CSE 460: Software Analysis and Design

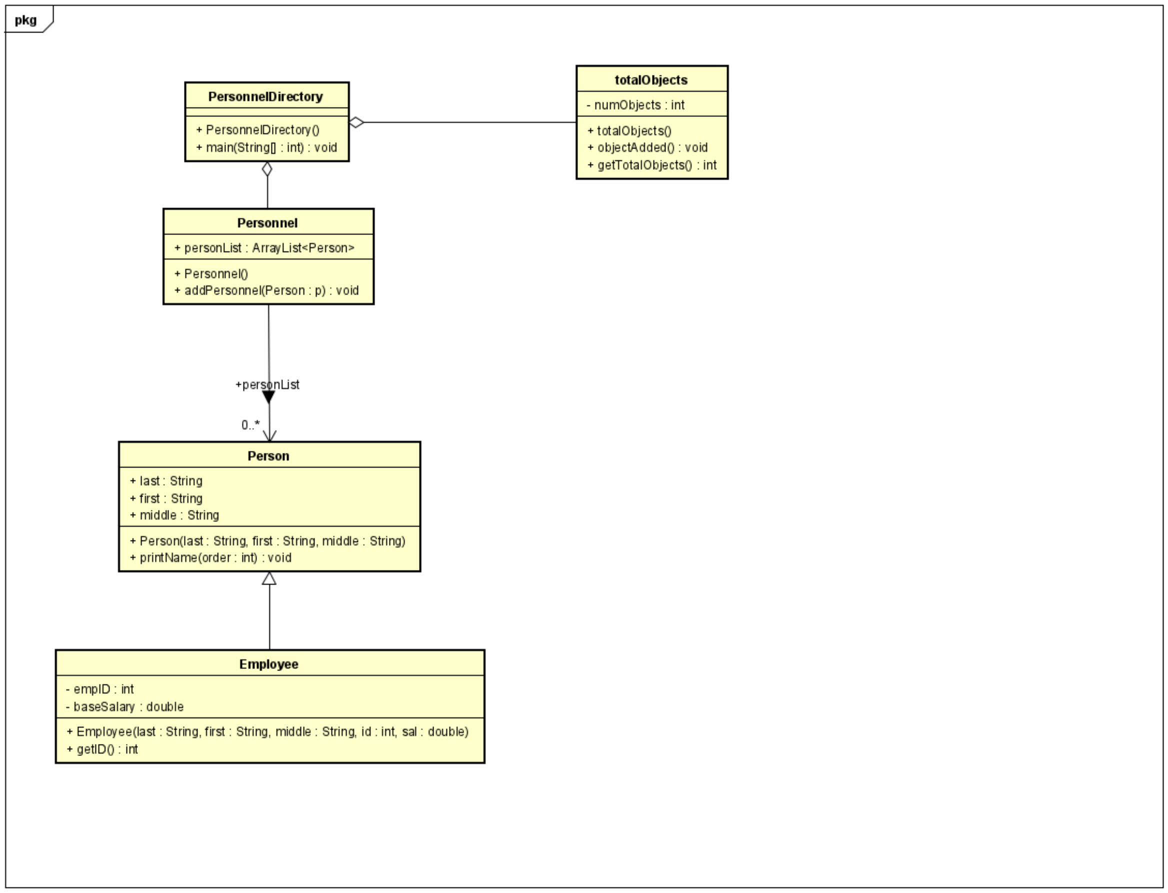
Directory Management System Phase I Submission

Directions:

Complete your work for Phase I Parts 1 and 2 in this document. Save and submit as a **single** PDF titled “Last Name_First Name_Directory Management System Project_Phase I_Submission”.

Phase I, Part 1

Use the Astah tool to draw the class diagram for the current implementation of the university system. Use correct UML notations. When you have completed the diagram, take a clear screenshot and paste it in the space provided.


pkg
PersonnelDirectory
+ PersonnelDirectory()
+ main(String[] : int) : void
totalObjects
- numObjects : int
+ totalObjects()
+ objectAdded() : void
+ getTotalObjects() : int
Personnel
+ personList : ArrayList<Person>
+ Personnel()
+ addPersonnel(Person : p) : void
+personList
0..*
Person
+ last : String
+ first : String
+ middle : String
+ Person(last : String, first : String, middle : String)
+ printName(order : int) : void
Employee
- empID : int
- baseSalary : double
+ Employee(last : String, first : String, middle : String, id : int, sal : double)
+ getID() : int

Phase I, Part 2

In the code, identify object-oriented concept violations, content coupling, common coupling, control coupling and stamp coupling situations. Copy and paste the code segments that shows each coupling situation in the space provided. *You may use additional space as necessary.*

1. **Object-Oriented Concept Violations**

   Encapsulation:
   ```
   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;
   }
   }
   ```
   Abstraction:
   ```
   public class Person {
        public String last;
        public String first;
        public String middle;
   ……
   }
   ```

   How would you fix these violations?
   Encapsulation: to fix the modularity concerns, separating unrelated concerns and encapsulating them into separate modules by refactoring the code into a getter class to return the index of the founded person and place the interface method within Personnel class.

   Abstraction: Since the variables are used to define the last name, first name, and middle should not be accessible from outside of Person class, these variables should be in private to block the access to do information hiding. The other classes access the interface should access the data abstraction through getter methods.

2. **Content Coupling**
   ```
   Person p1 = new Person(lastN, firstN, " ");
   per.personList.add(p1);
   ```

   How would you fix this?
   Instead of connect to inner working of Person class, using the method 'public void addPersonnel(Person p) {personList.add(p);}' to perform the add action in personList.

3. **Common Coupling**
   ```
   if(per.personList.get(i).first.equals(firstN) &&
   per.personList.get(i).last.equals(lastN)){
        found = true;
        loc = i;
   }
   ```

How would you fix this?
First, make the String variables first, last, and middle as private in order to block the access of data from other classes. Create a return method in Personnel class to return a complete list, then perform same necessarily functionalities in main class by calling this return method. Create return methods in Person class to get the first and last name separately, then get the name by calling these methods.

4. **Control Coupling**

```
 public void printName(int order){
 if(order == 0) {
      System.out.println(first + "  " + middle + "  " + last);
 }
else if(order == 1) {
      System.out.println(last + " ," + middle + " " + first);
 }
 else if(order == 2) {
      System.out.println(last + " ," + first + " " + middle);
 }
}

for(int i=0; i<per.personList.size(); i++) {
per.personList.get(i).printName(order);}
```

How would you fix this?
Instead of using one method, creating three different methods to perform the function of the order of 0, 1, 2. Then in wherever use printName method, replace the calling printName function by the new one of three printName method individualy.

5. **Stamp Coupling**

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