Different learners learn in their own way, but to me the best way to make your way through a module is, once again, to take the code and work your way through it and then try to add functionality. I’ve been trying to show you this with demo code with concepts associated with the module and expecting you to do the same with the module code, but this time I’m going to use the code provided by the module to do a demonstration. I’d like you to review it in groups and then we’ll come back together.

First warm up questions:

Before we organized our objects as a linked list. In this module we are organizing our structures in a hierarchy. What would be the advantages of doing it this way over a linked list?

If you’d prefer to download the code and run it from Eclipse, you can get a copy here:

<https://www.dropbox.com/s/b550ll6wpie4ddp/Mod6%20Code.zip?dl=0>

The more updated version can be found from this link:

https://onlinegdb.com/yVg1VWexz

Divide up into groups and review the code together. See if you can answer the below questions for each file. We’ll come back together after some time and discuss it together.

\_\_\_ Part 1 : Understanding what’s there \_\_\_

Go through each file and identify its purpose. Draw out the hierarchy as you go to help you understand what’s going on.

Organization.java

1. What is the purpose of this file?
2. Why start with this file for design?

Employee.java

1. What is the purpose of this file?
2. Why is it an interface?
3. Could it have been an abstract class? If so, does it make sense to do so? If not, why?

*When designing your application, why would it be a good idea to start with the above two files?*

Gender.java

1. Why is this in a separate file?
2. Does it have to be?

GenericEmployee.java

1. Why is this an abstract class instead of an interface?
2. If only contractEmployees have end dates, why do we need to include a getEndDate?
3. If interfaces enforce specific methods, why are we able to compile this class by itself without all those methods created yet?
4. Why isn’t add supervisee here?

Supervisor.java

1. What is the super constructor doing here?
2. How is the internal data stored?
3. There are two variables here with the same name, but they are serving different purposes and they are even different types. Fix this by naming one better.
4. The if statement and the for loop are particularly interesting. Explain to each other what is going on here.
5. Why use the linked list set function instead of just get?

NonManagerEmployee

1. What is the purpose of this file?

ContractEmployees.java

1. What's the purpose of this file?
2. We are making our way to OrganizationImpl. Why do we have to examine and compile each employee type first?
3. Compile this file.

OrganizationImpl.java

1. What is the purpose of this file?
2. Why does the only constructor take in an employee? Why does it have an employee root?
3. How does the allEmployees function work? Check out the stream breakout code if needed.

\_\_\_ Part 2: Add to the Driver \_\_\_

OrganizationDriver.java

1. Add a way to print a single employee. Change as few files as possible.

Ans: added to string methods for Supervisor and NonManagerEmployee classes

1. Add a way to print an entire organization with System.out.println

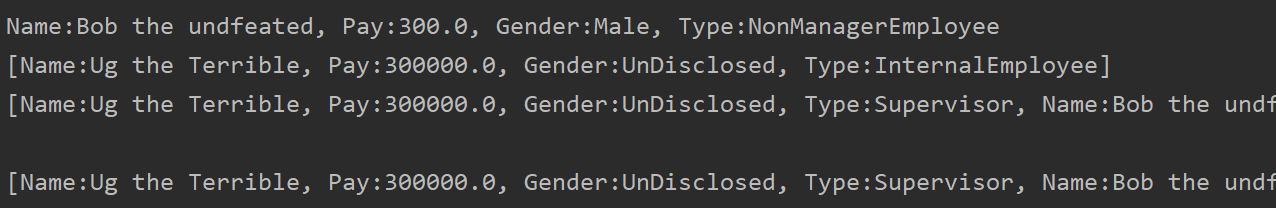
Ans: added a to string method under OrganizationImpl class to print entire organization.

