EDICE
myslíme v ...

Vývoj aplikací ve SwiftUI

Od základů až po publikaci na App Store

FILIP VABROUŠEK

GRADA

GRADA

EDICE
myslíme v ...

FILIP VABROUŠEK

Vývoj aplikací ve SwiftUI

Od základů až po publikaci na App Store

GRADA
Publishing

Filip Vabroušek

Vývoj aplikací ve SwiftUI

Od základů až po publikaci na App Store

Vydala Grada Publishing, a.s.
U Průhonu 22, Praha 7
obchod@grada.cz, www.grada.cz
tel.: +420 234 264 401
jako svou 10 635. publikaci

Odpovědná redaktorka Věra Slavíková
Grafická úprava a sazba Filip Vabroušek
Vysázeno v systému Typst
Počet stran 192
První vydání, Praha 2026
Vytiskly Tiskárny Havlíčkův Brod a.s.

© Grada Publishing, a.s., 2026
Cover Design © Grada Publishing, a.s; Filip Vabroušek, 2026

Upozornění pro čtenáře a uživatele této knihy
Všechna práva vyhrazena. Žádná část této tištěné či elektronické knihy nesmí být reprodukována a šířena v papírové, elektronické či jiné podobě bez předchozího písemného souhlasu nakladatele. Neoprávněné užití této knihy bude trestně stíháno. Automatizovaná analýza textů nebo dat ve smyslu čl. 4 směrnice 2019/790/EU a použití této knihy k trénování AI jsou bez souhlasu nositele práv zakázány.

Názvy produktů, firem apod. použité v knize mohou být ochrannými známkami nebo registrovanými ochrannými známkami příslušných vlastníků.

ISBN 978-80-271-8528-3 (ePub)
ISBN 978-80-271-8527-6 (pdf)
ISBN 978-80-271-3117-4 (print)

Obsah

1 Základy ... 9
1.1 Operační systémy ... 9
1.1.1 Programovací jazyky na počítačích Apple ... 9
1.1.2 Mac ... 10
1.2 Instalace Xcode ... 10
1.3 Umělá inteligence při vývoji aplikací ... 11
1.3.1 Inteligentní funkce v Xcode ... 11
1.3.2 Generativní nástroje ... 11
1.4 Playground ... 11
1.4.1 Vytvoření prvního hřiště ... 11
1.4.2 Uspořádání oken ... 11
1.5 Frameworky ... 12
1.6 Application Loop ... 13

2 Framework Foundation ... 14
2.1 Úvod ... 14
2.2 Matematické funkce ... 14
2.3 Práce s jednotkami ... 14
2.3.1 Sečítání jednotek ... 15
2.4 Práce s daty a časem ... 15
2.4.1 Formátování data ... 16
2.4.2 Formátování data pro určitý region ... 16
2.4.3 Práce s časovými pásmy ... 16
2.5 Práce s textovými řetězci ... 17
2.5.1 Lokální formátování řetězců ... 17
2.5.2 Nahrazení části textového řetězce ... 17
2.5.3 Zjišťování pořadí slov ... 17
2.6 Časovač ... 18
2.7 Asynchronní programování a správa vláken ve Swiftu ... 18
2.7.1 Hlavní vs. vedlejší vlákno ... 18
2.7.2 Fronty operací ... 19
2.7.3 Grand Central Dispatch (GCD) ... 19
2.7.4 Quality of service ... 20
2.7.5 Lokální modely umělé inteligence ... 21

3 Základy a pohledy SwiftUI ... 22
3.1 Základy SwiftUI ... 22
3.1.1 View ... 23
3.1.2 Modifikátory ... 27
3.2 Základní vestavěné pohledy ... 32
3.2.1 Designový jazyk ... 32
3.2.2 Color View ... 32
3.2.3 Image ... 32
3.2.4 AsyncSequence ... 35
3.2.5 Framework UIKit ... 36
3.2.6 Under the hood ... 38

3.2.7 Základy layoutu ... 40
3.2.8 Mřížky ... 43
3.3 Tvorba tvarů a efektů ... 46
3.3.1 Základní tvary ... 46
3.3.2 Efekty ... 50
3.3.3 Materials ... 57

