Что такое ООП? Кто-нибудь знает?

Объектно-ориентированное программирование (ООП) — это подход, при котором программа рассматривается как набор объектов, взаимодействующих друг с другом. У каждого есть свойства и поведение. ООП ускоряет написание кода и делает его более читаемым.

Идеология объектно-ориентированного программирования (ООП) разрабатывалась, чтобы связать поведение определенного объекта с его классом. Людям проще воспринимать окружающий мир как объекты, которые поддаются определенной классификации (например, разделение на живую и неживую природу).

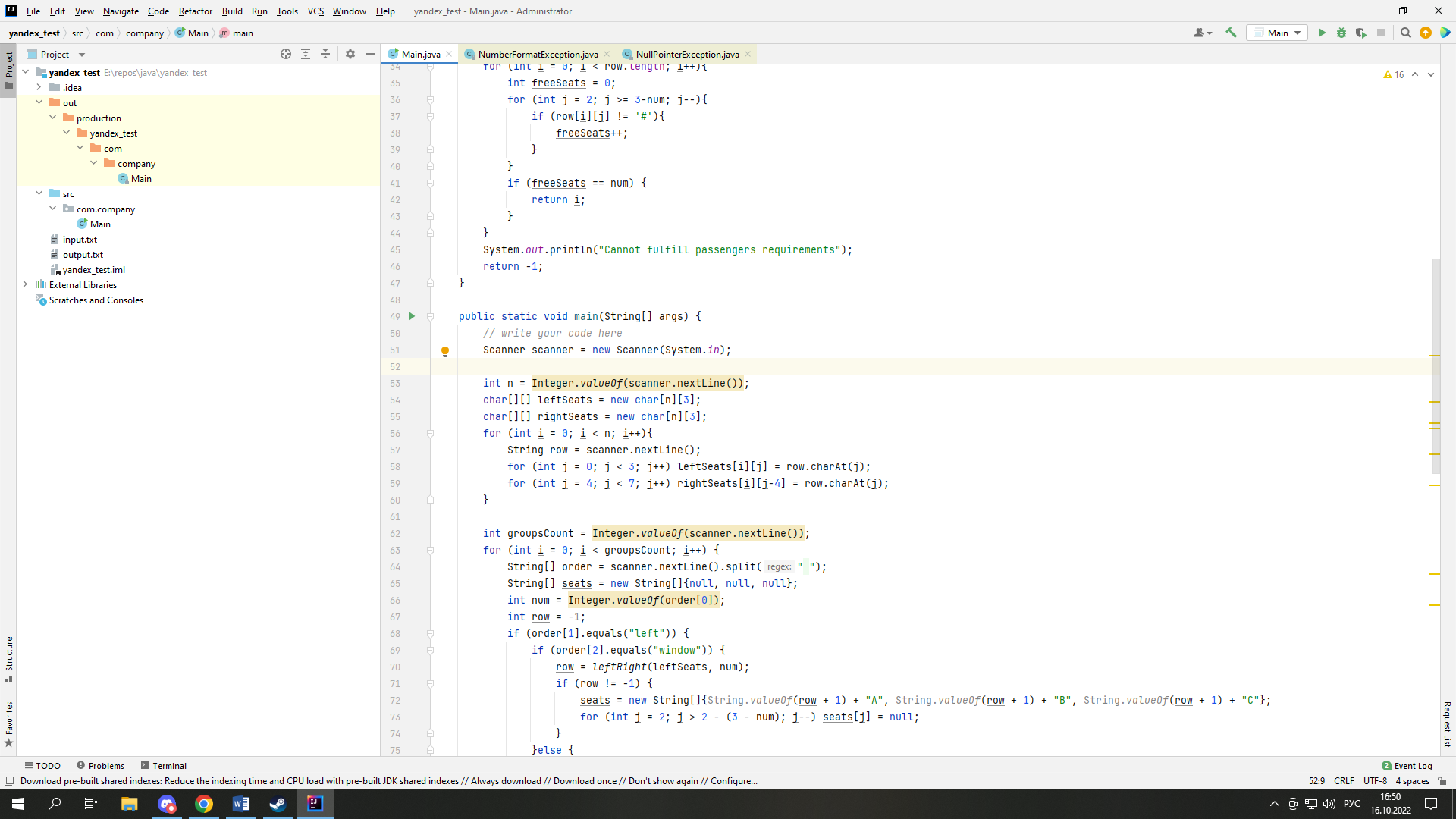
Кто-нибудь что-нибудь понял? И такой херни на просторах интернета навалом.

Я бы даже сказал что тут и хабр не поможет.. конечно трансформеры крутые, но такое



