Obsah

Kapitola 1

O knize

1.1 Úvodem

Vážený čtenáři,

tato kniha by vás měla povzbudit v cestě k osvojení vašeho prvního programovacího jazyka. Autor této knihy nepředpokládá žádnou Vaší předchozí zkušenost s programováním. V případě, že již určitou zkušenost máte, některé kapitoly pro vás mohou být rutinní. Kniha by vás měla postupně provázet vytváření zkušeností postupného ovládání stroje od základních výpočetních operací k tvorbě pokročilejších nástrojů.

Kniha je psána ve sledu, který bych zvolil kdybych se sám měl učit programovat. Programování je bezesporu myšlenkově náročná operace. Mým záměrem je postupně tuto náročnost redukovat a proměnit psaní programu v lehkost.

Cestu k této lehkosti Vám neumím jednoduše předat, pedagogický talent jsem nikdy nepocítil. Tato kniha by proto měla být odložena vždy, bude-li mít čtenář nutkání si něco sám vyzkoušet. To považuji za absolutně nejrychlejší a zároveň nejlepší způsob učení.

Proces tvůrčího programování obsahuje prvek intuice, která vychází podstatnou měrou ze zkušenosti. Pro podporu Vaší intuice při psaní programu je nutné ovládnout základní jazyk natolik, aby jste byli schopni pocítit určitou jistotu. Přestanete-li se již zabývat funkčností programu a získáte-li v ní jistotu, přijmete jazyk za vlastní nástroj, a tím je možné nechat hovořit právě vaší intuici. Intuici sám považuji za avantgardu logiky, protože je před ní vždy minimálně o krok napřed.

K vystavění intuice je nejvíce nezbytná schopnost pozorování. Samotná intuice nezmůže v komunikaci se strojem nic. Stroj neví co počítá, ale dokáže spočítat i to, co nevíte sami.

Stroj je rychlý nástroj pro ověření vaší intuice a je zapotřebí být pozorný. Můžeme to nazvat experimentální přístup, výsledky by měli být vámi vždy intuitivně a hlavně kriticky zkoumány. Zpětnou dekonstrukcí intuice, rozborem Vaší jistoty získáváte postupně dar rozumění věci. Dar je to danajský, rozumění je jen ustálená podoba, která musí být opět zpochybněna intuicí; nutno podotknout, že opakovaným prověřením se často jen více utvrzuje. Ale o tom později.

...

Není-li tento jazyk první, prosím Vás o shovívavost v podrobnostech, do kterých v úvodu této knihy zabíhám. Šíře znalostí, které se mohou vyskytovat u potenciálních čtenářů tohoto průvodce je pro autora první velikou neznámou.

V obou případech prosím o shovívavost ve způsobu popisu programovacího jazyka a prostředí Processing. Sám jako samouk a člo-

věk zaměřen především výtvarným směrem nemohu zaručit absolutní, stoprocentní a všeobecnou platnost všech tvrzení. Tímto vás tedy žádám o věcné podněty pro pozdější doplnění nebo přeformulování jednotlivých tvrzení.

Má snaha provést začínající i středně pokročilé uživatele jazykem bude vždy nezbytně nedostatečná, berte ji prosím spíše za průvodce nesnadnými začátky. Čtete-li knihu z jiných důvodů, než z důvodu učení se programovacímu jazyku, pokusím se text knihy proložit poznatky nabytými moji několikaletou zkušeností sdílení života se stroji. Dostala-li se vám tato kniha do rukou jinou cestou nebo dokonce náhodou nebo omylem, tedy věru netuším, co v ni dále najdete, a proto bych i vás rád pobídl alespoň k začtení se do světa skriptů a kódů.

1.2 Forma knihy

1.2.1 Uspořádání informací

Text je řazený do jednotlivých kapitol a dále stromově do dvou úrovní podkapitol. Pořadí kapitol by mělo odpovídat sledu informací nezbytných k plynulému učení se programovacímu jazyku Processing.

Pořadí a obsah jednotlivých kapitol vychází z mé vlastní zkušenosti. Z veliké míry koresponduje s podobnými průvodci. Původce píší samotní tvůrci a širší komunita uživatelů kolem programovacího jazyka Processing.

Zvolená forma textu odpovídá ověřeným postupům. Následující text bude podléhat určitým zákonitostem. V knize se objeví několik typů textu, které budou vždy odlišeny vlastní formou zápisu.

1.2.2 Pravidla formátování v knize

Pravidla jsou následující:

Obyčejný text: popis ve formě klasického textu

kód:
```
/**
* Barevný text v šedém rámování bude znait samotný
* strojový kód, žvdy ve ěform, která je žji
* srozumitelná Processingu.
*/

boolean pravda = true;
```

Poznámky na okraji pro shrnutí nebo zdůraznění informace budou v šedé bublině

L + T: klávesové zkratky, psané pomocí „malých kapitálek“

ovník: výrazy a pojmy, jejichž definici můžete nalézt ve slovníku na konci knihy

olean: jednotlivé příkazy a oficiální příkazy Processingu, které jsou zároveň zařazeny do slovníku

- Shrnutí nebo zvýraznění důležitých informací v textu se nachází na postranní bublině v šedé barvě.

Jakou barvu má bublina značící otázku?

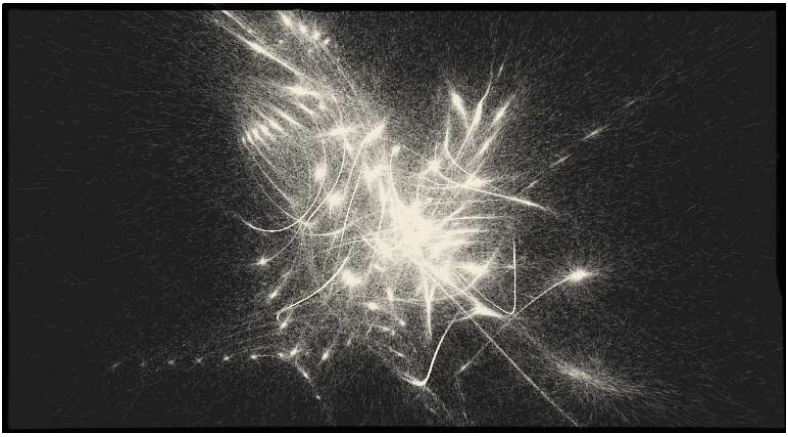
- Kontrolní otázky v textu se nachází v bublině modré barvy.

Kapitola 2

Postaveno Processingem

[tři příklady zde]
Instalace. Web applet. Video. Fyzický výstup?

2.1 idea space - a cyclic universe



- Autor: Karsten Schmidt
- Rok: 2004
- Médium: generovaná grafika - černobílý tisk
- Anotace:

  Stills of an ongoing visualization project of a space with a steadily increasing number of moving particles attracted by slowly moving, invisible gravitational centres. the cyclic nature of the space itself acts as four dimensional history, causing each particle to leave a persistent trace in time as well as in space. the paradoxical result of this setup is that whereas the number of particles is approaching infinity there's no increase in computational cost.

  As the particles move through space they become attracted by the various, initially randomly positioned gravitational centres. the force of attraction follows the classic "inverse square law"in physics, meaning a particle is a lot more influenced and accelerated by a close attractor than by ones further away. the more particles are in a highly active gravitational reqion of the space, the more clearly lines start to appear, showing the trajectory of these particles through space as well as time towards the locally strongest gravitational center.

Kapitola 3

Cíle knihy

3.1 Konktrétní zadání

Cílem této knihy je provést uživatele začátečníka sérií příkladů, které budou v průběhu ilustrovat znalosti nebytné k naprogramování postupných stádií programu. Jako první cíl si vytyčíme jednoduchý interaktivní program. Interaktivní program tedy takový, který dokáže určitým způsobem reagovat na uživatele. Tento program by ve výsledku měl přečíst polohu myši a na jeho základě provést proměnu v obraze.

Na úvod je třeba říci, že vytyčení konkrétního cíle není vždy nezbytné. Processing již počítá s možností, že uživatel nebude od počátku znát konkrétní zadání a bude jen volně experimentovat s funkcemi jazyka. Proto Processing automaticky pojmenovává nové projekty, v žargonu Processingu Sketche, podle data svého vytvoření.

Kapitola 4

Co je to programovací jazyk?

4.1 Počátky programovacího jazyka

Výchozím bodem v tvorbě počítačové logiky byla přirozená lidská komunikace v podobě řeči. Člověk používá jazyk a při tvorbě zcela nového systému pravidel zákonitě sahá po známém prostředku.

Na počátku ale nebyl lidský jazyk. Se stroji se v jejich raných fázích vývoje hovořilo čistě strojovým jazykem, už v podobě číselných řad, nebo později holého strojového kódu, který přešel na logiku pokročilejších obvodů. Jazyk se objevil již v padesátých letech minulého století a osvědčil se jako prostředek pro člověka, který začínal být limitován oproti kapacitě stroje schopného obsáhnout složitější úlohy. Řešením pro vzrůstající složitost logiky programů se stala lidská řeč.

Řeč se v počátcích tvorby struktur pro program obsah výpočtu jevila jako člověku nejpřirozenější řešení. Velmi brzy tvůrci zjistili,

že za pomocí jazyka lze sice definovat složitý problém pro člověka, vyvstává tím ovšem první problém definice významu takového jazyka stroji.

Počítačový jazyk není zdaleka jeden, počítačových jazyků jsou celé rozvětvené rodiny. Konceptů jak lépe hovořit srozumitelně pro člověka a zároveň ke stroji je celá řada, žádný z nich nedokáže kopírovat plně lidský jazyk v jeho přirozené struktuře. Důvodů pro to je několik, jeden z nejpodstatnějších je fakt, že lidská řeč neodpovídá organizaci ve struktuře stroje, jednoduše proto že člověk své řeči sám nerozumí natolik, aby popsal sám všechny její zákonitosti logickou cestou.

Jednoduše řečeno ani lidský jazyk není popsán tak dokonale aby jsme ho byli schopni vysvětlit stroji. Pokusy o formální naplnění lidské řeči sahají do absolutních počátků interakce člověka se strojem a představují pro tvůrce strojů závažný problém. Tento problém v padesátých letech minulého století definoval Alan Turning, jeden ze zakladatelů výpočetní techniky tak jak ji dnes známe. Sestavil pro stroje experiment, takzvaný turningův test, jenž měl prověřit schopnosti stroje replikovat lidské chování, není náhodou že prostředek pro komunikaci v testu byla zvolena právě řeč. Test spočíval v modelové situaci ...

K tomu, abychom sdělili informaci, používáme jazyk. Jazyk musíme umět přizpůsobit tomu, aby informoval, sdělil jistou skutečnost - myšlenku sdělovanému subjektu. Jazyk má nutně několik úrovní, zdaleka ne všechny jsme schopni reflektovat. Úrovně, které jsme schopni reflektovat, jen částečně umíme logicky popsat. Logicky popsaným jazykem a vnitřně uceleným systémem jsme schopni vysvětlit pouhý fragment skutečnosti.

