ClassWorks

Classwork 1

Example Write a program that prompts a user for a sentence and then outputs the number of unique words in that sentence (i.e., the number of words that have no duplicate). **Hint:** you'll probably want to use the String class's `string.split(" ")` method to get an array of space-separated Strings (words) from the input String.

```
// Examples:
Here is a sentence with all unique words
8

These words repeat words
2

These words are words with more words
4

words words words repeat repeat repeat
0

```

Note: If your solution feels arduous, don't worry -- we'll return to this problem later in the course to see the best way to do it.

Submission

You will be submitting your assignments through Brightspace (access our course through MyLMU > Brightspace and find the submission link)!

What

You will be submitting your 1 source file `UniqueWords.java` file that accomplishes the specification above.

How

To submit this classwork:

1. Find its associated assignment page on this course's presence on Brightspace.
2. If you worked in a group (2 individuals maximum), have a single member upload it for both,

and place both group members' names at the top of *all* submitted files.
3. Upload the files required in the "What" section above.

Answer:

```
/*
 * This program prompts a user for a sentence and then outputs the number of
 * unique words in that sentence (i.e., the number of words that have no duplicate).
 */

import java.util.Scanner;
import java.util.HashSet;

public class UniqueWords {

    public static void main (String[] args) {

        Scanner scanny = new Scanner(System.in);
        String sentence = scanny.nextLine();
        String[] words = sentence.split(" ");

        HashSet<String> repeatedWords = new HashSet<String>();
        HashSet<String> uniqueWords = new HashSet<String>();

        for (int i = 0; i < words.length; i++ ) {

            if (uniqueWords.contains(words[i]) ) {
                repeatedWords.add(words[i]);
            } else {
                uniqueWords.add(words[i]); // uniqueWords will end up having all words
            }
        }
        //Remove repeated words to get unique words
        for (String word: repeatedWords) {
            uniqueWords.remove(word);
        }
        System.out.println(uniqueWords.size());

    }
}
```

Feedback:

10/10 Very nice!

Classwork 2

With the anticipated success of Forneymon (expected to hit stores after a tiny court-battle with Nintendo), the executives of Forney Industries have tasked you with creating a companion virtual card-game.

Towards this end, your boss has given you the following specifications to design the Forneymon card classes, which are going to find initial market penetration in the form of a memory game where cards are initially face-down and then flipped as players try to match them (just to get the brand out).

Important: Make sure to read the entire problem statement before writing any code!

Problem 1[40 Points]

Note: some spec elements are left intentionally vague; you are meant to decide the best class design practices that will complete the project!

Design a class called `ForneymonCard` that adheres to the following specifications.

- Every ForneymonCard has a name and a type of Forneymon (one of three options: "Burnymon", "Dampymon", "Leafymon").
- When creating a new ForneymonCard object, we will populate the fields with its given name and its type of Forneymon.
    - If the card was constructed with a blank name, `throw new IllegalArgumentException();`
    - If the card was constructed with a type that is anything other than the three allowed, `throw new IllegalArgumentException();`
- It's perfectly fine to define multiple constructors for a class, as long as those constructors do not share the same parameterizations.

  ForneymonCards can *also* be default constructed (no arguments given), at which point its name should be "MissingNu" and its type "Burnymon".
- When printed out (i.e., for toString()), a ForneymonCard shall return a String of the format: "Type: NameOfForneymon". E.g., a Burnymon named Burny would return: "Burnymon: Burny"

Problem 2[40 Points]

Design a class called `FlippingForneymonCard` that adheres to the following specifications.

- FlippingForneymonCards are ForneymonCards (share properties and methods) except that they may be placed face-up or face-down. You shall implement this property as a boolean field that is false when face-up and true when face-down.
- FlippingForneymonCards can be constructed as either face-up or face-down as long as a name and type of Forneymon are specified.
- FlippingForneymonCards can be default constructed (no arguments given), at which point its name should be "MissingNu", its type "Burnymon", and face-down.
- Each FlippingForneymonCard can `flip();` which makes it face-up if it was previously face-down, and vice versa. Return true if the card is face-down, and false otherwise.
- Define a method `int match (FlippingForneymonCard other);` that returns:
    - 2 if either it or the other FlippingForneymonCard are face-down.
    - 1 if both are face-up and share the same name *and* Forneymon type.
    - 0 if both are face-up and disagree on either the name *or* the Forneymon type.
- A FlippingForneymonCard that is currently face-down will instead return the string "?: ?" when toString() is called on it.
- A FlippingForneymonCard that is currently face-up will use the default toString() behavior.

