1. Definice algoritmů, formy zápisu, asymptotická složitost

Algoritmus

- posloupnost konečných kroků, které musíme učinit k dosažení žádaného cíle neboli přesný návod/postup, kterým lze vyřešit daný typ úlohy.
- v programování se algoritmem myslí teoretický princip řešení problémů
- typickým algortitmem je například kuchařský recept, kdy kuchař či osoba postupuje podle předem daného seznamu postupů
- **Vlastnosti:**
  - *resultativnost* - musí mít alespoň jeden výstup, algoritmus vede od zpracování hodnot k výstupu
  - *determinovanost* - každý krok v algoritmu musí být jednoznačně definován a musí být naprosto zřejmé co a jak se má provést a pokračování algoritmu
  - *fitnost* - musí mít začátek a konec a musí proběhnout v konečném počtu kroků
  - *elementárnost* - algoritmus se skládá z jednoduchých kroků
  - *obecnost* - algoritmus neřeší konkrétní probjém jako je např "3x7" ale obecnou třídu obdobných problemů - řeší tedy součin dvou celých čísel
- **Formy zápisu**:
- *slovně* - např. postup při vaření polívky na jejím balení
- *graficky* neboli vývojovým diagramem. každý diagram by měl mít start a konec(označují se oválem/obdelníkem se zaoblenými rohy) dále může obsahovat větvení(čtverec nakloněný o 45°) a dále dílčí příkaz zpracování(klasický trojúhelník-druh provedení dané operace nebo její skupiny jejichž výsledkem je přeměna informace) dále výstup/vstup(zkosený obdélník). Jednotlivé elementy se spojují svislými/vodorovnými čárami a směr se udává šipkou

Asymptotická složitost

- způsob klasifikace počítačových algoritmů
- určuje operační náročnost algoritmu tak, že zjištuje jakým způsobem se bude měnit chování algoritmu v závislosti na změně velikosti vstupních dat
- zapisuje se pomocí Landauovy notace též Omikron notace

2. Druhy programovacích jazyků, syntaxe, sémantika, zdrojový kód, kompilátor, základní datové typy, deklarace, inicializace, přiřazení, mat. operace

Druhy programovacích jazyků

Programovací jazyk - komunikační nástroj mezi programátorem a počítačem, je to nějaký prostředek pro zápis algoritmů, nástoj na definování algoritmu; jazyk pro vytváření programů (binární podoba algoritmů
Dělení

1. **Podle míry abstrakce**
   1. *Vyšší*
      1. Procedurální - popisuje výpočet pomocí posloupností příkazů a určuje algoritmus, jak danou úlohu řešit
         1. Strukturované/sekvenční(Pascal, C, Basic, Fortran, Python) - program píšeme logicky, tedy od začátku programu směrem ke konci, vykonává se tak dlouho dokud se nevykoná poslední instrikce programu

        2. Objektově orientované(C++/C#, Java, Typescript, Rust) - program píšeme v libovolném pořadí, protože se jedná o krátké procedury které se spustí při startu a vykonají při definované události - nikdy ale nekončí, resp. autor musí zajistit jeho ukončení
    2. Neprocedurální - programování pomocí definic co se dělat má a ne jak se to má dělat
        1. Funkcionální
        2. Logické
2. *Nižší* - posloupnost strojových principů prováděných procesorem zapsané pomocí číselných kódů/instrukcí, které jsou uloženy jako sekvence bitů. pro překlad se používá kompilátor

2. Podle způsobu překladu a spuštění

1. *Kompilované* - Java, C/C++, Visual Basic. Před spuštěním jsou nejprve kompletně přeloženy kompilátorem. Výsledkem je větší rychlost, ale také větší náročnost na správně zapsaný kód.Výstupem kompilace je spustitelný binární kód (.exe), ze kterého nelze jednoduše zpětně odvodit, jak vypadal původní zdrojový kód.
2. *Interpretované* - Python, Ruby, PHP. Jsou pomalejší protože se překládají do strojového kódu za běhu

3. Podle typu platformy

1. *Webové* - HTML, CSS, PHP(Laravel, Nette), Ruby(on rails), JS/Typescript(Next.js, Angular.js, Astro) , Rust, Python(Flask, Django), C#(.NET, Blazor), WebAssembly, Java(Spring)
2. *Mobilní aplikace*
    1. Swift - vyvíjený firmou Apple, dá se použít pro psaní mob. aplikací, webových aplikací(Vapor) nebo serverových aplikací a také IoT
    2. Java a Kotlin(JetBrains) - pro Android, Kotlin je navržen pro interoperabilitu(schopnost různých systémů vzájemně spolupracovat, poskytovat si služby, dosáhnout vzájemné součinnosti) s knihovnami Javy a na některých knihovnách jádra dokonce závisí
    3. Hybrid - JS(React Native), Dart(Flutter), Python(KivyMD)
3. *Desktopové aplikace* - Java(JavaFX), Python(Tkinter), C++,

