OOP programming is not like procedural programming. The understanding of the four key principles will make your life easier.

Encapsulation - Hiding of data implementation by restricting access to public methods. Instance variables are kept private and member functions (Getter & Setter) are made public to achieve this.

Example from Employee Class

```
private int employeeId;
private String name;
```

Implemented only getters not setters. Thus we prevent the change to Id and Name from outside

```
public int getEmployeeId() { return employeeId; }

public String getName() { return name; }
```

```
private double baseSalary;
private Employee manager;
```

Need to be changed from outside. Therefore, we implemented both getter and setter.

```
public double getBaseSalary() { return baseSalary; }

public void setBaseSalary(double baseSalary) { this.baseSalary = baseSalary; }

public Employee getManager() { return manager; }

public void setManager(Employee manager) { this.manager = manager; }
```

Abstraction – Hiding the complexity. The concepts or an Idea may not be associated with any object. Meaning to say the intent of class rather than the actual implementation.

Inheritance – Sub/ Derived classes reuse the code from Super/ parent class. It expresses is-a or has-a relationship

Polymorphism – One name in many forms. Static and Dynamic polymorphism can be achieved using method overloading and method overriding respectively

Example, we need to know the employee's status. In a company, the sales executives performance measured by their target and programmer's performance measured by how many bugs they have fixed etc. In our case we have two group of employees. Technical – Measured using their number of checkins. Business – Measured using their budget. It varies from instance to instance. We can't implement the employeeStatus method in Employee class. Therefor, we make it as abstract method

```java
abstract public String employeeStatus();
```

We implemented it in BusinessEmployee and TechnicalEmployee

TechnicalEmployee

```java
@Override
public String employeeStatus() {
    // %d - integer, %s - String, %f - float/ decimal, %2f - two decimal places
    String result = String.format("%d %s has %d successful check ins",
                                    this.getEmployeeId(), this.getName(), this.checkIns);

    return result;
}
```

BusinessEmployee

```java
@Override
public String employeeStatus() {
    // %d - integer, %s - String, %f - float/ decimal, %.2f - two decimal places
    String result = String.format("%d %s with a budget of %.1f",
                                    this.getEmployeeId(), this.getName(), this.bonusBudget);

    return result;

}
```

Employee is a BusinessEmplyee or TechnicalEmployee. It establishes is-a relationship using inheritance principles and you can access (reuses) the instance variables and members functions in Employee class

Thus, you are observing dynamic polymorphism and Inheritance here.

Take look at `private Employee manager;`

How do we associate?

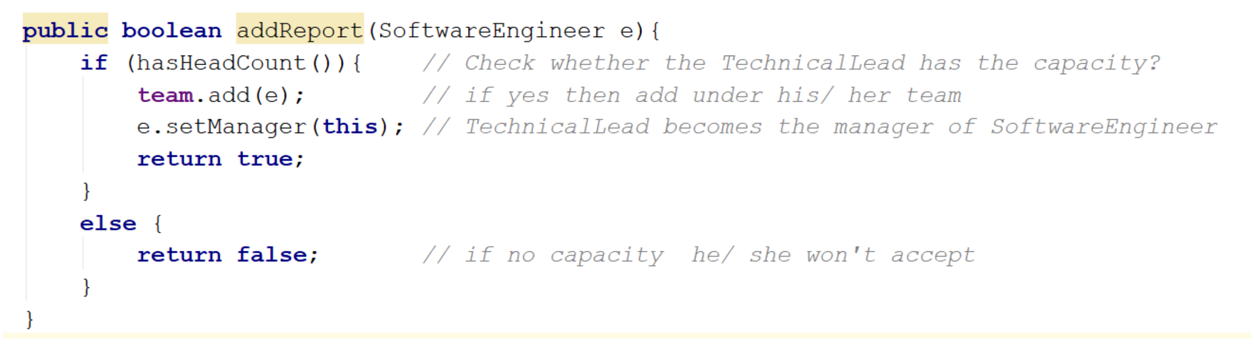
In realty, when do you come to know who is your manager? After joining the company, when your HR introduces you to your manager, correct?

Joining – Instantiation – Creating an Object - CompanyStructure

```
SoftwareEngineer charlie = new SoftwareEngineer( name: "Charlie");
```

