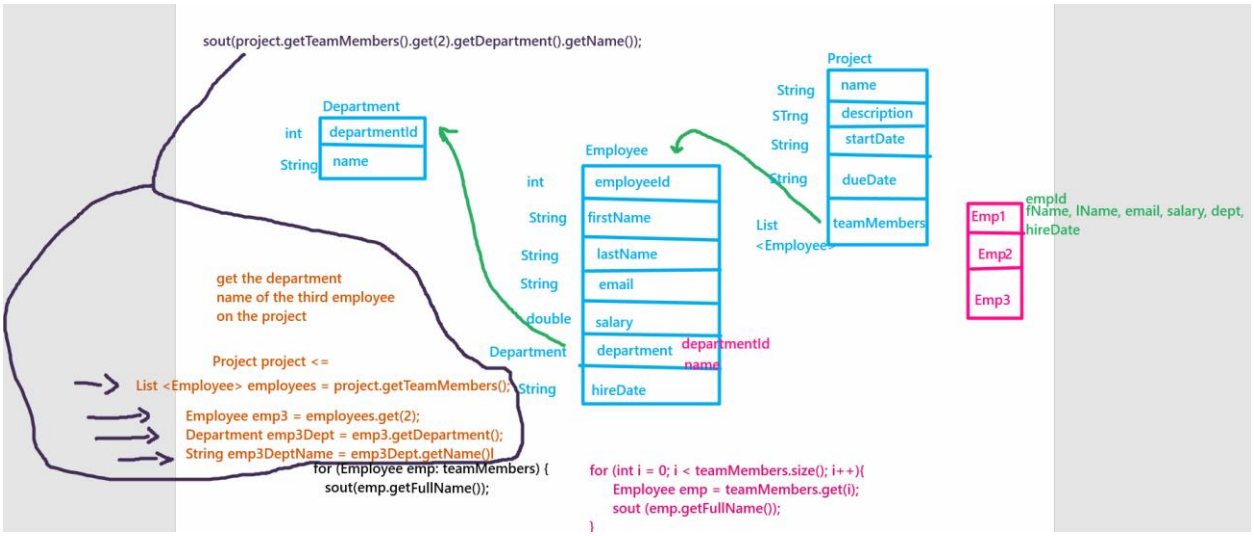
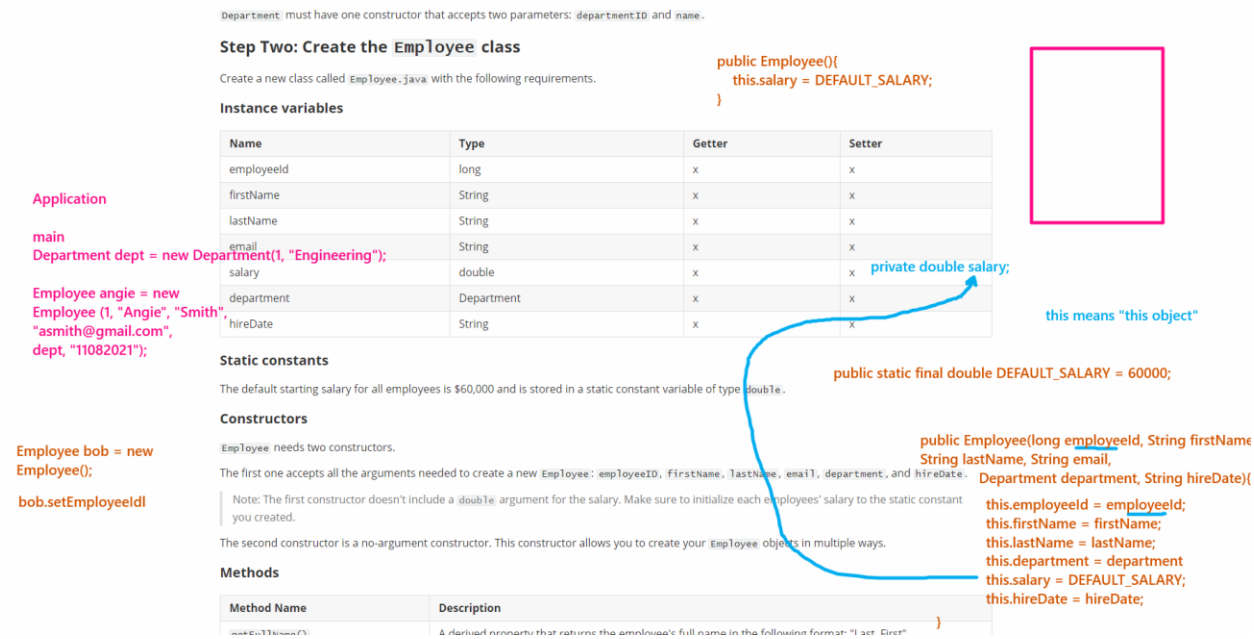

