1/14

* Data structure
* Algorithms & analysis
* OOP & Java

Collection of items (organized or unorganized)

* Linear
  + List [3] - [4] - [-1] - [8]
  + Stack: last-in-first-out
    - You can only put the item on top of the stacks
    - To take the most bottom, you need to remove all the items
  + Queue: first-in-first-out
    - You can only input from the back and remove from the front
    - Ex: 2,4,-1, 8 (add 5) = 5,2,4,-1,8 (remove) = 5,2,4,-1
* Non-Linear (Hierarchical)
  + Tree
  + Binary Tree
  + Binary Search Tree
* Graph
  + Undirected Graph
  + Directed Graph
* Non-positional (unordered)
  + Map
  + Hash Table

1/25/19

Interface is Abstract Class

Interface – as implements

* As instance variable
* Only method prototypes & constructs
* All methods are abstract
* For design that changes frequently

Abstracts class – partial implements

* Concrete methods & constructors
* Abstract methods
* Used for providing default behavior

A class can

* Inherit from only one class
* Implement many interfaces

1/16/19

Example Ambulance dispatch software

Calls 🡪 records

Location

Description

Priority

Public interface PrioritizedRecordSet{

Void add(record r);

Record getMax(); // removes & add // record with max priority

Int side(); // number of record

}

How it implements?

Array

Strategy 1: records unordered

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 5 | 10 | 2 | 3 |  |  |  |  |  |  |

Add 7 // add element to the end of array

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 5 | 10 | 2 | 3 | 7 |  |  |  |  |  |

GetMax; // search for maximum and remove it

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 5 | 2 | 3 | 7 |  |  |  |  |  |  |

Strategy 2: record ordered

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2 | 3 | 5 | 8 | 10 |  |  |  |  |  |

Add 6 // search and slot in

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2 | 3 | 5 | 6 | 8 | 10 |  |  |  |  |

GetMax //remove the last element

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2 | 3 | 5 | 6 | 8 |  |  |  |  |  |

Which strategy is better?

Faster?

Less memory?

Easier to develop?

package point;

public class Point {

private int x; // instance variables are typically private

private int y;

public Point() // default constructor

{

// x and y get default value 0

}

public Point(int x, int second)

{

this.x = x; // this.x is an instance variable

y = second; // x is a local variable

}

public int getX() // accessor

{

return x;

}

public int getY()

{

return y;

}

public void putX(int x) // mutator

{

this.x = x;

}

public double distance(Point other)

{

double dx = x - other.x; // dx and dy are local

variables

double dy = y - other.y;

return Math.sqrt(dx \* dx + dy \* dy);

}

}