4 Stav, ovládací prvky a navigace ... 60
4.1 Práce se stavem aplikace a tvorba ovládacích prvků ... 60
4.1.1 State ... 60
4.1.2 Ovládací prvky ... 61
4.2 Navigace a modální okna ... 79
4.2.1 Pickers ... 80
4.2.2 Modalová okna ... 83
4.2.3 Navigace ... 87
4.2.4 Operace se soubory ... 96
4.3 Práce s gesty a animacemi ... 99
4.3.1 Gesta ... 99
4.3.2 Animace ... 104
4.3.3 Transitions ... 106
4.4 Tvorba seznamu položek a To-Do listu ... 107
4.4.1 List ... 107
4.4.2 To-Do ... 114
4.4.3 Sdílení dat mezi Views ... 118

5 Pokročilé techniky a tvorba aplikací ... 120
5.1 Čtení geometrie view a tvorba vlastní grafiky ... 120
5.1.1 Geometrie ... 120
5.1.2 GeometryEffect ... 123
5.1.3 PreferenceKey ... 131
5.1.4 Kreslení na Canvas ... 132
5.1.5 TimelineView ... 133
5.2 Vývoj multiplatformní aplikace pro iOS, macOS, watchOS a visionOS ... 134
5.2.1 Direktiva #if ... 134
5.2.2 watchOS ... 135
5.2.3 tvOS ... 137
5.2.4 visionOS ... 138
5.3 Tvorba běžecké aplikace a widgetů ... 148
5.3.1 Vrstva Model ... 149
5.3.2 Tvorba UI ... 153
5.3.3 Obrazovka se seznamem běhů ... 158
5.3.4 Tvorba Widgetu ... 161
5.3.5 Nasazení SwiftUI ve starších aplikacích ... 168
5.4 Zpřístupnění ... 174
5.4.1 Základní prvky zpřístupnění ... 174
5.4.2 Režim AssistiveAccess ... 175
5.4.3 Dynamické písmo a adaptivní rozvržení ... 175
5.5 Testování aplikace ... 176
5.6 Lokalizace aplikace ... 177

6 Framework Combine ... 178
6.1 Publishers ... 178
6.2 Subject ... 178
6.2.1 CurrentValueSubject ... 179
6.3 Operátory ... 180
6.4 Použití ve SwiftUI ... 186

7 Odeslání aplikace ke schválení ... 187
7.1 TestFlight ... 187
7.1.1 Vývojářský účet ... 187
7.1.2 Certificates and provisional profiles ... 187
7.2 App Store Connect ... 187
7.3 Nahrání aplikace na App Store Connect ... 188
7.3.1 Odeslání aplikace ke schválení ... 188

8 Závěr ... 189

Úvod

Otevíráte knížku, která vás chce naučit programovat moderním, objektově orientovaným jazykem *Swift* a tvořit vlastní aplikace pro zařízení *Apple*. Kniha je rozdělena do sedmi částí, které vás postupně provedou od prvních kroků až po zveřejnění hotové aplikace:

Základy
Úvodní část vás provede operačními systémy firmy *Apple*, programovacími jazyky a vývojovým prostředím *Xcode*. Získáte tak technologický základ, na kterém budeme dále stavět.

Framework Foundation
Knihovna *Foundation* usnadňuje řadu běžných úloh. Ukážeme si práci s datovými typy, matematickými funkcemi, převody jednotek a formátováním hodnot.

Základy a pohledy SwiftUI
Seznámíte se se základy deklarativního frameworku *SwiftUI* – s pohledy a modifikátory, vestavěnými prvky, layoutem a mřížkami i s tvorbou vlastních tvarů a efektů v designovém jazyce *Liquid Glass*.

Stav, ovládací prvky a navigace
Naučíte se pracovat se stavem aplikace a ovládacími prvky, s gesty a animacemi, navigací mezi obrazovkami i s tvorbou seznamů a sdílením dat mezi pohledy.