Syntaxe - soubor pravidel, který definuje kombinaci symbolů jež jsou považovány za správně strukturovaný dokument v daném jazyce -> platí pro zdrojový kód nebo značkovací jazyky, kde dokument představuje data např. if( )
Sémantika - disciplína zabývající se významem příkazů jako je nap . | : | = | ==
Zdrojový kód - text, který píše programátor, je to posloupnost příkazů v programu
Datový typ - definuje druh nebo význam hodnot, kterých smí proměnná nabývat. V programu je označení pro id, které uchovává informaci, může nabývat (ne)známých informací - hodnota. její název je case-sensitive. Pojmenování proměnné nesmí začínat číslicí, nepoužívá se diakritika, interpunkce, prázdné znaky(krom podtržítka)

Jednoduché datové typy

1. int - celé číslo, signed int - hodnoty od minus do plus, unsigned int - pro kladná čísla, short int - pro čísla od minus do cca 65k
2. float, double - reálné číslo, odděluje se tečkou
3. char - znak (a,V, W, 2, / , % atd.), v pocitaci jsou znaky definovány v ASCII/UNICODE tabulce od 1-255
4. string - řetězec/text
5. Boolean - hodnoty false(0) nebo true(jakékoli celé nenulové číslo)
6. Void - funkce bez navratové hodnoty

Inicializace - při deklaraci proměnné do ní přiřadim hodnotu např. String name = "Michal";
Deklarace - int i; bool isPressed;

Matematické operace

1. klasické operátory - +,-,*,/
2. %(modulo) - zjištění zbytku po dělení
3. relační operace - ==,<=,>=,!= - vrací boolovskou hodnotu
4. konjunkce(&&) a disjunkce(||)

= - přiřazení

sizeof - vrací velikost datového typu nebo objektu v bajtech, velikost se může lišit na PC v závislosti na překladači

3. Řídící programové struktury – podmíněné příkazy

Pro provedení příkazu je potřeba splnění podmínky

Příkazy

1. If, else - používají se k rozhodnutí o dalších krocích programu na základě podmínek, zkrácený zapis - Ternární operátor -(podminka)? pravda:lez
2. switch - pro více násobné větvení, přehlednější narozdíl od if-else if-else
   -case - označení případu, break - ukončení případu, default - nepovinný, nastane v případě že case nevyhovuje podmínce ve switchi

4. Řídící programové struktury – cykly

-používáme pro provedení opakované činnosti

Typy cyklu

1. For - cyklus se známým počtem iterací(opakování), můžeme také použít pro iteraci přes prvky pole/kolekce
2. While - cyklus s podmínkou na začátku
3. Do while - cyklus s podmínkou na konci, provede se minimálně 1 pak porovná podmínku a na jejím základě rozhodne zda bude dále pokračovat

5. Složené datové typy – pole, množina, struktura, lineární seznam

1. **Pole(Array)** - datová struktura sdružující vždy konečný počet prvků(int, string, char atd.) stejného datového typu a jehož velikost při běhu programu zůstává neměnná - homogenní dat. typ
   - deklarujeme pomocí [ ]
   - používá k uložení více hodnot do jedné proměnné, namísto deklarování samostatných proměnných pro každou hodnotu
   - k jednotlivým prvkům se přistupuje pomocí indexu(celé čísla), 1. prvek v poli má index 0!
   - počet prvků můžeme předem určit nebo vložit hodnoty a velikost se nastaví sama
   - pomocí forEach loopu můžeme vypsat jednotlivé prvky
2. **Množina(Set)** - struktura bez uspořádání, rychlé dotazování na přítomnost prvku
   - jedná se o kolekce kde je každá položka jedinečná, duplicita není povolená
   - neexistuje ani klíč (mapy) ani struktura (seznamy)
   - prochází se pomocí třídy Iterator nebo forEach cyklem
   - Dělí me je na
     1. TreeSet - seřazená, nepovoluje null
     2. LinkedHashSet - neseřazená ale o to rychlejší, povoluje null
     3. HashSet - nejrychlejší, řazení při vložení a povoluje null
3. **ArrayList** - lineární struktura, třída dědí z rozhraní List, uspořádaná kolekce(sekvence) je asynchronní
   - přístup k libovolnému prvku v konstantním čase – stejně tak přidání prvku na konec
   - nevýhodou je nutnost kopírování všech prvků do nového pole (rozpínání a zmenšování kolekce), operace vkládání a mazání
   - Uživatel tohoto rozhraní má přesnou kontrolu nad tím, kam je v seznamu vložen každý prvek
   - implicitně narůstá o 50% své velikosti
4. **Struktura** - deklarace složeného dat. typu, který definuje skupinu proměnných, s vlastním dat. typem, v op. paměti, se kterými umožňuje pracovat jako s celkem
   - hodnoty jsou pojmenovány slovy, nikoli indexem

