Logo, company name

Description automatically generated

**UNITAR GRADUATE SCHOOL**

**Course: ITWM5113 Software Design and Development**

**Course Instructor: Simon Lau**

**Assignment Submission**

**(Group/Individual) underline**

Assignment Title: Assignment 2

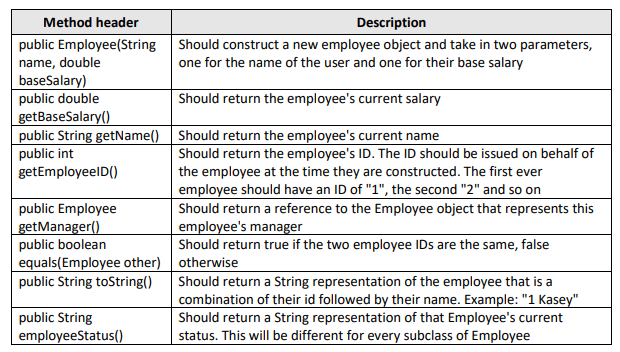
| **Name** | **Student ID** | **Section** |
| --- | --- | --- |
| NAGESWARY A/P NAGAPPAN | MC220517421 |  |
| PARTHIBAN A/L MARIMUTHU | MC220517304 |  |
| VITHYLINGAM A/L LETCHUMANAN | MC220517420 |  |

# PROJECT DESCRIPTION

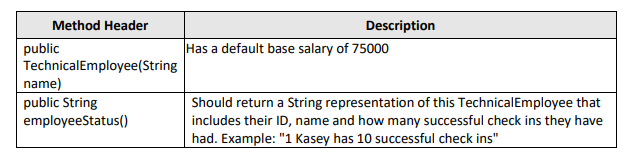
JAVA program depicting the company structure. Usage of inheritance, interfaces and abstract classes to relate objects to one another has been utilised.

It’s an object ecosystem that includes each of the following classes:

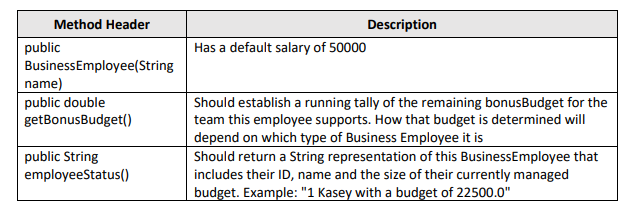
1. Employee.java



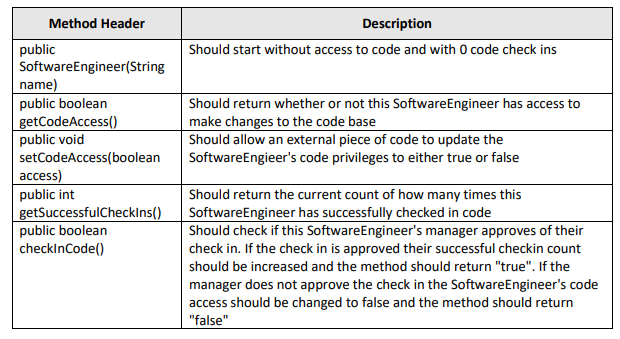
1. TechnicalEmployee.java



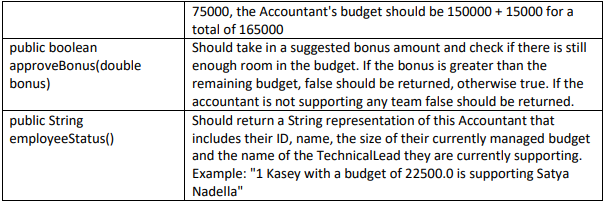
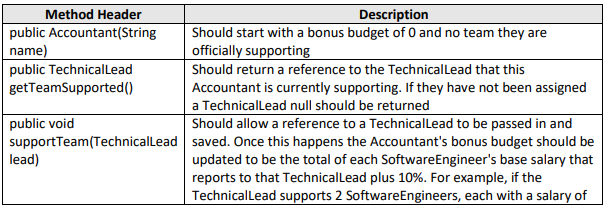
1. BusinessEmployee.java



