Core Java Course
Data Structures

Dr. Jeff Blessing
MSOE

Agenda

- Vectors
  - ArrayLists
- Hash Tables
- Linked Lists
- Stacks and Queues

2

Why special data structures?

- Any Java class that has attributes (instance variables) can serve as a data structure
- The most challenging thing though is to handle data that has variable size or size unknown at compile time
- Using arrays is one of possible solutions however in many cases arrays just aren't flexible enough
- An array can have one and only one data type for which it was declared (but it can be `Object`)
- If an array grows we have to copy all data over

3

Container Classes

- A **container** is a special data structure that can accommodate a variable number of data objects (instances of other data structures).
- A **container class** is a container data structure plus access methods to handle data objects inside
- Container data classes in Java can accommodate both homogenous (of the same type) and heterogeneous (of more then one type) data objects
- Unlike arrays, container classes are supposed to automatically maintain the "right size"

4

Container Classes (Cont.)

- Java has the following container classes
  - **Vector** - an array-like object that can grow or shrink automatically when new entries are added or old ones are deleted
  - **Hash Table** - a table with a method for directly referencing records by doing arithmetic transformation on keys into table addresses.
  - **Linked List** - a set of data objects arranged very efficiently in memory and allowing for sequential access from one item to the next (or previous). Linked Lists cannot provide direct access to data

5

Vectors

```
import java.util.Vector
```

- Java Vectors are not like vectors in physics and math. The later ones are one-dimensional arrays of integers or floating point numbers, which have fixed size.
- Vectors in Java consist of a variable number of data members, who's type is Object.
- Since all Java classes are derived from class `Object` a Vector can contain objects of any class
- When an object is retrieved from a Vector it has to be cast to its proper type

6

Vectors (Cont.)

- Vectors cannot contain basic data types like integer and floating point numbers. Those numbers have to be wrapped into Integer or Float classes to be used with a vector
- When data outgrows Vector size it reallocates itself to create new capacity.
- Reallocation is a time-consuming process so we wouldn't want to do that frequently
- On the other hand, requesting too much spare capacity just in case waists memory

7

Vectors (Cont.)

- Constructors
  - `Vector();`
    creates an empty vector with initial capacity of 10
  - `Vector(int initialCapacity);`
    creates an empty vector with the specified capacity
  - `Vector(int initialCapacity, int capacityIncrement);`
    creates an empty vector, specifies the initial capacity and capacity growth increment. If capacity increment is not specified it will double every time the existing capacity is exceeded

8

Vectors (Cont.)

- Adding elements
  - `void addElement(Object element);`
    appends an element to the end
  - `int size();`
    returns the number of components in the vector
  - `void setSize(int n);`
    sets the size to be exactly `n` elements. If the original size was less then `n` new elements are allocated, or if it was greater then `n` the vector is truncated
  - `int trimToSize();`
    reduces the vector's capacity to its actual size. **Question**: what is the difference between capacity and size?

9

Vectors (Cont.)

- Vector does not have [] operator, so reaching data is possible only through access methods
  - `void setElementAt(Object elm, int n);`
    puts a new value at certain index. Caution: before using `setElementAt()` make sure that there is an element at that index.
  - `Object elementAt(int index);`
    gets the element's value (can also use `get(int index);`)
  - `void insertElementAt(Object e,int n);`
  - `void removeElementAt(int n);`
    inserts/ removes an element at certain position

10

ArrayLists

- Vectors in Java are thread safe
  - This requires extra overhead
- Most applications are still single threaded
  - Why require all the overhead that's not being used
- ArrayLists are vectors without the synchronization required to make them thread safe

11

BitSet class

- BitSet class is a fixed-size data structure to store and handle a sequence of bits. BitSet is far more efficient then a Vector of Boolean values
  - `BitSet(int numBits);`
  - `boolean get(int bitNumber);`
  - `void set(int bitNumber);`
  - `void clear(int bitNumber);`
  - `void and(BitSet otherSet);`
  - `void or(BitSet otherSet);`
  - `void xor(BitSet otherSet);`

12

Hash Tables

- Hash Tables are convenient way to store data when direct comparison indexing is not possible or too complicated
- A Hash Table consists of a number of "buckets" referenced by an internal index. The external access key is converted into index via some hashing algorithm.
- Hashing significantly reduces number of directly addressable elements

13

Hash Tables (Cont.)

- A good hashing algorithm makes data in the hash table to be distributed evenly, preferably, not more than one element per index
- The situation when we have more then one element per bucket is called ***collision***. Collisions require an additional algorithm to be resolved (possibly, direct comparison)
- Java has built-in support for Hash Tables through its `Hashtable` class

14

Hashtable Class

- Constructors
  - `Hashtable();`
    creates an empty hash table
  - `Hashtable(int initialCapacity);`
    creates an empty hash table with the specified initial capacity (number of buckets)
  - `Hashtable(int initialCapacity, float loadFactor);`
    creates an empty hash table with the specified capacity and load factor. Load factor is the maximum share of non-empty buckets. After load factor is exceeded the table will be re-hashed into a larger one

15

Hashtable Class (Cont.)

- Access functions
  - `Object get(Object key);`
    returns an associated object or `null`
  - `Object put(Object key, Object value);`
    puts a new object at the specified key. If the object already exists it gets replaced and the old value is returned, otherwise, this method returns `null`
  - `Object remove(Object key);`
    removes the key and the object. Returns the value removed. If the key is not in the table does nothing and returns `null`
  - `int size();`
    returns actual size (number of elements) of the table

16

Hashable objects

- In order to be able to use `Hashtable` class the elements of the table need to have `hashCode`() method implemented
- Class Object has a default implementation of `hashCode`() method that is based on object's memory bit pattern. The default algorithm is normally inefficient and needs to be replaced
- Examples of `hashCode`() algorithm
- HashtableTest code example

17

Enumerations

- `Enumeration` is an interface that allows to handle element sets with unknown data types and number of elements. `Enumeration` consists of two methods
  - `boolean hasMoreElements();`
  - `Object nextElement();`
- Enumerations are useful when we want to retrieve all elements from a Vector or a Hashtable (and also a Linked List described below)
  - `Enumeration elements(); //Vector, Hashtable`
  - `Enumeration keys(); // Hashtable`

18

Linked Lists

- The disadvantage of Vectors or Hash Tables is that they have no notion of sequence in which the elements are inserted.
- The data structure that preserves this sequence and allows to access elements in the same order they were added is a Linked List
- To support that data structure in Java a `LinkedList` class can be created
- Linked List Architecture Example
- ListTest Code Example

19

20