4.2 Programovací jazyk

V případě programovacího jazyka si je nutné uvědomit, že programujeme (informujeme) stroj. Dnešní stroje popisují jen zanedbatelně velikou skutečnost fyzicky nepřesahující dimenze součástí stroje[1], ale tím zpětně i sebe sama a druhé lidi[2], symbolicky a nepřímo popisujeme zkratku skutečnosti nepoměřitelně daleko větší.

4.3 Jednoduchý programovací jazyk

Snaha po zpřístupnění programování širší veřejnosti dala již na konci dvacátého století vzniknout rodině jazyků, které jsou patřičně zjednodušeny tak, aby je mohli obsluhovat i neodborníci.

Výchozím bodem pro zjednodušení programování je odpověď na situaci, kdy programovací jazyky vytvářeli především lidé se zvláštním nadáním pro ryze technické uvažování. Technické uvažování je zvláštní nadání, pro které nemá každý člověk správné predispozice. Programování dnes znamená především určitý stupeň svobody při komunikaci se strojem.

K míře svobody, která má své silné kritiky[3], se nyní nechci vyjadřovat, ale zjednodušeně z pohledu pouhého uživatele který používá daný nástroj, schopnost programovat činí z uživatele již potenciálního strůjce vlastních nástrojů.

Hovořím-li o stroji, mám dnes na mysli spotřební počítač. Termín **stroj** používám záměrně pro zdůraznění jisté formy strojového přemýšlení v historickém kontextu.

[1] nejvíce pak procesoru paměti, skutečnost na pevném disku a grafických procesorů

[2] potažmo jakkoli širší skutečnost (zvířata, rostliny, cokoli o čem ani dále netušíme)

[3] včetně mne samotného

Nástroje jsou pak programy zkonstruované pro jistou činnost. Obvykle je nástroj vyvíjen za jedním účelem, který plní uživatelsky co nejpřívětivější cestou. Tato cesta je pro uživatele snadno schůdná a nabízí mu standartní škálu dovedností nástroje.

Aniž bychom si to často uvědomovali, současná vizuální kultura je ovlivněna těmito nástroji daleko více, než je na první pohled zřejmé. Technické možnosti jsou současným tržním hladem pro inovaci konfiskovány a proměňovány ve zboží. V této situaci je důležité znát nástroje i jejich vznik pro reflexi nebo kritiku v širších souvislostech.

Tato kniha je spíše než-li jednomu nástroji věnována programu pro tvorbu takových nástrojů. Jak již vyplývá z této definice použití Processingu není limitováno jen úhlem pohledu autora tohoto textu. Návod by se měl stát spíše pobídkou k co nejrozmanitější tvorbě vlastních nástrojů, sloužících opět k co možná nejširší škále možných účelů.

Processing vychází z koncepce snadného přístupu k programování. Za běžných okolností by se vnímavý člověk měl být schopen naučit jednoduché struktuře programu a schopnosti vytvořit vlastní program v průběhu několika dní. Na druhou stranu, právě predispozice našeho uvažování jsou natolik rozmanité, že takovou prognózu nelze brát jinak než-li jen za orientační.

Programovací jazyk Processing vychází z laboratoří MIT, kde se celé oddělení věnuje právě koncepcím zjednodušování programovacích jazyků .... historie small talk atd. zde

4.4 Dokonalost jazyka

Co je to dokonalý jazyk? Nejprve je zapotřebí říci, že absolutně dokonalý jazyk neexistuje. Jazyk si můžeme představit jako sys-

tém vzájemných vztahů, který je schopen popsat jednotlivé symboly nebo objekty. Symboly můžeme nazvat předměty, tyto předměty dále mají své vlastní hodnoty a vlastnosti, jazyk kromě definic takových vlastností operuje a popisuje jednotlivé jevy a vztahy mezi těmito předměty. Jednodušší popis jazyka je v pojetí výpočetní techniky určitý ucelený systém schopný popsat rozmanité problémy řešitelné strojem.

Co činí předem jakýkoli jazyk absolutně nedokonalým je nejprve fakt, že jakýmkoli jazykem nedokážeme vyjádřit původ jazyka, tj. jeho strůjce; člověka. Jazyk použitý pro instruktáž stroje je vnitřně konzistentní a funguje logicky hermeticky, tj. nepřipouští jiný než jeden výklad konkrétního textu. Pojetí dokonalosti jazyka ve smyslu vnitřní logické konzistence je naprostou nezbytností v pojetí interpretace strojem, na druhou stranu téměř nepřekonatelnou překážkou v případě abstraktnějších úvah o programování jako takovém.

Použiji zde pro názornost rozdíl programovacího jazyka s jazykem českým. Český jazyk, stejně tak jako jakýkoli mluvený nebo psaný jazyk, je jazykem organickým ustáleným po staletí užívání. Jazyk jak ho známe slouží ke komunikaci mezi lidmi, lze tedy použít například pro popis krajiny. Přestože k dokonalému popisu krajiny stěží kdy můžeme dojít, těmito slovy které se opírají o určitou sdílenou zkušenost, lze poměrně dobře přiblížit určitý obraz věcí.

Pokusili bychom se pro popis krajiny použít jazyk programovací, dostaneme se velmi rychle do nesnází. Programovací jazyk není jazykem určeným primárně pro předání informací mezi lidmi, ale pro komunikaci člověka se strojem. Jeho vnitřní logická konzistence, tvrdá logická struktura, která nedovoluje v jeden okamžik jinou než jednu interpretaci, je jeho velikou předností při definici exaktních parametrů. Podobnost s řečí spočívá ve vazbě slov, které reprezentují jednotlivé hodnoty a operace. Hlavní odlišnost je v jeho

syntetickém původu, jedná se o jazyk umělý a účelu ke kterému byl zkonstruován. Programovací jazyk je přednostně zkonstruovaný pro definici známého a pochopeného. V případě neznámých nebo nepoznaných veličin, je programovací jazyk víceméně k ničemu.

Chceme-li komunikovat se strojem musíme tedy svůj způsob vyjadřování přizpůsobit logicky dokonalému jazyku - vnitřní logice fungování stroje. Počítač není navržen k tomu aby něčemu rozuměl, počítač je navržen k řešení jasně definovaných otázek. Tato kniha se pokusí srozumitelnou formou popsat jeden z možných způsobů jak si osvojit takový jazyk a potažmo způsob uvažování, který vede k jasné definici problému. Hovoříme-li o programování, máme na mysli proces tvorby jisté logické struktury. Osvojení si programování spočívá ve schopnosti definovat problém nebo jasně formulovat otázku, tak aby stroj na ni mohl odpovědět.

Vtip celé věci spočívá v tom, že ovládneme-li formálně jazyk určený stroji, můžeme prostřednictvím tohoto stroje hovořit i ke člověku, tj. popisovat pocity z rozkvetlých luk.

4.5 Volba vhodného jazyka

Ve výpočetní technice se nachází celá škála programovacích jazyků i prostředí. Tyto jazyky mají svoji genezi a byli historicky vyvíjeny především počítačovými odborníky. Jejich dokonalost lze těžko ocenit z vnějšího pohledu, a to právě z důvodu jejich konstrukce, která odpovídá a částečně podléhá určitým účelům, ke kterým byli tyto jazyky původně navrženy. Celistvý pohled na vývoj programovacích jazyků zde není možné obsáhnout. Základní rozdělení programovacích jazyků dle historického vývoje můžeme dnes označit na dvě skupiny, jazyky procedurální a objektově orientované.

Processing se svojí stavbou na základech Javy řadí k objektově

orientovaným jazykům.[4] Toto rozdělení pojednává o koncepci jazyka a jeho základních struktur.

Rozdíl mezi procedurálním pojetím a objektově orientovaným jazykem spočívá především v logice kódu a jeho následném uspořádáním. Procedurální jazyk se dá považovat za předchůdce objektově orientovaného pojetí. Ačkoli se nedá obecně tvrdit, který přístup je lepší, jedná se o dvě rozdílná paradigmata a dnes masivně převládá objektově orientované paradigma.

Rozdílné pohledy lze ilustrovat na popisu nějakého jevu. Jev se dá popsat různými způsoby. Jedna perspektiva bude více hovořit o aktérech jevu, ty rozdělí do objektů a jejich vlastností a možností. Druhá perspektiva popíše celistvou situaci jako sérii jevů které je možné kdykoli zopakovat.

V posledních přibližně dvaceti letech se mezi programovacími jazyky postupně objevuje tendence po větší srozumitelnosti a potažmo zjednodušení programování jako takového. Programování v této koncepci zjednodušování již není jen jazykem odborníků, ale je demokraticky přístupný širší veřejnosti z rozmanitých - prioritně netechnických oborů.

Tato tendence postupně dala vzniknout celé rodině programovacích jazyků [5] které mají ve snaze přiblížit potenciál výpočetní techniky blíže k jiným, netechnickým oborům. V neposlední řadě i oborům výtvarným. V technických kruzích je ovšem již sama disciplína psaní programů považována za tvůrčí činnost. V pojetí výtvarného umění dnes převažuje pohled na programování jako na velmi technickou zdatnost, rigidní a notně limitovanou činnost.

Jazyk, který je nutně limitovaný svojí nezbytnou dokonalostí

[4] Jeho společnými příbuznými jsou kromě *Javy* jazyky jako *C++*, *C#*, nebo *VisualBasic*

[5] vyjmenovat jazyky zde + historie?

ovšem nemusí nutně limitovat svého uživatele ve sdělení. Uživatel se ovšem musí do určité míry, osvojením si základních struktur, přizpůsobit stroji k uskutečnění takové zdárné komunikace.

4.6 Proč zrovna Processing?

4.7 Kritika Processingu

[Důvody proti Processingu]

4.8 Tvorba softwaru

Programy jsme dnes zvyklí spíše konzumovat a využívat než opravdu sami vytvářet. Vytváření programu je náročný proces a tvorba uživatelsky přátelského prostředí pro tvorbu programu je proces opravdu složitý.

Processing se svým způsobem neliší od žádného jiného programu který běžně využíváme. Jedná se o sadu příkazů a samotné programovací prostředí, které nám dovoluje určitou formou vytvářet svébytný program. I když se jedná již o programování nelze jej běžně zaměňovat s klasickou tvorbou dospělého nástroje.

Zde si musíme uvědomit, že náš potencionální produkt - program bude vždy spíše banální v porovnání se samotným prostředím Processingu. Procesingový kód je výčet všech možností, které můžeme při tvorbě našeho programu využít. Obecně se dá říci, že Processing je nástroj pro tvorbu speciálních nástrojů. Výsledek našeho programování bude vždy obsahovat poměrně velkou část Processingu samotného.

Je dobré okamžitě od začátku pochopit k čemu Processing lze

využít a k čemu jej opravdu chceme využívat. Processing se především hodí k rychlé tvorbě programu, ověření teze nebo spontánního nápadu. Tvorba dospělejších nástrojů není sice nemožná, obecně ale platí, že Processing bude svými zkrácenými zápisy a jednoduchostí prostředí velmi nápomocný v začátcích. Může se ovšem stát více nevhodným v případě pokročilejších projektů.