Here is an example using the FlippingForneymonCard class:

ForneymonCardTest.java

```java
package forneymon.cardgame;

public class ForneymonCardTest {

    public static void main (String[] args) {
        FlippingForneymonCard burny = new FlippingForneymonCard("burny",
"Burnymon", false);
        // "Burnymon: burny"
        System.out.println(burny);
        burny.flip();
        // "?: ?"
        System.out.println(burny);

        FlippingForneymonCard missingNu = new FlippingForneymonCard();
        // "?: ?"
        System.out.println(missingNu);
        missingNu.flip();
        // "Burnymon: MissingNu"
        System.out.println(missingNu);

        // 2
        System.out.println(burny.match(missingNu));
        burny.flip();
        // 0
        System.out.println(burny.match(missingNu));
    }

}
```

Problem 3[20 Points]

Provide 3 sample unit tests that verify the functionality of your FlippingForneymonCard's match method.

You may choose whatever testing method you please (e.g., JUnit test file or some simple System.out.println statements).

Note: were this not simply an exercise, you should thoroughly test *all* of your methods!

Submission

You will be submitting your assignments through Brightspace (access our course through MyLMU > Brightspace and find the submission link)!

What

You will be submitting your 2 classes in their associated `.java` files, and your 3 unit tests in whatever text document or source file you please.

How

To submit this classwork:

1. Find its associated assignment page on this course's presence on Brightspace.
2. Upload your files to the assignment page.
3. If you worked in a group, have a single member upload it for everyone, and place all group members' names at the top of *all* submitted files.

Answer

ForneymonCard

```java
/**
*Design a class called ForneymonCard that adheres to the following specifications.
*Every ForneymonCard has a name and a type of Forneymon (one of three options:
*"Burnymon", "Dampymon", "Leafymon").
*When creating a new ForneymonCard object, we will populate the fields with its
*given name and its type of Forneymon.
*If the card was constructed with a blank name, throw new IllegalArgumentException();
*If the card was constructed with a type that is anything other than the three
*allowed, throw new IllegalArgumentException();
*ForneymonCards can *also* be default constructed (no arguments given), at which
*point its name should be "MissingNu" and its type "Burnymon".
*When printed out (i.e., for toString()), a ForneymonCard shall return a String
*of the format: "Type: NameOfForneymon". E.g., a Burnymon named Burny would return: "Burnymon: Burny"
*/

public abstract class ForneymonCard {

    //Fields
    private String name;
    private String type;

    //Constructors
    ForneymonCard(String name, String type) {
        if( name.isEmpty() ) {
            throw new IllegalArgumentException();
        } else {
            this.name = name;
        }

        if( type.equals("Burnymon") || type.equals("Leaftymon") || type.equals("Dampymon")) {
            this.type = type;
        } else {
            throw new IllegalArgumentException();
        }
    }

    ForneymonCard() {
        this.name = "MissingNu";
        this.type = "Burnymon";
    }

    //Methods
    @Override
    public String toString() {
        return type + ": " + name;
    }

    //Getters and Setters
    public String getName() {return name;}
    public String getType() {return type;}
}
```

FlippingForneymonCardJunitTests

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

public class FlippingForneymonCardJunitTests {

    @Test
    public void testmatch() {
        FlippingForneymonCard defaultCard = new FlippingForneymonCard();
        FlippingForneymonCard defaultCard2 = new FlippingForneymonCard();
        assertEquals(defaultCard.match(defaultCard2), 2);

        FlippingForneymonCard burny = new FlippingForneymonCard("burny", "Burnymon", false);
        FlippingForneymonCard burny2 = new FlippingForneymonCard("burny", "Burnymon", true);
        assertEquals(burny.match(burny2), 2);
        assertEquals(burny2.match(burny), 2);

        burny2.flip();
        assertEquals(burny.match(burny2), 1);
        assertEquals(burny2.match(burny), 1);

        burny.flip();
        assertEquals(burny.match(burny2), 2);
        assertEquals(burny2.match(burny), 2);

