Контрольное задание по предмету
«Объектно-ориентированное программирование»

1 Семестр

Описание

Контрольная работа состоит из нескольких частей, каждая из которых базируется на предыдущей контрольной работе и изменяет (или дополняет) ее.

Выбор варианта

Вариант выбирается произвольным образом и один для всех контрольных работ.

Содержание отчета

Отчет предоставляется в печатной форме. Содержит:

- титульный лист;
- задание для выполнения;
- окончательный вариант исходного кода (после решения последней части контрольной работы) на языке Java;
- исходный код класса (-ов) с использованием которого(-ых) производилось тестирование конструкторов и методов реализованных классов.

Контрольная работа № 1. Задание

Создайте пакет. Имя создаваемого пакета должно соответствовать варианту.

Для каждой используемой переменной или константы, атрибута, метода, класса выбирать имена в соответствии с конвенциями именования.

Реализуйте классы в соответствии с вариантом.

Определите поля класса как приватные (доступные только внутри класса), а методы и конструкторы – публичными (доступные всем другим классам).

При использовании значений по умолчанию (например, при реализации конструкторов) для этих самых значений использовать статичные публичные константы. Имена констант записываются следующим образом: все буквы заглавные, слова разделяются между собой символами подчеркивания.

Создайте отдельный класс для тестирования конструкторов и вызовов методов, созданных в соответствии с вариантом классов.

Используйте массивы вместо коллекций.

Вариант № 1

Пакет – polyclinic.

1. Создайте публичный класс **PatientsCard** – медицинской карты человека, зарегистрированного в поликлинике. Класс не хранит в явном виде информацию о поликлинике.

Поля:

- каждый пациент характеризуется именем, фамилией, адресом проживания;
- каждый пациент имеет страховой полис, и характеризуется его номером.

Конструкторы:

- конструктор может принимать имя и фамилию (номер полиса = 0 и адрес – пустой);
- конструктор может принимать имя, фамилию и адрес (номер полиса = 0);
- конструктор может принимать имя, фамилию, адрес и номер полиса.

Методы:

- создайте метод получения имени;
- создайте метод изменения имени;
- создайте метод получения фамилии;
- создайте метод изменения фамилии;
- создайте метод получения номера страхового полиса;
- создайте метод изменения номера страхового полиса;
- создайте метод получения адреса;
- создайте метод изменения адреса.

2. Создайте публичный класс **Polyclinic** – поликлиники некоторого района.

Поля:

- поликлиника характеризуется номером и адресом;
- класс хранит явным образом массив карт пациентов, зарегистрированных в поликлинике.

Конструкторы:

- конструктор может принимать номер и адрес поликлиники (длина массива пациентов = 0);
- конструктор может принимать номер и адрес, а так же массив пациентов.

Методы:

- создайте метод получения номера поликлиники;
- создайте метод изменения номера поликлиники;
- создайте метод получения адреса поликлиники;
- создайте метод изменения адреса поликлиники;
- создайте метод, возвращающий общее число пациентов, зарегистрированных в поликлинике;
- создайте метод, возвращающий ссылку карту пациента по номеру страхового полиса;
- создайте метод, возвращающий ссылку на массив карт пациентов, проживающих по заданному адресу;
- создайте метод удаления карты (принимает в качестве входного параметра номер полиса, удаляет соответствующий этим данным элемент из массива карт);
- создайте метод добавления карты (принимает в качестве входного параметра ссылку на экземпляр класса **PatientsCard**, расширяет массив карт путем добавления нового элемента в конец массива);
- создайте метод, возвращающий массив карт;
- создайте метод, возвращающий массив карт, отсортированный по адресам.

Вариант № 2

Пакет – university.

1. Создайте публичный класс **Student** – студента некоторой специальности некоторого университета. Класс не хранит явным образом информацию о специальности, номере группы\потока, предметах, университете.

Поля:

- каждый студент характеризуется именем, фамилией, годом поступления, уникальным 6-тизначным номером зачетной книжки.

Конструкторы:

- конструктор может принимать имя и фамилию. При этом номер зачетной книжки – 0;
- конструктор, принимающий имя, фамилию, номер зачетной книжки.

Методы:

- создайте метод получения имени;
- создайте метод изменения имени;
- создайте метод получения фамилии;
- создайте метод изменения фамилии;
- создайте метод получения номера зачетной книжки;
- создайте метод изменения номера зачетной книжки;
- создайте метод получения года поступления;
- создайте метод изменения года поступления.

