Контрольное задание по предмету «Объектно-ориентированное программирование на языке Java» 1 Семестр

Описание

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

Содержание

1. Выбор варианта
2. Содержание отчета
3. Задание 1
4. Задание 2
5. Задание 3

1 Выбор варианта

Вариант выбирается в соответствии с номером зачетной книжки. Номер варианта определяется как последняя цифра зачетной книжки, деленая на 2. Результат округляется до большего целого. Например, если последняя цифра зачетной книжки 5, то номер варианта – 3.

2 Содержание отчета

Отчет предоставляется в печатной форме. Содержит:

1) Титульный лист
2) Задание на выполнение
3) Окончательный вариант исходного кода (после решения последней части контрольной работы) на языке Java.
4) Исходный код класса(ов) с использование которого(ых) производилось тестирование конструкторов и методов реализованных классов.

3 Задание 1

1. Создайте пакет. Имя создаваемого пакета должно соответствовать варианту.
2. **Для каждой используемой переменной или константы, атрибута, метода, класса выбирать имена в соответствии с конвенциями именования.**
3. Реализуйте классы в соответствии с вариантом.
4. Определите поля класса как приватные (доступные только внутри класса), а методы и конструкторы – публичными (доступные всем другим классам).
5. 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.
6. Создайте отдельный класс для тестирования конструкторов и вызовов методов созданных в соответствии с вариантом классов.
7. Если написано массив – значит массив, а не ArrayList.

Вариант № 0

1. Пакет – policlinic.
2. Создайте публичный класс OutpatientsCard – карты человека, зарегистрированного в поликлинике. Класс не хранит в явном виде информацию о поликлинике.
- каждый пациент характеризуется именем, фамилией, адресом проживания;
- каждый пациент имеет страховой полис, и характеризуется его номером;
- 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;
- 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;
3. Создайте публичный класс Policlinic – поликлиники некоторого района.
- поликлиника характеризуется номером и адресом;
- класс хранит явным образом массив карт пациентов, зарегистрированных в поликлинике;
- конструктор может принимать номер и адрес поликлиники (длина массива пациентов = 0)
- конструктор может принимать номер и адрес, а так же массив пациентов;
- 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 OutpatientsCard, расширяет массив карт путем добавления нового элемента в конец массива);
- создайте метод, возвращающий массив карт;
- создайте метод, возвращающий массив карт, отсортированный по адресам;

Вариант № 1

1. Пакет – university.
2. Создайте публичный класс Student – студента некоторой специальности некоторого университета. Класс не хранит явным образом информацию о специальности, номере группы\потока, предметах, университете.
- каждый студент характеризуется именем, фамилией, годом поступления, уникальным 6-тизначным номером зачетной книжки.
- конструктор может принимать имя и фамилию. При этом номер зачетной книжки - 0;
- конструктор, принимающий имя, фамилию, номер зачетной книжки;
- 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;
3. Создайте публичный класс Group – студенческой группы.
Класс не хранит явно специальность и имя университета.
- каждая группа имеет свой номер (уникальный, в пределах специальности)
- класс хранит явным образом массив студентов;
- 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;
- 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 № 2

1. Пакет – *organization*.
2. Создайте публичный класс *Employee* – работника некоторой организации:
Класс не хранит явным образом номер или имя подразделения и организации, в которой работает работник.
- каждый работник занимает определенную должность;
- каждый работник получает определенное жалование;
- каждый работник характеризуется именем и фамилией;
- конструктор может принимать имя и фамилию (должность – инженер, жалование – 30к руб.);
- конструктор может принимать имя, фамилию, должность, жалование;
- 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.
3. Создайте публичный класс *Department* – подразделения некоторой организации.
Класс не хранит явным образом номер подразделения и имя организации, частью которой является.
- разные подразделения имеют разные имена;
- класс хранит явным образом массив своих работников;
- конструктор может принимать имя подразделения (в этом случае количество работников = 0);
- конструктор может принимать массив работников;
- 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);

Вариант № 3

1. Пакет – logistics.
2. Создайте публичный класс Truck – грузового автомобиля. Класс не хранит явным образом информацию о грузе, водителе, маршруте.
- каждый грузовой автомобиль характеризуется гос. регистрационным номером, маркой, грузоподъемностью (т), объемом кузова (куб. м).