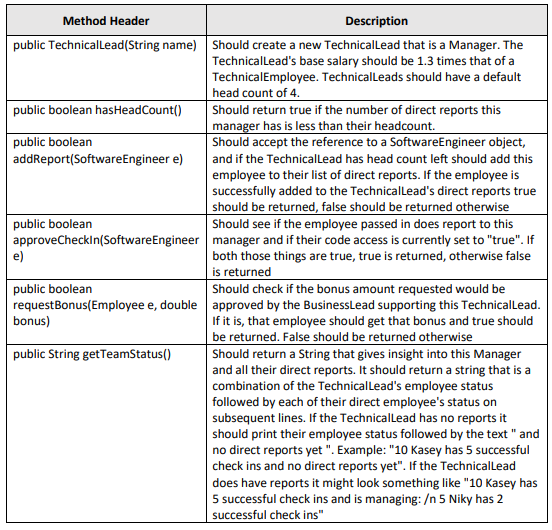
1. SoftwareEngineer.java



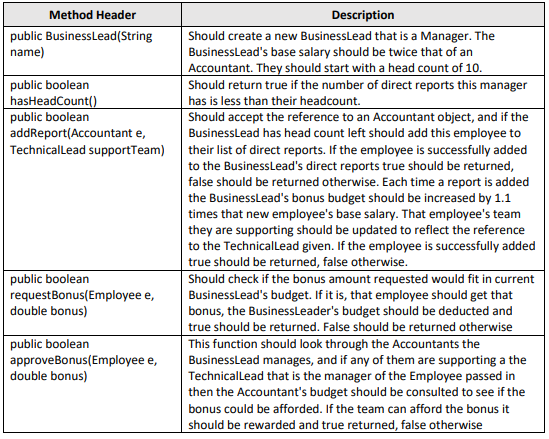
1. Accountant.java



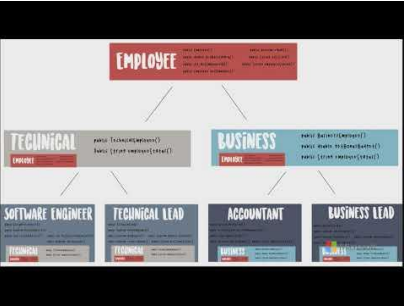
