This exam has 6 questions worth a total of 25 points and is to be completed in 80 minutes. The exam is closed book except for one double-sided, handwritten cheat sheet. No calculators or other electronic devices are permitted. Give your answers and show your work in the space provided. Write the statement below in the blank provided and sign. You may do this before the exam begins.

“I have neither given nor received any assistance in the taking of this exam.”

---

<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
<th>Name</th>
<th>Student ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1/2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1/2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>25</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- There may be partial credit for incomplete answers. Write as much of the solution as you can, but we may deduct points if your answers are much more complicated than necessary.

- **Work through the problems with which you are comfortable first.** Do not get overly captivated by interesting design issues or complex corner cases you’re not sure about.

- Not all information provided in a problem may be useful, and **you may not need all lines.** For code-writing questions, write only one statement per line and do not write outside the lines.

- Unless otherwise stated, all given code on this exam should compile. All code has been compiled and executed, but in the event that we do catch any bugs in the exam, we’ll announce a fix. Unless we specifically give you the option, the correct answer is not, ‘does not compile.’

1. (½ pt) **Your Thoughts** What makes you happy? How do you feel about CS 61BL so far?
2. (½ pt) **So It Begins** Write the statement on the front page and sign. Write your name, ID, and your lab section. Write the names of your neighbors. Write your name in the corner of every page.

3. Evaluate

(a) (4 pts) In each blank, write what Java would display as the result of running the `main` method.

```java
public class Point2D {
    public int x;
    public int y;

    public static void main(String[] args) {
        Point2D p = new Point2D();
        Point2D[] tri = new Point2D[3];
        tri[0] = new Point2D();
        p.x = 1;
        p.y = 3;
        tri[0].x = p.x;
        tri[0].y = p.y;
        p.x = 2;

        System.out.println(p.x + ", " + tri[0].x);  ___ , ___

        tri[1] = p;
        tri[1].y = 4;

        System.out.println(p.y + ", " + tri[0].y);  ___ , ___

        tri[2] = tri[1];
        p.x = 5;
        tri[2] = new Point2D();
        tri[2].x = 0;
        tri[2].y = tri[1].y;

        System.out.println(p.x + ", " + tri[2].x);  ___ , ___
        System.out.println(p.y + ", " + tri[2].y);  ___ , ___
    }
}
```
(b) (4 pts) In each blank, write what Java would display as the result of running the main method.

```java
public class Dog {
    public String name;

    public Dog(String n) {
        name = n;
    }

    public static void flop(Dog a, Dog b) {
        Dog temp = a;
        a = b;
        b = temp;
        System.out.println(a.name);  
    }

    public static void swap(Dog a, Dog b) {
        String temp = a.name;
        a.name = b.name;
        b.name = temp;
        System.out.println(a.name);  
    }

    public static void rename(Dog d, String newName) {
        String oldName = d.name;
        oldName = newName;
        System.out.println(d.name);  
    }

    public static void main(String[] args) {
        Dog a = new Dog("Angie");
        Dog b = new Dog("Boogie");
        Dog c = new Dog("Cocoa");
        Dog d = new Dog("Doggo");
        flop(a, b);
        System.out.println(a.name);  
        swap(c, d);
        System.out.println(c.name);  
        rename(a, d.name);
        System.out.println(a.name);  
    }
}
```
4. Linked Lists

(a) (1 pt) Write a single Java assignment statement that declares and assigns the variable p to have the value shown below.

![Linked List Diagram]

```
IntList p = _____________________________;
```

(b) (4 pts) Implement toIntArray, which returns a new int[] containing the same values as the given SLList. For full credit, use recursion. For up to 3 pts partial credit, use iteration.

**Remember to write only one statement per line. You may not need all the lines.**

```java
public class SLList {
    private static class IntNode {
        public int item;
        public IntNode next;
    }
    private IntNode sentinel;
    private int size;

    public int[] toIntArray() {
        return _____________________________;
    }

    private int[] helper(______________________) {
        ...
        ...
        ...
        ...
    }
}
```
(c) (4 pts) Implement `DLList.duplicate`, which **destructively** copies each node in the list, resulting in a list that's twice as long. For example, if the list initially contained [1, 2, 3], it should contain [1, 1, 2, 2, 3, 3] after the call to `duplicate`.

```java
class DLList<BleepBlorp> {
    private class Node {
        public BleepBlorp item;
        public Node prev;
        public Node next;

        public Node(BleepBlorp i, Node p, Node n) {
            item = i;
            prev = p;
            next = n;
        }
    }
    private Node sentinel;
    private int size;

    public void duplicate() {
        Node p = sentinel.next;

        while (________________________) {
            Node copy = new Node(________________________);

            __________________________;
            __________________________;
            __________________________;
            __________________________;
            __________________________;
            size *= 2;
        }
    }
}
```

5. (0 pts) **PNH** In which Martian region did the Mars Express detect an aurora on 14 August 2004?
6. **Testing & Git** In lab, we designed an `Account` class with an `int balance` and a `parent` account.

(a) (4 pts) Implement `copyAll`, which takes in an `Account[]` and returns a copy of all the accounts in the same sequence. Each account may have a `parent`, and each `parent` may have more parent accounts in a long chain of accounts. Make sure to copy every `parent` account and setup the parent connections so that changing any of the `Account` objects in either array should have no effect on the other.

Assume that there are no null accounts and no two accounts share the same `parent` account.

```java
public class Account {
    public int balance;
    public Account parent;

    public Account(int balance) {
        this.balance = balance;
    }

    public Account(int balance, Account parent) {
        this.balance = balance;
        this.parent = parent;
    }

    public static Account[] copyAll(Account[] accounts) {
        Account[] result = new Account[accounts.length];
        for (int i = 0; i < accounts.length; i += 1) {
            Account acct = accounts[i];
            Account copy = _________________________________;
            _________________________________;
            _________________________________;
            while (_______________________________) {
                _________________________________;
                _________________________________;
                _________________________________;
                _________________________________;
            }
        }
        return result;
    }
}
```
(b) (2 pts) Implement testMystery, a JUnit test for an unknown Account class method called mystery which takes in an Account[] and returns an Account[]. testMystery should make sure that mystery is a non-destructive method and doesn’t change any of the input accounts. You may use Account.copyAll and assume that it is correctly implemented. Assume that the Account class has an equals method so you may use assertArrayEquals for comparing arrays.

```java
public class AccountTest {
    @Test
    public void testMystery() {
        // Randomly generate new Accounts objects for testing purposes.
        Account[] testAccounts = makeTestAccounts();

        // Add test code here...

    }
}
```

(c) (1 pt) A student needs help with Git! git status displays the following message.

```
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
(use "git reset HEAD <file>..." to unstage)
modified:  lab02/Account.java
modified:  lab02/Path.java

Changes not staged for commit:
(use "git add <file>..." to update what will be committed)
(use "git checkout -- <file>..." to discard changes in working directory)
modified:  lab02/Account.java
```

Which one command should they run first to create a single commit with all of lab02?

Choose only one circle and fill it in completely.

- [ ] git reset HEAD lab02/Account.java
- [ ] git add lab02/Account.java
- [ ] git checkout -- lab02/Account.java
- [ ] git commit -m "Completed lab 2"
- [ ] git push origin master