CS206

List151 continued

More Exceptions
Testing List151Impl

• Perfect testing would exercise and validate every line of code
  • A perfect test suite can be as hard to write as the code it is testing
  • Alternative: test-driven development
    • write the tests first, then write code that always satisfies all tests
  • Tests should be written pretending you do not have the code, but rather only a pseudocode
  • Tests:
    • Construct: Make different capacities
    • Construct: Hold different object types
    • Add(item): Add 1 item? Two items, Three items (once you get to three you can assume more — kind of proof by induction.)
      • how do you know they are added?
      • Is order preserved?
    • Add(item): what happens when you run out of space?
    • Add(item): wrong type addition should be caught by compiler.
    • Add(index, item): what happens in each index of out range condition?
    • Add(index, item): what happens when there is no room to add?
    • ETC.
public static void main(String[] args) {
    System.out.println("Test A: adding consecutive integers to List151 with capacity of 10
    Result should be 0; 0,1; 0,1,2; etc");
    for (int i = 0; i < 4; i++) {
        List151Impl<Integer> test = new List151Impl<>(10);
        for (int j = 0; j <= i; j++) {
            test.add(j);
        }
        System.out.println("\n"+i+":");
        test.display();
    }

    System.out.println("Test B: Fill a list to capacity, then overfill");
    List151Impl<Integer> test = new List151Impl<>(10);
    for (int i = 10; i < 20; i++) {
        test.add(i);
    }
    System.out.println("Should be numbers 10..19 in positions 0..9");
    test.display();
    System.out.println("\nOverfill!!");
    for (int i = 100; i < 105; i++) {
        if (test.add(i)) {
            System.out.println("Should have returned false!!");
        }
    }
    System.out.println("Should Still be numbers 10..19 in positions 0..9");
    test.display();
}
Analyzing List151Impl

- Design Goals:
  - robustness
    - OK
  - adaptability
    - OK
  - reusability
    - good
- Design principles
  - Modularity
    - good
  - Abstraction
    - good
  - encapsulation
    - good
- Conclusion: On the right track but could be improved

- Robustness
  - only ever throws IndexOutOfBoundsException but this is not appropriate in many cases
    - Users get misleading idea of problem
- Adaptability
  - Once created, the number of items that can be stored is fixed.
    - Cannot grow or shrink.
Growable List151Impl

```java
public boolean add(Y t) {
    if (count >= arra.length)
        return false;
    arra[count] = t;
    count++;
    return true;
}
```

- Rather than returning false, make the underlying array larger
  - Algorithm:
    - Create a new, larger array
    - copy all of the items from the old array to the new array
    - replace the old array with the new array
- This needs to be done in more than one place (both add and add at index), so do this work in a private method
Robustness and Exceptions

• Sometimes there is no existing Exception that fills your need
• Need to create a custom Exception

• Exception is a class just like other so it can be extended just like any other class
MaxSizeExceedededException

- More often than not, custom exception just extends Exception and implements some constructors.

```java
public class MAxSizeExceedededException extends Exception {
    public MAxSizeExceedededException() {
        super("I'm full");
    }
    public MAxSizeExceedededException(String message) {
        super(message);
    }
}
```
public boolean add(int index, Y t) throws IndexOutOfBoundsException, MaxSizeExceeded Exception {
    if (index > count) {
        throw new IndexOutOfBoundsException("Can only add where there are already items");
    }
    if (index < 0) {
        throw new IndexOutOfBoundsException("Cannot store to negative location");
    }
    count++;
    if (count >= arra.length) {
        if (!grow()) {
            throw new MaxSizeExceeded Exception("There is no space left in the Data Structure");
        }
    }
    if (count >= arra.length)
        throw new IndexOutOfBoundsException("No space left");
    for (int i = (count - 1); i >= index; i--) {
        arra[i] = arra[i - 1];
    }
    arra[index] = t;
    return true;
}
public static void main(String[] args) {
    List151Impl<Integer> test = new List151Impl<>(10);
    try {
        for (int i = 0; i < 400; i++) {
            test.add(i);
        }
    } catch (MaxSizeExceeded Exception mxe) {
        System.err.println("They do not fit");
    }
    test.display();
    test.clear();
    test.display();
    try {
        for (int i = 0; i < 40; i++) {
            test.add(i);
        }
    } catch (MaxSizeExceeded Exception mxe) {
        System.err.println("They do not fit");
    }
    System.out.println("\n\nTest 3");
    try {
        for (int i = 1; i < 70; i++) {
            test.add(i*2, i + 2000);
        }
    } catch (MaxSizeExceeded Exception mxe) {
        System.err.println("They do not fit");
    } catch (IndexOutOfBoundsException iobe) {
        System.err.println("Illegal access attempt " + iobe.toString());
    }
    test.display();
}
Exceptions