        FlippingForneymonCard missingNu = new FlippingForneymonCard();
        assertEquals(missingNu.match(burny), 2);
        assertEquals(burny.match(missingNu), 2);

    }
}
```

FlippingForneymonCard.java

```java
/**
 * Design a class called FlippingForneymonCard that adheres to the following specifications.
 *
 * FlippingForneymonCards are ForneymonCards (share properties and methods)
 except that they may be placed face-up or face-down. You shall implement this
  property as a boolean field that is false when face-up and true when face-down.
 * FlippingForneymonCards can be constructed as either face-up or face-down as
 long as a name and type of Forneymon are specified.
 * FlippingForneymonCards can be default constructed (no arguments given), at
 *which point its name should be "MissingNu", its type "Burnymon", and face-down.
 * Each FlippingForneymonCard can flip(); which makes it face-up if it was
 *previously face-down, and vice versa. Return true if the card is face-down, and false otherwise.
 * Define a method int match (FlippingForneymonCard other); that returns:
 * 2 if either it or the other FlippingForneymonCard are face-down.
 * 1 if both are face-up and share the same name *and* Forneymon type.
 * 0 if both are face-up and disagree on either the name *or* the Forneymon type.
 * A FlippingForneymonCard that is currently face-down will instead return the
 *string "?: ?" when toString() is called on it.
 * A FlippingForneymonCard that is currently face-up will use the default
 *toString() behavior.
 */

public class FlippingForneymonCard extends ForneymonCard {

    //Fields

    private boolean isFaceDown;

    //Constructors

    FlippingForneymonCard(String name, String type, boolean isFaceDown) {
        super(name, type);
        this.isFaceDown = isFaceDown;
    }

    FlippingForneymonCard() {
        super();
        this.isFaceDown = true;
    }

    //Methods

    public void flip() {
        isFaceDown = !isFaceDown;
    }

```

```java
    public int match (FlippingForneymonCard other) {

        if (isFaceDown || other.isFaceDown()) {
            return 2;
        } else if (super.getName().equals(other.getName())
            && super.getType().equals(other.getType())) {
            return 1;
        } else {
            return 0;
        }
    }

    @Override
    public String toString() {
        return isFaceDown ? "?: ?" : super.toString();
    }

    //Getters and Setters

    public boolean isFaceDown () {return isFaceDown;}
}

```

Feedback

10/10 Perfect! Clean and concise, nice use of a ternary statement

Classwork 3

Step 1: download the partial IntList.java implementation here:

IntList.java

Step 2: implement the following two methods. You may add any private helper methods you like, but do not add any fields nor modify the public interface.

Method
`public void prepend(int toAdd);` Adds the given int toAdd to the first position in the IntList.
`public void insertAt(int toAdd, int index);` Inserts the given int toAdd at the specified index within the IntList. If there are any ints at indices `>= index`, move them one right.

Step 3: make some tests to verify the correct functionality of your code. Do not submit these tests though, they're just for you!

Submission

You will be submitting your assignments through Brightspace (access our course through MyLMU > Brightspace and find the submission link)!

What

You will be submitting your 1 source file `IntList.java` file that accomplishes the specification above.

How

To submit this classwork:

1. Find its associated assignment page on this course's presence on Brightspace.
2. If you worked in a group (2 individuals maximum), have a single member upload it for both, and place both group members' names at the top of *all* submitted files.
3. Upload the files required in the "What" section above.

Answer

```java
public class IntList {

    // Fields
    private int[] items;
    private int   size;
    private static final int START_SIZE = 8;

    // Constructor
    IntList () {
        items = new int[START_SIZE];
        size  = 0;
    }

    public int getAt(int index) {
        indexValidityCheck(index);
        return items[index];
    }

    public void append(int toAdd) {
        checkAndGrow();
        items[size] = toAdd;
        size++;
    }

    public void prepend (int toAdd) {
        insertAt(toAdd, 0);
    }

    public void insertAt(int toAdd, int index) {
        if(index <= size) {
            checkAndGrow();
            shiftRight(index);
            items[index] = toAdd;
            size ++;
        } else {
            throw new IndexOutOfBoundsException();
        }
    }

    public void removeAt(int index) {
        shiftLeft(index);
        size--;
    }

    public int getSize() {
        return size;
    }

    private void indexValidityCheck (int index) {
    if (index < 0 || index >= size) {
        throw new IndexOutOfBoundsException();
    }
}