- конструктор может принимать гос. рег. номер (остальные поля принимают значения по-умолчанию некоторой модели грузового автомобиля, выбираемой студентом)
- конструктор может принимать гос. регистрационный номер, марку, грузоподъемность (т), объемом кузова (куб. м)
- создайте метод получения гос. рег. номера
- создайте метод изменения гос. рег. номера
- создайте метод получения марки
- создайте метод изменения марки
- создайте метод получения грузоподъемности
- создайте метод изменения грузоподъемности
- создайте метод получения объема кузова
- создайте метод изменения объема кузова

3. Создайте публичный класс TruckFleet – парка автомобилей некоторой логистической организации. Класс не хранит явным образом информацию о грузах, водителях, маршрутах.
- класс явным образом хранит массив грузовых автомобилей;
- конструктор может принимать число грузовых автомобилей (в этом случае инициализируется соответствующий массив, но сами элементы не инициализируются);
- конструктор может принимать массив грузовых авто;
- создайте метод, возвращающий общее число грузовиков;
- создайте метод, возвращающий ссылку на грузовик по его гос. рег. номеру;
- создайте метод, возвращающий ссылку на массив грузовиков, меньше заданной грузоподъемности;
- создайте метод, возвращающий ссылку на массив грузовиков, меньше заданного объема кузова;
- создайте метод удаления грузовика (принимает в качестве входного параметра гос. рег. номер, удаляет соответствующий этим данным элемент из массива грузовиков);
- создайте метод добавления грузовика (принимает в качестве входного параметра ссылку на экземпляр класса Truck, расширяет массив путем добавления нового элемента в конец массива);
- создайте метод, возвращающий массив всех авто;
- создайте метод, возвращающий массив грузовиков, отсортированный по грузоподъемности;

Вариант № 4

1. Пакет – text.
2. Создайте публичный класс Paragraph – абзаца текстового документа;
Класс не хранит явным образом местоположение в тексте, число строк.
- каждый абзац характеризуется строкой, непосредственно содержащей весь текст абзаца;
- каждый абзац характеризуется отступом красной строки (число символов, а **не** сантиметры);
- конструктор по-умолчанию (без параметров) создает «пустой» абзац – характеризующийся пустой строкой, и отступом = 0;
- конструктор может принимать значение отступа (в этом случае строка – пустая);
- конструктор может принимать значение отступа и строку – текст.
- создайте метод получения строки текста;
- создайте метод изменения строки текста;

- создайте метод получения отступа красной троки;
- создайте метод изменения отступа красной троки;
3. Создайте публичный класс Text – текста.
- класс характеризуется максимальным числом символов в строке;
- класс явным образом хранит в себе массив абзацев;
- конструктор по-умолчанию (длинна массива абзацев = 0, число символов 80);
- конструктор может принимать массив абзацев (число символов в строке = 80);
- конструктор может принимать массив абзацев и число символов в строке;
- создайте метод, возвращающий общее число абзацев;
- создайте метод, возвращающий общее число строк текста;
- создайте метод, возвращающий ссылку на абзац по его номеру (номер абзаца = номеру в массиве);
- создайте метод, вставляющий абзац после абзаца (принимает ссылку на новый абзац и номер абзаца, после которого нужно вставить новый)
- создайте метод удаления абзаца по его номеру;
- создайте метод изменения абзаца по его номеру (принимает ссылку на новый абзац и номер абзаца, который нужно заменить новым);
- создайте метод, возвращающий массив абзацев;

Вариант № 5

1. Пакет – bank.
2. Создайте публичный класс Account – счета банка.
Класс не содержит в явном виде информации о его «владельце»
- каждый счет характеризуется своим уникальным номером, остатком средств на счете (в рублях);
- конструктор может принимать номер счета (остаток = 0);
- конструктор может принимать номер счета, остатком средств на счете
- создайте метод получения номера счета
- создайте метод изменения номера счета
- создайте метод получения остатка средств на счете
- создайте метод изменения остатка средств на счете
3. Создайте публичный класс Client – клиента банка.
- клиент характеризуется именем, фамилией, серией и номером паспорта;
- класс явным образом хранит массив счетов;
- конструктор может принимать имя, фамилию, паспортные данные (размер массива счетов = 0);
- конструктор может принимать имя, фамилию, паспортные данные и массив счетов;
- создайте метод, возвращающий ссылку на счет по его уникальному номеру;
- создайте метод, возвращающий массив всех счетов;
- создайте метод, возвращающий суммарный остаток на всех счетах;
- создайте метод, возвращающий массив счетов с положительным остатком на счете;
- создайте метод удаления счета по его номеру;
- создайте метод добавления счета (принимает в качестве входного параметра ссылку на счет, расширяет массив счетов путем добавления нового счета в конец массива):
- создайте метод уменьшения размера остатка счета (принимает ссылку на счет и размер суммы);
- создайте метод увеличения размера остатка счета (принимает ссылку на счет и размер суммы);

