html academy
интерактивные
онлайн-курсы

Раздел 9:

# Типизированный JavaScript

# Статическая типизация

Приём в языке программирования, при котором тип переменной указывается во время объявления переменной и не может быть изменён позже

ru.wikipedia.org/wiki/Статическая_типизация

# Преимущества

medium.freecodecamp.com/why-use-static-types-in-javascript-part-1-8382da1e0adb

# Преимущества

– Помогает находить ошибки в программе до её запуска

medium.freecodecamp.com/why-use-static-types-in-javascript-part-1-8382da1e0adb

# Преимущества

– Помогает находить ошибки в программе до её запуска

– Улучшает читаемость кода

medium.freecodecamp.com/why-use-static-types-in-javascript-part-1-8382da1e0adb

# Преимущества

– Помогает находить ошибки в программе до её запуска

– Улучшает читаемость кода

– Фиксирует поведение методов и функций

medium.freecodecamp.com/why-use-static-types-in-javascript-part-1-8382da1e0adb

# Преимущества

– Помогает находить ошибки в программе до её запуска

– Улучшает читаемость кода

– Фиксирует поведение методов и функций

– Упрощает изменение и рефакторинг кода

medium.freecodecamp.com/why-use-static-types-in-javascript-part-1-8382da1e0adb

# Преимущества

– Помогает находить ошибки в программе до её запуска

– Улучшает читаемость кода

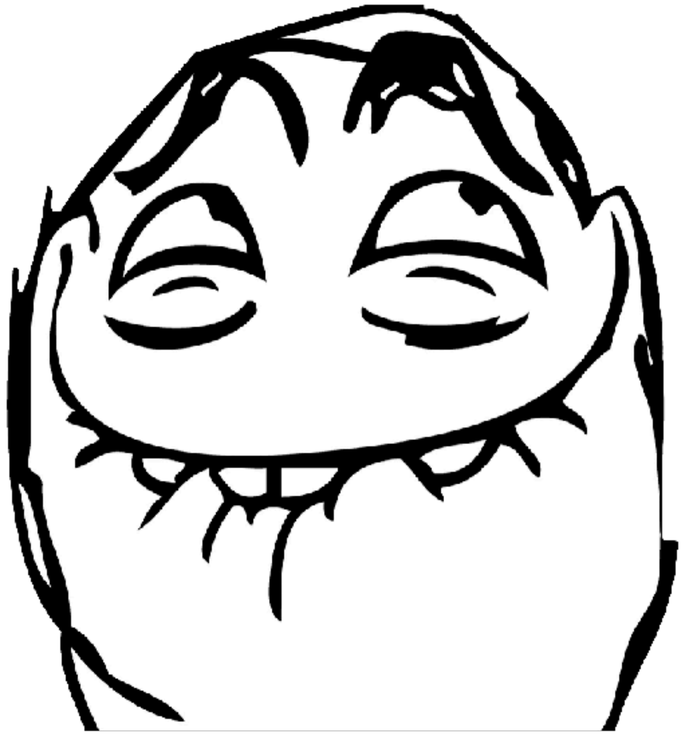
– Фиксирует поведение методов и функций

– Упрощает изменение и рефакторинг кода

– Улучшает работу «умных» редакторов кода

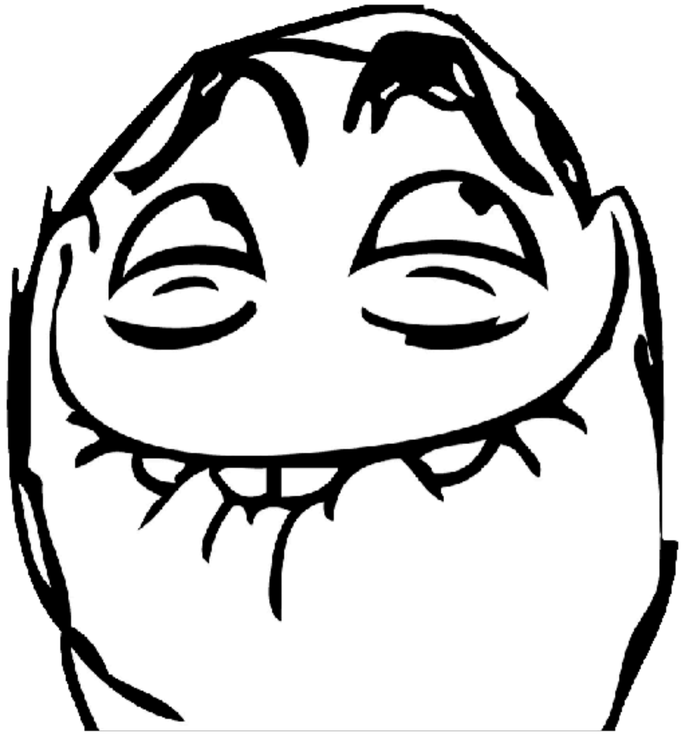
medium.freecodecamp.com/why-use-static-types-in-javascript-part-1-8382da1e0adb