• Must catch every exception type
  • if a function is said to throw exceptions A and B, need
    • catch (A a) {}
    • catch (B b) {}

• Inside a catch use System.err.print... for printing.

• OK
  • end your program in a catch block (after a a parting message).

• NOT OK
  • to fail to catch an exception and have your program die.
  • throw an exception from the main method
Word Counting

• Task
  • count the number of occurrences of each word in a text
  • “this is a test. this is only a test. this completes the test”
  • <this, 3>, <is, 2>, <a, 2>, <test, 3>, <only, 1>, <completes, 1>, <the, 1>
How

• Always try to reuse things you know.
  • So, store everything in List151Impl
    • not limited by size of array
  • Use KeyValue.java to store the word and counts. word is key, count is value.
    • Adjust KeyValue so values can be changed.
  • Override equals so two KV are equal if their keys are .equal
Main loop for word Counter

```java
void countFile(String filename) {
    try (BufferedReader br = new BufferedReader(new FileReader(filename))) {
        String line;
        while (null != (line = br.readLine())) {
            String[] ss = line.replace("-", " ").split("\s+");
            for (String token : ss) {
                token = token.toLowerCase().replace(".", ".").replace("", ",").replace("?", ".").replace("!", ".");
                if (token.length() > 0) {
                    // take the token i.e. word, lower case it, then get rid of punctuation
                    token = token.toLowerCase().replace(".", ".").replace("", ",").replace("?", ".").replace("!", ".");
                    if (token.length() > 0) {
                        KeyValue<String, Integer> wordS = findWord(token);
                        if (wordS == null) {
                            // if have not already seen the word, add it to the arraylist
                            wordS = new KeyValue<>(token, 0);
                            try {
                                counts.add(wordS);
                            } catch (MAxSizeExceededException mex) {
                                System.err.println("Stopping reading document, cannot hold more words");
                                return;
                            }
                        }
                        wordS.setValue(wordS.getValue() + 1); // increment the number of times the word has been seen
                    }
                }
            }
        }
    } catch (FileNotFoundException e) {
        System.err.println("Error in opening the file:" + filename);
        System.exit(1);
    } catch (IOException ioe) {
        System.err.println("Error reading file " + ioe);
        System.exit(1);
    }
}
```
findWord

• use getInstance()!!!!!

@Override // in KeyValue override equals appropriately
public boolean equals(Object other) {
    if (other instanceof KeyValue<?, ?>) {
        return key.equals(((KeyValue<U,V>) other).key);
    } else {
        return super.equals(other);
    }
}

// in WordCount class just use getInstance
private KeyValue<String, Integer> findWord(String w) {
    KeyValue<String, Integer> fakeInstance = new KeyValue<>(w, 0);
    return counts.getInstance(fakeInstance);
}
Grow

- Algorithm:
  - Create a new, larger array
  - copy all of the items from the old array to the new array
  - replace the old array with the new array
  - If cannot grow return false, otherwise return true.

```java
public class List151Impl<Y> implements List151<Y> {
    /** Do not make a list larger than this */
    private static final int MAX_LIST_SIZE = 408;
    /** The actual number of items stored */
    private int count;
    /** The array in which all the data is actually stored */
    private Y[] arra;
```