4 Задание 2

1. В уже созданном пакете реализуйте (измените) классы и интерфейсы в соответствии с вариантом.
2. По-прежнему, поля класса – приватные (доступные только внутри класса).
3. 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.
4. Создайте отдельный класс для тестирования конструкторов и вызовов методов созданных в соответствии с вариантом классов.
5. Поля – даты, должны иметь тип *java.util.Date*.
6. При переименовании чего-либо следует пользоваться средствами рефакторинга среды разработки.

Вариант № 0

1. Пакет – policlinic.
2. Создайте абстрактный класс *MedicalInsurancePolicy* – полиса мед. страхования. Класс содержит:
приватные 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
3. Создайте класс *ObligatoryMedicalInsurancePolicy* – полиса обязательного медицинского страхования. Этот класс расширяет (наследует) класс *MedicalInsurancePolicy*. Дополнительных элементов этот класс не предлагает.
4. Создайте класс *VoluntaryMedicalInsurancePolicy* – полиса добровольного медицинского страхования. Этот класс также расширяет (наследует) класс *MedicalInsurancePolicy*. Этот класс определяет дополнительные элементы:
приватные 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
5. Измените поля, конструкторы, методы класса *OutpatientsCard*, связанные с номером мед. полиса. Теперь этот класс должен работать не с числом (номер мед. 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*)
6. Измените класс *Policlinic*.
- измените **реализацию** метода, возвращающего ссылку на карту пациента по номеру страхового полиса, а также метода удаления карты пациента по номеру полиса. **Методы по-прежнему принимают номер страхового полиса, изменяется только реализация**.
7. Переименуйте класс *OutpatientsCard* в *SocialOutpatientsCard*.
8. Создайте интерфейс *OutpatientsCard*, содержащий методы
- получения имени;
- изменения имени;
- получения фамилии;
- изменения фамилии;
- получения адреса;
- изменения адреса;
- получения ссылки на мед. полис
- изменение ссылки на мед. полис
9. Класс *SocialOutpatientsCard* должен реализовывать интерфейс *OutpatientsCard*.
10. Создайте класс *Bill* – счет за лечение, содержащий 3 приватных 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),
а также открытые геттеры и сеттеры к ним.
11. Создайте класс *PaidOutpatientsCard* – карты человека, пользующегося платными услугами поликлиники. Класс должен реализовать интерфейс *OutpatientsCard* и, помимо методов и полей, необходимых для реализации этого интерфейса, он должен содержать следующие элементы:
- приватный список (класс *ArrayList<Bill>*), содержащий информацию о том сколько и когда пациент оставил денег в поликлинике (иными словами, список содержит экземпляры класса *Bill*)
- 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>*)
- 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)
12. Измените класс *Policlinic*. Добавьте методы:
- метод, возвращающий общее число пациентов, обслуживаемых по полисам ОМС

- метод, возвращающий общее число пациентов, обслуживаемых по полисам ДМС
- 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 № 1

1. Пакет – *university*.
2. Создайте класс *Payment* – плата за обучение, содержащий:
приватные 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
3. Создайте класс *ContractStudent*, расширяющий (наследует) класс *Student*. Этот класс добавляет:
- приватное поле – связный список платежей (класс *LinkedList*<*Payment*>)
- приватное поле – стоимость обучения за семестр (в задаче предполагается что эта стоимость в течение всего периода обучения меняться не будет)
- 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*>)
- геттер и сеттер для списка платежей и стоимости обучения
- метод, возвращающий размер задолженности студента на текущий момент (метод не принимает параметров. Исходим 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
4. 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
5. Определите класс *Olympiad*, реализующий интерфейс *Event*.
Содержит приватные поля – дата и название города, а так же место (целое число), которое занял студент на олимпиаде.
Реализовать методы доступа (геттеры и сеттеры) для приватных полей.
6. Определите класс *Conference*, реализующий интерфейс *Event*.
Содержит приватные поля – дата и название города, а так же название доклада (статьи), с которым (которой) студент выступал на конференции.
Реализовать методы доступа (геттеры и сеттеры) для приватных полей.