sout(project.getTeamMembers().get(2).getDepartment().getName());
Department
int departmentId
String name
Employee
int employeeId
String firstName
String lastName
String email
double salary
Department department
String hireDate
Project
String name
STrng description
String startDate
String dueDate
List <Employee> teamMembers
Emp1
Emp2
Emp3
empId fName, lName, email, salary, dept, hireDate
get the department name of the third employee on the project
Project project <=
List <Employee> employees = project.getTeamMembers();
Employee emp3 = employees.get(2);
Department emp3Dept = emp3.getDepartment();
String emp3DeptName = emp3Dept.getName()
for (Employee emp: teamMembers) {
sout(emp.getFullName());
for (int i = 0; i < teamMembers.size(); i++){
Employee emp = teamMembers.get(i);
sout (emp.getFullName());
}


Department must have one constructor that accepts two parameters: departmentID and name.
Step Two: Create the Employee class
Create a new class called Employee.java with the following requirements.
Instance variables
Name | Type | Getter | Setter
employeeId | long | x | x
firstName | String | x | x
lastName | String | x | x
email | String | x | x
salary | double | x | x
department | Department | x | x
hireDate | String | x | x
Static constants
The default starting salary for all employees is $60,000 and is stored in a static constant variable of type double.
Constructors
Employee needs two constructors.
The first one accepts all the arguments needed to create a new Employee: employeeID, firstName, lastName, email, department, and hireDate.
Note: The first constructor doesn't include a double argument for the salary. Make sure to initialize each employees' salary to the static constant you created.
The second constructor is a no-argument constructor. This constructor allows you to create your Employee objects in multiple ways.
Methods
Method Name | Description
A derived property that returns the employee's full name in the following format: "Last, First"
public Employee(){
this.salary = DEFAULT_SALARY;
}
Application
main
Department dept = new Department(1, "Engineering");
Employee angie = new Employee (1, "Angie", "Smith", "asmith@gmail.com", dept, "11082021");
Employee bob = new Employee();
bob.setEmployeeId
private double salary;
this means "this object"
public static final double DEFAULT_SALARY = 60000;
public Employee(long employeeId, String firstName
String lastName, String email,
Department department, String hireDate){
this.employeeId = employeeId;
this.firstName = firstName;
this.lastName = lastName;
this.department = department
this.salary = DEFAULT_SALARY;
this.hireDate = hireDate;
}

public class Employee{

private String firstName;
private String lastName;

public String getFullName(){
return lastName + ", " + firstName;
}

hireDate	String	x

Static constants

The default starting salary for all employees is $60,000 and is stored in a static constant variable of type `double`

Constructors

`Employee` needs two constructors.

private String fullName;

The first one accepts all the arguments needed to create a new `Employee`: `employeeID`, `firstName`, `lastName`

> Note: The first constructor doesn't include a `double` argument for the salary. Make sure to initialize each em you created.

The second constructor is a no-argument constructor. This constructor allows you to create your `Employee` obj

Methods

Method Name	Description
`getFullName()`	A derived property that returns the employee's full name in the foll
`raiseSalary(double percent)`	A method that raises the employee's salary by x percent

Step Three: Create the `Project` class

Create a new class called `Project.java` with the following requirements.

: of Private Modifiers on Inheritance

if a Parent class has a private property or method, the child class CANNOT access it directly.

to set a private property in a parent class must use a setter

- the following example:

Access modified rule above the inheritance relationship

```
Vehicle {

String privateMethod() {
eturn "private";
```

```
public class GarageDemo {

  public static void main(String args[]) {

        Car myCar = new Car();
        myCar.setup();

        myCar.privateMethod();

        …
```

This is an invalid call.

super means parent

....) {

this means current class

);

ructors on Parent Classes: Example

e child class, Truck will now have to implement a
vith a super(...) call.

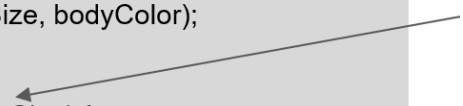
```
psvm(S[] args){
    Truck myTruck = new Truck(4, 450, "Blue");
```

```
k extends Vehicle {

ublic Truck(int numberOfWheels, double engineSize, String bodyColor) {
    super(numberOfWheels, engineSize, bodyColor);
```

```
icle {

ublic Vehicle(int numberOfWheels, double engineSize, String bodyColor)

        this.numberOfWheels = numberOfWheels;
        this.engineSize = engineSize;
        this.bodyColor = bodyColor;
```

The super(...) call is a call to the parent constructor, providing any required parameters

