CSE 373

OCTOBER 2ND – DICTIONARY ADT

TODAY'S LECTURE

- **Project 1**
  - JUnit
  - Generics
  - Iterators
- **Dictionary**
  - ADT
  - Implementations
- **Analysis**

OVERLOAD

- **Overload form is out**
  - https://goo.gl/forms/2pFBteeXg5L7wdC12
  - Many of you have already been added
  - If you haven't fill out this form ASAP and we'll fill our remaining seats

PROJECT 1

- Checkpoint 1 due Wednesday
- Remember, 50% of lost points back
- Teams of up to 2, specify clearly

JUNIT: TESTING FRAMEWORK

A Java library for unit testing, comes included with Eclipse

- JUnit is distributed as a "JAR" which is a compressed archive containing Java .class files

```
import org.junit.Test;
import static org.junit.Assert.*;

public class name {
  ...

  @Test
  public void name() { // a test case method
    ...
  }
}
```

A method with @Test is flagged as a JUnit test case and run

JUNIT ASSERTS AND EXCEPTIONS

A test will pass if the assert statements all pass and if no exception thrown.
Examples of assert statements:

- `assertTrue(value)`
- `assertFalse(value)`
- `assertEquals(expected, actual)`
- `assertNull(value)`
- `assertNotNull(value)`
- `fail()`

Tests can expect exceptions

```
@Test(expected = ExceptionType.class)
public void name() {
   ...
}
```

JUNIT

- **Use assertions to prescribe expected behavior**
  - If a test “asserts” something should happen, the test will fail if it doesn’t
- **Use the testing cases from Friday to create good test cases**

JUNIT

- This is new for you, but it is important to learn now.
- Projects will have more testing later in the quarter
- Checkpoint 1 is a good opportunity to experiment and learn the framework on low stakes

GENERICS

- **Projects in this course will use Java generics**
  - Allows implementation of data structures for non-specific data types
  - https://docs.oracle.com/javase/tutorial/java/generics/index.html
  - Oracle tutorial is pretty good here

ITERATORS

- **An iterator is a Java object that goes over a collection of data**
  - Supports two functions
    - `boolean hasNext()`: returns true if the iterator has another object
    - `E next()`: returns the next object from the data structure
      - “E” is a Java generic and it represents whatever data is actually in the data structure.

ITERATORS

- **What is “next”?**
  - Depends on how we want to iterate through the elements
  - Examples:
    - BFSIterator
    - PathIterator
    - DuplicateIterator
    - SortedIterator

DICTIONARY ADT

- New abstract data type

DICTIONARY ADT

- New abstract data type

DICTIONARY ADT

- **New abstract data type**
  - Dictionary (aka Map)
  - Data – Key and Value pairs

DICTIONARY ADT

- **New abstract data type**
  - Dictionary (aka Map)
  - Data – Key and Value pairs
    - Keys: must be comparable, used for lookup

DICTIONARY ADT

- **New abstract data type**
  - Dictionary (aka Map)
  - Data – Key and Value pairs
    - Keys: must be comparable, used for lookup
    - Values: the actual data itself

DICTIONARY ADT

- **New abstract data type**
  - Dictionary (aka Map)
  - Data – Key and Value pairs
    - Keys: must be comparable, used for lookup
    - Values: the actual data itself
  - Example (Store inventory):

DICTIONARY ADT

- **New abstract data type**
  - Dictionary (aka Map)
  - Data – Key and Value pairs
    - Keys: must be comparable, used for lookup
    - Values: the actual data itself
  - Example (Store inventory):
    - Keys: IDs (barcodes)
    - Values: Product information

DICTIONARY ADT

- Operations

DICTIONARY ADT

- **Operations**
  - `insert(key, value):` inserts the key, value pair into the dictionary. Overwrites the old value if the key is already in the dictionary.

DICTIONARY ADT

- **Operations**
  - `insert(key, value):` inserts the key, value pair into the dictionary. Overwrites the old value if the key is already in the dictionary.
  - `find(key)`: returns the stored value for a particular key in the dictionary, returns null if not found.

DICTIONARY ADT

- **Operations**
  - `insert(key, value):` inserts the key, value pair into the dictionary. Overwrites the old value if the key is already in the dictionary.
  - `find(key)`: returns the stored value for a particular key in the dictionary, returns null if not found.
  - `delete(key):` removes the key, value pair denoted by the key from the dictionary.

SET ADT

- Slightly different from Dictionary

SET ADT

- Slightly different from Dictionary
- No values, the set only cares if a key is present or not

SET ADT

- Slightly different from Dictionary
- No values, the set only cares if a key is present or not
- Find, insert and delete have few differences

SET ADT

- Slightly different from Dictionary
- No values, the set only cares if a key is present or not
- Find, insert and delete have few differences
- Possible to implement other functions from sets

SET ADT

- Slightly different from Dictionary
- No values, the set only cares if a key is present or not
- Find, insert and delete have few differences
- Possible to implement other functions from sets
  - Union, intersection, difference

APPLICATIONS

- **Store information in key, value pairs**
  - Very common usage pattern

APPLICATIONS

- **Store information in key, value pairs**
  - Very common usage pattern
    - Phone directories
    - Indexing
    - OS page tables
    - Databases

IMPLEMENTATIONS

- Important to allow fast operations over the keys

IMPLEMENTATIONS

- **Important to allow fast operations over the keys**
  - Dependent on what the client uses most

IMPLEMENTATIONS

- **Important to allow fast operations over the keys**
  - Dependent on what the client uses most
  - Could be many lookups and few inserts

IMPLEMENTATIONS

- **Important to allow fast operations over the keys**
  - Dependent on what the client uses most
  - Could be many lookups and few inserts
- **Keys and Values should be stored together in some way**

IMPLEMENTATIONS

- **Important to allow fast operations over the keys**
  - Dependent on what the client uses most
  - Could be many lookups and few inserts
- **Keys and Values should be stored together in some way**
  - Both objects in one node
  - Paired arrays (one stores keys and the other values)

SIMPLE IMPLEMENTATIONS

- **Linked Lists**
    - How would this work?
    - What other properties can we utilize here?

