Факультет компьютерных технологий и прикладной математики
Кафедра вычислительных технологий
02.03.02

Паттерны программирования
Лабораторная работа № 3. Наследование. Конструкторы. Обмен сообщениями. Диаграмма классов. Диаграмма последовательностей. JSON.

Каждое задание должно быть загружено на личный git-репозиторий отдельным коммитом. Лабораторная работа выполняется в одной папке. Защита работы возможна на любой лабораторной работе от 1 до 16. Каждое из шести заданий проверяется отдельно с учетом вопросов преподавателя. Задание засчитывается отдельно, лабораторная работа зачтена в случае выполнения всех 6 заданий.

Если часть задач выполнена в один коммит, работа не проверяется. Если все коммиты сделаны в один час, работа не проверяется.

Часть заданий выполняется по вариантам.

Все задания дальше будут представлять из себя выполнение одного общего программного продукта. Необходимо отследить прием на должности, увольнение и смену должностей в некоторой абстрактной фирме с несколькими отделами. В каждом отделе есть должность руководителя, перечень выполняемых работ и штатное расписание (набор должностей). Одного из сотрудников выбирают как контактное лицо. В базе хранится информация о соискателях должностей с их трудовыми книжками, соискателей и сотрудников можно принять на работу, уволить с должности. Хранится информация о зарплате. Зарплата может быть только оклад, может быть фиксированная премия в процентах от общей месячной зарплаты, может быть фиксированная надбавка в рублях, может быть выплачиваемая в отдельных случаях премия в процентах от оклада. Может быть что-то одно, может чего-то одного не быть, может быть голый оклад, может быть все вместе, может быть штраф в процентах от зарплаты. Для отдела должна быть возможность упорядочить по зарплате в текущем месяце. Должна быть возможность получить список самых высокооплачиваемых и низкооплачиваемых сотрудников. Работа выполняется в рамках паттерна MVC.

Для дальнейшего решения указанной задачи необходимо теперь спроектировать все классы, участвующие в указанной модели.

Задание 1. Наследование

Задачи

1. Реализуем расчет зарплат. Для этого создадим класс Salary с методом get_salary, оставим метод пустым.
2. Создадим наследника Fix_sal класса Salary, с конструктором, принимающим один параметр, одним полем @fixed и переопределим метод get_salary, заставив его возвращать указанное поле.
3. Создать наследника класса Fix_sal класс Rub_sal с конструктором, содержащим два параметра, оклад и размер надбавки. В конструкторе вызывать конструктор родителя. Поле @add_rub, после этого переопределите метод get_salary, заставив его возвращать оклад с надбавкой. В этом методе обязательно вызвать переопределенный метод родителя.
4. Создать наследника класса Fix_sal класс Percent_sal с конструктором, содержащим два параметра, оклад и размер надбавки в процентах. В конструкторе вызывать конструктор родителя. Поле @add_percent, после этого переопределите метод get_salary, заставив его возвращать оклад с надбавкой, или просто оклад, случайным образом определяя, устанавливать надбавку, или нет. В этом методе обязательно вызвать переопределенный метод родителя.
5. Создать наследника класса Percent_sal класс Rub_percent_sal с конструктором, содержащим три параметра, оклад, размер надбавки в рублях и размер надбавки в процентах. В конструкторе вызывать конструктор родителя. Поле @add_rub, после этого переопределите метод get_salary, заставив его возвращать оклад с надбавкой. В этом методе обязательно вызвать переопределенный метод родителя.
6. В итоге получилось 4 класса, реализующие 4 формата выдачи зарплаты. Далее необходимо учесть еще 4 формата зарплаты, реализовав возможность штрафовать сотрудника. Для этого от каждого из уже созданных классов создать наследника Fine_sal, Fine_rub_sal, Fine_percent_sal, Fine_rub_percent_sal, добавив соответствующее поле, переопределив конструктор (с вызовом конструктора родителя) и метод get_salary.
7. В итоге получилось восемь классов, реализующих 8 форматов выдачи зарплаты, осталось реализовать еще 8 форматов выдачи, учитывающих возможность фиксированной премии в каждом месяце в процентах от общей зарплаты(ВАЖНО, не от оклада). Для этого от каждого из указанных классов постройте наследника Premium_sal, Premium_rub_sal, Premium_percent_sal,

Premium_rub_percent_sal, Premium_fine_sal, Premium_fine_rub_sal, Premium_fine_percent_sal, Premium_fine_rub_percent_sal. В каждом из классов добавьте соответствующее поле, переопределите конструктор, использовав вызов конструктора родителя, переопределите основной метод, используя вызов метода родителя.

8. Постройте диаграмму классов получившейся иерархии.

Вопросы.

a. Объясните, в чем заключается принцип наследования?

b. В чем заключается механизм переопределения методов, как вызывать в переопределенном методе метод родителя?

c. Опишите принципы функционирования конструкторов при использовании наследования.

d. Что изображается на диаграмме классов? Как на диаграмме классов изображается наследование? Переопределение методов?

Задание 2. Ассоциация

Задачи.

1. Добавьте в класс Post поле @salary класса Salary. Реализуйте отношение композиции для этих двух классов, то есть в произвольный момент времени доступ к объектам класса Salary или их создание должно быть возможно только внутри методов объектов класса Post. Продумайте метод salary=, который позволит создавать объект соответствующего наследника класса Salary для объекта класса Post. В методе salary= реализуйте один аргумент — хэш, разберитесь, как это реализуется. Реализуйте геттер, который будет возвращать числовое значение заработной платы.

