CSE 123 Spring 2026 Practice Final Exam

Name of Student: _____

Section (e.g., AA):_____ Student UW **Email** :_____@uw.edu

Do not turn the page until you are instructed to do so.

Rules/Guidelines:

- You must not begin working before time begins, and you must stop working **promptly** when time is called. Any modifications to your exam (writing *or* erasing) before time begins or after time is called will be reported as academic misconduct to the university.
- You are allowed one page of notes, no larger than 8.5 x 11 inches. You may not access any other resources or use any electronic devices (including calculators, phones, or smart watches, among others) during the exam. Using unauthorized resources or devices will be reported as academic misconduct to the university.
- In general, you are limited to Java concepts or syntax covered in class. You may not use `break`, `continue`, a `return` from a `void` method, `try/catch`, or Java 8 stream/functional features.
- You are limited to the standard Java classes and methods listed on the provided reference sheet. You do not need to write import statements.
- If you abandon one answer and write another, ***clearly cross out*** the answer(s) you do not want graded and ***draw a circle or box*** around the answer you do want graded. When in doubt, we will grade the answer that appears in the space indicated, and the first such answer if there is more than one.
- If you require scratch paper, raise your hand and we will bring some to you.
- If you write an answer on scratch paper, please ***write your name and clearly label*** which question you are answering on the scratch paper, and ***clearly indicate*** on the question page that your answer is on scratch paper. Staple all scratch paper you want graded to the ***end*** of the exam before turning in.
- Answers must be written as proper Java code. Pseudocode or comments will not be graded.
- The exam is not graded on code quality. You are not required to include comments.
- You are also allowed to abbreviate `System.out.print` and `System.out.println` as `S.o.p` and `S.o.pln`, respectively. You may **NOT** use any other abbreviations.

Grading:

- There are six problems. Each problem will receive a single E/S/N grade.
- Minor syntax errors will be ignored as long as it is unambiguous what was intended (e.g. forgetting a semicolon, misspelling a variable name where there is only one close option). Major syntax errors, or errors where it is unclear what was intended, may have an impact on your grade.

Advice:

- Read all questions carefully. Be sure you understand the question *before* you begin your answer.
- The questions are not necessarily in order of difficulty. Be sure you at least attempt every question.
- Write clearly and legibly. We cannot award credit for answers we cannot read.
- If you have questions, raise your hand to ask. The worst that can happen is we will say "I can't answer that."
- Ask questions as soon as you have them. Do not wait until you have several questions.

Initial here to indicate you have read and agreed to these rules:

This page is intentionally left blank

Nothing written on this page will be graded

1. Comprehension

Part A - Runtime Analysis

Analyze the worst-case running time of each operation below. Choose the *most accurate* (fastest) correct running time, if more than one choice is correct. Unless otherwise stated, *n* is the length of the input data structure. None of the methods provided throw any exceptions. Assume no resizing is necessary.

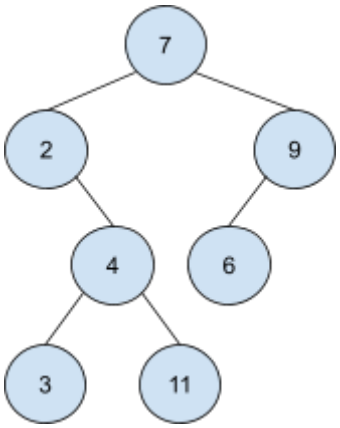
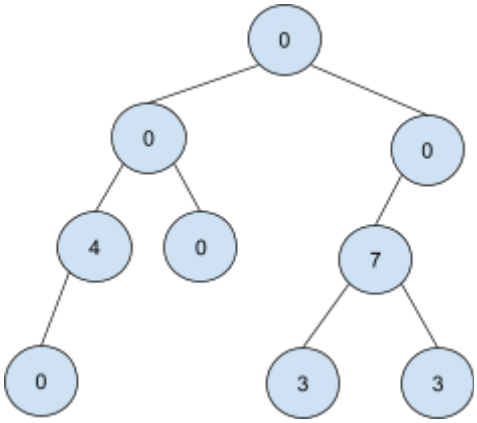
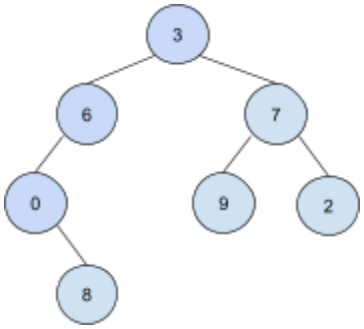
Operation	**O(1)**	**O(*n*)**	**O(n^2)**
Setting the value at the root of a binary tree to a different value.			
Removing a value from an `ArrayIntList`.			
Removing a value from a `LinkedIntList`.			
Adding a value to the end of an `ArrayIntList`.			
Adding a value to the end of a `LinkedIntList`.			
Searching for a value in an array.			

