Przykład dirty coda na przykładzie: factory function, higher hierarhcy function -> bo mamy funkcje ktora towrzy inna funkcje i zwraca

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Czcionka

Opis wygenerowany automatycznie

Ulepszony -> poprawione nazewinictwo - uzycie factory function klasy Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Object oriente programming maybe worth to get into

Functional vs OOP vs procedural. Niektore paradygmaty I jezyki maja swoje interal rule które nadpisują general rule, warto się ich trzymać.

CleanCode General rules:

1. Readable and meaningful names
2. Slim, concise (zwięzłe) functions
3. Understandable code flow

Clearn Architecture to co innego niz clean code. Clean architecuture jest more generale – focus on the project as a whole, clean code focus on a single file/ problem.

How to write clean code:  
Obraz zawierający tekst, zrzut ekranu, Czcionka, diagram

Opis wygenerowany automatycznie

# Names

Nazewnictwo powinno byc meaningful. Uzywajmy takich nazw żeby wiedzieć co się dzieje w srodku, bez potrzeby wchodzenia do srodka.

Variables & Constans – data containersy, uzywamy rzeczownikow albo którkich fraz z przymiotnikami np. const userData = {} || is valid = …  
Functions/ Methods – komendy/ działanie do wykonania. Uzywajmy czasownikow, lub ktorkich fraz z przymitoniami np. sendData(), inputIsValid()  
Class – używamy do tworzenia „rzeczy”. Uzywajmy rzeczownikow lub krótkich fraz z rzeczownikami np. Users || RequestBody.

Name Casing:  
snake\_case – w pythnie dla varabili/ funkcji/ metod  
camelCase – w js/ javie dla variabli/ funkcji/ metod  
PascalCase – w pythonie javie/ js dla klas  
kebab-case – w htmlu, dla htmlowych customowych elementow.

Variable || Constants || Properties:  
Wartość możę być obiektem np. listą/ mapą/ data structure - opisz wartość np. user | database. Gdy jest to wymagane możesz dosarczyc więcej detali ale tak by nie było redundancy np. authenticatedUser | sqlDtabase   
Wartośc możę być liczbą lub stringiem – opisz wartość uzytwana np. age | name. Gdy jest to wymagane możesz dosarczyc więcej detali ale tak by nie było redundancy np. firstName.  
Wartość logiczna true/false - odpowiedz na pytanie np. isActive | loggedIn. Gdy jest to wymagane możesz dosarczyc więcej detali ale tak by nie było redundancy np. isActiveUser.

Przykład do przemyślenia user\_data vs user kiedy nazwac zmienną user a keidy user\_data. Gdy tworzymy obiekt który posiada bd w sobie proeprty chyba lepiej to nazwac user po prostu, a jezeli przekazujemy do tego obiektu do konstruktora jakiś dict z danymi np. to może się nazywać user\_data.

Function:  
Funkcja która wykonuje jakas operacje – tu nazwa funkcji powinna opisywac operacje np. getUser() getUserByEmail()  
Funkcja która zwraca boolową wartość – tu powinno odpowiadać na pytanie prawda fałsz np. isValid() emailIsValid()  
Wyjątkiem są gettery i seter niby jak funkcja ale używamy żeby dostać porperty jakies z klasy wiec zmienna na której wykonamy jakas operacje wiec nazewncitwo powinno być bardziej jak zmiena.

Naming classes:  
Nazywajac klase opisz object kotry się w niej bd znajdował np. Product, Customer, SQLDatabase, Database. Wybierajac nazwy metod rozroznialnie. Uzywaj tych samych patternów w roznych miejscach uzywajac fetch słowa żeby pobrać dane, używaj go wszędzie.

# Comments

Raczej unikaj komentarzy, kod sam się powinien opisywac. Jeżeli mialbys np. uzyc komentarza żeby wydzielić seksej na klase i stałe to może warto się zastanwoic czy nie lepiej utowryzc nowy katalog i czy nie ma za dużo kodu w jednym miejscu. Usuwaj kod jak go nie używasz niż raczej trzymać.

Documentation String maja sens gdy uzywamy jakeigos sharedowego liba, dzielimy się kodem. Normlanie komanrzy używamy tez dla sekcji todo/ albo jakiś ostrzezen/ dodatkowych informacji.