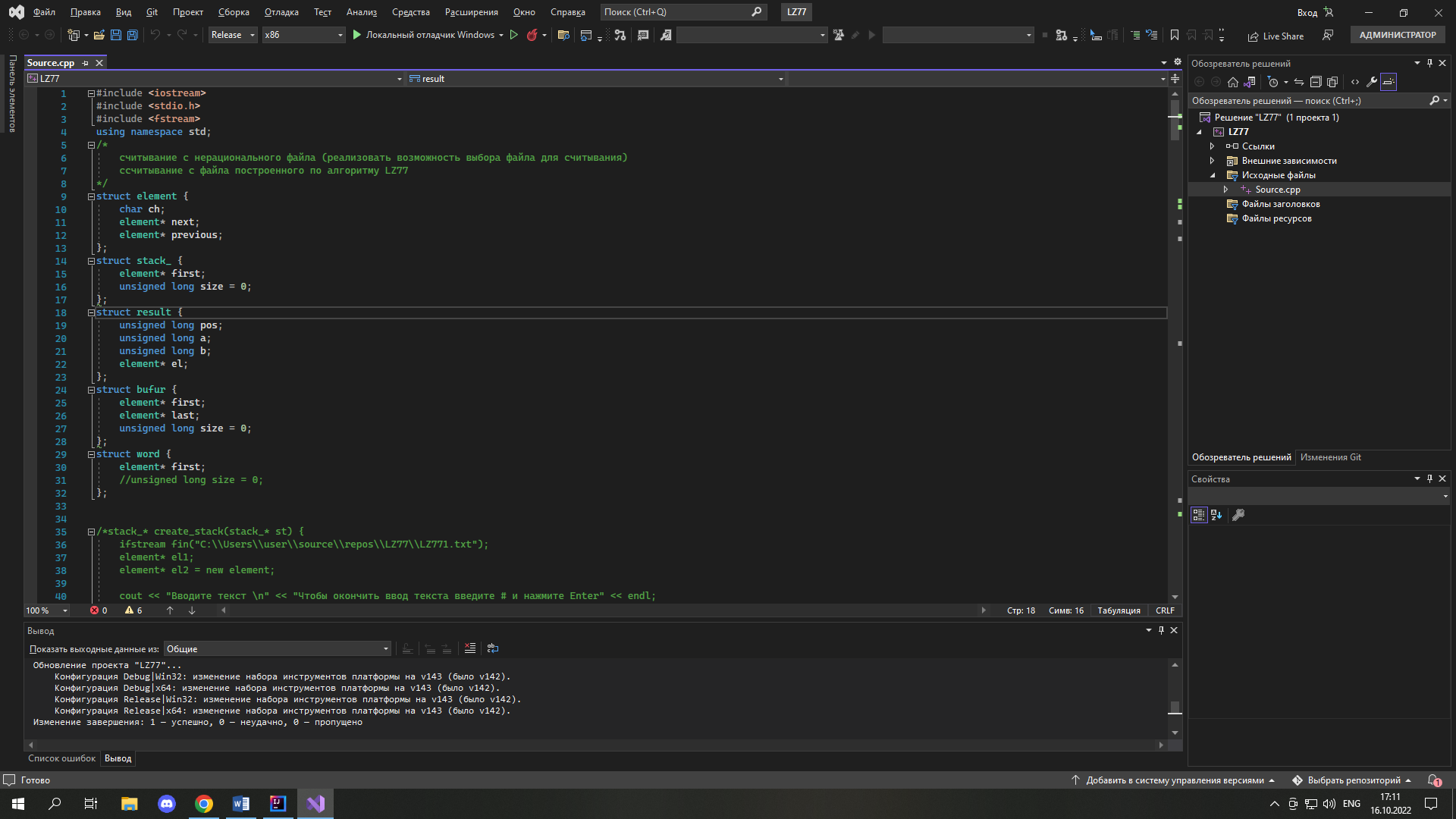
Сразу предупрежу мы НЕ будем углубляться в составление кода, особенности оформления и прочее, т.к. это скорее всего вам объяснят отдельно, а у нас не очень много времени и желания.

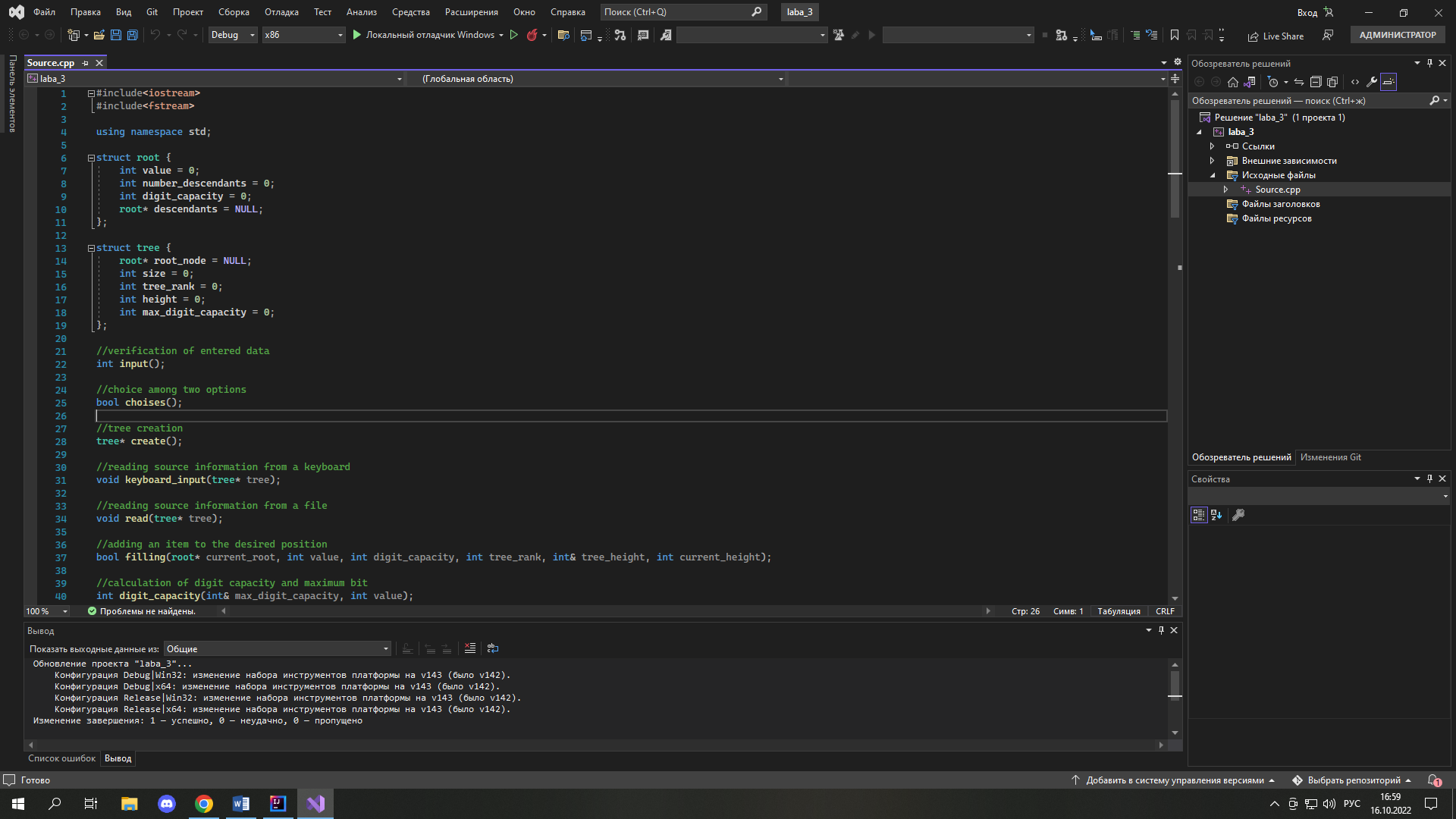
Поговорим в общих чертах:



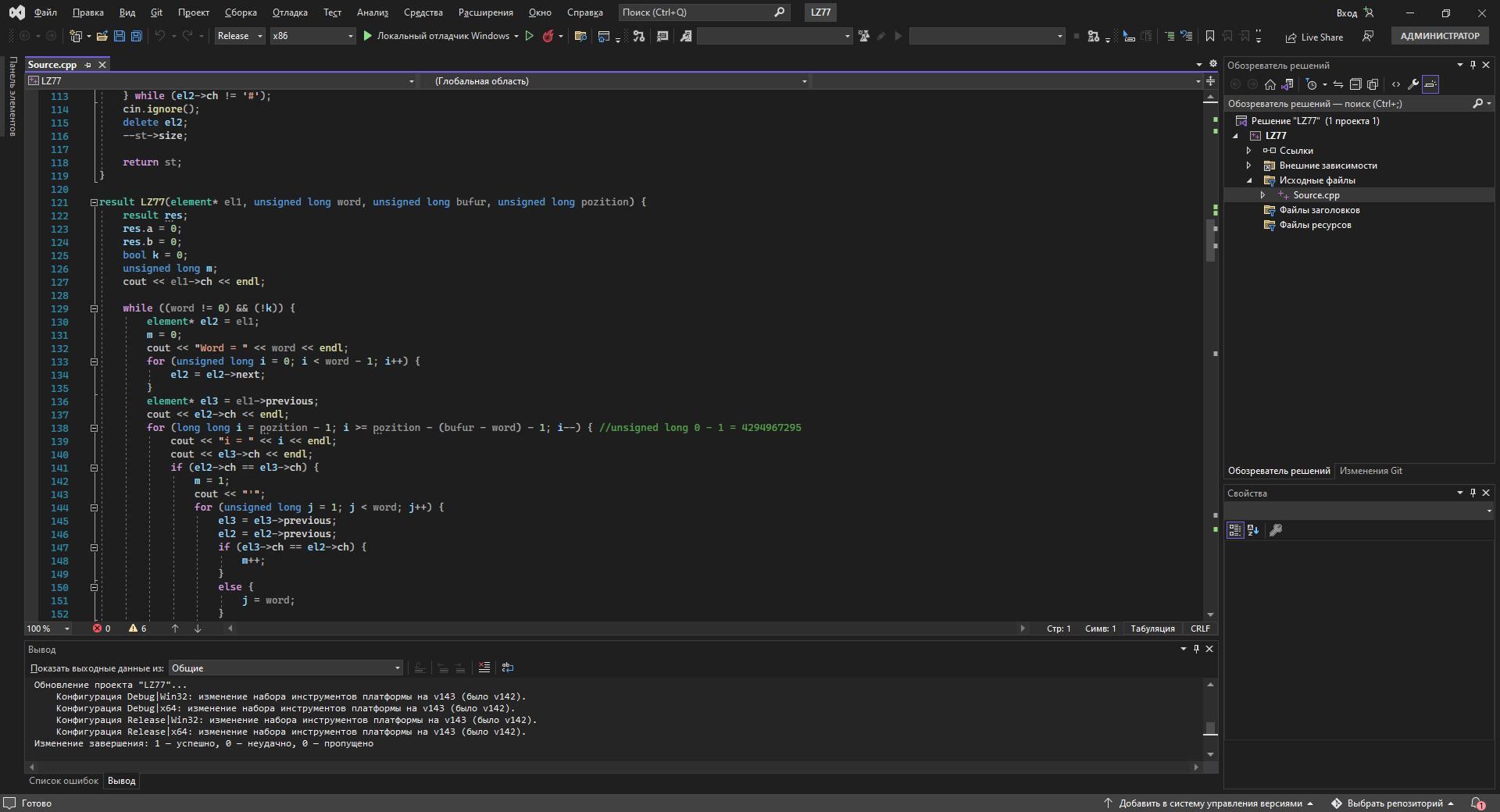
Вопрос: что из себя представляют эти строчки кода (n, I, j, leftSeats, rightSeats)? – это 3 переменные и 2 массива с которыми мы производим какие-то операции: объявление, заполение, изменение и прочее.

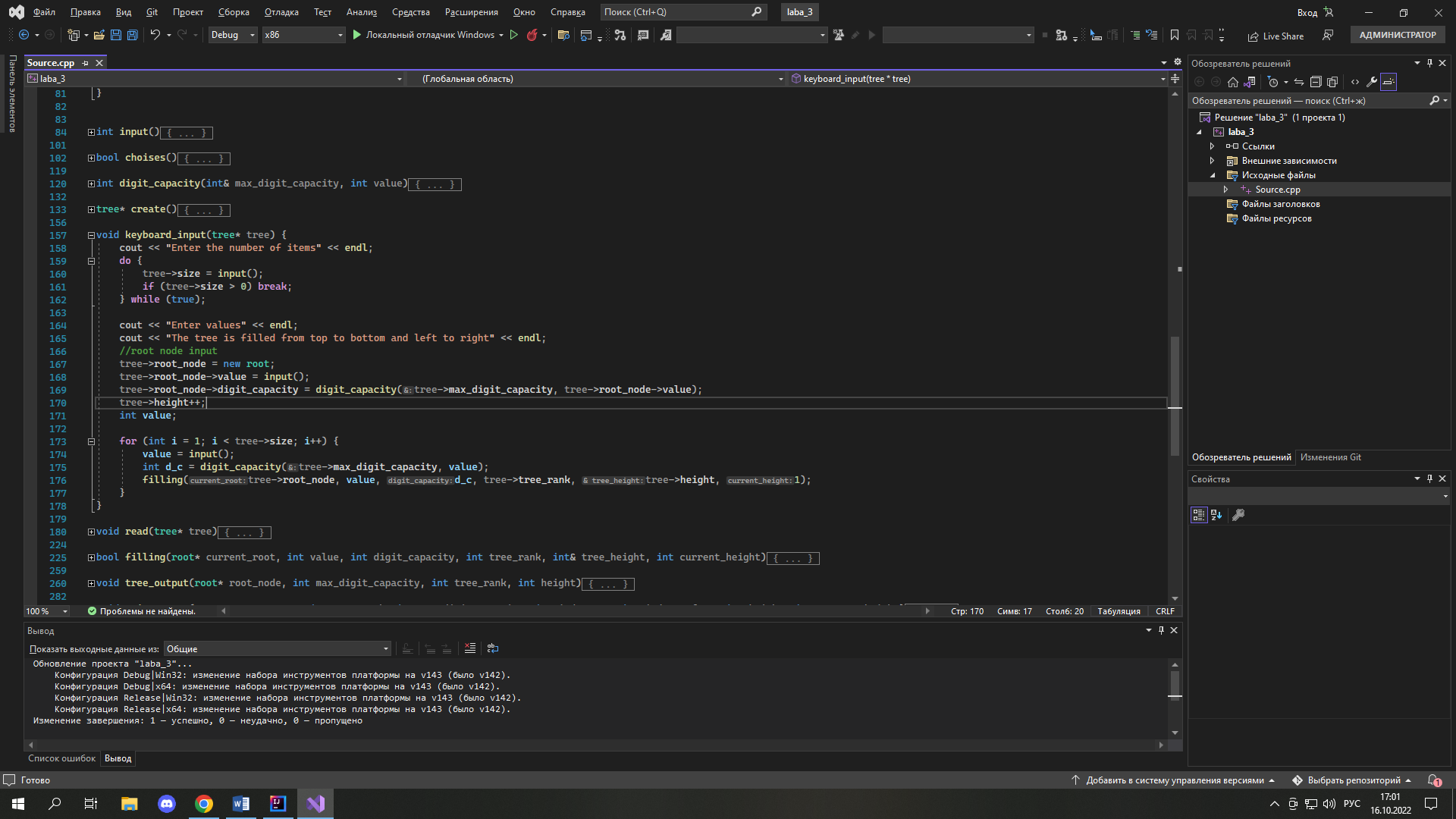
Может кто-нибудь из тех, кто копался в программировании даже сталкивался со структурами данных?