4.8.1 Otevřenost softwaru

Jazyk nazvaný Processing je jedním z jazyků, který byl vytvořen v diskurzu zjednodušování programování. Jako každý jiný programovací jazyk je i Processing navržen pro jisté účely. V případě tohoto programovacího jazyka se nejvíce jedná o důraz na rychlý vývoj a zjednodušené nakládání s obrazem i prostorem. Z více technického pohledu pak Processing vyniká otevřeností zdrojového kódu a důraz na multiplatformnost.

Z pohledu vývojáře je velmi důležité, že jazyk i programovací prostředí Processing je v současnosti **otevřený software**, což znamená, že prostředí i samotný zdrojový kód je volně k dispozici. Processing je dále šiřitelný pod **MIT licencí**. Pro vývojáře otevřenost zdrojového kódu znamená zásadní věc, jednoduše pro dosažitelnost celého zdroje, který se dá následně například implementovat do různých prostředí. Další možnost vývojáře je rozšířit jazyk o vlastní knihovnu, a tím participovat na projektu. Otevřenost kódu teoreticky navyšuje počet možných participantů a de facto celý projekt udržuje v dlouhodobém horizontu naživu.

Z pohledu samotného uživatele je velmi příjemné, že samotný software je k dostání zdarma na stránkách projektu. Za jeho užívání není nutné platit žádné poplatky, a to i v případě komerčních užití. V případě potřeby vyjádření vděku za práci autorů je možné zmínit kdekoli ve Vašem produktu **Built with Processing**.

Processing na první pohled není ničím zvláštní programovací jazyk. V podstatě by se dalo říci, že se jedná pouze o rozsáhlou knihovnu pro jazyk Java. To co Processing řadí mezi oblíbené softwary pro tvorbu dnes je nejvíce přívětivá komunita uživatelů s velmi odlišnými stupni znalostí a úhly pohledu. Zvláštní důraz je v komunitě kladen na poskytnutí co největší podpory právě začínajícím uživatelům. Tomu odpovídá i počet rozmanitých průvodců a rozsáhlá, velmi dobře stručně napsaná dokumentace ke každému z příkazů v Processingu.

K otevřenosti v kódu v neposlední řadě přistupuje i celá řada velmi zkušených tvůrců. Tím se uživatel na jakémkoli stupni znalostí může kdykoli naučit nové postupy nebo může svobodně recyklovat algoritmy druhých uživatelů. Processing a čím dál více jeho uživatelů ctí filosofii otevřeného softwaru která (mimo jiné) hlásá: „Vědomosti nesmí být privatizovány!“

4.8.2 Procesualíta, živý program

Hlavní doménou Processingu je schopnost vytvářet „živé“ programy, tj. programy běžící v reálném čase. Již jméno programovacího jazyka Processing napovídá akcent v čase se odvíjejících událostí.

Obraz vytvořený tímto způsobem, na první pohled zaměnitelný s videem, nemusí například podléhat časové omezenosti, nebo může určitým způsobem reagovat na své okolí. Generovaný obraz může být například takto vtažen do kontextu, ve kterém se nachází. Možností jak nakládat s těmito specifickými možnostmi je nepřeberné množství.

[rozvést více]

4.8.3 Empirický přístup k programování

I když zásadní odlišnost s jinými programovacími jazyky bychom hledali stěží, Processing proslul zejména pro snadnost použití. Kompilovat program není otázkou nastavování kompileru a veškerých jeho parametrů, program jednoduše po stisku tlačítka *RUN* běží (je-li správně napsán). Samozřejmě tento redukcionistický přístup má své nevýhody, speciálně při rozsáhlejších projektech tato jednoduchost může dokonce omezovat. Processing ovšem jako svobodný software lze naimplementovat do řady jiných prostředí, a potřebujete-li si kompilační proces nastavit sami, nic Vám například nebrání použít Processing jen jako knihovnu do *Javy*.

Tato základní jednoduchost na druhou stranu nezdržuje uživatele od myšlenkového toku psaní programu. Častou kontrolou výsledku kódu uživatel může lépe interagovat se samotným tvarem programu.

Nazývám tento způsob programování empirický, tedy přístup, kdy podle zkušenosti s běžícím programem je dotvářen i samotný zdrojový kód. Mnoho technicky zaměřených lidí by zřejmě mohlo tento postup kritizovat pro přílišnou reduktivnost a amatérský přístup k programování. Zde bych oponoval faktem, že motivace lidí vytvářející například instalaci do galerie nezajímá příliš dokonalost (když už tedy vůbec) programu. Program je v pojetí tvůrců jen prostředníkem pro další sdělení, a jestliže toto sdělení předá je to dobrý program.

Proces poznávání struktur jazyka při tvorbě obrazu prostřednictvím Processingu bych přirovnal k postupu od začátečnické malby na tkaninu k postupnému tkání gobelínu. Zde je nutno podotknout, že ne každý tvůrce využívající Processing chce tkát gobelín a najde-li nástroj vhodný „jen“ pro „malbu“ na plátno, je to dobrý nástroj.

[????]

Kapitola 5

Processing jako prostředí

Processing představuje ucelené programovací prostředí tzv. IDE[1]. Jedná se o kompletní prostředí určené především k rychlému vývoji aplikací. Samotný program je jednak otevřeným softwarem[2], a také zdarma ke stažení pro všechny majoritní platformy. Processing je k dispozici pro **GNU / Linux**, Mac OS i pro Microsoft Windows na stránkách projektu processing.org. Porcessing je teoreticky možné spustit v jakémkoli prostředí umožňujícím chod Javy. Celý jazyk i samotné IDE vychází pod licencí **GNU / GPL** jedná se tedy o svobodný software. Díky této licenci můžete jakýkoli výstup z tohoto softwaru můžete publikovat, použit pro jakékoli účely včetně například užití v komerčních aplikacích.

[1] Integrated Development Enviroment

[2] tj. s otevřeným veřejně dostupným zdrojovým kódem

5.0.4 Sketch

Sketch, neboli náčrt, označuje samostatný projekt Processingu. Dělením projektů na sketche jsou uspořádány jednotlivé projekty Processingu. Sketch je ve své podstatě složka obsahující alespoň stejně jeden stejně nazvaný soubor s příponou **.pde*. Sktech při vytvoření nevyžaduje název, je jí přidělen pouze aktuální datový kód zaznamenávající datum vytvoření. Tímto decentním způsobem Processing toleruje například nejasnost záměru autora při vytváření nového díla, jeho název či pracovní název lze tímto způsobem přiřadit až později, po nabytí jasnějších obrysů.

Program si vystačí pouze se samotným zdrojovým kódem tj. souborem (soubory) **.pde*. Ovšem v případě operujeme-li s externími daty stojícími mimo zdrojový kód, jakékoli jiné soubory, můžeme využít dvě možnosti. První možnost je soubor, se kterým potřebujeme operovat tažením myši přesunout na textové pole Processing IDE. Touto operací bude soubor zapsán do naší sketche automaticky. Druhá možnost je operaci provést manuálně, stačí vytvořit v adresáři sketche adresář s názvem **DATA**. Processing tuto složku automaticky rozpozná a soubory v této složce budou dobře dostupné pro pozdější operace.

Ke standartním adresářům, které se často objevují ve struktuře skteche, si stačí zapamatovat jen dva již zmíněný adresář nazvaný *DATA*. Tento adresář je využíván při práci s jakýmikoliv vnějšími daty jako jsou např. obrázky, zvuky, videa, vektorové grafiky nebo například textové soubory. Druhý adresář nazvaný *CODE* obsahuje externí zdrojový kód, popřípadě kód kompilovaný v podobě knihovny, mající nejčastěji přípony **.java*, **.class*, **.jre*. Tento adresář se nalézá v systémové cestě daného projektu a umístíme-li zde soubory budou též dobře dostupné z daného projektu. Adresář *CODE* nás zatím nemusí příliš zajímat, později se jeho prostřednictvím můžeme

pokusit rozšiřovat funkce Processingu.

5.0.5 Sketchbook a uspořádání

Místo na disku, které Processing využívá k uskladnění sktechí se nazývá povšechně sketchbook a je v podstatě pouze adresář obsahující jednotlivé projekty. Koncepce sketchbooku spočívá v uspořádání jednotlivých projektů do organizované formy a ačkoli nevnáší do jednotlivých projektů sám o sobě řád může napomoci k vytvoření řádu vlastního. Ze své zkušenosti mohu říci, že organizace jednotlivých projektů do jakékoli struktury má opravdu smysl. Mnou preferovaná struktura obsahuje jednotlivé roky následně ještě dělené do měsíců, ale možnosti organizace záleží čistě na uživatelových preferencích. Tímto chci spíše ilustrovat možnosti uspořádání než-li nezbytnost, ale přítomnost vašeho systému od počátku vřele doporučuji.

Ve sketchbooku se dále nachází jeden speciální adresář nazvaný *libraries*. Adresář v sobě uchovává externí rozšíření Processingu o komunitní knihovny. Standartně Processing již své základní knihovny nese v sobě, tj. jsou standartně přidány do samotného Processingu. V tomto adresáři se nacházejí knihovny které budete moci použít později. Na stránkách projektu *http://processing.org* můžete nalézt velké množství komunitních knihoven s dobrou dokumentací, volně ke stažení. Veškeré tyto knihovny tedy Processing potřebuje nalézt v tomto adresáři.

5.1 Základní prostředí

Základní rozhraní tvoří textový editor s několika nezbytnými funkcemi. Patrně nejdůležitější je v liště tlačítko (1) tzv. RUN. Které

kompiluje a spouští program aktuálně rozpracovaný v textovém editoru. (2) Tlačítko STOP naopak program zastaví. V případě chyby v programu lze též využít jako vynucené zavření programu. Zbylé tlačítka slouží k diskovým operacím.

5.1.1 Základní diskové operace

Ty operace jež něco zapisují či načítají z disku. Tyto opera zahrnují veškerou manipulaci se sketchí, její vytvoření, uložení, načtení či export do webového formátu či kompilaci do spustitelné aplikace.

Processingová sketch nemusí být nutně uložena aby jste ji mohli spustit. Nové úpravy, které v textovém poli napíšete budou dočasně uloženy v závislosti na vašem operačním systému jinde. Tímto způsobem můžete experimentovat s kódem aniž by jste přišli o předchozí verzi.

Mezi základní operace patří vytvoření nové sketche (3), otevření předešlé (4), uložení (5) a export (6).

Tlačítka mají dále speciální vlastnosti s podržením tlačítka tlačítko s novou sketchí otevře také nové okno editoru. V kombinaci s načtením předchozí sketche (4) otevřete též sketch v novém editoru aniž by jste ztratili předchozí okno. U uložení se modifikace projevuje funkcí uložit jako. U exportu stisknutím klávesy *SHIFT* přepínáte mezi exportem samostatné aplikace nebo tzv. appletu, aplikace schopné běžet v prohlížeči nebo obecně na webových stránkách.

5.1.2 Editor

Editor je textové pole a je vaším hlavním komunikačním nástrojem s Processingem. Veškeré informace které zadáte do tohoto pole budou interpretovány samotným Processingem. Textový editor není nijak dokonalý, pro začátek si s ním ovšem vystačíme. Formát který tento editor produkuje je ve své podstatě textový soubor s příponou **.pde*.