Vertical Formatting – odsperaowanie od siebie w pionie rzeczy powinny dzielic się na konteksty. Konteksty tez powinny być blisko siebie ejzeli sa ze sobą powiazane np. wywolana metoda A w metodzie B powinna znajdować się blisko niej. Zwykle najpierw ida pulibc -> private props -> public -> private methods.

# Function

Powinnismy mininmalizowac liczbe parametrów w funkcji. Zawsze staraj się unikac metod z liczbą paramertrów wieksza niż 3. Minimalizuja liczbe parametrow dobrze jest mieć w pamięci tez kolejność parametrow tak by cos było intuicyjne np. login i pass po sobie <- to jest dość ituicyjne Dobry przykład, zamiast robic metode która przyjuje emial i pass mamy obiekt i metode w srodku która nie przyjmuje zadnych arguemtnow tylko processuje na obieckie w którym jest.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

Żeby rozwiazac problem z kolejnoscia parametrow jakby moglaby nie być intuicyjna to można przekazac dict/ mape z nazwa propek wtedy z niej wynika bezpośrednio co jest czym i nie wymaga korzystania z podpowiedzi IDE

Obraz zawierający tekst, zrzut ekranu, Oprogramowanie multimedialne, oprogramowanie

Opis wygenerowany automatycznie

Wyjątkiem może być metoda która w swojej nazwie definuje ze może uzyc duza liczbe parametrow Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

NIGDY nie mutuj obiektow które przekazujesz w metodach, chbyua ze to wynika z opisu np. addId(user) <- choć to tez jest srednie. Najlepiej mieć klase User i w niej metode kotra dodawalaby takie id.

Piszac funckje staraj się pisać funkcje tak by zajmowały możliwe najmniej miejsca. Splituje metode możliwe na mniejsze metody z pojedyncza odpowiedzialnoscia

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

W progrmaowaniu rozoniaczy level abstrakcji poziomy -> high level np. isEmail(email) <- metoda która to realizuje w srodku logika, wyzsza abstrakcja. Low level <- wykorzystując niskopoziomowe api metode wykonujemy operacej email.includes(„@”)

Obraz zawierający tekst, zrzut ekranu, Czcionka, symbol

Opis wygenerowany automatycznie

Przy splitowaniu funkcji można zastosować sprawdzanie poprzez:

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

W kontekście rozbijania do małych metod ważne jest reuseability co rralizuej zasade DRY – don;t repeat yourself. Nie pisze tego samego kodu w kilku miejscach. Pisz go tylko raz.

Piszac funkcje staraj się pisać pure funkcje czyli z przewidywalnym outputem. Np. funkcja która nie jest pure bd generowała losowy rand(). Not pure funkcje maja tez side effecty – np. tworząc usera tworzymy dla niego sesej, stworzenie sesji zmieni stan całego programu/ systemu. Takie side efekty albo rand idki sa potzrebne ale staraj się je robic spodziewając się takich stanów. Jeżeli spodziewasz się side efektu jakiegoś w twoej funkcji zastanow się czy z nazwy tej metody która tor obi wynika taki side efekt czy nie, jeżeli nie to warto ta logike by przenieść gdzies. Przykład później gdzie lgoowanie nie jest w scopie isValid tylko sprawdzane jest poza nim bo to logowanie to side effect błędu nie jest to czescia logiki zawartej w nazwie metody. Processowanie to metoda która wywoluje to isValid tam powinien być log

**Obraz zawierający tekst, zrzut ekranu, Czcionka, oprogramowanie

Opis wygenerowany automatycznie**

**Przykład na wyodbrnienie side effektow fajny jest w unit-testing-clean.js pliku gdzie jest to pokazane ze testy pomagają wyodbrnic pojawianie się side effectowi potrzebe refactora.**

# Control structures and errors (if statements)

KOD dla skejci w katalogu: exampleForFunctionManyIfStatments

Nalezy unikac głeboko zagniezdzonej struktury kontroli (wiele ifów/ forow). By uniknąć takie zagdzniezadnie należy uzywac facotry functions lub polimoryfzmu.

Piszac checki w if statementach staraj się robic je pozytywnymi bo lepiej się czytac czyli

Uzywaj struktury guarda – czyli jeżeli masz jakiś if statment od którego zależy wykonanie całego kodu poniżej to możesz ten kod zwracać i zrobić metode typu guard. Takie guardy umozliwaja unikniecie wykonywania nie potrzegnego kodu i podnosza czytelność. Zwroc uwagę ze jest tam if (! 🡨) negacja. Zauwaz tez ze guard może być ale nie musi w tej samej metodzie.