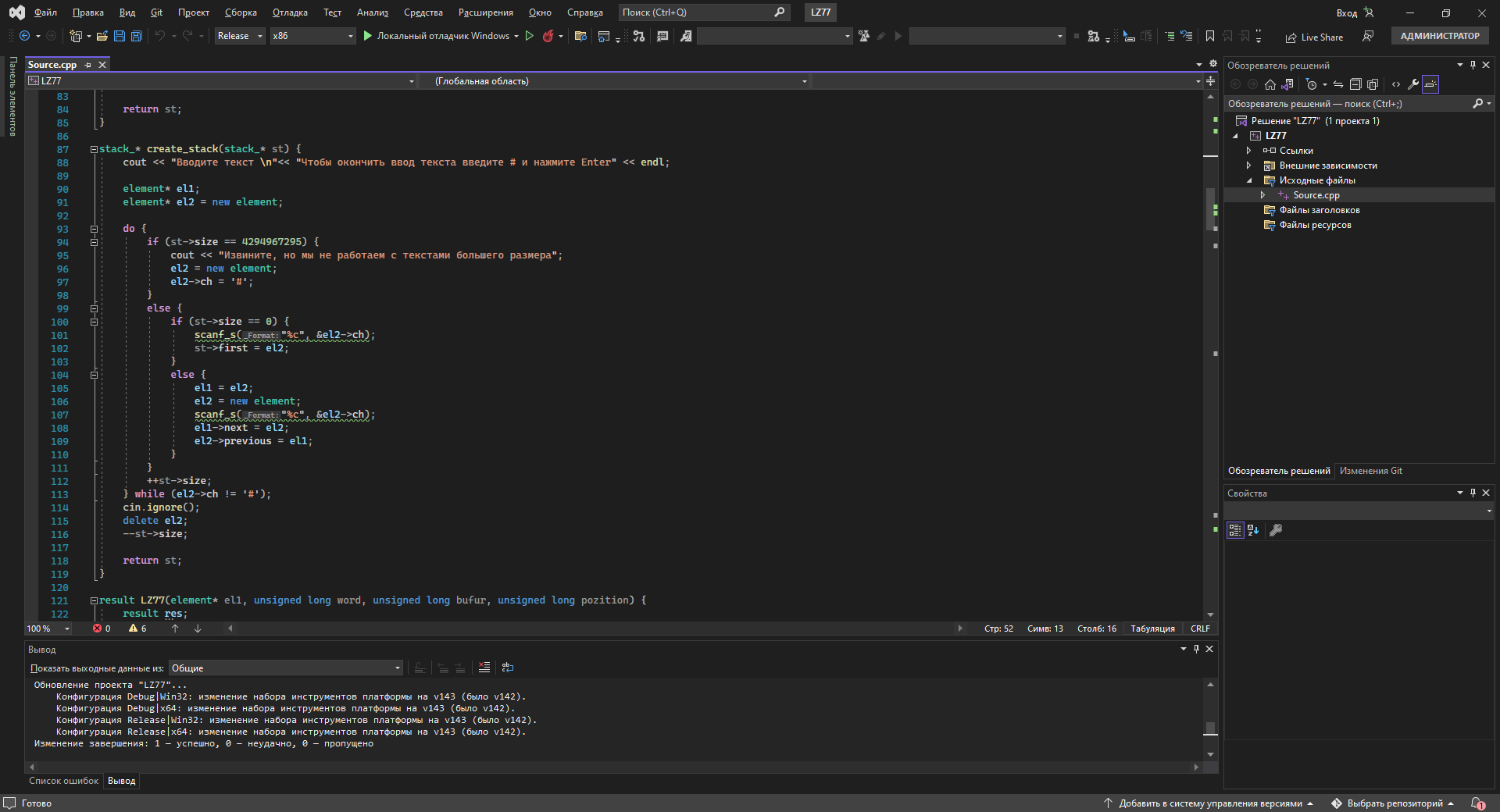
Если посмотреть на примеры, то это единица, хранящая в себе несколько параметров. Они используются, когда нужно вернуть или передать разом несколько параметров, можно (извините за тавтологию) структурировать данные для удобства чтения кода, или (с чем вы наверняка столкнётесь) можно сделать реализацию списков/динамических массивов и пр. (скорее всего у вас либо уже есть, либо будет поставлена задача в процессе обучения)





Кстати обращение к элементам структур идет через точку “.” (или если вы пользуетесь динамикой в плюсах, то обращаться нужно через “->” как указано в примере.