2. Внесите изменения в класс Post_list, переписав с учетом новой информации методы сериализации и десериализации в YAML и TXT. При необходимости внесите изменения в класс Post.

3. Протестируйте написанный методы.

4. Расширьте диаграмму классов, добавив в нее все написанный классы. Отметьте на диаграмме отношения композиции и агрегации.

Вопросы.

1. Что такое ассоциация, в чем разница между композицией и агрегацией?

2. Как на диаграмме классов обозначаются указанные отношения между классами?

Задание 3. Классы данных

Задачи.

1. Создать класс Employee. Храните в этом классе информацию о фамилии имени отчестве, дату рождения, паспортные данные, телефон,

адрес, электронный адрес. Реализуйте конструктор, геттеры и сеттеры. В случае большого количества аргументов воспользуйтесь аргументом метода — хэш.

2. Реализуйте наследника Employee Skilled_employee, добавив два поля о количестве лет опыта и поле строка с описанием опыта работы.

Задание 3. Работа с коллекциями данных

1. Создайте по аналогии с классами Post_list и Department_list класс Employee_list, реализовав соответствующие методы.

2. В классе Post_list реализуйте метод, строящий экземпляр класса post_list, возвращающий коллекцию должностей, относящихся к заданному в аргументе отделу.

3. В классе Post_list реализуйте метод, строящий экземпляр класса post_list, возвращающий коллекцию вакантных должностей, относящихся к заданному в аргументе отделу.

4. В классе Post_list реализуйте метод, строящий экземпляр класса post_list, возвращающий коллекцию должностей, в названии которых содержится введенная в аргументе строка как подстрока.

5. В классах Department_list Employee_list построить методы, возвращающие коллекцию, содержащую элементы, ФИО или название которых содержит введенную строку как подстроку.

6. В классе Employee_list построить методы, возвращающие коллекцию, содержащую людей, конкретного возраста, младше заданного возраста, старше заданного возраста, находящихся в конкретном возрастном диапазоне.

Задание 4. Обмен сообщениями.

Задания.

1. Реализуйте класс Job, имеющий 5 полей — должность, сотрудник, дата начала работы, дата увольнения (может быть пустым) и процент от ставки(может работать на пол ставки).

2. Создайте класс Job_list — коллекцию элементов класса Job. Реализуйте методы add, get, choose, delete и прочие методы по аналогии с другими классами коллекциями. Отслеживайте выполнение принципа инкапсуляции.

3. Реализуйте сериализацию и десериализацию этого объекта класса.

4. В классе Post создайте поле объекта @jobs_list. Напишите метод принять на должность с аргументом класса Employee и аргументом значением от 0 до 1 — размер ставки. В результате этого метода должен создаваться объект класса Job, с заполненными соответствующим образом полями. Далее должен вызывать метод вступить в должность у Employee, в котором элемент класса Job добавляется к внутренней

коллекции Job_list класса Employee. Нарисовать диаграмму последовательностей для данного метода.

5. Реализовать метод уволить с должности в классе Post с аргументом Job. Нарисовать диаграмму последовательностей для данного метода.

Вопросы.

1. Что обмен сообщениями, как он обозначается в UML диаграммах, расскажите о диаграммах последовательностей.

Задание 5. Комплексная сериализация.

Задания.

1. Продумайте и реализуйте сериализацию и десериализацию всех рассматриваемых данных в YAML. Составьте диаграмму последовательностей для каждой из этих операций. Максимально разделите алгоритм по методам применяемых классов.

2. Продумайте и реализуйте альтернативную сериализацию и десериализацию всех рассматриваемых данных в JSON. Обеспечьте одинаковый интерфейс всех методов(название методов и количество аргументов должно совпадать вне зависимости от способа сериализации) Составьте диаграмму последовательностей для каждой из этих операций. Максимально разделите алгоритм по методам применяемых классов.

Задание 6. Продолжение работы с коллекциями.

Задания.

1. Для класса Jobs_list напишите метод, составляющий новый список, содержащий записи о всех элементах, где должность равна аргументу метода.

2. Для класса Jobs_list напишите метод, составляющий новый список, содержащий записи о всех элементах, где человек указан в аргументах метода.

3. Для класса Jobs_list напишите метод, составляющий новый список, содержащий записи о всех текущих должностях, где человек указан в аргументах метода.

4. Для класса Jobs_list напишите метод, строящий Employee_list всех сотрудников.

5. Для класса Post напишите метод, строящий Employee_list всех сотрудников, находящихся сейчас на данной должности.

6. Для класса Post_list напишите метод, строящий Employee_list всех сотрудников, находящихся сейчас на данных должностях.

7. Для класса Department напишите метод, строящий Employee_list всех сотрудников, находящихся сейчас на данных должностях.

8. Для класса Department_list напишите метод, строящий Employee_list всех сотрудников, находящихся сейчас на данных должностях.

9. Постройте метод класса Employee, рассчитывающий суммарную зарплату данного человека на всех текущих должностях. Для этого в классе Job реализуйте метод get_salary, который умножает заработок на текущую ставку.

10. Постройте методы класса Employee_list, сортирующие людей по зарплате, если они трудоустроены в нашей фирме и другой метод, выводящий первые k человек с самой высокой зарплатой.