CS 61B Spring 2024

Midterm 1

Thursday February 15, 2024

Intro: Welcome to CS61B Midterm 1!

Your Name: Solutions

Your SID: _____ Location: _____

SID of Person to your Left: _____ Right: _____

Formatting:

- ◯ indicates only one circle should be filled in. ☐ indicates more than one box may be filled in. **Please fill in the shape completely.** If you change your response, **erase as completely as possible**.
- Anything you write that you ~~cross out~~ will not be graded.
- You may not use ternary operators, lambdas, streams, or multiple assignment.

Tips:

- This midterm is worth **100 points**, and there are a lot of problems on this exam. Work through the ones 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.
- **We will not give credit for solutions that go over the number of provided lines or that fail to follow any restrictions given in the problem statement.**
- There may be partial credit for incomplete answers. Write as much of the solution as you can, but bear in mind that we may deduct points if your answers are much more complicated than necessary.
- Unless otherwise stated, all given code on this exam compiles. All code has been compiled and executed before printing, but in the unlikely event that we do happen to catch any bugs in the exam, we'll announce a fix.

Write the statement below in the same handwriting you will use on the rest of the exam: "I have neither given nor received help on this exam (or quiz), and have rejected any attempt to cheat. If these answers are not my own work, I may be deducted 9,876,543,210 points on the exam."

_____

_____

_____

_____

_____

Signature: _____

1 WWJD

(15 Points)

For each of the following lines, write the result of each statement: Write "CE" if a compiler error is raised on that line, "RE" if a runtime error occurs on that line, "OK" if the line runs properly but doesn't print anything, and the printed result if the line runs properly and prints something. If a line errors, assume that the program continues to run as if that line did not exist. Blank lines will receive no credit.

```
1  public interface Sphere {
2       default void getRadius() { System.out.println(0); }
3  }
4
5  class Planet implements Sphere {
6      private int radius = 5;
7
8      @Override
9      public void getRadius() { System.out.println(this.radius); }
10
11     public void orbit(Planet p) { System.out.println("orbit planet"); }
12 }
13
14 class Exoplanet extends Planet {
15     static int distance = 10;
16
17     public void getDistance() { System.out.println(distance); }
18
19     @Override
20     public void orbit(Planet p) { System.out.println("orbit exoplanet"); }
21 }
```

(a) Fill in the blanks below:

```
Class Main {
    public void main(String[] args){
        Planet kepler = new Exoplanet();              1: OK
        Sphere jupiter = new Sphere();                2: CE
        Sphere arion = new Exoplanet();               3: OK
        Planet earth = new Planet();                  4: OK

        kepler.orbit((Exoplanet) arion);              5: Orbit Exoplanet
        ((Exoplanet) earth).orbit(kepler);            6: RE
        ((Exoplanet) kepler).getDistance();           7: 10
        earth.getRadius();                            8: 5
        arion.getDistance();                          9: CE
        earth.radius;                                 10: CE
        Exoplanet.getDistance();                      11: CE
    }
}
```

Solution:

1. This is ok for Java as we are assigning a variable with a static type of a Planet to be an Exoplanet. This is fine in compile time as the variables static type, Planet, is a Superclass of the newly instantiated objects class, Exoplanet. This is also why it works for runtime!

2. Sphere is an interface. As an interface it does not have a constructor so "new Sphere()" gives a compiler error as it is calling a constructor which does not exist!

3. Similar to line 1, this is ok because of the superclass-subclass relationship between the static type of the variable and the newly instantiated object. Exoplanet is a subclass of Planet which implements Sphere.

4. This is ok as it is assigns a variable to a newly instantiated object of the same class of its static type.

5. In compile time using kepler's static type. Planet, we lock in the function Signature of void orbit(Planet p). We do this despite it being arion being casted as an exoplanet. Why? Because there is no orbit method that takes in an Exoplanet, but there is one that takes in plane and we can assign a planet to be an exoplanet! At runtime we go to kepler's dynamic type and look for a method signature that is the same as the one we locked at compile time. We then find it in the dynamic types class, Exoplanet, and use that method.

6. We first check the casting of earth to check its validity. In compile time it finds that the casting and the static type of earth are related to each other via superclass and subclass, but it does not check which one is a subclass of the other. In Runtime, we check earth's dynamic type (Planet) and find that it is not a subclass of Exoplanet, therefore giving us a runtime error.