6. Podprogramy – předávání parametrů, ukazatele a dynamická alokace

- podprogramy se v jazycích odvozených z C nazývají funkce
- Funkce by měla mít
  1. Návratovou hodnotu - má jednu návratovou hodnotu a dat. typ se píše před název fce
     - fce main má vždy hodnotu int
     - při nenavraccování dat použijeme datový typ Void
  2. Název - id fce, za ním by měly následovat ( ) - okud nebudou, reprezentuje se pouze adresa v paměti jejího začátku
  3. Parametry - píšeme je do ( ) , skládá se z datového typu a jména, oddělují se čárkou
     - popisuje očekávaná data, se kterými bude fce pracovat
     - void nemá žádné parametry
  4. Return - příkazem můžeme kdykoli ukončit fci, uvádíme za něj návratovou hodnotu fce, po provedení bloku příkazů se ukončí
- Předávání parametrů
  1. Hodnotou - vytváří se kopie proměnné pro fci a spolu s ní zaniká
  2. Odkazem - odkazujeme na již existující proměnnou, pokud s ní budeme pracovat dále, změny se projeví
- Ukazatele(Pointer) - proměnná, která uchovává pouze adresu v paměti,
  - technika je užitečná, když neznáme přesnou velikost paměti při kompilaci a chcete ji alokovat nebo uvolnit během běhu programu
  - před proměnnou, na kterou poukazujeme a chceme získat jeji hodnotu píšeme *(dereference)
  - hodnotu získáme pomocí znaku &
- Dynamická alokace paměti - proces, při němž program vytváří paměťový blok na haldě (heap) a získává k němu přístup pomocí ukazatelů
  - fce malloc(), která vrací ukazatel na začátek alokované paměti, v C++ můžeme použít operátor new pro alokaci objektů
  - uvolnění paměti - v C pomocí fce free(nazevPole) a v C++ delete[ ] nazevPole

7. Embedded systémy a Arduino (digitální a analogové piny, PWM, IIC, SPI)

Embedded systémy

- čj. zabudovaný systém, vestavěný systém
- jednoúčelový systém , ve kterém je řídící počítač zabudován do zařízení jež ovládá
- jedná se kombinaci hardwaru a softwaru vestavěného do zařízení
- jsou určené pro předem definovanou činnost
- zakladní konstrukční jednotkou je **mikropočítač** nebo **mikrokontrolér** a dále nějakou množinu periferních zařízení
- Mezi nejznámější výrobce mikropočítačů patří **Raspberry** a mezi mikrokontroléry je to **Arduino** či rozšířená verze od firmy **Espressif** nebo **Wemos**
- mezi potřebné periferie řadíme **analogové** a **digitalní** převodníky, seriová rozhraní UART, SPI nebo I2C, PWM generátory, GPIO.
- k temto řídícím jednotkám můžeme připojovat moduly
- ty pak můžeme rozdělit na vstupní(sbírají hodnoty, data- senzory, klávesnice a tlačítka, gyroskopy a kamery), výstupní(vypisují, prezentují data-displeje, led moduly, čerpadla) a vstupně výstupní(RTC, ethernetové moduly či čtečky karet) nebo komunikační zažízení(Bluetooth, LORA, Wifi, GSM sítě). Dále relé, shieldy

Arduino

- italská firma, která vyrábí vývojové desky
- původně bylo vytvořeno jako náhrada za drahé vývojové sety
- první deska vznikla v roce 2005 a měla velký úspěch, že se ho tvůrci rozhodli poskytnout celému světu
- jedná se open source projekt, proto se můžeme setkat s různými klony
- srdcem Arduina je procer od firmy Atmel
- funguje na Tranzistorově-tranzistové logice
- programují se v C/C++ a Wiring, microPythonu dokonce i Rustu či Node.js
- 2 funkce - setup()-spouští se na začátku(zahájení komunikací, inicializace a deklrace pinů)
    - loop() - opakuje se dokola dokud se Arduino nevypne
- program se skládá ze 2 funkcí s nenávratovou hodnotou - setup(slouží k nastavení pinů, komunikace-BLE,Wifi,ESP-NOW,webserver) a loop(provádění úkonů)
- Existuje mnoho platforem jako je Arduino Uno a Nano 3.0 osazena procesorem ATmega328,
- Digitální piny - pro komunikování s moduly, shieldy se používají I/O porty - piny
    - pro práci s piny používáme např. fci pinMode(cisloPinu, čtení/zápis);
    - čtení z pinu - digitalRead(12)
    - zapisování - digitalWrite(cisloPinu, 1/0);
- Anologové piny - pro komunikování s moduly, shieldy se používají I/O porty - piny(A0-A5 nebo PWM) podporující analogové fce - analogWrite(cisloPinu, 1/0) a analogRead(12)
    - jedná se však o PWM simulovánní, hodnoty nabývají rozsaho 0-255(až 1 bajt)

