## packages

| package | usage |
| --- | --- |
| conf | object definition, it's usually fixed and able to be utilized in other packages |
| service | implementation of business requirements, associate with conf package to solve problems |

## modules

### utils

- get_json_list_by_file
  - Get json list from resource. It'll find the file from `/resources/` path by the input file name. Noted that it might raise `JSONDecodeError` and `FileNotFoundError`.

### conf.employee_definition

- EmployeeBase
  - This abstract class defines some intrinsic fields, like `eid`, `first_name`. In general, all the employees will have these fields. The only abstract method is `print_info`.
- Employee
  - This class extends `EmployeeBase` and has two more fields, `salary` and `manager`. If an employee has to set up a manager, invoke `set_manager` to bind the relation, `set_manager` will invoke `Manager.register(Employee)` as well . About `Manager.register(Employee)`, the input employee will be regarded as one of the members of the Manager.
- Manager
  - This class extends `Employee` and has one more field, `_member_list`. `register` allows to add the input `Employee` to be one of the members it has.
- EmployeeJson
  - It implements pydantic. Most of the validations have been done by pydantic, like properties' name check, type check. For `manager` and `salary`, Optional[int] indicates that undefined property or null is allowed for the corresponding field. If we want to have advanced validation, take `first_name_must_be_english_letter` as your reference.

### service.employee_service

- EmployeeMapper
  - The mapper aims to transfer `EmployeeJson` to `Employee`. In the `__init__`, it initializes a new dict which is quite crucial to the mapper, the key would be id, value would be `Employee` or `Manager`. It offers a few upsides:
    - improve the performance of the transformation,
    - since the kye is id, it makes sure there's only one `Employee` or `Manager` by id in the dict
- get_employee_list
  - This is the main function of this module. In the beginning, it'll get json list by the input file name and transfer each json data to `EmployeeJson`, and then store the data by `EmployeeJson.get_id()` with a dict. Duplicate id might be found here and raise a ValueError.

Afterward, `EmployeeMapper` will be leveraged and does the following:

- map every `EmployeeJson` to `Employee`
- set up manager if `EmployeeJson.get_manager()` is not None
- check if manager id couldn't be found with the dict.
- return all the `Employee` which are created by the mapper

Before it returns the list from `EmployeeMapper`, the list will be sorted by the following rules in order:

- first_name
- if the `Employee` is also a `Manager, move to the top
- if the `Employee` doesn't have a 'Manager', move to the top

- print_employee_list
  - It'll for loop the input `Employee` list and invoke `Employee.print_info()`. Firstly, print the name. Secondly, if the `Employee` is also a `Manager`, print the members it has. Lastly, invoke `get_total_salary` to print the total salary.
- get_total_salary
  - Sum the total salary from the input `Employee` list