    /*
     * Expands the size of the list whenever it is at
     * capacity
     */
    private void checkAndGrow () {
        // Case: big enough to fit another item, so no
        // need to grow
        if (size < items.length) {
            return;
        }

        // Case: we're at capacity and need to grow
        // Step 1: create new, bigger array; we'll
```

```java
        // double the size of the old one
        int[] newItems = new int[items.length * 2];

        // Step 2: copy the items from the old array
        for (int i = 0; i < items.length; i++) {
            newItems[i] = items[i];
        }

        // Step 3: update IntList reference
        items = newItems;
    }

    /*
     * Shifts all elements to the right of the given
     * index one left
     */
    private void shiftLeft (int index) {
        for (int i = index; i < size-1; i++) {
            items[i] = items[i+1];
        }
    }

    /*
     * Shifts all elements to the left of the given
     * index one right
     */
    private void shiftRight (int index) {
        for (int i = size; i > index; i--) {
            items[i] = items[i-1];
        }
    }

    @Override
    public String toString(){
        String toOutput = "{ ";
        for(int i = 0; i < size; i ++){
            toOutput += items[i] + " ";
        }
        toOutput += "}";
        return toOutput;
    }
}
```

```java
public class Main{

    public static void main(String[] args) {
        IntList list = new IntList();
        System.out.println(list);

        for(int i = 0; i < 10; i ++){
            list.append(i);
        }
        System.out.println(list);

        for(int i = 0; i < 10; i ++){
            list.prepend(i);
        }
        System.out.println(list);

        list.insertAt(11, 2);
        System.out.println(list);
        list.insertAt(11, 0);
        System.out.println(list);
        list.insertAt(11, list.getSize()-1);
        System.out.println(list);
        list.insertAt(11, list.getSize());
        System.out.println(list);
        list.insertAt(11, list.getSize()+1);
        System.out.println(list);

    }
}
```

Feedback

10/10

Classwork 4

You are designing your own web browser, because why the hell not? Everyone else seems to be doing it these days, and you'll be damned if ForneyFox isn't off the ground before Chrome totally takes over!

So, you decided to start at the basics:

Design a web browser navigation suite that can be used to (1) visit sites, (2) return users to previously visited sites, and (3) move forward to previously visited sites that were returned from (just like how you (1) visit sites on a browser and can (2) hit the back button or (3) hit the forward button).

For now, during development, we'll have users enter their navigation commands by command prompt input.

Here are the viable commands:

- `visit www.site.com`: navigates to the specified URL and is flagged as the "currently viewing"
- `back`: navigates to most recent site on which the above `visit` command was invoked.
- `forward`: navigates to the most recent site from which the above `back` command was used. The "forward" list is wiped after visiting a new site through the `visit` command.
- `exit:` quits the application

These commands are then processed in the main method of your WebNavigator class.

Your task: complete the WebNavigator class outlined below to achieve the above specified behavior. An example useage is found below:

```
// Example Command Line Interaction

> visit www.google.com
Currently Viewing: www.google.com

> visit www.reddit.com
Currently Viewing: www.reddit.com

> back
Currently Viewing: www.google.com

> back
Currently Viewing: www.google.com

> forward
Currently Viewing: www.reddit.com

> forward
Currently Viewing: www.reddit.com

> visit www.facebook.com
Currently Viewing: www.facebook.com

> back
Currently Viewing: www.reddit.com

// Visiting another site after moving back wipes
// the "forward" collection
> visit www.amazon.com
Currently Viewing: www.amazon.com