2. Создайте публичный класс **Group** – студенческой группы. Класс не хранит явно специальность и имя университета.

Поля:

- каждая группа имеет свой номер (уникальный, в пределах специальности);
- класс хранит явным образом массив студентов.

Конструкторы:

- конструктор может принимать номер группы (в этом случае количество студентов = 0);
- конструктор может принимать номер группы, количество студентов (массив студентов только инициируется, но элементы его - пустые);
- конструктор может принимать массив студентов.

Методы:

- создайте метод получения номера группы;

- создайте метод изменения номера группы;
- создайте метод, возвращающий общее число студентов группы;
- создайте метод, возвращающий ссылку на студента по номеру зачетной книжки;
- создайте метод удаления студента (принимает в качестве входного параметра номер зачетной книжки студента, которого нужно удалить, удаляет соответствующий этим данным элемент из массива студентов);
- создайте метод добавления студента (принимает в качестве входного параметра ссылку на экземпляр класса **Student**, расширяет массив студентов путем добавления нового элемента в конец массива);
- создайте метод, возвращающий массив студентов;
- создайте метод, возвращающий массив студентов, отсортированный по фамилиям (и если одинаковые фамилии – то по именам).

Вариант № 3

Пакет – organisation.

1. Создайте публичный класс **Employee** – работника некоторой организации. Класс не хранит явным образом номер или имя подразделения и организации, в которой работает работник.

Поля:
- каждый работник занимает определенную должность;
- каждый работник получает определенное жалование;
- каждый работник характеризуется именем и фамилией.

Конструкторы:
- конструктор может принимать имя и фамилию (должность – инженер, жалование – 30к руб.);
- конструктор может принимать имя, фамилию, должность, жалование.

Методы:
- создайте метод получения имени;
- создайте метод изменения имени;
- создайте метод получения фамилии;
- создайте метод изменения фамилии;
- создайте метод получения должности;
- создайте метод изменения должности;
- создайте метод получения жалования;
- создайте метод изменения жалования.

2. Создайте публичный класс **Department** – подразделения некоторой организации. Класс не хранит явным образом номер подразделения и имя организации, частью которой является.

Поля:

- разные подразделения имеют разные имена;
- класс хранит явным образом массив своих работников.

Конструкторы:

- конструктор может принимать имя подразделения (в этом случае количество работников = 0);
- конструктор может принимать массив работников.

Методы:

- создайте метод получения имени подразделения;
- создайте метод изменения имени подразделения;
- создайте метод, возвращающий общее число работников подразделения;
- создайте метод, возвращающий суммарную зарплату всех работников, относящихся к данному подразделению;
- создайте метод, возвращающий ссылку на работника по фамилии и имени;
- создайте метод увольнения работника (принимает в качестве входных параметров фамилию, имя, должность работника, которого нужно удалить, удаляет соответствующий этим данным элемент из массива работников);
- создайте метод приема работника на работу (принимает в качестве входных параметров ссылку на экземпляр класса **Employee**, расширяет массив работников путем добавления нового элемента в конец массива);
- создайте метод, возвращающий массив работников отдела;
- создайте метод, возвращающий массив работников отдела, отсортированный по фамилиям (и если одинаковые фамилии – то по именам).

Контрольная работа № 2. Задание

В уже созданном пакете реализуйте (измените) классы и интерфейсы в соответствии с вариантом.

По-прежнему, поля класса – приватные (доступные только внутри класса).

При использовании значений по умолчанию (например, при реализации конструкторов) для этих самых значений использовать статичные публичные константы. Имена констант записываются следующим образом: все буквы заглавные, слова разделяются между собой символами подчеркивания.

Создайте отдельный класс для тестирования конструкторов и вызовов методов, созданных в соответствии с вариантом классов.

Поля – даты, должны иметь тип **java.util.Date**.

При переименовании чего-либо следует пользоваться средствами рефакторинга среды разработки.

Вариант № 1

Пакет – polyclinic.

1. Создайте абстрактный класс **MedicalInsurancePolicy** – полиса медицинского страхования. Класс содержит:

Поля:

- номер полиса;
- название страховой компании.