7. Определите класс *Competition*, реализующий интерфейс *Event*.
Содержит приватные поля – дата и название города, а так же название проекта и выигранная сумма (0 – если нет =)))
Реализовать методы доступа (геттеры и сеттеры) для приватных полей.
6. 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
7. Измените класс *Student*. Он должен реализовывать интерфейс *Activist*. Для реализации методов интерфейса, добавить приватный поле – 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)
Кроме того, внесите изменения в конструкторы класса, для того, чтобы они инициировали список событий.
8. 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 во-время счет по контракту)

Вариант № 2

1. Пакет – *organization*.
2. Создайте перечисление *JobTitles* названий должностей, предусмотреть следующие должности: начальник подразделения (DepartmentBoss), инженер (Engineer), секретарь (Clerk), директор (BigBoss). Можете добавить еще должностей по-желанию.
3. Класс *Employee* сделайте абстрактным. Добавьте следующие элементы:
приватные поля:
- дата приема на работу (экземпляр класса *Date*);
конструкторы:
- конструктор, принимающий имя, фамилию, должность, жалование, дату приема;
методы:
- гетер и сеттер даты приема;
- абстрактный открытый 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.

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

- Добавьте реализацию метода, возвращающего ежемесячную премию. Она вычисляется как число полных лет, которые проработал сотрудник в компании, деленое 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*).
- Добавьте такие же конструкторы, что и в классе *Employee*

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

- Добавьте реализацию метода, возвращающего ежемесячную премию. Этот метод возвращает 0.
- Добавьте такие же конструкторы, что и в классе *Employee*.

6. Создайте класс *BusinessTravel* – командировки. Этот класс содержит:

Приватные 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:
- открытые 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)

7. 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.

8. Класс *FullDayEmployee* должен реализовывать интерфейс *BusinessTraveller*. Для этого определите приватное 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.

9. Измените класс *Department*.

Добавьте методы:
- открытый метод, возвращающий список (*ArrayList*<*FullDayEmployee*>) штатных сотрудников;
- открытый метод, возвращающий список (*ArrayList*<*HalfDayEmployee*>) внешних совместителей;
- открытый метод, возвращающий список (*ArrayList*<*BusinessTraveller* >) сотрудников, находящихся в командировке а данное время;

- открытый метод, возвращающий список (*ArrayList<BusinessTraveller* >) сотрудников, находящихся в командировке указанного числа (принимается в качестве параметра метода);

Вариант № 3

1. Пакет – logistics.
2. Создайте абстрактный класс ContainerCargo – груза, находящегося в контейнере. Этот класс содержит:
- приватное поле, содержащее вес контейнера с грузом (weight) в килограммах
конструкторы:
- без параметров
- принимающий один параметр – вес
методы:
- открытые методы получения и изменения веса
- открытый абстрактный метод, возвращающий объем контейнера
- открытый абстрактный метод, возвращающий объем контейнера в заданных единицах измерения (единица измерения передается как входной параметр метода, и имеет тип VolumeUnitEnumeration)
3. Создайте перечисление VolumeUnitEnumeration – единиц измерения объема. Возможные значения перечисления: CubicMetre, Littre, CubicCentimetre.
4. Создайте класс BoxedCargo – груза, помещенного в контейнер прямоугольной формы. Этот класс расширяет (наследует) класс ContainerCargo. Он содержит следующие элементы:
приватные поля:
- высота
- ширина
- длина
(все три поля хранят значения в метрах)
конструкторы:
- конструктор без параметров,
- конструктор, принимающий параметры: вес, высота, ширина, длина.
открытые методы:
- гетеры и сеттеры полей: высота, ширина, длина
- метод, возвращающий объем контейнера
- метод, возвращающий объем контейнера в заданных единицах измерения (единица измерения передается как входной параметр метода, и имеет тип VolumeUnitEnumeration)
5. Создайте класс TankedCargo – жидкого груза в контейнере цилиндрической формы. Этот класс расширяет (наследует) класс ContainerCargo. Он содержит следующие элементы:
приватные поля:
- высота
- радиус
(оба поля хранят значения в метрах)
конструкторы:
- конструктор без параметров,
- конструктор, принимающий параметры: вес, высота, радиус.
открытые методы:
- гетеры и сеттеры полей: высота, радиус
- метод, возвращающий объем контейнера

