Q1. (W) Set up SSH, git, and IDE and your account on Chime

Kyra Zhou

Q2. (W) Complete the Fibonacci task on Chime

Fibonacci Task part 1 on Chime

Fibonacci Task part 2 on Chime

(a) Why would one argue that the provided tests for Fibonacci are not sufficient?

- If the tests have not go through all the branches in the codes, then it is not sufficient to say the code is robust because without testing all the branches, one of them might have unexpected bugs remaining.

(b) Why would it be a bad idea to write a test that times whether FibonacciTable runs faster than Fibonacci?

- The run time will differentiate if the number of test cases are relatively big, otherwise the difference between run time is too tiny that we are not able to distinguish. Since it takes a lot of time to write a sufficient amount of tests, it will be a bad idea in practice.
- Even with the sufficient amount of test data, we have to iteratively do the same test on both programs to enable us to conclude because there might be some environmental factors involved in each test which might influence an individual test round. By multiple test rounds, we can finally give the correct conclusion.

Q3. What does instruction coverage mean as a test coverage metric? Give an example showing how 100% instruction coverage does not mean your program is bug free.

- Instruction coverage: The amount of instructions or branches of the program that have been gone through during testing. For a valid input, the program will execute properly if the percentage coverage is high.
- It might still have bugs if we have not set up a path to deal with invalid inputs. For instance, in the fibonacci program, if we receive a number that is smaller than 1, and we have no path to deal with it, we will receive an error such as null pointer exception.

Q4. (W) Compare and contrast a typical functional language and a typical imperative language.

- For instance, we pick `OCaml` and `Java`.
- Programmer Focus: In `OCaml`, we care about what to do but not how to achieve the goal; while in `Java`, we care about the both what and how to achieve the objectives.
- Expressions and Statements: In `OCaml`, we only have expressions which evaluate into values; while in `Java`, we have both statements and expressions, thus state changes are very important in imperative language.
- Primitive flow control: In `OCaml`, we implement recursion a lot and function calls are the primitive flow control; while in `Java`, we have loops, conditions and function calls.
- Primitive manipulation unit: In `OCaml`, we have functions as the first-class objects and data collections; In `Java`, we have instances of structures or classes;

Q5. Write Java code to test whether your Java environment performs tail-recursion optimisations or not.

- No, java does not perform tail-recursion optimisations.
- Because in the `Factorial` functions, the run time of the tail-recursion version is similar or even slower than the version without tail-recursion.

Question 5 code on Github

Q6. Write a static function lowestCommon that takes two long arguments and returns the position of the first set bit in common, where position 0 is the LSB. If there is no common bit, the function should return -1. For example lowestCommon(14,25) would be 3. Your solution should use at least one break statement.

Question 6 code on Github

Q7. Identify the primitives, references, classes and objects in the following Java code:

```
double d = 5.0;
int i[] = {1,2,3,4};
LinkedList<Double> l = new LinkedList<Double>();
Double k = new Double();
Tree t;
float f;
Computer c = null;
```

- Primitives: `d`, `f`
- References: `i`, `l`, `k`, `t`, `c`
- Classes: `LinkedList<Double>`, `Double`, `Tree`, `Computer`

- Objects: `new LinkedList<Double>()`, `new Double()` > A reference is an entity which provides a way to access object of its type. An object is an entity which provides a way to access the members of it's class or type

Q8. You met the idea of linked lists in FoCS. Complete the first part of the 'Classic collections' task on Chime.

Classic Collections Part 1

- (a) How is an empty list represented?
  - When we set both the head and the tail of the list as null.
- (b) How does the toString method implement the list traversal?
  - Both the LinkList Class and the Node Class overrides the default `toString()` method.
  - We first call the `toString()` method from the LinkedList Class because we have only access to a LinkedList object in the Main class.
  - The `toString()` method from the LinkedList Class checks whether the list is empty or not based on its value of the head element.
  - If the head element is not null, then we use the head element to access the `toString()` method from the Node Class.
  - If the next element is null, then we return a string representation of the current value.
  - If the next element is not null, then we add the current value as a part of the return value and recursively use next to call the `toString()` method.
  - Eventually, the value of the Nodes will be passed as a string back to the `toString()` method from the LinkedList Class and formatted for output.
- (c) Given that you are making use of the addFirst method why would it be considered better style to use a static factory method to construct a list rather than doing all the work in the constructor?
  - We need to overload the constructor to allow us to add a first element when there exists a head element.
  - If we already know the elements we want to add into the linked list, we have to manually add them one by one using the constructor; while the `create` static factory method frees our hands and let the for loop to do the tedious job for us.
- (d) Give the UML class diagram for your code.

```
// TODO
```