7. Like line 6, we check the casting of kepler. The compile time finds that they have a subclass superclass relationship (as planet is the superclass of exoplanet). But unlike A.6 we do not get a runtime error because we find that the dynamic type of kepler matches the Exoplanet type! Moving on to method selection, in compile time we go to exoplanet's class and look for the getDistance() Method and successfully find it! We then lock in the method signature void getDistance() at the end of compile time. Moving on to runtime, we go to kelper's dynamic type (Exoplanet) and search for the method signature we locked in at compile time. We then find it in Exoplanet.getDistance() and print the static

integer distance as described in the method. Since the static variable hasn't been modified we print out 10.

8. In Compile Time we go to earth's static type, Planet, and lock in the method signature void getRadius(). Then in Runtime we go to Earth's dynamic type, Planet, and search for the same method signature. In this case this happens to be the same method so we execute the body of that function, which prints out the radius giving us a result of 5.

9. In compile time we go to the static type of arion which happens to be Sphere. We then, within that interface, look for the method getDistance(). But there is no method with that name defined in the Sphere interface therefore giving us a compiler error!

10. This returns a compiler error as it is trying to access a private instance variable outside of the class. The only way to access instance variables is if they are noted as public when they are declared, therefore causing the compiler to give an error.

11. This also returns a compiler error because the method getDistance() in the Exoplanet class is not declared to be static. The only way for a class to use a method is if the method is declared to be static when created.

(b) Within the blank below, briefly explain why changing the access modifier on line 2 to, `private default void getRadius()`, would result in a compilation error. Answer in 10 words or less.

Methods in an interface must be public.

Solution: In Java, the methods in an interface are by-default public, even when unspecified. Furthermore, interface methods must be public. The reasoning behind this is that an interface expects its methods to be overridden, and if the methods were private then it would mean that other classes can't access the methods they are supposed to overwrite!

(c) Within the class `Exoplanet`, we define a new method below. Which of the following changes will make `Exoplanet` compile?

```
public void fixRadius() { super.radius = 6; }
```

- ● Change the `private` keyword to `public` on line 6.
- ○ Modify `fixRadius()` above to be `private`.
- ○ Modify `fixRadius()` above to be `static`.

Solution: Since Exoplanet is a subclass, it is a different class. This means that it would not be able to access radius as it is a private instance variable, which means that other classes can't access it! This means that if we keep the private keyword then the new method is accessing an instance variable which it cannot do.

(d) True or false: we can create another interface that extends `Sphere`.

- ● True
- ○ False

Solution: An interface can extend another interface – they do not implement each other as interfaces generally do not provide any of the implmenetation for the methods specified.

(e) It is _____ possible to implement two interfaces in the same class.

- ◯ Always
- ● Sometimes
- ◯ Never

Solution: If the two interfaces have conflicting method signatures but different return types, then it is not possible to implement both interfaces in the same class.

2 Comparable Computers

(20 Points)

AJ′s computer versions are represented by a string containing non-negative integers where each term is separated by periods, for example: "123.4.5". Implement the `compareTo` method in the `Computer` class to compare computer versions. A term with the higher numerical value is considered greater. The version with the larger, most significant term is considered greater. See examples below for expected behavior.

Hint: Use `Character.getNumericValue(char c)`, which returns the integer value of the char, so e.g. '0' becomes 0.

```
// Test Case 1: Different numerical values
String str1 = "1.2.3";
Computer c1 = new Computer(str1);
String str2 = "1.2.4";
Computer c2 = new Computer(str2);
System.out.println(c1.compareTo(c2)); // Output is less than zero (1.2.3 < 1.2.4)

// Test Case 2: Different lengths
String str3 = "1.0.0";
Computer c3 = new Computer(str3);
String str4 = "1";
Computer c4 = new Computer(str4);
System.out.println(c3.compareTo(c4)); // Output is zero (1.0.0 == 1)

// Test Case 3: Different numerical values
String str5 = "2.0";
Computer c5 = new Computer(str5);
String str6 = "1.999";
Computer c6 = new Computer(str6);
System.out.println(c5.compareTo(c6)); // Output is greater than 0 (2.0 > 1.999)
```

```
public class Computer implements Comparable<Computer> {

    String version;

    public Computer(String version) {
        this.version = version;
    }

    @Override
    public int compareTo(Computer other) {
        int i = 0, j = 0;

        while (i < this.version.length() || j < other.version.length()) {
            int num1 = 0, num2 = 0;

            while (i < this.version.length() && this.version.charAt(i) != '.') {
                char c = this.version.charAt(i);

                num1 = num1 * 10 + Character.getNumericValue(c);
                i += 1;
            }

            while (j < other.version.length() && other.version.charAt(j) != '.') {
                char c = other.version.charAt(j);

                num2 = num2 * 10 + Character.getNumericValue(c);
                j += 1;
            }

            if (num1 != num2) {

                return num1 - num2;
            }

            i += 1;
            j += 1;
        }

        return 0;
    }
}
```