1. TechnicalLead.java



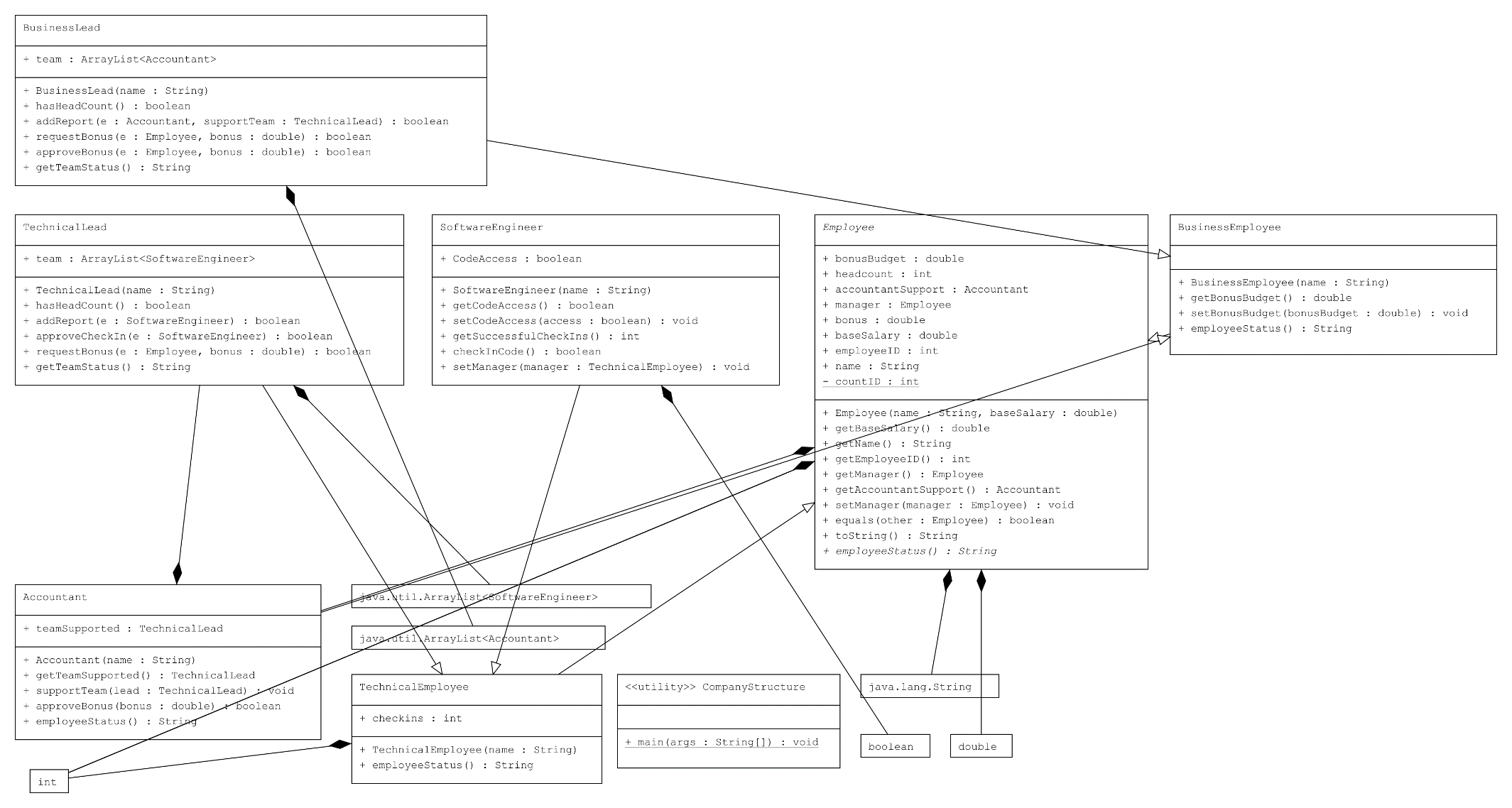
1. BusinessLead.java



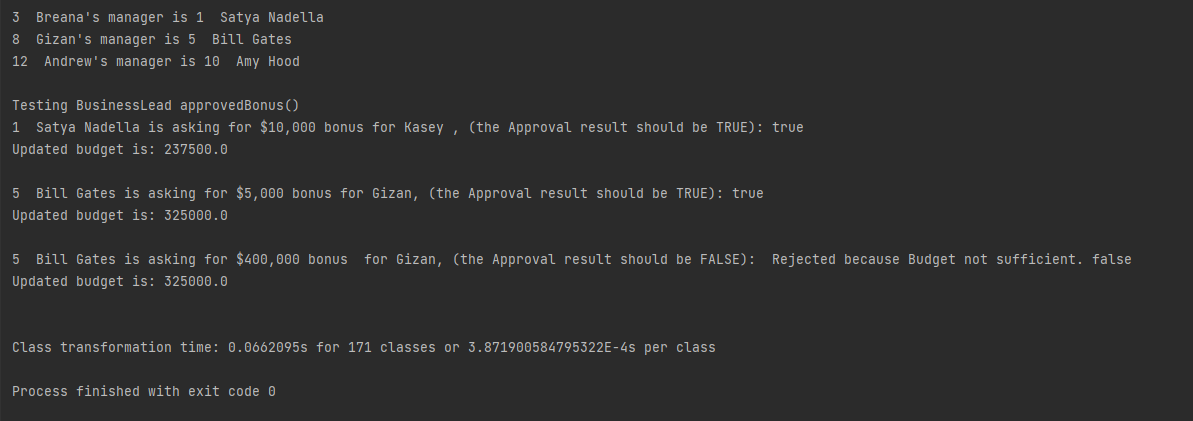
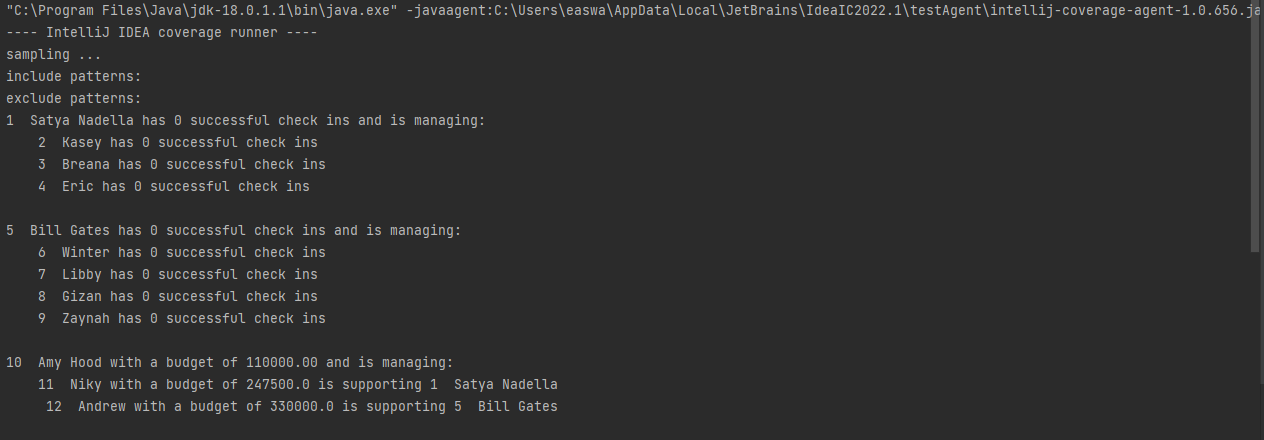
# PROJECT STRUCTURE