// See? doesn't go back to reddit
> forward
Currently Viewing: www.amazon.com

> exit
Goodbye!

```

Your application should have the above behavior and you should verify its functionality with extra tests as well!

You may use *any* of the classes in the Java Collections framework for this assignment! Don't get used to the freedom, you can't use it on your homework yet.

Currently, the given shell structures your solution but does not operate as intended. Complete the shell in areas marked with "//TODO":

WebNavigator.java

Here are some added details:

- Before visiting a site, the current site can be considered `null`; users should not be able to use `back` commands to return to null.
- Moving back at the first visited site or forward at the last visited site keeps the current site where it is.
- If you're clever with your data structure choice, you only need to write ~13 - 15 lines of code to complete this assignment.

Need a hint on how your WebNavigator is meant to behave? Just head to your own web browser!

Submission

You will be submitting your assignments through Brightspace (access our course through MyLMU > Brightspace and find the submission link)!

What

You will be submitting your 1 source file `WebNavigator.java` file that accomplishes the specification above.

How

To submit this classwork:

1. Find its associated assignment page on this course's presence on Brightspace.
2. If you worked in a group, have a single member upload it for both, and place both group members' names at the top of *all* submitted files.
3. Upload the files required in the "What" section above.

Answer

```java
import java.util.Stack;
import java.util.Scanner;

public class WebNavigator {

    // Fields
    private String current;
    private Stack<String> history;
    private Stack<String> forward;

    // Constructor
    WebNavigator () {
        this.forward = new Stack<String>();
        this.history = new Stack<String>();
        this.current = null;
    }

    public boolean getNextUserCommand (Scanner input) {
        String command = input.nextLine();
        String[] parsedCommand = command.split(" ");

        // Switch on the command (issued first in input line)
        switch(parsedCommand[0]) {
        case "exit":
            System.out.println("Goodbye!");
            return false;
        case "visit":
            visit(parsedCommand[1]);
            break;
        case "back":
            back();
            break;
        case "forward":
            forw();
            break;
        default:
            System.out.println("[X] Invalid command, try again");
        }

        System.out.println("Currently Visiting: " + current);

        return true;
    }

    /*
     *  Visits the current site, clears the forward history,
     *  and records the visited site in the back history
     */
    public void visit (String site) {
        history.push(site);
        current = site;
        while (!forward.empty()) {
            forward.pop();
        }
    }

    /*
     *  Changes the current site to the one that was last
     *  visited in the order on which visit was called on it
     */
    public void back () {
        if (!history.empty()) {
            forward.push(history.pop());
            if (!history.empty()) {
                current = history.peek();
            }
        }

    }

    public void forw () {
        if (!forward.empty()) {
            history.push(forward.pop());
            if (!forward.empty()) {
                current = forward.peek();
            }
        }
    }

    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        WebNavigator navi = new WebNavigator();

        System.out.println("Welcome to ForneyFox, enter a command from your ForneyFox user manual!");
        while (navi.getNextUserCommand(input)) {}
        //Took this line out because it prints Goodbye two times.
        //System.out.println("Goodbye!");
    }

}

```

Feedback

7/10

Test 1: Visits sites successfully Expected: A B C D E Actual: A B C D E Correct! (2 Points)

Test 2: Can return back to a previously visited site (2 Points) Expected: A B C B A Actual: A B C B A Correct! (2 Points)

Test 3: Can move forward to a site previously returned from (2 Points) Expected: A B C B C Actual: A B C B B Incorrect! (0 Points)

Test 4: Visiting a new site successfully wipes the forward stack (2 Points) Expected: A B C B D B D Actual: A B C B D B B Incorrect! (0 Points)

Test 5: back or forward on empty stack does not crash program (1 Point) Expected: null null Actual: null null Correct! (1 Point)

Test 6: back or forward after first input does not change location (1 Point) Expected: A A A Actual: A A A Correct! (1 Point)

Natalia Dibbern Score: 6 / 10 (+1 DA)

Classwork 5

For each of the following prompts, analyze the **worst-case** asymptotic (Big-O, O(g(n))) runtime complexity of each specified method in the context of its class. The value of n (i.e., the size of the input) is given in the third column of the spec below. **Show your work for each answer.**

Source	Method	n
Classwork 1T Source	`main`	`words.length`
IntList Source	`getAt`	`size`
`insertAt`	`size`	
IntLinkedList Source	`prepend`	`size`
`getIteratorAt`	`index (parameter)`	

Submission

What

You will be submitting 1 plain text (.txt) file with answers and work corresponding to the above.

How

To submit this classwork:

1. Find its associated assignment page on this course's presence on Brightspace.
2. Upload your file itself (i.e., not zipped) on the assignment page.
3. If you worked in a group, have a single member upload it for everyone, and place all group members' names at the top of *all* submitted files. Also, mention in the Brightspace submission comment that you worked in a group, and with whom.

Answer

Classwork 1T: main()

