -multiple unicode value of each character by
      A factor that depends on the characters position
      In the string

      EX/ (ASCII)
      "Java" -> hash
      J = 74, a = 97, v = 118
      74 * (31)^3 + 97 * (31)^2 + 118 * 31 + 97 = 2,301,506

- **-primitive types**
  - -if key is int, use the int
  - -if char, short, byte, just cast to an int
  - -if long, float, double, manipulate the internal binary representation

there is a hash code contract we will learn later

Scaling/compressing hash codes

-use the modulus operator to compress an integer
Remainder = hash code % n
Remainder lies in the range [0, n-1]

Resolving collisions

1. First approach
   -Look for a unused entry in the table, "open addressing"

2. Second approach
   -Each element in the table can be associated with more than one search key
   -each element becomes a list
   -called "separate chaining" (or bucket)

Lecture 10 - 11

Recursion with linked list

Open addressing (not preferred)

1. Probing = find unused position in hash table
2. Different types of probing
   -Linear probing
      -when collision happens at k, check k+1 and so on
      -if you reach end of array then wrap around
   -quadratic probing
      -consider elements at k + j^2 (K+1, k+4, k+9…)
   -double hashing
      -increment of 1 for linear probing, j^2 is for quadratic
      Is replaced with a second hash function

lots of problems using open addressing

Hash table 10/16

Wednesday, October 16, 2019 1:57 PM

Removal

- -3 states in each hashtable cell
  - -occupied, neverused, removed
- -when you delete something it needs to be set to removed
  - -if you set it to neverused the program will stop searching for the data
  - -if its removed then it will keep searching until it finds the data somewhere else

Clustering

- -problem with open addressing
- -generates groups of consecutive elements in table

Separate Chaining (bucket approach)

- -second approach to resolve collision
- -each element is a bucket
- -Bucket can be represented as array, list, etc, typically it is a linked list
  - -Search
    - -find bucket with key, look through bucket for element
  - -insert
    - -look for item, insert in the found bucket if not found
  - -remove
    - -look for the item and remove it from the bucket
- -you can keep a bucket in a sorted order if you want

Load factor = measure of cost of collision resolution

- -A = # of entries in table / size of table
- -open addressing = A <= 1
- -separate chaining = A has no maximum value
- -As A increases the number of comparisons increases
- -For reasonable efficiency,
  - -open addressing = A under 0.5
  - -chain addressing = A under 1
- -Rehashing
  - -when the load factor becomes to large, resize the hash table
  - -also compute a new hash index for each key

Hashing in java

- -hashCode() method
  - -part of the object class
  - -returns hashcode for any object in memory

*****if you override equals you need to satisfy the "hash code contract*****

-hash code contract
    -if a = b then a.hashcode = b.hashcode
    -inverse, and converse is not true

-if you override equals it is best to override hashcode as well

Collection & Map 10/21

Monday, October 21, 2019 2:02 PM

Abstract Data types

Map <- Sorted Map <- treemap

<- abstract map <- hashmap <- linkedhashmap

Sets

- Collection of elements
- No duplicates
- No ordering
- Goals
  - be able to find/remove elements quickly
  - Without searching through all elements

- Finding elements is based on equals()
- **Need to define your own equals(object) method**

HashSet

-elements need hashcode() method

LinkedHashSet

-Supports ordering of elements

-elements can be retrieved in order or insertion

TreeSet

-elements must be comparable, implement comparable

-guarantees elements in set are sorted

You can create one type of set out of another

Map

- Maps are not iterable themselves, can iterate through keys or values
- Unordered collection of keys
- Each key has a value
- Can use key to retrieve value Ex/ A["key1"]

-key methods

-void put(K key, V val)

-V get(obj key)

-V remove(obj K)