Pokročilé techniky a tvorba aplikací
Proniknete do pokročilejších technik – čtení geometrie a vlastní grafiky, vývoje multiplatformních aplikací pro *watchOS*, *tvOS* a *visionOS*, tvorby běžecké aplikace s widgety, zpřístupnění, testování a lokalizace.

Framework Combine
Reaktivní framework *Combine* slouží ke zpracování asynchronních událostí a datových toků. Vysvětlíme si jeho principy a propojení se *SwiftUI*.

Odeslání aplikace ke schválení
Závěrečná část vás provede zveřejněním hotové aplikace – od přípravy v prostředí *App Store Connect* až po odeslání aplikace ke schválení společnosti *Apple*.

Ing. Filip Vabroušek

Filip Vabroušek je softwarový inženýr a full-stack vývojář mobilních, webových a desktopových aplikací. V současné době pracuje jako učitel na FAI UTB, a zabývá se efektivním využitím AI pro vývojáře a automatizací procesů pomocí agentické AI. Mimo programování se zabývá 2D/3D motion designem a procedurálními simulacemi v programu Houdini. Závodně se věnuje atletice (běh na dlouhé tratě). Jeho práce je dostupná na webu filipvabrousek.com.

1 Základy

V této kapitole se seznámíme s operačními systémy *Apple*, pro něž budeme v knize vyvíjet aplikace. Seznámíme se také s potřebnými nástroji a jejich instalací.

1.1 Operační systémy

Operační systém představuje rozhraní mezi hardwarem počítače a uživatelskými aplikacemi. Je uložený na pevném disku počítače a úlohy vykonává pomocí procesoru. Dokáže uživatele odstínit od hardwaru počítače. Stejný program tak bude fungovat i na jiném počítači s odlišným hardware, ale se stejným operačním systémem. Nejrozšířenějším operačním systémem je systém *Windows*, který běží na procesorech *Intel* nebo *AMD*. Mezi dalším známé systémy patří například *Linux* s otevřeným zdrojovým kódem.

Apple pro své platformy vytvořil šest hlavních operačních systémů, tyto operační systémy jsou přizpůsobeny zařízení, na kterých běží:

- *iOS* pro telefony *iPhone*,
- *iPadOS* pro tablety *iPad* – jedná se o přejmenovanou a upravenou verzi *iOS*,
- *macOS* pro počítače Mac,
- *watchOS* pro chytré hodinky *Apple Watch*,
- *tvOS* pro multimediální settop box zařízení *AppleTV*,
- *visionOS* pro AR/VR headset, který Apple označuje jako prostorový počítač (*Spatial Computer*) *Apple Vision Pro*.

1.1.1 Programovací jazyky na počítačích Apple

Do počátku 80. let byl veškerý systémový software pro počítače *Apple* vyvíjen převážně v assembleru – nejprve pro mikroprocesor MOS 6502, později pro *Motorola* 68000.

Assembler představuje nízkoúrovňový jazyk symbolických instrukcí, které odpovídají strojovým operacím procesoru. Výhodou assembleru je maximální efektivita a kontrola nad hardwarem, avšak za cenu nízké přehlednosti, obtížné údržby a výrazné závislosti na konkrétní architektuře procesoru.

Počátkem 80. let vznikl jazyk Objective-C, který vytvořili Brad Cox a Tom Love; *Apple* jej převzal později prostřednictvím společnosti *NeXT*. Tento jazyk vycházel z tehdy dominantního jazyka C, ke kterému přidal objektově orientované prvky inspirované jazykem Smalltalk. Objective-C tak umožnil vytvářet modulární a znovupoužitelný kód prostřednictvím tříd, dědičnosti a zasílání zpráv.

Navzdory své technické vyspělosti si však jazyk Objective-C nezískal širší popularitu mimo prostředí Applu. Důvodem byla především jeho neobvyklá syntaxe, nižší čitelnost a vyšší náchylnost k chybám, zejména v oblasti správy paměti a práce s ukazateli.

Na konferenci WWDC 2014 *Apple* představil moderní jazyk Swift, který byl koncipován jako nástupce Objective-C. Swift kombinuje výkon jazyka C s bezpečností, přehledností a moderní syntaxí inspirovanou jazyky, jako *Python*, *Ruby* či *Rust*.

