[**Объекты Apple (name, isWinter), Pear (name, type), Orange (name, country)**](#_vr6ndlox0s81) **1**

[**Создаём объект Basket, в котором будем хранить наши объекты**](#_8ho9slxofdw2) **2**

[**Create Car and Engine**](#_re6evlkt40ww) **3**

[**Inject Engine object through the Car constructor**](#_r912pr5sl16l) **3**

[**Давайте добавим метод hasTurboEngine для типа Car**](#_92vvie6let63) **4**

[**Приватные методы и проперти**](#_w0jyc8apdz9x) **4**

[**Приватность на примере Корзины с фруктами**](#_6lekyuh9ep18) **5**

[**Пишем программу с рендерингом**](#_70i4xlv8wyhp) **7**

Во всех заданиях объекты нужно создавать с помощью конструктора.  
Там где нужно будет создать объекты с помощью литерала объекта, будет об этом сказано!

### Объекты Apple (name, isWinter), Pear (name, type), Orange (name, country)

Создайте конструкторы для этих 3-х типов фруктов (с пропертями, указанными в скобках)

Насоздавайте с помощью этих конструкторов по 5 фруктов каждого типа.  
  
Вопрос, как правильно делать:  
var apple1 = new Apple();

apple1.name = ‘white naliv’;

apple1.isWinter = true;

или

var apple1 = new Apple(‘white naliv’, true);

Зелёный вариант более красивый. Более компактный. Позволяет проинициализировать объект стартовыми значениями в момент создания.

Если понадобится далее изменить какое-то проперти, то это можно ведь сделать без проблем вот так:  
apple1.name = ‘white naliv2’;

Поэтому не забываем в конструктор добавлять эти самые параметры.  
На примере Apple:  
  
function Apple(name, isWinter) {

this.name = name;

this.isWinter = isWinter;

}

### Создаём объект Basket, в котором будем хранить наши объекты

Попробуйте спроектировать этот объект сами исходя из следующей спецификации:  
1. мы должны “приказать” корзине положить в себя продукт, передав данный продукт (подсказка, вызвать метод у корзины, передав ей параметром продукт)

2. мы должны иметь возможность сказать корзине: дайВСЕЯблоки, дайВСЕАПельсины, дайВсеГруши

Имея спецификацию, что вот корзина умеет делать, мы можем начать ей пользоваться:  
  
**var basket = new Basket();**

**basket.addProduct(apple1);**

**basket.addProduct(apple2);**

**basket.addProduct(orange1);**

**basket.addProduct(orange2);**

**var allApplesFromBasket = basket.getAllApples();**

**for (var i=0; i < allApplesFromBasket.length; i++) {**

**console.log(allApplesFromBasket[i].name);**

**}**

то есть выше мы положили 2 яблока в корзину, значит метод **getAllApples** должен вернуть нам эти все яблоки.  
  
Всё, есть ТЗ, есть как бы пример использования… осталось подумать, а как же мы реализуем внутренности нашего класса (конструктора) Basket?

нужно ответить на вопросы:   
1. где будем хранить фрукты (каждый тип фрукта в отдельном массиве?)  
2. у нас один метод для добавления всех типов фруктов **addProduct.** Как внутри метода узнать, в какой массив закинуть переданный продукт?  
Почитайте про **constructor.name**  То есть с помощью if-else детектим, в какой массив нужно запушить переданный продуктДа, мы как разработчики корзины мы должны знать заранее, какие типы продуктов наша корзина можем принимать

**Заказчик пришёл и такой: чуваки, срочно нужна возможность очищать корзину полностью.**

Мы такие сразу начинаем использовать эту функциональность:  
**basket.clear();  
var allApplesFromBasket = basket.getAllApples();**

**console.log(allApplesFromBasket.length === 0); // ведь после очистки у нас в корзине не должно остаться ни одного яблока**

А теперь давайте же добавим в наш конструктор такой метод clear (ну.. мы там просто берём и очищаем наши массивы). Самый простой способ очистить массив - переприсвоить новый пустой массив: this.apples = [];

### Create Car and Engine

Создайте 4 объекта типа **Car.** Каждый объект типа Carимеет 3 проперти**:  
model (string), weight (number), \_engine (**тип **Engine**, имеющий сво-ва**: power (number), capacity (number), hasTurbo (boolean)**

*Объект engine должен создаваться внутри конструктора Car. То есть все машины, созданные с помощью этого конструктора будут иметь движок с одними и теми же характеристиками.  
  