- метод, возвращающий объем контейнера в заданных единицах измерения (единица измерения передается как входной параметр метода, и имеет тип VolumeUnitEnumeration)
6. Создайте интерфейс CargoTransport, описывающий следующие методы:
- метод получения рег. номера (строка)
- метод изменения рег. номера
- метод получения марки (строка)
- метод изменения марки
- метод получения грузоподъемности (вещественное число)
- метод изменения грузоподъемности
- метод получения максимально возможного суммарного объема перевозимого груза (вещественное число)
- метод изменения максимально возможного суммарного объема перевозимого груза
- метод, возвращающий массив контейнеров, перевозимых транспортом
- метод, принимающий массив контейнеров, перевозимых транспортом
- метод, добавляющий контейнер к общему грузу (метод принимает ссылку на добавляемый контейнер)
- метод, удаляющий контейнер (метод принимает ссылку на удаляемый контейнер)
- метод, возвращающий суммарный объем контейнеров, перевозимых транспортом, в заданных единицах измерения (единица измерения передается как входной параметр метода, и имеет тип VolumeUnitEnumeration)
- метод, возвращающий ссылку на наиболее тяжелый контейнер
7. Создайте класс CargoShip грузового судна. Класс должен реализовывать интерфейс CargoTransport. Этот класс содержит:
приватные поля:
- рег. номер
- марка
- максимальная грузоподъемность
- максимальный объем перевозимого груза
- список перевозимых грузов (экземпляр класса LinkedList<ContainerCargo>)
конструкторы:
- конструктор может принимать рег. номер (остальные поля принимают значения по-умолчанию, выбираемые студентом, список инициализируется пустым – число элементов 0)
- конструктор может принимать регистрационный номер, марку, грузоподъемность (т), объем (куб. м) (список инициализируется пустым – число элементов 0)
методы:
-методы, реализующие интерфейс CargoTransport
8. Измените класс Truck – он должен реализовать интерфейс CargoTransport. Замените массив на класс ArrayList<ContainerCargo>. Добавьте соответствующие реализации методов.
9. Измените класс TruckFleet
- переименуйте его в CargoDeliveryBase
- он должен работать с элементами типа CargoTransport (и может содержать объекты как типа CargoShip, так и Truck).

Вариант № 4

1. Пакет – library.

2. Создайте класс Printing – некоторого печатного издания. Класс содержит следующие элементы:
- приватные поля, содержащие название и год издания
- 2 конструктора: по-умолчанию и с параметрами (название, год)
- гетеры и сеттеры названия и года издания
- метод получения «возраста» печатного издания (для получения информации о текущей дате использовать класса java.util.Calendar)
3 Создайте перечисление GenreEnumeration – литературного жанра. Возможные значения: постапокалиптика, киберпанк, фэнтези… (жанры можно подсмотреть здесь). Жанры описываемые перечислением выбираются студентом самостоятельно.
4. Создайте класс Book. Этот класс расширяет (наследует) класс Printing. Помимо названия и года издания каждая книжка характеризуется Фамилией автора и жанром. Класс добавляет следующие элементы:
- приватные поля, содержащие фамилию автора и жанр (экземпляр перечисления GenreEnumeration)
- конструкторы: без параметров и с параметрами (фамилия, название, год издания, жанр)
- гетеры и сеттеры полей
5. Создайте класс Article – статьи в журнале (сборнике статей). Этот класс характеризуется фамилией автора и названием.
Приватные поля:
- фамилия автора
- название статьи
конструкторы:
- по-умолчанию
- с 2-мя параметрами (автор, название)
методы:
- гетеры и сеттеры для полей
6. Создайте класс ScienceDigest – сборника статей. Этот класс расширяет (наследует) класс Printing. Помимо названия сборника и года издания, сборник характеризуется списком статей (экземпляр класса ArrayList<Article>. Элементы класса:
- приватное поле – список статей (класс ArrayList<Article>)
- конструктор по-умолчанию и с параметрами (название сборника, год, список статей)
методы:
- гетер и сеттер для списка статей
- методы добавления статьи в конец списка (принимают в качестве параметра ссылку на статью)
- метод удаления статьи из списка (метод принимает в качестве параметра название статьи)
- метод поиска статьи по имени автора. Возвращает ссылку на список статей ArrayList<Article> указанного автора.
- метод поиска статьи по названию. Возвращает ссылку на статью.
7. Создайте класс FreeLibraryCard – читательского билета. Карта содержит информацию о читателе – имя, фамилия, список печатной продукции, которые в данным момент у него на руках (он их читает – кэп). Класс содержит следующие элементы:
Приватные поля:
- фамилия читателя,
- имя читателя,

