Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

Programovací jazyk Dart

Bakalářská práce

Autor: Jaroslava Otmanová
Studijní obor: ai3-k

Vedoucí práce: Ing. Jiří Štěpánek

Hradec Králové Červen 2015

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracovala samostatně a s použitím uvedené literatury.

V Lounech dne 17.8.2015 Jaroslava Otmanová

Poděkování:

Děkuji vedoucímu bakalářské práce Ing. Jiří Štěpánkovi za metodické vedení práce a podnětné rady a informace, které mi pomohly.

Anotace

Dnes se používá několik jazyků v oblasti webové tvorby, mezi které patří i JavaScript. Vzhledem k nedostatkům tohoto jazyka se společnost Google rozhodla vytvořit zcela nový jazyk, který by se zjištěným mezerám v JavaScriptu měl vyvarovat a podat tak vývojářům nový prostředek. Programovací jazyk Dart má dnešnímu IT světu co nabídnout. Ukazuje nové myšlenky a cesty k dosažení cíle a zároveň vychází vstříc zaběhnutým konvencím v oblasti syntaxe. Dart nabízí možnost konverze kódu do JavaScriptu, tudíž napsaný kód bude možné provést i v dalších prohlížečích. Bylo provedeno testování kódu Dartu versus kódu JavaScriptu v období samotného jazyka i Dart nástroje dart2js pro kompilaci. Testování proběhlo na výpočtu Fibonacciho posloupnosti a Eratosthenově sítu. Ve všech případech bylo zpracování kódu relativně obdobné, bez větších výkyvů. Verze jazyka se v průběhu psaní práce několikrát změnila, ale velké změny nebyly zaznamenány. Vývojová komunita ale ohlásila, že se s vývojem přesune primárně k nástroji dart2js a Dart virtual machine se do prohlížeče integrovat nebude.

Annotation

Title: Dart, the programming language

Today there are used variety of programming languages for web development, which include also JavaScript. Considering the shortage of this programming language, Google company has decided to create an entirely new language that would have identified gaps in JavaScript to avoid them and offer a new tool for developers. Dart, programming language, has a lot to offer to today's world. It shows a new ideas and ways to achieve a goal and it is also meeting established conventions in syntax. Dart offers the possibility of converting code into JavaScript, thus written code can be provided in other browsers. Test were conducted on Dart vs JavaScript code and optimized code through dart2js tool. Testing was provided by computations of Fibonacci sequence and Eratosthen sieve. In all cases, there were similarities between the computing times without bigger deflection. The version of

the Dart language was changed many times during the writing, but there was not a lot of major changes. The development community release a message that the main priority of development in a future will be focus on the dart2js tool and Dart virtual machine will not be implemented to the Chrome browser.

Obsah

Seznam obrázků

Seznam tabulek

Seznam kódů

Seznam grafů

1 Úvod

S nárůstem používání a využívání jazyka JavaScript se objevuje stále více jeho nedostatků. Tyto nedostatky se projevily hlavně díky vlivu používání jazyka JavaScript v rozměrech, na které nebyl postaven. Tato situace se řeší vývojem knihoven, které mají zalátat vzniklé mezery. To postupem času znepřehlednilo samotný jazyk jako celek. Vývojáři ve společnosti Google se rozhodli pro jiné, nové řešení. Místo přidáváním knihoven vytvořit jazyk nový. Vyvinuly tedy Dart, který se hned v samotném návrhu pokusí vychytat co nejvíce chyb a bezpečnostních rizik. Jazyk používá zaběhnuté konvence psaní syntaxe, ale vkládá nové myšlenky do skladby jazyka jako takového. Snaží se vychytat dosud nalezené nedostatky ostatních jazyků a usnadnit tak vývojáři práci. Dart je stále ve vývoji, ale již má svou stabilní verzi. Pro provoz v prohlížečích se používá konverze do jazyka JavaScript skrze nástroj dart2js. V nedávné době bylo vývojáři Dartu oznámeno, že původně zamýšlené zabudování Dart virtual machine do prohlížečů pro ostrý provoz na webu se zavrhuje a vývoj se tak bude soustředit na vylepšování obsahu jazyka Dart a také na vylepšování nástroje dart2js pro kompilaci kódu.

2 Cíl práce

Cílem této práce je obeznámení s programovacím jazykem Dart. Bude stručně vysvětlen princip, do jaké kategorie programovacích jazyků se Dart řadí i kdo za Dartem v pozadí stojí. Budou zde představeny základní myšlenky i aspekty jazyka a nastíní se jeho využití. Výzkumnou otázkou práce bude porovnání provozu JavaScriptu skrze samotný kód jazyka a nástroje dart2js a samotného Dartu na praktické ukázce demonstrující dvě matematické úlohy a to Fibonacciho posloupnost a hledání prvočísel dle Eratosthenova síta. Dále je v praktické části popis konstrukcí jazyka na jednotlivých ukázkách z jedné aplikace z pohledu použitelnosti, syntaxe, konstrukcí a přívětivosti pro vývojáře.

3 Metodika zpracování

Hlavní myšlenkou této práce je seznámení se s jazykem Dart. V praktické části pak porovnání dvou jazyků a to Dart a JavaScript s pomocí nástroje dart2js a také praktická ukázka nových konstrukcí, které Dart nabízí. Výstupem by mělo být srovnání výkonnosti, použitelnosti a přehlednosti obou jazyků na zvolených úlohách.

Předpokladem pro testování kódů je myšlenka, že jazyk Dart bude vhodnější na použití a rychlejší než JavaScript. Ukazatelem vhodnějšího použití je jasná orientace v kódu, jeho čitelnost a náročnost. Ukazatelem rychlosti je rychlost zpracování kódu a zobrazení požadovaného výsledku skrze webové rozhraní.

Jazyk JavaScript bude interpretován jak samotným JS kódem, tak skrze Dart nástroj dart2js pro kompilaci kódu Dart do JS.

Testování je zvoleno na dvou algoritmech a to počítání Fibonacciho posloupnosti a hledání prvočísel skrze Eratosthenovo síto. Rozhraní pro manipulaci bude skrze webový formulář.

Z důvodu nepodporování Dartu v prohlížečích, bude testování probíhat v prostředí Dartium, které obsahuje samotné DartVM.

Posledním příkladem jsou úryvky kódu z aplikace pro zaznamenávání úkolů. Úryvky byly do práce vybrány hlavně z hlediska zápisu, konstrukcí, metod a použitých tříd. Zbytek kódu, který zde není uveden, by se nijak zvlášť nelišil od běžných programátorských praktik.

4 JavaScript

4.1 Základní informace

JavaScript je skriptovací jazyk s náznakem objektově orientovaného programování z roku 1997, kdy byl standardizován organizací ECMA. Jeho autorem je Brendan Eich ze společnosti Netscape [1]. JavaScript má podobnou syntaxi jako jazyky C a Java. Je to interpretovaný jazyk, nemusí se tedy kompilovat, což má jistou výhodu v přenositelnosti. Jazyk slouží hlavně pro úpravu webových stránek a dodání dynamičnosti webu, pro což byl stvořený. Dnešní aplikace ale jsou mnohem náročnější a tak se naráží na potíže s možnostmi tohoto jazyka.

4.2 JavaScript a web

JavaScript je úzce spjat s webovou tvorbou. Kód jazyka je umístěn buď přímo v HTML kódu, nebo v samostatném souboru, na který vede odkaz. Spouštění skriptu probíhá na straně klienta, tedy až po načtení souborů. Pro jazyk tím plynou některé nevýhody. Nemá přístup k uživatelským datům, to kvůli bezpečnosti, protože pracuje pouze v prohlížeči. Nevýhodou jazyka je také možnost, že v době načítání stránky bude u uživatele JavaScript vypnutý, tudíž se skript s kódem neprovedou [2]. Pokud na skriptu silně závisí funkčnost stránek, nastává problém.

Dart běží jak na klientově tak serverové straně, nicméně JavaScript potřebuje pro běh na serveru knihovnu Node.js [3]. Jako i na další pokročilejší funkce má JavaScript další knihovny, Dart má tyto věci již vestavěné v sobě samém naimplementované již od začátku a nepotřebuje hledat další komponenty.

5 Dart

5.1 Historie

V roce 2011 uvedla společnost Google na konferenci GOTO [4] nový jazyk Dart s verzí 1.0. Projekt jako takový byl započat vývojáři Lars Bak a Kasperk Lund [5]. Jazyků, které ovlivnily tento projekt je více – v první řadě určitě JavaScript, dále Erlang, Stongtalk a Smalltalk. Velké a složité aplikace je těžké psát v dostupných jazycích pro web, proto bylo potřeba přijít s něčím novým, inovativním. Je potřeba zrychlit běh aplikací, zpřehlednit kódování - hlavně strukturu kódu samotného - a také sjednotit roztříštěný „trh“ verzí jazyka JavaScript a jeho podpory v moderních prohlížečích. Dalším požadavkem je určitě i možnost sjednotit všechny zařízení, ve kterých jsou aplikace dostupné na jednotnou podporu technologií. Dart se stále vyvíjí a během psaní této práce se verze několikrát změnila. Aktuální verze jazyka je 1.11.1 (červenec 2015).

5.2 Základní informace

Dart je objektově orientovaný jazyk vyvíjený jako open-source. Podobný je moderním programovacím jazykům typu Java, takže přechod z jiných programovacích jazyků nedělá problémy s porozuměním dané struktury.
Hlavní část programu je metoda **void main()**, která se provádí jako první.
V Dartu je volitelné typování, to zajišťuje určitou volnost při psaní kódu. Na volbu typování má určitě návaznost přítomnost dvou módů běhu Dart aplikace a to produkční a vývojový mód. Vývojový mód je přívětivý pro vývojáře možností kontroly konzistence typování proměnných a poskytnutí zpětné vazby. Výchozí produkční mód všechny určené typy ignoruje, nastaví dynamické typování **var** a běží s tímto nastavením dál [17].
Jak může být z následujícího kódu patrné, dvě proměnné *cislo* a *cislo2* obsahují obě numerickou hodnotu, ale nemají stejný typ. Proměnná *cislo* má svou výhodu v dynamičnosti. Proměnná *cislo2* má zase svou výhodu v čitelnosti a odlaďování. Ve vývojovém módu má nesporné výhody dát předem vědět, co bude daná proměnná obsahovat a to zejména pro vývojáře jako prostředek pro daleko lepší čitelnost a orientaci a dále následné ladění kódu. Pro vývojový a produkční mód mají řádky

zapsané v daném kódu stejnou výchozí sémantiku [17]. Toto dynamické typování v produkčním módu je dané tím, že Dart je dynamický jazyka a tudíž musí v tomto módu ignorovat statické typování. Díky přeskočení typových kontrol se zrychlí i běh celého kódu.

```
void main() {
  // proměnná obsahující číslo, ale typu var
  var cislo = 11;
  //proměnná typu integer
  int cislo2 = 17;

}
```

Kód 1: Prezentace typování a viditelnosti
Zdroj: vlastní zpracování

Viditelnost proměnných je v Dartu řešena buď **public** a nebo **private** označením. Privátní proměnné se na začátku označí podtržítkem. Dále se používají jmenné konstruktory, zřetězení operací na objektu, generické třídy, zkrácený zápis funkce a další.