# Недостатки



# Недостатки

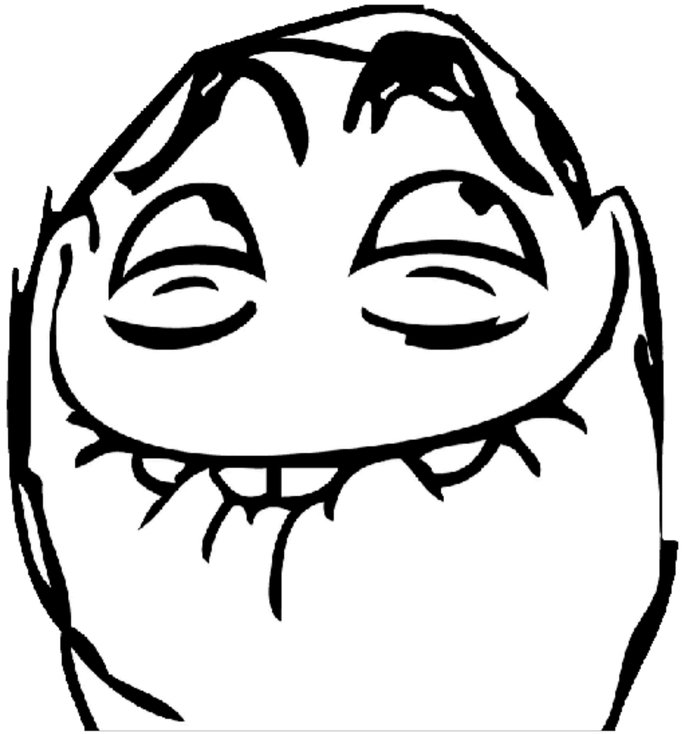
– Увеличивает количество кода



# Недостатки

– Увеличивает количество кода

– Нужно понимать в общих чертах как работает система типов



# Типизированный JavaScript

# Типизированный JavaScript

– Flow
  https://flow.org

# Типизированный JavaScript

– Flow
  https://flow.org

– TypeScript
  https://www.typescriptlang.org

# Типизированный JavaScript

- Flow
  https://flow.org
- TypeScript
  https://www.typescriptlang.org
- Kotlin JavaScript Compiler
  https://kotlinlang.org/docs/tutorials/javascript/kotlin-to-javascript/kotlin-to-javascript.html

# Типизированный JavaScript

- Flow
  https://flow.org
- TypeScript
  https://www.typescriptlang.org
- Kotlin JavaScript Compiler
  https://kotlinlang.org/docs/tutorials/javascript/kotlin-to-javascript/kotlin-to-javascript.html
- Scala.JS
  https://www.scala-js.org/

# Статическая типизация

На примере языка TypeScript

typescriptlang.org

# Объявление

```
const myNum: number = 12;

let name: string = `Строка`;

var isTrue: boolean = true;
```

# Вывод типа

```
const myNum = 12;

let name = `Строка`;

var isTrue = true;
```

# Специальные типы

```
const nullData: null = null;
const undefinedData: void = undefined;
const iCanBeAnything: any = `LALA2`;

let iCanBeAnythingToo;
iCanBeAnythingToo = `LALA2`;
iCanBeAnythingToo = 123;
```

# Массивы и объекты

```
const messages: Array<string> = ['hello', 'world', '!'];

const aboutMe: { name: string, age: number } = {
  name: 'Preethi',
  age: 26,
};

const namesAndCities: { [name: string]: string } = {
  Preethi: 'San Francisco',
  Vivian: 'Palo Alto',
};
```

# Функции

```
const calculateArea = (radius: number): number => {
  return 3.14 * radius * radius;
};

const printNumber = (number: number,
                     calculator: (number) => number = calculateArea) => {
  console.log(calculator(number));
};

printNumber(12);
printNumber(12, null);
```

# Именованные типы

```ts
type PaymentMethod = {
  id: number,
  name: string,
  limit: number,
};

const myPaypal: PaymentMethod = {
  id: 123456,
  name: 'Preethi Paypal',
  limit: 10000,
};

type Email = string;

const academy: Email = 'mail@htmlacademy.ru';
```