HashMaps 10/23

Wednesday, October 23, 2019 2:01 PM

-Can store other data structures in the key, value portions of the map

Ex/ HashMap<List<String>, String>

- Can contain a String arraylist in the maps keys
- If the arraylist changes then the key changes
    - meaning the location of the value in the map moves
- If multiple arraylist with nothing inside you will get the wrong values

Algorithmic Complexity 10/23

Wednesday, October 23, 2019 2:24 PM

Class notes pwp is good

Critical Section - portion of code that dominates the runtime

Ex/

```
A
For loop(){
        B       <- critical section
{
C
```

B is executed n times
A, C are executed once

T(n) = 1 + n + 1 = O(n)

Ex /

```
A
For(){
        B
        For(){
                C <- critical section
        }
}
D
```

A, D = once
B = n times
C = n^2 times

T(n) = 1 + n + n^2 + 1 = 1 O(n^2)

Ex/

```
A
For(int I =0; I < n; i++){
        For(int j = I + 1; j < n; j++){
                B
        }
}
```

A = once
B = (n-1) + (n-2)…. 3 + 2 + 1 = 1/2n(n-1)

Trees 10/25

Trees
- -hierarchical data structures
- -one to many relationship

Tree node
- -contains data
- -only 1 parent
- -unlimited children

Sibling = node with same parent
Descendent = children node and their descendants
Subtree = portion of tree that is a tree by itself

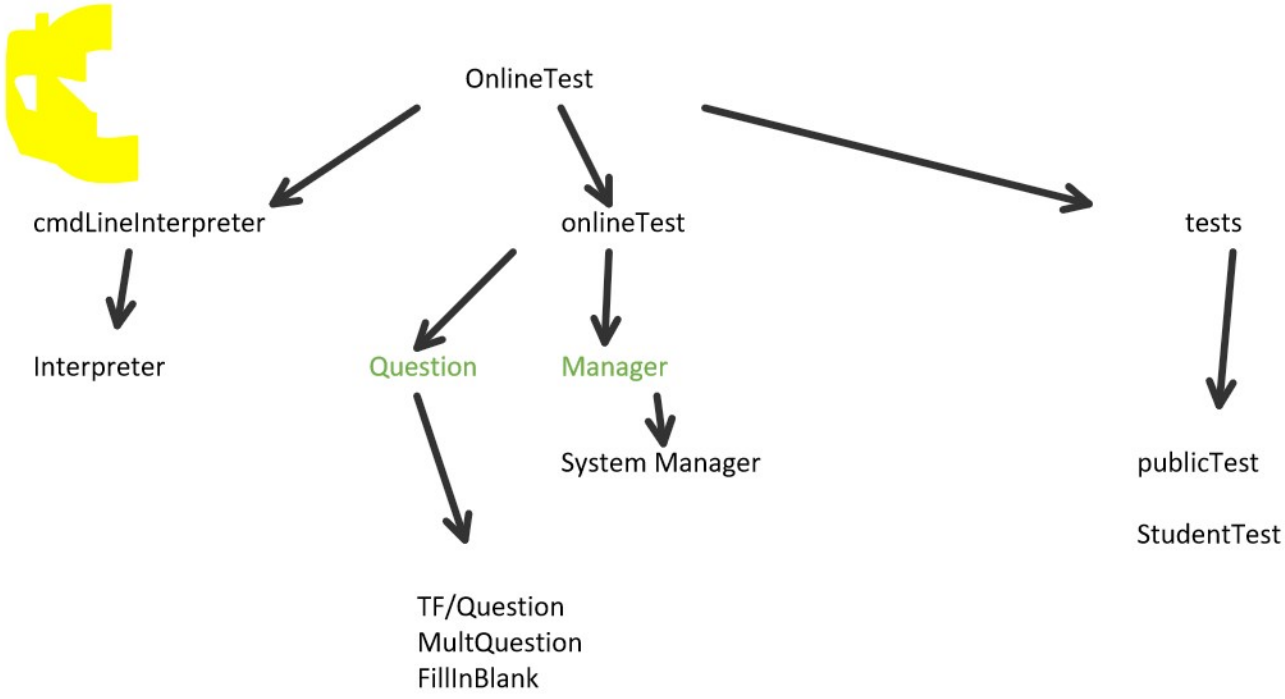
Level = measure of nodes distance from root, prop of node
Height = max level of any node in tree, prop of tree

