



12. - 13. 11. 2024 Praha / ONLINE

Hybridní konference
pro **Java vývojáře**



Clean Code a best practices v době Copiloty

Milan Lempera | JavaDays 2024



O mně

- <https://www.linkedin.com/in/milanlempera/>
- ~ 15 let se věnuji vývoji převážně webových aplikací
- dříve školení a konzultace SPA (Angular.js, Angular, React)
- Tech Lead v Commity.cz
- Inspiraci hledám v Clojure
- <https://www.meetup.com/gdgjihlava/>
- robohrani.cz

commity.cz

Jak může vypadat použití Copilota?

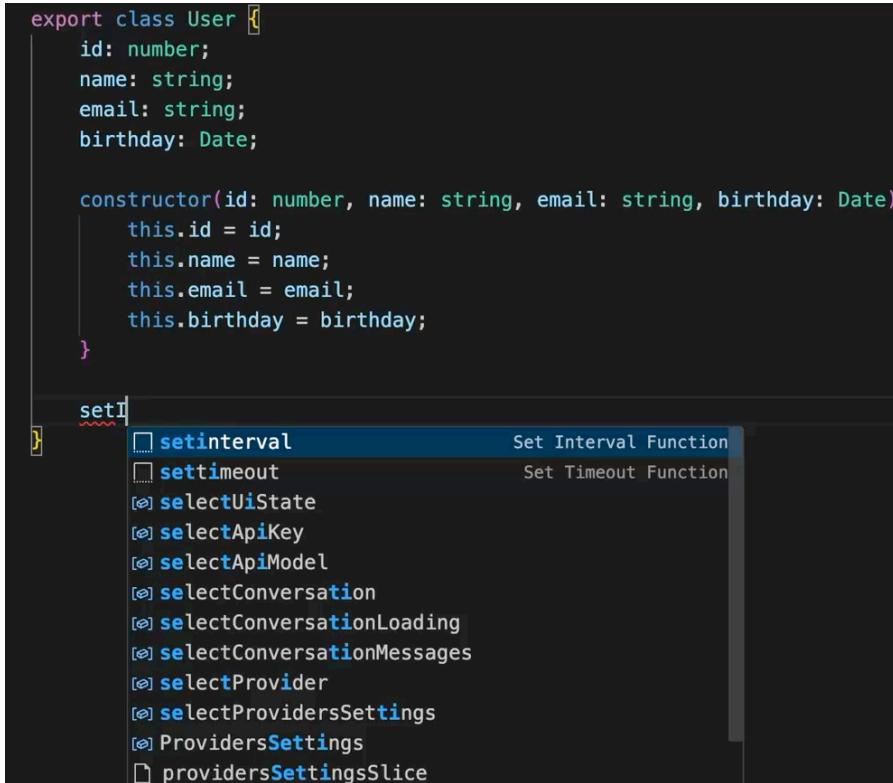
```
export class User {  
    id: number;  
    name: string;  
    email: string;  
    birthday: Date;  
  
}  
|
```

Jak může vypadat použití Copilota?

```
export class User {
    id: number;
    name: string;
    email: string;
    birthday: Date;

    constructor(id: number, name: string, email: string, birthday: Date) {
        this.id = id;
        this.name = name;
        this.email = email;
        this.birthday = birthday;
    }

    setI
}
```



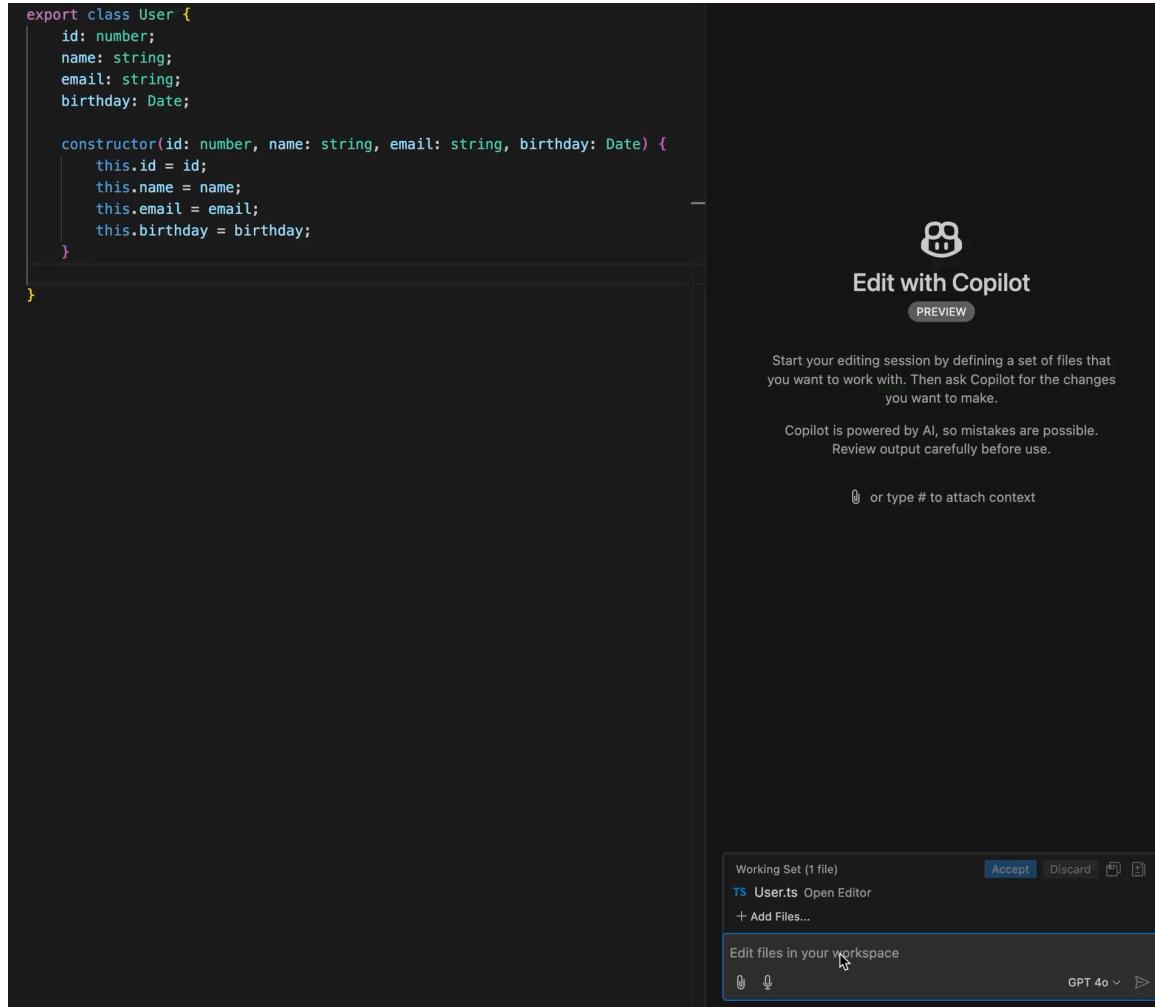
A screenshot of a code editor showing a completion dropdown menu. The code above defines a class 'User' with properties 'id', 'name', 'email', and 'birthday'. A method 'constructor' is defined. Below it, a method 'setI' is partially typed, and a completion dropdown is open. The dropdown contains several items:

- setinterval Set Interval Function
- settimeout Set Timeout Function
- selectUiState
- selectApiKey
- selectApiModel
- selectConversation
- selectConversationLoading
- selectConversationMessages
- selectProvider
- selectProvidersSettings
- ProvidersSettings
- providersSettingsSlice

Jak může vypadat použití Copilota?

```
export class User {  
    id: number;  
    name: string;  
    email: string;  
    birthday: Date;  
    < 1/2 > Accept Tab Accept Word ⌘ ⌘ ...  
    // Define a new User class  
  
    constructor(id: number, name: string, email: string, birthday: Date) {  
        this.id = id;  
        this.name = name;  
        this.email = email;  
        this.birthday = birthday;  
    }  
}
```

Jak může vypadat použití Copilota?



Jak může vypadat použití Copilota?

- Trigger suggestion
- Accept
 - suggestion
 - next word
 - next line
- Instructions in comments
- Instructions in chat

Postupné zlepšování

Smysluplná jména

Souběžnost *Vývoj*

Třídy *Formátování*
Systémy

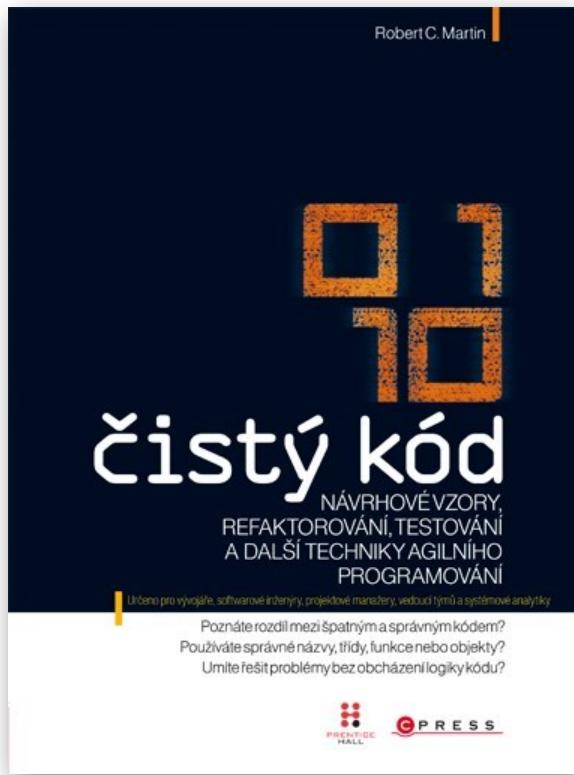
Funkce

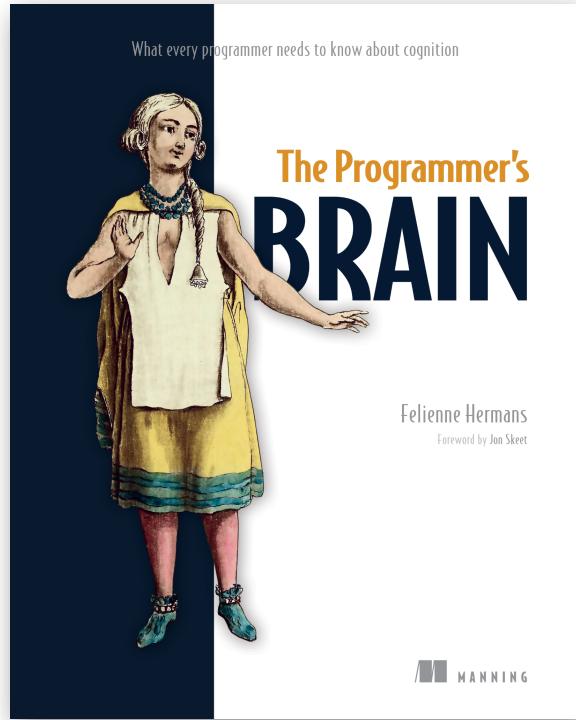
Komentáře
Hranice

Objekty a datové struktury

Jednotkové testy

Zpracování chyb





<https://www.manning.com/books/the-programmers-brain>

Postupné zlepšování

Smysluplná jména

Souběžnost Vývoj

Třídy Formátování
Systémy

Funkce

Hranice Komentáře