# PROJECT CLASS DIAGRAM



# PROJECT TEST RESULT



# PROJECT CODE

|  |  |
| --- | --- |
| **File Name** | **Code** |
| Accountant.java | //Inheriting the properties of the parent class BusinessEmployee - Employee via the extends public class Accountant extends BusinessEmployee {  public TechnicalLead teamSupported;  //to call teamSupported from TechnicalLead; if Accountants have TechnicalLead   public Accountant(String name)  {  //Should start with a bonus budget of 0 and no team they are  //officially supporting  super(name); //Inheritance from parent class via superclass constructor to get the accountant name  bonusBudget=0; //starts with budget 0  }   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   //if Accountant have TL or not; will be returned accordingly  return teamSupported;  }   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  //75000, the Accountant's budget should be 150000 + 15000 for a  //total of 165000   this.teamSupported=lead;  for (int i=0; i<lead.team.size(); i++) //based on the team size, the budget is calculated  {  this.bonusBudget+=lead.team.get(i).getBaseSalary()\*1.1;  }   }   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.   double requestedBonus=bonus;  if (requestedBonus<=getBonusBudget())  {  return true;  } else  {  System.*out*.print(" Rejected because Budget not sufficient. ");  return false;  }  }   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"   return this.toString()+" with a budget of "+ getBonusBudget()+" is supporting "+ this.getTeamSupported();  }  } |
| BusinessEmployee.java | public class BusinessEmployee extends Employee // BusinessEmployee inherits Employee parent class  {   public BusinessEmployee(String name)  {  //Has a default salary of 50000  super(name,50000.00);  }   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  return bonusBudget;  }   public void setBonusBudget(double bonusBudget)  {  this.bonusBudget = bonusBudget;  }   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"  String s= String.*format*("%.2f",bonusBudget);//reduce the double to 2 decimals  return this.toString()+" with a budget of "+ s;  } } |
| BusinessLead.java | import java.util.ArrayList; // to contain the Array collection for headcount  public class BusinessLead extends BusinessEmployee{  public ArrayList<Accountant> team; // Used to call array collection from accountant headcount   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.  super(name);  this.baseSalary=getBaseSalary()\*2;  this.headcount=10;  this.team=new ArrayList<Accountant>();   }   public boolean hasHeadCount()  {  //Should return true if the number of direct reports this manager  //has is less than their headcount.  if(this.team.size()<this.headcount) //headcount = 10  {  return true;  } else  {  return false;  }  }   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.  if (hasHeadCount())  {  team.add(e);  e.setManager(this);  this.bonusBudget+=e.baseSalary\*1.1; //Each time a report is added  //the BusinessLead's bonus budget increased by 1.1  e.supportTeam(supportTeam);  supportTeam.accountantSupport=e; // That employee's team they are supporting should be  // updated to reflect the reference to the TechnicalLead given.   return true; //If the employee is successfully added  //true should be returned, false otherwise.  } else  {  return false;  }  }   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  if (bonus<=getBonusBudget())  {  this.bonusBudget-=bonus;  e.bonusBudget+=bonus;  return true;   } else  {  return false;  }  }   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   for (int n=0;n<team.size();n++)  {  if((team.get(n).getTeamSupported()).equals(e.manager) && team.get(n).approveBonus(bonus))  {  e.bonus += bonus;  team.get(n).bonusBudget -= bonus;  return true;  }  }  return false;  }    public String getTeamStatus()  {  //to verify the status of the team during the test call  if (team.size()==0) //checking if the employee has direct report?  {  return this.employeeStatus()+ " and no direct reports yet";  } else  {  String teamStat ="";   for (int i=0;i<team.size();i++)  {  teamStat +=(" "+ team.get(i).employeeStatus()+"\n ");  }   return this.employeeStatus()+" and is managing: \n"+teamStat;   }  } } |