# Дженерики (*Generics*)

```
type KeyObject<T> = { key: T };

const numberT: KeyObject<number> = {key: 123};

const stringT: KeyObject<string> = {key: `Preethi`};

const arrayT: KeyObject<Array<number>> = {key: [1, 2, 3]};
```

# Объединения (*Join-type*)

```ts
const even: 2|4|6|8|10 = 10;
const stringOrNumber: string|number = `Hello!`;

const stringOrUndefined: string|void|undefined = undefined;

const nullOrUndefined: void = undefined;
```

# Возможно (*Maybe*)

```ts
type Maybe<T> = T|void|null;

let something: Maybe<string> = null;

let maybeNumber: Maybe<number> = 100;
```

# Необязательные параметры (*Optional*)

```
const callMe = (y:string, x?:number) => {
  console.log(y, x);
};

callMe(`param`);
callMe(`param`, undefined);
callMe(`param`, null);
callMe(100);
callMe();
```

# Как понять что делает функция *validate*

```
const {validate} = require(`./validator`);
const schema = require(`./schema`);

const data = {
  name: `Say my name`
};

validate(data, schema);
```

# Возможные варианты

# Возможные варианты

– Прочитать тело функции:

# Возможные варианты

– Прочитать тело функции:

    + понятно как всё устроено

# Возможные варианты

– Прочитать тело функции:

  + понятно как всё устроено

  - отнимает время

# Возможные варианты

– Прочитать тело функции:

  + понятно как всё устроено

  - отнимает время

  - можно случайно рекурсивно прочесть весь интернет

# Возможные варианты

– Прочитать тело функции:

  + понятно как всё устроено

  - отнимает время

  - можно случайно рекурсивно прочесть весь интернет

– Переименовать в функцию вида *getValidationErrors*

# Возможные варианты

– Прочитать тело функции:

  + понятно как всё устроено

  - отнимает время

  - можно случайно рекурсивно прочесть весь интернет

– Переименовать в функцию вида *getValidationErrors*

  + понятно, что что-то возвращается

# Возможные варианты

– Прочитать тело функции:

  + понятно как всё устроено

  - отнимает время

  - можно случайно рекурсивно прочесть весь интернет

– Переименовать в функцию вида *getValidationErrors*

  + понятно, что что-то возвращается

  - непонятно, что имеется ввиду — кто такие *errors* и что с ними делать

# Возможные варианты

– Прочитать тело функции:

  + понятно как всё устроено

  - отнимает время

  - можно случайно рекурсивно прочесть весь интернет

– Переименовать в функцию вида *getValidationErrors*

  + понятно, что что-то возвращается

  - непонятно, что имеется ввиду — кто такие *errors* и что с ними делать

– Добавить документацию в которой расписать что и как тут работает

# Возможные варианты

– Прочитать тело функции:

  + понятно как всё устроено

  - отнимает время

  - можно случайно рекурсивно прочесть весь интернет

– Переименовать в функцию вида *getValidationErrors*

  + понятно, что что-то возвращается

  - непонятно, что имеется ввиду — кто такие *errors* и что с ними делать

– Добавить документацию в которой расписать что и как тут работает

  + всё понятно и не нужно читать код

# Возможные варианты

– Прочитать тело функции:

  + понятно как всё устроено

  - отнимает время

  - можно случайно рекурсивно прочесть весь интернет

– Переименовать в функцию вида *getValidationErrors*

  + понятно, что что-то возвращается

  - непонятно, что имеется ввиду — кто такие *errors* и что с ними делать

– Добавить документацию в которой расписать что и как тут работает

  + всё понятно и не нужно читать код

  - отнимает время на написание

# Возможные варианты

– Прочитать тело функции:

  + понятно как всё устроено

  - отнимает время

  - можно случайно рекурсивно прочесть весь интернет

– Переименовать в функцию вида *getValidationErrors*

  + понятно, что что-то возвращается

  - непонятно, что имеется ввиду — кто такие *errors* и что с ними делать

– Добавить документацию в которой расписать что и как тут работает

  + всё понятно и не нужно читать код

  - отнимает время на написание

  - отнимает время на поддержание актуальности

# Альтернативное решение

Хранить информацию о том, что принимает функция на вход и что возвращает прямо вместе с кодом


**"Now! *That* should clear up a few things around here!"**

# Статическая типизация

```
const validate = (data: object, schema):Array<ValidationError> => {
  const errors:Array<ValidationError> = [];
  for (const key of Object.keys(schema)) {
    for (const error of validateField(data, key, schema[key])) {
      errors.push(error);
    }
  }
  return errors;
};
```