ESP

- čínské čipy, které byli určené do shieldů pro síťovou komunikace ale nakonec díky překvapivému výkonu vznikly samostatné disky s čipy
- nabízí komunikace přes Wifi, Bluetooth a Low energy, ESP-NOW, MQTT, některé desky jsou obohacené o Lora či SIM8000L
- dále je možno naprogramovat asynchronní webserver, komunikovat pomocí HTTP a JSON, či ukládat dat do databázi jako je MySQL nebo Influx
- dále dokáže pomocí API komunikovat např. s Telegramem, Firebase nebo Grafanou
- nabízí dual-core CPU a vrežimu spánku spotřebuje 5 mikroAmpér

- PWM - vlnovka na arduinu - pulzní šířková modulace - rychlé střídání napětí 0 a +5V - projeví se snížená účinnost(ledka ve skutečnosti bliká, motor se točí pomaleji)
- IIC-seriová sběrnice od firmy Phillips, pro připojení periferií s nízkou rychlostí přenosu dat, pomocí pinů SCL(Synchronous Clock) a SDL(Syn. Data) můžeme pomocí 4 vodičů připojit až 128 zařízení, LCD Displeje
- SPI -Serial Peripheral Interface - používá se pro přenos dat mezi MCU a periferiemi
    - rychlejší než UART a I2C, jasně oddělené piny
        - možné pomocí pinů MISO(Master in Slave out) přenos dat z periferie k masterovi, MOSI(Master out Slave in),
        - SCK(Serial Clock)-hodinový signál pro synchonizaci
        - CS(Chip Select) - označuje aktivní periferie
- UART - liší se asynchronní komunikací, vodiče RX-reciever a TX-transmitt

8. Základy OOP – pojmy třída, objekt, zapouzdření, instanční a statické metody a atributy, konstruktory, „destruktory“ (finalize)

-v OOP se odprošťujeme od psaní logických skriptů do programu pod sebou ale skládáme komponenty

- základními koncepty jsou třídy, zapouzdření, polymorfismus, dědičnost, abstrakce,
- **Třída(class)** - základní konstrukční prvek OOP, předpis/předloha/vzor pro vnik objektů
  - říká nám jaké vlastnosti a metody mají objekty z odvozených tříd
- **Atributy -** proměnné uchovávající v objektu, popisují jeho stav, přistupování pomocí getterů a setterů
- **Konstruktory -** inicializace objektu, metoda volána při vytváření objektu, Java automaticky vytváří defaultní konstruktor pokud není definovaný
- **Destruktory -** v C++ s destruktor zapisuje tyldou, Java nemá tradiční destruktor ale garbage collector který automaticky sleduje objekty a neodkazované eliminuje
- **Objekt -** je instance třídy**,**vytvoření pomocí klíčového slova new, poté se pracuje s daty
- **Zapouzdření -** umožňuje skrýt atributy a metody tak, aby zůstaly použitelné jen pro třídu zevnitř, zaručuje, že objekt nemá přístup k "vnitřnostem" jiných objektů
- **Instanční metody -** metody příslušící konkrétní instanci, pro své volání pomocí tečky potřebují objekt např. Math.pow()
  - každá instanční metoda má proměnnou this, která odkazuje na objekt, pro který je tato metoda volána
- **Statické metody** - metody příslušící celé třídě, nepotřebují k spuštění instanci např třída main()

9. Dědičnost a polymorfismus v OOP

Dědičnost - třída, která je z nějaké zděděná se nazývá potomek

- třída, ze které jsme dědili se nazývá rodič
- defunujeme ji **extends** nazevDedeneTridy{} který se píše za jméno třídy
- dědií se metody a atributy **public** a **protected**, **private** nikoli
- dědit bez zapouzdření je možně v rámci balíků jelikož nedefinuje specifikátory explicitních přístupů
- k vynucení datových členů rodiče se používá kvalifikátor **super**
- konstruktory základní třídy se nedědí
- dědíme abychom vylepšili chování nebo změnu původní třídy nebo polymorfismu