Objekty a datové struktury

Jednotkové testy

Zpracování chyb

Různé druhy nejasností v kódu

Nedostatečné znalosti 🤷

```
2 2 2 2 2 + n
```

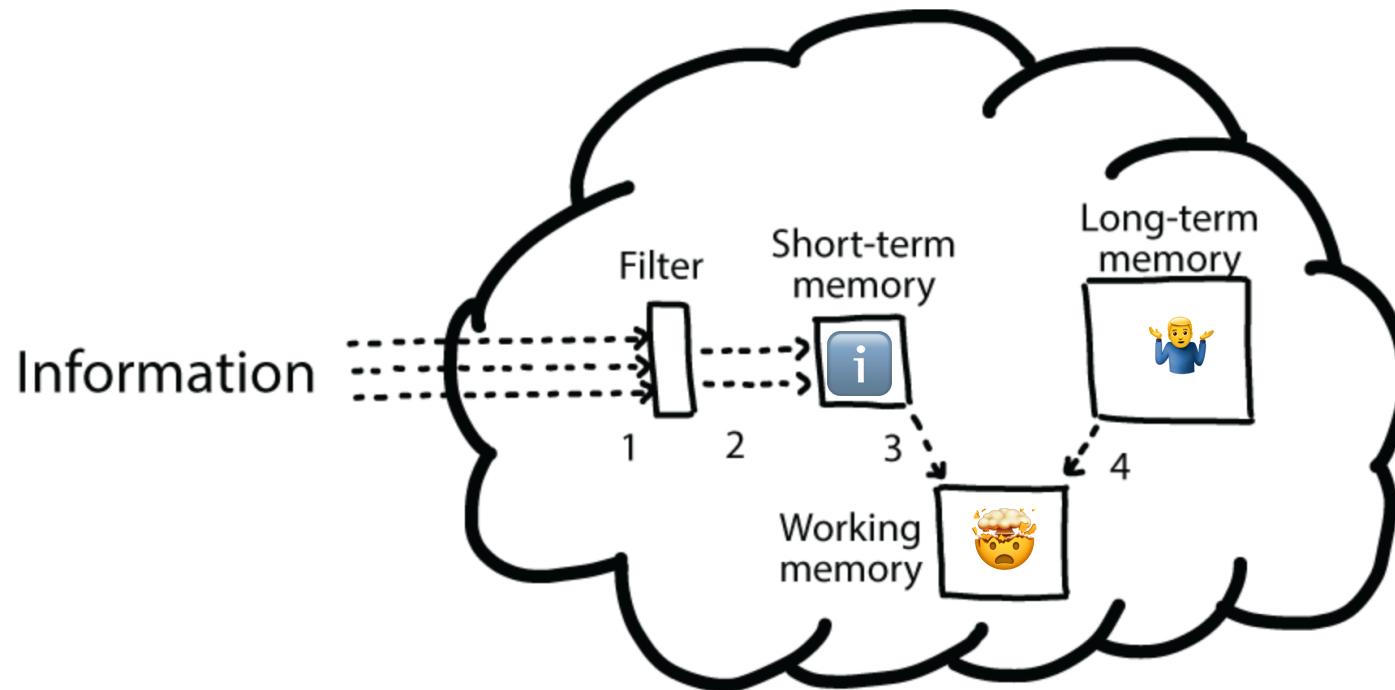
Nedostatek informací ⓘ

```
public class BinaryCalculator {  
    public static void main(Integer n) {  
        System.out.println(Integer.toBinaryString(n));  
    }  
}
```

Nedostatek “výpočetního výkonu” 😬

```
1 LET N2 = ABS (INT (N))  
2 LET B$ = ""  
3 FOR N1 = N2 TO 0 STEP 0  
4     LET N2 = INT (N1 / 2)  
5     LET B$ = STR$ (N1 - N2 * 2) + B$  
6     LET N1 = N2  
7 NEXT N1
```

Kognitivní proces



Prohlédněte si tento kód a zkuste si ho zapamatovat

```
public class InsertionSort {  
    public static void main (String [] args) {  
        int [] array = {45,12,85,32,89,39,69,44,42,1,6,8};  
        int temp;  
        for (int i = 1; i < array.length; i++) {  
            for (int j = i; j > 0; j--) {  
                if (array[j] < array [j - 1]) {  
                    temp = array[j];  
                    array[j] = array[j - 1];  
                    array[j - 1] = temp;  
                }  
            }  
        }  
        for (int i = 0; i < array.length; i++) {  
            System.out.println(array[i]);  
        }  
    }  
}
```

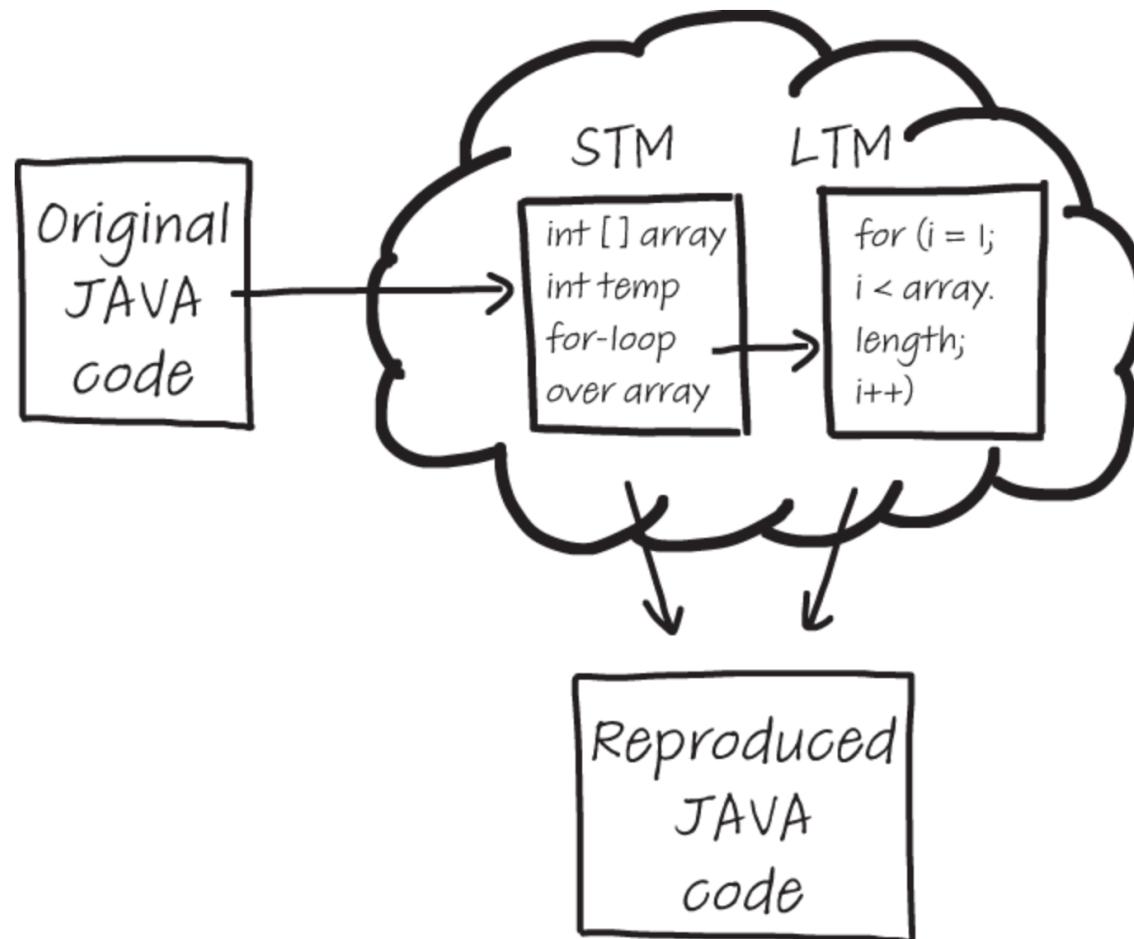
Prohlédněte si tento kód a zkuste si ho zapamatovat