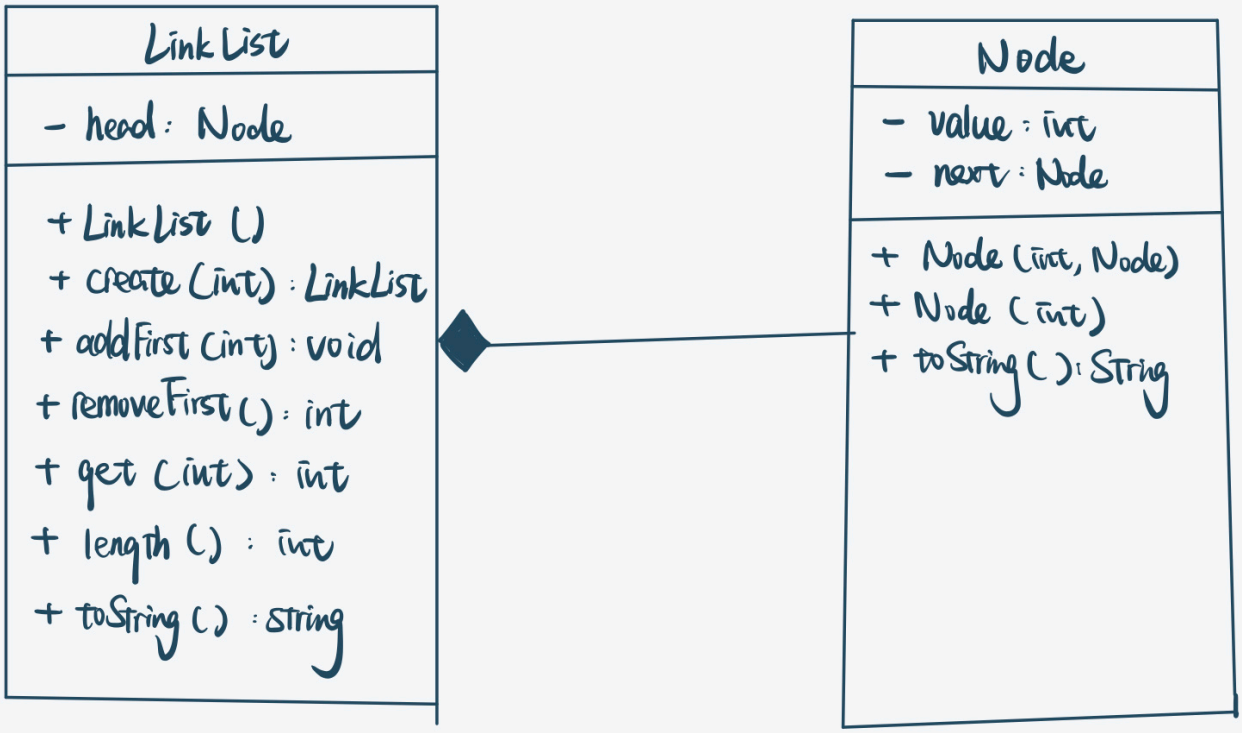
Q9. In mathematics, a set of integers refers to a collection of integers that contains no duplicates. You typically want to insert numbers into a set and query whether the set contains numbers. One approach is to store the numbers in a binary search tree.

(a) The diagram below represents this approach. Implement it in Java, using the names of entities to decide what they should do. Provide appropriate tests using JUnit.

(b) (b) The BinaryTreeNode class can be reused for other solutions. Create a class FunctionalArray that uses BinaryTreeNode to create a functional array of ints. Your class should have a constructor that creates a tree of a given size (passed as an argument); a void set(int index, int value) method; and a int get(int index) method. You should make the functional array zero-indexed to match java's normal arrays (i.e. the first element has index 0). Requests for indices outside the limits should result in an exception.

Question 9 code on Github

Q10. Explain why this code prints 0 rather than 7.


LinkList
- head : Node
+ LinkList ()
+ create (int) : LinkList
+ addFirst (int) : void
+ removeFirst () : int
+ get (int) : int
+ length () : int
+ toString () : String
Node
- value : int
- next : Node
+ Node (int, Node)
+ Node (int)
+ toString () : String

```
public class Test {
    public int x=0;
    public void Test() {
        x=7;
    }
    public static void main(String[] args) {
        Test t = new Test();
        System.out.println(t.x);
    }
}
```

- There is no customised constructor in the class, because the function `Test()` has a `void` return type, it is thus a public method.
- When we instantiate an object using the default constructor, the value of the variable x is set to 0.
- Hence, when we print it out, it gives a value of 0 as a result.