Однако и то, и то – это всего лишь выделенный кусочек памяти, способный лишь хранить в себе нули и единицы. Мы можем писать код для их преобразования или оформлять в функции, которые можно вызывать в ходе решения (кстати не нужно объяснять, что такое функция или всё же стоит?)

//////////////////////////////////рубрика поясняем за хуйню  
  
Представьте что в нескольких местах вам нужно выполнить схожие команды, причем не какое-нибудь инкрементирование переменной или вычисление суммы, а хоть сколько бы не было объемный код. Ввод данных с проверкой на дурака/вычисление пересечения объектов в заданном диапазоне значений или элементарное сложение всех соответствующих полей структур.

Эти операции займут несколько строк, тем самым будут засорять ваш код, но что хуже это будет ухудшать читабельность кода. Поэтому эти участки кода выносятся в отдельные функции, в которые можно передавать параметры и которая может возвращать результаты вычислений.

Замечательная вещь, у которой есть всего одно правило использования: название данных функций должно быть максимально говорящим.  
  
/////////////////////////////////конец рубрики

А теперь представьте, что и несколько параметров и функционал описывающий взаимодействие с этой информацией скомпонованы в один блок. Мало того можно делать из них конструктор, позволяющий на основе базовых блоков строить гигантские творения логического искусства (не побоюсь этого слова).

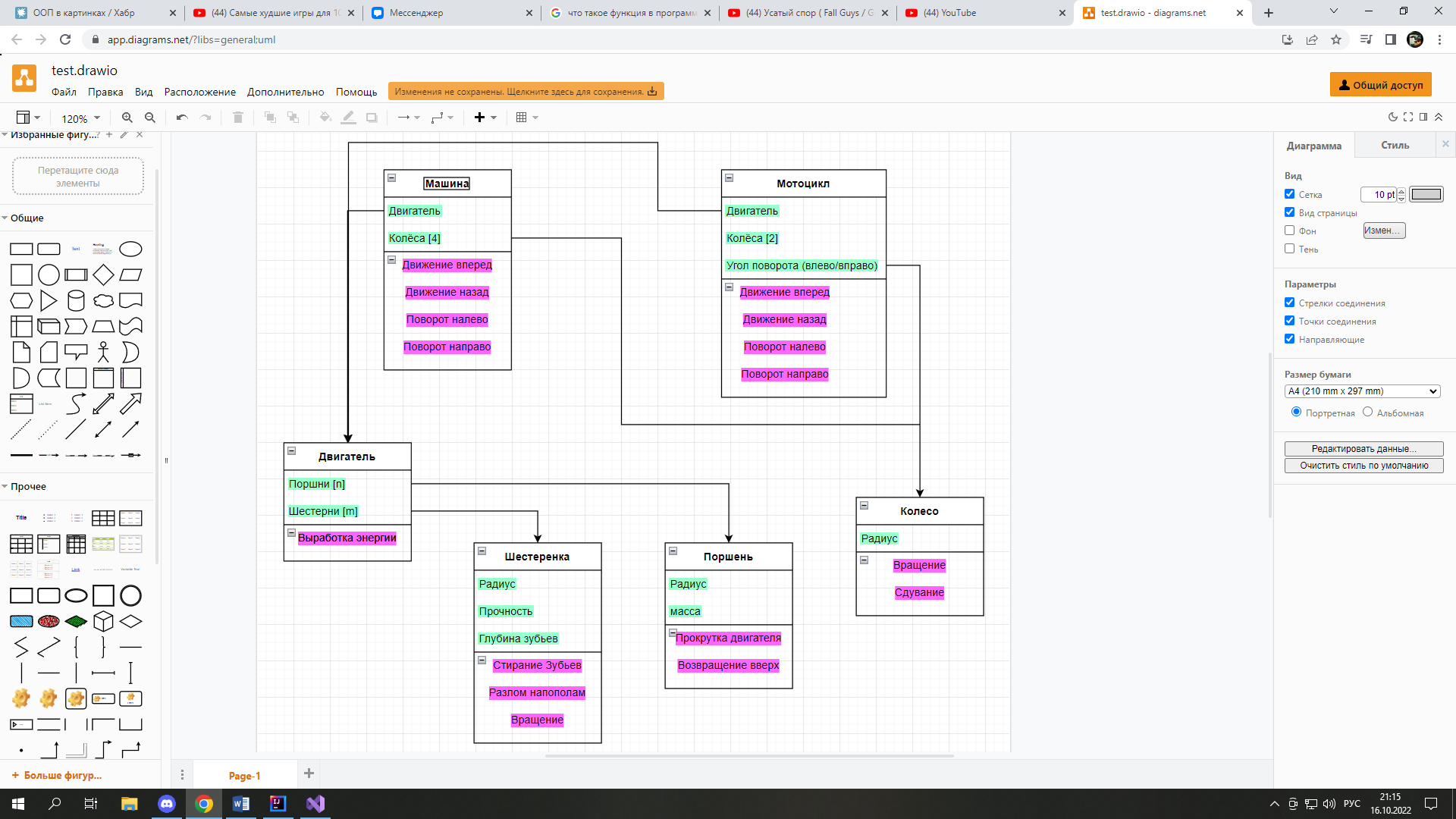
По сути это и есть ООП. Ты создаёшь класс, хранящий в себе поля и методы, которые можно вызвать при работе с объектом этого класса.

Например:

У нас есть класс Шестерёнка, у которой есть параметры прочность, радиус, глубина зубьев. И методы такие как: стирание зубьев и разлом на пополам.