Obraz zawierający tekst, zrzut ekranu, Czcionka, linia

Opis wygenerowany automatycznie

Budujac kod zamiast ifkow jak cos ma się wywalić to używaj errorow throwuj exceptiony/ custom errory. Catchuj i wyświetlaj w nim bledy. No i chba nie ma nic złego zeyb tak throwniety expcetion wyswietlic i processowac potencjalnie dalej, jezlei to możliwe. Dobra pakrtyka jest jednak by to co jest w try catchu było jakby w scopie całej metody, czyli nie wychodziło poza nią. Czyli jeżeli mamy for { } i w nim try catch to ten try catch powinien być w metodzie innej .

Jeżeli mamy jakiś duplikacyjną logiek która się powtarza to można dodac factory funkcje. Przykład później gdzie zaleznie od tego co jest w transctions inne metody trafiają do dicta i będą wywoływane z procesora. Unikamy w ten psosob sprawdzania w kilku miejscach czy cos jest typu cred card, paypal, plan.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Jeżeli jakies wartosic się powtrzaja i sa stałe w wielu ifach warto rzowazyc globbalny definicje enums, lub constansa.

# Classes, objects, data containers

Object-oriented principle (SOLID, law of demeter), polymorphism

**TODO: look for OOP course info.**

Roznica miedzy obiektem a data contianerem. Obiekt – ma prywatne wartości propertiesy, publiczne API metody. Obikety skupijaa się na bizneswoej logice (w OOP). W obiektach perferowan jest abstrakcja nad konkretnoscia. Data cotianer/ data strucute – to publiczny properitesy, prawie zawsze brak API (metod). Data structure skupiają się na transporcie i przechowywaniu danych. W Data Cotinaerach mamy konkretne dane.

## Polimorfizm

W folderze polimorfizmExample jest kod

Polimoryphsm – uzywany w kontkesie klasy, jeżeli mamy kilka zwrotek, gdzie kod się duplikuje, ale to co jest wywoływane się rozni. Warto zrobić wtedy więcej wyspecjalizowanych klas które przejma ta logike wyolania a logika warunkowa korabyla np. zduplikowana będzie hadnlniet’a np. mmay taka klase i metody

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

W efekcie rozbijamy to na 3 klasy Express Insured Standard i extendujemy je o Delivery base klasę. W efkei mamy klasy które wywoluja intersujące nas konkretne metody

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Czcionka

Opis wygenerowany automatycznie

W wyolaniu wtedy zrobimy ifa i zaleznie od sytuacji inna instancje zrobimy. Wtedy ta logika wyleciała z klasy co jest plusem

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Warto tez dodac interfejs do takich klas z metodami które sa w jego scopie.

Obraz zawierający tekst, Czcionka, zrzut ekranu

Opis wygenerowany automatycznie

## Klasy

Zasada klasy powinny być małe, raczej wolimy kilka mały klas niż duże klasy.

W klasach koncepcja zakłada wykonanie jednej czynności, w przypadku klas koncept jest inny. Klasa powinna mieć pojedynczą odpowiedzilanosc – single-rsponsibility principle (SRP). Pojedyncza odpowiedzialności odpowiada odpoweidzialnoscia w ramach jakiegoś obiektu np. Customer -> i akcje z nim związane.

Spojnosc w ramach klasy, dla funkcji to były poziomy abstrakcji high/ low. High który odowlywal się do abstrkacyjnej metody nazwanej która była latwa w odczycie co robi, skladala się z dalszej logiki. Gdize low level orientował się wokół niskopoziomowych funkcji jezyka jak .includes(). W kontekście klas mówimy o „Cohesion” [Kohiżin] – odpowiada to na pytania   
W jakim stopniu metody klasy wykorzystują jej propertiesy? -> gdy kazda metoda używa wszystkich to mamy wysoką spójność (cohesion), gdy nie używa class properites (no cohesion) to mówimy o data structue/ cotainerze z utility metodami.  
Złotym środkiem jest high cohesive nie trzeba uzywac wszystkich propek, ale staramy się by zuywal jak najwięcej