Q11. What are the benefits of using private state in conjunction with public getter and setter methods? Are there any disadvantages?

- Advantage:
  - Avoid illegal states of the objects because we can apply pre-evaluation in the setter method to force the user give valid arguments.
  - Some of the fields are implementation details and are not necessary to be seen, so setting them to private prevents them to be accessed outside the class.
- Disadvantage:
  - We have to apply setter and getter methods if we want to modify or change the value of the field. This requires extra logic and codes.

Q12. Python does not offer a private access modifier and everything is public. Instead, programmers should follow the convention that a variable name should be prefixed with an underscore ('_') to signal it should not be accessed or edited externally. Discuss the advantages and disadvantages of this approach compared to Java.

- Advantage:
  - Avoid the confusion between access modifiers, programmers do not need to write extra setter and getter method to access and change the variable values.
- Disadvantage:
  - Convention is not equivalent to syntax, so programmers who are not familiar with this convention might accidentally access and modify the variable values from outside of the class, this might lead to bugs.

Q13. Complete the Matrices programming task on Chime

Part 1 Code

Part 2 Code

- (a) Why is it necessary for the assertions to have the form assertThat(actual).isWithin(tolerance).of(expected);?
  - Because we are dealing with `double` which has a precision limit and when we manipulate the two matrices, for instance, one of the element in the matrix is 7.00000000000001 and the corresponding element in the other matrix is 1.0, when added together, this gives us 8.0 while the correct answer is 8.0000000000001.
- (b) Why have the static factory methods been put in the Shapes class rather than Matrix?
  - One of the principles of Object-oriented programming is encapsulation, which indicates that a unit class should bundle related data and the methods operated on the data together.
  - The Shape class contains different types of Matrix and the Matrix class contains different operations on the Matrix. If we put all the static methods into the Matrix class, then it will grow a lot in size and become hard to maintain.

Q14. This question considers representing a 2D vector in Java.

- (a) Develop a mutable class Vector2D to embody the notion of a 2D vector based on floats (do not use Generics). At a minimum your class should support addition of two vectors; scalar product; normalisation and magnitude.

Code for Part(a) and Part(c) on Github

- (b) What changes would be needed to make it immutable?
  - Set the fields of the matrix as final so that their values can only be assigned only once.
  - Make all the fields as private so that we do not allow direct access.
  - No setter methods provided for the fields.
  - Initializing all the fields using a constructor performing deep copy.
  - Performing cloning of objects in the getter methods to return a copy rather than returning the actual object reference
- (c) Contrast the following prototypes for the addition method for both the (i) mutable, and (ii) immutable versions.
  - public void add(Vector2D v)
  - public Vector2D add(Vector2D v)
  - public Vector2D add(Vector2D v1, Vector2D v2)
  - public static Vector2D add(Vector2D v1, Vector2D v2) (d) How can you convey to a user of your class that it is immutable?
  - Using unit testing to ensure that when constructing the vector, we have taken a deep copy of the value, and after initialization, the value of the vector is unable to be reassigned.

Q15. Explain in detail why Java's Generics do not support the use of primitive types as the parameterised type? Why can you not instantiate objects of the template type in generics (i.e. why is new T() forbidden?)

- Java's Generics only support reference types and not primitive types, because at compile time, primitive types has their own bytecode instructions for loading and storing onto the virtual machine stack.
- Although it is not impossible to implement, it will make the compiler complicated.
- Type erasure is applied at compile time and if we instantiate the objects as generics then its type will be erased at compile time.

Q16. (W) Complete the second part of the 'Classic collections' task (using generics) on Chime

Code on Chime

Reference on Generics and static inner class and methods

> You could have either made Node an instance inner class or a generic class with its own type parameter. What are the pros and cons of these options?

- Instance inner class is a non-static nested class and can access the generic type of the outer class; while a generic class with its own type parameter is a nested static class and cannot access the generic type of the outer class.
- Instance inner class:
  - Pros: Every instance of the container class will be tied closely to the instance of the inner class.
  - Cons: The instance of the inner class can interact with instance of the outer class, thus harder to maintain and debug.
- Generic class with its own type parameter:
  - Pros: Can only interact with the container class's static field and thus more efficient and easy to understand.
  - Cons: Cannot interact with the container class's non-static fields and have limited use.

Q17. Complete the Sorting task on Chime.

(a) When implementing Merge Sort you will need some temporary space for the merging part of the algorithm. You will find that you are unable to create new arrays of type T i.e. if you try T[] temp = new T[10] you will get a compile error. You can use a plain Object array or a new ArrayList⟨T⟩ instead. Bearing in mind the concept of type-erasure why does this fail to compile? Why does Object[] or ArrayList⟨T⟩ work? * Because during compile time, the compiler applies type erasure which gets rid of the type.