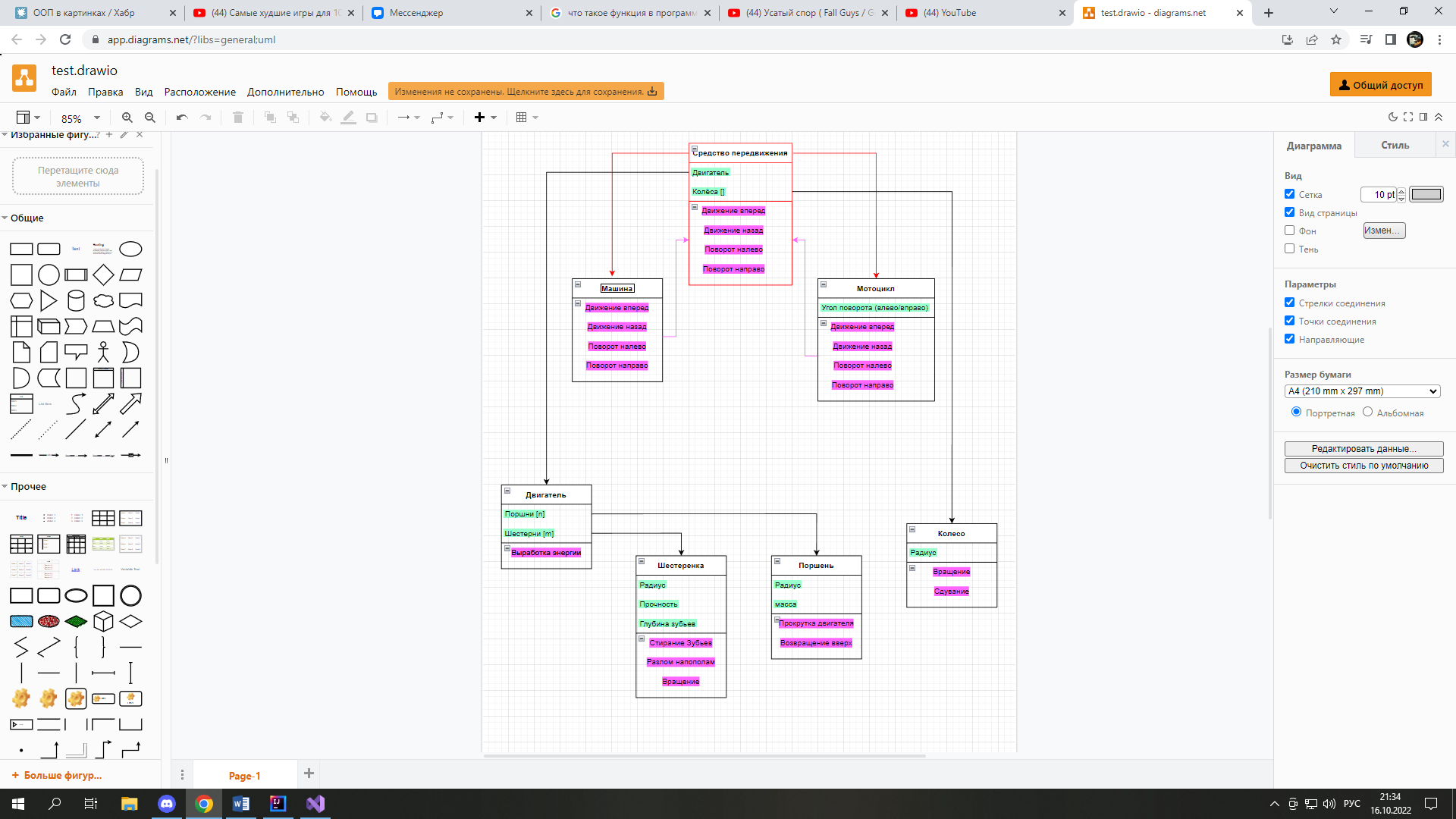
Есть Двигатель у которого есть поршни и шестерни, способный вырабатывать энергию

И прочее как представлено на слайде



Получилась вот такая схема в ней представлены описания классов, их поля и их методы. Однако если посмотреть на слайд то какую особенность можно заметить среди всех этих классов? (правильно машина и мотоцикл почти идентичны)

Тобеж по сути.. Можно вынести из классов общие поля и общие методы. Да это тут есть и это называется наследование. Когда несколько классов имеют схожие поля и схожие методы. Это как раз та модульность, о которой я говорил. Можно объединить мотоцикл и машину в общий класс Средство передвижения. Причем классы наследники могут иметь собственные реализации методов (разный принцип увеличения скорости у авто и байка) и дополнительные поля данных (как угол поворота у мотоцикла).

