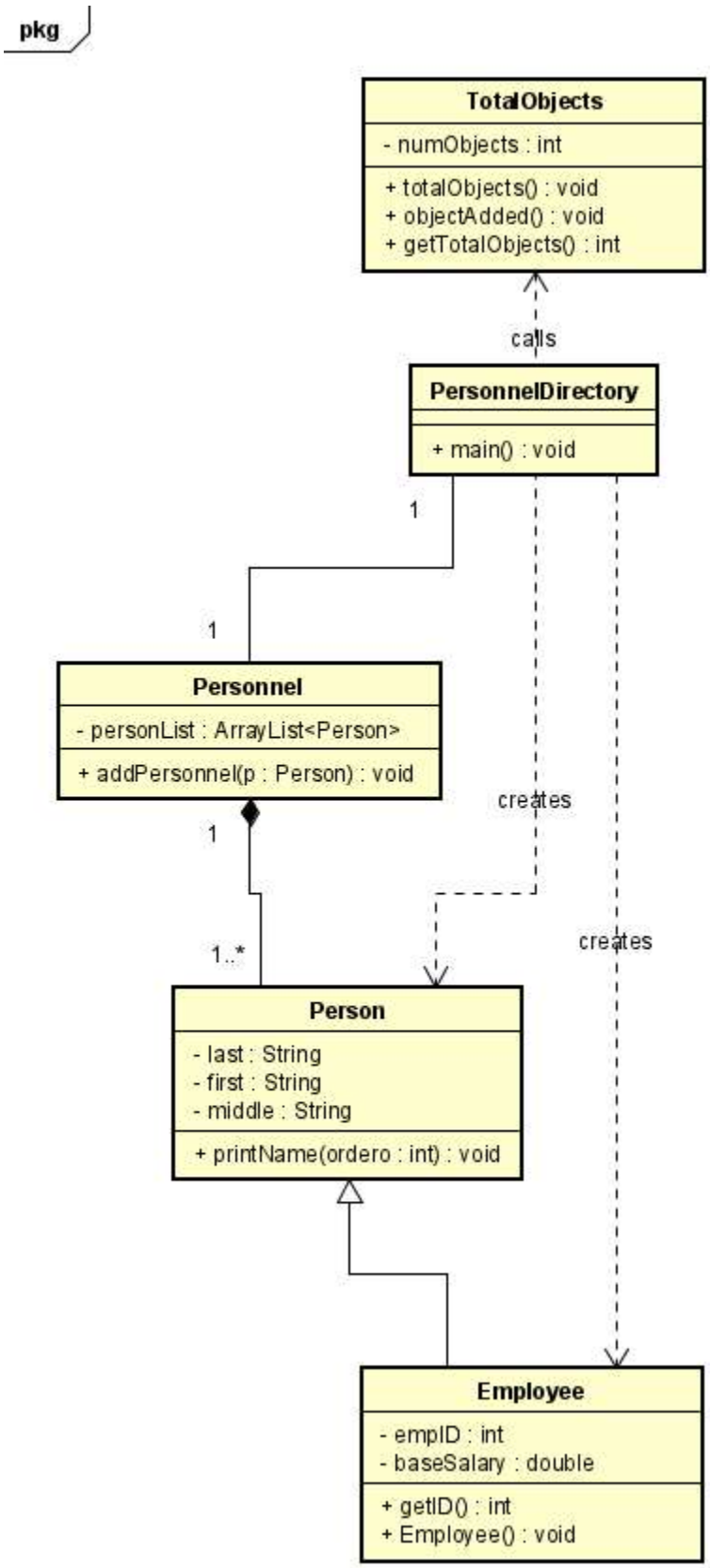
Directory Management System Project

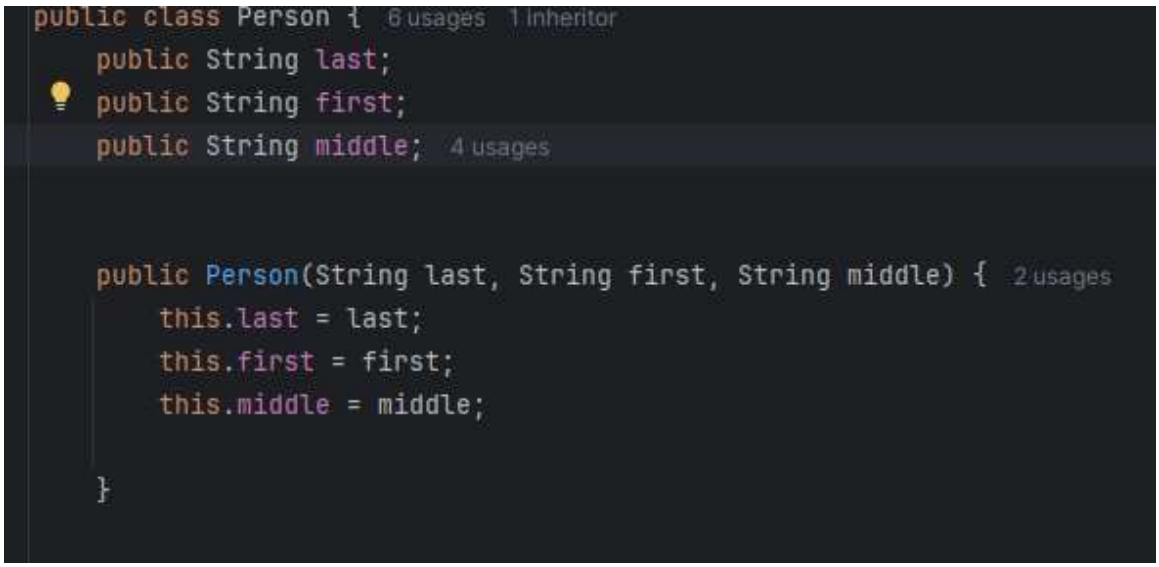
Phase I Part 1

Use Astah to draw a class diagram diagram. Use proper UML notation. Take a clear screenshot of your completed diagram and paste it on this page.


pkg
TotalObjects
- numObjects : int
+ totalObjects() : void
+ objectAdded() : void
+ getTotalObjects() : int
calls
PersonnelDirectory
+ main() : void
1
1
Personnel
- personList : ArrayList<Person>
+ addPersonnel(p : Person) : void
creates
1
1..*
creates
Person
- last : String
- first : String
- middle : String
+ printName(ordero : int) : void
Employee
- empID : int
- baseSalary : double
+ getID() : int
+ Employee() : void