SIMPLE IMPLEMENTATIONS

- **Linked Lists**
  - How would this work?
  - What other properties can we utilize here?
  - Sortedness? Singly or doubly-linked?
  - Duplicate finding?

SIMPLE IMPLEMENTATIONS

- **Arrays**
  - Sortedness?
  - Resizing?
  - <Key, Value> Pairing?

SIMPLE IMPLEMENTATIONS

- **Are there benefits of one over the other?**
  - Need methods of analytical analysis

ALGORITHM ANALYSIS

- **Important topic. Why?**
  - Show that an implementation is better.

ALGORITHM ANALYSIS

- **Important topic. Why?**
  - Show that an implementation is better.
- **What do we mean by better?**
  - Fewer clock cycles
  - More efficient memory usage
  - Correctness

ALGORITHM ANALYSIS

- **Math review**
- **Logarithms**
  - $\log_2 x = y$ when $x = 2^y$
  - How does this grow?

ALGORITHM ANALYSIS

- **Math review**
- **Logarithms**
  - $\log_2 x = y$ when $x = 2^y$
  - How does this grow? Slowly
  - A balanced tree has a height ~$\log_2 n$
  - $\log_k x$ differs from $\log_j x$ by a constant factor

ALGORITHM ANALYSIS

- **Operations**
  - $\log(A*B) = \log(A) + \log(B)$
  - $\log(A/B) = \log(A) - \log(B)$
  - $\log(A^B) = B * \log(A)$

ALGORITHM ANALYSIS

- Floor and ceiling

ALGORITHM ANALYSIS

- **Floor and ceiling**
  - Integer rounding, computers operate in integer quantities
    - Clock cycles
    - Memory bytes

ALGORITHM ANALYSIS

- **Floor and ceiling**
    - Integer rounding, computers operate in integer quantities
        - Clock cycles
        - Memory bytes

Floor : $\lfloor X \rfloor$ denotes largest integer $\leq$ x

Ceiling: $\lceil X \rceil$ denotes smallest integer $\geq$ x

ALGORITHM ANALYSIS

- Operations

ALGORITHM ANALYSIS

- **Operations**
  - Arithmetic
  - Comparisons
  - Memory reads/writes
- **Loops and functions are just chains of these operations.**

ALGORITHM ANALYSIS

```
Int value = 0;
for(int i; i = 0; i < 10){
     value++;
}
```

ALGORITHM ANALYSIS

```
Int value = 0;
for(int i; i = 0; i < 10){
     value++;
}
```

How long does this take?

ALGORITHM ANALYSIS

```
Int value = 0;
for(int i; i = 0; i < N){
     value++;
}
```

How long does this take?

ALGORITHM ANALYSIS

- Principles of analysis

ALGORITHM ANALYSIS

- **Principles of analysis**
  - Determining performance behavior

ALGORITHM ANALYSIS

- **Principles of analysis**
  - Determining performance behavior
  - How does an algorithm react to new data or changes?

ALGORITHM ANALYSIS

- **Principles of analysis**
  - Determining performance behavior
  - How does an algorithm react to new data or changes?
  - Independent of language or implementation

ALGORITHM ANALYSIS

- Example: find()
- Suppose an array with 5 elements
- One implementation has a sorted array, the other is unsorted
- For which one will find() be faster?
- How long will it take?

ALGORITHM ANALYSIS

- Find(1)

1	2	3	4	5			

4	2	5	3	1			

ALGORITHM ANALYSIS

- Find(1)
- How many operations?

1	2	3	4	5			

4	2	5	3	1			

ALGORITHM ANALYSIS

- Find(4)?

1	2	3	4	5			

4	2	5	3	1			

ALGORITHM ANALYSIS

- Not a good representation of how the algorithm actually behaves.
- Want to access the algorithm on the whole, not just over a few inputs

ALGORITHM ANALYSIS

- Not a good representation of how the algorithm actually behaves.
- Want to access the algorithm on the whole, not just over a few inputs
- This is why testing alone isn’t enough

ALGORITHM ANALYSIS

- Possible solutions?

ALGORITHM ANALYSIS

- **Possible solutions?**
  - Average case: find the average performance over all inputs

ALGORITHM ANALYSIS

- **Possible solutions?**
  - Average case: find the average performance over all inputs
  - Worst case: how long the program takes to complete the worst case problems.

ALGORITHM ANALYSIS

- **Possible solutions?**
  - Average case: can be difficult to compute

ALGORITHM ANALYSIS

- **Possible solutions?**
  - Average case: can be difficult to compute
  - What is the average case for binary search?

ALGORITHM ANALYSIS

- **Possible solutions?**
  - Worst case: is most commonly used

ALGORITHM ANALYSIS

- **Possible solutions?**
  - Worst case: is most commonly used
  - Easily compared and gives a good estimate of the robustness of an algorithm

NEXT CLASS

- **Asymptotic Analysis**
  - Efficiency and runtime
  - bigO notation
  - Array and LinkedList dictionaries