- class InsertionSort
- int [] array = {45,12,85,35,89,39,...}
- int tpm
- for-loop i
- for-loop j
- if (... < ...)
- swap
- print in for

Co se děje ve vašem mozku?

```
public class InsertionSort {  
    public static void main (String [] args) {  
        int [] array = {45,12,...};  
        int temp;  
        for (int i = 1; i < array.length; i++) {  
            for (int j = i; j > 0; j--) {  
                if (array[j] < array [j - 1]) {  
                    // swap j with j - 1  
                    temp = array[j];  
                    array[j] = array[j - 1];  
                    array[j - 1] = temp;  
                }  
            }  
        }  
        //print array  
        for (int i = 0; i < array.length; i++) {  
            System.out.println(array[i]);  
        }  
    }  
}
```

Co se děje ve vašem mozku?



Proč je čtení neznámého kódu těžké?

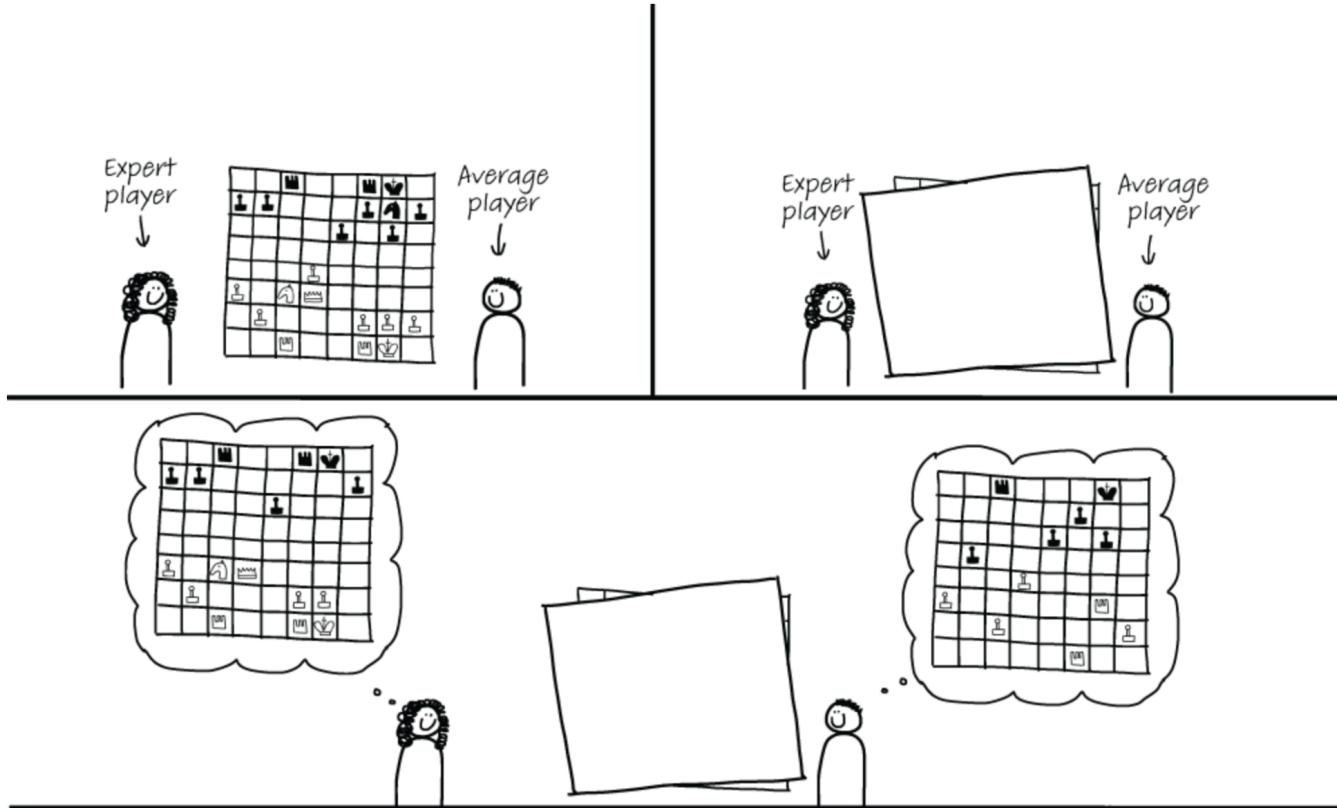
```
void execute(int x[]){
    int b = x.length;

    for (int v = b / 2 - 1; v >= 0; v--)
        func(x, b, v);

    // Extract elements one by one
    for (int l = b-1; l > 0; l--)
    {
        // Move current to end
        int temp = x[0];
        x[0] = x[l];
        x[l] = temp;

        func (x, l, 0);
    }
}
```

Lze trénovat krátkodobou paměť?



Chunking theory - De Groot's chess experiment

Kolik slov dokážete zapamatovat?

θ|z̄ φ\c<φ ↴|z̄)

abk mrtpi gbar

cat loves cake



Chunking ve zdrojovém kódu

```
public class InsertionSort {  
    public static void main (String [] args) {  
        int [] array = {45,12,...};  
        int temp;  
        for (int i = 1; i < array.length; i++) {  
            for (int j = i; j > 0; j--) {  
                if (array[j] < array [j - 1]) {  
                    // swap j with j - 1  
                    temp = array[j];  
                    array[j] = array[j - 1];  
                    array[j - 1] = temp;  
                }  
            }  
        }  
        //print array  
        for (int i = 0; i < array.length; i++) {  
            System.out.println(array[i]);  
        }  
    }  
}
```

Pojmenování



Pojmenování

- Jsou smysluplná
 - Dají se přečíst/vyslovit
 - Jdou vyhledat
 - Nejsou zakódovaná
-
- Používají pojmy z domény řešení
 - Používají “oborové pojmy”
 - Mají jedno slovo pro jeden pojem
 - Nelžou



Čitelnost



Kontext

Pojmenování

The screenshot shows a ChatGPT interface with the title "ChatGPT o1-preview >". A message from the AI says: "Thought for 5 seconds" followed by "Pro mě je v promptu důležité několik klíčových aspektů:" and a numbered list of 5 tips. Below the list, the AI adds: "Doufám, že ti tyto tipy pomohou psát efektivní prompty!" At the bottom, there's a message input field labeled "Message ChatGPT" and a send button.