А вот model, weight передавайте параметрами в конструктор Car*

*подсказка:****function Car(model, weight) {***

***……..***

***}***

### Inject Engine object through the Car constructor

В предыдущей таске, у нас каждый объект авто, созданный с помощью конструктора Car имеет один и тот же движок. Это логично, ведь инстанс типа Engine создаётся внутри конструктора Car, поэтому он там жёстко захардкоден.   
То етсь по сути у нас респонсибилити по созданию движка лежит на автомобиле, а ведь автомобиль не создаёт в реальной жизни движок, он им владеет, но не создаёт.  
В реальной жизни тачка создаётся сама по себе (корпус как бы), а внутренности в неё запихиваются уже после (либо же прямо во время создания корпуса).  
  
Давайте сделаем так же. Теперь нам, перед тем как создавать авто (то есть перед тем как мы будем писать **new Car(‘reno’, 1231),** нам нужно сначала создать 4 объекта типа **Engine**

То есть у нас есть 4 движка, и теперь при создании авто мы инджектим (внедряем) в него нужный движок 3-им параметром  
  
var **turboEngine** = new Engine(true, 2.4, 323);

var carWithTurbo = new Car(‘reno’, 23123, **turboEngine**);

То есть мы теперь внедряем уже готовый движок внутрь автомобиля во время создания этого самого автомобиля.

### Давайте добавим метод **hasTurboEngine** для **типа Car**

**(**добавляем метод в конструктор, и как результат - все объекты, созданные с помощью этого конструктора будут иметь такой метод у себя)

Этот метод должен вернуть true или false, зависит от того, какой тип движка у автомобиля. То есть Car внутри этого метода обращается к своему проперти this.\_engine и спрашивает у него его проперти hasTurbo.

### Приватные методы и проперти

Заметили, что у Car проперти \_engine имеет в названии нижнее подчёркивание?  
  
Это договорённость, которая говорит о том, что такие сво-ва и методы являются ПРИВАТНЫМИ (private).   
  
Всё остальное - это публичное. Именно публичные методы являются ИНТЕРФЕЙСОМ, то есть тем, через что можно с объектом взаимодействовать.

Для чего приватные нам сво-ва? Это значит, что этими сво-вами может пользоваться только сам объект. То есть у нас вот есть объект типа **Car (с большой буквы)  
  
var car1 = new Car(xxxxxxxxxxxx);**

То есть мы пользуемся объектом, мы им владеем, так как вот мы создали этот объект вот здесь в коде у себя.  
И мы имеем право взаимодействовать с этим объектом через его интерфейс (набор публичных сво-в и методов).  
  
Мы можем прочитать публичное сво-во:  
window.alert(car1.model);

или его изменить:  
car1.model = ‘new model value’;  
  
Но так мы не имеем права писать (но технически всё будет работать):

alert(car1.\_engine.hasTurbo);

То есть мы не имеем права обращаться к приватному проперти объекта. То есть создатель спецификации-конструктора решил сделать это проперти приватным, значит им может пользоваться только сам объект внутри своих методов, но извне объекта мы не имеем права его дёргать.  
  
Как в реальной жизни: мы садимся в тачку и взаимодействуем с ней через интерфейс. Мы именно ТАЧКЕ даём команды, не имеем права давать команду прямо двигателю (то есть мы не должны открывать капот и прямо там “зажигать” двигатель). Это называется инкапсуляция (сокрытие деталей), то есть нам дали объект типа Car и всё, мы понятия не имеем. из чего он там внутри состоит, мы только дёргаем приватные методы и проперти.

Именно поэтому, так как \_engine - приватное св-во, мы, как пользователи объектов car1, car2,... не можем узнать, какой там движок (с турбо или без) напрямую у движка. Ведь движок сидит внутри Car и мы должны спрашивать об этом именно тачку.

А уже тачка полезет в свой приватный метод \_engine и спросит у него.

то есть вместо кода:

alert(car1.\_engine.hasTurbo);

мы пишем:  
alert(car1.hasTurboEngine());

то есть мы не лезем в приватное проперти, а работаем с публичными методами.  
  
Да, мы можем сделать \_engine публичным и сказать: нужно что узнать о двигателе - обращайтесь к публичному проперти:

alert(car1.engine.hasTurbo);

Но не будет ли это избыточным раскрытием для пользователя внутреннего устройства авто? Будет… Но всё зависит от задачи. Нужно всегда искать правильную золотую середину.

### Приватность на примере Корзины с фруктами

Глядите, у нас есть метод: getAllApples();  
То есть мы такие   
**var apples =** **basket.getAllApples();**А можно ведь достучаться до проперти-массива и работать с ним:  
**var apples =** **basket.apples;**

Зачем нам тогда метод **getAllApples?** Аналогично зачем нам метод **addProduct,** когда мы можем напрямую запушить в массив:  
basket.apples.push(apple1);

Всё дело опять же в инкапсуляции и чрезмерном раскрытии информации.  
Ну не круто, пользователю корзины, знать о её внутреннем устройстве. Ну не твоё это дело, лезть в конкретный массив и оттуда вытягивать эти яблоки.  
  