5.3 Instalace Dartu

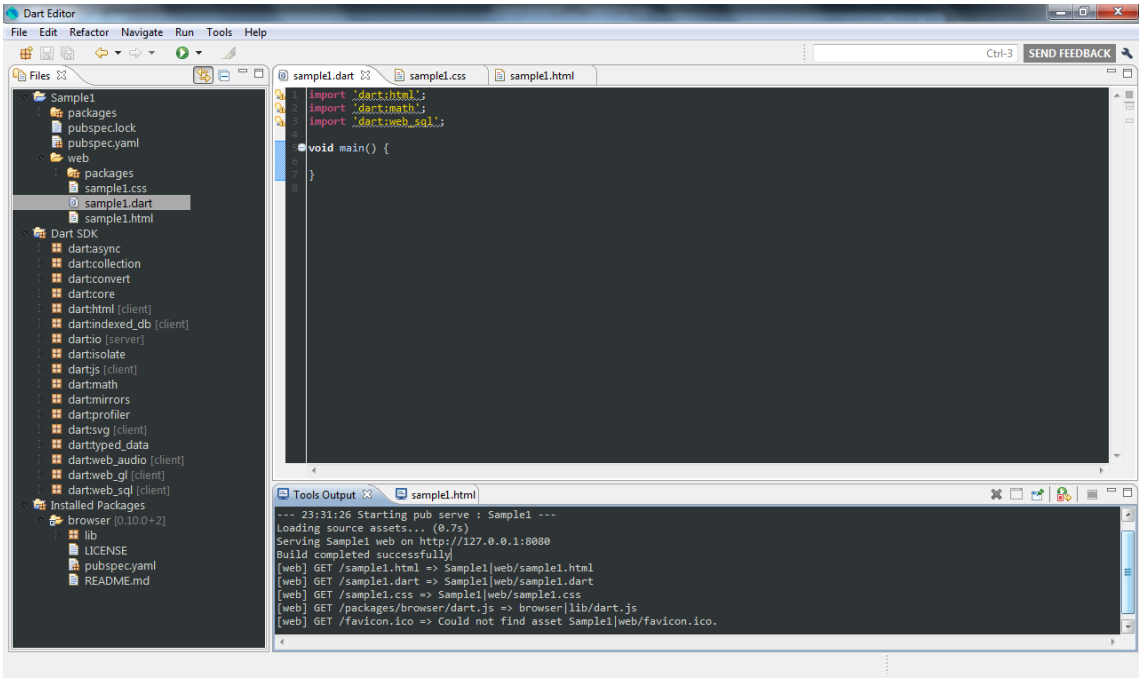
Instalace prostředí pro Dart je v zásadě velmi snadná. Z domovské adresy Dartu (zdroj) stačí stáhnout zip soubor s Dart SDK. Možností je, samozřejmě více. Je možné stáhnout i soubor s Dart editorem a Dartiem, co by webovým prohlížečem pro bezproblémový běh Dart aplikací.

5.4 Podpora OS

Dart je podporován na všech předních operačních systémem, jmenovitě Linux, Windows (pro verze Vista, 7 a 8) a Mac OSX.

5.5 Dart Editor

Editor pro Dart je založen na editoru Eclipse. Vyžaduje Java runtime 6.0 a vyšší. Je nápomocný při psaní kódu i používání knihoven a to prostřednictvím našeptávače. Obsahuje další základní funkce jako refaktoring, stromovou strukturu tříd a metod, debugger, průběžné varování na chyby v kódu a další.



Obr. 1: Dart Editor
Zdroj: vlastní zpracování

V editoru je integrován překladač z jazyka Dart do jazyka JavaScriptu s názvem dart2js [6], který umožňuje bezproblémové spuštění aplikací mimo podporu jazyka Dart. Tento compiler přeloží kód do jazyka JavaScript v co nejoptimalizovanější formě. Většinou je takto strojově přeložený kód lépe optimalizován než s programátorovým kódováním.

Praktickou volbou při novém projektu je zvolit krok „Vytvořit nový projekt“. Zde se volí, jakého typu projekt bude. Při programování prázdné webové aplikace vznikne základní struktura pro web – soubor pro css styly, dart soubor a html soubor; tyto soubory mají přednastavenou základní ukázkovou kostru, se kterou se dále pracuje. K dispozici je mnoho možností pro práci se šablonami, například aplikace pro

prohlížeč Chrome, webová aplikace s responzivním designem nebo aplikace využívající polymer.dart.

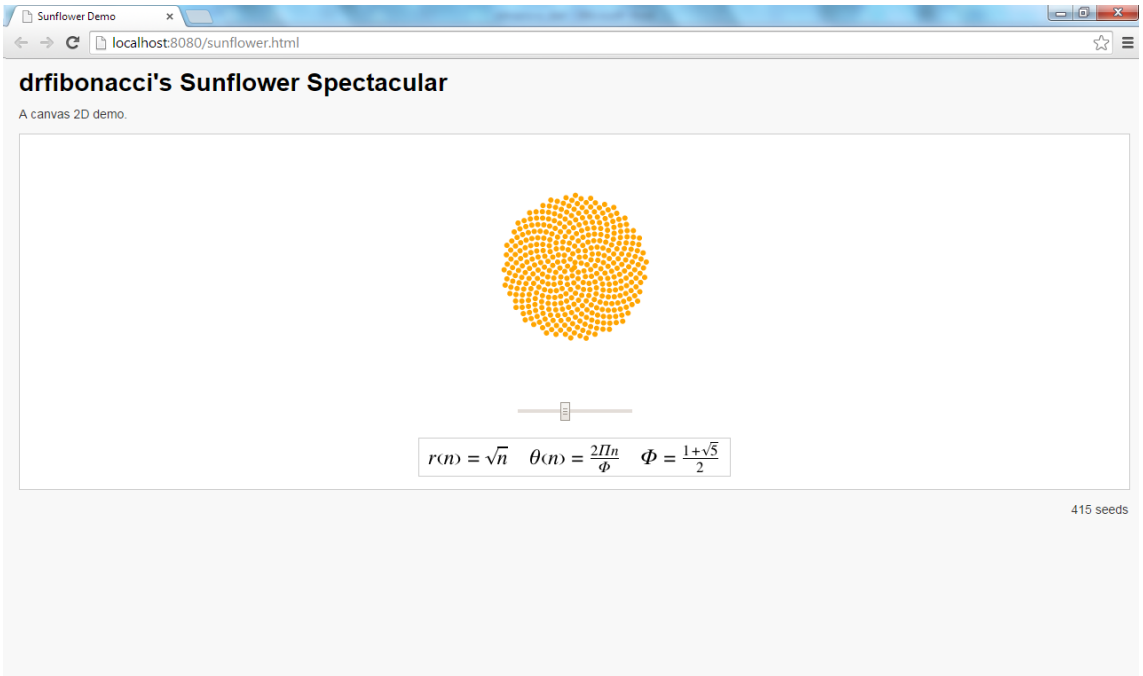
Ovládání editoru je intuitivní a jednoduché. Vše je vidět na Obr. 1: Dart Editor.

Existují i alternativy pro psaní kódu optimalizované na programovací jazyk Dart v podobě pluginů pro IDE jako například pro Eclipse nebo pro textové editory jako je Sublime Text nebo Emacs. Poslední novinkou je signifikantní vylepšení pro WebStorm 10 stojící jako plnohodnotný editor pro Dart vývoj [9].

5.6 Dartium

Jelikož je Dart celkem nový mezi technologiemi, není implementován do prohlížečů. Aby se měl kód Dartu kde spustit, byl pro něj vyvinut vlastní prohlížeč s názvem Dartium. Je to jádro prohlížeče Chromium s přidaným Dart Virtual Machine.

Prohlížeč Dartium není doporučován pro běžné využití namísto klasického prohlížeče a to z bezpečnostních důvodů. Dartium byl vyvinut jako technologická ukázka pro Dart a mohou vzniknout bezpečnostní chyby v nestabilních podmínkách[8].



Obr. 2: Dartium
Zdroj: Ukázkové kódy vývojového prostředí

Implementace DartVM do běžně využívaných prohlížečů zatím není v dohledné době v realizaci. Vývojáři jazyka vydali prohlášení, ve kterém odstupují od

původního plánu implementovat DartVM do prohlížeče Chrome a následné rozšíření mezi ostatní prohlížeče. Více o tomto tématu viz [8].

5.7 Nástroj dart2js

Nástroj dart2js je navržen pro kompilaci Dart kódu do JavaScript kódu [11]. Tento nástroj je vestavěn do Dart Editoru pro snadný přístup k této funkci. Spuštění tohoto nástroje je možné pomocí funkce **Pub Build** nebo přímo přes příkaz se stejným názvem.

5.8 Dart virtual machine

Kód jazyka Dart může být spuštěn skrze příkazový řádek nebo skrze Dart Editor. Avšak pokud se jedná o webovou aplikaci, nelze použít *dart* příkaz pro její spuštění (zdroj). Webové aplikace se spouštějí přes webové prostředí.

Dart skripty mohou běžet ve dvou módech a to produkční a kontrolovaný. Produkční mód je nastaven jako výchozí mód. Kontrolovaný mód je doporučován jako umožnění pro vývoj a testování skriptů a aplikací [7].

V kontrolovaném módu je umožněné dynamické kontrolování procesů, kde některé porušení typování může vést ke zvýšení výskytů výjimek. Naproti tomu v produkčním módu nemá statické typování vliv na chod aplikace (zdroj).

DartVM je defaultně integrován do prohlížeče Dartium, tím umožňuje přímé spuštění Dart kódu bez nutnosti kompilace do kódu v JavaScriptu. Integrací DartVM do ostatních prohlížečů by bylo umožněno zrušení nutnosti kompilaci do JavaScriptu pro chod Dart aplikací.

5.9 Pub

Nástroj, který je programátorovi k dispozici již s Dart Editorem nebo primárně s Dart SDK. Tento nástroj je využíván ke správě Dart packages.

5.10 Podpora

Byť je Dart novou technologií, díky internetovým komunitám na několika sociálních sítích je podpora Dartu hojná. Tvůrci jazyka samotné komunity založili, ale vývojáři a programátorští nadšenci tyto komunity udržují a denně přibývá několik

příspěvků, článků, návodů, atd. V provozu je také blog o vývoji jazyka a probíhá také mnoho přednášek (i v České republice). Seznam oficiálních komunit pro podporu programování v jazyce Dart je dostupný na oficiálních stránkách jazyka [10].

6 Dart jako programovací jazyk

6.1 Knihovny

Knihovny tvoří nástavbu na základ jazyka. Pro přidání knihovny se využívá **import**. Pro vestavěné knihovny stačí za import příkazem zadat **dart** a název knihovny, například následující část kódu naimportuje knihovny pro využití **html**, **math**, **collection** a další.

```
import 'dart:html';
import 'dart:math';
import 'dart:collection';
```

Kód 2: Import knihoven
Zdroj: vlastní zpracování

Základní knihovnou je **dart:core** (zdroj), která je automaticky naimportována do každého Dart projektu a obsahuje základní funkce, jako jsou základní operace s čísly, textové řetězce, seznamy, a další (zdroj).
Knihovna **dart:html** slouží pro práci s HTML prvky a DOM. Pomocí **querySelector(String s)** jsou vybírány jednotlivé elementy. Třída **Element** obsahuje konstruktory pro jednotlivé elementy html [13].

```
var outputList = querySelector('#ouputList');
var newLi = new LIElement();
newLi.text = 'Text';
```

**Kód 3: ukázka querySelector a elementu **
Zdroj: vlastní zpracování

Ukázkový kód obsahuje *querySelector* pro vyhledání html tagu pomocí *id* a také vytvoření nového elementu pro html tag <li>. Na třetím řádku kódu se k položce seznamu přiřadí text.
Dále je možné s novými DOM elementy pracovat pomocí třídy Element a html tagu daného elementu, například **Element.a()** [13] pro vytvoření nového elementu odkazu.

Pro matematické funkce, které nejsou přístupny pod základní knihovnou je využívána knihovna **dart:math**. Ta pokrývá například goniometrické funkce, práci s logaritmy a exponenty, minimum či maximum ze dvou čísel nebo třídu pro práci s náhodnými hodnotami [13].