Thought for 5 seconds

Pro mě je v promptu důležité několik klíčových aspektů:

- Jasnost a přesnost:** Čím konkrétněji a jednoznačněji je otázka formulována, tím lépe mohu poskytnout přesnou odpověď.
- Kontext:** Poskytnutí relevantního kontextu mi pomáhá lépe pochopit tvé potřeby a přizpůsobit odpověď tvým očekáváním.
- Specifické požadavky:** Pokud máš konkrétní požadavky na formát, délku nebo styl odpovědi, je užitečné je zmínit předem.
- Jednoduchost:** Vyhnutí se nadbytečným informacím nebo složitým formulacím může pomoci k lepšímu pochopení otázky.
- Cíl nebo účel:** Uvedení toho, co od odpovědi očekáváš nebo jaký problém se snažíš vyřešit, mi umožňuje poskytnout relevantnější informace.

Doufám, že ti tyto tipy pomohou psát efektivní prompty!

Message ChatGPT

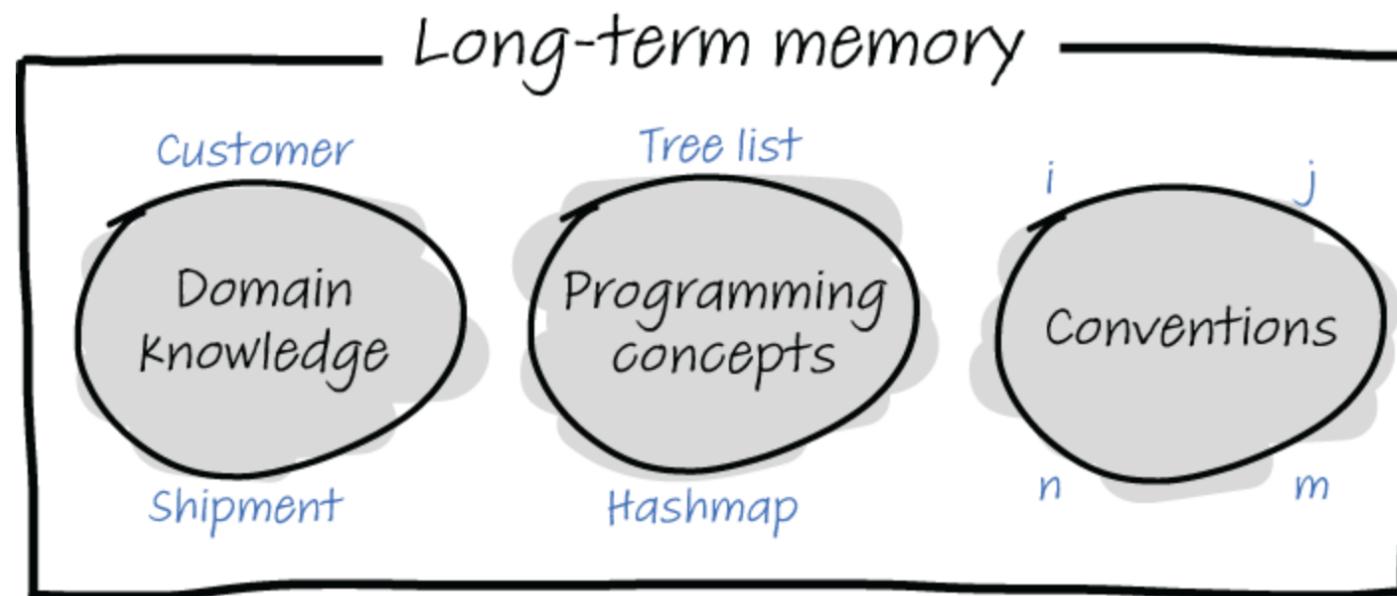


Čitelnost



Kontext

Pojmenování





Majáky v kódu

```
# A class that represents a node in a tree

class Node:
    def __init__(self, key):
        self.left = None
        self.right = None
        self.val = key

# A function to do in-order tree traversal
def print_in_order(root):
    if root:

        # First recur on left child
        print_in_order(root.left)

        # then print the data of node
        print(root.val)

        # now recur on right child
        print_in_order(root.right)

print("Contents of the tree are")
print_in_order(tree)
```

Pojmenování - výzkumy

Jeden ze čtyř code review obsahoval připomínky týkající se pojmenování.
Připomínky k názvům identifikátorů se objevily v 9 % případů.

Počáteční pojmenování má dlouhodobý dopad.
V jedné code base se pojmenování nezlepší s věkem kódu.

Kód se špatnými názvy obsahuje více chyb.

Účastníci našli v průměru o 19 % více chyb při čtení programů,
v nichž identifikátory tvořila slova, než písmena a zkratky.

Mezi písmeny a zkratkami nebyl zjištěn významný rozdíl v rychlosti.

Formatování



Formátování pomáhá chunkingu

```
public class InsertionSort {  
    public static void main (String [] args) {  
        int [] array = {45,12,...};  
        int temp;  
        for (int i = 1; i < array.length; i++) {  
            for (int j = i; j > 0; j--) {  
                if (array[j] < array [j - 1]) {  
                    // swap j with j - 1  
                    temp = array[j];  
                    array[j] = array[j - 1];  
                    array[j - 1] = temp;  
                }  
            }  
        }  
        //print array  
        for (int i = 0; i < array.length; i++) {  
            System.out.println(array[i]);  
        }  
    }  
}
```

Formátování

Vyřešený problém

Namísto diskusí o formátování máme “opinionated” nástroje

- IDE umí nastavit formátování podle konfiguračním souboru
- Prettier, eslint
- biome.js

Existují pluginy pro prettier na různé jazyky - php, ruby, kotlin, java, rust, elm, ...

Každý jazyk má specifické formátry (ktlint, ktfmt, ...)

Paradigmata



Paradigmata

OOP

- Zapouzdření (data + kód)
- Dědičnost
- Polymorfismus
- Abstrakce

Další principy:

- S.O.L.I.D
- Návrhové vzory
- Demeters rule
- Tell, don't ask

...

FP

- Pure functions
- Imutabilita
- High order functions
- Rekurze
- Lazy evaluation

Rozlišujte

- Data
- Calculations
- Actions

Data - Calculations - Actions

Actions

- Časově závislé
- Počet volání je důležitý
- Např. odeslání emailu, zápis do DB,...