Solution: The idea of this question is to compare the versions of computers by overriding the `compareTo` method. We want to compare each digit until there is nothing to be compared so we create pointers (index) for both computers' versions and we iterate through them with the first while loop. The question is asking us to compare two components of the version. For example if we have 2 versions like 123.123 and 1.23123, we do not need to compare the digits coming after "." since $123 > 1$ already. So in order to capture the

digits between ".", we have another 2 while loops that are iterating through digits, capturing them, and comparing them at the end. In the case of `num1 == num2`, we want to compare the digits coming after ".". Though it does not exist, the closest example would be comparing numbers with multiple decimals.

3 Connect The Dots

(20 Points)

Justin is playing connect-the-dots! Justin is given a number n and a rectangular 2-D array `paper`, whose elements contain exactly 1 copy of the numbers 1, 2, ..., n, and all remaining cells filled with 0. For example, the following is an example of a valid `paper` with $n = 6$:

0	0	0	2
0	1	0	0
0	0	0	0
0	3	0	0
4	6	0	5

Justin will draw a straight line starting from the cell marked 1 to the cell marked 2, then to the cell marked 3, and so on until reaching the cell marked n. Write a function `lineLength`, which determines the total length of the lines drawn (adjacent cells are distance 1 apart).

In the above example, the lines from 1 to 2, 2 to 3, 3 to 4, 4 to 5, and 5 to 6 are $\sqrt{5}$, $\sqrt{13}$, $\sqrt{2}$, 3, and 2 units long, respectively. This is derived from the Pythagorean theorem – for example, 1 to 2 is 1 unit up, 2 units to the right for a total of $\sqrt{1^2 + 2^2} = \sqrt{5}$ units along the diagonal.

The total length of these lines is thus $\sqrt{5} + \sqrt{13} + \sqrt{2} + 3 + 2 \approx 12.256$ units. Therefore, if `lineLength` received this `paper` as input, it would return 12.256.

The `Point` class below represents a point in two-dimensional space with integer coordinates and is used in parts (a) and (b) below.

```
1   public class Point {
2       public int x;
3       public int y;
4
5       public Point(int x, int y) {
6           this.x = x;
7           this.y = y;
8       }
9   }
```

(a) Complete the `distance` method below so that it returns the distance between two `Point` objects. You do not need to use all the blanks.

```
public static double distance(Point a, Point b) {

    return Math.sqrt(Math.pow((b.x - a.x), 2) + Math.pow(b.y - a.y), 2);

    _____;

    _____;
}
```

Solution: The distance function calculates the Euclidean distance between two points (a and b) using the Pythagorean theorem: the square root of the sum of the squares of the differences in their x and y coordinates. This solution uses 1 line, but there are many 2 or 3 line solutions that are also correct.

(b) Implement the `lineLength` method so that it determines to total length of the lines drawn. The method implemented in part a may be helpful.

```
public static double lineLength(int[][] paper, int n) {
    Point[] points = new Point[n + 1];

    for (int i = 0; i < paper.length; i += 1) {

        for (int j = 0; j < paper[0].length; j += 1) {

            int z = paper[i][j];

            points[z] = new Point(i,j);
        }
    }

    double total = 0.0;

    for (int k = 2; k < points.length; k += 1) {

        total += distance(points[k - 1], points[k]);
    }
    return total;
}
```

Solution: First we create a `Point` array of $n+1$ size, where we intend to store the positions of points 1 to n in the indices 1 to n, respectively, while keeping index 0 as a placeholder for irrelevant information. Then, looking at line 8, we see that we are accessing the 2D array `paper` by index i then index j. Thus, we need index i to be from 0 to `paper.length`, and j to be from 0 to paper[0].length(`paper[i].length` would also work).