- список изданий onHandList (экземпляр класса LinkedList<Printing>), которые читатель еще читает (не вернул)
Конструкторы:
- по-умолчанию (инициирует пустой список)
- с параметрами: фамилия, имя, список
открытые методы:
- сеттеры и гетеры полей
- добавления издания в список onHandList (методы принимают ссылку на объект типа Printing)
- удаления издания из списка onHandList (метод принимает название издания)
8. Создайте интерфейс LibraryCard – читательского билета. Интерфейс описывает методы:
- гетеры и сеттер имени и фамилии читателя
- добавления издания в список onHandList (методы принимают ссылку на объект типа Printing)
- удаления издания из списка onHandList (метод принимает название издания)
9. Класс FreeLibraryCard должен реализовывать интерфейс LibraryCard
10. Создайте класс VipLibraryCard, реализующий интерфейс LibraryCard. Этот класс отличается от класса FreeLibraryCard тем, что он основан на двух списках типа ArrayList<Printing>, а не LinkedList<Printing>.
11. Создайте класс Library – библиотека. Библиотека содержит 2 реестра: 1) реестр печатных изданий и 2) реестр читательских билетов. Элементы класса Library:
Приватные поля:
- список печатной продукции (LinkedList<Printing>).
- список читательских билетов (ArrayList<LibraryCard>).
Конструкторы:
- по-умолчанию (создает списки нулевой длины)
- с параметрами (список печатной продукции, список читательских билетов);
Открытые методы:
- метод, реализующий поиск в реестре издание по параметрам (название, фамилия автора, год издания – ищет как книги, так и статьи)
- метод, реализующий поиск в реестре карточки читателя по параметрам (фамилия и имя читателя)

Вариант № 5

1. Пакет – bank.
2. Создайте класс *AccountNumberGenerator*. Этот класс содержит:
- статичную приватную переменную типа *int*, с начальным значением 0;
- открытый статичный метод *getNext*() – который увеличивает значение статичной переменной на 1 и возвращает ее новое значение;
- открытый статичный метод *getCurrent*(), который возвращает значение статичной переменной;
- открытый метод reset(), устанавливающий значение переменной в 0;
Этот класс нужно использовать для генерации номеров счетов.
3. Создайте перечисление Currency – валюты. Перечисление содержит элементы (валюты) – USD (доллар США), EUR (евро), JOY (Йена), TRY(Лира), AED (Дирхам), RUB (Рубль). Можете добавить еще валют.
4. Класс *Account* сделайте абстрактным. Добавьте следующие элементы:
поля:
- приватное поле, содержащее комиссию за обслуживание.

- приватное поле, содержащее валюту счета.
конструкторы:
- конструктор, принимающий 3 параметра – номер счета, остаток на счете, размер комиссии за обслуживание счета (валюта – рубль)
- конструктор, принимающий 4 параметра – номер счета, остаток на счете, размер комиссии за обслуживание счета, валюта
методы:
- гетер и сеттер комиссии за обслуживание
- гетер и сеттер валюты (**сеттер, помимо установки валюты, пересчитывает комиссию и остаток на счете**)
- метод, вычитающий комиссию из остатка
- метод списывания суммы со счета (принимает параметр – списываемая сумма)
- метод пополнения счета (принимает параметр – сумма, на которую увеличивается остаток)
Удалите метод:
- изменения значения остатка средств на счете (сеттер)
5. Создайте класс *DebitAccount*, дебитовой карты. Этот класс расширяет (наследует) класс *Account*.
Этот класс не добавляет свои методы и поля, и не переопределяет методы и поля суперкласса. Класс определяет аналогичные суперклассу конструкторы, в которых просто вызывает соответствующий конструктор суперкласса.
6. Создайте класс *CreditAccount*, кредитной карты. Этот класс расширяет (наследует) класс *Account*. Добавьте следующие элементы:
Поля:
- приватное поле – процентная ставка (годовая, в процентах)
- приватное поле – лимит по кредитной карте
- приватное поле – начисленные проценты
- приватное поле – начисленные комиссионные
Конструкторы:
- аналогичные суперклассу конструкторы, в которых просто вызывает соответствующий конструктор суперкласса, остальные переменные = 0;
- конструктор, принимающий параметры: номер счета, остаток на счете, размер комиссии за обслуживание счета, валюта, процентная ставка, лимит по карте (в заданной валюте), начисленные проценты и комиссионные = 0;
Методы:
- гетеры и сеттеры процентной ставки и лимита по карте.
- *гетеры* для начисленных процентов и комиссионных за обслуживание.
- метод начисления процентов (если остаток меньше лимита по карте - увеличивается сумма начисленных процентов на величину = «(лимит - остаток) * (процентная ставка / число дней в текущем году) / 100». Для определения числа дней в текущем году использовать класс *Calendar*.
Переопределить методы:
- метод, вычитающий комиссию из остатка – вместо уменьшения остатка на величину комиссии, метод должен увеличить «начисленные комиссионные» на величину комиссионных.
- метод пополнения счета – вместо просто увеличения остатка, средства сначала идут на погашения начисленных комиссионных, затем начисленных процентов, затем на пополнение остатка.
7. Создайте интерфейс *Client*. Интерфейс описывает следующие методы:
- метод, возвращающий ссылку на счет по его уникальному номеру;
- метод, возвращающий список (класс ArrayList<Account>) всех счетов;