Intro to your manager

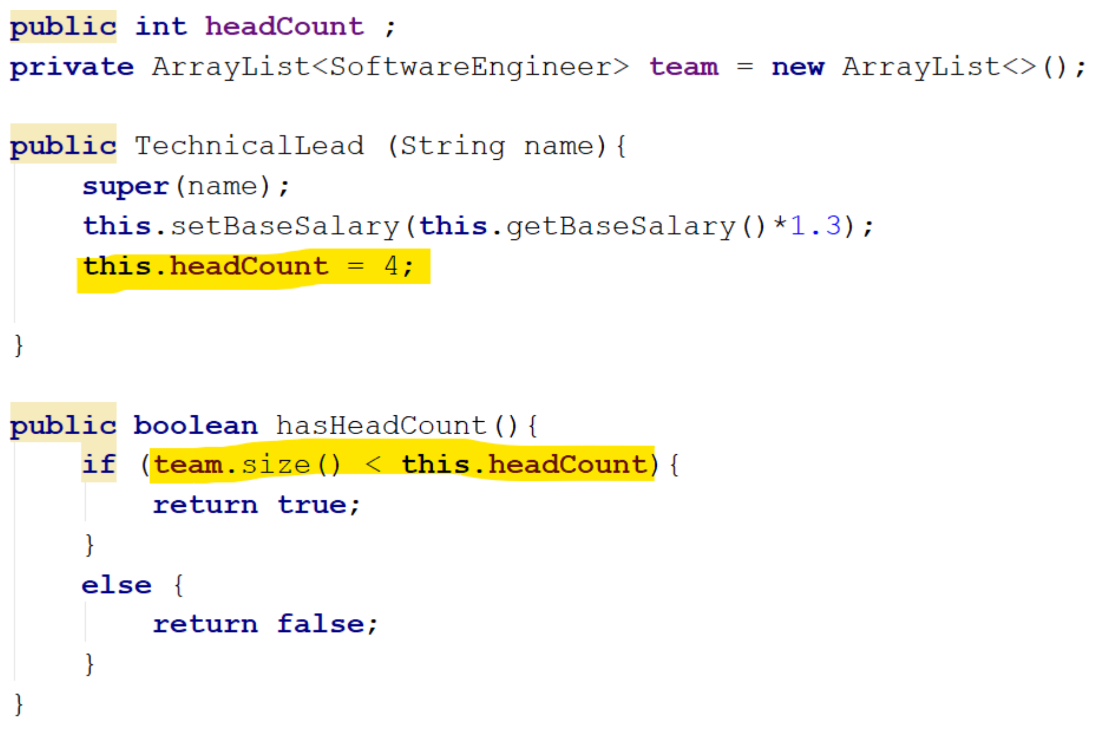
public boolean addReport(SoftwareEngineer e)	Should accept the reference to a SoftwareEngineer object, and if the TechnicalLead has head count left should add this employee to their list of direct reports. If the employee is successfully added to the TechnicalLead's direct reports true should be returned, false should be returned otherwise



```
public boolean addReport(SoftwareEngineer e){
    if (hasHeadCount()){     // Check whether the TechnicalLead has the capacity?
        team.add(e);         // if yes then add under his/ her team
        e.setManager(this);  // TechnicalLead becomes the manager of SoftwareEngineer
        return true;
    }
    else {
        return false;        // if no capacity  he/ she won't accept
    }
}
```

How do we know TechnicalLead’s head count?

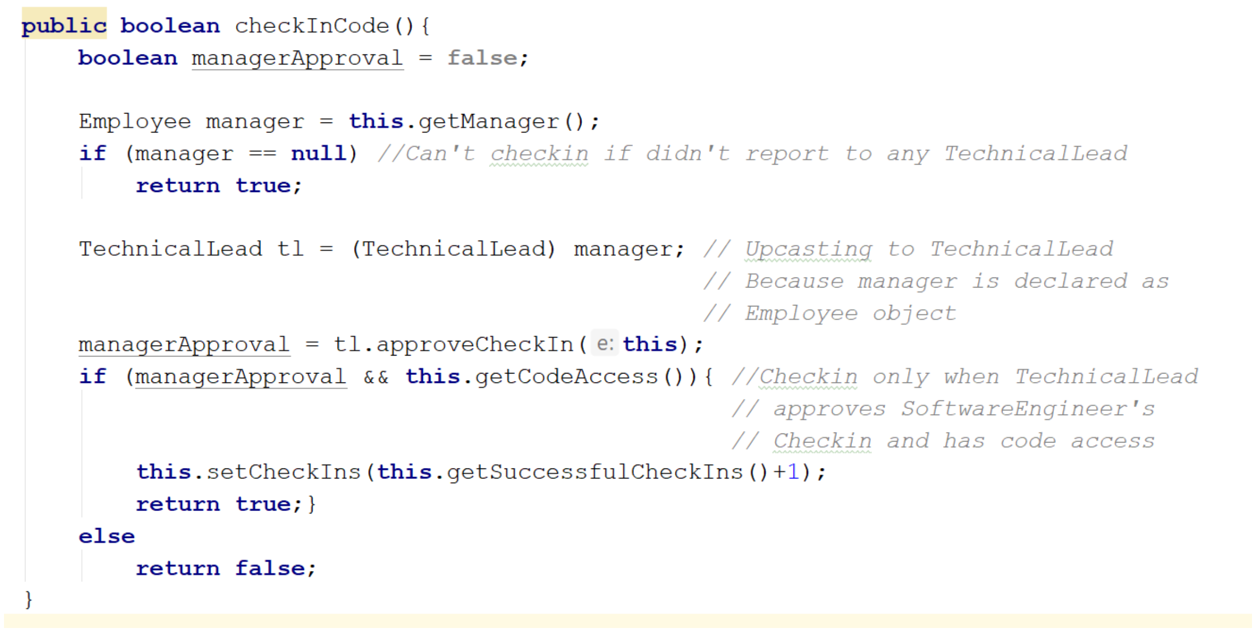
	TechnicalEmployee. TechnicalLeads should have a default head count of 4.
public boolean hasHeadCount()	Should return true if the number of direct reports this manager has is less than their headcount.



```
public int headCount ;
private ArrayList<SoftwareEngineer> team = new ArrayList<>();

public TechnicalLead (String name){
    super(name);
    this.setBaseSalary(this.getBaseSalary()*1.3);
    this.headCount = 4;

}

public boolean hasHeadCount(){
    if (team.size() < this.headCount){
        return true;
    }
    else {
        return false;
    }
}
```