Pokročilejší a doplňující operae s kolekcemi je umožněna knihovnou **dart:collection**.

Dart umožňuje naimportovat knihovny i dle adresářové cesty.

6.2 Datové typy

Datové typy, ve srovnání s ostatními jazyky, jsou standartní. Liší se pouze v malých detailech.

Inicializační hodnota všech typů proměnných je **null**, protože všechny typy jsou v Dartu objekty [5].

Typ **var** využijeme ve všech případech, kdy nechceme zadávat typ. Představuje univerzální a jednoduché použití, pokud jsme nejdříve nerozhodli, jaký datový typ daný objekt bude.

Numerické hodnoty mohou nabývat pouze typů **double** a **int**. Supertype těchto dvou typů je **num. Num** definuje základní operace s čísly. Pro rozšířené operace nabízí Dart knihovnu **math** s množstvím funkcí.

V jazyce Dart nefiguruje typ **array**, jak je to ve většině jazyků. Jsou zde k použití čtyři typy **Collection**s: **List**, **Set**, **Queue** jak znázorňuje Obr. 3: Rozdělení kolekcí níže. Jejich použití se odvíjí od toho, jaké vlastnosti dané struktury jsou potřeba. Supertype všech kolekcí je **Iterable**. **Map** nerozšiřuje **Collection**s, ale je to užitečný typ na uložení dat [12]. Skrz tyto třídy se generují zajímavé a užitečné metody. Některé z nich jsou použity v kódu, který je k nalezení dále v práci.

6.2.1 Kolekce

Kolekce jsou zajímavé, z hlediska jejich použití i rozdělení, proto pár slov k nim.

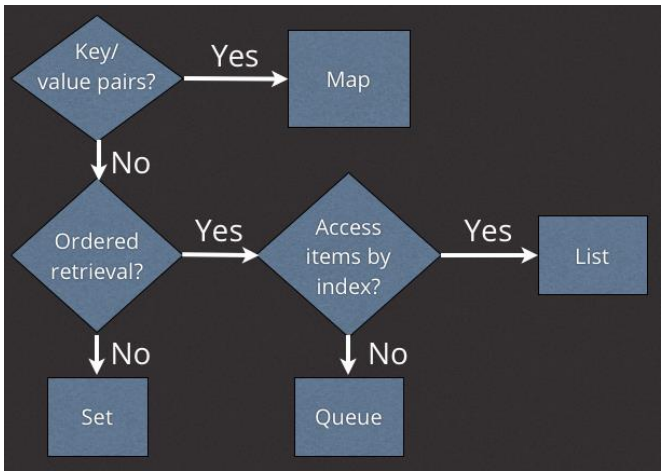
List je využíván jako klasický **array**. Využívá se pro uspořádaný seznam položek, zero-indexed, Pokud je tedy potřeba pracovat s indexy daných položek, je list to pravé.

Set je neuspořádaná kolekce, která kontroluje jedinečnost každé položky. Užitečnou metodou pro **set** je **intersection**. Ta vrací nový **set** obsahující společné prvky dvou seznamů [12].

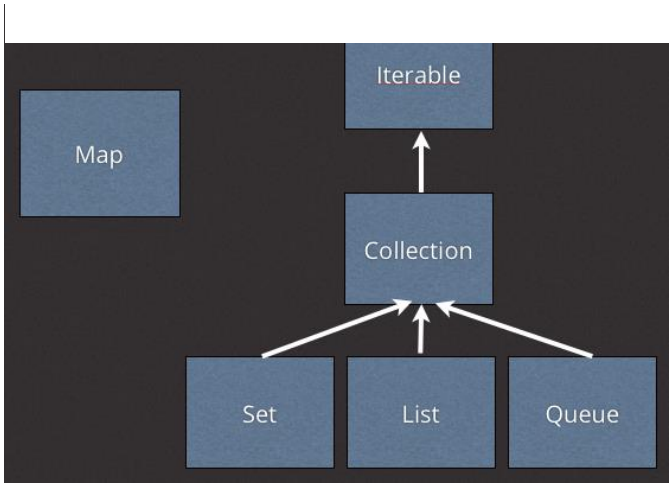
Queue slouží k uložení prvků za podmínky, že nepotřebujete prvky indexovat, ale stále je udržet uspořádané.

„Queue v Dartu může být považováno za first-in-first-out nebo last-in-first-out datovou strukturu“ (převzato z [12]).

Pokud je potřeba využití slovníků, je zde struktura **Map**. Slouží standardně pro přidělení klíče k prvku. Klíč musí být unikátní a nesmí být **null** [12].



Obr. 4: Rozhodnutí, jakou strukturu kolekcí zvolit
Zdroj: [12]



Obr. 3: Rozdělení kolekcí
Zdroj: [12]

6.3 Třídy

Vytváření tříd má obecně být intuitivní, jasné a na první pohled rozpoznatelné. JavaScript pro třídy používá strukturu funkce, což je na první pohled trochu matoucí, obzvláště v porovnání s jazyky, které využívají *class* výrazy, ke kterým se Dart řadí.

```
var Person = function () {};
...
var person1 = new Person();
```

Kód 4: Vytvoření třídy a její instance v JavaScriptu
Zdroj: [28]

Třída v Dartu se tedy sestrojí klasickým výrazem *class*, názvem třídy a jejím obsahem [26].Konstruktor může mít více podob, záleží na stylu psaní a potřebách třídy. První možnost je vypsání všech potřebných hodnot jako argumenty konstruktoru a dále pomocí *this* přiřadit nebo využít méně psaní a napsat vše rovnou na místo argumentů.

```
class Person {

  String jmeno, prijmeni;
  num vek;

  ...

}
```

Kód 6: Vytvoření třídy v Dartu
Zdroj: vlastní zpracování

```
//plny konstruktor
Person(String jmeno, String prijmeni, num vek) {
    this.jmeno = jmeno;
    this.prijmeni = prijmeni;
    this.vek = vek;
  }
//zkracene zapsany konstruktor
Person(this.jmeno, this.prijmeni, this.vek);
```

Kód 5: Konstruktory v Dartu
Zdroj: vlastní zpracování

Pokud se konstruktor nenadefinuje, automaticky se použije výchozí konstruktor, který nemá žádné argumenty [26] ani tělo.

Dart samozřejmě disponuje i dědičností za účelem redukce kódu a zlepšení jeho čitelnosti. Jedna třída může rozšířit druhou skrze výraz *extends*. Zdědí se tak vlastnosti a může se odkazovat na metody a konstruktory pomocí výrazu *super* rodičovské třídy [26]. Odvozená třída si může přepsat metody z rodičovské třídy [26]. Rodičovská třída ale může být pouze jedna přímá.

Následující příklad předkládá ukázku použití výrazu *extends* použitou na třídě *FluffyBunny*, která rozšiřuje třídu *Bunny*. Na druhém řádku je také k vidění použití výrazu *super* pro odkazování se na metodu rodičovské třídy. Tento příklad je převzat z ukázkového kódu oficiálního Dart seriálu. Na jednoduchém příkladu jasně poukazuje na využití rozšíření tříd.

```
class FluffyBunny extends Bunny {
  num fluffiness;

  FluffyBunny(String name, this.fluffiness) : super(name);
}
```

Kód 7: Ukázka použití rozšíření tříd
Zdroj: [8]

Jelikož Dart není staticky typovaný programovací jazyk, není zde k dispozici přetěžování metod ani konstruktorů. Jména metod musí být jedinečná. Pokud je ale potřeba druhý konstruktor třídy, je možno použít jmenný konstruktor (name constructor), který je definovaný formou *ClassName.constructorName* [26].

Následující příklad, který rovněž pochází z oficiálního Dart seriálu, jednoznačně poukazuje na použití jmenného konstruktoru, zejména potřeby dvou odlišných potřebách zápisu. Pro proměnné *xml* a *json* je potřeba jejich rozdílného obsahu, proto i dvou odlišných konstruktorů.

```
class FluffyBunny {
  FluffyBunny.fromXml(String xml) {
    // …
  }

  FluffyBunny.fromJson(String json) {
    // ...
  }
}

var xml = "<bunny><name>floppy</name></bunny>";
var json = '{"bunny":{"name":"peter"}}';

var floppy = new FluffyBunny.fromXml(xml);
var peter = new FluffyBunny.fromJson(json);
```

Kód 8: Použití jmenného konstruktoru
Zdroj: [8]

6.4 Funkce

Dart nabízí plno interakcí s funkcemi jako, jsou volitelné parametry, vnořené funkce i možnost mít funkci jako argument jiné funkce (zdroj). Dále je také k dispozici zkrácený zápis funkce. Když například funkce jen vrací nějaký objekt, není nutno používat složené závorky pro tělo funkce, ale celá funkce se dá zapsat na jeden řádek. Jako v následujícím příkladu, který je převzat z praktické části, kde se objevuje i přidání podmínky a to vše na jednom řádku. První část těla funkce říká, že pro číslo menší či rovno číslu 2 vrací funkce hodnotu 1, pro ostatní případy pak proběhne daný výpočet.Rozhraní (Interface)

```
int fibon(int f) => f <= 2 ? 1 : fibon(f - 2) + fibon(f - 1);
```

Kód 9: Zkrácený zápis funkce
Zdroj: vlastní zpracování

Rozhraní jsou v Dartu často používaná, mnohdy více než rozšíření [26]. Každá třída si implicitně definuje rozhraní jako kolekci charakteristik a chování [26][27] se stejným názvem jako má daná třída [27]. Rozhraními jsou i abstraktní třídy. *Interface* v Dartu není předdefinovaný výraz, pro využití takové implementace se využívá výraz *implements*.

Následující kód ukazuje implementování rozhraní na třídách *FluffyBunny* a *Bunny*. V kódu by následovalo implementování typických věcí pro danou třídu *FluffyBunny* s ohledem na výchozí nastavení *Bunny*.

Nyní také může třída *FluffyBunny* mít nového rodiče, pokud je to potřeba.

```
class FluffyBunny implements Bunny {
  num fluffiness;

  FluffyBunny(String name, this.fluffiness);

  ...

}
```

Kód 10: Ukázka výrazu implements
Zdroj: [8]

6.5 Asynchronní programování s Dartem

Při synchronním volání hlavní vlákno aplikace po volání nějaké akce čeká na své dokončení. Aplikace tak nemůže pokračovat ve svém cyklu, vykonávat další operace a uživatel může mít pocit, že aplikace se zasekla. Pokud je potřebné, aby aplikace dále pokračovala ve své práci a přijímala požadavky na další operace, použije se asynchronní volání. Tím pádem se akce, která je volána, spustí mimo hlavní vlákno a vytvoří si vlákno samostatné. Nezdržuje se tak chod hlavního vlákna samotné aplikace a zaručí se hladký průběh interakce s uživatelem. Asynchronní volání je využíváno hlavně u webových serverů a aplikací, u kterých se očekává dlouhotrvající operace. Totiž v případě, že by server volalo více klientů, každý by musel čekat, než se klientům před ním odešle ze serveru odpověď, což je nežádoucí. Je prioritou, aby se klienti obsluhovali najednou.

Dart aplikuje asynchronní operace pomocí **Future**. Future reprezentují prostředky, které podávají hodnoty, které se vyskytnout v dalším chodu programu, ale v nynější době nejsou k dispozici. Když přijde daná funkce Future na řadu, v prvé řadě vrátí nedokončený objekt, který v budoucnu převezme dané hodnoty a dokončí se. Samozřejmě může nastat chyba, když se objektu nedostane hodnota, která je požadována. Tato chyba může být v rámci nedokončené input operace, chybou vstupu od uživatele, přerušení spojení, atd.