Then, we can assign `points[z]` to `new Point(i,j)`. Just this line is sufficient because of the problem statement: "all remaining cells filled with 0". The line efficiently captures the necessary points from Point 1 to n, which are the only ones needed for our computation. Since all of the other points hold a value of 0, it does not matter if the 0's positions are overriding each other, since their position is irrelevant to the final computation.

Then, we would need to sum up the distances. We will start the for loop at `k = 2` and terminate at n. In each iteration, we calculate the distances of Point k - 1 and Point k (so it starts from calculating Point 1 to Point 2 and ends at Point $n-1$ and n). At last, we return the total distance.

There were a few alternate solutions, particular in the manner that you index into `points`. They were awarded credit if the final answer returned was identical to the intended solution, for all possible values of `paper` and `n`.

4 Class Signatures

(13 Points)

Fill in the class signatures so that the code below compiles and runs without errors.

```
1  B bb = new B();
2  A aa = new A();
3  A ab = new B();
4  D db = new B();
5  A ac = new C();
6  B bc = new C();
7  A ad = new D();
8  E ed = new D();
9
10 public __(a)__ A {
11     // Code not shown
12 }
13
14 public __(b)__ B ____(f)____ _(g)_ {
15     // Code not shown
16 }
17
18 public __(c)__ C ____(h)____ _(i)_ {
19     // Code not shown
20 }
21
22 public __(d)__ D extends _(j)_ implements _(k)_ {
23     // Code not shown
24 }
25
26 public __(e)__ E {
27     public default void e() {
28         System.out.println("E!");
29     }
30 }
```

(a): ● class ○ interface

(b): ● class ○ interface

(c): ● class ○ interface

(d): ● class ○ interface

(e): ○ class ● interface

(f): ● extends ○ implements

(g): ○ A ○ B ○ C ● D ○ E

(h): ● extends ○ implements

(i): ○ A ● B ○ C ○ D ○ E

(j): ● A ○ B ○ C ○ D ○ E

(k): ○ A ○ B ○ C ○ D ● E

Solution:

Parts (a) - (e): The default keyword is used in interfaces to provide a default implementation that gets overridden by classes implementing the interface. Therefore, we know the only interface is E.

A, B, C, D have to be classes since we've instantiated objects belonging to these classes in the code given to us (instantiated objects cannot have a dynamic type of an interface).

Parts (f) - (k): Now, let's take a look at the code given to deduce the class relationships.

Object ab has static type A but dynamic type B, indicating that B is a sub-class of A.

Object db has static type D but dynamic type B, indicating that B is a sub-class of D.

Object ac has static type A but dynamic type C, indicating that C is a sub-class of A.

Object bc has static type B but dynamic type C, indicating that C is a sub-class of B.

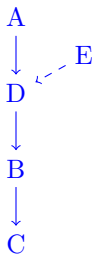
Object ad has static type A but dynamic type D, indicating that D is a sub-class of A.

Object ed has static type E but dynamic type D. Since we know E is an interface, that means D implements the interface E.

From these relationships, we know

1. A is a superclass of B, D, C
2. D is a superclass of B
3. B is a superclass of C

which means the class/interface lineage is:



So, we have

- `public class A`
- `public class B extends D`
- `public class C extends B`
- `public class D extends A implements E`
- `public interface E`

5 HOF

(16 Points)

Consider the following interface, which represents an arbitrary function. `apply` takes an input of type A and returns an output of type B.

```
1  interface Func<A, B> {
2      B apply(A x);   // ... all other methods omitted ...
3  }
```

Complete the `compose` method, which takes in two functions f and g and returns a new function that, when applied, returns $g(f(x))$, or the result of applying f and then applying g. Be sure to complete the classes `Composer` and `Helper` in order for `compose` to behave as expected.

```
class Composer<X, Y, Z> {

    Func<X, Z> compose(Func<X, Y> f, Func<Y, Z> g) {

        return new Helper(f, g);
    }

    private class Helper implements Func<X, Z> {
        Func<X, Y> f;
        Func<Y, Z> g;

        Helper(Func<X, Y> f, Func<Y, Z> g) {

            this.f = f;

            this.g = g;
        }

        @Override

        public Z apply(X x) {

            return g.apply(f.apply(x));
        }
    }
}
```

Solution: We know that we want compose to return a `Func` that applies f and then applies g. Looking at the parameters in the `compose` method, we can see that $f : x \to y$ and $g : y \to z$. Thus, $g(f(x))$ is described by $(g \circ f) : x \to z$. In the `compose` method, we will return a private `Helper` class, where all the real work is done.