Polymorfismus - schopnost přeměnit se do různýc forem, umožňuje přepsat metodu u každé podtřídy tak, aby dělala, co chceme

- funguje pouze u objektů odvozených tříd
- metoda pro využití polymorfismu musí bít členem základní třídy
- nastává pouze u dědění
- zvířata, která všechny mluví mluvit() - každě ale odlišně
- **Abstraktní třída(abstract)** - třída, v níž je deklarováno ale neimplementováno jedna nebo víc metod - ty se označují jako abstraktní, nemají definované tělo -> upraví si je v podtřídách
- nesmí se z ní vytvářet objekty, nemůže být private, používá se jako předek pro další třídy
- **Rozhraní(interface)**
- rozhraním objektu se myslí to jak je viděn zvenku, definuje formu metod(její parametry a návratové typy) a konstanty
- metody v rozhraní jsou public a abstract, konstanty public,static a final
- pomocí rozhraní řekneme jak se s objekty zděděné třídy může jednat
- **Anonymní třída** - lokální třída na jedno použití
- jelikož je v daném místě ihned použita není ptřeba pojmenovávat -> anonymní
- používá se pro vytvoření objektu, který bude pouze předán metodě

10. Kolekce v jazyce JAVA (seznamy, množiny a mapy)

1. **Seznam** - lineární struktura
    - Uspořádaná kolekce (sekvence)
    - Uživatel tohoto rozhraní má přesnou kontrolu nad tím, kam je v seznamu vložen každý prvek
    - k prvkům se přistupuje podle jejich celočíselného indexu (pozice v seznamu) a vyhledávat prvky v seznamu.
    1. **ArrayList** - lineární struktura, třída dědí z rozhraní List, uspořádaná kolekce(sekvence) je asynchronní
        - přístup k libovolnému prvku v konstantním čase – stejně tak přidání prvku na konec
        - nevýhodou je nutnost kopírování všech prvků do nového pole (rozpínání a zmenšování kolekce), operace vkládání a mazání
        - Uživatel tohoto rozhraní má přesnou kontrolu nad tím, kam je v seznamu vložen každý prvek
        - implicitně narůstá o 50% své velikosti
    - **LinkedList** - spojový seznam, výhodou je absence relokace a nevýhodou absence libovolného přístupu v realném čase
2. **Množiny** - struktura bez uspořádání, rychlé dotazování na přítomnost prvku
    - jedná se o kolekce kde je každá položka jedinečná, duplicita není povolená
    - neexistuje ani klíč (mapy) ani struktura (seznamy)
    - prochází se pomocí třídy Iterator nebo forEach cyklem
    - Dělí me je na
        1. TreeSet - seřazená, nepovoluje null
        2. LinkedHashSet - neseřazená ale o to rychlejší, povoluje null
        3. HashSet - nejrychlejší, řazení při vložení a povoluje null
3. **Mapy** - kolekce, která mapuje klíče na hodnoty(asociativní pole)
    - TreeMap - prvky seřazené od A-Z nebo přizpůsobené pořadí, povoluje NULL jako hodnoty, spotřebuje méně paměti - pomalejší na výkon,porovnávání klíčů pomocí compare() nebo compareTo()
    - HashMap - neposkytuje řazení, povoluje jeden NULL jako klíč a a libovolný počet NULL hodnot, spotřebuje více paměti ale je rychlejší, porovnávání klíčů pomocí equals()
    - LinkedHashMap -zachovává pořadí prvků při řazení, povoluje jeden NULL jako klíč a libolný počet NULL hodnot, je pomalejší než HashMap protože spotřebuje více paměti ale funguje stejně

11. Tvorba grafického uživatelského rozhraní (GUI) – principy JavaFX aplikace

- GUI - graphic user interface - umožňuje ovládat pomocí interaktivních grafických ovládacích prvků
- je to prakticky vše s čím se user setkává - frontend -> Okno ->tlačítka, popisky, ikony = pomocí tohoto komunikuje uživatel s programem běžící v terminálu
- V Javě se GUI dá vytvořit pomocí JavaFX a nástroje SceneBuilder a Java FXML soubor
- **JavaFX**
- vstupním bodem programu je třída, potomek abstraktní třídy Application(musí se zdědit) - iniicializuje(init()) a vytváří(start()) GUI, metoda stop() ukončuje program
- pro aplikace na PC a Android
- **skládá se z**
- Stage(jeviště) - reprezentuje okno apliace
- Scene(scéna) - může jich v oně být více, jen 1 viditelná
- Node(herci) - jakkýkoli viditelný prvek scény, lze ho upravovat,
    - zinteraktivnit pomocí animací a efektů
    - dokáže reagovat na události