Cílem *Swiftu* je zvýšit produktivitu vývojářů, minimalizovat běhové chyby a současně zachovat vysoký výkon nativního kódu. Jazyk byl od počátku navržen tak, aby byl interoperabilní s *Objective-C*, což umožnilo plynulý přechod existujících projektů.

Z historického hlediska je vhodné připomenout i vývojovou linii, která vedla ke vzniku *Objective-C*. Jazyk *Simula* z konce 60. let položil základy objektově orientovaného programování, které později rozvinul *Smalltalk*. *Objective-C* pak tyto principy převzal a aplikoval na jazyk C, čímž vznikl hybridní jazyk spojující efektivitu nízkoúrovňového programování s konceptem objektů a zpráv.

Swift na tuto tradici navazuje, avšak zároveň ji překračuje – odstraňuje historické nedostatky předchozích jazyků, zavádí silný typový systém, automatickou správu paměti a důraz na bezpečnost a čitelnost kódu.

1.1.2 Mac

Pro vývoj aplikace v *Xcode* a její nahrání do obchodu *App Store* je potřeba *macOS nebo iPadOS*. Aplikace je možné vytvořit i na *iPadu*, ale pro lepší vývojářský zážitek je lepší *Mac*. I když je možné *Mac* používat v cloudu, doporučuji si zakoupit jakýkoli *Mac*, který podporuje nejnovější operační systém. V době psaní knihy *macOS Tahoe* (26.0). Pro co nejdelší softwarovou podporu, doporučuji zakoupit nejnovější *Mac*, který byl představený v aktuálním roce (v době psaní knihy *MacBook Neo* (2026) nebo *MacBook Air* s procesorem M5 (2026) nebo novější).

Primární integrované vývojové prostředí (používá se zkratka IDE z anglického *Integrated Development Environment*) pro vývoj aplikací pro všechny platformy. *Xcode* lze stáhnout zdarma z *Mac App Store*.

Vývojářský-účet
Xcode je možné používat i bez vývojářského účtu. Na vývoj na reálném zařízení stačí běžný *Apple Account*, na nahrání do *App Storu* a přístup k beta verzím software je potřeba mít vývojářský účet.

Apple developer členství
Pro zveřejnění aplikace na *App Store* je potřeba vývojářský účet. *Apple* si za tento účet účtuje 99 $ na rok. Na adrese https://developer.apple.com/programs/enroll/ je možné ho vytvořit.

Mobilní zařízení Apple
I když je možné aplikaci testovat na simulátoru, je před vydáním doporučováno otestovat aplikaci na reálném *Apple* zařízení. Toto zařízení by mělo podporovat *iOS 27*, *iPadOS 27* nebo novější. Pro vývoj aplikací pro *watchOS 27* na reálném zařízení jsou potřeba *Apple Watch*.

1.2 Instalace Xcode

Xcode lze stáhnout zdarma z *Mac App Store*.

1. Na Macu otevřeme aplikaci *App Store*.
2. Do vyhledávacího pole zadáme *Xcode*.
3. Nainstalujeme *Xcode* klepnutím na tlačítko Get/Získat.

Po instalaci si můžeme *Xcode* přizpůsobit. Nastavení otevřeme přes menu *Xcode* → *Settings* (úplně vlevo v horní liště), případně klávesovou zkratkou `Cmd + ,`. V předvolbách lze zvolit barevné schéma a písmo editoru a přepínat mezi světlým a tmavým režimem.

1.3 Umělá inteligence při vývoji aplikací

Vývoj aplikací dnes stále častěji doprovázejí nástroje umělé inteligence. Pomáhají psát i vysvětlovat kód, navrhovat řešení a urychlovat rutinní práci – a to jak přímo v *Xcode*, tak v podobě samostatných nástrojů.

1.3.1 Inteligentní funkce v Xcode

Xcode obsahuje prediktivní doplňování kódu, které přímo na zařízení napovídá celé řádky i bloky podle kontextu projektu. Novější verze umožňují připojit do editoru také jazykové modely (například *ChatGPT* nebo modely dalších poskytovatelů). Model pak dokáže vysvětlit chybu, navrhnout opravu, doplnit kód či vygenerovat dokumentaci, a to s ohledem na obsah celého projektu.