Running the method `m2` below (which invokes `m1`).
`public void m1(int[] data) {`
`  int x = Math.min(data.length, 100);`
`  for (int i = 0; i < x; i++) {`
`  System.out.println(data[i]);`
`  }`
`}`
`public void m2(int[] data) {`
`  for (int i = 0; i < data.length; i++) {`
`  m1(data);`
`  }`
`}`			

Running the method `m1` below.
`public void m1(int[] data) {`
`  for (int i = 1000; i < data.length; i++) {`
`  for (int j = 1000; j < data.length; j++) {`
`  System.out.println(data[j]);`
`  }`
`  }`
`}`			

Part B - Binary Tree Traversals

For each of the following binary trees, indicate which type of traversal is shown: pre-order, in-order, post-order, or none of these.

	7 2 3 4 11 9 6	☐ pre-order ☐ in-order ☐ post-order ☐ none
	0 4 0 0 3 3 7 0 0	☐ pre-order ☐ in-order ☐ post-order ☐ none
	3 6 7 0 9 2 8	☐ pre-order ☐ in-order ☐ post-order ☐ none

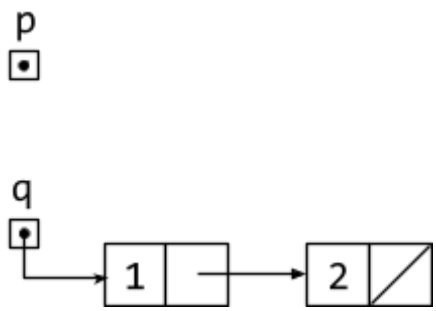
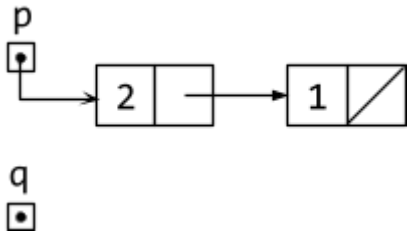
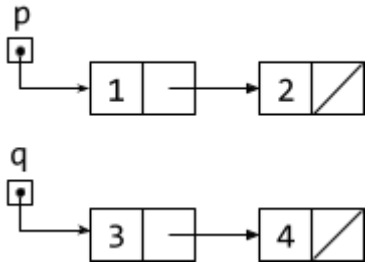
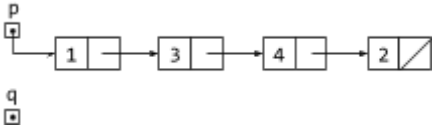
2. Code Tracing

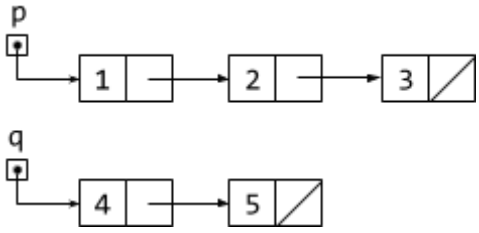
Part A: Linked Node Manipulation

In the following table, the “Before” column shows a diagram of some linked nodes, the “Code” column specifies some code to be applied to the nodes in the before diagram, and the “After” column shows a diagram of the nodes after the code has been applied.

