ASSIGNMENT 2 (30%)

Company Structure

PROJECT DESCRIPTION

For this assignment, you are going to practice using inheritance, interfaces and abstract classes to relate objects to one another. The following is a description of each class and its behavior. It is up to you to decide which classes should extend, implement or abstract which pieces to maximize your code sharing.

Create an object ecosystem that includes each of the following classes:

EMPLOYEE

Method header	Description
public Employee(String name, double baseSalary)	Should construct a new employee object and take in two parameters, one for the name of the user and one for their base salary
public double getBaseSalary()	Should return the employee's current salary
public String getName()	Should return the employee's current name
public int getEmployeeID()	Should return the employee's ID. The ID should be issued on behalf of the employee at the time they are constructed. The first ever employee should have an ID of "1", the second "2" and so on
public Employee getManager()	Should return a reference to the Employee object that represents this employee's manager
public boolean equals(Employee other)	Should return true if the two employee IDs are the same, false otherwise
public String toString()	Should return a String representation of the employee that is a combination of their id followed by their name. Example: "1 Kasey"
public String employeeStatus()	Should return a String representation of that Employee's current status. This will be different for every subclass of Employee

TECHNICAL EMPLOYEE

Method Header	Description
public TechnicalEmployee(String name)	Has a default base salary of 75000
public String employeeStatus()	Should return a String representation of this TechnicalEmployee that includes their ID, name and how many successful check ins they have had. Example: "1 Kasey has 10 successful check ins"

ASSIGNMENT 2 (30%)

BUSINESS EMPLOYEE

Method Header	Description
public BusinessEmployee(String name)	Has a default salary of 50000
public double getBonusBudget()	Should establish a running tally of the remaining bonusBudget for the team this employee supports. How that budget is determined will depend on which type of Business Employee it is
public String employeeStatus()	Should return a String representation of this BusinessEmployee that includes their ID, name and the size of their currently managed budget. Example: "1 Kasey with a budget of 22500.0"

SOFTWARE ENGINEER

Method Header	Description
public SoftwareEngineer(String name)	Should start without access to code and with 0 code check ins
public boolean getCodeAccess()	Should return whether or not this SoftwareEngineer has access to make changes to the code base
public void setCodeAccess(boolean access)	Should allow an external piece of code to update the SoftwareEngieer's code privileges to either true or false
public int getSuccessfulCheckIns()	Should return the current count of how many times this SoftwareEngineer has successfully checked in code
public boolean checkInCode()	Should check if this SoftwareEngineer's manager approves of their check in. If the check in is approved their successful checkin count should be increased and the method should return "true". If the manager does not approve the check in the SoftwareEngineer's code access should be changed to false and the method should return "false"

ACCOUNTANT

Method Header	Description
public Accountant(String name)	Should start with a bonus budget of 0 and no team they are officially supporting
public TechnicalLead getTeamSupported()	Should return a reference to the TechnicalLead that this Accountant is currently supporting. If they have not been assigned a TechnicalLead null should be returned
public void supportTeam(TechnicalLead lead)	Should allow a reference to a TechnicalLead to be passed in and saved. Once this happens the Accountant's bonus budget should be updated to be the total of each SoftwareEngineer's base salary that reports to that TechnicalLead plus 10%. For example, if the TechnicalLead supports 2 SoftwareEngineers, each with a salary of

ASSIGNMENT 2 (30%)

	75000, the Accountant's budget should be 150000 + 15000 for a total of 165000
public boolean approveBonus(double bonus)	Should take in a suggested bonus amount and check if there is still enough room in the budget. If the bonus is greater than the remaining budget, false should be returned, otherwise true. If the accountant is not supporting any team false should be returned.
public String employeeStatus()	Should return a String representation of this Accountant that includes their ID, name, the size of their currently managed budget and the name of the TechnicalLead they are currently supporting. Example: "1 Kasey with a budget of 22500.0 is supporting Satya Nadella"