1. Add the ability to add an employee from a passed in employee object.

Ans: added a addEmployee(Employee employee, String supervisorName) method under OrganizationImpl class

1. Create an entire organization and test.

Ans:

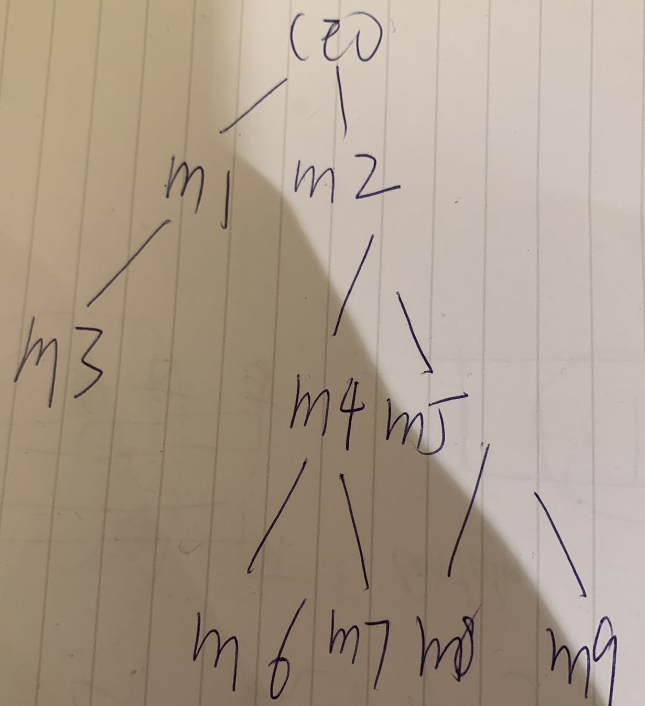
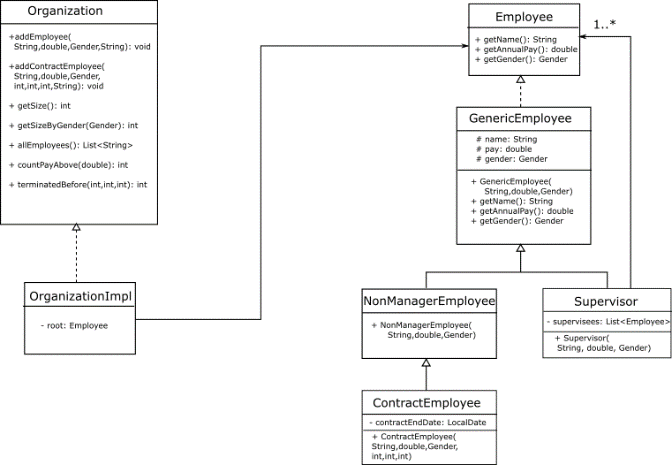


1. When printing an entire organization does the print order make sense?

Ans: No, It follows a dfs pre-order pattern, bfs order should be more appropriate.

Questions:

* Draw out the inheritance of this code.

* How is this a “hierarchical data structure?

Ans: it is made up of various levels of employees that are arranged in a specific order

* Why would removing an employee be more complicated than adding one?

Ans: An employee may have a supervisor and supervisees, if remove this employee, we need to put the supervisees and their supervisees into a right position. Adding an employee just need to add the employee in a supervisee list of a supervisor.

* How is this a recursive data structure?

Ans: a supervisor has supervisees, supervisees may have other supervisees…

* How is this design a composite design pattern?

Ans: employee class is composited in supervisor class, since it has a list of employees, which are supervisees.

* Why can’t we get an employee by number?

Ans: we only have pointer to the organization, so to get an employee in the given data structure we can only search from the top.

**Challenge TASK1: Make the print better for testing. For example, Make it print supervisor for a supervisor, contractor for a contractor, or nonManager for a nonManager. Make it print to show who supervises whom or just have the supervisor's name printed beside each employee.**

\_\_\_ Part 3 : Altering existing functionality \_\_\_

Counting

1. Do a code trace of the getSize method in OrganizationDriver.

Ans:

//Main.java

System.*out*.println(monsterCorp.getSize());

public int getSize() {  
 return root.count(b -> true);  
}

// GenericEmployee.java

public int count(Predicate<Employee> condition) {  
 if (condition.test(this)) {  
 return 1;  
 }  
 return 0;  
}

//supervisor.java

public int count(Predicate<Employee> condition) {  
 Stream<Employee> stream = this.supervisee.stream();  
 return this.supervisee.stream()  
 .mapToInt(b -> b.count(condition))  
 .sum()  
 + super.count(condition);  
}

1. What's the difference between the NonManagerEmployee and Supervisor count methods?

Ans: NonManagerEmployee has only 0 or 1 count since it has no supervisees.