Complete the table, filling in either the before picture, the code, or the after picture. You should not create any new `ListNode` objects or modify any `.data` fields, and **there should be only one instance of each node with a specific value.** The after picture does not need to show any temporary references that were created.

Your `ListNode` diagram format doesn’t have to match that of the problem, so long as it is clear what you intend. In your code, you may use as many temporary references as you’d like to accomplish your goal, but you may *not* create any new `ListNode` objects.

Before	Code	After
		
		

	`p.next.next.next = q;`
	`q = q.next;`
	`q.next = p;`
	`p = p.next;`
	`q.next.next = null;`
	`q = null;`	

Part B: Inheritance Tracing

```
public class Vehicle {
    public void honk() {
        System.out.println("honk");
    }
}

public class Car extends Vehicle {
    public void blinker() {
        blinker(1);
    }

    public void blinker(int n) {
        for (int i = 0; i < n; i++) {
            System.out.println("blink");
        }
    }
}

public class Motorcycle extends Vehicle {
    public void wheelie(int n) {
        System.out.print("W");
        for (int i = 0; i < n; i++) {
            System.out.print("e");
        }
        System.out.println("!");
    }
}
```

```
public class Truck extends Car {
    public void honk() {
        System.out.println("HONK!!!");
    }
}

public class Sedan extends Car {
    public void blinker() {
        blinker(2);
    }
}

public class HondaCivic extends Sedan {
    public void honk() {
        super.honk();
        System.out.println("hoooonkkkk");
    }
}

public class Harley extends Motorcycle {
    public void wheelie() {
        wheelie(5);
    }
}
```

Assume the following variables have been defined:

```
Car var1 = new Sedan();
HondaCivic var2 = new HondaCivic();
Vehicle var3 = new Truck();
Motorcycle var4 = new Harley();
```

Now consider the following code. For each line, place an X next to the correct outcome. If the code runs without error, also indicate what output is printed by filling in the blank. If it produces no output, write "none". If the output includes multiple lines, you may indicate line breaks with a slash character "/" in the output.

`var1.blinker();`		Compile-time error
		Run-time error
		Runs without error and prints: _____

`Sedan sedan = (Sedan) var2;` `sedan.honk();`		Compile-time error
		Run-time error
		Runs without error and prints: _____

`Car car = (Car) var3;` `car.blinker(2);`		Compile-time error
		Run-time error
		Runs without error and prints: _____

`var4.wheelie();`		Compile-time error
		Run-time error
		Runs without error and prints: _____

Part C: Recursive Tracing

Consider the following method:

```
public static boolean mystery(String s) {
    if (s.length() <= 1) {
        return true;
    } else if (s.charAt(0) != s.charAt(s.length() - 1)) {
        return false;
    } else {
        return mystery(s.substring(1, s.length() - 1));
    }
}
```

For each of the following method calls, provide what the `mystery` method will return:

Method Call	Returned Value
mystery(“”);	
mystery(“a”);	
mystery(“ab”);	
mystery(“aba”);	
mystery(“cabac”);	

3. Data Structure Design

You are asked to create a new data structure called `PriorityIntQueue` that represents a priority queue of integers. A priority queue is like a queue (i.e., only allows adding and removing in first-in, first-out or "FIFO" order), but each item is added with a “priority,” and when an item is removed, the highest priority item is removed.

Since we are working with integers, we will consider the **smallest** integer as the highest priority item. For example, consider the following code snippet:

```
PriorityIntQueue pq = new PriorityIntQueue();
pq.add(1);
pq.add(3);
pq.add(2);
pq.remove();  // returns 1
pq.remove();  // returns 2
pq.remove();  // returns 3
```

Now, consider the following incomplete implementation of the `PriorityIntQueue`:

```
public class PriorityIntQueue {
	private IntList list; // the list of data used to store the priority queue

	public PriorityIntQueue() {
		this.list = // TODO: Which implementation should we use?
	}

	// adds an element to the queue
	public void add(int element) {
		// TODO: Implement add
	}

	// removes and returns the smallest integer
	public int remove() {
		if (this.list.isEmpty()) {
			throw new NoSuchElementException();
		}

		list.remove(0);
	}
}
```

Implement a working `add(int element)` so that when `remove()` is called, the smallest value is removed.