Конструкторы:

- без параметров (названия компании нет, номер полиса 0);
- с 2-мя параметрами: номером полиса и названием компании.

Методы:

- геттеры и сеттеры полей.

2. Создайте класс **ObligatoryMedicalInsurancePolicy** – полиса обязательного медицинского страхования. Этот класс расширяет (наследует) класс **MedicalInsurancePolicy**. Дополнительных элементов этот класс не предлагает.

3. Создайте класс **VoluntaryMedicalInsurancePolicy** – полиса добровольного медицинского страхования. Этот класс также расширяет (наследует) класс **MedicalInsurancePolicy**. Этот класс определяет дополнительные элементы.

Поля:

- общая сумма страховки;
- выплаченная страховая сумма.

Конструкторы:

- без параметров (общая сумма страховки – 100 000, выплаченная стразовая сумма = 0);
- с одним параметром – общая сумма страховки (выплаченная страховая сумма = 0).

Методы:

- геттеры и сеттеры полей.

3. Измените поля, конструкторы, методы класса **PatientsCard**, связанные с номером медицинского полиса:

- теперь этот класс должен работать не с числом (номер медицинского полиса), а с экземпляром класса **MedicalInsurancePolicy**;
- приватное поле, содержащее номер полиса, нужно удалить. Вместо него добавить поле типа **MedicalInsurancePolicy**;
- конструкторы, принимающие номер полиса, теперь принимают в качестве входного параметра ссылку типа **MedicalInsurancePolicy**;
- методы получения номер полиса изменить на методы получения ссылки на полис (тип **MedicalInsurancePolicy**).

4. Измените класс **Polyclinic**:

- измените реализацию метода, возвращающего ссылку на карту пациента по номеру страхового полиса;
- измените реализацию метода удаления карты пациента по номеру полиса (методы по-прежнему принимают номер страхового полиса, изменяется только реализация).

5. Переименуйте класс **PatientsCard** в **SocialPatientsCard**.

6. Создайте интерфейс **PatientsCard**, содержащий методы:

- получения имени;
- изменения имени;
- получения фамилии;
- изменения фамилии;
- получения адреса;
- изменения адреса;

- получения ссылки на медицинский полис;
- изменение ссылки на медицинский полис.

7. Класс **SocialPatientsCard** должен реализовывать интерфейс **PatientsCard**.

8. Создайте класс **Bill** – счет за лечение.

Поля:
- дата (экземпляр класса **java.util.Date**);
- сумма за оказанные в этот день медицинские услуги;
- вид медицинской услуги (перечисление, возможные значения этого перечисления –стоматология, эндокринология, хирургия, проф. осмотр, и т.п. (придумайте самостоятельно еще несколько вариантов).

Методы:
- геттеры и сеттеры к приватным полям.

9. Создайте класс **PaidPatientsCard** – карты человека, пользующегося платными услугами поликлиники. Класс должен реализовать интерфейс **PatientsCard** и, помимо методов и полей, необходимых для реализации этого интерфейса, он должен содержать следующие элементы.

Поля:
- приватный список (класс **ArrayList<Bill>**), содержащий информацию о том, сколько и когда пациент оставил денег в поликлинике (иными словами, список содержит экземпляры класса **Bill**).

Конструкторы:
- конструктор без параметров (инициирующий список нулевой длины);
- конструктор может принимать имя и фамилию (полис = null и адрес – пустой, инициируется список нулевой длины);
- конструктор может принимать имя, фамилию и адрес (полис = null, инициируется список нулевой длины);
- конструктор может принимать имя, фамилию, адрес и ссылку на экземпляр полиса (**MedicalInsurancePolicy**);
- конструктор, принимающий имя, фамилию, адрес, ссылку на экземпляр полиса и список счетов (экземпляр **ArrayList<Bill>**).

Методы:
- геттер и сеттер для списка счетов;
- метод, возвращающий общую сумму, заплаченную пациентом поликлинике;

- метод, возвращающий список счетов в заданный день (если счет только один, возвращает список из одного элемента, если счетов в заданный день не было, возвращает список из 0 элементов);
- метод, добавляющий счет в конец списка счетов;
- метод, удаляющий счет с соответствующей датой и размером платежа (передаются в качестве параметров метода).