| CompanyStruture.java | public class CompanyStructure {  public static void main(String[] args)  {  //New Technical Lead & Software Enginerrs  TechnicalLead CTO = new TechnicalLead("Satya Nadella");  SoftwareEngineer seA = new SoftwareEngineer("Kasey");  SoftwareEngineer seB = new SoftwareEngineer("Breana");  SoftwareEngineer seC = new SoftwareEngineer("Eric");   //Assign SoftwareEngineers under Technical Lead Satya Nadella  CTO.addReport(seA);  CTO.addReport(seB);  CTO.addReport(seC);   System.*out*.println(CTO.getTeamStatus());   //New Technical Lead & Software Enginerrs  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");   //Assign SoftwareEngineers under Technical Lead Bill Gates  VPofENG.addReport(seD);  VPofENG.addReport(seE);  VPofENG.addReport(seF);  VPofENG.addReport(seG);   System.*out*.println(VPofENG.getTeamStatus());   //New Business Lead and accountants  BusinessLead CFO = new BusinessLead("Amy Hood");  Accountant actA = new Accountant("Niky");  Accountant actB = new Accountant("Andrew");   //add business lead reporting line  CFO.addReport(actA, CTO); // Niky is the accountant supporting Technical Lead Satya Nadella  CFO.addReport(actB, VPofENG); // Andrew is the accountant supporting Technical Lead Bill Gates   System.*out*.println(CFO.getTeamStatus());        //List out all the reporting line and managers  System.*out*.println(seB + "'s manager is " + seB.getManager().toString());  System.*out*.println(seF.toString() + "'s manager is " + seF.getManager().toString());  System.*out*.println(actB.toString() + "'s manager is " + actB.getManager().toString());  System.*out*.println();   //Testing criteria for bonus approvals  System.*out*.println("Testing BusinessLead approvedBonus()");  System.*out*.print(seA.getManager() + " is asking for $10,000 bonus for "+seA.getName()+" , (the Approval result should be TRUE): ");  System.*out*.println(CTO.requestBonus(seA, 10000));  System.*out*.println("Updated budget is: "+seA.getManager().getAccountantSupport().getBonusBudget()+"\n");   System.*out*.print(seF.getManager() + " is asking for $5,000 bonus for "+seF.getName() +", (the Approval result should be TRUE): ");  System.*out*.println(VPofENG.requestBonus(seF, 5000));  System.*out*.println("Updated budget is: "+seF.getManager().getAccountantSupport().getBonusBudget()+"\n");   System.*out*.print(seF.getManager() + " is asking for $400,000 bonus for "+seF.getName()+", (the Approval result should be FALSE): ");  System.*out*.println(VPofENG.requestBonus(seF, 400000));  System.*out*.println("Updated budget is: "+seF.getManager().getAccountantSupport().getBonusBudget()+"\n");  System.*out*.println();  } } |
| Employee.java | public abstract class Employee {  private static int *countID*; //static variable to accumulate the number of employees  public String name; //to store the name  public int employeeID; //to store the employee ID  public double baseSalary; // to store the base Salary  public double bonus; //to store the bonus value  public Employee manager;  public Accountant accountantSupport;  public int headcount=0;  public double bonusBudget;   //construct a new employee object and take in two parameters,one for the name of the user and one for their base salary.  public Employee (String name, double baseSalary)  {  this.name=name;  this.baseSalary=baseSalary;  *countID*++;  this.employeeID=*countID*; //assign the id to employee  // The first employee have an ID of "1", the second "2" and so on. This counter will increase and assign accordingly  }   public double getBaseSalary() //Returns the employee's current salary  {  return this.baseSalary;  }   public String getName()//Returns the employee's current name  {  return this.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  return this.employeeID;  }   public Employee getManager()  {  //Should return a reference to the Employee object that represents this employee's manager  return manager;  }   public Accountant getAccountantSupport()  {  return accountantSupport;  }   public void setManager(Employee manager)  {  this.manager=manager;  }   public boolean equals(Employee other)  {  //Should return true if the two employee IDs are the same, false otherwise  return this.getEmployeeID()==other.getEmployeeID();  }   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"  return getEmployeeID()+ " " + getName();  }   //declaring as abstract method without body.  //body will be provided by the inherited subclass under separate java files  public abstract String employeeStatus();  } |