What is the runtime of `remove()` if an `ArrayIntList` is used? What is the runtime if a `LinkedIntList` is used?

Likewise, what is the runtime of your `add(int element)` implementation if an `ArrayIntList` is used? What is the runtime if a `LinkedIntList` is used?

	`ArrayIntList`	`LinkedIntList`
`remove()`		
`add(int element)`		

Is one implementation more advantageous than the other? Support your answer by using the runtimes of each implementation. Try to limit your answer to at most five sentences.

4. LinkedList Programming

Consider a method in the `LinkedIntList` (see the reference sheet) class called `pairUp(int value, int target)` that adds a given value in front of all target numbers in the list. For example, if the original contents of the list were:

`list = [1, 7, 1, 1]`

Then, after running `list.pairUp(0, 1)`, the list's contents would be:

`list = [0, 1, 7, 0, 1, 0, 1]`

Part A: `LinkedList` with recursion

Below is a buggy recursive solution to the problem:

```
public void pairUp(int value, int target) {
    pairUp(this.front, value, target);
}
private void pairUp(ListNode curr, int value, int target) {
    if (curr != null) {
        pairUp(curr.next, value, target);
        if (curr.data == target) {
            ListNode curr = new ListNode(value, curr);
        }}} // Parentheses on the same line for the sake of brevity.
```

You decide that changing the return type from `void` to `ListNode` can help us solve the problem. Implement `pairUp` with the provided method designs below. You may not use any loops to solve this problem; you must use recursion.

```
public void pairUp(int value, int target) {

}
private ListNode pairUp(ListNode curr, int value, int target) {
```

Part B: `LinkedList` with iteration

Now implement it iteratively. You should not use any recursion.

```
public void pairUp(int value, int target) {
```

5. Recursive Programming

This problem is about a method called `makePhrases` that takes two parameters: a list of strings named `words`, and an integer named `length`. Your method should print out all possible phrases of exactly `length` characters made up of strings from `words` separated by spaces.

For example, suppose the variable `words` contained a reference to the following list:

```
["i", "am", "a", "cat", "with", "toys"]
```

Then the call `makePhrases(words, 3)` would produce the following output:

```
i a
a i
cat
```

Notice that the spaces between words are included in the character count. For example, the phrase `"i a"` has length 3. Notice also that order matters: `"i a"` and `"a i"` are considered separate phrases.

Similarly, the call `makePhrases(words, 6)` would produce the following output:

```
i am a
i a am
i with
i toys
am i a
am a i
am cat
a i am
a am i
a with
a toys
cat am
with i
with a
toys i
toys a
```

Again, notice that the spaces between words are included in the character count and that order matters.

You may assume that words is not null and that length is greater than or equal to 0. It is acceptable (but not required) for your method to print an extra space at the beginning or end of each phrase. However, this space should not be included in the character count – only spaces between words are included. (That is, you may print `"i a "` or `" i a"` instead of `"i a"`, but `"i a "` and `" i a"` should still be considered to have length 3.)

You may print the possibilities in any order, but to earn an E, you must print all possibilities, and you must not print any possibility more than once.

Fill in the blanks in the partial solution below to implement `makePhrases` and its private helper method.

Do not remove or change any existing lines of code. You may fill in explicitly underlined blanks and add code in the provided whitespace. Some correct solutions may leave some empty spaces blank.