10. Измените класс **Polyclinic**. Добавьте методы:
- метод, возвращающий общее число пациентов, обслуживаемых по полисам ОМС (обязательного медицинского страхования);
- метод, возвращающий общее число пациентов, обслуживаемых по полисам ДМС (добровольного медицинского страхования);
- метод, возвращающий общее число пациентов, хотя бы один раз пользовавшихся платными услугами поликлиники (не через ДМС);
- метод, возвращающий общую сумму перечислений (оплат по счету) в заданный месяц и год.

Вариант № 2

Пакет – university.

1. Создайте класс **Payment** – плата за обучение.

Поля:
- дата (экземпляр класса **java.util.Date**);
- размер суммы, переведенной студентом на счет университета.

Конструкторы:
- без параметров (дата содержит null, сумма – 0);
- с двумя параметрами – датой и суммой.

Методы:
- геттеры и сеттеры для приватных полей.

2. Создайте класс **ContractStudent**, расширяющий (наследует) класс **Student**. Этот класс добавляет следующие члены класса.

Поля:
- приватное поле – связный список платежей (класс **LinkedList<Payment>**);
- приватное поле – стоимость обучения за семестр (предполагается, что эта стоимость в течение всего периода обучения меняться не будет).

Конструкторы:

- конструктор без параметров (инициирующий список нулевой длины);
- конструктор может принимать имя и фамилию (номер зачетной книжки – 0, инициирующий список нулевой длины);
- конструктор, принимающий имя, фамилию, номер зачетной книжки (инициирующий список нулевой длины);
- конструктор, принимающий имя, фамилию, номер зачетной книжки, список платежей (экземпляр класса **LinkedList<Payment>**) .

Методы:

- геттер и сеттер для списка платежей и стоимости обучения;
- метод, возвращающий размер задолженности студента на текущий момент (метод не принимает параметров). Исходите из предположения, что плата за обучение не меняется. Зная текущую дату и год поступления, можно выяснить, сколько семестров студент проучился, и определить сумму, которую он должен был внести за весь срок обучения;
- метод, добавляющий платеж в конец списка платежей;
- метод, удаляющий платеж с соответствующей датой и размером платежа.

3. Создайте интерфейс **Event** – мероприятия, в котором участвовал студент, описывающий методы:

- геттеры и сеттеры для даты проведения мероприятия (работают с экземплярами класса **java.util.Date**);
- геттеры и сеттеры для названия города, в котором проводилось мероприятие.

4. Определите класс **Olympics**, реализующий интерфейс **Event**.

Поля:

- дата проведения олимпиады;
- название города, где проходила олимпиада;
- место (целое число), которое занял студент на олимпиаде.

Методы:

- методы доступа (геттеры и сеттеры) для приватных полей.

5. Определите класс **Conference**, реализующий интерфейс **Event**.

Поля:

- дата выступления студента на конференции;
- город, где проходила конференция;

- название доклада (статьи), с которым (которой) студент выступал на конференции.

Методы:
- методы доступа (геттеры и сеттеры) для приватных полей.

6. Определите класс **Competition**, реализующий интерфейс **Event**.

Поля:
- дата соревнования;
- город, где проходило соревнование;
- название проекта;
- выигранная сумма (0 – если нет).

Методы:
- методы доступа (геттеры и сеттеры) для приватных полей.

7. Определите интерфейс **Activist** – участника различных конкурсов, олимпиад и т.п. Интерфейс определяет следующие методы:
- метод, возвращающий общее количество мероприятий, в которых участвовал студент;
- метод, возвращающий число призовых мест, занятых на олимпиадах;
- метод, возвращающий число докладов на конференциях;
- метод, возвращающий строку, состоящую из названий проектов (разделенных переходом на новую строку), за которые студент получил вознаграждение на соревнованиях.

8. Измените класс **Student**. Он должен реализовывать интерфейс **Activist.** Для реализации методов интерфейса, добавить следующие члены класса.

Поля:
- список событий (класс **ArrayList<Event>**) в которых участвовал студент.

Помимо методов, реализуемых в соответствии с интерфейсом **Activist,** добавить следующие **методы**:
- вставки информации о событии в конец списка событий;
- удаления события из списка по дате;
- поиска события по дате (метод возвращает ссылку на событие).

Конструкторы:
- внесите изменения в конструкторы класса, для того, чтобы они инициировали список событий.

9. Измените класс **Group**.