Binary Tree
- -max of two children from each node

Tree Traversal
- geeksforgeeks
- -Breadth first = closer nodes first, top down
- -Depth first = Pre-order, in-order, post-order
  - -pre-order = Root, Left, Right
  - -in-order = Left, Root, Right
  - -post-order = Left, Right, Root

Proj 6 Design

Monday, October 28, 2019 12:03 PM



Binary Search Tree

Monday, October 28, 2019 1:46 PM

Sorted Binary Tree

Smaller values left subtree
Larger values right subtree

Binary Tree Insertion

Wednesday, October 30, 2019 2:09 PM

Binary Search Properties

-Time of search

-proportional to height

-balance = O(log(n)) time

-degenerate = O(n) time

-Traversal

-O(n)

-Requires

-ability to compare key values

Binary Search Tree Construction

-insertion and deletion

-maintain key property

-smaller values in left

-larger values in right

Binary search tree insertion

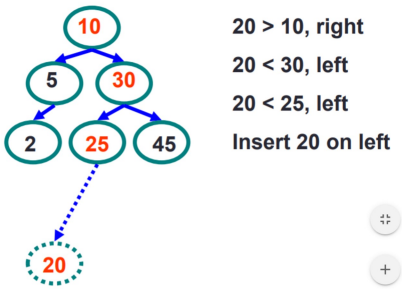
1. Search for value X
2. Search will end at node Y (if X not in tree)
3. If X < Y, insert new leaf X as new left subtree for Y
4. If X > Y, insert new leaf X as new right subtree for Y

-O(log(n)) operation for balanced trees

-insertions may unbalance tree**********

EX/

- Insert (20)



20 > 10, right

20 < 30, left

20 < 25, left

Insert 20 on left

Binary Search Tree Deletion

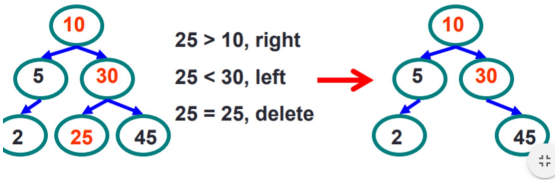
1. Search for value X
2. If X is a leaf, delete X
3. Else
   a. Replace with largest Y on left subtree
      i. Or smalles Z on right subtree
   b. Delete replacement value Y or Z from subtree

-O(log(n)) operation for balanced tree

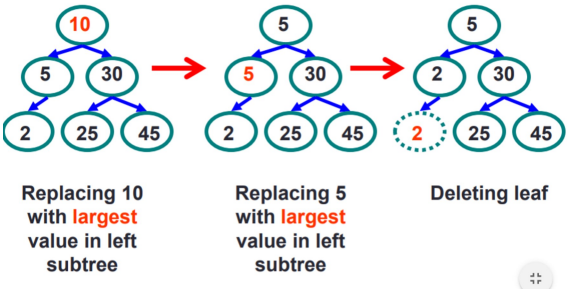
-deletions may unbalance tree**********

EX/

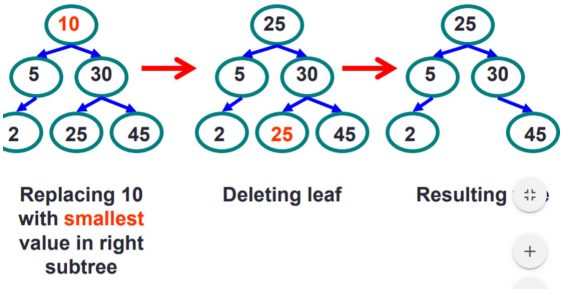
- Delete (25)