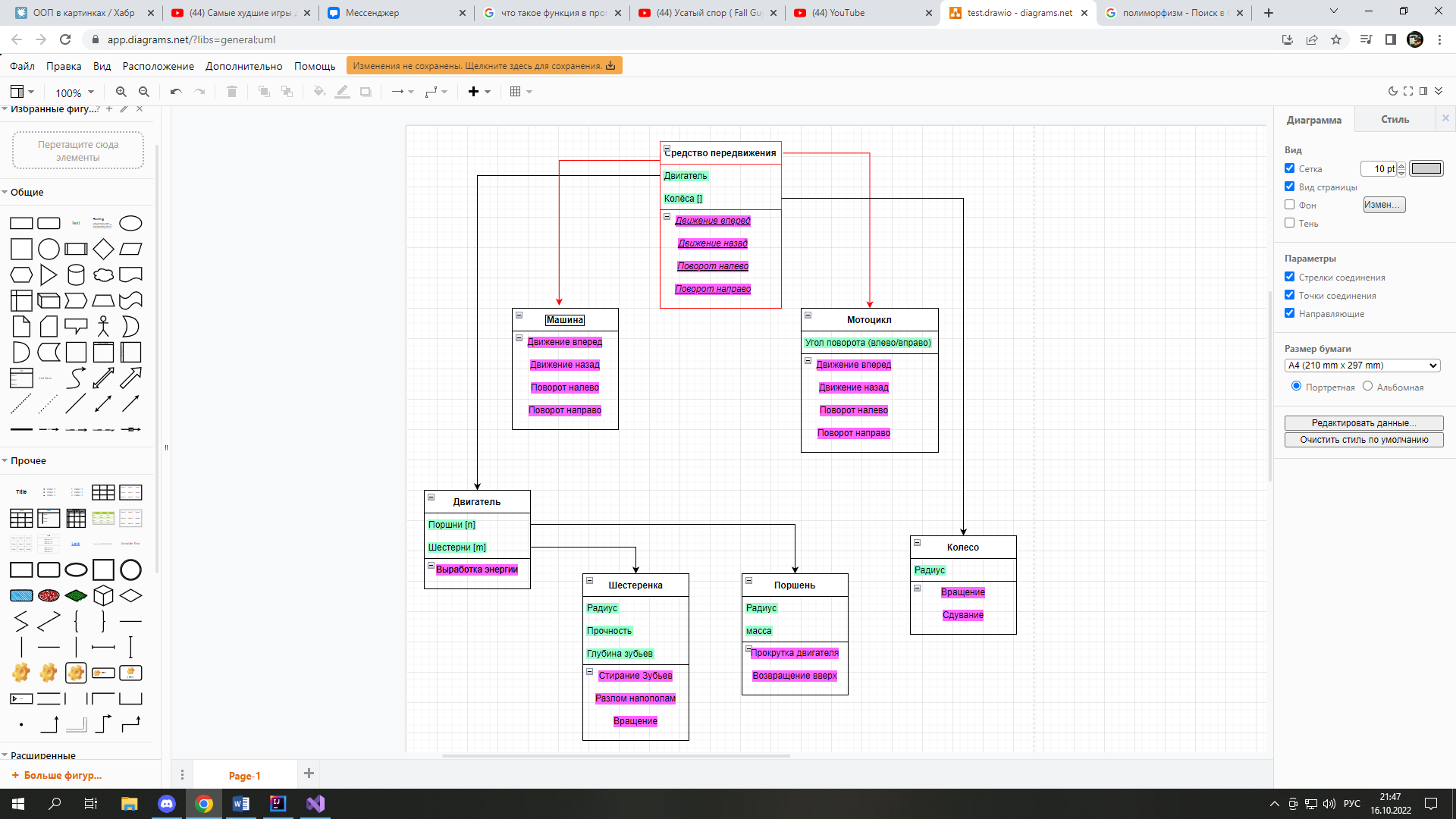


Однако наследование нужно делать с умом, т.к. переизбыток наследования может только запутать вас и пользователя. Например нету смысла объединять шестерёнки, поршни и колёса по параметру Радиус.

**Это мы рассмотрели 1й принцип ООП: Наследование.**

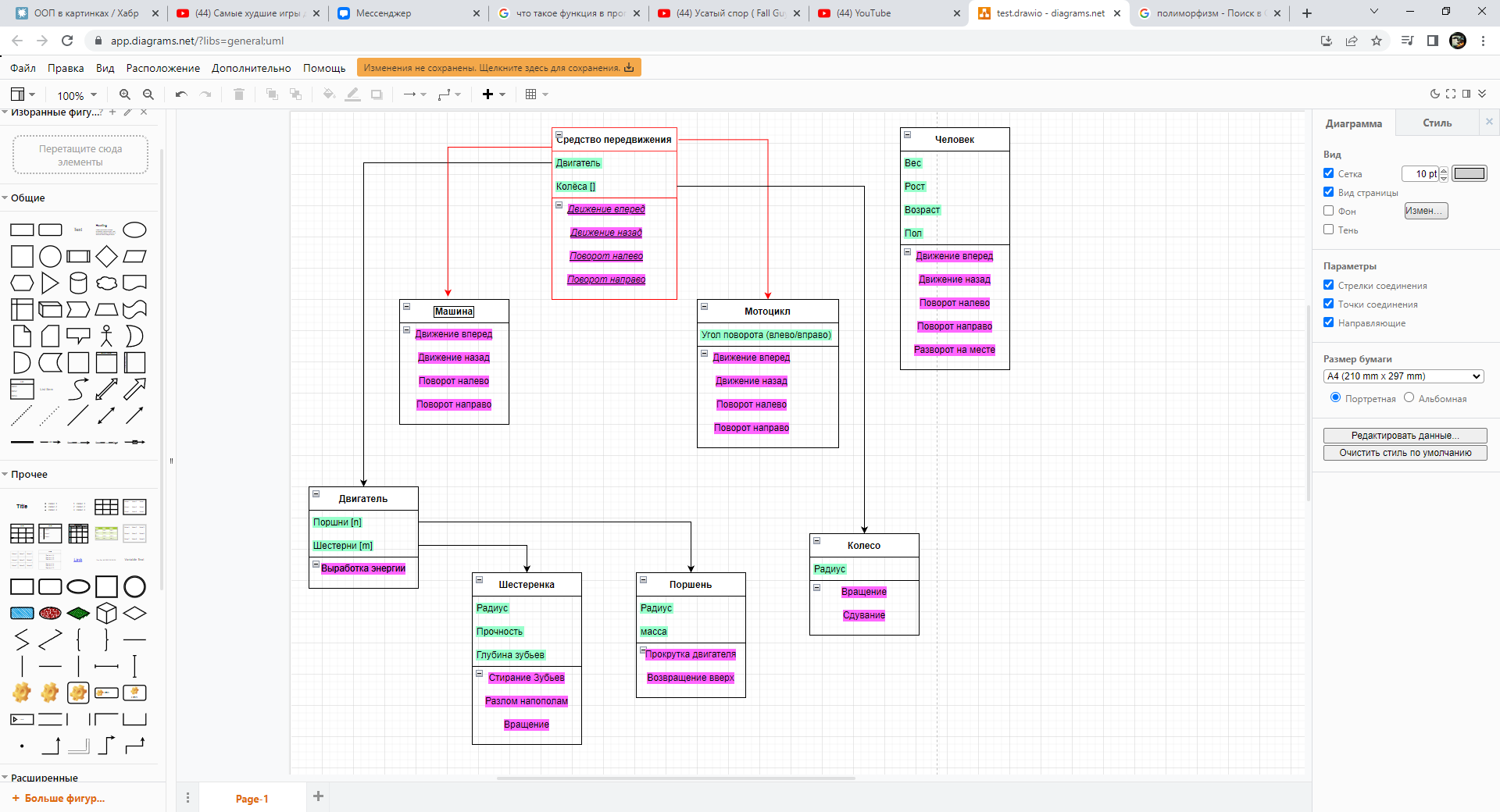
Однако у авто и мотоциклов управление происходит абсолютно по разному и нам приходится переопределять эти методы, что является тратой времени кода. + к тому мы не можем просто взять и создать транспортное средство – не существует такого предмета (лишь описание группы). Поэтому мы сделаем этот класс Абстрактным (его наследники могут существовать, а он сам нет, а также у каждого из них должны быть переопределены методы описанные в классе родителе).