12. Správci rozložení

- správci rozložení = layouty
- automatické pozicování prvku v panelu podle nastavených pravidel
    1. **Pane** - layout pro ukládaní dalších prvků, umožňuje nastavit velikost pomocí metod setMinSize(), setMaxSize(), setPrefSize() které je nejpoužívanější, nemá pozicovácí algoritmus, vždy od levého horního rohu, prvky se pozucují absolutně
    2. **FlowPane** - skládá další nody za sebe tak jak je přidáváme, pomocí Hgap a Vgap můžeme nastavit mezery mezi nody, pomocí setAlignment() lze nastavit pozici uzle
    3. **HBox** - skládá prvky do vodorovného řádku, lze nastavit vlastní okraje a mezery pomocí setSpacing()
    4. **VBox** - skládá prvky do svislého sloupce, stejné modifikování a vlastnosti jako u HBoxu
    5. **BorderPane** - častá základ aplikace - 5 částí-> horní, dolní, levá, prává a střed, může v nich být **pouze 1 node!**,
    6. **GridPane** - ze všech nejsložitější, nepravidelná mřížka, nody mohou zabírat několik řádků/sloupců, metody addRow() a addColumn()
    7. **TilePane** - pravidelná mřížka, preferovaný počet řádků/sloupců, potomkům lze nastavovat zarovnání okraje
    8. **Stackpane** - uzly vkládá na sebe jak je přidáme
    9. **Anchorpane** - nody kotví ke stranám
    10. **TitledPane** - rozbalovací panel

13. JavaFX Nody

- prvek
- převážná část UI je složena v balíčku javafx.scene.control
- lze nastavovat vzhled, animovat pomocí CSS s prefixem -fx-

1. Label(popisek) - needitovatelný text
2. Button(tlačítko) - umožňuje zpracovat akci na klepntí, může zobrazovat text/obrázek, setOnAction() - metoda na kliknutí, parametrem je actionevent
3. Image(obrázek) - ImageView img = new ImageView(new Image(getClas().getResourceAsstream("/url")));
4. RadioButton - jedná se o potomka třídy ToggleButton → může být vybrán nebo zrušen, b jsou přepínače kombinovány do skupin, kde je poté možný výběr pouze 1 z N
5. Checkbox(zaškrtávač) - vhodné na současný výběr mnoha možností
6. Textfield - přijímá a zobrazuje zadávání textu od usera, deklarace stejná jako u předešlých
7. ScrollBar - posuvník

14. Výjimky(exception) a jejich zpracování

- způsob signalizace vážných problémů, nemusí znamenat chybu spíše neobyčejnou událost, které je zapotřebí věnovat pozornost
- výhodou je řešení chyby odděleně od zbytku kódu
- dělíme je kontrolované - musíme je ošetřit Exception a níž, a nekontrolované - nemusíme je ošetřit, Error a Runtime a podřazené
- rodičem je třída Throwable
- vlastní výjimky dědíme ze třídy Exception
- nemůžeme každou část k´du zachycovat do výjimky - nepřehledné a zatěžují program
- Exception je objekt, který se vytvoří při nenormální situaci v programu, uchovává info o povaze vzniklého problému
- výjimku vyvoláme přes throw -> objekt je předán,řešen a zachycen přes catch
- Error - nezachytáváme je
- Exception - musíme ošetřit
- výjimky mžeme ošetřit 2 způsoby
- Propagovat ji - předáme informaci nadřazené metodě o možnosti vzniku výjimky - throws Error{}
- Obsloužit ji v metodě kde vznikla
    - try - vymezuje kód , kde může výjimka vzniknout
    - catch - je povinný, zde je kód, kde se ošetřuje