Добавьте **методы**:

- метод, возвращающий список студентов – активистов (которые хоть один раз участвовали в каком-то мероприятии);
- метод, возвращающий список «привилегированных» студентов – которые хоть раз занимали призовое место на олимпиаде или в конкурсе;
- метод, возвращающий число активистов в группе;
- метод, возвращающий число бюджетников в группе;
- метод, возвращающий число контрактников в группе;
- метод, возвращающий список должников (не оплативших вовремя счет по контракту).

Вариант № 3

Пакет – *organisation*.

1. Создайте перечисление **JobTitles** названий должностей, предусмотреть следующие должности:

- начальник подразделения (DepartmentBoss);
- инженер (Engineer);
- секретарь (Clerk);
- директор (BigBoss);
- придумайте еще 2-3 должности.

2. Класс **Employee** сделайте абстрактным. Добавьте следующие элементы.

Поля:

- дата приема на работу (экземпляр класса **Date**).

Конструкторы:

- конструктор, принимающий имя, фамилию, должность, жалование, дату приема.

Методы:

- гетер и сеттер даты приема;
- абстрактный public метод, возвращающий ежемесячную премию.

Измените:

- приватное поле, содержащее должность сотрудника, должно быть экземпляром перечисления **JobTitles**;
- измените конструкторы и методы, работающие с названием должности, в соответствии с тем, что поле теперь имеет тип перечисления.

3. Создайте класс **FullDayEmployee** штатного сотрудника, расширяющий (наследующий) класс **Employee**.

Методы:
- добавьте реализацию метода, возвращающего ежемесячную премию. Она вычисляется как число полных лет, которые проработал сотрудник в компании, деленное на 20. Кроме того, если зарплата начисляется в январе (то есть текущий месяц – январь), премия увеличивается на размер оклада (для определения текущей даты используется класс **Calendar**).

Конструкторы:
- Добавьте такие же конструкторы, что и в классе **Employee**.

4. Создайте класс **HalfDayEmployee** – внешнего совместителя, расширяющий (наследующий) класс **Employee**.

Методы:
- добавьте реализацию метода, возвращающего ежемесячную премию. Этот метод возвращает 0.

Конструкторы:
- добавьте такие же конструкторы, что и в классе **Employee**.

5. Создайте класс **BusinessTravel** – командировки. Этот класс содержит следующие члены класса.

Поля:
- дата отбытия с предприятия в командировку;
- дата прибытия;
- стоимость трансфера до места и назад;
- суточные.

Конструкторы:
- конструктор по умолчанию, не инициирующий поля (пустой конструктор);
- конструктор с параметрами: дата отбытия с предприятия в командировку, дата прибытия, стоимость трансфера до места и назад, суточные.

Методы:
- гетеры и сеттеры полей;
- метод, возвращающий число полных дней между датами отбытия и прибытия;

- метод, возвращающий общую сумму затраченных на командировку денег (трансфер + суточные * кол-во дней).

6. Создайте интерфейс **BusinessTraveller** – работника, направляемого в командировку. Этот интерфейс описывает следующие методы:

- метод добавления информации о командировке;
- метод удаления информации о командировке (принимает параметр дату отбытия);
- метод, возвращающий (ссылку на) экземпляр класса **BusinessTravel** по дате (если введенная дата попадает в интервал между началом и концом командировки);
- метод, возвращающий среднюю продолжительность командировок работника;
- метод, возвращающий средний интервал между командировками в днях.

7. Класс **FullDayEmployee** должен реализовывать интерфейс **BusinessTraveller**. Для этого определите.

Поля:

- поле типа **ArrayList< BusinessTravel>** – этот список будет содержать всю информацию о командировках сотрудника. При создании экземпляра сотрудника, в конструкторах, это поле инициируется списком нулевой длины. На основе этого поля реализуйте методы, описываемые в интерфейсе.

8. Измените класс **Department**.

Добавьте **методы**:

- метод, возвращающий список (**ArrayList<FullDayEmployee>**) штатных сотрудников;
- метод, возвращающий список (**ArrayList<HalfDayEmployee>**) внешних совместителей;
- метод, возвращающий список (**ArrayList<BusinessTraveller>**) сотрудников, находящихся в командировке в данное время;
- метод, возвращающий список (**ArrayList<BusinessTraveller>**) сотрудников, находящихся в командировке указанного числа (принимается в качестве параметра метода).