```java
import java.util.Scanner;

  public class UniqueWords {

      public static void main (String[] args) {
          Scanner input = new Scanner(System.in);   // C1
          System.out.println("Enter a sentence.");

          String[] words = input.nextLine().split(" "); // n
          int count = 0;        //C2
          boolean unique;
          for (int i = 0; i < words.length; i++) { //n
              unique = true;    //C3
              for (int j = 0; j < words.length; j++) { //n
                  if (i != j && words[i].equals(words[j])) { //C4
                      unique = false;     //C5
                      break;
                  }
              }
              if (unique) {count++;} //C6
          }
          System.out.println("There are " + count + " unique words in that sentence.");             //C7
          input.close();
      }

  }
```

$T(n) = C1 + n + C2 + n(C3 + nC4 + nC5) + C6 + C7 \implies O(n^2)$

IntList:

insertAt:

```java
    public void insertAt(int toAdd, int index) {
        indexValidityCheck(index, 0, size + 1); //C1
        size++; //C2
        checkAndGrow(); //C3n
        shiftRight(index); //C4n
        items[index] = toAdd; //C5
    }

private void checkAndGrow () {
        if (size < items.length) {  //C1
            return;
        }
        int[] newItems = new int[items.length * 2]; //C2
        for (int i = 0; i < items.length; i++) {   //nC3
            newItems[i] = items[i];     //nC4
        }
        items = newItems; //C5
    }

    private void indexValidityCheck (int index, int lower, int upper) {
        if (index < lower || index >= upper) { //C1
            throw new IndexOutOfBoundsException(); // C2
        }
    }

    private void shiftRight (int index) {
        for (int i = size; i > index; i--) { //nC1
            items[i] = items[i-1]; //nC2
        }
    }
```

checkAndGrow:

$$T(n) = C1 + C2 + nC3 + nC4 + C5 \implies O(n)$$

shiftRight:

$$T(n) = nC1 + nC2 \implies O(n)$$

InsertAt then is:

$$T(n) = C1 + C2 + n(C3 + C4) + C5 \implies O(n)$$

getAt:

```java
    public int getAt(int index) {
        indexValidityCheck(index, 0, size);
        return items[index];
    }
```

$T(n) = C1 \implies O(1)$

IntLinkedList:

Prepend:

```java
    public void prepend (int toAdd) {
        Node currentHead = head; //C1
        head = new Node(toAdd); // C2
        head.next = currentHead; //C3
        size++; //C4
    }
```

$T(n) = C1 + C2 + C3 + C4 \implies O(1)$

getIteratorAt:

```java
    public Iterator getIteratorAt (int index) {
        if (index > size || index < 0) { //C1
            throw new IllegalArgumentException();
        }
        Iterator it = new Iterator ();
        while (index > 0) { //nC5
            it.next(); // n(C2 + C3)
            index--; // n(C4)
        }
        return it;
    }

        public void next () {
            if (current == null) {return;} //C2
            current = current.next; // C3
        }
```

$T(n) = C1 + n(C2 + C3 + C4 + C5) \implies O(n)$

Feedback

10/10 Beautiful submission!

Classwork 6

This is a **BONUS** classwork assignment meant to help you prepare for Exam II. If you do not turn it in, you will not be penalized, and it will not count against your classwork grade. However, for each method you successfully complete below, you will receive 2^n bonus *homework* points, where n is the number of methods (out of a max of 3) that you successfully complete.

Binary Tree Algorithms

For the two methods in the following section, use our `BinaryTreeNode` class, replicated below.

BinaryTreeNode.java

```java
package tree.binary;

public class BinaryTreeNode {

    private String data;
    private BinaryTreeNode left, right;

    BinaryTreeNode (String s) {
        data = s;
        left = null;
        right = null;
    }

    public void add (String s, String child) {
        if (child.equals("L")) {
            left = new BinaryTreeNode(s);
        } else if (child.equals("R")) {
            right = new BinaryTreeNode(s);
        } else {
            throw new IllegalArgumentException();
        }
    }

    public BinaryTreeNode getChild (String child) {
        return (child.equals("L")) ? left : right;
    }

    public String getString () {
        return data;
    }

    public void doubleTree () {
        throw new UnsupportedOperationException();
    }

    public static boolean sameTree (BinaryTreeNode n1, BinaryTreeNode n2)
{
        throw new UnsupportedOperationException();
    }

}
```

Problem 1: Add a method to the BinaryTreeNode class called `doubleTree` that modifies a binary tree by duplicating each node and placing that duplicate at the original's left-child reference. Preserve the tree structure.