TECHNICAL LEAD

Method Header	Description
public TechnicalLead(String name)	Should create a new TechnicalLead that is a Manager. The TechnicalLead's base salary should be 1.3 times that of a TechnicalEmployee. TechnicalLeads should have a default head count of 4.
public boolean hasHeadCount()	Should return true if the number of direct reports this manager has is less than their headcount.
public boolean addReport(SoftwareEngineer e)	Should accept the reference to a SoftwareEngineer object, and if the TechnicalLead has head count left should add this employee to their list of direct reports. If the employee is successfully added to the TechnicalLead's direct reports true should be returned, false should be returned otherwise
public boolean approveCheckIn(SoftwareEngineer e)	Should see if the employee passed in does report to this manager and if their code access is currently set to "true". If both those things are true, true is returned, otherwise false is returned
public boolean requestBonus(Employee e, double bonus)	Should check if the bonus amount requested would be approved by the BusinessLead supporting this TechnicalLead. If it is, that employee should get that bonus and true should be returned. False should be returned otherwise
public String getTeamStatus()	Should return a String that gives insight into this Manager and all their direct reports. It should return a string that is a combination of the TechnicalLead's employee status followed by each of their direct employee's status on subsequent lines. If the TechnicalLead has no reports it should print their employee status followed by the text " and no direct reports yet ". Example: "10 Kasey has 5 successful check ins and no direct reports yet". If the TechnicalLead does have reports it might look something like "10 Kasey has 5 successful check ins and is managing: /n 5 Niky has 2 successful check ins"

ASSIGNMENT 2 (30%)

BUSINESS LEAD

Method Header	Description
public BusinessLead(String name)	Should create a new BusinessLead that is a Manager. The BusinessLead's base salary should be twice that of an Accountant. They should start with a head count of 10.
public boolean hasHeadCount()	Should return true if the number of direct reports this manager has is less than their headcount.
public boolean addReport(Accountant e, TechnicalLead supportTeam)	Should accept the reference to an Accountant object, and if the BusinessLead has head count left should add this employee to their list of direct reports. If the employee is successfully added to the BusinessLead's direct reports true should be returned, false should be returned otherwise. Each time a report is added the BusinessLead's bonus budget should be increased by 1.1 times that new employee's base salary. That employee's team they are supporting should be updated to reflect the reference to the TechnicalLead given. If the employee is successfully added true should be returned, false otherwise.
public boolean requestBonus(Employee e, double bonus)	Should check if the bonus amount requested would fit in current BusinessLead's budget. If it is, that employee should get that bonus, the BusinessLeader's budget should be deducted and true should be returned. False should be returned otherwise
public boolean approveBonus(Employee e, double bonus)	This function should look through the Accountants the BusinessLead manages, and if any of them are supporting a the TechnicalLead that is the manager of the Employee passed in then the Accountant's budget should be consulted to see if the bonus could be afforded. If the team can afford the bonus it should be rewarded and true returned, false otherwise