1.3.2 Generativní nástroje

Vedle funkcí vestavěných přímo v *Xcode* existují i samostatné generativní nástroje, které pracují s celým projektem z příkazové řádky nebo z editoru. *Claude Code* (od společnosti *Anthropic*) a *Codex* (od společnosti *OpenAI*) patří mezi tzv. agentické nástroje, kterých existuje celá řada: zadáme jim úkol v přirozeném jazyce a ony samy procházejí soubory projektu, navrhují a provádějí úpravy, spouštějí příkazy a ověřují výsledek. Hodí se k psaní nového kódu, refaktorování, hledání chyb i k vysvětlování cizích projektů. Představují užitečného pomocníka, výsledný kód je však vždy vhodné zkontrolovat a otestovat.

1.4 Playground

Vývojové prostředí *Xcode* pracuje ve dvou základních režimech. První režim slouží k plnohodnotnému vývoji aplikací. Vyžaduje vytvoření celého projektu, který má složitou strukturu sestávající z mnoha souborů a nastavení, což je pro počáteční seznamování se s jazykem zbytečně komplikované.

Druhým režimem je tzv. *Playground*. Je to interaktivní prostředí určené k experimentům, testování algoritmů a rychlému ověřování nápadů. V tomto prostředí se kód ihned po napsání překládá a spouští, přičemž výsledky vidíme okamžitě bez nutnosti sestavovat celou aplikaci. V této učebnici budeme pro výklad základů jazyka využívat výhradně tento režim.

1.4.1 Vytvoření prvního hřiště

Abychom mohli začít psát kód, musíme nejprve spustit aplikaci *Xcode*. Jakmile aplikace běží, vytvoříme nové prostředí pro experimenty následovně:

1. V horní liště zvolíme **File → New → Project → Playground**.
2. V otevřeném okně vybereme šablonu pro *iOS* a v ní zvolíme Blank (prázdný dokument).
3. Po kliknutí na Next zadáme název souboru, vybereme složku, kam se má uložit, a stiskneme Create.

1.4.2 Uspořádání oken

Pracovní plocha prostředí *Playground* je rozdělena do několika částí:

- **Panel navigátoru (vlevo):** Zobrazuje soubory, které jsou součástí daného hřiště. Na začátku zde uvidíte pouze hlavní soubor s kódem.
- **Editor kódu (uprostřed):** Hlavní okno, do kterého budeme psát náš program.
- **Panel náhledu (vpravo):** V této části se u každého řádku okamžitě zobrazuje výsledek operace (např. hodnota, která byla do proměnné uložena).

- **Konzole (dole):** Zde se zobrazuje textový výstup programu, například výsledky volání funkce `print`. Pokud konzoli nevidíte, lze ji zobrazit ikonou v pravém horním rohu nebo zkratkou `Shift + Cmd + C`.

Než začneme, vymažeme veškerý text, který *Xcode* do nového souboru automaticky vygenerovalo, a napíšeme na první řádek náš první program:

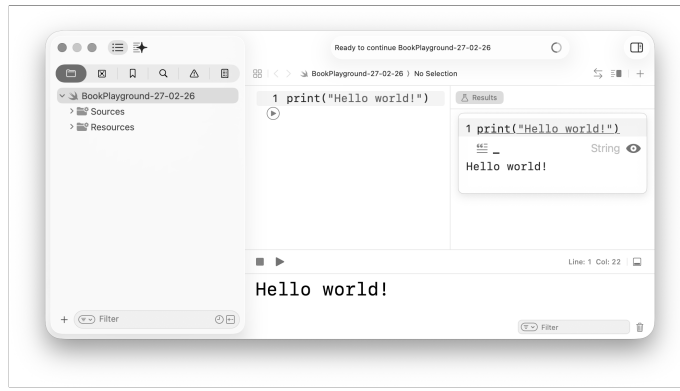
```
print("Hello world!")
```

Ihned po dopsání tohoto řádku se v panelu náhledu vpravo objeví text

```
Hello world!
```

a tentýž text se vypíše i do konzole ve spodní části okna.