Textový editor má několik funkcí, které vám mohou usnadnit práci. Odlišuje například mezi příkazy Processingu odlišnými barvami, tato zdánlivá drobnost speciálně v začátcích napomůže nebo v případě objemnějších kódů může výrazně zpřehlednit organizaci pro-

gramu. Jedná se o programátorskou konvenci obecně v anglickém jazyce nazývanou **syntax highlighting**.

Další konvencí se kterou se často setkáte je tzv. **indenting**, zarovnávání kódu do úhledných paragrafů. V Processingu si ze začátku vystačíte se standartím zarovnáním, posléze lze zarovnávání aktivovat stiskem kombinace kláves *CONTROL (APPLE) + T*.

Více o konvencích se dočtete na straně **??**

Horní lišta editoru označuje záložky, kliknutím na šipku v na pravé straně lišty se vám zobrazí možnosti operací se záložkami. Pro začátek s nimi nebudeme operovat. V případě delšího kódu záložky slouží k lepší organizaci struktury programu. V podstatě se dá říci, že každá záložka odpovídá jednomu souboru ve sketchi. Co je to sketch popíšu níže.

Jak můžete vidět na obrázku editor operuje i českými znaky, ačkoli bych jejich používání z důvodů internacionalizace neradil používat. V komentářích tj. textem uvozeným dvěma dopřednými lomítky //, tedy text, který kompiler ignoruje, jsou diakritická znaménka přípustná.

Naopak v případě názvu proměnných, nebo jakýchkoli funkčních definic, je použití diakritiky v devadesáti procesntech nefunkční.[3]

[3] standartní jazykové kódování, ve kterém Processing operuje je *UTF-8*

Kapitola 6

Processing jako jazyk

6.1 Soustředěná činnost

Než začneme doopravdy programovat musíme si uvědomit, že tento proces vyžaduje velikou dávku koncentrace. Obecně se nedá říci jaké fyzické prostředí je k programování ideální. To se samozřejmě může lišit. Obecně platí, že fyzický prostor, který k programování potřebujete je prostor, ve kterém se můžete sami soustředit na práci.

Koncentrace při psaní programu je specifická dovednost, kterou se začátečník učí jen stěží. Bohužel pouhým čtením této knihy se již vyvádíte z potencionální koncentrace. Učení se programovat je náročný proces a vyžaduje velikou dávku koncentrace sám o sobě. Okamžité vyzkoušení nových poznatků pomáhá zápisu informací do vaší dlouhodobé paměti a opakování je samozřejmě nezbytné k utužení těchto struktur. Co je bohužel při počátečním programování nezbytné je například hledání v referencích a spolu se soustředěním na program samotný je celý proces učení se na koncentraci velmi náročná věc.

V problematice programování obecně platí, že není nic složitějšího, než číst cizí kód, nebo dokonce kód vlastní s odstupem času. Při programování se lidská mysl dostává do stavu mimořádné bdělosti a koncentrace. Tento stav je často popisován programátory jako tzv. **flow**. Jendá se o stav ve kterém mysl vědomě udržuje celý program[1] v paměti.

Pokaždé když si programátor takový stav přivodí, vše v kódu[2] se pro něj jeví srozumitelné. Formátování a standarty velmi napomáhají k rychlému navození bdělého stavu, protože sami o sobě jsou jistou formou jazyka.

6.2 Základní pravidla a zvyklosti

V programování obecně platí jisté zákonitosti. Není tomu jinak i u pravidel, které nejsou důležité pro čtení kódu strojem ale pro člověka. Speciální zákonitosti formátování kódu mají ryze praktickou funkci. Jedná se o standart dodržovaný programátory, tak aby mezi sebou byli schopni sdílet své úsilí.

6.3 Syntax

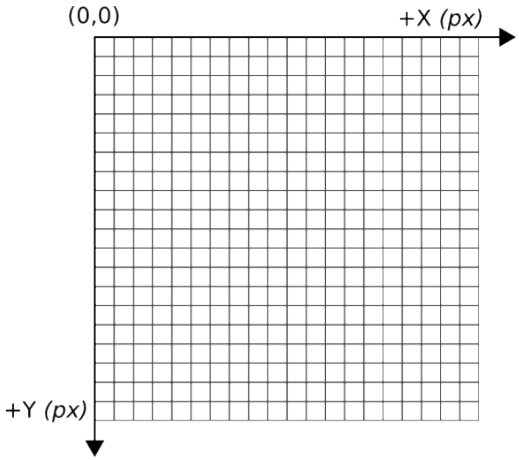
6.4 Orientace v prostoru

Plocha programu by sa dala přirovnat k listu papíru, popřípadě k malířskému plátnu. Ve standartním dvoudimenzionálním módu má dva základní parametry *X* a *Y*, ty označují souřadnice, ve kterých se veškeré kreslící operace pohybují. Důležité je zmínit, že oproti jisté

[1] nebo větší části programu

[2] podle míry ovládnutí programovacího jazyka

konvenci v matematických grafech levý horní roh nese hodnotu $X = 0$, $Y = 0$. Směrem dolů hodnota Y přibývá stejně tak jaho hodnota X přibývá směrem doprava.



Tato konvence je převzata ze standartu počítačové grafiky, kdy první pixel v levém horním rohu nese souřadnicovou hodnotu právě $X = 0$, $Y = 0$. Obrácená osa Y se může ze začátku jevit matoucí. Důvody pro zdánlivé převrácení os pochopíme později například právě při operacích se samotnými obrazovými body, pixely, které jsou standartně uspořádány od levého horního rohu doprava a níže.

Plochu je možné představit si jako prázdný prostor, na kterém je možno zobrazovat grafiku. Tvary nebo například text se zobrazují právě prostřednictvím zdrojového kódu tj. instrukcemi psanými v editoru.

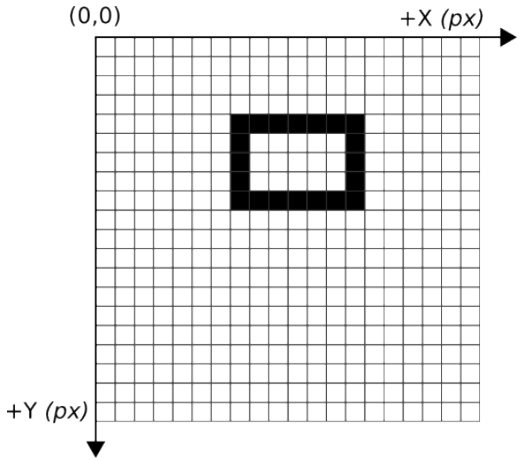
Na první pohled by se mohlo zdát, že například při zobrazení obdélníku, příkaz:

```
rect(x,yš,říka,švýka);
```

jehož výsledkem je kresba pouhého obdélníku je zbytečně komplikovaný, oproti jiným zobrazovacím metodám. Důležité je si zde uvědomit koncepci Processingu, který tímto zobrazením primitivních objektů sestavuje celý obraz. To co se zpočátku může zdát jako nadbytečná práce, tj. psaním koordinátů každého ze zobrazovaných objektů se posléze ukáže jako sofistikovaný a velmi ulehčující způsob přemýšlení o obraze. Veškeré parametry oddělené čárkou v kulatých závorkách příkazu *rect* náleží samotným vstupním hodnotám. Příkaz *rect* očekává čtyři parametry. Parametry mohou být celá čísla nebo čísla s desetinnou čárkou. Pro názornost, zadáme-li do parametrů ”natvrdo” hodnoty:

```
rect(8,6,7,5);
```

Výsledné zobrazení na naší pomyslné ploše bude následující:



Zamyslíme-li se například o tom, že bychom v jakémkoli jiném nástroji chtěli zobrazit vícero, například tisíc obdélníků, na první

pohled jednodušší GUI [3] grafické editory nám nedovolí tuto operaci učinit jinak, než že musíme všech tisíc obdélníků nakreslit. V případě Processingu nám bude stačit vytvořit rutinu pro kreslení libovolného počtu obdélníků a pak tuto rutinu spustit.

Zde se již dostáváme k samotnému jádru Processingového přemýšlení. Programování obecně dokáže velmi ulehčit operace jejichž pravidelnost dokážeme popsat. Veškeré umění psaní programu tedy spočívá v definicích těchto chování a redukci složitých jevů na jednoduché rovnice.

Celé řemeslné umění psaní kódu v podstatě záleží na eleganci v zápisu složitějších vztahů mezi různými parametry. Dovednosti se člověk učí postupně, osvojení si gramatické korektnosti a logické posloupnosti se mohou zpočátku jevit zbytečně komplikované, ovšem i po ovládnutí pouhých pár jednoduchých pravidel lze Processing využít kreativním způsobem.

V další části se budeme zabývat stavbou programu. Pod odpudivým názvem se skrývají právě tato jednoduchá pravidla, která by měla tvořit základ dalším experimentům.

[3] zkratka pro ”Guided User Interface”klasické grafické editory jako Gimp nebo Photoshop

6.5 Stavba programu

6.5.1 Logika programování

Pro začátek je dobré si představit program jako sadu instrukcí. Každá instrukce má svůj význam a své místo. Instrukce se píší v programovacím jazyku a jejich interpretace je vždy pro stroj jednoznačná. Stroje podle svého návrhu nedělají nic kromě toho co mají takovým způsobem instruováno.

Veškeré programy které používáte, dokonce i programy, které nevíte že používáte byly někdy naprogramovány lidmi pomocí programovacích jazyků. Programovací jazyk je pouze, plánovací forma zápisu programu. K tomu, aby byl program spuštěn, musí být v případě jazyku Processing převeden do strojového kódu. Strojový kód se liší od toho zdrojového, našeho čitelného plánovacího jazyka, tím že není čitelný pro člověka, je čitelný pro stroj.

Počítač je pouhý stroj. Umí dnes opravdu rychle vykonávat sled banálních operací. Programovací jazyk slouží k vytvoření smyslu ve sledu banálních operací. Procesu převodu ze zdrojového kódu do strojového se nazývá kompilace.

O kompilaci toho nemusíme naštěstí vědět mnoho, vše bylo již tvůrci Processingu a jazyku Java shrnuto pod tlačítko spustit.

6.5.2 Základní datatypy

Ke stavbě programu potřebujeme stavební materiál. Pro základní pochopení fungování programu je nezbytné nejdříve pochopit základní datatypy. Datatyp si lze obecně představit jako obálku na informaci.

Processing rozlišuje mezi jednotlivými datatypy. Informace, které se nacházejí v paměti se musí nacházet pod správným datatypem, tak aby program věděl jak s nimi operovat.

Pro datatyp je možné představit si různé paralely, má oblíbená je podobnost s obálkami nebo nádobami. Různé datatypy si můžeme představit jako tvary nádob. Do různých nádob, lišících se tvarem, se směstná různý obsah. Typy obsahů se dají názorně představit na rozdílu mezi textovou informací a informací číselnou.