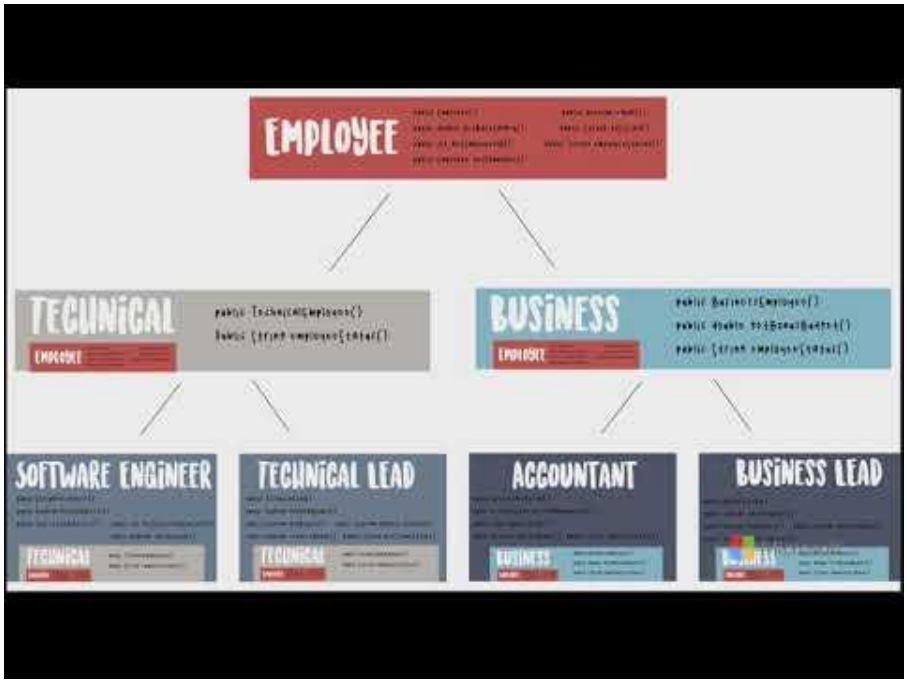
Here is my testing code that you can use to see if things are set up properly:

```
public class CompanyStructure {

public static void main(String[] args) {

TechnicalLead CTO = new TechnicalLead("Satya Nadella");

SoftwareEngineer seA = new SoftwareEngineer("Kasey");

SoftwareEngineer seB = new SoftwareEngineer("Breana");

SoftwareEngineer seC = new SoftwareEngineer("Eric");

CTO.addReport(seA);

CTO.addReport(seB);
```

```
CTO.addReport(seC);

System.out.println(CTO.getTeamStatus());

TechnicalLead VPofENG = new TechnicalLead("Bill Gates");

SoftwareEngineer seD = new SoftwareEngineer("Winter");

SoftwareEngineer seE = new SoftwareEngineer("Libby");

SoftwareEngineer seF = new SoftwareEngineer("Gizan");

SoftwareEngineer seG = new SoftwareEngineer("Zaynah");

VPofENG.addReport(seD);

VPofENG.addReport(seE);

VPofENG.addReport(seF);

VPofENG.addReport(seG);

System.out.println(VPofENG.getTeamStatus());

BusinessLead CFO = new BusinessLead("Amy Hood");

Accountant actA = new Accountant("Niky");

Accountant actB = new Accountant("Andrew");

CFO.addReport(actA, CTO);

CFO.addReport(actB, VPofENG);

System.out.println(CFO.getTeamStatus());

}

}
```

ASSIGNMENT 2 (30%)

PROJECT HELP



INSTRUCTIONS

- Please make sure that the code is fully documented
- Apply all professional coding principles you have learnt so far
- When project is complete, you need to publish it on ***github***

RUBRIC

Your submission will be graded based on the following criterias:

	Unsatisfactory (25%)	**Fair (50%)**	**Good (75%)**	**Excellent (100%)**
Delivery **5 Points**	• Completed less than 50% of the requirements. • Does not comply with requirements (does something other than requirements).	• Completed between 50-64% of the requirements. • Delivered on time, and in correct format.	• Completed between 65-89% of the requirements.	• Completed at least 90% of requirements

ASSIGNMENT 2 (30%)

Coding Design Standards **5 Points**	• There is no software design at all • No programmer name included • Poor use of white space (indentation, blank lines) making code hard to read. • Disorganized and messy • Ambiguous identifiers.	• There is a short description of the software project • Includes name, and assignment title. • White space makes program fairly easy to read. • Organized work. • Good use of variables.	• There is a comprehensive description of the software project • Good use of white space. • Organized work. • Good use of variables and constants • Minimum line-wrap	• There is a comprehensive description of the software project with UML diagrams • Excellent use of white space. • Creatively organized work. • Excellent use of variables and constants. • No magic numbers. • Correct identifiers for constants. • No line-wrap
Documentation **10 Points**	• No documentation included.	• Basic documentation has been completed including a summary of requirements. • Purpose is noted for each function.	• Purpose is noted for each function and control structure. • One sample run included	• Specific purpose is noted for each function, control structure, input requirements, and output results.
Development and Runtime **10 Points**	• The code doesn't do what it is supposed to do • Does not execute due to syntax errors. • Does not execute due to runtime errors (endless loop, crashes etc.) • User prompts are misleading or non-existent. • No testing has been completed.	• The code does what it is supposed to do, but with unnecessary instructions and potential improvements to the logic • Executes without errors. • User prompts contain little information, poor design. • Some testing has been completed.	• The code does what it is supposed to do • Executes without errors. • User prompts are understandable, minimum use of symbols or spacing in output. • Thorough testing has been completed	• The code does what it is supposed to do and also it is optimized to run quickly • Executes without errors excellent user prompts, good use of symbols, spacing in output. • Thorough and organized testing has been completed and output from test cases is included.

End of Lesson