Pro získání dat a hodnot, které Future reprezentuje, používá se buď *async* a *await (převzato z [18]).* Tyto komponenty pomáhají programátorovi využívat výhody asynchronního programování tak, aby vypadalo jako synchronní a zároveň, aby se nemuselo používat Future API (převzato z [18]). *Async* a *await* byly přidány do Dart jazyka ve verzi 1.9, do té doby bylo potřeba využívat Future API.

Přístup ke komponentám asynchronního kódu jsou dostupné pomocí knihovny **dart:async** [13].

V Dartu hodně metod vrací Future, jednou z nich je metoda **bind()**, která v následujícím příkladě váže *HttpServer* s hostitelem a portem. Metoda **then()** registruje zpětné volání pro úspěšně dokončené Future. V tomto případě pak metoda **bind()** vrátí objekt *HttpServer [29].*

```
HttpServer.bind('127.0.0.1', 4444)
      .then((server) => print('${server.isBroadcast}'))
      .catchError(print);
```

Kód 11: Future, bind(), then()
Zdroj: [29]

6.6 Polymer.dart

Knihovna Polymer je moderní nástroj na zjednodušení práce s vývojem webových aplikací, kde se aplikují metody znovu-použitelnosti, rychlého a usnadněného vývoje [19]. Klasické elementy HTML disponují svou strukturou, atributy a metodami, nicméně mnohdy je potřeba vytvořit svou vlastní strukturu personalizovanou na daný projekt. Polymer knihovna dává k dispozici syntaxi pro jednodušší definování vlastních elementů [19]. K dispozici jsou také šablony a vestavěné elementy, které jsou předem připraveny pro použité na webu. Hlavní výhodou této knihovny je možnost přizpůsobení danému projektu, znovu-použitelnost, méně kódu a také přehlednější kód.

Dart knihovna Polymer.dart je implementací knihovny Polymer, která dává všechny její výhody k dispozici pro vývoj Dart aplikací.

```
import 'package:polymer/polymer.dart';
```

Kód 12: import knihovny Polymer.dart
Zdroj: vlastní zpracování

Polymer aplikace v jazyku Dart většinou neobsahují hlavní metodu main() [22], jak tomu bývá při běžných aplikacích.
Celá struktura elementů knihovny Polymer.dart se definuje pomocí specifického tagu uvedého v následujícím kódu, které obsahuje atribut *name* s vlastním názvem.

```
<polymer-element name="dart-show">
  <template>
    <style>
    </style>
  </template>

  <script type="application/dart" src="dart-show.dart"></script>
</polymer-element>
```

Kód 13: Definování polymer elementu
Zdroj: vlastní zpracování

Uvnitř elementu může být přítomna šablona a chování pro element. Je možné využít odkazování jen na jednu záležitost, pak element obsahuje čistě jen funkčnost nebo vzhled [22]. Tato struktura se definuje v html dokumentu pro definici elementu. V samotném html dokumentu aplikace se importuje šablona pro element a daný element se již používá pro své účely.

```
<!-- example of your own custom element -->
<link rel="import" href="packages/polymerTemplate/dart-
show.html">
...
<dart-show></dart-show>
```

Kód 14: Použití polymer elementu
Zdroj: vlastní zpracování

Použití oproti standardní knihovně Polymer je obdobné až stejné. Snaha o její integraci do prostředí jazyka Dart je pochopitelná, nicméně má problém

s kompatibilitou verzí. Současná Dart knihovna Polymer.dart je kompatibilní s knihovnou Polymer verze 0.16. Na kompatibilitě s novou verzí Polymer 1.0 se pracuje a bližší informace se budou objevovat v nových uvolněních Dartu [21].

6.7 Souhrn

Dart je komplexní jazyk, ve kterém se dají napsat webové aplikace bez přidání dalších knihoven a modulů. Nicméně pro provoz na klientské straně není rozšířená podpora pro Dart virtual machine, tudíž v praxi je využitelná pouze server-side implementace. V tom nejspíše spočívá problém pro využití, které se tím omezuje pouze na serverovou implementaci a popřípadě vývoji v jazyku Dart, ale nutnou kompilaci do JavaScriptu. To ale nemusí být na škodu z hlediska bezproblémového a intuitivního praní Dart kódu, které je k vývojáři velmi vstřícné.

Dart má své přednosti v mnoha ohledech. Ať už je to jeho syntaxe a přehlednost kódu, která se odvíjí od samotné struktury jazyka nebo také podporované komponenty jazyka, které jsou přívětivější, než v JavaScriptu. Mezi ty se určitě řadí například plná podpora objektově orientovaného programování včetně dědičnosti i standartního zápisu tříd a metod, pak také volitelné typování, jmenné konstruktory public a private proměnné, zkrácený zápis funkce a další. To poskytuje komfort psaní, přehlednost a znovupoužitelnost.

Velké plus má také jazyk v rychlosti reakci na dané problémy ve vývoji jazyka. Pokud je objevena chyba a nahlášena do systému, vývojáři Dart v co nejbližší době začnou pracovat na jejím odstranění a hlášením průběhu opravy chyby. Tudíž zde funguje i velká komunitní podpora, která se nepřímo podílí.

Rychlý vývoj má i své dvě stránky. Negativní věci jsou například adaptace programátora na změny, náhlá oznámení o změně zamýšleného vývoje (nadále nepodporování zamýšlené integrace Dart virtual machine do prohlížeče atd.) a další. Na druhou stranu se objevují skvělé věci jako například rychlý vývoj zásuvných modulů pro již existující vývojová prostředí, vývoj nástroje dart2js pro ještě lepší běh kompilovaného JavaScriptu nebo nemalá adaptace asynchronního kódu pomocí výrazů *async* a *await*, která byla velkou novinkou od uvedení jazyka a to ve verzi 1.9.

7 Testovací kód

Testování rychlosti zpracování kódu a vypsání požadovaného výsledku proběhlo na výpočtu Fibonacciho čísel a prvočísel skrze Eratosthenovo síto. Vstupy jsou zadávány skrze webový formulář. Zpracované výsledky se původně vypisovaly pod formulář, nicméně to zabíralo dvakrát více času, než se výsledek zobrazil, tudíž je vypisována jen doba běhu samotného zpracování výpočtů.

Testována je hlavně rychlost provádění kódu a to na prvních dvou příkladech. Dále pak rychlost zpracování množství dat a zvládnutí paměťové náročnosti, které je vyžadováno výpočty, zejména pak požadavek na prvočísla. Na posloupnosti čísel je také testována rekurze a to jak si s ní daný jazyk poradí a do jaké hloubky.

Paměťová náročnost je omezena jednak vývojovým prostředím, a také prohlížečem, ve kterém jsou kódy testovány. Dart Editor je sice odvozen od editoru Eclipse, ale nemá možnost nastavit využití paměti, takže tento faktor není ovlivnitelný. Je ale samozřejmě zjistitelné z chybové hlášky, ve které části kódu byla překročena paměťová dotace. Prohlížeč Dartium by měl mít stejné parametry, jako starší Chromium. Když se porovnal výsledek pro kompilovaný kód z dart2js nástroje v prohlížečích Dartium a aktualizovaný Chrome (verze 43), Dartium si vedlo hůře.

Stejné porovnávací kódy byly testovány na jednom počítači a ve stejném prohlížeči ve stejnou dobu a za stejných podmínek, co se týče běhu dalších procesů na pozadí. Zároveň byl dán k dispozici co největší výkonnostní prostor daného stroje.

7.1 HTML stránka

Základem pro zpracování algoritmů je webový formulář. Dart Editor disponuje základy projektů, které je možné zvolit při zakládání nového projektu. Pro tuto ukázku je zvolena šablona pro webovou aplikaci. Používané jsou prvky HTML5 a styly CSS.

Pro vstup od uživatele je využito pole **input** typu **number** [14]. Opatření typem **number** má za výhodu kontrolu vstupu. Pokud uživatel napíše číslo, vše je v pořádku. Když bude ale vložen text, nebude nijak zareagováno.

```
<input type="number" id="fibon" placeholder="Číslo: ">
```

Kód 16: Input pole
Zdroj: vlastní zpracování

Požadovaná výsledná čísla se vypíší do pole **<output>**, na které bude v Dart kódech odkazováno.

```
<p>
        Čas:
        <output id="outPrimeFib"></output><br>
        Výsledná čísla:<br><br>
        <output id="outFibon"></output>
</p>
```

Kód 15: Output pole
Zdroj: vlastní zpracování

7.2 Fibonacciho posloupnost

První algoritmus byl zvolen rekurzivní. Je to sice nejméně efektivní metoda, ale

```
int fibon(int f) => f <= 2 ? 1 : fibon(f - 2) + fibon(f - 1);
```

Kód 17: Rekurzivní funkce fibonacciho posloupnosti
Zdroj: vlastní zpracování

zajímavé zjištění, jak si s rekurzí daný jazyk poradí.

Zkrácený zápis funkce ušetří místa a působí přehledně na jeden řádek. Návratový typ je **int**. Proměnná *f* zastupuje zadané číslo uživatelem. Princip funkce je následující: pokud je *f* menší nebo rovno *2*, pak výsledek bude *1*. Jinak nastupuje rekurzivní volání funkce. Funkce je klasická, známá, ale zajímavé je její zapsání.

Druhý algoritmus výpočtu Fibonacciho posloupnosti byl zvolen explicitně zadaný vzorec.

Proměnná *F* definuje opakující se vzorec. Samotná funkce vrací **int**. Funkce **pow()** patří do knihovny **math** [13] a slouží k výpočtu exponenciální funkce. Funkce **sqrt()** patří také do knihovny **math** [13]. Na druhé straně zaokrouhlení pomocí **round()** spadá pod třídu **Num** a vrací **int** [13]. Pokud je potřeba vracet po zaokrouhlení **double**, je pro to funkce **roundToDouble()**.

```
double F = (1 + sqrt(5))/2;

int fibon2(int f) {
  double n = pow(F, f)/sqrt(5) - pow(1 - F, f)/sqrt(5);
  return n.round();
}
```

Kód 18: Explicitní funkce fibonacciho posloupnosti
Zdroj: vlastní zpracování

Dále je důležitá fáze sloučení funkčnosti s webovým formulářem.
Pro práci s webovými prvky slouží knihovna **html**. Pomocí **var** objektu *input* se bude odkazovat na prvek html dokumentu s určitým **id**, který je zde reprezentován symbolem „**#**“. To představuje mnohem lepší práci a hlavně přehlednost než možnosti v JavaScript. Pro ukázku následující kód v JavaScript vybere prvek s id. JavaScript má v tomto nevýhodu takovou, že je moc upovídaný. Potřebuje na na jednotlivé možnosti svou metodu, na rozdíl od Dartu, kde stačí jedna metoda pro výběr a specifikace daného typu prvku je již na vývojáři.
Na druhou stranu, práce s knihovnou JQuery je stejně tak jednoduchá jako s Dartem.

```
var input = querySelector('#fibon');
```

Kód 20: querySelector pro výběr prvku dle id v Dartu
Zdroj: vlastní zpracování

```
var x = document.getElemenetByID(“prvek“);
```

Kód 19: Výběr prvku dle id v jQuery
Zdroj: vlastní zpracování

```
$(document).ready(function(){
	$(“#prvek“).funkce();
});
```

Kód 21: Výběr prvku dle id v JavaScriptu
Zdroj: vlastní zpracování

Nyní k samotnému naslouchání události napsání čísla uživatelem. Na *input* prvek formuláře, do kterého uživatel napíše požadované číslo, se naváže **Key** listener. Uvnitř této funkce budou zapsané všechny podmínky potřebné pro chod funkcí. Při změně hodnot pole *input* se bude dynamicky měnit obsah výstupního pole.