Supervisor use a recursive call to count itself as well as its supervisees.

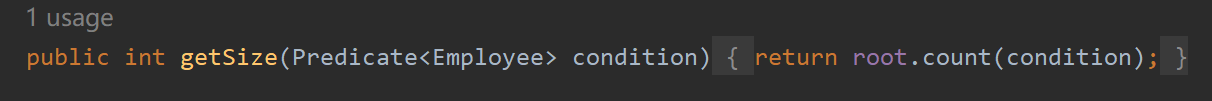
1. Implement an optional way to directly send in a predicate to getSize().

Ans:

public int getSize(Predicate<Employee> predicate) {  
 return root.count(predicate);  
}

1. Test by counting all employees with a salary of 300.00.

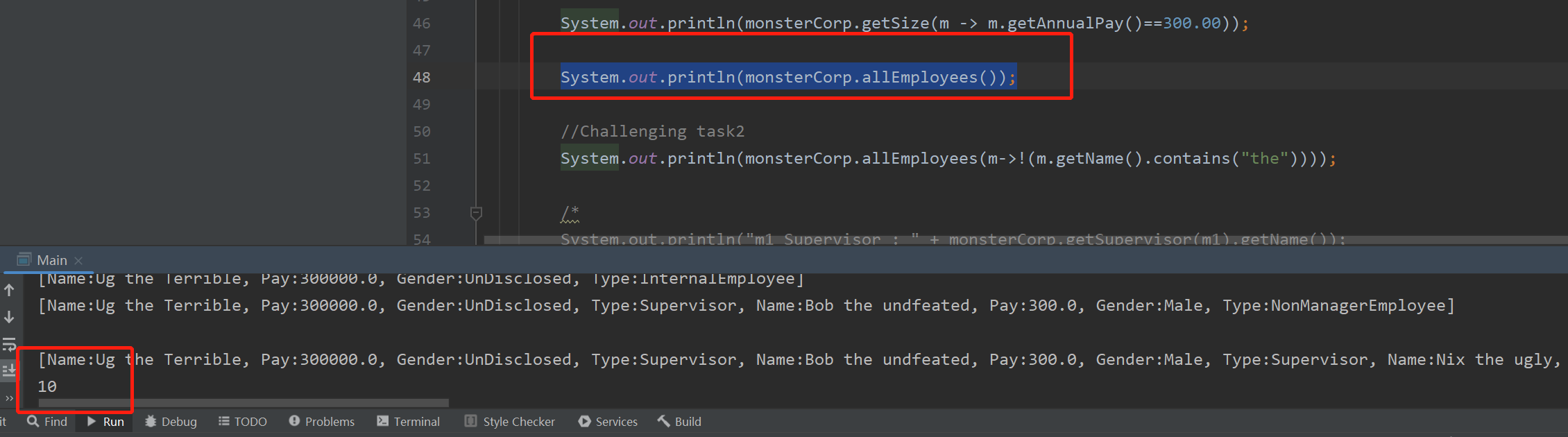
Ans: 5



Getting a list of names

1. Test out the method to get all employees by name.

Ans: 10



1. Do a code trace of this method.

Ans：

//main.java

System.out.println(monsterCorp.allEmployees());

//OrganizationImpl.java

public List<String> allEmployees() {  
 return root.toList().stream().map(e->e.getName()).collect(Collectors  
 .*toList*());  
}

//Employee.java

List<Employee> toList();

**Challenge TASK2: Alter the getSize to accept a predicate and get a list of employees meeting certain conditions by passing in a predicate. Tip: don't forget to declare the Predicate with the generic type <Employee>**

**Ans: Sue**

**You can test this with commented out line: "System.out.println(MonsterCorp.allEmployees(m->!(m.getName().contains("the"))));"**

\_\_ Part 4 : Add functionality \_\_\_

1. Add the ability to get the supervisor for a sent employee
2. ^ there’s something you’ll need to override first

**ChallengeTASK 3: Add the ability to remove an employee. Add better exception handling such as trying to remove an employee that doesn’t exist. Add better feedback and more printing options to better see what’s happening. Add the ability to sort internal employee lists.**