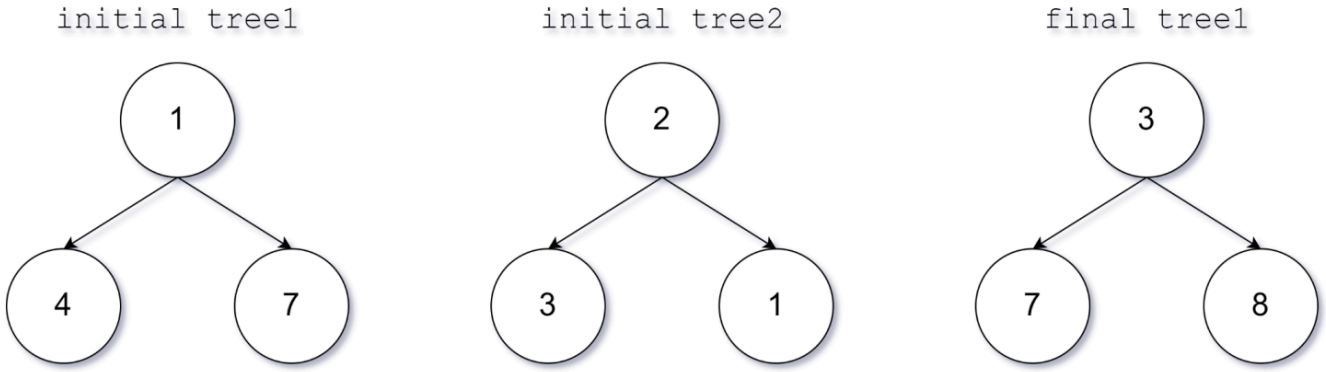
```
public static void makePhrase(List<String> words, int length) {
    makePhrase(words, length, "");
}
private static void makePhrase(List<String> words, int length, String phrase) {




    if (_____________________________________________________) {  // base case



        System.out.println(phrase);
    } else {
        for (int i = 0; i < words.size(); i++) {


                                                                  // choose


            if (_____________________________________________________) {

                makePhrase(_____________________________________);   // explore
            }


                                                                  // unchoose




        }
    }
}
```

6. Binary Tree Programming

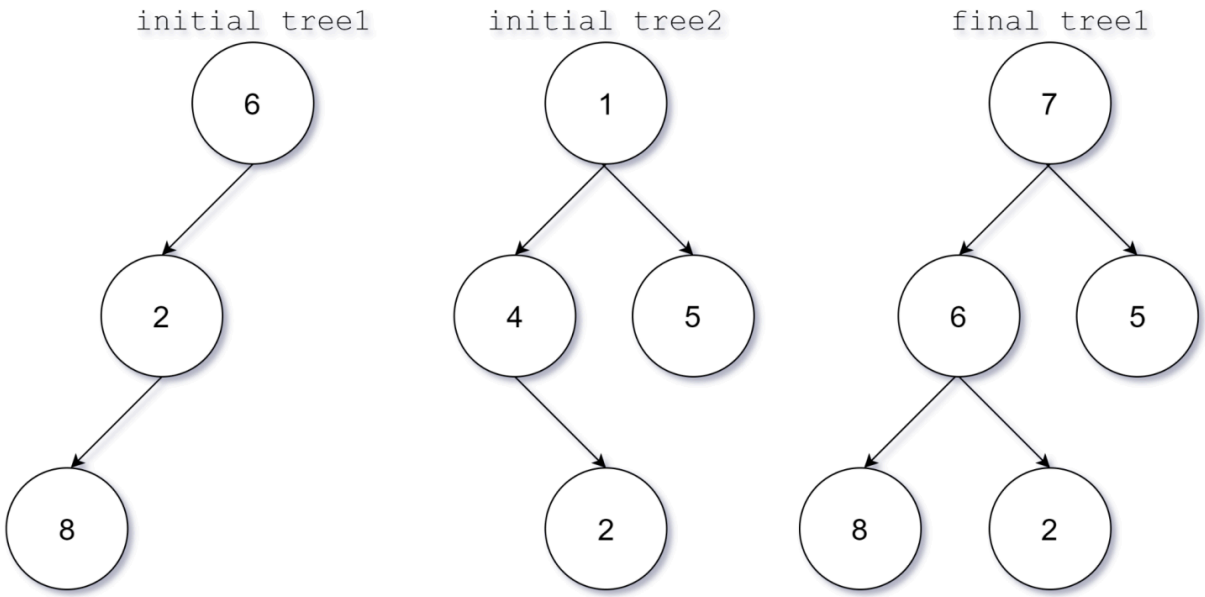
Write a method called `add` that takes as a parameter a reference to a second binary tree and that adds the values in the second tree to this tree. If the method is called as follows:

```
tree1.add(tree2);
```

it should add all values in `tree2` to the corresponding nodes in `tree1`. In other words, the value stored at the root of `tree2` should be added to the value stored at the root of `tree1`, and the values in `tree2`'s left and right subtrees should be added to the corresponding positions in tree1's left and right subtrees. The values in `tree2` should not be changed by your method.