Processing potřebuje nejdříve vědět zda nádoba obsahuje text nebo číslo, aby mohl s touto nádobou operovat. Například, nelze provést matematickou operaci mezi dvěma slovy. Sčítat, odčítat, dělit či násobit se dají pouze čísla.

6.5.3 Seznam základních datatypů

Datatypů je jen několik, dají se vcelku snadno zapamatovat:

```
int mojeCeleCislo = 1;

float mojeCisloSDesetinnouCarkou = 1.33;

boolean mojePravdaCiNepravda = true;

String mujSlovniRetezec = "Ahoj svete!";

char mujJednotlivyZnak = 'A';

color mojeBarva = color(255,127,1);
```

Takovému zápisu bude již Processing rozumět. Vepsáním těchto řádků do processingového editoru již dochází k alokaci vašich informací ve správných datatypech v paměti. Vysvětleme si nyní několik nejasností.

První slovo na každém řádku označuje odlišný datatyp (náš tvar nádoby), následuje jméno proměnné (název pro naší nádobu), poté následuje rovnítko které již přiřazuje obsah - hodnotu do proměnné (nádoba je naplněna).

Každý příkaz je ukončen středníkem *;* .

Podivné názvy jako například *mojeCeleCislo* tzv. proměnných, jen ukazují, že název pro svoji proměnnou můžete zvolit téměř podle libosti. Co do výjimek v názvech proměnných. Proměnná nesmí mít v názvu mezeru tedy prázdný znak. Podle zvyklostí by dále proměnná měla začínat malým písmenem a potřebujete-li nutně pro název použít víceslovný název použijte velké písmeno namísto mezery.

Proměnná nesmí být číslo, ale slovní název. Programátorské konvence při pojmenovávání proměnných jsou ryze praktické. Proměnná by měla mít co možná nejkratší a nejvýstižnější název. Toto je spíše dobré doporučení než-li pravidlo. Správným pojmenováním proměnných docílíte větší přehlednosti v kódu kratší délkou si ušetříte zbytečné psaní.

6.5.4 Tisk do konzole

Tento jednoduchý program, zatím jen definoval pár proměnných do správných datových typů zatím nedělá nic zajímavého. Celá alokace probíhá uvnitř programu. Processing dělá většinu svých operací skrytě. Program kreslí nebo jinak interaguje s uživatelem jen je-li o to požádán.

Nejjednodušší výstup z programu je tisk do tzv. konzole. Konzoli jsme si již krátce uvedli v kapitole *Základní prostředí*. V Processingu se konzole nachází pod textovým editorem. Toto černé pole má pouze textový výstup a slouží k odladění programu.

Tiskem do konzole si nyní můžeme například zkontrolovat obsah našich proměnných. To můžeme provést následujícími dvěma způsoby:

```
print(mojeCeleCislo);
```

```
println(mujSlovniRetezec);
```

Oba příkazy tisknou obsah našich proměnných. Jediný rozdíl mezi příkazy je ten, že druhý příkaz končí znakem:

```
"\n"
```

jedná se o tzv. *newline character*, přidáním speciálního charakteru příkaz tiskne pokaždé na nový řádek. Není nutné si pamatovat tento speciální znak, postačí když si zapamatujeme příkaz *println(cokoli);*

Spustíme-li program nyní, v konzoli se nám ukáže výstup z našeho programu:

```
1ahoj ěsvte!
```

Zde je názorně vidět, že tisk pomocí pouhého příkazu *print* nepoložil další příkaz na nový řádek, a tedy vytiskl *1ahoj* dohromady.

Tisk do konzole se hojně používá při ladění programu. Kdykoli se na něco potřebujete kódu zeptat můžete tak učinit tímto prostým příkazem.

V konzoli se tisk projevuje bílou barvou. Další možný obsah konzole jsou chyby. Ty jsou označeny barvou červenou. Processing vám chybou naznačuje, že něco není v pořádku s vaším kódem. V případě takto jednoduchého programu se v drtivé většině případů bude jednat o překlep, či zapomenutý znak.

Programovací jazyk bohužel (naštěstí ?) neodpouští žádné překlepy. Tedy bude stačit jeden chybný znak v programu aby celý program nebyl spustitelný.

Bohužel chybové hlášení se nedá označit za dokonalé a pro začátečníky bude velmi složité dobrat se pomocí chybových hlášení k původu chyby. Na zlepšení chybových hlášek se pracuje již dlouhou dobu a některé základní chyby Processing v angličtině umí dobře popsat. Většinou ale Processing tiskne řetězec svých chyb, které byly

nastartovány chybou vaší a chybová hlášení čítající několik desítek řádků vám toho nakonec o původní chybě moc nesdělí.

V případě chyby se vás Processing bude snažit přesměrovat na řádek, kde chyba pravděpodobně vznikla. V případě takto jednoduchých programů, jaké si zde ukazujeme, chybu identifikuje zcela bezchybně, bohužel tomu tak nebude ve všech případech.

6.5.5 Základní operace s datatypy

Náš program nyní nedělá nic světoborného. Definuje pár proměnných a následně některé z nich tiskne do konzole. Operujeme zde stále s abstraktními hodnotami které se zapisují do paměti programu a ty pak z paměti zpětně získáváme.

Nyní si ukážeme co s již definovanými proměnnými můžeme dělat. Jednotlivé datatypy mají různé přípustné operace. Zkusme si letmo projít možnosti našich proměnných.

- Integer a Float, neboli čísla: *int a float*

  První proměnná *mojeCeleCislo*, které byla přiřazena hodnota *1*, je takzvaný *Integer*, tedy celé číslo. Tento typ může mít hodnotu[4] v rozsahu *-65575 až 65576* [upřesnit] . V rozmezí těchto hodnot můžeme provádět různé matematické operace mezi čísly:

```
int prvniCislo = 1;
int druheCislo = 5;
int tretiCislo;

tretiCislo = prvniCislo + druheCislo;

println(tretiCislo);
```

[4] na 32-bitových strojích

Tímto jsme provedli základní matematickou operaci, sečetli jsme dvě čísla a následně jsme výsledek vytiskli do konzole.

Další možné operace jsou:

```
// aritmetické operace
a + b;
a - b;
a * b;
a / b;

// půpírstky
a += b;
a -= b;

// půpírstek, úbytek o 1
a ++;
a --;

// modulo (řpebytek po ědlení)
a % b;

// a čěkonen logické porovnávání
// ěšvtí, šmení
a < b;
a > b;

//ěšvtí nebo rovná se, šmení nebo rovno
a <= b;
a >= b;

// shoda, neshoda
a == b;
a != b;

// pozor výsledkem porovnávání žji není číslo!
// ale ěďodpov ANO nebo NE, TRUE nebo FALSE
// pravda nebo nepravda, datatyp boolean
```

Celá čísla neboli *int* nebudou mít problém se sčítáním, odčítáním a násobením celých čísel. V případě dělení může nastat logicky problém, výsledek nemusí být celé číslo. Budeme-li chtít výsledek takové operace uchovat v paměti, tj. zapsat pod naši proměnnou, musíme použít datatyp operující s desetinou čárkou, nebo výsledek zaokrouhlit na celé číslo.

například:

```
int a = 10;
int b = 3;
float c;

c = a / b;

println(c);
```

Vytiskne *3.0*. Pozor, to není správný výsledek! Kde se stala chyba? Processing operující s celými čísly, speciálně při dělení předpokládá výsledek opět za celé číslo. Tedy ono zaokrouhlení provádí již sám. Zde si dovolím nesouhlasit s tvůrci, kteří se tímto snaží začátečníky vyvarovat chyb. Stačí si nyní pamatovat, že pro přesné dělení čísel bychom měli dělit vždy číslem s desetinnou čárkou, tj. s datatypem *float*.

Tedy správně by dělení mělo proběhnout takto:

```
float a = 10;
float b = 3;
float c;

c = a / b;

println(c);
```

Tisk do konzole již ukazuje správnou hodnotu *3.333333* .

Pro další operace s čísly bych pro přesnost výsledků důrazně doporučil používat jen float. Další operace mohou vypadat například takto:

```
float a = 3;
float b;

// sq je funkce pro "square", číslo na druhou
b = sq(a);
println(b);

// sqrt je funkce pro "square root", odmocninu
b = sqrt(a);
println(b);

// pow je funkce pro "power", číslo na N-tou
// mínusová čísla v N jsou odmocniny
b = pow(a,3);
println(b);

b = pow(a,-3);
println(b);

// atp.
```

- *char a String*, znak a řetězec znaků, text

  S textem nelze operovat stejně jako s čísly, je logické, že nemůžeme násobit texty mezi sebou. *String* je speciální datatyp pro uchovávání textů v paměti a nakládá se s ním speciálními funkcemi. Prozatím nám bude stačit, když si ukážeme velmi jednoduchou operaci s textem, spojení dvou řetězců dohromady.

  Pro označení hodnoty řetězce se používají dvojité uvozovky. S textem se pracuje následovně:

```
String prvniSlovo = "Ahoj";
String druheSlovo = "ěsvte!";
String slovniSpojeni;

slovniSpojeni = prvniSlovo + "␣" + druheSlovo;

println("ěDv␣spojená␣slova:␣" + sovniSpojeni);
```

Všimněte si prosím vloženeé mezery mezi slova "". Uvozená mezera je počítána také jako řetězec textu. Výsledným tiskem do konzole tedy dostaneme následující řetězec textu:

```
ě
Dv spojená slova: Ahoj ěsvte!
```

Zde jsme provedli jednu ze základních operací s textem, spojování řetězců. *String* lze také spojit s jednotlivými znaky, nebo také s čísly, výsledkem bude ovšem vždy další *String*.

```
int a = 1;
int b = 2;
String slova = "test";

// "slova = slova + ěnco" lze také nahradit znaménkem "+="
// stejným znaménkem které u čísel znamená růpírstek
// tedy namísto:
// slova = slova + " " + a + " " + b;
// ůžmeme zkrátit na:

slova += "␣" + a + "␣" + b;

println(slova);
```

Výsledkem bude: *test 1 2*.

Řetězce se dají dále porovnávat seřazovat, dá se v nich vyhledávat znak či slovo a tak podobně. Pro náše účely zatím postačí si uvědomit rozdílné operace mezi textem a číslem.

- *boolean*, neboli pravda nebo nepravda

  Boolean je nejjednodušším datatypem v Processingu, může vyjadřovat pouze dva stavy. Pravdu *true*, nebo nepravdu *false*. Žádnou jinou hodnotu *boolean* nepřijímá a je proto ,co se paměti týče, velmi úsporný datatyp. Operace s booleany se nazývají logické operace a vypadají následovně:

```
boolean prvniTvrzeni = true;
boolean druheTvrzeni = false;
boolean tretiTvrzeni;

// "&&" značí logickou operaci AND
// výsledek bude TRUE, pravda, JEN pokud
// prvniTvrzeni a druheTvrzeni bude pravda
tretiTvrzeni = prvniTvrzeni && druheTvrzeni;

println(tretiTvrzeni);

// "||" ědv svislé čáry je OR, tj "nebo"
// výsledek bude TRUE, pravda pokud
// prvniTvrzeni nebo druheTvrzeni je pravda
tretiTvrzeni = prvniTvrzeni || druheTvrzeni;

println(tretiTvrzeni);

// posledni zakladni operaci je porovnani
// muzeme porovnat dva booleany pomoci "=="
// dvojiteho rovnitka
tretiTvrzeni = (prvniTvrzeni == druheTvrzeni);

println(tretiTvrzeni);

// pro negativni vysledek je zaporne porovnani "!="
// vykricnik pred booleanem vzdy znaci opak

tretiTvrzeni = (prvniTvrzeni != druheTvrzeni);
```

```
println(tretiTvrzeni);
```

Boolean je možné si představit jako vypínač světla. Zapnuto a vypnuto jsou jediné dva stavy podobného vypínače. Booleany často řídí tok programu, je možné si je představit jako přepínače mezi jednotlivými stavy programu.