První třídící podmínkou je, jestli pole vůbec nějakou hodnotu obsahuje. Jestli je hodnota přítomna, převede se na numerický typ **int**, pro tuto potřebu je v Dartu jednoduchá funkce **int.parse().** Obráceně, konverze čísla do **string**u se provází jednoduše skrze **toString()**. Podmínky a procházející cyklus **for** jsou zde stejné, jako např. v jazyce Java.

Přiřazení textu výstupnímu poli se provede pomocí zápisu **output.text**, přičemž *output* je proměnná pro html prvek **<output>**. **.text** aktualizuje obsah prvku, tzn., přepíše původní obsah. Na tento objekt se dá navěsit mnoho dalších metod, které samozřejmě slouží pro práci s html dokumentem.

```
input.onKeyUp.listen((_){
	…
});
```

Kód 22: KeyEvent pro input pole
Zdroj: vlastní zpracování

```
if(input.value == "") {
            output.text = "Prázdné pole.";
          } else {

             nums = int.parse(input.value);
             String outFib = "";

             if((nums > 0) && (nums <= 45)) {
                  for(int i = 1; i <= nums; ++i) {
                    outFib += fibon2(i).toString() + ", ";
                  }

               output.text = outFib + "...";

             } else {
               output.text = "Zadejte, prosím, pouze čísla od 1
do 45.";
             }

          }
```

Kód 23: Tělo onKeyUp.listen metody
Zdroj: vlastní zpracování

7.3 Eratosthenovo síto

Základem pro Eratosthenovo síto je dvojité procházení cykly a „vyhazování“ složených čísel. **Input** pole formuláře hlídá zadání čísla, takže stačí již hlídat jen vstup čísla větší jak 1 a menší jak hodní stanovený limit. Princip spočívá v nastavení složených čísel na hodnotu „0“. To pro pozdější vytřídění na vypsání.

Tento kód využívá datovou strukturu **List<int>**, na které nejdříve využívá její konstruktor **List.generate(int length, Function E generator(int index), {bool growable: true})**.

```
var erastList = new List<int>.generate(erastNum + 1, (int e) => e);
```

Kód 24: Vygenerování datové struktury List
Zdroj: vlastní zpracování

Tento konstruktor naplní list čísly zadané podmínkou. V tomto našem případě od 0 do zadaného čísla. První dvě čísla se ihned nastaví na hodnotu „0“ a nastává

procházení seznamu pomocí **forEach** cyklu. Následuje podmínka a další for cyklus pro násobky. Zde se nevyužívá nic odlišného od jiných jazyků.

Po vytřídění seznamu je zvoleno odejmutí složených čísel další užitečnou metodou nad strukturou **List** a to **where(Function bool test(E element))**. Odpadá nutnost použití roztřiďovacího cyklu. Testovací podmínkou bude, zda číslo nenabývá hodnotu 0. Zbývalo by již jen vypsat daná čísla do nějakého výstupu. K tomu by sloužil html prvek **<output>** a využila by se metoda **join(String separator="")**, která si projde seznam a na jednotlivé prvky aplikuje metodu **toString()** a následně je za pomoci definovaného separátoru zřetězí [13]. Kvůli problémům s dobou trvání vypsání výsledků do html stránky není ve výsledném provedení využito níže ukázaného kódu, který ukazuje použití metody **join()**. Místo toho je vyhodnocen a zobrazen pouze čas provádění kódu a vypsání bylo použito pouze pro kontrolu správnosti výsledků.

```
output.text= erastList.where((e) => e != 0).join(", ");
```

Kód 25: Podmínka pro prvočísla a výstupní text
Zdroj: vlastní zpracování

```
erastList[0] = erastList[1] = 0;
        for(int e in erastList) {
          if(e != 0) {
            for(int f = 2; (f * e) <= erastNum; f++) {
              erastList[f * e] = 0;
            }
          }
        }
```

Kód 26: ForEach cyklus a samotné síto
Zdroj: vlastní zpracování

7.4 Ukázka přehlednosti kódu

Tato část má přednést funkční kód, na kterém je ukázána použitelnost a přehlednost Dart jazyka. Jedná se o části kódu aplikace pro správu úkolů.

```
library stopar.model;

import 'package:polymer/polymer.dart';

class Ukol extends Observable {
  static bool POVINNY_NAZEV = true;
  static const MAX_NAZEV_DELKA = 40;
  static const MAX_POPIS_DELKA = 200;
  static const SOUCASNE = 'Současné';
  static const NEVYRIZENE = 'Nevyřízené';
  static const HOTOVE = 'Hotové';

  @observable int ukolID;
  @observable String nazev = '';
  @observable String status = NEVYRIZENE;
  @observable DateTime vytvoreno;
  @observable DateTime update;

  Ukol(this.nazev, this.popis, this.status);

  bool get ulozeno => ukolID != null;
}
```

Kód 27: Ukázkový kód, třída Úkol
Zdroj: DartEditor sample s úpravami

Na tomto kódu je pro ukázku prvků Dartu hned několik možností. Nejdříve je patrné přiřazení k určité knihovně a naimportování *polymer.dart* knihovny, která byla v projektu potřeba. Je tedy ihned jasné, jaké komponenty jsou potřeba v dané třídě.

Jasné a nezaměnitelné je definování třídy *Ukol*, která rozšiřuje třídu *Observable*, která je, mimojiné, používána na odpozorování změn na *Observable* typech, kterých je zde definovaných celkem pět, ty jsou blíže zmíněné dále v textu.
V samotné třídě jsou jako první definované proměnné *const*, které jsou stanovené. Dále jsou zde proměnné typů int, String a DateTime. Je jasně definované, jaké mají proměnné typy, což zlepšuje přehlednost a čitelnost kódu i následné ladění.
Konstruktor třídy je další položkou, které je velmi přehledně zapsaná. Zde je využit i zkrácený zápis a to definování parametrů rovnou v hlavičce kostruktoru, takže celý výraz se zapíše na jeden řádek.
Stejně tak jako funkce *get ulozeno*. Pro celý zápis by byly ještě za potřebí složené závorky a výraz *return*. To se dá ale obejít a využít zkráceného zápisu s využítím operátoru „=>“. Opět použito pro přehlednost a lepší čitelnost. Může se také říct, že se také šetří počet znaků napsaného kódu. Metoda má také uvedený návratový typ, který jasně udává výstup z metody, tudíž nevzniknou žádné nepřesnosti a nechtěné chyby.

```
attached() {
    submitPopisek = ukol.ulozeno ? 'Aktualizovat' : 'Vytvořit';
    statusSelectedIndex = taskStatusOptions.indexOf(ukol.status);
    if (!ukol.ulozeno) {
      statusSelectedIndex = taskStatusOptions.indexOf(ukol.status);
      previousStatus = ukol.status;
    }
  }
```

Kód 28:Ukázka kódu, funkce attached()
Zdroj: DartEditor sample s úpravami

Další zajímavou položkou v kódu je funkce *attached()*, která zde slouží pro operaci s popiskem elementu *submit* a statusem úkolu. Nejdříve je proměnné *submitPopisek* typu *String* přiřazen obsah a to na základě podmínky, která je zde implementována otazníkem a dvojtečkou a to v následujícím smyslu: „pokud je úkol uložen (*ukol.ulozeno*), pak přidej popisek ‚Aktualizovat‘, jinak přidej popisek ‚Vytvořit‘“. Rychlé zapsání podmínky na jeden řádek bez nutnosti rozepisování celé podmínkové konstrukce.

Přistupování k indexům je, jak lze očekávat, přes metodu *indexOf*, která vrací číslo indexu, pokud je prvek nalezen. Pokud není prvek nalezen, tak vrací hodnotu -1, přičemž se prochází od začátku do konce a vrací se první nalezený prvek.

```
bool validateTitle() {
    int len = ukol.nazev.length;
    bool valid = false;
    if (len == 0 && Ukol.POVINNY_NAZEV) {
      titleErrorMessage = 'Název  je povinný';
    } else if (len > maxTitleLength) {
      titleErrorMessage = 'Název musí být méně než $maxTitleLength
znaků';
    } else {
      titleErrorMessage = '';
      valid = true;
    }
    return valid;
  }
```

Kód 29: Ukázka kódu, funkce validateTitle()
Zdroj: DartEditor sample s úpravami

Další funkce v pořadí je *validateTitle()*, která vrací logickou hodnotu, tudíž určitě musí obsahovat výraz *return* s korespondující hodnotou. Další zajímavou věcí je vkládání proměnných do textového řetězce. Zde zapsáno pomocí výrazu *$maxTitleLength*, tudíž nemusí se použít skládání řetězce pomocí operátoru plus a přerušovat tak plynulost celého výrazu. Editor taktéž zvýrazní vloženou proměnnou, aby byla zřetelněji rozpoznatelná.

Je zde také vidět rozšíření *if-else* konstrukce pomocí *else if* a navázání tak další podmínky. Dart disponuje konstrukcí *switch*, nicméně pro použití na třech podmínkách vyjde zápis de facto stejně.

```
vytvorUkol() {
    DateTime now = new DateTime.now();
    var random = new Random();
    ukol.ukolID = random.nextInt(1000 * 1000);
    ukol.vytvoreno = now;
    ukol.update = now;
    appModel.ukoly.add(ukol);
  }
```

Kód 30: Ukázka kódu, funkce vytvorUkol()
Zdroj: DartEditor sample s úpravami

Následující úryvek kódu se týká funkce *vytvorUkol()*, jejíž úkolem je zpracování nově zadaného úkolu. Obsahuje metodu pro vytvoření aktuálního datumu a času *new DateTime.now()*. Dále je zde znázorněna práce s třídou *Random* pro náhodná čísla, zde je skrze metodu *nextInt* nastavena maximální hodnota. Random generuje nezáporná čísla. Data se poté ukládají do modelu aplikace. Třída *Random* vyžaduje import knihovny *dart:math*.

```
smazatUkol() {
    if (window.confirm('Jsi si jist smazáním?')) {
      appModel.ukoly.remove(ukol);
    }
  }
```

Kód 31: Ukázka kódu, funkce smazUkol()
Zdroj: DartEditor sample s úpravami

Výše zapsaná funkce *smazatUkol()* se dotazuje přes vyskakovací okno prohlížeče. Typ dotazu je potvrzovací a dán výrazem *confirm*, v závorce pak text dotazu. Typ okna *confirm* má ve vlastnostech pouze potvrzovací tlačítko navázané na odstranění úkolu ze seznamu úkolů.

```
if (!validateTitle() || !validateDescription() || !ukol.isValid) return;
```

Kód 32: Ukázkový kód, podmínka bez dané navrácené hodnoty
Zdroj: DartEditor sample s úpravami

Výše zapsaná řádka kódu také zobrazuje jednoduchost v čisté podobě. Podmínka *if* bez *else* části, kde je pouze navrácení hodnoty (v tomto případě se ani neprovede žádné navrácení) nepotřebuje ani zápis závorek. Samozřejmě, pro kombinace podmínek slouží klasické operátory pro *and* a *or* v zápisu „&&“ a „||“.
Jak může být vidět, Dart poskyte robusní prostředky pro tvorbu aplikací od těch nejjednodušších až po komplexní. Jazyk podává kompletní nástroj pro tvorbu znovupoužitelných konstrukcí s využitím objektově orientovaných principů. Výhodou jsou také zkrácené zápisy funkcí a podmínek, které urychlují práci a zlepšují přehlednost.

Ukázky jsou zkopírované z prostředí DartEditor a to pro barevné odlišení pro lepší orientaci. Kód v této podkapitole je čistě ukázkový pro zápis a konstrukce. Jeho spuštěním je webová aplikace pro osobní správu úkolů.