**Это мы рассмотрели 2й принцип ООП: Абстракция.**



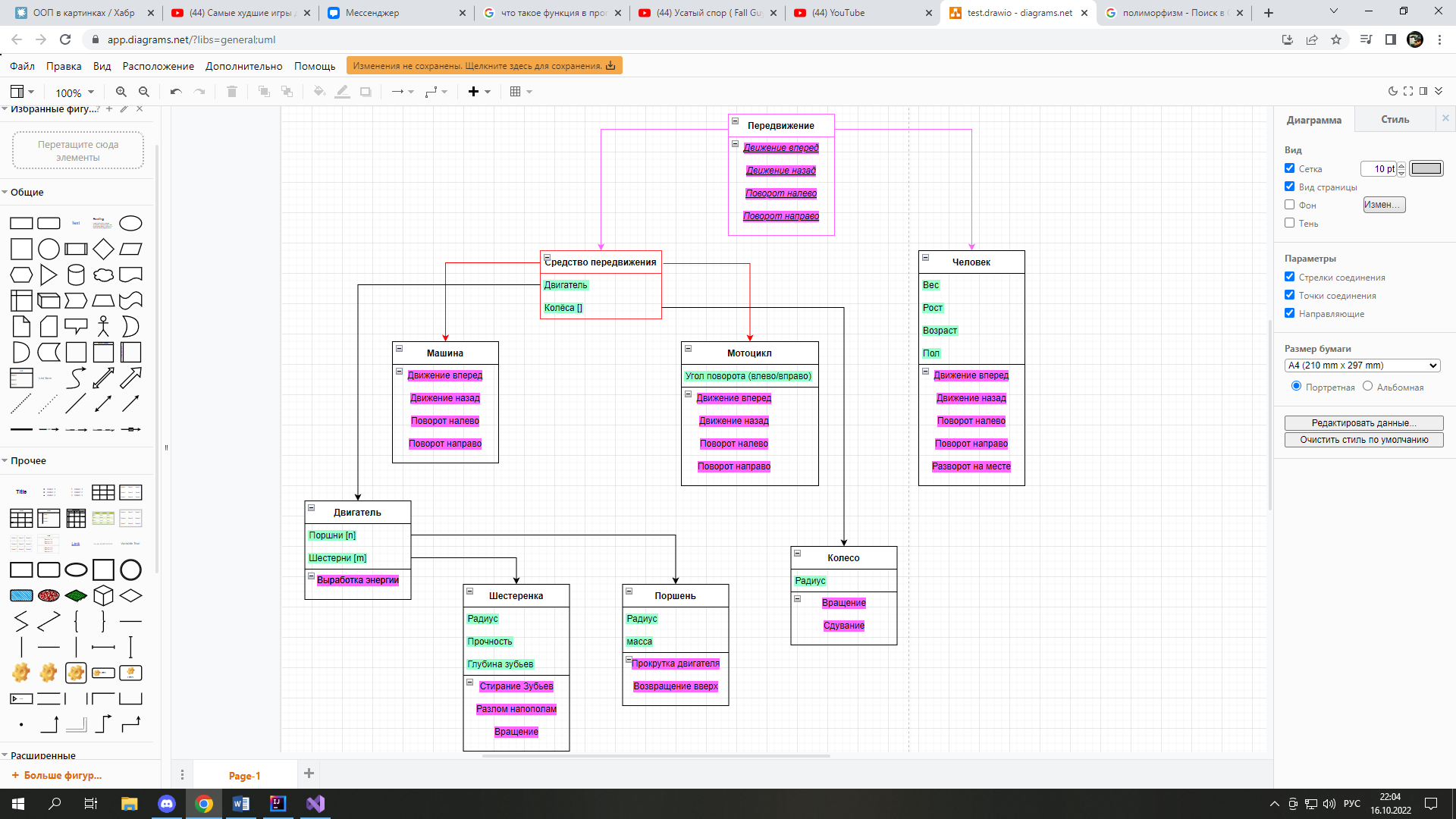
А что если мы хотим добавить в эту систему человека

Он также может ходить во все стороны, но в отличие от своих железных друзей он может развернуться на месте (добавим для наглядности)



Как мы видим общих полей у него со средствами передвижения у него нет, но есть общие методы -> нужно сделать объединение по ним и Это называется Интерфейс – фигня позволяющая объединить классы по общим методам с условием их переопределения в каждом из классов.

Не напоминает чем-то абстрактный класс?))



Добавим интерфейс Передвижение в котором объявим общие методы без реализации.

**Это мы рассмотрели 3й принцип ООП: Полиморфизм (свойство системы, позволяющее иметь множество реализаций одного интерфейса).**

P.S. Опять же с ним перебарщивать не стоит – объединять шестеренки и колёса по интерфейсу вращающиеся смысла нет.

В итоге мы получили систему передвигающихся объектов, связанных между собой общими свойствами или методами, а также классы имеющие в качестве своих параметров объекты других классов.  
Остаётся вопрос имеет ли доступ двигатель к изменению параметров шестеренок и поршней. Не понятно составлен вопрос? Перефразирую: *Можем ли мы имея автомобиль выкинуть из двигателя пару шестерёнок и продолжить движение словно ни в чем не бывало?* Скорее всего нет. Для этого существует **4й принцип ООП: Инкапсуляция.**

Заключается он в предоставлении разных уровней доступа разным пользователям. Приведем пример:

Стирание зубьев у шестеренки - это внутренний естественный процесс, однако нам не позволено их стачивать поэтому статус этого метода **Private (исключительная особенность этого класса).**

Разлом шестерни напополам может вызвать не только сама шестерня в результате производственного брака, но и мимо проезжающий поезд, который сбил вас на железнодорожном переезде поэтому статус Этого метода **Public (общий доступ).**

Есть еще несколько статусов влияющих на уровень доступа, но сейчас остановимся на этих т.к. я устал объяснять а вы наверняка слушать.

Назовите 4 принципа ООП и что они делают. (хорошо – те кто ответили молодцы, а остальные перепишите у него эти определения, т.к. это база)

Дома вам нужно придумать любую систему, в которой будет присутствовать Наследование и Полиморфизм будь то акулы/люди/лодки/байдарки, супы/мет/грешники в аду и прочее

Креативьте короче

А схемы нужно нарисовать в знакомом вам <https://app.diagrams.net/>

В разделе Диаграмма класса.

Оформление такое же как в примере:

* Классы – Черная рамка с 3мя отделами (название/поля/методы)
* Методы – Розовая заливка теста
* Поля – Зеленая заливка текста
* Абстрактный класс – Красная рамка и стрелки
* Интерфейс – Розовая рамка и стрелки

При сдаче будем спрашивать вас по рандомному полю является ли оно публичным или приватным и объяснение почему это так.

Всем спасибо