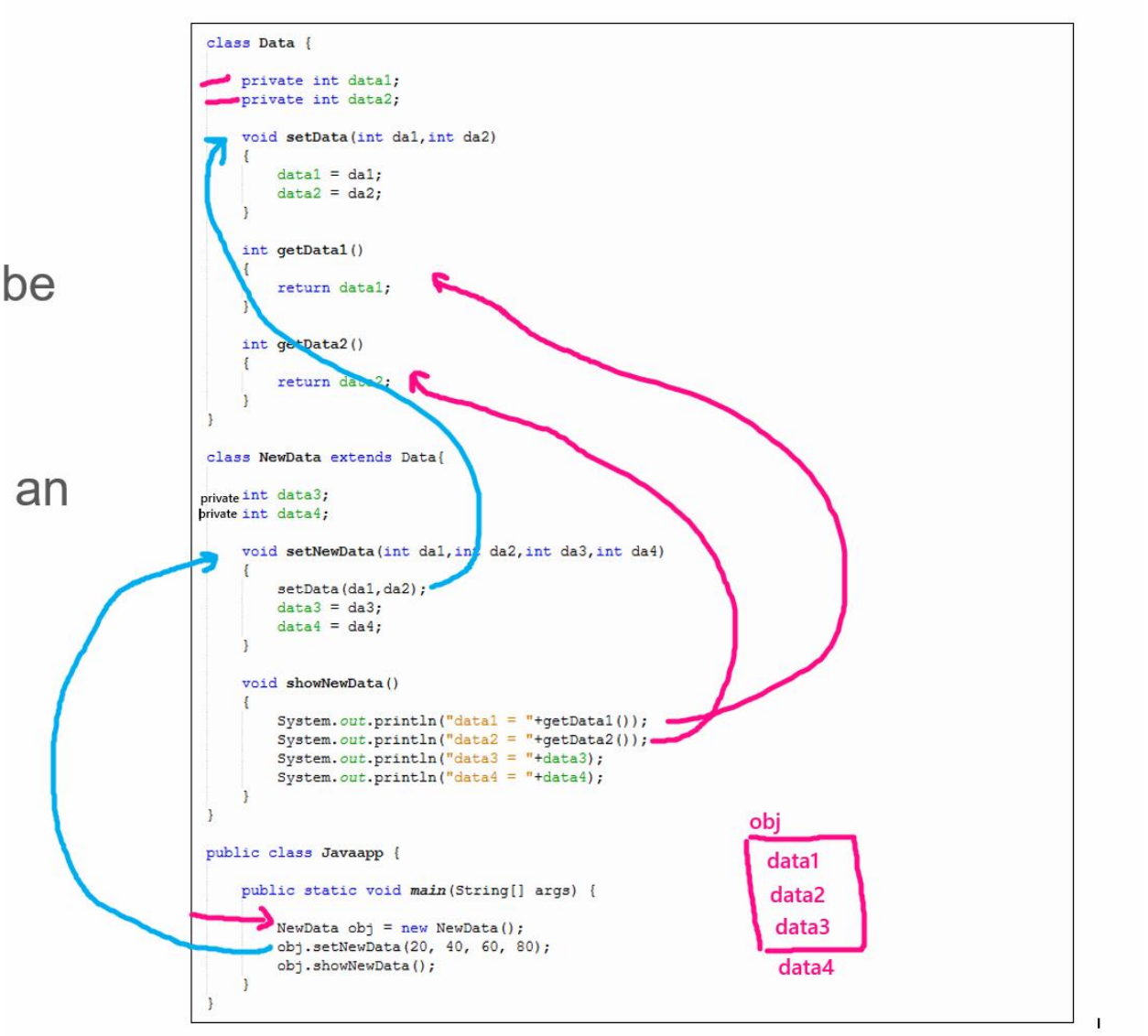
```
public class Truck extends Vehicle {

        public Truck(int numberOfWheels, double engineSize, String bodyColor) {
                super(numberOfWheels, engineSize, bodyColor);
        }

        public Truck(int numberOfWheels, double engineSize) {
                super (numberOfWheels, engineSize);
        }
```



```
public Truck() {            // default constructor, because it requires no params
   super(10, 14.3, "blue");
}
```

be

an



```
class Data {

    private int data1;
    private int data2;

    void setData(int da1,int da2)
    {
        data1 = da1;
        data2 = da2;
    }

    int getData1()
    {
        return data1;
    }

    int getData2()
    {
        return data2;
    }
}

class NewData extends Data{

    private int data3;
    private int data4;

    void setNewData(int da1,int da2,int da3,int da4)
    {
        setData(da1,da2);
        data3 = da3;
        data4 = da4;
    }

    void showNewData()
    {
        System.out.println("data1 = "+getData1());
        System.out.println("data2 = "+getData2());
        System.out.println("data3 = "+data3);
        System.out.println("data4 = "+data4);
    }
}

public class Javaapp {

    public static void main(String[] args) {

        NewData obj = new NewData();
        obj.setNewData(20, 40, 60, 80);
        obj.showNewData();
    }
}
```