Для всех задач необходимо дополнительно создавать класс Application, который будет содержать статический метод main.

**Задача 1**

Создать класс Person, содержащий поля, определяющие фамилию, имя. Реализовать методы equals и hashcode таким образом, чтобы они зависели от полей класса и соблюдался контракт между ними. Реализовать метод toString таким образом, чтобы он выводил информацию о человеке в формате

Фамилия: <Фамилия>, Имя: <Имя>

Конструктор класса Person должен генерировать фамилию и имя по паттерну Фамилия-X, Имя-Y, где X,Y- случайные числа от 1 до 20. В методе main сгенерировать список из 50 человек. Вывести на экран только информацию о тех людях, у которых совпадает и фамилия, и имя. Выводиться должны только уникальные записи, т.е. если в списке есть 3 человека с одинаковыми фамилиями и именем, выводиться должна только 1 запись.

После этого создать множество людей с реализацией HashSet и добавить в него 2000 случайно сгенерированных людей. После генерации вывести на экран число элементов в данном множестве.

**Задача 2**

Создать класс Person c единственным полем int birthDate. Поле принимает случайное значение от 1 до 365 включительно и определяет день рождения (день с начала года).

Проверить вероятность существования двух людей, имеющими день рождения в один день, выполнив 100 000 итераций для каждой группы людей. Группы состоят от 10 до 30 человек. Вывести на экран строки в формате:

<Количество человек в группе>: <Полученная вероятность совпадения дня рождений (в процентах, с округлением до 2 цифр после разделителя)>

|  |
| --- |
| Примерные результаты (могут незначительно отличаться): |
| 10: 11,69 %  11: 14,11 %  12: 16,72 %  13: 19,44 %  14: 22,32 %  15: 25,27 %  16: 28,38 %  17: 31,48 %  18: 34,68 %  19: 37,91 %  20: 41,16 %  21: 44,39 %  22: 47,54 %  23: 50,74 %  24: 53,84 %  25: 56,89 %  26: 59,85 %  27: 62,69 %  28: 65,45 %  29: 68,09 %  30: 70,63 % |

Убедиться, что выполняется парадокс дней рождений: «В группе, состоящей из 23 или более человек, вероятность совпадения дней рождения хотя бы у двух людей превышает 50%.»

**Задача 3**

Необходимо реализовать логику работы терминалов для пополнения баланса телефона.

Каждый терминал должен обладать следующими свойствами:

* Адрес терминала
* Фирма-владелец терминала (использовать enum)
* Комиссия за операцию
* Сумма, свыше которой комиссии нет

Каждый терминал должен иметь метод пополнения баланса, в который передаются 2 параметра: номер телефона и сумма пополнения.

Для метода пополнения баланса использовать интерфейс TopUpBalance. Для описания терминала использовать абстрактный класс Terminal, реализующий интерфейс TopUpBalance. Реализация метода в абстрактном классе должна выводить на экран сообщение в формате:

Внесена сумма <Сумма> для пополнения телефона <Телефон>. Баланс пополнен на сумму <Сумма после комиссии>. С уважением, ваш <Фирма-владелец терминала>, <Адрес терминала>

Суммы округлить до копеек.

С помощью принципа полиморфизма, реализовать для терминала АО «Тинькофф» минимальную сумму для пополнения 100. В случае пополнения на сумму ниже, терминал должен выдавать соответствующее сообщение (формат произвольный)

Реализовать классы согласно таблице:

|  |  |  |  |
| --- | --- | --- | --- |
|  | QiwiTerminal | EleksnetTerminal | TinkoffTerminal |
| Фирма-владелец | ООО «Киви» | ООО «Элекснет» | АО «Тинькофф» |
| Комиссия за операцию | 7% | 11% | 9% |
| Сумма без комиссии | 1000 | 300 | 600 |

Создать объекты терминалов, согласно таблице:

|  |  |
| --- | --- |
| Фирма-владелец | Адрес |
| ООО «Киви» | г. Москва, ул. Проходчиков, д.3 |
| ООО «Элекснет» | г. Москва, Измайловский пр-т, 73/2 |
| ООО «Элекснет» | г. Москва, ул. Бакунинская, д.1 |
| ООО «Элекснет» | г. Новосибирск, ул. Героев-Десантников, д. 4 |
| АО «Тинькофф» | г. Москва, Ленинградский пр-т, д. 15 |

В методе main протестировать пополнения в различных терминалах на различные суммы.

**Задача 4**

В системе дистанционного банковского обслуживания (ДБО) используются сущности Client (организация-клиент), User (пользователь организации), Branch (подразделение банка), Account (счет). Сущности имеют следующие поля:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Сущность | Client | User | Branch | Account |
| Общие поля | String uuid;  String name; | String uuid;  String name; | String uuid;  String name; | String uuid;  String name; |
| Уникальные поля | String address; | String jobTitle | String address; | String accountNumber |

Необходимо создать общий базовый класс Entity для данных сущностей, в который вынести общие поля. Создать конструктор, принимающий параметр String name и устанавливающий его значение объекту. Поле id должно устанавливаться значением null;

Создать интерфейс Validatable, содержащий метод void validate().

Создать класс ValidatorStub с методом validateEntity(Entity entity), который будет валидировать передаваемую ему сущность. В классе ValidatorStub создать множество объектов типа Entity. Валидация сущностей должна происходить следующим образом:

* Если данная сущность уже существует в множестве объектов, тогда этой сущности необходимо присвоить uuid сущности из множества. При проверке на существовании необходимо проверять как общее поле name, так и уникальные поля сущности.
* Если данной сущности нет в множестве объектов, необходимо добавить в множество объектов данную сущность. При этом данной сущности присваивается случайный uuid.

При валидации выводить на экран информацию о валидируемом объекте на экран до и после валидации

В абстрактном классе Entity реализовать метод validate(), передавая объект этого типа в ValidatorStub.validateEntity(this).

В методе main протестировать выдачу уникальных uuid различным сущностям и одного и того же uuid идентичным сущностям.