6.5.6 Podmínka

Uvedením do stavů programu se můžeme dostat k první opravdové struktuře v programu. Zkusme nyní nastínit jak taková struktura vypadá. Patrně nejpřímější řízení dějů v programu je podmínka. Podmínka říká, jestliže je něco pravda, spusť následné příkazy. Uvedu zde krátký příklad podobné struktury:

```
boolean prepinac = false;
int hodnota;

if(prepinac == true){
    hodnota = 1;
}else{
    hodnota = 0;
}

println(hodnota);
```

Velmi prostá struktura je v tomto příkladě vytvořená jednou podmínkou. Podmínka je v Processingu (a řadě jiných jazyků) značena slovem *if*, ”jestli”. ”Jestli”potřebuje dostat odpověd v kulatých závorkách, ta může být jen pravda nebo nepravda, k tomu se hodí nejlépe již zmíněný *boolean*, nebo například porovnávání dvou čísel, znaků nebo řetězců znaků, tj. operace které nám vrátí *TRUE* nebo *FALSE*.

Definujeme-li tedy náš *boolean* na začátku jako nepravdu, podmínka se v tomto případě nenaplní a program spustí kód uvozený

složenými závorkami následujícími až slovo *else*. V našem příkladě se jedná o druhou větev podmínky, ta sice není povinná ale pro ukázku jsem ji zde rovnou zmínil. Tedy za ukončením *if* a složených závorek můžeme dále slovem *else*, tzn. ”jestliže ne”, říci co se stane v případě nenaplnění naší podmínky.

V tomto konkrétním případě se k proměnné *hodnota* přiřadí číslo *0*.

To program následně ověřuje tiskem do konzole, kde se objeví *0*.

6.6 Hodnota a její zobrazení

K čemu jsou vlastně hodnoty dobré? Tisk do konzole je jen kontrolní mechanismus většinou se nejedná o výslednou podobu programu.

Po celou dobu sčítání a odčítání hodnot jsme nezavolali žádnou funkci, která by kreslila na plátno.

Nyní je na čase ukázat si jakým způsobem Processing rozumí kresbě. Ty samé hodnoty které máme nyní v paměti mohou být použity pro jakýkoli kreslící výstup. Řekněme že chceme z těchto hodnot zobrazit například elipsu. K tomu potřebujeme zavolat funkci pro tvorbu elipsy, pokud ji nechceme zrovna z nějakých důvodů popisovat matematicky (to je samozřejmě dokonale možné).

Processing nemá předdefinované žádné složité tvary. Pracuje sám o sobě pouze s tvary primitivními, jako je bod, linka, trojúhelník, obdélník a elipsa. Pomocí těchto tvarů lze zkonstruovat nepřeberné množství obrazů. Představíme-li si nyní digitálně zpracovanou fotografii, můžeme například říci že je zkonstruována z bodů. Jednotlivé body, tj. pixely mají jinou barevnou hodnotu a takto poskládaný obraz se ve výsledku jeví jako fotografie.

Problém v případě konstrukce syntetické fotografie nespočívá

v geometrii; ta je známa, mřížka bodů s počtem šířky krát výšky obrazových bodů digitální fotografie. Problém je v barevných hodnotách, které neznáme a synteticky jakoukoli matematickou funkcí těžko obsáhneme.

Dat pro výpočet fotorealistického obrazu potřebujeme opravdu hodně, nástroje které dokáží simulovat fotorealistický obraz existují a není jich málo. Dokonce i v Processingu existují podobné rozšíření které buď přímo zpracovává jí a zajištují obrazový výstup.

Tento příklad je trochu extrémní, ale naprosto pravdivý. Věc kterou se snažím ilustrovat je ta, že i s minimálním počtem primitivních geometrických tvarů lze docílit téměř nekonečné (spočitatelně obrovské) množství obrazů, což by nám mělo nadlouho stačit.

Nyní zpět k hodnotám. Máme-li již jakékoli hodnoty v paměti programu, můžeme kterékoli z nich namapovat na jakoukoli kreslící funkci. Zde začíná naše experimentální část, často se totiž stává, že výsledek kreslení neumíme plně předpovědět a teprve zobrazením nám kresba vzhledem k hodnotě začne dávat smysl.

Processing je, co se týče grafického výstupu, navržen pro přímou experimentální interakci vznikajícího programu s uživatelem. To znamená, že pokaždé kdy chcete vidět výsledek kódu, stačí pouze stisknout tlačítko *RUN*. Zpětná vazba spolu s vaší interpretací a úmyslem má potom veliký vliv na následné úpravy v kódu. Tímto způsobem můžete doslova ”vysochat”výslednou podobu programu.

6.6.1 Zobrazení

Jak zobrazit hodnoty, které uchovává program v paměti? Jakákoli hodnota lze použít například jako jeden z argumentů v kreslících funkcích.

Již zmiňovaný *rect()*, tedy rectangle - obdélník, vyžaduje ke svému

úspěšnému vykreslení čtyři parametry, čtyři číselné hodnoty [5]. Tyto parametry, hodnoty, můžeme buďto zadat jako přímé číselné hodnoty, nebo do těchto parametrů můžeme zadat naši proměnnou která obsahuje číslo. Názorně:

```
// první způsob vykreslení obdéníku
rect( 10, 8, 40, 20 );

// druhý způsob vykreslení obdélníku
int prvniCislo = 10;
int druheCislo = 8;

rect( prvniCislo, druheCislo, 40, 20 );
```

Již nyní je nám zřejmé jak hodnoty mohou ovlivnit zobrazení. Namísto hodnot zadaných číselně se zde v druhém způsobu vykreslení obdélníku objevují naše proměnné, které mají již zadaný obsah. Výsledek zobrazení bude v obou případech stejný, ovšem z hlediska struktury programu je druhý způsob daleko flexibilnější.

Představme si nyní, že hodnota zadaná prvním způsobem zápisu se již nikdy v průběhu programu nemůže proměnit. V případě druhého způsobu zápisu získáváme díky možnosti změny jednoho parametru kontrolu nad kreslícím výstupem.

Veškeré příkazy které jsme si zatím ukázali byli spuštěny pouze jednou. To znamená, že obdélník byl vykreslen a program, který již neměl žádné další příkazy jednoduše skončil.

V další kapitole si předvedeme jak vdechnout procesům pohyb a stálost, další kapitola je věnovaná animaci.

[5] zde nezáleží jestli se jedná o celé číslo int nebo číslo s desetinnou čárkou float

6.7 Základní funkce programu

Processing rozlišuje mezi jednotlivými programovacími přístupy. Textový editor se automaticky přizpůsobí způsobu programování. Tímto přístupem se Processing snaží co nejvíce nablížit začátečníkovi. Základní prostředí nám dovoluje pouze nakreslit tvary a ukončit program. K pohybu v čase musíme Processingu definovat základní strukturu programu.

Nejzákladnější dvě funkce se kterými se budeme při programování setkávat jsou funkce *setup()* a *draw()*.

- *setup()* je funkcí, která bude spuštěna jednou na začátku programu. Zde se nejčastěji nastavují počáteční parametry které si potřebujeme připravit než se spustí kreslící funkce.
- *draw()* je funkce pro opakované kreslení, bude spouštěna stále po dobu chodu programu, zde budeme nejčastěji kreslit tvary a provádě proměnlivé výpočty pro animaci.

void draw() je základní funkce pro spuštění překreslování plátna

Základní funkce pro vyvolání překreslování výstupního okna je funkce *draw()*. Funkce je uvozena slovem *void*. Více o funkcích se dozvíme v kapitole. Jedná se o základní smyčku, která zajišťuje jakoukoli proměnlivost, tedy animaci v obraze. Smyčka je de facto operace, která v programu vyjadřuje rozměr času.

6.8 Animace a interakce

6.8.1 Animace

Statická kresba na plátno je jen jeden z možných výstupů. Animace v Processingu znamená překreslovat plátno pokaždé jinými obrazy.

Processing je jazyk využívaný zjeména v pohyblivém obraze. Nyní si ukážeme základní postup při tvorbě pohyblivého obrazu.

Velmi zjedoušeně řečeno smyčka *draw()* je prostředek pro změnu, pozor není změnou sama o sobě. Smyčka zajišťuje sousledné volání příkazů v neustálém trvání[6]. V Processingu si v podstatě uvědomíme, že psaním příkazů do smyčky píšeme pravidla jakoby pro jedno okénko ve filmu. Pohyb je vždy zajištěn proměnou našich datatypů, které v tomto okénku figurují.

Tedy například napíšeme-li takto, samotnou smyčku:

```
void draw(){
    // 60x za řvteinu šěsputná operace
}
```

Vše uzavřeno do složených závorek bude spuštěno šedesátkrát za vteřinu. Bude-li obsah základní funkce *draw()* neměnný, tj. budeme-li volat šedesátkrát za vteřinu to samé, animace bude ovšem v případě vizuálního výstupu jen abstraktním pojmem.

V tomto příkladě již dochází ke spuštění smyčky. Výsledkem bude pouze šedivá plocha v rozměrech sto krát sto pixelů, jelikož na plátno nebylo zatím nic vykresleno. Následujícími příkazy *background()* a *rect()* nakreslíme nejprve plné pozadí a následovně obdélník. Zápis bude vypadat následovně:

```
void draw(){
    // pozadí
    background(255);
    // obdélník se čřtymi parametry
    rect( 10 , 10 , 30 , 30 );
}
```

Takový zápis již provádí animaci a kreslí na plátno. Animace ovšem v tomto případě nebude patrná jelikož zatím se v obraze nic

[6] po dobu běhu programu

Animace je proměna obrazu v čase, jednotlivá okénka se v čase musí lišit.

neproměňuje. Veškeré hodnoty jsou statické tudíž se vykresluje jeden čtverec na bílém pozadí na stále stejném místě.

K rozhýbání objektů jednoduše potřebujeme proměnit jeden z parametrů v čase.Abychom docílili animace zkusme například vložit namísto parametru pro kresbu čtverce v ose X počítadlo okének. Processing má proměnnou s údajem o počtu uběhlých okének pod názvem *frameCount*.