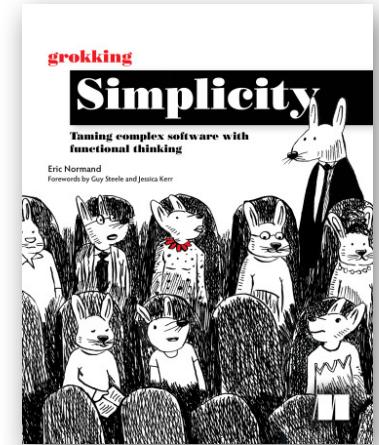
Calculations

- Výpočty, filtrace, mapování dat
- Nejsou závislé na čase
- Počet volání nehraje roli

Data

- Faktické informace
- Vyžadují interpretaci
- Serializovatelné

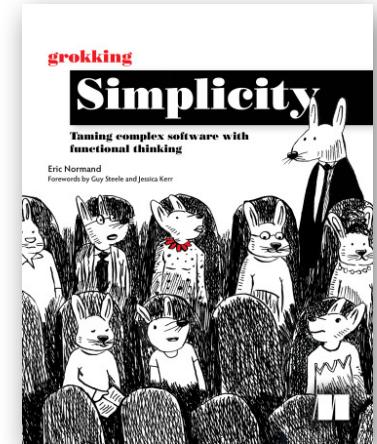
<https://www.manning.com/books/grokking-simplicity>



Data - Calculations - Actions

Actions

- Bezpečná změna stavu
- Garantované pořadí
- Spustí se ve správný čas a správném počtu



Calculations

- Správnost "logiky"
- Testovací strategie
- Výsledku výpočtu můžeme nahradit hodnotou (Referential transparency)

<https://www.manning.com/books/grokking-simplicity>

Data

- Správné struktury pro efektivní přístup a zpracování
- Obsahují vše co je potřeba pro operace?
- Dlouhodobý záznam

App state

```
const state = {
  todos: [
    {label: 'koupit mléko', complete: false}
    {label: 'umýt auto', complete: true}
  ]
  filter: {
    completed: false
  }
}
```

transformace

```
getTodosByFilter(state)
```

renderování

```
const Todos = ({visibleTodos}) => <List items={visibleTodos} />
```

Funkce



Funkce

```
function sendOrderConfirmation(order) {  
    const emailTemplate = `  
        Dear {{customerName}},  
        Thank you for your order #{{orderId}} ....`;  
  
    const orderDate = new Date(order.orderDate).toLocaleDateString();  
  
    const orderItems = order.items.map((item, index) => {  
        return `${index + 1}. ${item.name} - Quantity: ${item.quantity} - Price: ${item.price}`;  
    }).join('\n');  
  
    const emailContent = emailTemplate  
        .replace('{{customerName}}', order.customerName || 'Customer')  
        .replace('{{orderId}}', order.orderId)  
        .replace('{{orderDate}}', orderDate)  
        .replace('{{orderItems}}', orderItems)  
        .replace('{{totalPrice}}', order.totalPrice.toFixed(2));  
    Různá úroveň abstrakce  
  
    sendEmail(order.customerEmail, 'Your Order Confirmation', emailContent);  
}
```

Data uvnitř funkce

Řeší více věcí

Funkce

```
const confirmationEmailTemplate = `  
Dear {{customerName}},  
Thank you for your order #{{orderId}} ....`;  
  
function prepareOrderEmail(confirmationEmailTemplate, order) {  
    const emailContent = ...  
  
    return {  
        subject: 'Your Order Confirmation',  
        body: emailContent  
    }  
}  
  
function sendOrderConfirmation(order) {  
    const email = prepareOrderEmail(confirmationEmailTemplate, order)  
    sendEmail(order.customerEmail, email.subject, email.body);  
}
```

Clean code:

- Malá
- Dělá jen jednu věc
- Má jednu úroveň abstrakce

Eric Normand

- Data
- Calculations
- Actions

Počet argumentů funkce

Objektově orientované programování

- 0 - ideální
pracujeme s daty objektu
- 1
`fileExists(fileName),
fileOpen(fileName)`
- 2
`assertEquals(expected, actual)`
- 3
`assertEquals(message, expected, actual)`
- Výstupní argumenty
Ideálně žádný

Funkcionální programování

- 0 - podezřelá
nejspíš impure funkce, Action
čte globální data?
- 1
`fileExists(fileName),
fileOpen(fileName)`
- 2
`getIn(path, object)`
- 3
`getIn(path, object, defaultValue)
setInPath(path, value, object)`
- Výstupní argumenty
funkce typicky něco vrací,
pokud ne, jde o Action

Argumenty funkce

Boolean parametrům se vyhněte

```
const price = applyDiscount(order, true);

const price = applyDiscount(order, { isVip: true });

...
const price = applyDiscount(order, { isPartner: true });
const price = applyDiscount(order, { is...: true });

const price = applyDiscount(order, { priceList: PriceList.VIP });
const price = applyDiscount(order, PriceList.VIP);
```

Argumenty funkce

Preferujte object argument pro související data

```
const packageSize = derivePackageSize(width, height);  
  
const packageSize = derivePackageSize(packageDimensions);  
  
const packageDimensions = { width: 123,  
                           height: 456,  
                           depth: 789 };  
  
const packageSize = derivePackageSize(packageDimensions);
```

Funkce - Single return

```
function calculateDiscount(price, coupon) {  
    let discount = 0;  
  
    if (coupon === "DISCOUNT10") {  
        discount = price * 0.10;  
    } else if (coupon === "FREESHIPPING") {  
        discount = 0;  
    } else if (coupon === "BLACKFRIDAY") {  
        discount = price * 0.50;  
    } else {  
        discount = 0;  
    }  
  
    return discount;  
}
```

Funkce - Early return

```
function calculateDiscount(price, coupon) {  
    if (coupon === "DISCOUNT10") {  
        return price * 0.10;  
    } else if (coupon === "FREESHIPPING") {  
        return 0;  
    } else if (coupon === "BLACKFRIDAY") {  
        return price * 0.50;  
    } else {  
        return 0;  
    }  
}
```

Potřebujeme **else if** ?