- Delete (10)



- Delete (10)

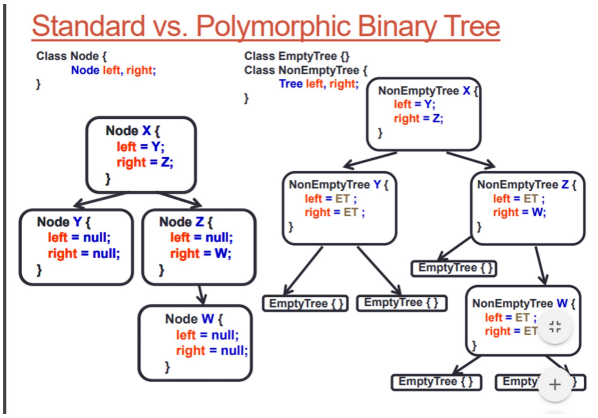


Polymorphic List & Tree

Wednesday, October 30, 2019 2:33 PM

Polymorphism = something occurring in several different forms

Polymorphic tree can hold different data types or trees within itself



Can create polymorphic lists as well

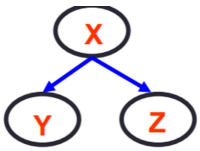
Singleton Design Pattern = one instance of a class or value accessible globally

Heaps and Priority Queues 11/4

Monday, November 4, 2019 2:08 PM

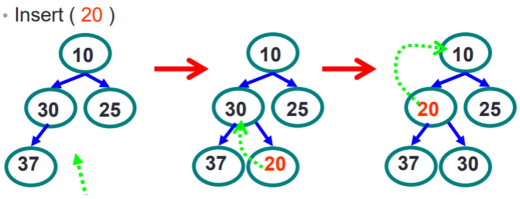
Complete Binary Tree
- -Levels at h are as far LEFT as possible

Heaps, 2 key properties
- -complete binary tree (shape property)
- -value at node (value property)
  - -Miniheap
    - <= to subtrees, (X <= Y, X <= Z)
  - -Maxheap
    - >= to subtrees (X >= Y, X >= Z)



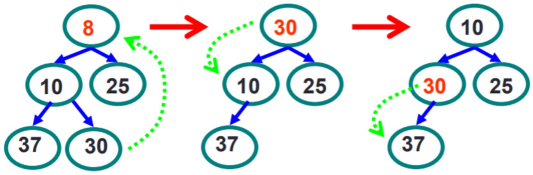
Heaps are BALANCED TREES
- -Height = log2(n) = O(log(n))
- -smallest/largest element always at the top of the heap
- -Heap can track either min or max BUT NOT BOTH

Key operations
- -insert(data d)
  - -**works up the tree**
  - -add d to end of tree
  - -while d < parent
    - swap with parent
  - -O(log(n))



- -getSmallest() = get smallest node at root
  - -**works down the tree**
  - -replace root with x (rightmost node) at end of tree
  - -while x > child
    - -swap x with smallest child
  - -return smallest node found
  - -O(log(n))

- getSmallest ()



Key applications

-heapsort

-priority queue

Heap implementation

-can implement heap as array

Heaps and Priority Queues 11/6

Wednesday, November 6, 2019 2:00 PM

Synchronization 11/13

Wednesday, November 13, 2019 1:59 PM

Data race - problem with multiple threads editing the same thing at the same time
-whoever edits it last overrides any prev edit

Lock - entity that can be held by only one thread at a time
-only one thread can acquire a lock at a time
-enforces mutual exclusivity
-protects critical section

-every java object has a lock
-acquiring lock example
Ex/

```
Object x = new Object();  // We can use any object as "locking object"
synchronized(x) {     // try to acquire lock on x on entry
        ...           // hold lock on x in block
}                     // release lock on x on exit
```

-lock is released when block terminates