In the private `Helper` class constructor, we assign our instance variables `this.f` and `this.g` to their respective parameters in the constructor.

For the method signature of `apply`, we know that $(g \circ f) : x \to z$, which means that we must take in a parameter of type `X`. The return type also must be of type `Z`. However, the parameter name can be any valid variable name, though most commonly provided solution uses the parameter name `x`.

Finally for the method body, we return the result of applying `f` first, then `g`, which evaluates $g(f(x))$. Note that you may NOT write `return f.apply(g.apply(x));` because this will instead evaluate $f(g(x))$ (and will also run into a compile-time error if `X`, `Y`, and `Z` are distinct).

Note: answers with access modifiers `private`, `protected`, and no access modifier (package private) for `public Z apply(X x)` were not deducted points.

6 Lost in Pointers

(16 Points)

Vanessa has implemented a circular `DLList` class, and has created a circular `DLList` (as seen on the reference sheet). However, Dylan the Hacker has gotten into her system and changed one of the `next` or `prev` pointers to point to another node in the `DLList` (or the sentinel). Write a function `fixDLList` that fixes the changed pointer in the instance of the DLList to be the original correct `DLList`.

You may assume that this function is defined in `DLList`, and therefore has access to the private variables of `DLList`.

```
public void fixDLList() {
    int count = 0;
    for (Node n = sentinel.next; count <= size && n != sentinel; n = n.next) {

        count += 1;
    }

    if (count == size) { // The wrong pointer is a .prev pointer
        sentinel.next.prev = sentinel;

        for (Node n = sentinel.next; n != sentinel; n = n.next) {

            n.next.prev = n;
        }
    } else { // The wrong pointer is a .next pointer
        sentinel.prev.next = sentinel;

        for (Node n = sentinel.prev; n != sentinel; n = n.prev) {

            n.prev.next = n;
        }
    }
}
```

Solution:

There are two parts to this question: first, determining whether it's a prev or a next pointer that's wrong, and second, fixing that wrong pointer.

The first for loop goes through the entire list in the next direction. It starts at the first item (sentinel.next) and iterates through, stopping at the sentinel. If all the next pointers are right, then it will conclude at the sentinel, where count is going to be equal to size.There are two ways the next pointers can be wrong: either a next pointer goes "backward" to a node earlier in the list, which creates an infinite loop (imagine a list of $1 \to 2 \to 3 \to 4$, where 3's next goes backward to 1), or a next pointer goes "forward" to a node which skips at least one node (imagine a list of $1 \to 2 \to 3 \to 4$, where 1's next goes forward to 3). The first case makes count too high; that is why the stop condition is count $\leq$ size and not count $<$ size, to account for when there's an infinite loop from a wrong next pointer. The second case makes the count too low by triggering the other stop condition too early of n going to the sentinel. In order for count to be correct (equal to size), every next pointer must be correct.

The second part of the function fixes the wrong pointer. If a prev pointer is wrong, then there is a loop through all next pointers (which must be right), assigning the next item's previous at every step along the way (overwriting what used to be there, including the wrong pointer). The same logic is used in the opposite direction if a next pointer is wrong (going through prev pointers to assign next pointers continually).

Nothing on this page is worth any points.

Bonus Question

(0 Points)

Write the name of a food dish whose ingredients were first domesticated on as many distinct continents as possible. For example, a pizza: the tomato was domesticated in South America, wheat in Asia, and olives/olive oil in Europe. So the "score" for this food is 3.

Thai Curry (6): Chili Pepper (North America), Potato (South America), Beef (Europe), Onion (Asia), Tamarind (Africa), Coconut (Oceania)

Feedback

(0 Points)

Leave any feedback, comments, concerns, or more drawings below!

yokover