Midterm 1 Practice  
(September 22, 2017)

This exam is closed-book, notes, and technology.

You have 50 minutes to complete this exam.
Please do not open the test until the instructor says time has begun.
Please stop writing once the instructor has called time.
Failure to stop writing will result in a zero on the exam.

Remember you are here to learn.
Relax and think of this as yet another learning experience.

Good luck, have fun!

UID (Not Your Name): ____________________ Solutions

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
</tr>
<tr>
<td>Total</td>
<td>100</td>
</tr>
</tbody>
</table>


Problem 1: The Impossible Waves

(a) Consider the following Java class

```java
public class Point {
    public int x;
    public int y;
    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public void translate(Point p) {
        // Point B
        x += p.x;
        y += p.y;
    }

    public static void run(int x, int y) {
        Point p1 = new Point(x, y);
        Point p2 = new Point(y, x);
        // Point A
        p1.translate(p2);
        // Point C
    }
}
```

Give complete stack-and-heap diagrams outlining the state of memory at each of the given points above. Assume that program execution begins with a call to `run(3, 7)`. The style of your diagrams should be identical to that presented in class. You should reproduce the diagrams completely at each point of the program. (Hint: remember that objects on the heap possess a tag indicating their type.)
Problem 1:  The Impossible Waves

(a) Consider the following Java class:

```java
public class Thingy {
    public static int c;
    public int x;
    public int y;
    public Thingy(int x, int y) {
        this.x = x;
        this.y = y;
        this.c += 1;
    }
}
```

Assume that program execution begins with a call to `new Thingy(3, 7)`. Will the code compile? If so, what will be the value of the fields of the resulting `Thingy` object after the call. If not, how could you correct the code so that it does compile?

If it compiles:

\[
\begin{align*}
    x &= 3 \\
    y &= 7 \\
    c &= 1
\end{align*}
\]

(b) Consider the following Java code. Write the output of each `System.out.println` in `main` in the comments provided within the code.

```java
// in Cell.java
public class Cell {
    public int x;
    public Cell(int x) { this.x = x; }
}

// In Program.java
public class Program {
    public void change1(int x) {
        x = 5;
    }

    public void change2(Cell c) {
        c.x = 5;
    }

    public void change3(Cell c) {
        c.x = 5;
        c = new Cell(0);
    }
}
```
public static void main(String[] args) {
    int x1 = 6;
    change1(x1);
    System.out.println(x1); // Output: 6

    Cell c2 = new Cell(7);
    change2(c2);
    System.out.println(c2.x); // Output: 5

    Cell c3 = new Cell(8);
    change3(c3);
    System.out.println(c3.x); // Output: 5
}

(c) Consider the following Java class

public class Majiggy {
    public static int k = 0;
    public int t;

    public Majiggy(int t){
        t = 5;
        this.t = t;
        k += 1;
    }

    public static void main(String[] args) { // It compiles!
        Majiggy a = new Majiggy(9);
        Majiggy b = new Majiggy(10);

        System.out.println(a.t); // Output: 5
        System.out.println(b.t); // Output: 5
        System.out.println(Majiggy.k); // Output: 2
    }
}

Does this code compile successfully? If so, inline the output of the program in the comments above. If does not compile, state below why this is the case.
Problem 2: Story Time

(a) Consider the following Java interface and classes:

```java
public interface I {
    public void f();
}
public class C1 implements I {
    public void f() { System.out.println("C1.f"); }
    public void g() { System.out.println("C1.g"); }
}
public class C2 implements I {
    public void f() { System.out.println("C2.f"); }
    public void h() { System.out.println("C2.h"); }
}
```

For each of the following variable declarations and method calls, determine if the method call typechecks. If it does, say what the method call prints to the console; if it does not, say in one sentence why it does not typecheck.

- I i = new C1(); i.g(); **Does not typecheck; I has no g method**
- C2 c = new C2(); c.h(); **Typechecks: "C2.h"**

(b) Consider the following Java class that implements a pair of an integer and string:

```java
public class Pair {
    private int fst;
    private String snd;

    public Pair(int fst, String snd) {
        this.fst = fst;
        this.snd = snd;
    }

    public int getFst() { return fst; }

    public String getSnd() { return snd; }

    public String toString() {
        return "(" + fst + ", " + snd + ")";
    }
}
```

Modify the Pair class so that it is generic in its carrier type. Inline your changes into the code listing above.
Problem 3: Origami Birds

For each of the following methods:

1. Give a mathematical function or recurrence that models the time complexity of the method. State explicitly what operations your function tracks as well as what the input to the function represents. If your model is a recurrence relation, solve that relation for an explicit mathematical function.