# Когда стоит использовать статическую типизацию

# Когда стоит использовать статическую типизацию

– Программа очень важна для бизнеса

# Когда стоит использовать статическую типизацию

– Программа очень важна для бизнеса

– Код скорее всего придётся многократно рефакторить

# Когда стоит использовать статическую типизацию

– Программа очень важна для бизнеса

– Код скорее всего придётся многократно рефакторить

– Программу будет поддерживать большая команда или её нужно передавать заказчику

# Когда не стоит использовать статическую типизацию

# Когда не стоит использовать статическую типизацию

– Некритичная подсистема

# Когда не стоит использовать статическую типизацию

– Некритичная подсистема

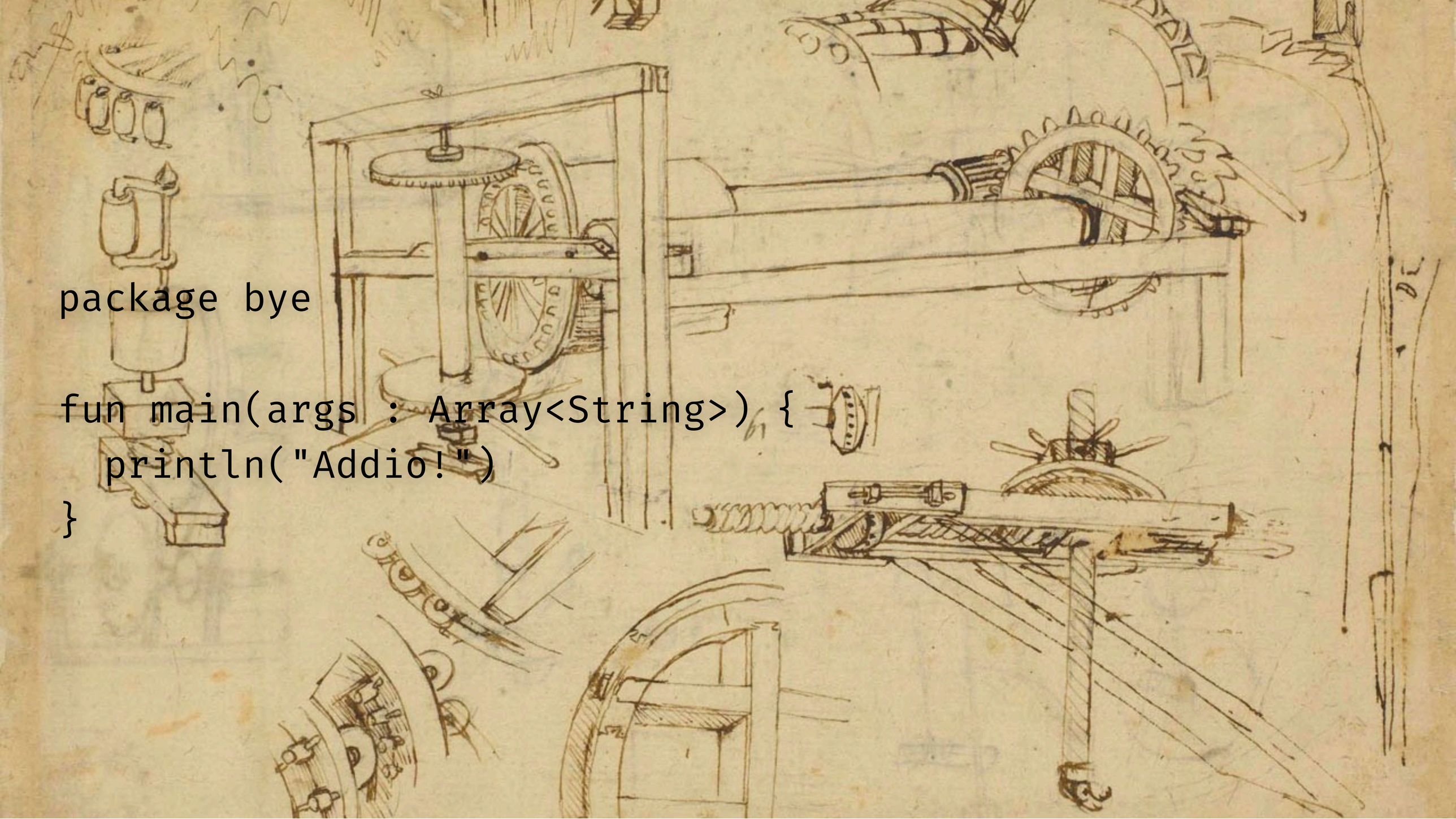
– Прототип, который скорее всего выкинут

# Когда не стоит использовать статическую типизацию

– Некритичная подсистема

– Прототип, который скорее всего выкинут

– Это ваш домашний проект



```
package bye

fun main(args : Array<String>) {
  println("Addio!")
}
```