When the SoftwareEngineer can check-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"

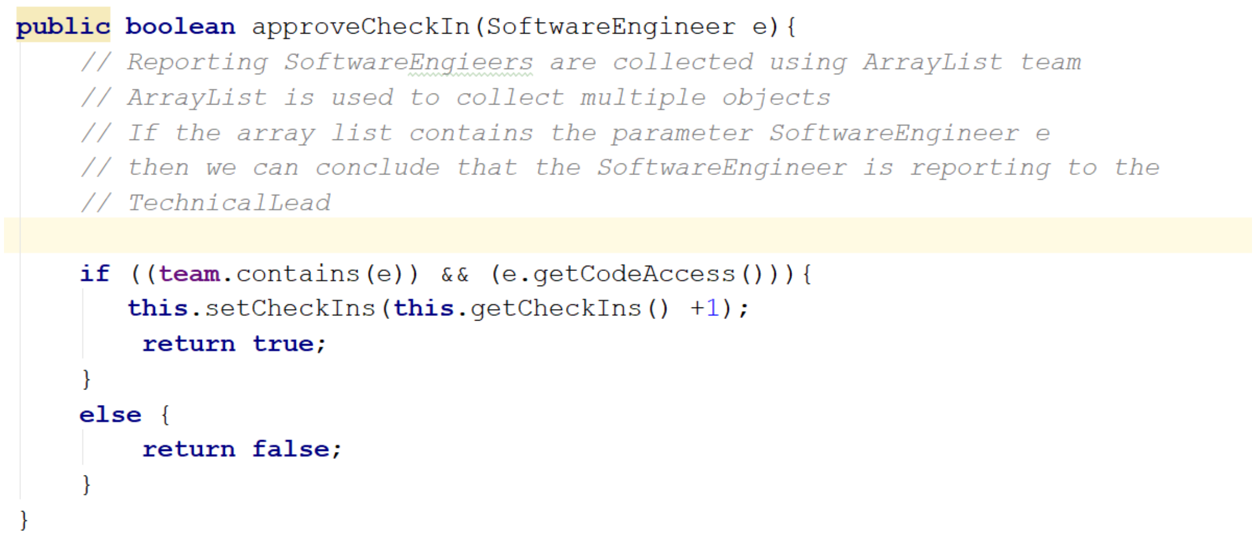


```
public boolean checkInCode(){
    boolean managerApproval = false;

    Employee manager = this.getManager();
    if (manager == null) //Can't checkin if didn't report to any TechnicalLead
        return true;

    TechnicalLead tl = (TechnicalLead) manager; // Upcasting to TechnicalLead
                                                // Because manager is declared as
                                                // Employee object
    managerApproval = tl.approveCheckIn( e: this);
    if (managerApproval && this.getCodeAccess()){ //Checkin only when TechnicalLead
                                                  // approves SoftwareEngineer's
                                                  // Checkin and has code access
        this.setCheckIns(this.getSuccessfulCheckIns()+1);
        return true;}
    else
        return false;
}
```

When will TechnicalLead approve?

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

How do we know the SoftwareEngineer is reporting to the TechnicalLead?



```java
public boolean approveCheckIn(SoftwareEngineer e){
    // Reporting SoftwareEngieers are collected using ArrayList team
    // ArrayList is used to collect multiple objects
    // If the array list contains the parameter SoftwareEngineer e
    // then we can conclude that the SoftwareEngineer is reporting to the
    // TechnicalLead

    if ((team.contains(e)) && (e.getCodeAccess())){
       this.setCheckIns(this.getCheckIns() +1);
        return true;
    }
    else {
        return false;
    }
}
```