- finally - nepovinný, provede se bez ohledu na vznik výjimek

15. Vlákna a paralelní programování

- *paralelní zpracování* =>provádění 2 a více úloh současně
- 2 druhy zpracování
  1. na bázi procesů - současné provádění mezi více programy
  2. na bázi vláken - souběžné zpracování v rámci jednoho programu
- **Vlákno** - běžící část programu -> více částí programu může běžet současně - asynchronně
- může se nacházet ve 4 stavech
  1. Běžící - vlákno se provádí
  2. Pozastavené - vlákno je pozastaveno - obnovení probíhá od doby pozastavení
  3. Blokované - nelze přistoupit ke zdroji, jelikož ho používá jiné vlákno
  4. Ukončené - provádení bylo zastaveno a nelze obnovit
- pozastavení vlákna vůči ostatním je určeno jeho prioritou - ta nemá smysl pokavaď běží 1 vlákno
- Vlákno můžeme vytvořit 2 způsoby
  1. děděním ze třídy Thread
  2. implementovat rozhraní Runnable
- **Thread** - metoda run() - vstupní bod vlákna -> musíme vždy implementovat
  - sleep() - pozastaví vlákna
  - start() - spouští vlákno
- **Hlavní vlákno** - všechny aplikace v Javě se pouští jako nové vlákno -> nazývá se jako hl. vlákno
  - vlákna, která spustí hlavní vlákno se nazývají dceřinná
  - je prvním a posledním vláknem aplikace
  - odkaz na hlavní vlákno získáme voláním statické metody currentThread() - vrátí odkaz na vlákno kde je tato metoda
- **Priorita** - celé číslo od 1-10(nejvyšší priorita)
  - výchozí priorita je na čísle 5
  - vyšší priorita nemá přednost -> obírá tak nižší o zdroje
  - nastartujeme metodou setPriority(priorityValue)
- **Synchronizace** - v danou chvíli může zdroj používat jen 1 vlákno
  - řešit se to dá pomocí semaforu -> ten mají každý objekt v Javě
  - synchronized() - aktivuje semafor - vlákno, které vstoupí jako první ovládne semafor a ostatní vlákna čekají na uvolnění
  - zde je problémem deadlock -> když máme více zdrojů a každé vlákno si zabere jedno z nich
- **Komunikace** - používáme metody
  1. wait() - pozastaví aktuální vlákno až do volán metod
  2. notify() - spustí znovu vlákno
  3. notifyAll() - znovu spustí více vláken

16. Událostmi řízené programování

- událost je interakce usera s GUI např. stiskem myši nebo klávesy
- v JavaFX je událost reprzentována objektem třidy javafx.event.Event nebo kteroukoli její podtřídou
- *3 vlástnosti událostí*
  - zdroj - myš
  - cíl - objekt Circle
  - typ - kliknutí myši
- obslužný program - část kódu, který je spouštěn při reakci na událost
- *mechanismus tpracování události*
  - Výběr cíle události
  - Konstrukce trasy události
  - Přechod trasy události
- **Hlavní třídy**
  - Event - reprezentuje událost, rodičem všech událostí
  - EventTarget - objekt, který má toto rozhraní je cílem události
  - EventType - instance této třídy definuje typ události

- EventHandler - instance této třídy představuje obslužnou rutinu
    - má metodu handle() která se při události volá
  - **Mouse Event** - MOUSE_PRESSED -> reaguje na stisk tlačítka myši
        - MOUSE_CLICKED -> reaguje na kliknutí
  - **Action Event** - představující nějaký typ akce. Tento typ události se široce používá k reprezentaci různých věcí, například když bylo stisknuto tlačítko, když skončil KeyFrame a další podobná použití.

17. Fáze vývoje informačního systému a UML diagramy

1. Analýza
   - seznámit se s požadavky klienta a uvědomit si rizika
   - splnitelnost - zda je přání zákazníka splnitelné
   - dále se musíme domluvit na ceně, času a kvalitě
2. Návrh
   - vychází z analýzy
   - např na papír -> Figmy, Adobe XD
3. Implementace
   - převádíme návrh do zdrojového kódu
   - za účelem spustitelné aplikace
4. Nasazení
   - nasadíme software do provozu a do prostředí, jaké požaduje klient

- Testování
  - ověřujeme tím správnost a funkčnost našeho softwaru
  - mělo by nastat v každém bodě vývoje
  - Validace - kontrola údajů při zadání dat např. aby user nemohl zadavat nereálné věci
  - Verifikace - ověření a potvrzení pravdivosti dat např. email ve správném formátu
  - v testovací fázi vývoje se odstraňujií chyby a navíc díky přicházíme na nové vychytávky našeho softwaru
- Analýza
  - obsahuje 2 požadavky
    - Funkční - co má software dělat, co vše v něm bude
    - Nefunkční - požadavky klienta na design, výkonnost
  - znalosti o softwaru získáváme:
    - Dialogem se zákazníkem
    - Dialogem s expertem - vychytávky a na co si dát pozor
    - Prozkoumávání již existujícího softwaru např z Githubu
    - Dotazníkem
    - Prototypováním
    - Brainstorming s moderátorem
    - společný vývoj aplikací za přítomnosti moderátora, písaře, zákazníků a vývojářů
  - výsledky analýzy se reprezentují pomocí UML diagramů - způsob, jak vizuálně znázornit architekturu, návrh a implementaci komplexních softwarových systémů:
    - Use-case diagramu - diagram případu použití a specifikace
    - Případy použití čili koncentrování se na výjimky

