1. ~~описать базовый класс «Товар», от которого будут наследоваться конкретные типы товаров (стиральная машина, чипсы, корвалол и т.д. (можно придумать свои). У каждого товара есть название, вес, цена, дата доставки и свой уникальный параметр (для стиральной машины – это наличие сушилки). Для наследников переопределить метод ToString(). Необходимо собрать заказ из товаров – найти общий вес, общую стоимость, а также вывести на консоль информацию о всех товарах в заказе. Дополнительно вывести список в алфавитном порядке без использования LINQ. Логика работы:~~
   1. пользователь вводит параметры заказа (Например, самые дешёвые товары: 1 стиральная машина (самая дешёвая), 1 чипсы (самые дешёвые), 1 Корвалол (самый дешёвый));
   2. расчёт оптимального заказа;
   3. вывод пользователю оптимального заказа: общие параметры и содержащиеся в нём товары с указанием количества;
2. ~~реализовать хранение в файле данных о доступных товарах;~~
3. реализовать подсчёт через калькулятора из предыдущего задания:
   1. создать класс, наследующий Calculator;
   2. сделать перегрузки методов Add, Substract, Multiply, Divide:
      1. товар + заказ = заказ с новым товаром;
      2. заказ + товар = заказ с новым товаром;
      3. заказ – заказ = первый заказ, в котором нет товаров из второго заказа. Вывести исключение, если во втором заказе есть то, чего нет в первом;
      4. заказ / товар = заказ, в котором убрали все товары данного типа;
      5. заказ / число = заказ, в котором количество каждого типа товара было уменьшено в определённое число раз;
      6. заказ \* число = заказ, в котором количество каждого типа товара было увеличено в определённое число раз;
   3. если логгер не null, то выводить в консоль информацию о методах при их вызове. Желательно также перегрузить операторы для данных типов.
4. написать генератор тестовых данных для заказа и метод редактирования товара:
   1. множество расширений метода GenerateOrder() – истинно случайный набор товаров;
   2. GenerateOrderBySum(double maxSum) – случайный набор товаров, чья сумма меньше, чем maxSum);
   3. GenerateOrderBySum(double minSum, double maxSum) – случайный набор товаров, чья сумма больше minSum, но меньше, чем maxSum;
   4. GenerateOrderByCount(double maxCount) – случайный набор товаров, общее количество которых = maxCount;
5. с помощью LINQ вывести заказы:
   1. дороже/дешевле указанной цены;
   2. имею в составе товары определённого типа;
   3. отсортированы по весу;
   4. с уникальными названиями;
   5. отправлены до указанной даты;
6. написать для калькулятора заказов модульные тесты. Требования:
   1. тесты должны храниться в отдельном проекте (в том же решении);
   2. отдельный тест на каждый тип операции калькулятора;
   3. реализовать тест на подсчёт выражения на основании заранее подготовленного файла JSON;
   4. реализовать тест на исключительные случаи (деление заказа на ноль, вычитание большего заказа из меньшего, умножение с выходом за пределы int). Добавить обработку таких случаем через try-catch;
   5. тест(ы) на корректное редактирование товара (название не может быть пустым, цена и вес не могут быть отрицательными);
7. через консоль требуется запрос у пользователя путь к файлу с описанием одного заказа (набор товаров) в формате JSON. Необходимо пройтись целиком по этому набору и поместить каждый товар в словарь типа <тип товара, int>. Затем вывести строковое представление этого массива в консоль построчно.