Funkce - Early return

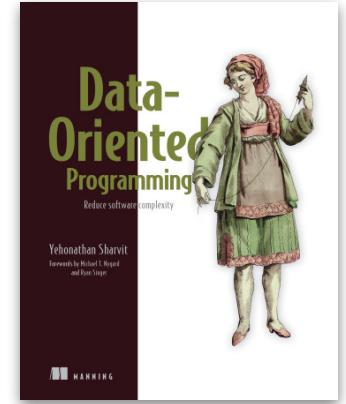
```
function calculateDiscount(price, coupon) {  
    if (coupon === "DISCOUNT10") {  
        return price * 0.10;  
    }  
    if (coupon === "FREESHIPPING") {  
        return 0;  
    }  
    if (coupon === "BLACKFRIDAY") {  
        return price * 0.50;  
    }  
    return 0;  
}
```

Objekty a datové struktury



Data-Oriented Programming

- Oddělte data a kód
- Data reprezentujte generickými strukturami
- Data jsou imutabilní
- Oddělte schéma dat od reprezentace



<https://www.manning.com/books/data-oriented-programming>

Klasické OOP

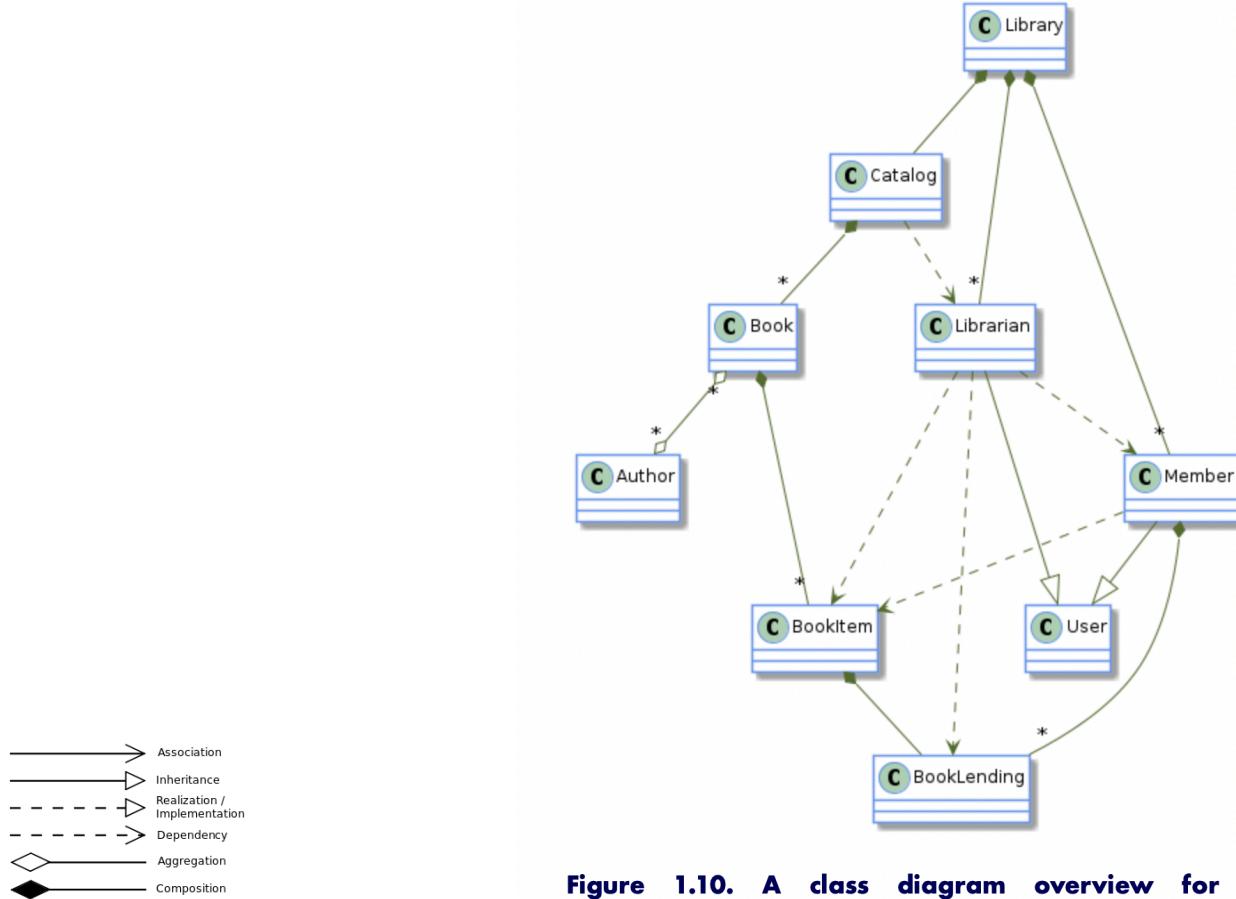


Figure 1.10. A class diagram overview for a Library management system

DOP - oddělte data a kód

```
class AuthorData {  
    constructor(firstName, lastName, books) {  
        this.firstName = firstName;  
        this.lastName = lastName;  
        this.books = books;  
    }  
  
    class NameCalculation {  
        static fullName(data) {  
            return data.firstName + " " + data.lastName;  
        }  
    }  
  
    class AuthorRating {  
        static isProlific (data) {  
            return data.books > 100;  
        }  
    }  
  
    var data = new AuthorData("Isaac", "Asimov", 500);  
    NameCalculation.fullName(data);  
    // → Isaac Asimov
```

```
function createAuthorData(firstName, lastName, books) {  
    return {  
        firstName: firstName,  
        lastName: lastName,  
        books: books  
    };  
}  
  
function fullName(data) {  
    return data.firstName + " " + data.lastName;  
}  
  
function isProlific (data) {  
    return data.books > 100;  
}  
  
var data = createAuthorData("Isaac", "Asimov", 500);  
fullName(data);  
// → Isaac Asimov
```