If `tree1` has a node that has no corresponding node in `tree2`, then that node is unchanged. For example, if `tree2` is empty, `tree1` is not changed at all. It is also possible that `tree2` will have one or more nodes that have no corresponding node in `tree1`. For each such node, create a new node in `tree1` in the corresponding position with the value stored in `tree2`'s node. For example:



You are writing a method that will become part of the `IntTree` class. You are free to create as many private helper methods to solve the problem, but you may not call any other method in the `IntTree` class. Lastly, you may not use any auxiliary data structures to solve this problem (e.g., no arrays, `Lists, Stacks, Queues, Strings`, etc.).

```
public void add(IntTree other) {
```

This page is intentionally left blank

Nothing written on this page will be graded

CSE 123 Final Exam Reference Sheet

(DO NOT WRITE ANY WORK YOU WANTED GRADED ON THIS REFERENCE SHEET. IT WILL NOT BE GRADED)

Methods Found in ALL collections (`List`, `Set`, `Map`)

`clear()`	Removes all elements of the collection
`equals(`**`collection`**`)`	Returns `true` if the given other collection contains the same elements
`isEmpty()`	Returns `true` if the collection has no elements
`size()`	Returns the number of elements in a collection
`toString()`	Returns a string representation such as `"[10, -2, 43]"`

Methods Found in both `List` and `Set` (ArrayList, LinkedList, HashSet, TreeSet)

`add(`**`value`**`)`	Adds value to collection (appends at end of list)
`addAll(`**`collection`**`)`	Adds all the values in the given collection to this one
`contains(`**`value`**`)`	Returns `true` if the given value is found somewhere in this collection
`iterator()`	Returns an Iterator object to traverse the collection's elements
`remove(`**`value`**`)`	Finds and removes the given value from this collection
`removeAll(`**`collection`**`)`	Removes any elements found in the given collection from this one
`retainAll(`**`collection`**`)`	Removes any elements *not* found in the given collection from this one

`List<Type>` Methods

`add(`**`index, value`**`)`	Inserts given value at given index, shifting subsequent values right
`indexOf(`**`value`**`)`	Returns first index where given value is found in list (-1 if not found)
`get(`**`index`**`)`	Returns the value at given index
`lastIndexOf(`**`value`**`)`	Returns last index where given value is found in list (-1 if not found)
`remove(`**`index`**`)`	Removes/returns value at given index, shifting subsequent values left
`set(`**`index, value`**`)`	Replaces value at given index with given value

`Map<KeyType, ValueType>` Methods

`containsKey(`**`key`**`)`	`true` if the map contains a mapping for the given key
`get(`**`key`**`)`	The value mapped to the given key (`null` if none)
`keySet()`	Returns a `Set` of all keys in the map
`put(`**`key, value`**`)`	Adds a mapping from the given key to the given value
`putAll(`**`map`**`)`	Adds all key/value pairs from the given map to this map
`remove(`**`key`**`)`	Removes any existing mapping for the given key
`toString()`	Returns a string such as `"{1=90, d=60, c=70}"`
`values()`	Returns a `Collection` of all values in the map

`Math` Methods

`abs(`**`x`**`)`	Returns the absolute value of `x`
`max(`**`x, y`**`)`	Returns the larger of `x` and `y`
`min(`**`x, y`**`)`	Returns the smaller of `x` and `y`
`pow(`**`x, y`**`)`	Returns the value of `x` to the `y` power
`random()`	Returns a random number between `0.0` and `1.0`
`round(`**`x`**`)`	Returns `x` rounded to the nearest integer