8 Testování

K vyhodnocování pracování kódu byl použit nástroj pro vývojáře, který je integrován do jádra prohlížeče od společnosti Google, který se zdál být rychlou a vhodnou volbou. Dále byly nastaveny stopky do vnitřku kódu. Dart disponuje třídou **Stopwatch**, která od započetí do svého konce měří čas, který je dále přístupný např. v uplynulé době celkově se všemi komponenty skrze třídu **Duration**. Bohužel metody **elapsedMilliseconds()** u **Stopwatch** a zároveň **toMillisecond()** u **Duration** zaokrouhlují na celé jednotky **int** [13]. Proto je vhodnější nechat vypsat celou dobu i s dalšími řády.

```
outputPrime.text = watch.elapsedMilliseconds.toString() + " " +
watch.elapsed.toString() + " " + watch.elapsed.inMicroseconds
        .toString();
```

Kód 33: Operace s třídou Stopwatch
Zdroj: vlastní zpracování

Problémem při vypisování všech čísel ze seznamu byl takový, že trvalo příliš dlouho, než se všechny vypsaly i přesto, že samotný výpočet trval krátkou dobu, přičemž šlo čistě jen o zobrazení vypsaného textu. Proto bylo vypisování textu čísel vypuštěno a místo toho je na výstupu jen doba běhu algoritmu a počet prvočísel, který se dostane z délky seznamu samotných prvočísel.

```
erastList = erastList.where((e) => e != 0);
output.text = erastList.length.toString();
```

Kód 34: Výpis počtu prvočísel
Zdroj: vlastní zpracování

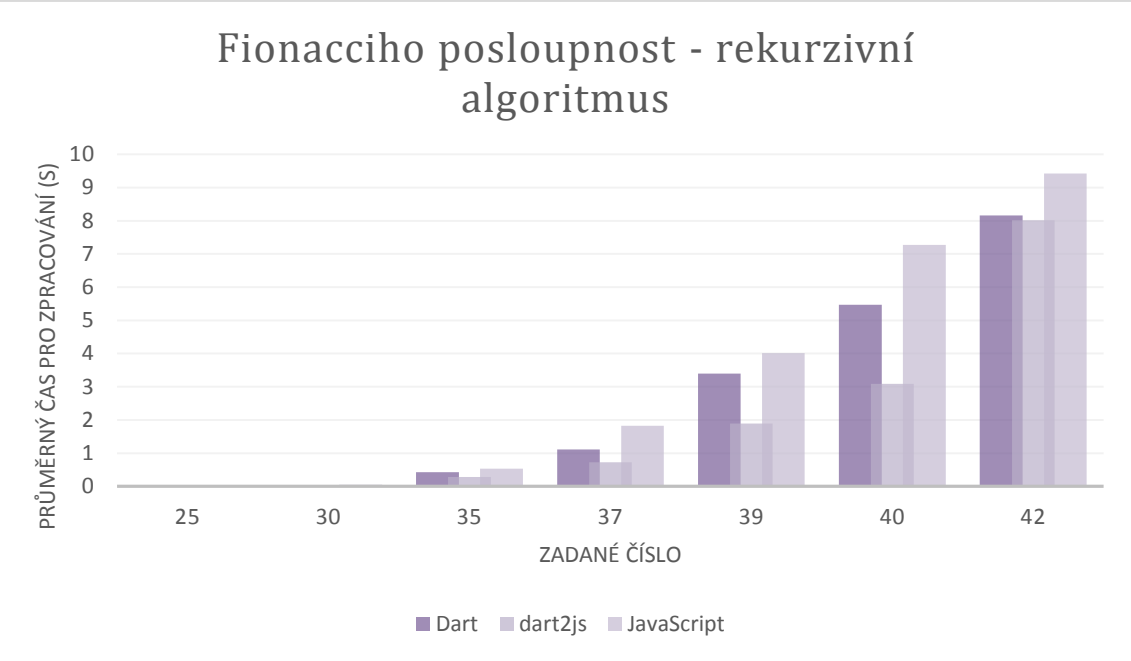
Jelikož se jednotlivá data od sebe významně nelišila, byla zprůměrována a průměrná hodnota použita v grafické podobě. Všechna nasbíraná data jsou zaznamenána v tabulkové podobě v příloze.

8.1 Fibonacciho posloupnost

Obr. 5: Fibonacciho posloupnost v prohlížeči Dartium
Zdroj: vlastní zpracování

8.1.1 Rekurzivní alogoritmus

Výsledky dopadly obdobně v souvislosti porovnání kódů jazyků. Kompilovaný JavaScript byl v některých případech nepatrně rychlejší, v některých případech zase rychlejší o hodně. JavaScript ve své surové formě dopadl hůře, než srovnávací kódy. Bohužel se ale nepodařilo získat data pro čísla do hodnoty 45 pro jazyk JavaScript a to ani kompilovaný, protože prohlížeč Dartium odmítl spolupráci ve formě výpadku ve všech z několika pokusů tyto data získat. Následující graf shrnuje tato data. Data pro hodnotu 45 v grafu nejsou zahrnuta a to kvůli výše zmíněnému problému.

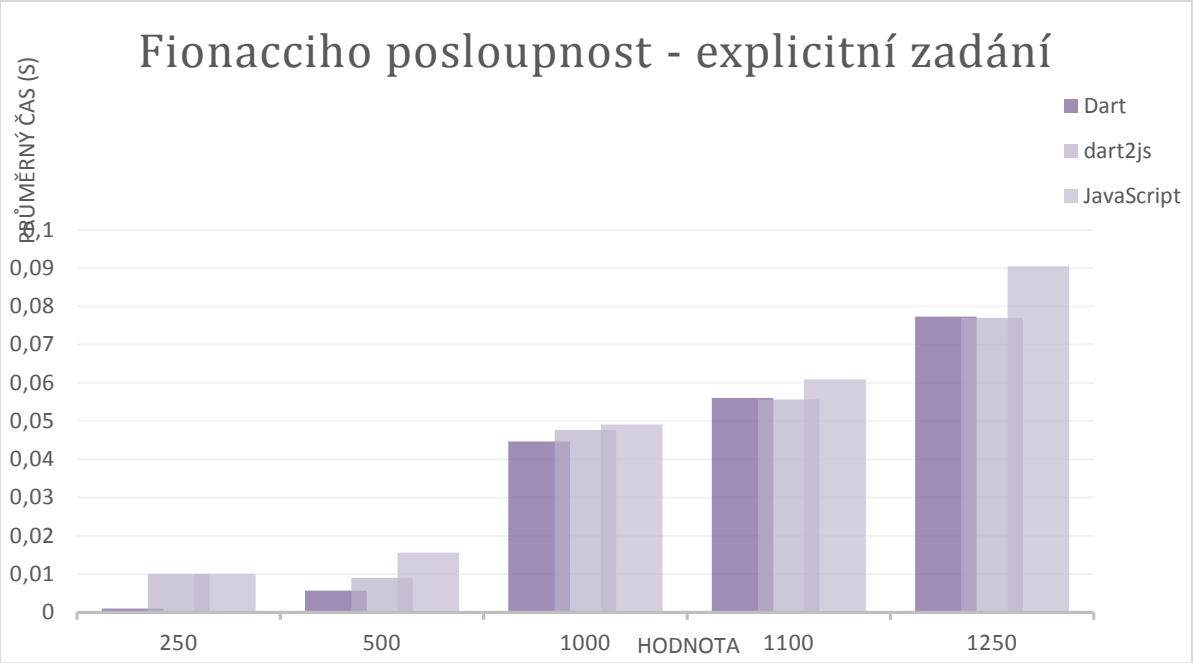


Graf 1: Srovnání běhu pro fibonacciho posloupnost - rekurzivní algoritmus
Zdroj: vlastní zpracování

8.1.2 Explicitní zadání

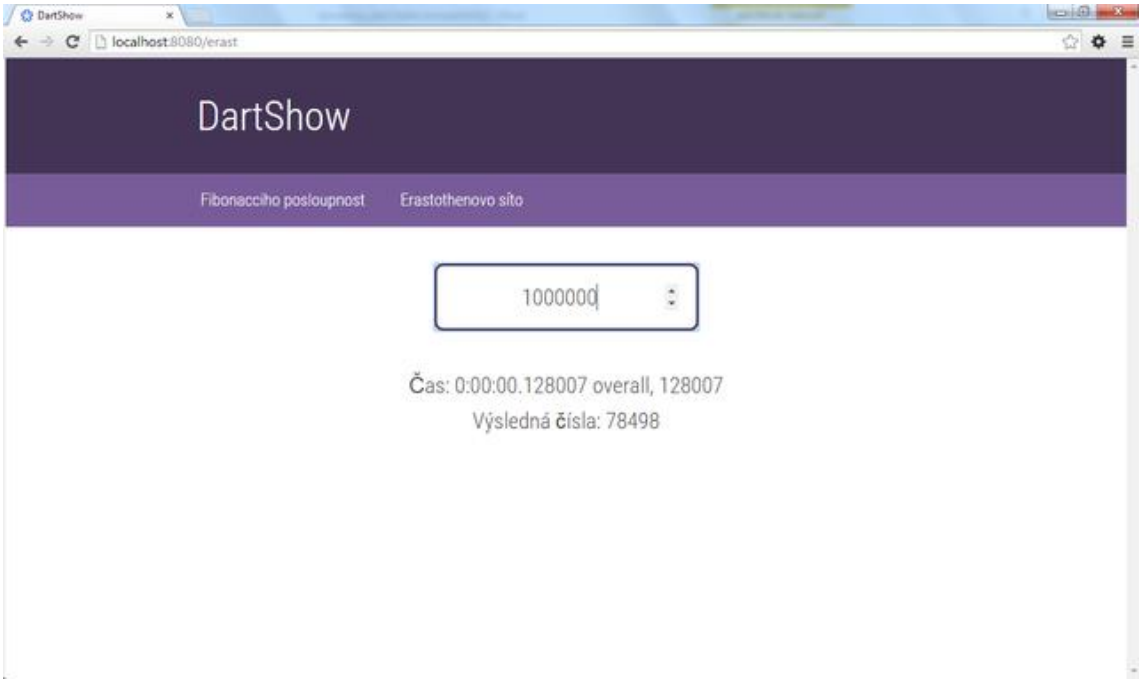
Pro explicitní zadání bylo klíčové zvolit rozsah pro zadaná čísla. Nakonec, vzhledem k paměti, padla horní hranice na číslo 1250 a dolní hranice, vzhledem k reálně zjistitelné době běhu algoritmu, na 250.

Výsledky dopadly obdobně pro Dart a z něho kompilovaný JavaScript, v některých případech nepatrně lépe dopadl Dart. Všechna data pro oba jazyky shrnuje následující graf.