## Law of Demeter & Tell don’t ask

Unikanie w ramach obiektow składni: this.x.y.z zagniedzania takiego (chbya ze używamy data containera wtedy dozowlone). Trzymanie się tej zasady to wypleninie princpile of least knowledge – czyli nie zalezenie od wewnętrznych „obcych” form innych obiektow których bezpośrednio nie znasz. By wypelnic law of Demeter, to kod w metodzie powinien mieć dostep do wewnętrznych internali (props i metod) – czyli takich do których obiekt przynależy, obiektów które obiekt stor’euje jako propy w tym obieckie, obiektow które otrzyma poprzez parametr metody, obiektow które wywtorzy w metodzie.

Przykład złamanie kodu dla date który korzysta z obiektu lastPurchase wew. Cusotmera:

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

Rozw byłoby zorbienie metody w Customer klasie która zwracalby ten zagniedzony obiekt – jest to jednak rozw. Niedealne. Dla tego ze łamie zasadę „Tell, Don’t ask!” czyli zasady która ma mowic klasie co ma robic, a nie pytac ją o to.

Obraz zawierający tekst, Czcionka, zrzut ekranu

Opis wygenerowany automatycznie

Spelnieeniem tej zasdy tell dont ask byłoby zamian metody deliverPurchaseBYdate na deliverPurchasei przekazanie ifnoramcji o last purchase wtedy nie musimy się zastanawiać po co nam ta data tam przekzywana i co ona tam robi.

Obraz zawierający tekst, zrzut ekranu, Czcionka, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

## SOLID

S – Single Responsiblity  
O – Open-Closed  
L – Liskov Subsitution  
I – Interface Segregation  
D – Depenednecy Inversion

S - klasa nie powinna sie zmeiniac dla wiecej niz jednego powodu. Ponizej mamy dwie inne odpowiedzilanosci dla tego jest to złamanie zasady. Porszuaja inna logike nie zwiazana ze sobą Obraz zawierający tekst, Czcionka, linia, numer

Opis wygenerowany automatycznie

To np. nie powinno łamac bo porusza się w ramach tej samej odpowiedzilanosci autentykacji uzytkownika

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

O – class powinna by c otwarta na extension zamknieta na modyfikacje. W tej zasadzie majac zestaw metod gdbysmy chceli dodac kolejna i wymusialoby to zmiane poprzednich oznacza to złamanie tej zasady

Obraz zawierający tekst, zrzut ekranu, Czcionka, oprogramowanie

Opis wygenerowany automatycznie

Rozw byłoby polimorfizm i klasa i exentedowanie base klasy – extension to te klasy kotre dodajemy do base Classy.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Opis wygenerowany automatycznie

L – Liskov susbstituion obiekty powinny być możliwe zastepywalne przez instancje subklas bez zmian zachowania. Oznacza to ze na subklasie możliwe jest wywloanie metody dostępnej w baseklasie i z rezultatem spodziewanym tak jakby bazować tylko na baseklasie. Przykład poniżej nie jest dobry bo jakbys dodali teraz pingiwna to on nie może latac wiec metoda fly nie dzial ajak powinna. Wniosek ze nasze rozbicie na baseKlase i subKlasy było nieprawidlowe Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Obraz zawierający zrzut ekranu, tekst, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

Prawidlowe powinno być stworzenie FlyingBird klasa która zaciagalby dopiero bird jako najwyzsza abstrakcje

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

I – Interface Segregation – wiele konkrentych interfejsow klienckich jest lepsze niż jeden general purpose inerfejs. Przykład ponizje tutaj ważne jest rozbicie na dwa interfejsy Database i Remote bo connect nie istnieje w kontkescie inmemoerydatabase.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

D – dependency Inversion -powinnismy polecagac na abastrakcji nie na konkretyzacji. Biorac przykład wyżej z databasami gdybyśmy tworzyli instancje Appki kotra ma się polaczyc z baza danych można by dac tam SQL || InMemory w konsturktorze i jezlei ejst SQL to if i tam connect do niej. Takie podejście nie spelnia dependency inversion. By to poprawić konsturktor powinien przyjmować abstrakcje czyli Database. Logika z połaczania się powinna by cwywalona z konsturktora i przenisieona do odpowiedzialności która zostanie spełniona na wyższym poziomie.Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Czcionka

Opis wygenerowany automatycznie