Při psaní řádků kódu by se měl tento výpis sám aktualizovat. Pokud se kód nespustí automaticky, stiskněte tlačítko Play (modrý trojúhelník), které se objeví vedle čísla řádku, nebo použijeme zkratku CMD+SHIFT+ENTER.



Obrázek 1.1: Hello world program v prostředí Playground

1.5 Frameworky

Vývoj jednoduché aplikace s pomocí pouhého programovacího jazyka by mohl trvat dlouhou dobu. Pro usnadnění vývoje se používají frameworky. Framework je softwarový balíček, který může obsahovat předepsané bloky kódu a nástroje pro řízení stavu aplikace. Pro vývoj aplikací jsou tyto frameworky a jejich API nezbytné, proto se jimi budeme zabývat po zbytek této knihy. *Apple* postupně uvedl následující frameworky:

- **Foundation** – umožnil ukládání dat a základní operace s nimi.
- **AppKit** – byl vyvinutý pro tvorbu uživatelských rozhraní na *macOS*.
- **UIKit** – byl představen v roce 2008 a umožnil tvorbu uživatelských rozhraní pro platformu *iOS*. Je velmi rozšířený a používá se dodnes.
- **CoreGraphics** – byl naprogramován v jazyce C a slouží k 2D kreslení. Je integrován v novějších frameworcích.
- **SwiftUI** – v roce 2019 na konferenci *WWDC Apple* byl uveden nový framework **SwiftUI**, který výrazně zjednodušil vývoj aplikací. Právě tímto frameworkem se zabývá tato kniha.

1.6 Application Loop

Aplikace s grafickým uživatelským rozhraním (GUI) fungují jinak než jednoduché skripty, které se provedou a okamžitě ukončí. Aby taková aplikace mohla vzniknout a běžet, musí projít několika kroky.

Nejprve kompilátor v prostředí *Xcode* vezme náš zdrojový kód a propojí jej s knihovnami, které využíváme. K tomu přidá základní instrukce nutné pro komunikaci s operačním systémem. Výsledkem tohoto procesu je spustitelný program (executable), který můžeme nahrát do simulátoru, na reálné zařízení nebo publikovat na *App Store*.

Jakmile aplikaci spustíme, operační systém zahájí proces, který vstoupí do tzv. smyčky událostí (application loop). Jedná se o cyklus, který udržuje aplikaci neustále aktivní. Místo toho, aby program skončil, smyčka čeká na příchozí události – například dotyk prstu na obrazovku nebo stisknutí tlačítka – a zajišťuje jejich zpracování.

2 Framework Foundation

V této kapitole se seznámíme s frameworkem *Foundation*, který nám usnadní a urychlí práci s vestavěnými datovými typy. Jedná se především o převody dat, fyzikálních jednotek, měn a další.

2.1 Úvod

Framework *Foundation*, jenž je napsaný v jazyce Objective-C, usnadňuje časté operace, které potřebujeme řešit v naší aplikaci, jako například:

- používání matematických funkcí,
- převody jednotek,
- převody časů mezi regiony,
- převody jednotek měn – a mnohé další.

Pokud chceme framework použít v našem kódu, importujeme ho pomocí deklarace:

```
import Foundation
```

2.2 Matematické funkce

Mezi základní matematické funkce patří zjištění minima a maxima ze dvou čísel. Ve *Swiftu* toho lze docílit pomocí funkcí `min` a `max`.

```
min(22, 10)  // 10
max(22, 10)  // 22
```

Mezi další funkce se řadí:

- `pow(a, b)` – vrátí číslo a umocněné na mocninu b,
- `sqrt(a)` – vrátí odmocninu z čísla,
- `log(a)` – vrátí přirozený logaritmus z čísla a.

```
pow(3, 3)
sqrt(23)
log(21.0)
```

2.3 Práce s jednotkami

Framework *Foundation* nám jednoduše umožní formátovat údaje, např. měnu: Nejprve pomocí třídy `NSNumber` definujeme proměnnou `amount` a poté instanci třídy `NumberFormatter`, které přiřadíme formátovací styl na `.currency` (měna).