- метод, возвращающий список (класс ArrayList<Account>) счетов дебетовых карт;
- метод, возвращающий список (класс ArrayList<Account>) счетов кредитных карт;
- метод, возвращающий суммарный остаток на всех дебетовых счетах;
- метод, возвращающий сумму долга клиента (сумма начисленных процентов и комиссионных по всем кредитным счетам, а также отрицательный остаток по картам)
- метод, возвращающий список (класс ArrayList<Account>) счетов с положительным остатком на счете;
- метод удаления счета по его номеру;
- метод добавления счета (принимает в качестве входного параметра ссылку на счет):
- метод списывания средств со счета (принимает номер счета и размер суммы);
- метод пополнения счета (принимает номер счета и размер суммы);

8. Измените класс *Client* из предыдущей лабораторной работы.

а) переименуйте класс в *NaturalClient*.

б) класс должен реализовать интерфейс *Client*. Для этого:
- вместо массива счетов используйте список счетов (класс ArrayList<Account>).
- измените конструкторы
- реализуйте методы интерфейса

в) поля – имя, фамилия, серия и номер паспорта, а так же соответствующие гетеры и сеттеры оставьте.

5 Задание 3

Вариант № 0

1. Создайте класс объявляемого исключения InsufficientFundsException. Это исключение должно выбрасываться при попытке изменить значение поля выплаченной страховой суммы на значение, превышающее общую сумму страховки.
2. Переопределите метод equals(Object obj) и hashCode() в классах PaidOutpatientsCard и SocialOutpatientsCard, а так же Bill. Метод equals(Object obj) должен возвращать истину, только если сравниваемый объект obj является экземпляром того же класса, и значения всех полей равны. Стандартная практика реализации метода hashCode():
    a. Взять некоторое достаточно большое простое число
    b. Выполнить операцию исключающее или (^) между этим числом и значениями всех полей класса.
    c. Если поле x – примитив (boolean, byte, short, int, char) – берется само значение.
    d. Если поле x типа long, float, double, то берется результат метода hasCode соответствующего класса обертки (т.е. Long.hashCode(x), Float.hashCode(x), Double.hashCode(x)).
    e. Если поле x ссылочного типа, берется результат вызова его метода x.hashCode().
3. Переопределите метод toString() во всех классах пакета. Метод должен возвращать строку, содержащую полное описание класса, со значениями всех его полей. При реализации метода использовать класс StringBuilder.

Вариант № 1

1. Переопределите метод equals(Object obj) и hashCode() в классах *Olympiad* и *Conference*, а так же *Competition*. Метод equals(Object obj) должен возвращать истину, только если сравниваемый объект obj является экземпляром того же класса, и значения всех полей равны. Стандартная практика реализации метода hashCode():
    a. Взять некоторое достаточно большое простое число
    b. Выполнить операцию исключающее или (^) между этим числом и значениями всех полей класса.
    c. Если поле x – примитив (boolean, byte, short, int, char) – берется само значение.
    d. Если поле x типа long, float, double, то берется результат метода hasCode соответствующего класса обертки (т.е. Long.hashCode(x), Float.hashCode(x), Double.hashCode(x)).
    e. Если поле x ссылочного типа, берется результат вызова его метода x.hashCode().
2. Переопределите метод toString() во всех классах пакета. Метод должен возвращать строку, содержащую полное описание класса, со значениями всех его полей. При реализации метода использовать класс StringBuilder.
3. Создайте класс объявляемого исключения DublicateEventException. Это исключение должно выбрасываться при попытке добавить в список событий уже существующее в нем событие (такого же типа, с такой же датой и остальными параметрами).