```
=== Example ===
This tree:
  2
 / \
1   3

Is doubled to this tree:
       2
      / \
     2   3
    /   /
   1   3
  /
 1

```

Problem 2: Add a method to the BinaryTreeNode class called `sameTree`, which takes as input two references to two BinaryTreeNodes and determines if the two trees are equivalent (same Nodes at each position and same values in each corresponding Node).

You should appropriately test your solutions to the above to verify proper functionality!

Heap Operations

Use our BinaryHeap class (replicated below) for the following problem.

BinaryHeap

```
package tree.heap;

import java.util.ArrayList;

class BinaryHeap {

    ArrayList<Integer> heap;

    BinaryHeap () {
        heap = new ArrayList<Integer>();
    }

    /*
     * Continues to bubble values up the tree until we
     * find a node that is greater than it
     */
    public void bubbleUp (int index) {
```

```java
        if (index == 0) {return;}

        int parent = getParent(index);

        if (heap.get(parent) < heap.get(index)) {
            Integer temp = heap.get(index);
            heap.set(index, heap.get(parent));
            heap.set(parent, temp);
            bubbleUp(parent);
        }

    }

    public void insert (Integer toInsert) {
        heap.add(toInsert);
        bubbleUp(heap.size() - 1);
    }

    // Traversal helpers
    public int getParent (int index) {
        return (index - 1) / 2;
    }

    public int getChild (int index, char child) {
        int result = (index * 2) + 1;
        if (child == 'R') {
            result++;
        }
        return result;
    }

    public void print () {
        for (int i = 0; i < heap.size(); i++) {
            System.out.println(heap.get(i));
        }
    }

    public ArrayList<Integer> getSortedElements () {
        throw new UnsupportedOperationException();
    }

}
```

Problem 3: Implement the `getSortedElements` method, which returns an ArrayList with the heap's elements sorted from least to greatest using the Heap Sort algorithm covered in class. Note: this method should *not* modify the heap itself; you might consider looking up a method in the ArrayList class to help you.

Submission

What

You will be submitting up to 2 .java source files depending on how many methods you completed above. In particular, you will submit your extended BinaryTree.java source and / or your BinaryHeap.java source depending on your work above.

How

To submit this classwork:

1. Find its associated assignment page on this course's presence on Brightspace.
2. Upload your source files (.java) themselves (i.e., not zipped) on the assignment page.
3. If you worked in a group, have a single member upload it for everyone, and place all group members' names at the top of *all* submitted files. Also, mention in the Brightspace submission comment that you worked in a group, and with whom.

Answer

```java
public class BinaryTreeNode {
    private String data;
    private BinaryTreeNode left, right;

    BinaryTreeNode (String s) {
        data = s;
        left = null;
        right = null;
    }

    public void add (String s, String child) {
        if (child.equals("L")) {
            left = new BinaryTreeNode(s);
        } else if (child.equals("R")) {
            right = new BinaryTreeNode(s);
        } else {
            throw new IllegalArgumentException();
        }
    }

    public BinaryTreeNode getChild (String child) {
        return (child.equals("L")) ? left : right;
    }

    public String getString () {
        return data;
    }

    public void doubleTree () {
        if ( this.hasChild("L")) {
            this.left.doubleTree();
        }
        if( this.hasChild("R")) {
            this.right.doubleTree();
        }
        this.doubleNode();
    }

    public static boolean sameTree (BinaryTreeNode n1, BinaryTreeNode n2) {
        if (n1 == null && n2 == null) {return true;}
        if (n1 != null && n2 != null) {
            return (n1.data == n2.data
            && sameTree(n1.left, n2.left)
            && sameTree(n1.right, n2.right));
        }
        return false;
    }