Dále dosadíme proměnnou `amount` do funkce `string(from:)` a získáme výsledek `$9.41` (`9.41` amerického dolaru).

```
var amount = NSNumber(value: 9.41)
let format = NumberFormatter()
format.numberStyle = .currency
let price = format.string(from: amount)  //$9.41
```

Framework *Foundation* nám také umožňuje jednoduše převádět jednotky. Na tomto příkladu je ukázán převod z kilometrů na míle.

```
let distance = Measurement(value: 100, unit: UnitLength.kilometers)
let miles = distance.converted(to: .miles)

// 62.13711922373339 mi
```

2.3.1 Sečítání jednotek

Je také jednoduché sečíst jednotky různých délek. V tomto případě zadáváme 2 metry a 38 centimetrů a pomocí třídy `Measurement` a sečítáme je v proměnné `height`. Výsledek je `2.38` m, tedy 2 metry a 38 centimetrů.

```
let meters = Measurement(value: 2, unit: UnitLength.meters)
let centimeters = Measurement(value: 38, unit: UnitLength.centimeters)
let height = meters + centimeters
print(height)  // 2.38 m\n
```

2.4 Práce s daty a časem

Pro zobrazení aktuálního roku, hodiny, sekundy, nebo čísla aktuálního měsíce, lze využít struktury `Calendar`. Na prvním řádku si definujeme komponenty `.year`, `.month`, `.day` a do argumentu `from` napíšeme `Date()`, což nám vrátí aktuální datum. Na dalších řádcích si z data přečteme číslo měsíce, a napíšeme ho do konzole. Protože se jedná o `Optional`, musíme hodnotu konstanty rozbalit pomocí `!`.

```
let calendar = Calendar.current.dateComponents(
    [.year, .month, .day], from: Date())
let monthNumber = calendar.month
print("This month has a number \(monthNumber!).")
```

Pokud například chceme zobrazit aktuální čas v americkém formátu, lze použít třídu `Locale` s identifikátorem `en_US_POSIX`. Čas `15:38` v ČR bude zapsán jako `3:38 PM`.

```
let today = Date()
let hd = DateFormatter()
hd.locale = Locale(identifier: "en_US_POSIX")
hd.dateFormat = "hh:mm a"
hd.amSymbol = "AM"
hd.pmSymbol = "PM"
let USATime = hd.string(from: today)  // 3:38 PM
```

2.4.1 Formátování data

Nejprve definujeme dnešní datum do proměnné `today` pomocí struktury `Date`. Pomocí struktury `DateFormatter` můžeme naformátovat datum na formát, jaký si přejeme. V tomto případě nastavíme formát data `dd/MM/yyyy`, (den/měsíc/rok). Poté použijeme metodu `formatter.string(from: today)`, která nám vrátí datum v požadovaném formátu.

```
import Foundation

let today = Date()
let formatter = DateFormatter()
formatter.dateFormat = "dd/MM/yyyy"
let formattedDate = formatter.string(from: today)
```

2.4.2 Formátování data pro určitý region

Kromě časových pásem můžeme datum formátovat také podle konkrétního regionu a jeho jazykových zvyklostí. K tomu slouží struktura `Locale`. Tato konfigurace zajistí, že názvy měsíců, dnů a pořadí prvků v datu budou odpovídat dané zemi.

V následujícím příkladu nastavíme pomocí identifikátoru `zh_CN` lokalizaci na čínskou. Tuto lokalizaci poté přiřadíme do vlastnosti `.locale` našeho formátovače. Výsledný řetězec uložený v konstantě `chinaDate` bude obsahovat datum zapsané v čínských znacích a formátu.

```
let today = Date()
let formatter = DateFormatter()
formatter.dateStyle = .full
let chinaLocale = Locale(identifier: "zh_CN")
formatter.locale = chinaLocale
let date = formatter.string(from: today)
```

2.4.3 Práce s časovými pásmy

Při práci s datem a časem je často nutné zohlednit časová pásma, zejména pokud aplikace zpracovává data z různých částí světa. V následujícím příkladu si ukážeme, jak zformátovat aktuální čas pro konkrétní region.