Vyraz *frameCount* lze využít následovně:

```
void setup(){
    size(640,480);
}


void draw(){
    // pozadí
    background(255);
    // pohyb v ose X pomocí ěpromnné frameCount
    rect( frameCount % width , 10 , 30 , 30 );
}
```

Výsledkem tohoto zápisu bude již první animace. Čtverec se bude pohybovat rychlostí šedesáti pixelů za vteřinu z leva do prava. Tato hodnota v podstatě udává počet vykreslených cyklů funkce *draw()*, ta je v tomto případě pouze využita k pohybu vykreslovaného čtverce.

Tento pohyb je velmi názorný ukazuje, že hodnota počtu vykreslených okének lze použít například i jako hodnotu pro animaci. Stejně tak si můžeme založit i vlastní proměnnou, která bude mít identickou funkci.[7]

```
// šnae ěpromnná nazvaná řčpíznane: čpoítadlo
int pocitadlo = 0;
```

[7] v tomto konkrétním příkladě to sice nedává smysl, dá se použít stejně tak standartní proměnná *frameCount*, příklad je zde jen pro ilustraci možností využití proměnných

```
void setup(){
    size(640,480);

}

void draw(){
    background(0);
    rect( pocitadlo, 10 , 30 , 30 );

    // zde žji řčpiíáme hodnotu, lze napsat i zkrácným zápisem:
    // pocitadlo++;

    pocitadlo += 1;

    // zde ůžmeme žpouít modulo na omezení čpoítadla na šříku plátna

    pocitadlo = pocitadlo % width;

}
```

je to jedna ze základních proměnných Processingu, pomocí které lze například časovat animace.[8]

6.8.2 Interakce

Interakce se dá obecně definovat jako vzájemné působení. V počítačové terminologii se nejvíce má na mysli vzájemné působení člověka a stroje. Přímá interakce z podstaty vzájemného působení vyžaduje rozměr času animaci[9]. To umožňuje uživateli přímý vstup do proměnných, tedy do obsahu našich hodnot.

[8] existují i jiné zápisy například pomocí funkce *millis()* ... [doplnit]

[9] tedy program trvající v čase

Je dobré si uvědomit, že **interakce** je do jisté míry při práci s počítačem pozorovatelná neustále. Pouhý pohyb myši lze považovat za interakci člověka se strojem. Pohyb uživatele přímo proměňuje hodnoty vykreslující pozici kurzoru, stiskem klávesy v textovém editoru například píšeme text atd.

Při interakci je nutné do určité míry předvídat chování uživatele. Interakce primárně vychází z fyzické zkušenosti s předměty kolem nás. Interakcí je myšleno vše co může uživatel přímo ovlivnit.

Interakce se dá rozdělit do dvou základních oblastí na interakci přímou a nepřímou, nebo chcete-li na interakci vědomou a nevědomou. Mezi těmito kategoriemi neexistuje jednoznačná hranice.

Obecně je toto rozlišení spíše stupnicí od jednoduchých přímo a okamžitě viditelných jevů po uživatelově vstupu ke komplikovanějším procedurám spouštěným jen s částečným vědomím uživatele.

Základní tedy přímou interakci si můžeme ukázat na následujícím příkladě:

```
void setup(){
    size(640,480);
}

void draw(){
    background(255);
    // šěvimnte si ůparametr mouseX a mouseY
    rect(mouseX,mouseY,30,30);
}
```

Toto je jedna z nejzákladnějších možných interakcí. Vstupem pro pozici kresleného obdélníku se nám stávají dva parametry *mouseX* a *mouseY*.

6.9 Proměnlivost

Potřebujeme-li programu vdechnout život, musíme nejprve pochopit jakým způsobem se program může proměňovat. Proměnlivost se dotýká základní myšlenky programu trvajícího v čase. K tomu abychom takovou proměnlivost mohli okem pozorovat, program může komunikovat pomocí různých výstupů. Processing, který byl navržen s důrazem na vizuální výstup, bude nejčastěji komunikovat právě vizuálně. Obecně řečeno proměnlivost je jakákoli změna hodnot v délce běhu programu, lze si ji představit například jako rozdíl mezi dvěma okénky filmu.

U filmových okének je rozdíl sice velmi těžko definovatelný ilustruje ale dobře celý problém. Film nám spostředkovává iluzi pohybu sledem několika okének za vteřinu. Přesně stejně se chová i vizuální program. Rozdíl mezi filmem a programem je zhruba ten, že program může mít teoreticky dva a více různých konců, nebo nemusí končit vůbec. Zde by se dalo namítnout že program je také nakonec nositelem kauzálního sledu operací. Vtip je v tom, že stejný program spuštěný dvakrát nemusí produkovat stejný výsledek.

Změny v chodu programu se dá docílit několika způsoby. Základní změna v běhu programu je lidská intervence, která je z pohledu programu (respektive tvůrce programu) jen velmi těžko předpovídatelná. Změna v chování programu bez takového vnějšího zásahu je pak vždy umělá. (viz náhodnost)

Vraťme se ovšem k Proměnlivosti. Proměnlivost je nezbytnou součástí animace. Pohyb je proměna stavu v čase. Měli bychom zde mít vždy na paměti, že program nerozumí tomu co zrovna zobrazuje, program rozumí pouze hodnotám. Rozpohybovat lze tyto hodnoty, a tím posléze například jevy, které se v programu zobrazují a člověku dají nějaký smysl.

Pro příklad si můžeme ukázat jak proměna jedné hodnoty ovliv-

ňuje obraz. K tomu abychom mohli hodnotu proměňovat musíme ji nejdříve pojmenovat.

```
int y = 0;
```

Tímto jsme pojmenovali svoji proměnnou. Jednou z možných variant zápisu je okamžité přiřazení hodnoty, tedy y je nyní nula. Pro animaci hodnoty použijeme přírůstek, tedy při kresbě jednotlivého okénka přičteme k hodnotě 1. Abychom viděli nějaký výsledek, můžeme opět kreslit objekt, který bude využívat proměňující se hodnotu v čase.

```
void setup(){
    size(640,480);
}

void draw(){
    background(255);

    // prirustek o jedna
    // lze zapsat i zkracene y++;
    y += 1;

    // kresba
    line( 0 , y , width , y );

}
```

Po spuštění tohoto programu uvidíte animaci trvající 480 okének. Animace bude pokračovat i nadále, hodnota y se bude stále proměňovat. Kresba bude teoreticky probíhat mimo plátno. K tomu abychom kresbu udrželi v mezích plátna, můžeme použít jednoduchou podmínku, která bude vracet hodnotu y zpět na nulu.

```
if(y > height){
    y = 0;
}
```

Celá tato procedura jen ilustruje logiku programu. Hodnota y je zde přímo zobrazena, ale stejně tak se dá použít kdekoli jinde, tedy ne nezbytně na kresbu. Pro lepší představu jiného použití uvedu následující příklad.

Pod hodnotou si lze například představit pevně stanovenou hodnotu, dejme tomu výšku domu. Zadání výšky nemusí nepostaví dům ale stanoví hodnotu se kterou můžeme dál počítat. Známe-li výšku plánované budovy, můžeme například přidávat patra dokud se této výšky nedosáhne. Tento postup již lze považovat za logickou konstrukci a jedním ze základních postupů jak takové logiky v programu docílit je podmínka.

6.10 Podmínka

6.10.1 if

Podmínka je jeden ze základních stavebních kamenů programu. Podmínku lze použít ve všech případech, kdy předpokládáme odpověď ano nebo ne. Podmínka v Processingu stejně tak jako ve většině ostatních jazyků konstruována příkazem.

Konstrukce podmínky vypadá pak následnovně:

```
boolean pravda = true;

if(pravda == true){

    // spust nasledujici blok

}
```

Za povšimnutí zde stojí dvojité rovnítko. Tento speciální symbol se používá při porovnávání dvou stran. Proměnná nazvaná *pravda*

získala hodnotu *true*. V případě podmínky nezáleží na počtu proměnných v kulatých závorkách, otázka může být i více kombinovaná záleží na výsledky, který vždy musí být pravda nebo nepravda, tedy *true* nebo *false*. Pro zkonstruování takové věty lze použít následující znaménka.

```
boolean pravda = true;
int cislo = 3;
String text = "Lorem␣ipsum␣dolor␣et␣amet";

if(cislo == 3){

    // tato podminka je spustena

}


if(cislo != 3){

    // tato podminka neni spustena, znamenko nerovna se je opake

}


if(cislo > 3){

    // tato podminka neni spustena cislo neni vetsi nez 3

}


if(cislo < 3){

    // tato podminka neni spustena cislo neni mensi nez 3

}
```

```
if(cislo >= 3){

    // tato podminka je spustena, znamenko vetsi nebo rovna se, cislo

}


if(cislo <= 3){

    // tato podminka je spustena, znamenko mensi nebo rovna se, cislo

}

if(text.equals("Lorem␣ipsum␣dolor␣et␣amet")){

    // tato podminka je spustena, porovnavani textu je take mozne, os
    // k porovnavani celych retezcu, textu se pouziva funkce equals("

}
```

Takovým výčtem operací jsme si pokryli základní konstrukci podmínky. Další znaménka slouží ke skládání jednotlivých otázek. Znaménko pro logicke A je:

```
boolean a = true;
boolean b = false;

if(a == true && b == false){

    // tento blok bude spusten, a je pravda A b je pravda

}


if(a == true && b == true){
```

```
    // tento blok nebude spusten, b totiz neni pravda, b == fals

}
```

(”appersand”se na české klávesnici nevyskytuje, na anglické klávesnici jej naleznete většinou pod *SHIFT - 7*), pro logické NEBO se používá dvojitá svislá čára tzv. pipe:

```
boolean a = true;
boolean b = false;

if(a == true || b == false){

    // tento blok bude spusten, k tomu staci pouze ze a je pravd

}


if(a == true || b == true){

    // tento blok bude tedy opet spusten

}


if(a == false || b == true){

    // tento blok jiz spusten nebude, ani a ani b neni pravda

}
```

Za pomoci skládání otázek můžeme zkonstruovat celé věty a složitější otázky. V kombinaci s kulatými závorkami si vystačíme pouze s logickým AND a OR. Zkusme si ilustrovat situaci následovně:

```
boolean a = true;
boolean b = true;
```

```
boolean c = false;

if( (a && b) || c ){

    // tento blok bude šěsputn

}

if( !(a && b) || c ){

    // tento blok nebude šěsputn

}
```

Jaký je rozdíl mezi prvním a druhým blokem? Poslední specialita booleanovských operací je znaménko vykřičník. Takové znaménko před proměnou typu boolean znamená opak tvrzení tedy, je li *pravda* pravdou, *!pravda* je nepravdou tedy výsledkem je *false*.

```
boolean pravda = true;

println( pravda ); // tiskne true
println( !pravda ); // tiskne false
```

Dále si všiměte, že v podmínce chybí jakékoli porovnávací znaménko. Ve skutečném programování se setkáte spíše s tímto zápisem, je to zápis zkrácený. Namísto neustálého rozepisování *něco rovná se rovná se true* lze napsat pouze v kulatých závorkách. Je-li naše *něco* datového typu *boolean* Processing bude takovému zápisu rozumět. Tedy nejjednodušší zápis může vypadat i takto:

```
boolean a = true;

if(a)
    //pouze tento jediný řádek bude šěsputn
```

```
if(a){
    // cely tento blok
    // bude spusten
}
```

K úplnému zkrácení se dají i vypustit i složené závorky, to vede opět k rychlejšímu zápisu. Rozdílem takového zápisu je fakt, že taková podmínka spouští pouze jeden následující řádek. Přestože je to naprosto správný způsob programování, nekteří programátoři takový zápis neradi používají z důvodu zhoršení přehlednosti v kódu. Opět se s takovým zápisem často setkáte, člověk je jednoduše líný živočišný druh a *if* se v programování používá opravdu často.