`String` Methods

`charAt(`**`i`**`)`	Returns the character in this String at a given index
`contains(`**`str`**`)`	Returns `true` if this String contains the other's characters inside it
`endsWith(`**`str`**`)`	Returns `true` if this String ends with the other's characters
`equals(`**`str`**`)`	Returns `true` if this String is the same as *str*
`equalsIgnoreCase(`**`str`**`)`	Returns `true` if this String is the same as *str*, ignoring capitalization
`indexOf(`**`str`**`)`	Returns the first index in this String where *str* begins (-1 if not found)
`lastIndexOf(`**`str`**`)`	Returns the last index in this String where *str* begins (-1 if not found)
`length()`	Returns the number of characters in this String
`isEmpty()`	Returns `true` if this String is the empty string
`startsWith(`**`str`**`)`	Returns `true` if this String begins with the other's characters
`substring(`**`i, j`**`)`	Returns the characters in this String from index *i* (inclusive) to *j* (exclusive)
`substring(`**`i`**`)`	Returns the characters in this String from index *i* (inclusive) to the end
`toLowerCase()`	Returns a new String with all this String's letters changed to lowercase
`toUpperCase()`	Returns a new String with all this String's letters changed to uppercase
`compareTo(`**`str`**`)`	Returns a negative number if this comes lexicographically (alphabetically) before ot if they're the same, positive if this comes lexicographically after other.

`JUnit` Methods

`assertEquals(`**`expected, actual`**`)`	Tests that expected equals actual (using .equals)
`assertNotEquals(`**`expected, actual`**`)`	Tests that expected doesn't equal actual (using .equals)
`assertSame(`**`expected, actual`**`)`	Tests that expected equals actual (using ==)
`assertNotSame(`**`expected, actual`**`)`	Tests that expected doesn't equal actual (using ==)
`assertTrue(`**`actual`**`)`	Tests that actual is true
`assertFalse(`**`actual`**`)`	Tests that actual is false

Abstract Class Syntax

```
public abstract class AbstractClass{
    // an abstract class can contain fields
    private type name;

    // an abstract class can contain constructors
    public AbstractClass(...) {
        // initialize the object
    }

    public abstract returnType abstractMethod(...);

    public returnType implementedMethod(...) {
        ...
    }
}
```

Inheritance Syntax

```
public class Example extends BaseClass {
    private type field;
    public Example() {
       field = something;
    }
    public void method() {
       // do something
    }
}
```

```
public abstract class AbstractExample {
    private type field;

    public void method() {
       // do something
    }

    public abstract void abstractMethod();
}
```

ArrayIntList Class

```
public class ArrayIntList implements IntList {
    private int[] elementData;
    private int size;

    public static final int DEFAULT_CAPACITY = 10;
    public ArrayIntList() {...}
    public void add(int value) {...}
    public int get(int index) {...}
    public String toString() {...}
    public int indexOf(int value) {...}
    public boolean contains(int value) {...}
    public void add(int index, int value) {...}
    public void remove(int index) {...}
    public void set(int index, int value) {...}
    public int size() {...}
}
```

LinkedIntList Class

```
public class LinkedIntList extends AbstractIntList {
    private ListNode front;

    public LinkedIntList() {...}
    public LinkedIntList(int[] nums) {...}
    public void add(int index, int value) {...}
    public int remove(int targetIndex) {...}
    public int size() {...}
    public int get(int index) {...}

    public static class ListNode {
        public final int data;
        public ListNode next;

        public ListNode(int data) {...}
        public ListNode(int data, ListNode next) {...}
    }
}
```

IntTree Class

```
public class IntTree {
    private IntTreeNode overallRoot;

    public IntTree() {...}
    public IntTree(int[] arr) {...}
    public boolean contains(int value) {...}
    public String toString() {...}
    public void replace(int toReplace, int newValue) {...}

    private static class IntTreeNode {
        public final int data;
        public IntTreeNode left;
        public IntTreeNode right;

        public IntTreeNode(int data) {...}
        public IntTreeNode(int data, IntTreeNode left, IntTreeNode right) {...}
    }
}
```