Вариант № 2

1. Создайте класс объявляемого исключения IncorrectDateException. Это исключение должно выбрасываться при попытке задать дату прибытия более раннюю, чем дата отбытия.
2. Переопределите метод equals(Object obj) и hashCode() в классах *FullDayEmployee* и *HalfDayEmployee*, а так же *BusinessTravel*. Метод equals(Object obj) должен возвращать истину, только если сравниваемый объект obj является экземпляром того же класса, и значения всех полей равны. Стандартная практика реализации метода hashCode():
    a. Взять некоторое достаточно большое простое число
    b. Выполнить операцию исключающее или (^) между этим числом и значениями всех полей класса.
    c. Если поле x – примитив (boolean, byte, short, int, char) – берется само значение.
    d. Если поле x типа long, float, double, то берется результат метода hasCode соответствующего класса обертки (т.е. Long.hashCode(x), Float.hashCode(x), Double.hashCode(x)).
    e. Если поле x ссылочного типа, берется результат вызова его метода x.hashCode().
3. Переопределите метод toString() во всех классах пакета. Метод должен возвращать строку, содержащую полное описание класса, со значениями всех его полей. При реализации метода использовать класс StringBuilder.

Вариант № 3

1. Создайте класс объявляемого исключения ThisIsNotABlachWholeException. Это исключение должно выбрасываться если при попытке добавить контейнер в список перевозимых контейнеров их суммарный объем или вес превысит максимально возможный перевозимый объем или грузоподъемность соответственно.
2. Переопределите метод equals(Object obj) и hashCode() в классах BoxedCargo и TankedCargo. Метод equals(Object obj) должен возвращать истину, только если сравниваемый объект obj является экземпляром того же класса, и значения всех полей равны. Стандартная практика реализации метода hashCode():
    a. Взять некоторое достаточно большое простое число
    b. Выполнить операцию исключающее или (^) между этим числом и значениями всех полей класса.
    c. Если поле x – примитив (boolean, byte, short, int, char) – берется само значение.
    d. Если поле x типа long, float, double, то берется результат метода hasCode соответствующего класса обертки (т.е. Long.hashCode(x), Float.hashCode(x), Double.hashCode(x)).
    e. Если поле x ссылочного типа, берется результат вызова его метода x.hashCode().
3. Переопределите метод toString() во всех классах пакета. Метод должен возвращать строку, содержащую полное описание класса, со значениями всех его полей. При реализации метода использовать класс StringBuilder.

Вариант № 4

1. Переопределите метод equals(Object obj) и hashCode() в классах Book, Article, ScienceDigest. Метод equals(Object obj) должен возвращать истину, только если сравниваемый объект obj является экземпляром того же класса, и значения всех полей равны. Стандартная практика реализации метода hashCode():
    a. Взять некоторое достаточно большое простое число
    b. Выполнить операцию исключающее или (^) между этим числом и значениями всех полей класса.
    c. Если поле x – примитив (boolean, byte, short, int, char) – берется само значение.
    d. Если поле x типа long, float, double, то берется результат метода hasCode соответствующего класса обертки (т.е. Long.hashCode(x), Float.hashCode(x), Double.hashCode(x)).
    e. Если поле x ссылочного типа, берется результат вызова его метода x.hashCode().
2. Переопределите метод toString() во всех классах пакета. Метод должен возвращать строку, содержащую полное описание класса, со значениями всех его полей. При реализации метода использовать класс StringBuilder.
4. Создайте класс объявляемого исключения DuplicateArticleException. Это исключение должно выбрасываться при попытке добавить в список статей (сборника статей) уже существующую статью (с такой же фамилией и названием).

Вариант № 5

1. Создайте класс объявляемого исключения InsufficientFundsException. Это исключение должно выбрасываться при попытке списать со счета сумму, превышающую остаток или лимит по кредитной карте.
2. Переопределите метод equals(Object obj) и hashCode() в классах *DebitAccount* и *CreditAccount*. Метод equals(Object obj) должен возвращать истину, только если сравниваемый объект obj является экземпляром того же класса, и значения всех полей равны. Стандартная практика реализации метода hashCode():
    a. Взять некоторое достаточно большое простое число
    b. Выполнить операцию исключающее или (^) между этим числом и значениями всех полей класса.
    c. Если поле x – примитив (boolean, byte, short, int, char) – берется само значение.
    d. Если поле x типа long, float, double, то берется результат метода hasCode соответствующего класса обертки (т.е. Long.hashCode(x), Float.hashCode(x), Double.hashCode(x)).
    e. Если поле x ссылочного типа, берется результат вызова его метода x.hashCode().
3. Переопределите метод toString() во всех классах пакета. Метод должен возвращать строку, содержащую полное описание класса, со значениями всех его полей. При реализации метода использовать класс StringBuilder.