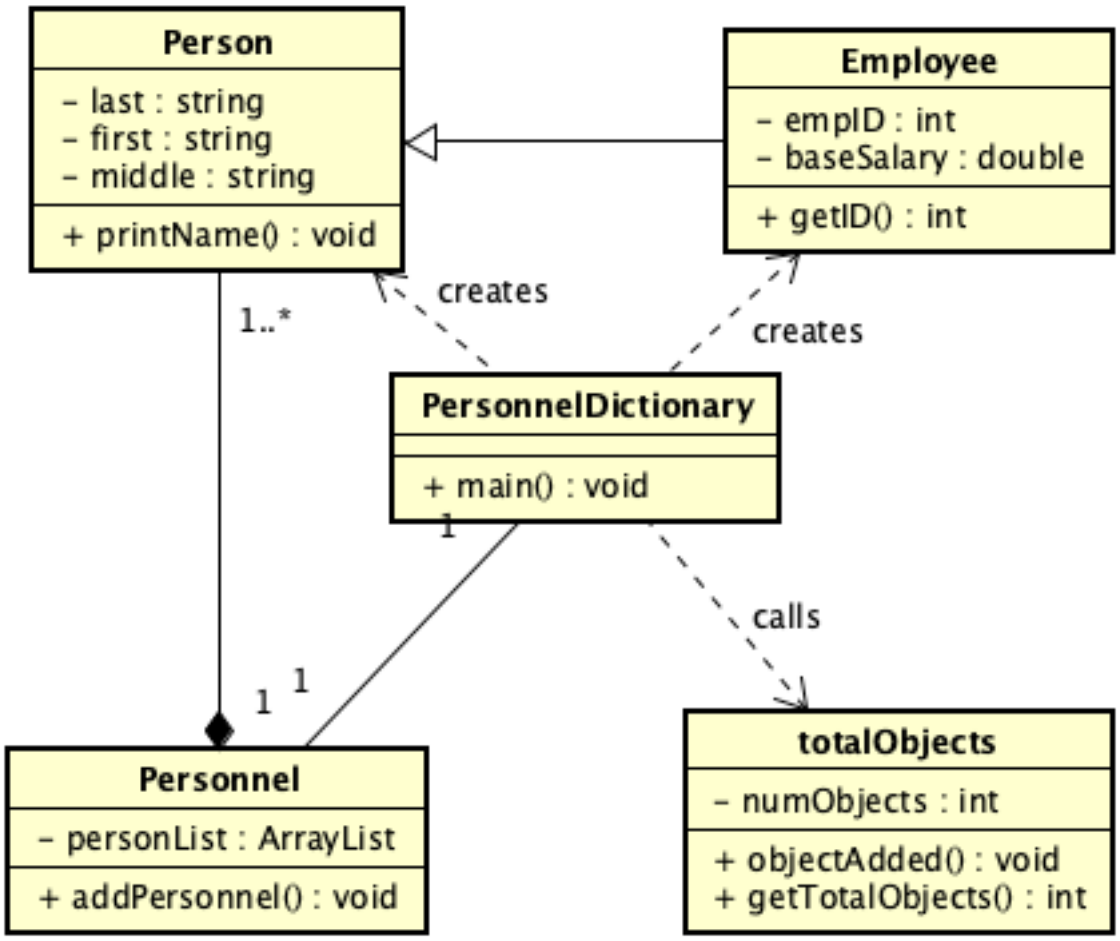
Directory Management System Project

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.



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.

Object-oriented concept violations:

```
public class Personnel {
    public ArrayList<Person> personList;
```

- Direct access to the personList attribute of the Personnel class from the PersonnelDirectory class. This breaks the encapsulation principle.
- Solution: Provide public getter and setter methods for the personList attribute in the Personnel class and use those methods instead of direct access.

```
public class Person {
    public String last;
    public String first;
    public String middle;
```

- Direct access to the attributes (first, last, middle) of the Person class from the PersonnelDirectory class.
- Solution: Provide public getter methods for these attributes in the Person class.

```
public class Employee extends Person{
    private int empID;
    private double baseSalary;
```

- baseSalary is a private attribute in Employee class and unable to modify or view.
- Solution: Provide getter and setter methods for baseSalary.

Content coupling:

```
}else
{
     System.out.println("not found");
     Person p1  = new Person(lastN, firstN, " ");
     per.personList.add(p1);
     total.objectAdded();
}
```

- PersonnelDirectory directly accesses and modifies the personList of the Personnel class.
- Solution: Use methods in the Personnel class to abstract this behavior.

Common coupling:

```
public class totalObjects
{
   private static int numObjects = 0;
   public totalObjects()
   {
      numObjects=0;
   }
   public void objectAdded()
   {
      numObjects++;
   }
   public int getTotalObjects()
   {
      return numObjects;
   }
}
```

- The totalObjects class has a static variable numObjects which can be accessed and modified by multiple classes.
- Solution: Encapsulate the numObjects variable properly and provide methods to increment or get its value.

Control coupling:

```
boolean found = false;
int loc =-1;
for(int i =0; i <per.personList.size(); i++)
{
     if( per.personList.get(i).first.equals(firstN) &&
per.personList.get(i).last.equals(lastN))
{
     found = true;
     loc = i;
}
}
if(found)
{
     System.out.println("Found");
```

```
    per.personList.get(loc).printName(0);
}else
{
    System.out.println("not found");
    Person p1  = new Person(lastN, firstN, " ");
    per.personList.add(p1);
    total.objectAdded();
}
```

- Control coupling occurs when one module (or method) controls the flow of another by passing it information on what to do. In PersonnelDirectory class case 2, it indicated if not found, system creates a new Person and added to personList.
- Solution: Using Factory Design pattern and getter method on personList.

Stamp coupling:

```
Employee e1  = new Employee(lastN, firstN, middleN, empID, salary);
per.addPersonnel(e1);
total.objectAdded();
```

- When searching for a person in the PersonnelDirectory class, both first and last names are used as arguments which might not always be necessary.
- Solution: Create a search method in the Personnel class that can handle different search criteria more effectively.

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.