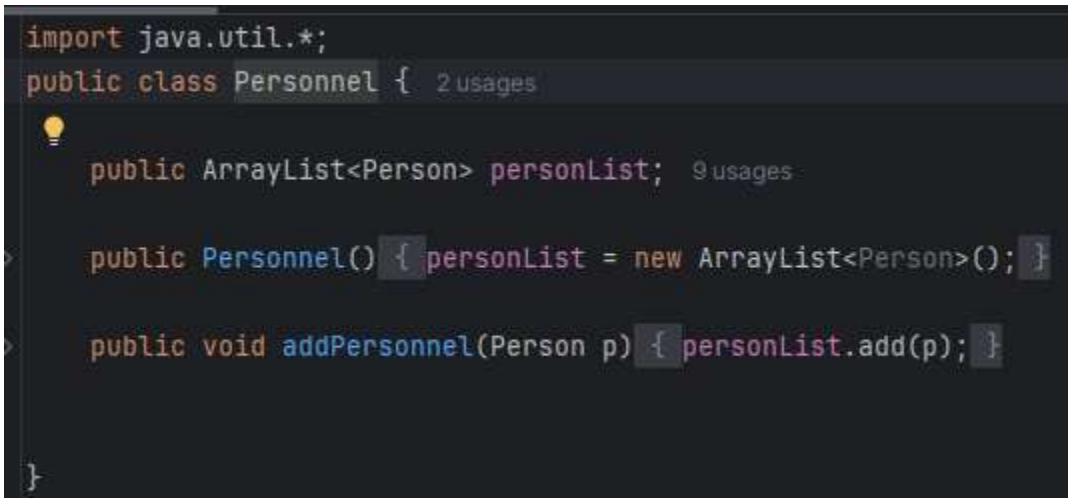
Phase I Part 2

Identify the places in the code where there are object-oriented concept violations, content coupling, common coupling, control coupling, and stamp coupling situations. On this page, paste the code segments that correspond to each situation and explain how you would fix object-oriented concept violations, common coupling, control coupling, and content coupling issues. You may add pages if necessary.

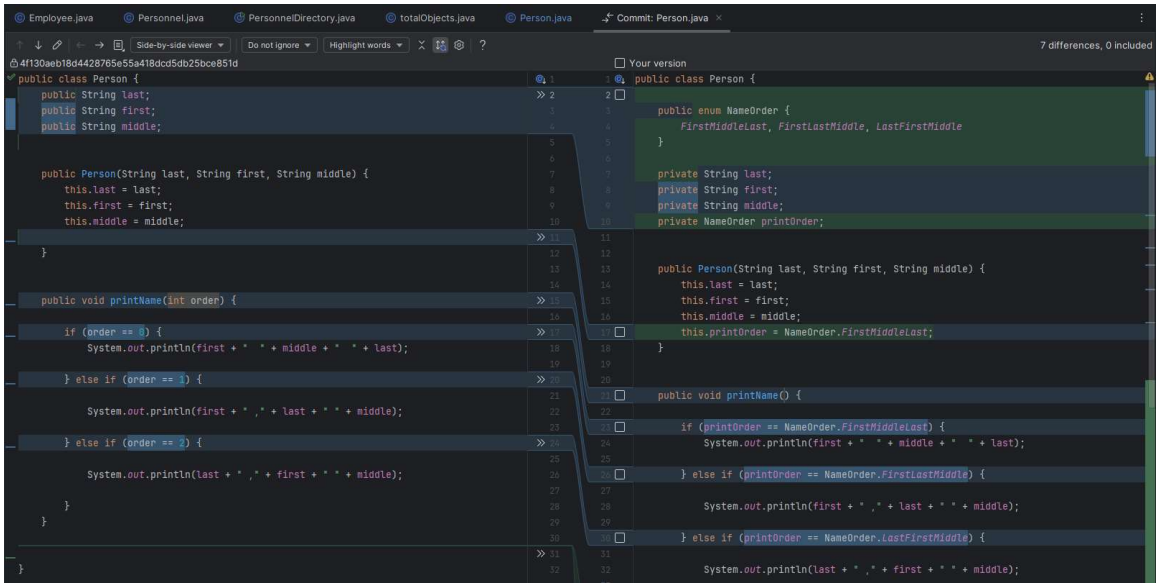


Given that the variables in this class are public, there is no information-hiding. I would fix this by classifying the variables private, and creating the appropriate setters and getters.

baseSalary has no setter. Or, getter. Therefore, we have no way of knowing the base salary of an employee, or of updating it after. I would create a getter and setter for this.



