CMSC 132A, Midterm 1 (Practice)
SOLUTION

Spring 2018

NAME:_____

UID: _____

Question	Points
1	15
2	15
3	15
4	15
Total:	60

This test is open-book, open-notes, but you may not use any computing device other than your brain and may not communicate with anyone. You have 50 minutes to complete the test.

The phrase "design a program" means follow the steps of the design recipe. Unless specifically asked for, you do not need to provide intermediate products like templates or stubs, though they may be useful to help you construct correct solutions.

You may use any of the data definitions given to you within this exam and do not need to repeat their definitions.

Unless specifically instructed otherwise, you may use any Java language features we have seen in class.

You do not need to write visibility modifiers such as `public` and `private`.

When writing tests, you may use a shorthand for writing check-expects by drawing an arrow between two expressions to mean you expect the first to evaluate to same result as the second. For example, you may write `"foo".length()` $\rightarrow$ `3` instead of `t.checkExpect("foo".length(), 3)`. You do not have to place tests inside a test class.

Problem 1 (15 points). A complex number can be represented as the combination of two real numbers, one for the "real" part and one for the "imaginary" part. Using two numbers of type `Double`, design a class representation of complex numbers. Be sure to include the constructor for complex numbers.

Design a method called `add` that adds together complex numbers. The sum of two complex numbers is a complex number whose real part is the sum of the real parts and imaginary part is the sum of the imaginary parts.

`SOLUTION:`

```
class Complex {
  Double real;
  Double imag;
  Complex(Double real, Double imag) {
    this.real = real;
    this.imag = imag;
  }

  // Add this and given complex numbers
  Complex add(Complex c) {
    return new Complex(this.real + c.real, this.imag + c.imag);
  }
}

new Complex(1, 2).add(new Complex(3, 4)) --> new Complex(4, 6)
```

Problem 2 (15 points). A phrase is a sequene of one or more words. Design a representation for phrases and a method for computing the number of words in a phrase.

SOLUTION:

```
interface Phrase {
  // Count the number of words in this single-word phrase
  Integer countWords();
}

class Last implements Phrase {
  String word;
  Last(String word) {
    this.word = word;
  }

  // Count the number of words in this single-word phrase
  Integer countWords() {
    return 1;
  }
}

class More implements Phrase {
  String first;
  Phrase rest;
  More(String first, Phrase rest) {
    this.first = first;
    this.rest = rest;
  }

  // Count the number of words in this multi-word phrase
  Integer countWords() {
    return 1 + this.rest.countWords();
  }
}
```

Problem 3 (15 points). Java has a built-in interface for representing predicates, which are functions of one argument that return either `true` or `false`. Here is its definition:

```
interface Predicate<T> {
  // Does this predicate hold on t?
  Boolean test(T t);
}
```

For example, here is a predicate on strings that determines if a string has 180 characters or fewer:

```
class Tweetable implements Predicate<String> {
  // Is the given string 180 characters or fewer?
  Boolean test(String s) {
    return s.length() <= 180;
  }
}
```

Here is another that determines if a string ends with `"!"`:

```
class Emphatic implements Predicate<String> {
  // Does the given string end with "!"?
  Boolean test(String s) {
    return s.endsWith("!");
  }
}
```

Design a class called `And` that implements `Predicate<T>` for any type `T`. When constructed, it should be given two predicates (on type `T`), and its `test` method should produce true if and only if both of these predicates produce true on the given argument.

[Space for problem 3.]

SOLUTION:

```
class And<T> implements Predicate<T> {
  Predicate<T> p1;
  Predicate<T> p2;
  And(Predicate<T> p1, Predicate<T> p2) {
    this.p1 = p1;
    this.p2 = p2;
  }

  // Do both of these predicates hold on t?
  Boolean test(T t) {
    return this.p1(t) && this.p2(t);
  }
}

new And<String>(new Tweetable(), new Emphatic()).test("Yo!")  --> true
new And<String>(new Tweetable(), new Emphatic()).test("No")   --> false
```

Problem 4 (15 points). Here is a data definition for binary trees of integers:

```
interface BT {}

class Leaf implements BT {}

class Node implements BT {
  Integer n;
  BT left;
  BT right;
  Node(Integer n, BT left, BT right) {
    this.n = n;
    this.left = left;
    this.right = right;
  }
}
```

Design a method `has` that is given a number and determines if the number appears in the binary tree.

You do not need to rewrite the interface and class definitions. Instead, for each piece of code, label which interface or class it belongs to.

[Space for problem 4.]

SOLUTION:

```
// BT
// Does this tree contain i?
Boolean has(Integer i);

// Leaf
// Does this leaf contain i?
Boolean has(Integer i) {
  return false;
}

// Node
// Does this node contain i?
Boolean has(Integer i) {
  return this.n.equals(i) ||
         this.left.has(i) ||
         this.right.has(i);
}
```