DOP - oddělte data a kód

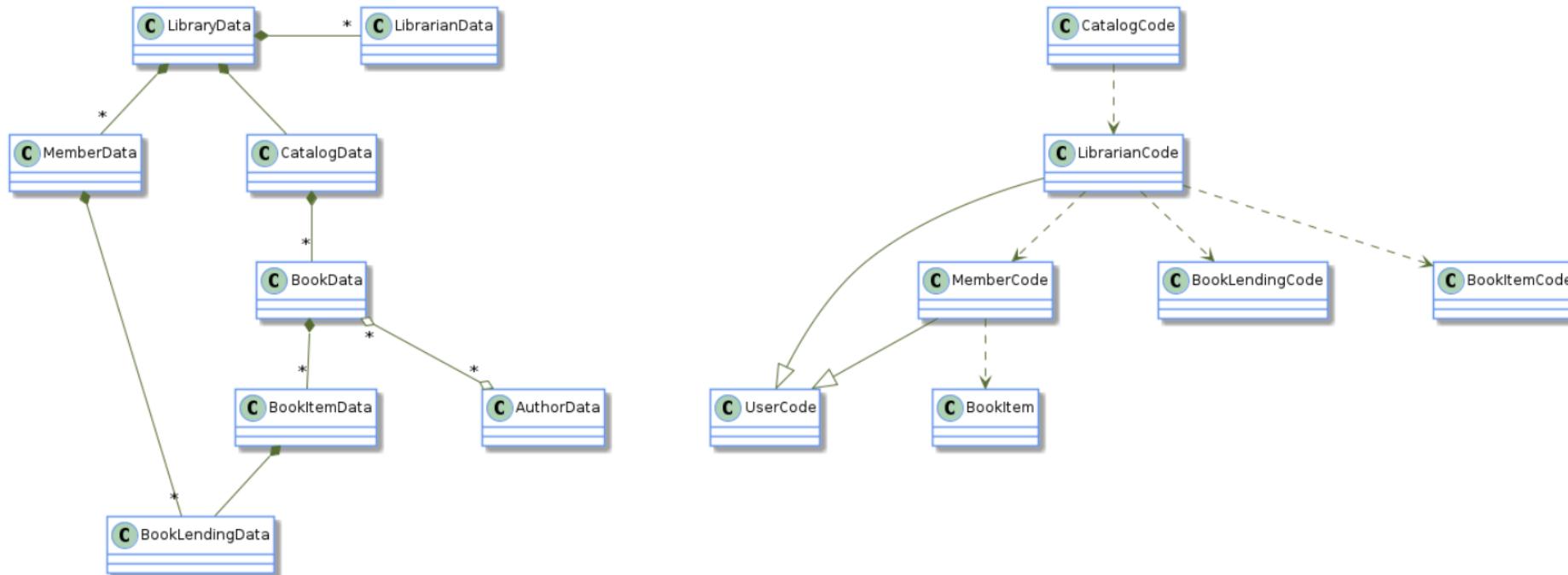
Výhody

- Kód lze opakovaně použít v různých kontextech
- Kód lze testovat izolovaně
- Systémy bývají méně složité

Cena

- Žádná kontrola nad tím, jaký kód má přístup k jakým datům
- Systémy jsou tvořeny více entitami

Více entit - méně komplexity



Oddělte schéma dat od reprezentace

{X} JSON Schema

```
var bookInfoSchema = {  
  "type": "object",  
  "required": ["title", "available"],  
  "properties": {  
    "title": {"type": "string"},  
    "available": {"type": "boolean"},  
    "subtitle": {"type": "string"},  
    "number_of_pages": {"type": "integer"},  
    "subjects": {  
      "type": "array",  
      "items": {"type": "string"}  
    },  
    "isbn": {"type": "string"},  
    "isbn_13": {"type": "string"}  
},  
};
```

Oddělte schéma dat od reprezentace

```
import { z } from 'zod';

const PersonSchema = z.object({
  id: z.number(),
  name: z.string(),
  age: z.number().gte(0).lte(150),
  email: z.string().email(),
});

const validationResult = PersonSchema.safeParse({
  id: 1,
  name: 'Alice',
  age: 25,
  email: 'alice@example.com',
});

if (!validationResult.success) {           // TS typy fungují správně
  console.log(validationResult.data.name, validationResult.data.email)
}
```

Oddělte schéma dat od reprezentace

```
const PersonWithAddressSchema = PersonSchema.merge(AddressSchema);
```

```
const BasicPersonSchema = PersonSchema.pick({
  name: true,
  email: true,
});
```

```
const PersonWithoutAgeSchema = PersonSchema.omit({
  age: true,
});
```

```
const stringOrNumber = z.union([z.string(), z.number()]);
```

Make impossible states impossible

```
fetchData<T>(url: string): Promise<{ data?: T; error?: string; isLoading: boolean }>
```

vrací objekt, který může volitelně obsahovat vlastnosti data, error a isLoading

- { isLoading: true }
- { isLoading: false, data: {...} }
- { isLoading: false, error: 'Some error' }

Ale také:

- { isLoading: true, data: {...} }
- { isLoading: true, error: 'Some error' }
- { isLoading: true, data: {...}, error: 'Some error' }

https://www.youtube.com/watch?v=lcgmSRJHu_8&ab_channel=elm-conf

Make impossible states impossible

```
type LoadingState = { status: 'loading' };
type SuccessState<T> = { status: 'success'; data: T };
type ErrorState = { status: 'error'; error: string };

type FetchState<T> = LoadingState | SuccessState<T> | ErrorState;

fetchData<T>(url: string): Promise<FetchState<T>>
```

Stavy, které nemohou nastat nejsou validní ani z pohledu typů.

https://www.youtube.com/watch?v=lcgmSRJHu_8&ab_channel=elm-conf

Model & typy

- Použití schémat posouvá váš focus na data
- Uvažujte o tom co je “zdroj pravdy”
 - Schéma modelu
 - DB schéma
 - BE schéma
- Ze “zdroje pravdy” odvozujte
 - Další omezení schématu
 - Schémata pro konkrétní případy
 - Typy
- Schémata uplatňujte na hranici systému, např.
 - Volání REST api
 - Form data
 - Databáze (pokud ji sdílíte s jinou službou)
- Zvažte jestli použití schémat uvnitř systému nepřinese další benefity

Závěr

- Dbejte na srozumitelnost
- Dobré názvy pomohou tobě, kolegovi i umělé inteligenci
- Jednodušší je lepší - data > calculation > action
- Principy funkcionálního programování vedou k jednoduššímu kódu
- Odvozuj a generuj místo psaní
- Typy používej jako výhodu, ne jako bič
- Hledejte nástroje co udělají práci za vás
 - LLM nástroje jsou v této oblasti velmi schopné
- Napoprve to nebude - začněte a postupně vylepšujte

Odkazy

Knihy:

- [Clean Code](#)
- [Programmer's Brain](#)
- [Grokking Simplicity](#)
- [Data Oriented Programming](#)
- [Data-Oriented Programming in Java \(MEAP\)](#)

Video

- [Make impossible states impossible](#)



12. - 13. 11. 2024 Praha / ONLINE

Děkuji vám za pozornost