    private void doubleNode() {
        BinaryTreeNode temp = this.left;
        BinaryTreeNode newNode = new BinaryTreeNode(this.data);
        this.left = newNode;
        newNode.left = temp;
    }
    private boolean hasChild(String child) {
        if (child == "L") {return this.left != null;}
        if (child == "R") {return this.right != null;}
        throw new IllegalArgumentException();
    }

    public static void preorderPrint (BinaryTreeNode n) {
     if (n == null) {return;}
     System.out.println(n.data);
     preorderPrint(n.getChild("L"));
     preorderPrint(n.getChild("R"));
    }

    public static void main(String[] args) {
        BinaryTreeNode trial = new BinaryTreeNode("2");
        trial.add("1", "L");
        trial.add("3","R");
        trial.doubleTree();
        preorderPrint(trial);

        BinaryTreeNode trial2 = new BinaryTreeNode("2");
        trial2.add("1", "L");
        trial2.add("3","R");

        BinaryTreeNode trial3 = new BinaryTreeNode("2");
        trial3.add("1", "L");
        trial3.add("3","R");

        System.out.println(sameTree(trial2,trial3));
        System.out.println(sameTree(trial,trial3));
    }
}
```

Feedback

None

Classwork 7

In this classwork, we'll get some practice using the Master Theorem... and by some, I mean a lot! You'll use the Master Theorem quite a bit in your future classes, so better to *master* it now!

For each of the following recurrences, determine (a) if the Master Theorem can be used at all, and if so, (b) which case of the Master Theorem applies, and (c) give its associated bound for T(n).

1. `T(n) = 8 * T(n / 2) + n`
2. `T(n) = 6 * T(n / 3) + n^2`
3. `T(n) = 3 * T(n / 4) + n^3`
4. `T(n) = 0.5 * T(n / 2) + n`
5. `T(n) = 4 * T(n / 4) + √n`
6. `T(n) = T(n) + n/2`
7. `T(n) = 5 * T(n / 5) + n/5`
8. `T(n) = 3 * T(n / 4) + n^0.9`
9. `T(n) = 64 * T(n / 4) + n^3`
10. `T(n) = 64 * T(n / 8) + n^n`

Submission

What

Submit a single text (.txt) or PDF (.pdf) document with your answers and shown work for the above.

How

To submit this classwork:

1. Find its associated assignment page on this course's presence on Brightspace.
2. Upload your file on the assignment page.
3. If you worked in a group, have a single member upload it for everyone, and place all group members' names at the top of *all* submitted files. Also, mention in the Brightspace submission comment that you worked in a group, and with whom.

Answer

Natalia Dibbern

ClassWork 7:

1. `T(n) = 8 * T(n / 2) + n`

   $log_2(8) = 3$ & $d = 1$, decomposition dominant $\implies \Theta(n^3)$
2. `T(n) = 6 * T(n / 3) + n^2`

   $log_3(6) < 2$ & $d = 2$ recombination dominant $\implies \Theta(n^2)$
3. `T(n) = 3 * T(n / 4) + n^3`

   $log_4(3) < 2$ & $d = 3$ recombination dominant $\implies \Theta(n^3)$
4. `T(n) = 0.5 * T(n / 2) + n`

   $a < 1$ we cannot apply master theorem
5. `T(n) = 4 * T(n / 4) + √n`

   $log_4(4) = 1$ & $d = 1/2$ decomposition dominant $\implies \Theta(n)$
6. `T(n) = T(n) + n/2`

   $b = 1$ we cannot apply master theorem
7. `T(n) = 5 * T(n / 5) + n/5`

   $log_5(5) = 1$ & $d = 1$ neutral $\implies \Theta(n \log n)$
8. `T(n) = 3 * T(n / 4) + n^0.9`

   $log_4(3) < 0.9$ & $d = 0.9$ recombination dominant $\implies \Theta(n^{0.9})$
9. `T(n) = 64 * T(n / 4) + n^3`

$log_4(64) = 3$ & $d = 3 \implies \Theta(n^3 \log n)$

10. `T(n) = 64 * T(n / 8) + n^n`

    $d = n$ asymptoticaly will always be larger (recomposition) $\implies O(n^n)$

Feedback

--