2. Give a tight upper-bound for your function using Big-O notation. You can simply state the upper-bound rather than formally proving it correct.

```java
public static int f1(int[] arr1, int[] arr2) {
    if (arr1.length != arr2.length) { throw new IllegalArgumentException(); }
    for (int i = 0; i < arr1.length; i++) {
        int temp = arr1[i];
        arr1[i] = arr2[i];  // |
        arr2[i] = temp;    // |
    }
    
    T(n) = #/ array assignments
    n = length of arr1 (also arr2)
    
    T(n) = 2n
    T \in O(n)
```

```
// pre: n \geq 0
public static boolean f2(int n) {
    if (n == 0) {
        return true;
    } else if (n == 1) {
        return false;
    } else {
        return f2(n - 2);
    }

    T(n) = # / subtractions
    n = magnitude of input to f2

    (Assume 2 divides n)
    T(0) = 0
    T(n) = 1 + T(n-2)

    Let k = # / unfoldings:
    T(n) = k + T(n-2k)
    Substituting for k:
    T(n) = \frac{n}{2} + T(n-2\frac{n}{2})
    = \frac{n}{2} + T(0)
    = \frac{n}{2}

    So T \in O(n)
```
Problem 4: Showdown with Grandfather

Write a complete Java class, BankAccount, that represents a bank account. A bank account records the account number and a history of the transactions made from this account. Your class should support the following constructor and methods:

- **BankAccount(int accountNum, int initial)**: creates a new bank account with the given name, account number, and initial balance.
- **int getBalance()**: returns the balance of this account.
- **void deposit(int amount)**: adds a transaction that deposits the given amount to the account. Throws an IllegalArgument Exception if the amount is negative.
- **void withdraw(int amount)**: adds a transaction that withdraws the given amount to the account. Throws an IllegalArgument Exception if the amount is negative or if the withdrawal would make the balance negative.
- **String toString()**: returns a string representation of the account of the form:

  "Account <account number>: <transactions>"

  where "<transactions>" is a comma-separated list of the transactions in-order, starting with the initial balance, e.g., "Account 841011: 300, -100, 250, -50". Note that deposits are positive amounts and withdrawals are negative amounts.

To author this class, you should use the List<T> interface and one of its concrete implementations, ArrayList<T> or LinkedList<T> from the standard library. The following methods of the List<T> interface will be useful here:

- **void add(T v)**: adds the given element to the end of the list.
- **T get(int index)**: retrieves the element found at the given index in the list.
- **int size()**: returns the number of elements in the list.

```java
public class BankAccount {
    private int acct;
    private List<Integer> transactions;

    public BankAccount(int acct, int initial) {
        this.acct = acct;
        transactions = new ArrayList<>();
        transactions.add(initial);
    }

    public int getBalance() {
        int balance = 0;
        for (int i = 0; i < transactions.size(); i++) {
            balance += transactions.get(i);
        }
        return balance;
    }

    public void deposit(int amount) {
        if (amount < 0) throw new IllegalArgumentException();
        else transactions.add(amount);
    }

    public void withdraw(int amount) {
        if (amount > getBalance()) throw new IllegalArgumentException();
        else transactions.add(-amount);
    }

    public String toString() {
        String ret = "Account "+ acct + ": ";
        for (int i = 0; i < transactions.size(); i++) {
            ret += " + "+ transactions.get(i);
        }
        return ret;
    }
}
```
Problem 5: While My Guitar Gently Weeps

Consider the following implementation of a generic linked list:

```java
public class Node<T> {
    public T value;
    public Node<T> next;
    public Node(T value, Node<T> next) {
        this.value = value;
        this.next = next;
    }
}

public class LinkedList<T> {
    private Node<T> first;
    // ...
}
```

Write a method for the LinkedList<T> class called `intersperse` that takes a value `v` of type `T` and places `v` between every element of the list. If the list contains zero or one element, then the method does nothing to it. For example, if the list contains the elements `[3, 6, 8, 4, 1]`, then `intersperse(0)` modifies the list so that it becomes `[3, 0, 6, 0, 8, 0, 4, 0, 1]`. You may not use any additional methods of the LinkedList<T> class in your code.

```java
public void intersperse(T v) {
    if (first != null) {
        Node<T> cur = first;
        while (cur.next != null) {
            Node<T> ins = new Node<>(v, cur.next);
            cur.next = ins;
            cur = cur.next.next;  // hop 2
        }
    }
}
```