Graf 2: Srovnání běhu pro fibonacciho posloupnost - explicitní zadání
Zdroj: vlastní zpracování

8.2 Eratosthenovo síto



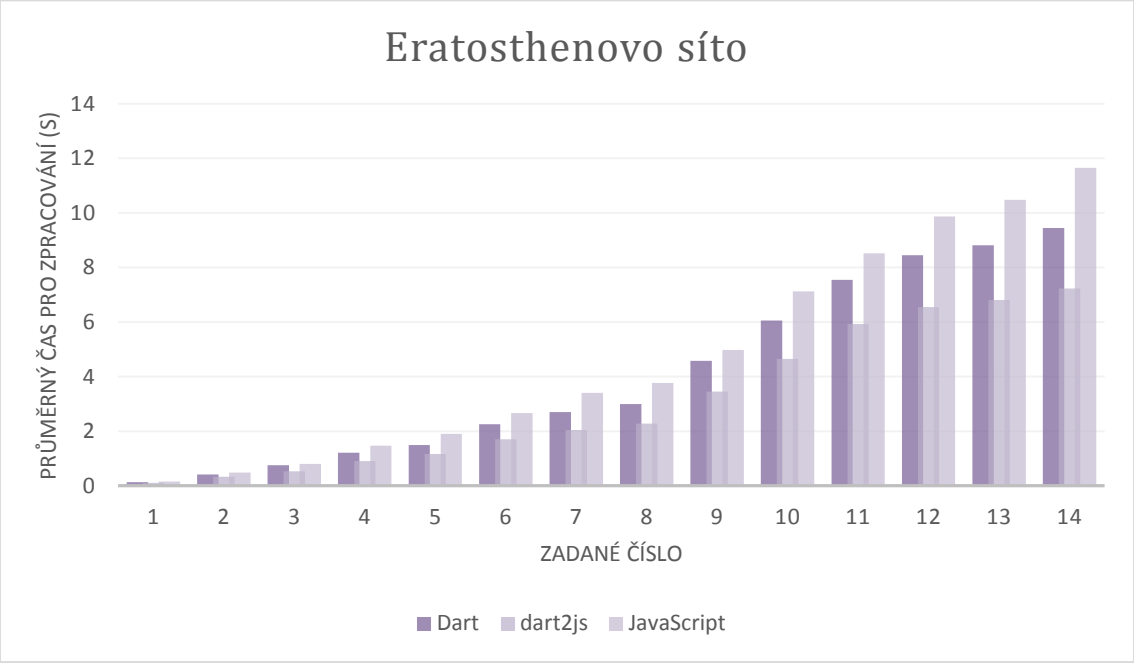
Obr. 6: Eratosthenovo síto v prohlížeči Dartium
Zdroj: vlastní zpracování

Hlavní otázkou je, jak velká čísla zvolit. Nemá smysl zkoušet malá čísla, která jsou vypočtená relativně ihned. Zaznamenatelný čas průběhu nastal až při 100 000. Nicméně ani při 1milionu se nepřebrodila hranice 1s. To nastává až při 10milionů, Při hranici 100milionů docházela paměť:

```
Exception: Out of Memory (package:DartFibonErast/erast.dart:40)
```

Kód 35: Out of Memory exception
Zdroj: vlastní zpracování

Horní hranice je tedy nastavena na 60 milionů.
Níže uvedený graf odpovídá hodnotám naměřeným hodnotám z kódu Dartu a JavaScriptu v obou formách.



Graf 3: Srovnání běhu pro Eratosthenovo síto
Zdroj: vlastní zpracování

Jak je z hodnot patrné, kompilovaný kód do JavaScriptu byl poněkud rychlejší, než kód Dartu, na vyšším množství dat je rozdíl veliký. V porovnání s čistým JavaScriptem je tento rozdíl o to větší, z grafu vypadá rozdíl obdobný mezi Dartem a nástrojem dart2js a Dartem a JavaScriptem. Zjištění z pohledu Dartu a nástroje dart2js je sice v rozporu s hypotézním tvrzením, ale nemusí to nutně znamenat zavrhnutí Dartu jako vhodnějšího prostředku.

U jazyku JavaScript a běhu v prohlížeči Dartium se několikrát stalo, že selhal a spadl. Tento jev nebyl, ve spolupráci s nástrojem debugger editoru DartEditor, zjištěn jako chybný z hlediska kódu.

9 Vývoj Dartu v průběhu zkoumání

Dart je nástroj celkem nový a v neustálém vývoji. Za dobu jeho zkoumání se jeho verze několikrát změnila. Nicméně upgrade verze je přirozená věc, nad kterou nemá význam se více pozastavovat. Zajímavější věcí je vývoj nástrojů, zamýšleného používání, nemluvě o rozšíření komponent.

Dart byl představen vývojářům v roce 2011, následně v roce 2013 byla uvolněna stabilní verze 1.0. V tomto stádiu byl jazyk ještě stále ve velkém vývoji, což s verzí 1.9 v půlce roku 2015 předčilo oznámení rozhodnutí, že Dart virtual machine se nebude integrovat do prohlížeče Chrome [16]. V této verzi se také představily *async* metody postavené na Future API.

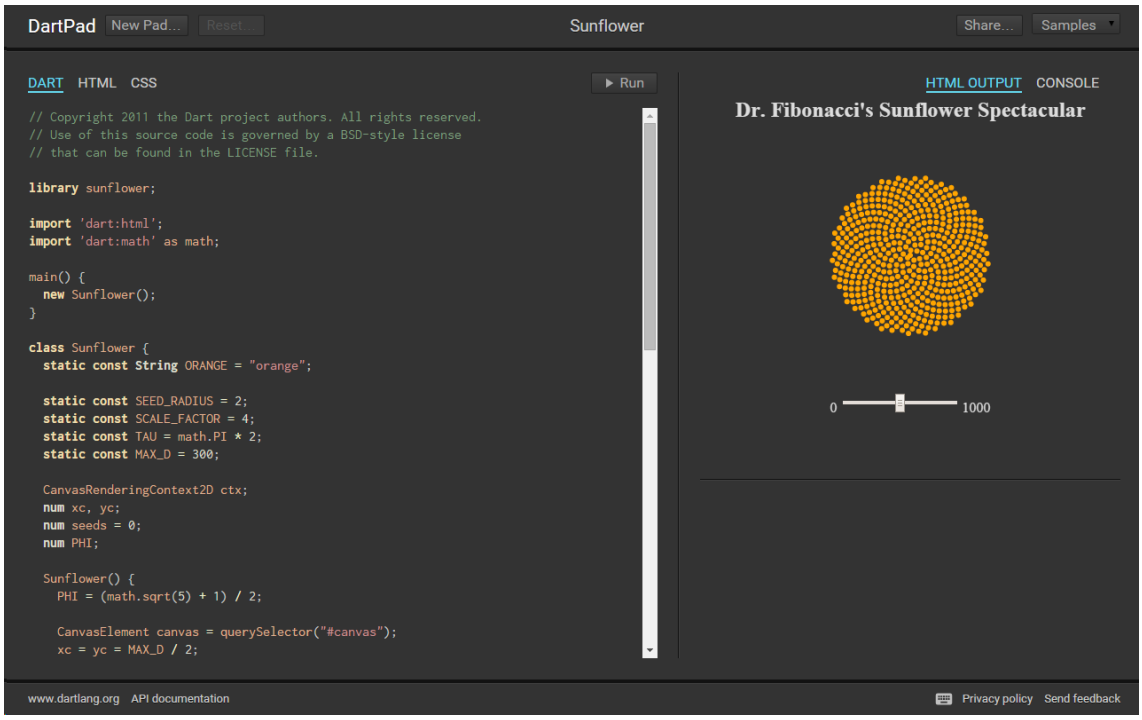
Dart editor byl pro raný vývoj aplikací dobrou volnou, vzhledem k novému prostředí. Editor poskytoval zázemí pro nové komponenty, ale s dobou, kterou tu Dart působí, přišly rozšíření pro stávající vývojová prostředí a tak mají vývojáři možnost zůstat u „svých“ prostředích a neinstalovat další. Prostředí, pro která jsou rozšíření k dispozici, přibývá, a tak v zásadě odpadá nutnost mít editor přímo pro jeden jazyk, což na začátku vývoje jazyka smysl mělo. Dart editor byl tudíž označen jako vysloužilý s verzí Dartu 1.11 [20]. K dispozici byl také nabídnut interaktivní online nástroj DartPad [24], který může sloužit především pro rychlé a pohodlné vyzkoušení a prozkoumání Dartu bez nutnosti instalace celého prostředí.

Knihovna Polymer dostala i svou Dart verzi, jak již bylo zmíněno. Polymer.dart je ale kompatibilní se zastaralou verzí Polymer, přičemž kompatibilita s verzí 1.0 je prozatím ve vývoji. Je jasné, že vývojáři Dartu chtějí kompatibilitu s různými prostředky vývoje webu, nicméně je v této oblasti poněkud těžké udržet krok. Tato aktualizace je tedy v očekávání.

Dalším bodem vývoje byl Dart virtual machine jako virtuální prostředí do prohlížeče, které bylo avizované pro integraci do prohlížeče Chrome a následnou snahu o integrace i do ostatních porhlížečů. Na konci března 2015 však bylo na oficiálním blogu Dartu oznámeno, že se již nepočítá s oficiálním zasazením DartVM do prohlížeče, ale bude se orientovat na vývoj právě nástroje dart2js [16].

„In order to do what's best for our users and the web, and not just Google Chrome, we will focus our web efforts on compiling Dart to JavaScript. We have decided not to

integrate the Dart VM into Chrome. Our new web strategy puts us on a path to deliver the features our users need to be more productive building web apps with Dart.“ (převzato z [16]).



Obr. 7: DartPad
Zdroj: [24]

Co se týče zdrojů, ze kterých jdou vytěžit informace ohledně jazyka, ty jsou trochu na vážkách. Hlavním zdrojem pro studium je zcela určitě oficiální web Dartu [8]. Zde jsou k nalezení základní informace, instalace prostředí, tutoriály, novinky a odkazy na další místa. Hlavní zdrojem pro samotné kódování je samozřejmě dokumentace [13], která je velmi přehledně a obsáhle zpracovaná. Jak již bylo zmíněno, podpora pro vývojáře z hlediska komunit je také rozsáhlá a to na různých sociálních sítích a podpůrných webech. Největší problém ale dává knižní podoba předávání informací. Několik knížek bylo vydáno v roce 2012, přičemž pár z nich bylo aktualizováno a to naposledy na podzim v roce 2014, kdy se objevily i nové knihy. Ano, na většinu věcí tato aktualizace stačí, nicméně jsou zde další věci, které jsou daleko novější a v těchto publikacích chybí. Nicméně, jsou na trhu k dispozici publikace, které odpovídají svým obsahem aktuální situaci, jako například nejnovější kniha Dart Essentials [25]. Tedy knihy tohoto typu zaměření jsou výhodnější pořizovat v elektronické podobě a také je v takové podobě publikovat.

10 Shrnutí a diskuze výsledků

Výsledky testování dopadly poněkud jinak, než bylo na počátku práce předpokládáno. Nicméně pro Dart jako programovací jazyk nevyšly vůbec špatně. V porovnání s jazykem JavaScript byl výkon obdobný až stejný, v některých případech kompilátoru dart2js malinko horší. Ale v žádném z výsledků doslova nepropadá. Pro detailnější porovnání by bylo potřeba využít větších rozměrů aplikace.

Co se týče samotného testování, byly zde nějaké pohnutky, které mohly být způsobeny prostředím, které se neustále vyvíjí a tak má stále na kontě nějaké poruchy. V aktivní komunitě podporující Dart je ale speciální internetová skupina věnujícím se hlášení a spravováním těchto chyb a tak se systém neustále zdokonaluje.

Konečný výsledek testování je tedy takový, že Dart ani nezklamal, ale ani se nijak zvlášť nepředvedl ve svých avizovaných přednostech. To se snad časem změní, obzvláště s nynějším zaměřením se na nástroj dart2js pro lepší výkon kompilovaného kódu.

11 Závěry a doporučení

Dart je novou technologií hlavně pro webový vývoj, snaha přinést novou myšlenku ve struktuře programování a vypořádat se s nedostatky jazyků navržených pro web minulých let, které jsou způsobené spíše jejich věkem a zamýšleným menším rozsahem použití.

Dart je celkově příjemným nástrojem pro vytváření webových částí i aplikací. Podobnost s ostatními jazyky usnadňuje naučení jazyka a nechává prostor pro soustředění se na ostatní aspekty jazyka. Budoucnost tento jazyk určitě má. Velké plus má tento jazyk v dokumentaci, která je velmi přívětivě zapsaná.

Co se týče porovnání s jazykem JavaScript, je Dart dalece příjemnější variantou jak na psaní, tak čtení a porozumění kódu. Přívětivost a použitelnost jsou jeho velké plus.