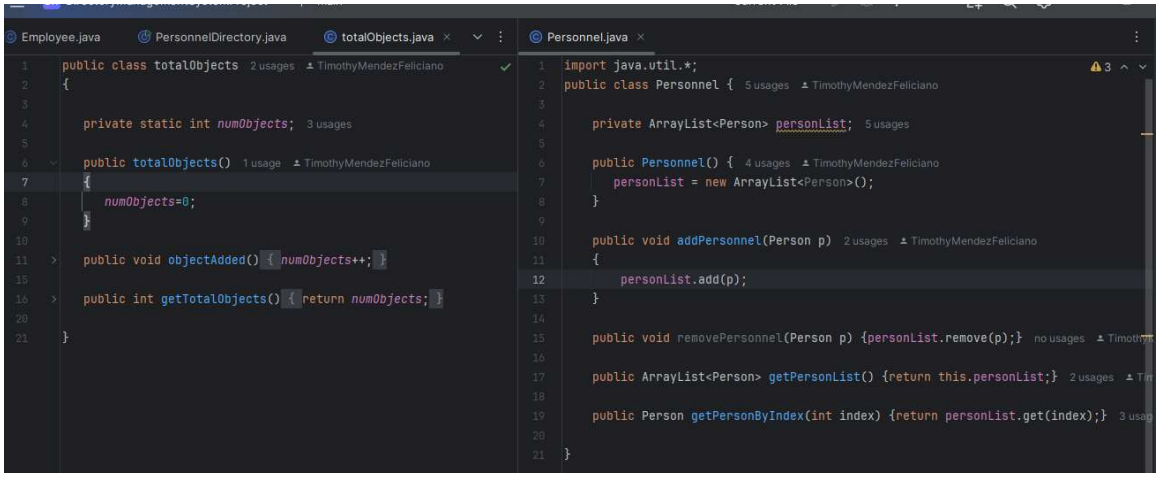
The variable personList has the same information-hiding as the previous. It'll require a getter and setter to work properly.



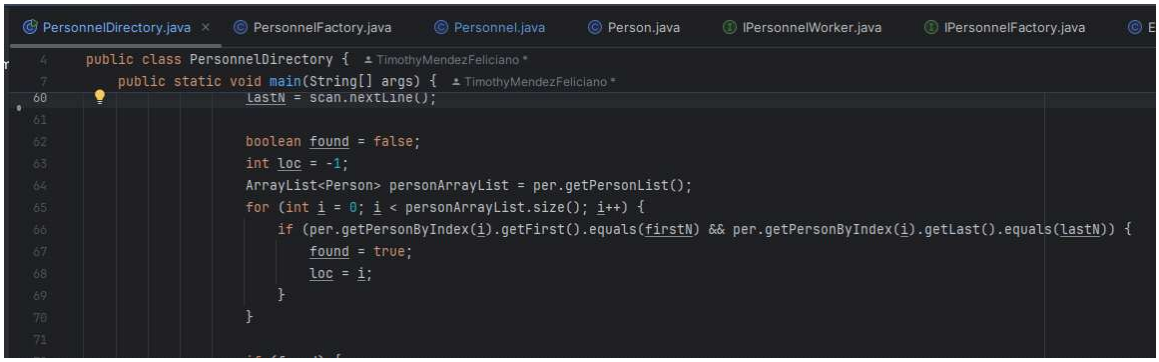
printName(int order) created a **control coupling error**. Forcing whoever calls it to know, by int, what the order represented. I have created an enum that holds the 3 printing options. This enum gets saved to the Person. It also has its own getters and setters. The enum is public. However, the object calling person need not have to control the order to display now.

```
public class Personnel {

	private ArrayList<Person> personList;

	public Personnel() {
	   personList = new ArrayList<Person>();
	}

	public void addPersonnel(Person p)
	{
		personList.add(p);
	}

	public void removePersonnel(Person p) {personList.remove(p);}

	public ArrayList<Person> getPersonList() {return this.personList;}

	public Person getPersonByIndex(int index) {return
personList.get(index);}

}
```

Missing functionality. We never know if they might want to remove Personnel. Also, making personList private and adding the appropriate getters. Be it getting the entire list, or byIndex.



The totalObjects class is largely irrelevent. As it gets updated only after adding personnel. This creates a **common coupling** issue. Since ArrayList<Person> personList keeps track of its own size. And so, totalObjects becomes redundant. I would replace this by simply using the size of personList, which the object itself is accesible via its own getter.



This is also another example of **Control Coupling**. There is no reason personnelDirectory should be iterating over personList. This should be done by the Personnel class. So, I will create a getPersonByName function within personnel that filters over personList, and provides a the specific person we are looking for.

Phase II Part 2

After you have incorporated the PersonnelFactory, use Ashta to draw a UML class diagram of the Personnel Directory. Use proper UML notation. When you have completed your diagram, take a clear screenshot of your diagram and paste it on this page.