| SoftwareEngineer.java | public class SoftwareEngineer extends TechnicalEmployee //SE inherits TE class {   public boolean CodeAccess; //to verify if SE has access or not to code   public SoftwareEngineer(String name)  {  //Should start without access to code and with 0 code check ins  super(name);  setCodeAccess(true);   }   public boolean getCodeAccess()  {  //Should return whether or not this SoftwareEngineer has access to  //make changes to the code base   return CodeAccess;  }   public void setCodeAccess(boolean access)  {  //Should allow an external piece of code to update the  //SoftwareEngieer's code privileges to either true or false   this.CodeAccess=access;   }   public int getSuccessfulCheckIns()  {  //Should return the current count of how many times this  //SoftwareEngineer has successfully checked in code  this.checkInCode();  return checkins;  }   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"   TechnicalLead manager=(TechnicalLead) this.getManager();  if (manager.approveCheckIn(this))  {  CodeAccess = true;  this.checkins++;  return true;   } else  {  CodeAccess=false;  return false;  }  }   public void setManager(TechnicalEmployee manager){  super.manager=manager;  } } |
| TechnicalEmployee.java | public class TechnicalEmployee extends Employee //TE inherits parent class Employee {  public int checkins =0; // to declare the data type for check-ins count   public TechnicalEmployee(String name)  {  //Has a default base salary of 75000  super(name,75000.00);  }   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"  return super.toString()+" has "+ checkins +" successful check ins";  }    } |
| TechnicalLead.java | import java.util.ArrayList; //to contain the array collection for SE public class TechnicalLead extends TechnicalEmployee //TL inherits TE class {  public ArrayList<SoftwareEngineer> team; //array to hold the SE list   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.   super(name);  this.baseSalary\*=1.3;  headcount=4;  this.team=new ArrayList<SoftwareEngineer>();    }   public boolean hasHeadCount()  {  //Should return true if the number of direct reports this  //manager has is less than their headcount.   if(team.size()<headcount)  {  return true;  } else  {  return false;  }  }   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   if(hasHeadCount())  {  team.add(e); // and if the TechnicalLead has head count left should add this employee to their list of direct reports.  e.setManager(this);//set the Technical Lead as manager of the Software Engineer.  return true;// If the employee is successfully added to the TechnicalLead's direct reports true should be returned,  } else  {  return false; // 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   if(e.getManager().equals(this) && e.getCodeAccess())  {//Should see if the employee passed in does report to this manager and if their code access is currently  // set to "true".  return true;// If both those things are true, true is returned, otherwise false is returned  } else  {  return false;  }   }   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   BusinessLead businessLead= (BusinessLead)getAccountantSupport().getManager();  if (businessLead.approveBonus(e, bonus))  {  return true;  } else  {  return false;  }  }   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    if (team.size()==0)  {  return this.employeeStatus()+ " and no direct reports yet";  } else  {  String teamStatus="";  for (int i=0;i<team.size();i++)  {  teamStatus+=(" "+team.get(i).employeeStatus()+"\n");  }  return this.employeeStatus()+" and is managing: \n"+teamStatus;  }  }  } |

**THANK YOU**