Как и не должен ты брать и напрямую пушить в нужный массив нужный продукт.  
basket.apples.push(apple1);

А вдруг корзина логирует\фиксирует (или подсчитывает вес при каждом добавлении) нового продукта.  
  
Да, такого требования не было изначально. Но если вдруг оно появится?  
basket.apples.push(apple1);

basket.apples.push(apple2);

basket.oranges.push(orange1);

basket.oranges.push(orange2);

…… таких 1000 строк…  
  
как будем в этом случае фиксить ситуацию?  
а если у нас добавление нового продукта инкапсулировано в методе, то нам нужно только этот метод исправить.  
  
Давайте так и сделаем.  
Пусть у корзины будет проперти-счётчик: **winterApplesCount**.  
То есть при добавлении продукта, если это яблоко и если это яблоко Зимнее, то счётчик инкрементим.

**basket.addProdict(apple1);**

**basket.addProdict(apple2);**

**console.log(basket.winterApplesCount === 1); //true если из двух яблок только одно - зимнее**

ну и давайте значит отрефакторим наш код, чтобы массивы стали приватными пропертями, поставив нижнее подчёркивание.  
  
Это даст понять всем пользователям разработанной нами корзины, что так делать нельзя:  
basket.\_apples.push(apple2);

basket.\_oranges.push(orange1);  
  
то есть технически так сделать пользователи нашей корзины смогут, но наша корзина придёт в инконсистентное (нецелостное, битые данные) состояние: по факту в корзине будет 4 яблока напрмиер зимних, а счётчик покажет - 0;  
  
Это как в автомобиле, мы можем залезть внутрь движка и попробовать его как-то завести.. но вот подхватятся ли другие все системы у автомобиля и будет ли авто в целом корректно работать?

**Замечание.** Ещё почему круто не знать о внутреннем устройстве корзины и как она там хранит эти фрукты? Прикиньте, что в один момент появился серверный разработчик и говорит: всё, все фрукты теперь хранятся на серваке. И если чувак добавит в корзину новый фрукт - сразу заливай его на сервак. У себя ничего не храни.  
А если чувак запросит ВСЕ фрукты - спрашивай у сервера - он тебе всё отдаст.  
  
Получается меняется инкапсулированная внутри логика. А сам интерфейс остаётся тем же. То есть клиент корзины (пользователь корзины) продолжает работать с корзиной как и работал, ничего не подозревая о том, что корзина теперь шлёт запросы на сервак:  
**basket.addProdict(apple1);**

**basket.addProdict(apple2);**

Всё как и раньше, но каждый такой вызов метода addProduct отправит запрос на сервак.

### Пишем программу с рендерингом

Ну.. Имеем теперь классы фруктов и корзины (а ведь это может быть КОРЗИНА товаров в интернет магазине, смысл один и тот же), ок, что дальше?  
  
Давайте напишем программу, которая будет иметь UI… Что она будет делать? Отображать список товаров в корзине…  
  
При этом наши классы (они же конструкторы) (фрукты и корзина) не будем нагружать респонсибилити отображения. То есть.. Это наш BLL (бизнес уровень, он выполняет только логику).  
  
Всей визуалицией пусть занимается уровень UI  
  
Добавьте новый класс, который будет называть **function BasketApp**

Это будет наш класс, который соединит отдельные компоненты воедино и будет ими управлять.

Позырьте картинку внизу



**BasketApp он же UI на картинке**

То есть мы можем создать объект класса BasketApp и сказать ему сначала **render** (чтобы он отрисовал кнопки управления).  
Вот так должен выглядеть HTML примерно:  


то есть мы вызвали render и он нам нарисовал кнопку createFruits и сказал, что корзина пуста.

Дальше мы нажимаем на кнопку **Create Fruits.** Наш листенер должен выполнить 2 операции: заполнить корзину и перерисовать её.   
1. взять и запустить метод **putFruitsToBasket** (чтобы он наполнил свою корзину фруктами (тот код, что вы писали в предыдущих заданиях).   
2. дальше - **renderBasket**, чтобы объект BasketApp взял свою корзину и пробежался по всем фруктам и нарисовал эти фрукты.

Получится такая картина:  


то есть по факту у нас получается такая диаграмма классов (кто объектами какого класса управляет, кто чья шлюшка)