Nejprve definujeme objekt `TimeZone` pomocí identifikátoru (např. `Europe/Prague`). Dále vytvoříme aktuální datum a instanci třídy `DateFormatter`, u které nastavíme styl data na `.long` (podrobné datum) a styl času na `.short` (krátký čas). Aby formátovač správně vypočítal posun hodin, přiřadíme mu do vlastnosti `timeZone` naši pražskou časovou zónu. Výsledný formátovaný řetězec získáme zavoláním metody `string(from:)`.

```
let pragueTimeZone = TimeZone(identifier: "Europe/Prague")
let today = Date()
let formatter = DateFormatter()
formatter.dateStyle = .long
formatter.timeStyle = .short
formatter.timeZone = pragueTimeZone
let pragueDate = formatter.string(from: today)

// "October 6, 2020 at 10:19 PM"
```

2.5 Práce s textovými řetězci

Ve Swiftu je práce s textovými řetězci jednoduchá, přehledná a velmi flexibilní. Je možné snadno vytvářet, upravovat a analyzovat text, a to bez nutnosti složité manipulace s jednotlivými znaky. Textové řetězce lze například formátovat podle požadovaného vzoru, nahrazovat vybrané části textu novým obsahem, nebo porovnávat pořadí slov a řadit text podle abecedy či lokálních pravidel.

2.5.1 Lokální formátování řetězců

Metoda `String.localizedStringWithFormat` umožňuje formátovat text s ohledem na regionální nastavení uživatele (lokalizaci). To je zásadní zejména u čísel, kde se automaticky mění oddělovač desetinných míst (např. tečka v USA vs. čárka v ČR).

Formátovací značka `%.2f` určuje, že se má hodnota typu `Double` nebo `Float` vypsat s přesností na dvě desetinná místa.

```
let amount = 12.341
let decimal = String.localizedStringWithFormat(
    "Decimals: %.2f", amount)

// "Decimals: 12.34"
```

2.5.2 Nahrazení části textového řetězce

Je také možné nahradit část textového řetězce jiným textem. Nejprve definujeme proměnné s naším textem a řetězcem k vyhledání `search`. Poté ze `search` ořežeme prázdné místo a případná zalomení řádků pomocí `.trimmingCharacters(in: .whitespacesAndNewlines)`.

```
var text = "Filip is 22"
var search = "22 "
search = search.trimmingCharacters(in: .whitespacesAndNewlines)
```

Dále definujeme proměnnou `range`, která nám vrátí rozsah hledaného textu v původním řetězci. Pomocí `.caseInsensitive` zajistíme, že se bude vyhledávat textový řetězec převedený na malá písmena. Poté si pomocí `if let` ověříme, že řetězec obsahuje daný rozsah, a protože jsem zase o rok starší, nahradíme daný rozsah pomocí `replaceSubrange()` textem 23.

```
var range = text.range(of: search, options: .caseInsensitive)
if let rangeToReplace = range {
    text.replaceSubrange(rangeToReplace, with: "23")
}
print(text)  // "Filip is 23"
```

2.5.3 Zjišťování pořadí slov

Framework *Foundation* umožňuje určit abecední pořadí řetězců pomocí metody `compare`. Ta vrací výčtový typ, který rozlišuje, zda první řetězec předchází druhý (`.orderedAscending`), zda následuje po něm (`.orderedDescending`), nebo zda jsou shodné (`.orderedSame`). Parametr argumentu: `.caseInsensitive` zajistíme, že při porovnávání nebude brán zřetel na velikost písmen.

V případě řetězců `"Orange"` a `"Apple"` bude výsledkem hodnota `.orderedDescending`, protože v abecedním pořadí písmeno `O` následuje po písmenu `A`.

Vážení čtenáři, právě jste dočetli ukázku z knihy ***Vývoj aplikací ve SwiftUI*.**
Pokud se Vám ukázka líbila, na našem webu si můžete zakoupit celou knihu.