6.10.2 else

K dalšímu rozšíření podmínek slouží příkaz *else* a dá se přeložit takto. Jestliže je něco pravda udělej toto, jestliže ne udělej něco jiného. Zápis vypadá takto:

```
boolean a = false;

if(a){

    // kdyby a bylo true, tak bu se spustil tento blok kodu

}else{

    // ale protoze je nase a false, spusti se nyni kod
    // v tomto bloku

}
```

6.10.3 ? :

Abychom podmínky vyčerpali nadobro ukážeme si poslední možný zápis, který je ještě úspornější.

```
boolean a = true;

if(a)
    background(0);


background( a ? 0 : 255 );
```

6.11 Náhoda

Proč má smysl v případě strojů vůbec hovořit o náhodě. Náhoda je z pohledu výpočetní techniky třeba vnímat jako vnější vliv. V rámci počítačové logiky náhoda v podstatě neexistuje. Každý jev má svoji příčinu a může mít svůj následek. Počítačové jazyky znají jen pojem takzvané pseudonáhodnosti.

Předložka pseudo, již nastiňuje určitou náhražku. Pseudonáhodnost je ve své podstatě strojová simulace náhody v čistě logickém prostředí. Náhodou pojmenováváme zpravidla jev, který nedokážeme pro jeho složitost předpovědět. Reálný svět vnímaný člověkem obsahuje takzvanou pravou náhodu, tedy to, co skutečně nelze vypozorovat.

Situace přenesení náhody do světa logického uvažování je velmi problematická proto, že člověku se velmi těžko stroji popisuje fakt srozumitelnosti. Některé jednoduché pravidla srozumitelné jsou, jiné z pohledu výpočtu podobně jednoduché již ne.

Pseudonáhodou člověk definuje takový výpočet, který samozřejmě vychází z logického výpočtu, člověk jej již ovšem nedokáže před-

povědět.

Konkrétní příklad jak lze v Processingu vygenerovat pseudonáhodné číslo je:

```
float nahodna = random(10);
println(nahoda);
```

Proměnná nazvaná nahoda nyní přijímá pseudonáhodnou hodnotu od nuly do deseti jež následovně vytiskne do konzole. Druhý možný zápis je zadání dvou čísel tedy rozsahu výsledné pseudonáhodné hodnoty.

```
float nahodna = random(10,20);
println(nahoda);
```

Jak je zřejmě pochopitelné náhoda se nyní bude pohybovat v rozmezí čísla od deseti do dvaceti.

6.12 Uspořádání, stuktura programu

6.13 Ukládání informací, paměť programu

6.14 Pokročilejší logické operace

6.15 Pravděpodobnost a trvání

Kapitola 7

Práce s daty

7.1 Parsing, získávání hodnot z externích dat

7.2 Vizualizace hodnot

Kapitola 8

Rozšíření Processingu

8.1 Knihovny

8.1.1 Vestavěné knihovny

Ve skecthbooku se dále nacházejí[1] veškeré rozšíření za pomocí takzvaných knihoven neboli libraries. Ty jsou umístěny v adresáři sketchbooku ve složce *libraries*. Knihovny jsou silným rozšířením celého jazyka a pokrývají mnoho možností s nakládaáním s externími daty, zajišťují komunikaci zařízeními, práci s videem, zvukem, nebo s trojdimenzionálními daty či vektorovým obrazem.

Knihovny jsou v podstatě moduly které se zaměřují většinou na jeden problém. Některé moduly umí například komunikovat s kamerou, jiné umí přenášet informace přes síť nebo komunikovat se zařízeními. Knihovna je sada příkazů, která rozšiřuje samotný jazyk o nové možnosti.

Knihovnami se také dá dobře ilustrovat k čemu uživatelé Pro-

[1] od verze 1.0

cessingu převážně programovací prostředí využívají. To sice nemusí být v žádném případě směrodatné pro vaší práci, může to být ovšem i velmi inspirativní. Knihovny Vám pomohou získat jistý přehled o kontextu užívání Processingu.

Knihovny se nejobecněji dělí na dvě základní skupiny. Jednu tvoří interní knihovny Processingu, které přichází již nainstalované se softwarem a knihovny vější, vytvořené komunitou uživatelů. Veškeré nainstalovaná knihovny, které má Processing k dispozici, lze nalézt pod tlačítkem sketch > Import Library.

Výčet komunitnách knihoven je již poměrné obsáhlý, pro ilustraci zde budeme hovořit jen o knihovnách vestavěných. Věstavěných knihoven je nepoměrně méně, jejich výčet se proměňuje relativně pomalu, jejich dokumentace je kvalitně zpracovaná:

- Video

  Základní rozhraní mezi Apple Qucktime a Processingem. U platformy Linux, tato knihovna dlouhodobě nefunguje kvůli závislosti na propietárním softwaru, musíme tedy využít alternativy. Například implementace nativní knihovny GStreamer.

- Network Zajišťuje základní síťovou + internetovou komunikaci.

- Serial Podpora pro komunikaci se sériovými porty, externí zařízeni typu (RS-232).

- PDF Export Knihovna pro export do PDF.

- OpenGL Technologie pro podporu java implementace akcelerované grafiky JOGL.

- Minim Využívá JavaSound API ke snadné obsluze zvukového výstupu.

- DXF Export Knihovna pro exportování vektorových dat ve formátu DXF (Autocad, 3D).
- Arduino Knihovna určená pro komunikaci s Arduinem.
- Netscape.JavaScript Nese metody pro komunikaci s Javascriptem a Processingovým appletem ve své webové podobě.
- Candy SVG Import Knihovna pro načítání vektorové grafiky ve formátu SVG (Inkscape, Adobe Illustrator)
- XML Import Podpora načítání XML tabulek.

Neříkají-li vám tyto zkratky nic, nevadí, jedná se většinou o datové standarty a typy zařízení se kterými můžete s pomocí knihoven pracovat. Každá z těchto knihoven má svoji dobrou dokumentaci na stránkách projektu, nebo přímo v tzv. *Examples*, příkladech pro každou knihovnu, lze tímto způsobem zobrazit základní nápovědu.

8.2 Nástroje

.

8.3 Komplexní program

.

8.4 Experimenty

Kapitola 9

Rejstřík pojmů

Slovník

background() funkce nastavuje barvu pozadí na plátně. Standartní barva je světle šedá. Zavoláním background s definicí barvy v kulatých závorkách bude vyplněna celá plocha jednolitou barvou. 47

boolean datatyp který může mít jen dva stavy *true* a *false*. 6

Built with Processing doslova znamená: „Postaveno s Processingem“. Jedná se o zvláštní komunitní frázi, která se objevuje se u projektů využívajících Processing. Fráze vyjadřuje vděk všem participantům a tvůrcům Processingu za tvorbu tohoto nástroje.. 17

DATA adresář uvnitř **sketche**, který obsahuje vaše externí soubory (obrázky,textové soubory, atd.). 22

draw() draw je kreslící funkcí Processingu, veškerý kód uzavřený v této funkci bude vykreslen jednou za okénko, v závislosti na náročnosti pak standartně šedesátkrát za vteřinu, tuto kreslící funkci je nezbytná pro animovaný výstup nebo interakci s uživatelem. 46–48

false nepravda, neboli 0. 71

flow nebo-li plynutí, tok, je zvláštním stavem mysli popisovaným programátory, jedná se o stav kdy je člověk plně zanořen do práce a jakékoli vyrušení z tohoto stavu si vyžaduje opětovné nastolování, podle zkušených progrmátorů nastolení takového stavu obvykle trvá 10-15 minut práce s kódem.. 27

frameCount proměnná držící údaj o počtu vykreslených okének od startu programu. 48

GNU / GPL jedna z licencí otevřeného softwaru zaručující otevřenost kódu, kterou v případě dalšího použití vyžaduje i u programů, které tento kód využívají. 21, 72

GNU / Linux Gnu Is not Unix, GNU je projekt založený Richardem Stallmanem, jedná se o operační systém a rodinu programů s otevřeným zdrojovým kódem.. 21

indenting je licence kompatibilni s licencí **GNU / GPL**, jedná se o speciální licenci Univerzity MIT –Massachusetts Institute of Technology. 17, 26

interakce *(lat. interactio od inter-agere, jednat mezi sebou)* znamená vzájemné působení, jednání, ovlivňování všude tam, kde se klade důraz na vzájemnost a oboustrannou aktivitu na rozdíl od jednostranného, například kauzálního působení.. 49

millis() funkce udávající počet uplynutých mili-vteřin od startu programu, narozdíl od proměnné frameCount udává tato funkce reálný časový údaj od startu programu (tj. nezávisí na počtu kreslených okének za vteřirnu). Tato hodnota je hodnotná při

přesném časování animace. V případě složiějších kreslících operací se pro časování nbo zvláště ukládání animací do videa je pro časování animace tařka nepoužitelná kvůli zdržení každého okénka.. 49

mouseX proměnná záskávající pozici kurzoru na plátně Processingového programu v ose X. 50

mouseY proměnná záskávající pozici kurzoru na plátně Processingového programu v ose Y. 50

otevřený software software s veřejně dostupným zdrojovým kódem. 17

rect() funkce pro kreslení obdélníku, přijímá čtyři parametry počátek x, počátek y, šířku a výšku. Centrování obdélníku se dá modifikovat funckí rectMode(). 44, 47

setup() základní funkce pro nastavení výchozích parametrů programu, tato funkce je spuštěna vždy na začátku běhu programu. 46

sketch nebo-li *náčrt*, je koncept v prostředí Processing uchovávání jednotlivých projektů do adresářů, jedna *sketch* je adresář, kam Processing ukládá data uživatele (tj. především kód a externí soubory). 71

slovník právě se zde nacházíte. 6

stroj Stroj je technické zařízení, které přeměňuje jeden druh energie nebo síly v jiný - ať už kvalitativně nebo kvantitativně. Původně byly stroje jen mechanické, ale dnes se tak označují

i zařízení pracující na jiných fyzikálních či technických principech - například elektrický transformátor. Strojem je v této knize téměř výhradně myšlen počítač. Počítač je programovatelný typ stroje který přijímá vstup, ukládá a zpracovává data a umožňuje výstup v požadovaném formátu.. 11

syntax highlighting barevné značky slouží k lepší orientaci programátora v kódu.. 25

true pravda, neboli 1. 71

void neboli prázdná funkce, z anlgického „prázdno“ funkce která nevrací zpět žádný výsledek, funkce která spouští sérii zadaných příkazů uzavřenýc ve složených závorkách za svoji definicí. 46