Code on Chime

Q18. Research the notion of wildcards in Java Generics. Using examples, explain the problem they solve.

- The definition: a collection of unknown which includes all kinds of collections.
- Purpose:
  - Cast a collection to a subclass of a class or a superclass of a class
  - Reading from a generic collection
  - Inserting into a generic collection

Reference of the wildcards in Java Generics

```
// If we have three classes, Both B and C inherits from A
public class A{}
public class B extends A{}
public class C extends A{}
// If the class A has the following method, we can only pass the List<A> type as its parameter
public void processElements(List<A> elements){
   for(A o : elements){
      System.out.println(o.getValue());
   }
}
// What if we want to call this method and pass List<B> and List<C> type as its parameter? We can use wildcards
public void processElements(List<?> elements){
   for(Object o : elements){
      System.out.println(o);
   }
}
```

Q19. Pointers are problematic because they might not point to anything useful. A null reference doesn't point to anything useful. So what is the advantage of using references over pointers?

- References do not support arithmetic operations so that prevent the references being misused (i.e., if someone maliciously add some value onto the pointer, it will points to a different address of the memory and may leads to bugs in the program)
- References are strongly typed so we cannot chang the type of the reference, immutability of the type of the reference prevent it being misused.
- References are implemented to ease garbage collector.

Q20. (W) Draw some simple diagrams to illustrate what happens with each step of the following Java code in memory:

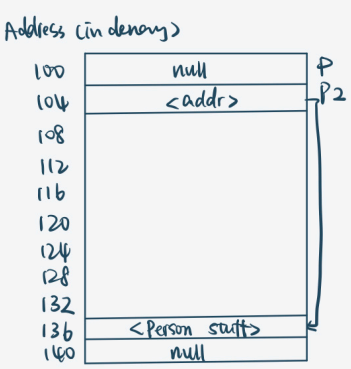
```
// TODO
Person p = null;
Person p2 = new Person();
p = p2;
p2 = new Person();
p = null;
```

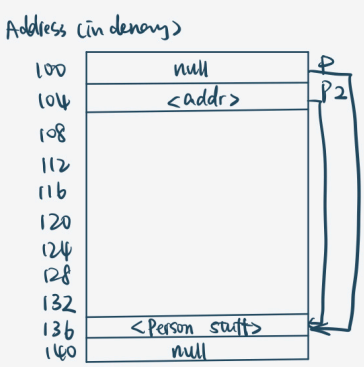
Q21. What would you say are the three most important concepts from the content for this supervision?

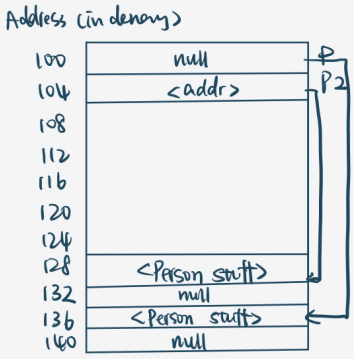
- The principles of Object-oriented programming:
  - Encapsulation
  - Abstraction
  - Inheritance (later in the course)
  - Polymorphism
- The difference between functional programming language and procedural programming language
- Immutability, difference between reference types and primitive types (i.e., memory storage format, relationship to the Generics etc.)

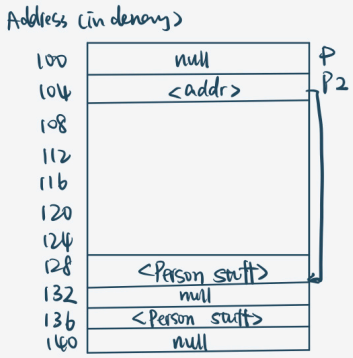
Q22. Give one question that you would like to discuss in the supervision.

- OOP widening and narrowing question:
  - Why the following operations give Compiler error:
    - bool -> int ?
    - char -> short ?


Address (in denary)
100
104
108
112
116
120
124
128
132
136
140
null
<addr>
<Person stuff>
null
P
P2


Address (in denary)
100
104
108
112
116
120
124
128
132
136
140
null
<addr>
<Person stuff>
null
P
P2


Address (in denary)
100
104
108
112
116
120
124
128
132
136
140
null
<addr>
<Person stuff>
null
<Person stuff>
null
P
P2


Address (in denary)
100
104
108
112
116
120
124
128
132
136
140
null
<addr>
<Person stuff>
null
<Person stuff>
null
P
P2