Vzhledem k výsledkům testování, které pro poslední verzi Dartu dopadly pro nástroj dart2js velmi pozitivně, není v obou verzích kódů drastický rozdíl. Nemusí to ovšem nutně znamenat zavrhnutí Dartu jako rychlejšího prostředku. Dart v použití s plnohodnotnými webovými aplikacemi může být vhodnější variantou. Navíc, toto testování bylo založeno na výběr jednoduchých matematických úkolů, složitější problémy by mohly dopadnout jinak, ale také obdobně, nemluvě o plnohodnotné webové aplikaci.

12 Seznam použité literatury

[1] Javascript: Historie. Samizdatová skripta [online]. [cit. 2014-07-09]. Dostupné z:http://skripta.lmsoft.cz/index.php?id=149

[2] JavaScript - výhody x nevýhody - Úvod do JavaScript - JavaScript. *Garth cz* [online]. 2014 [cit. 2014-07-09]. Dostupné z: http://www.garth.cz/uvod-do-javascriptu/javascript-vyhody-x-nevyhody/

[3] NEŠETŘIL, Jakub. JavaScript na serveru: Začínáme s Node.js. Zdroják [online]. 2010 [cit. 2014-07-09]. Dostupné z: http://www.zdrojak.cz/clanky/javascript-na-serveru-zaciname-s-node-js/

[4] Presentations -> Opening Keynote: Dart, a new programming language for structured web programming. Goto Conference [online]. 2014 [cit. 2014-07-09]. Dostupné z: http://gotocon.com/aarhus-2011/presentation/Opening%20Keynote:%20Dart,%20a%20new%20programming%20language%20for%20structured%20web%20programming

[5] STROM, Chris. *Dart for hipsters: fast, flexible, structured code for the modern web*. Dallas [u.a.]: Pragmatic Bookshelf, 2013. ISBN 978-193-7785-031.

[6] BUCKETT, Chris. *Dart in action: fast, flexible, structured code for the modern web*. London: Pearson Education [distributor], 2012, xxvi, 398 p. ISBN 16-172-9086-6.

[7] WALRATH, Kathy a Seth LADD. *Dart: up and running*. Farnham: O'Reilly, c2013, xv, 123 p. ISBN 14-493-3089-4.

[8] GOOGLE. *Dart: Structured web apps* [online]. 2014 [cit. 2014-07-09]. Dostupné z: https://www.dartlang.org/

[9] LADD, Seth. Live analysis results with WebStorm 10 and Dart. Dart News & Updates [online]. 2015 [cit. 2015-07-09]. Dostupné z: http://news.dartlang.org/2015/03/live-analysis-results-with-webstorm-10.html

[10] Support and Community | Dart: Structured web apps. GOOGLE. *Dart: Structured web apps* [online]. 2014 [cit. 2014-07-09]. Dostupné z: https://www.dartlang.org/support/

[11] Dart2js. Dart [online]. 2015 [cit. 2015-01-18]. Dostupné z: https://www.dartlang.org/tools/dart2js/

[12] Dart Tips, Ep 5: Collections In Dart. In: Dart: Structured web apps [online]. 2013 [cit. 2015-04-13]. Dostupné z: https://www.dartlang.org/dart-tips/dart-tips-ep-5.html

[13] Dart API Reference. Dart [online]. 2015 [cit. 2015-07-09]. Dostupné z: https://api.dartlang.org/apidocs/channels/stable/dartdoc-viewer/home

[14] CASTRO, Elizabeth a HYSLOP, Bruce. *HTML5 a CSS3: názorný průvodce tvorbou WWW stránek*. 1. vyd. Brno: Computer Press, 2012, 439 s. ISBN 978-80-251-3733-8.

[15] MARGORÍN, Marián. JQuery bez předchozích znalostí. Vyd. 1. Brno: Computer Press, 2011, 253 s. ISBN 978-80-251-3379-8.

[16] LADD, Seth. Dart for the Entire Web. Dart News & Updates [online]. 2015 [cit. 2015-07-10]. Dostupné z: http://news.dartlang.org/2015/03/dart-for-entire-web.html

[17] Runtime Modes - Dart Tips, Ep 2. Dart [online]. 2013 [cit. 2015-07-10]. Dostupné z: https://www.dartlang.org/dart-tips/dart-tips-ep-2.html

[18] Asynchronous Programming with Futures | Dart. Dart [online]. 2015 [cit. 2015-07-15]. Dostupné z: https://www.dartlang.org/docs/tutorials/futures/

[19] What is Polymer? - Polymer. Polymer [online]. 2015 [cit. 2015-07-20]. Dostupné z: https://www.polymer-project.org/1.0/docs/start/what-is-polymer.html

[20] LADD, Seth. The present and future of editors and IDEs for Dart. Dart News & Updates [online]. 2015 [cit. 2015-07-20]. Dostupné z: http://news.dartlang.org/2015/04/the-present-and-future-of-editors-and.html

[21] Polymer.dart [online]. 2015 [cit. 2015-07-20]. Dostupné z: https://www.dartlang.org/polymer/

[22] Define a Custom Element: Create a custom HTML element using Polymer. Dart [online]. 2015 [cit. 2015-07-20]. Dostupné z: https://www.dartlang.org/docs/tutorials/polymer-intro/

[23] MOORE, Kevin. Dart 1.9: The release you've been await-ing for. Dart News & Updates [online]. 2015 [cit. 2015-07-20]. Dostupné z: http://news.dartlang.org/2015/03/dart-19-release-youve-been-await-ing-for.html

[24] DartPad [online]. 2015 [cit. 2015-07-20]. Dostupné z: https://dartpad.dartlang.org/

[25] SIKORA, Martin. *Dart Essentials* [online]. PACKT Publishing, 2015 [cit. 2015-07-20]. ISBN 9781783989607. Dostupné z: https://www.packtpub.com/web-development/dart-essentials

[26] BALBAERT, Ivo a Dzenan RIDJANOVIC. *Learning Dart*. Birmingham: Packt Publishing, 2014, 1 online zdroj (388 stran). ISBN 978-1-84969-743-9.

[27] AKOPKOKHYANTS, Sergey. *Mastering Dart: Master the art of programming high-performance applications with Dart*. Birmingham: Packt Publishing, 2014. ISBN 978-1783989560.

[28] Introduction to Object-Oriented JavaScript. MOZILLA. *Mozilla Developer Network* [online]. 2015 [cit. 2015-07-24]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Introduction_to_Object-Oriented_JavaScript

[29] Dart:async API Docs. GOOGLE. *Dart API* [online]. 2015 [cit. 2015-07-31]. Dostupné z: https://api.dartlang.org/apidocs/channels/stable/dartdoc-viewer/dart:async

13 Přílohy

1) Medium DVD s přiloženým kódem
2) Tabulky dat získaných z měření běhu programu

Zadané číslo	Potřebný čas (sec)		
25	0,010 000	0,010 000	0,010 000
30	0,040 000	0,041 000	0,040 001
35	0,289 016	0,501 028	0,497 027
37	0,745 041	1,293 073	1,295 073
39	3,379 193	3,384 193	3,415 195
40	5,490 313	5,456 311	5,469 312
42	8,105 462	8,248 400	8,120 001
45	60,600 465	60,567 463	60,584 464

Tabulka 1: Rekurzivní zadání posloupnosti v Dartu, měření
Zdroj: vlastní zpracování

Zadané číslo	Potřebný čas (sec)		
25	0,010 000	0,010 000	0,010 000
30	0,040 000	0,040 000	0,040 001
35	0,281 015	0,280 015	0,275 015
37	0,727 040	0,726 042	0,732 041
39	1,881 206	1,890 107	1,899 108
40	3,162 180	3,054 174	3,058 173
42	8,012 457	8,022 458	8,022 459
45	N/A	N/A	N/A

Tabulka 2: Rekurzivní zadání posloupnosti v JS, měření
Zdroj: vlastní zpracování

Zadané číslo	Potřebný čas (sec)		
250	0, 010 000	0,001 000	0,001000
500	0,005 000	0,005 001	0,007 001
1000	0,042 003	0,044 003	0,048 003
1100	0,058 003	0,056 047	0,054 005
1250	0,079 004	0,076 004	0,077 005

Tabulka 3: Explicitní zadání posloupnosti v Dartu, měření
Zdroj: vlastní zpracování

Zadané číslo	Potřebný čas (sec)		
250	0,010 000	0,010 000	0,010 000
500	0,015 001	0,006 000	0,006 000
1000	0,048 003	0,047 002	0,048 002
1100	0,054 003	0,057 003	0,056 004
1250	0,078 004	0,076 005	0,077 005

Tabulka 4: Explicitní zadání posloupnosti v JS, měření
Zdroj: vlastní zpracování

Zadané číslo (mil.)	Potřebný čas (sec)		
1	0,136 010	0,138 008	0,131 008
3	0,392 023	0,433 025	0,413 024
5	0,775 044	0,704 040	0,772 044
8	1,245 072	1,215 069	1,178 067
10	1,537 088	1,486 085	1,469 084

15	2,292 131	2,236 128	2,234 126
18	2,784 159	2,663 152	2,672 151
20	3,092 177	2,959 169	2,951 169
30	4,738 271	4,500 257	4,483 257
40	5,977 342	6,103 349	6,106 349
50	7,375 422	7,654 438	7,615 436
55	8,451 483	8,456 484	8,439 483
57	8,876 508	8,785 502	8,782 503
60	9,503 544	9,520 545	9,301 532

Tabulka 5: Eratosthenovo síto Dart, měření
Zdroj: vlastní zpracování

Zadané číslo (mil.)	Potřebný čas (sec)		
1	0,109 201	0,109 200	0,109 205
3	0,322 216	0,318 216	0,359 219
5	0,530 228	0,535 229	0,542 229
8	0,906 250	0,902 250	0,899 249
10	1,215 258	1,132 263	1,141 264
15	1,716 296	1,698 296	1,695 265
18	2,068 316	2,036 315	2,034 116
20	2,291 329	2,268 328	2,272 329
30	3,453 198	3,449 395	3,443 395
40	4,666 465	4,659 464	4,637 463
50	5,964 139	5,913 536	5,891 535

55	6,552 375	6,544 573	6,555 573
57	6,832 589	6,797 988	6,791 587
60	7,250 613	7,244 613	7,199 610

Tabulka 6: Eratosthenovo síto JS, měření
Zdroj: vlastní zpracování

Příloha č. 1

FIM UHK
UNIVERZITA HRADEC KRÁLOVÉ
Fakulta informatiky a managementu
Rokitanského 62, 500 03 Hradec Králové, tel: 493 331 111, fax: 493 332 235

Zadání k závěrečné práci

Jméno a příjmení studenta: **Jaroslava Otmanová**

Obor studia: Aplikovaná informatika
Jméno a příjmení vedoucího práce: **Jiří Štěpánek**

Název práce:
Programovací jazyk Dart

Název práce v AJ:
Dart Programming Language

Podtitul práce:
Seznámení s jazykem Dart s praktickou ukázkou

Podtitul práce v AJ:
Getting to know the language Dart with a practical demonstration

Cíl práce: Seznámení s programovacím jazykem Dart. Popis jazyka, jeho využití a struktura. Pomocí praktické ukázky ve stylu části webové aplikace demonstrovat využití.

Osnova práce:
1. Úvod
2. Cíl práce
3. Teoretická část
   - 3.1 JavaScript
   - 3.2 Úvodní informace o Dartu
   - 3.3 Dart jako programovací jazyk
4. Praktická část
   - 4.1 Popis aplikace
   - 4.2 Struktura aplikace
5. Shrnutí
6. Závěr
7. Seznam použité literatury
8. Přílohy

Projednáno dne: 13.10.2014

Podpis studenta

Podpis vedoucího práce