18. SQL – pojem relační databáze, návrh tabulky (NF), základní operace (CRUD), primární klíč a cizí klíč

- Relační databáze - relační databáze jsou typem databázového systému, který ukládá a organizuje data do tabulek (neboli relací), které jsou vzájemně propojené pomocí klíčů.
- Tyto klíče slouží k identifikaci vztahů mezi daty v různých tabulkách. Relační databáze jsou založeny na relační algebře a teorii relací.
- Zde jsou některé klíčové prvky relačních databází:
  - **Tabulky (Relace)**: Data jsou organizována do tabulek, kde každá tabulka reprezentuje určitý typ entit (např. zákazníci, objednávky, produkty).
    - Každý řádek tabulky obsahuje data o jednom konkrétním záznamu.
  - **Sloupce:** Každá tabulka obsahuje sloupce, které představují jednotlivé atributy dat. Například, v tabulce zákazníků by sloupce mohly zahrnovat jméno, adresu, e-mail atd.

- **Klíče:** Klíče jsou klíčovým prvkem relačních databází. Primární klíč jednoznačně identifikuje každý záznam v tabulce.
  - Cizí klíče vytvářejí vazby mezi tabulkami, umožňující propojení dat.
- **Integrita dat:** Relační databáze často používají různé omezení, jako jsou unikátní klíče a omezení cizích klíčů, aby zajistily konzistenci a integritu dat.
- **SQL (Structured Query Language):** Pro manipulaci s daty v relačních databázích se běžně používá SQL.
  - SQL poskytuje standardizované dotazy pro vytváření, čtení, aktualizaci a mazání dat v databázi.
- **Normalizace:** Proces normalizace se používá k organizaci dat tak, aby byla minimalizována redundance a zajištěna integrita databáze.
  - Normalizace rozděluje tabulky na menší tabulky tak, aby byly data udržována v konzistentním stavu.

- Návrh tabulky - nejdříve znázorníme data pomocí ER diagramu
  - vytvoříme databázi příkazem - create database nazevDatabase
  - nebo vybrat existující nátev - use nazevDatabase
  - vytvoříme tabulku - create table nazevTabulky(atributy);
  - úprava tabulky - alter table(add / drop column)
  - odstraníme ji -> drop table nazevTabulky
  - v tabulce hledáme data - select * from nazevTabulky(where name = "Pepa") nebo (where name like "K%", "K_ _ _ _ _")
  - informace vložíme - insert into nazevTabulky(nazevSloupce) values(hodnota)
  - řádky = relace
  - sloupce = atributy
  - primární klíč
  - dočasná tabulka - create tepmorary table nazevtavulky()
  - tabulka by měla splňovat normální formy
    - 1NF - tabulky jde dosadit jednoduchý datový typ - nedá se dále dělit(nesmí být relace)
    - 2NF - splnění 1NF plus každý atribut každý atribut, který není primárním klíčem je závislý na primary keya na jeho podmnožině
    - 3NF - spnění 2NF, žádný atribut, který není primary key, není tranzitivně závislý na žádném klíči
- CRUD operace
  1. Create - operace umožňuje vytvořit nový záznam nebo entitu v databázi, nová data jsou vkládána do systému a jsou přidělována novým identifikátorům
  2. Read - Čtecí operace umožňují získávat informace z databáze, umožňuje vyhledávat, filtrovat a zobrazovat data uložená v systému
  3. Update - operace umožňuje aktualizovat existující záznamy v databázi, data, která již existují, jsou modifikována na základě nových informací
  4. Delete - umožňuje odstranit záznamy z databáze, smazání může být prováděno na základě určitých podmínek nebo pomocí identifikátorů
- primární klíč - identifikuje jednu věc, v celé tabulce nemůže být 2x
- cizí klíč(foreign key) - klíč, který nám vnaší tabulce odkazuje na jinou tabulku, jeho hodnota odpovídá hodnotě primárního klíče v odkazované tabulce

19. SQL – spojení tabulek, poddotazy, agregační funkce a skupiny (GROUP BY, HAVING, aj.)

- Spojení tabulek - využíváme v chvíli kdy chceme záznamy např. z dvou tabulek
  - Natural join - přirozené spojení
  - Cross join - křížové
  - Left, right join - vnější
  - Union koin - sjednocením
- Poddotazy - select avg(plat) from platy -> vrátí průměrné platy
  - přihlášení do mysql -> mysql.exe -h hostid(ip adresa) -u username -p test1234
- Agregační fce - agregační fce se často používají -> často je potřeba něco sečíst, zprůměrovat
  - avg() - vratí průměr
  - count() - vrátí počet hodnot
  - max() - vrátí nejvyšší hodnotu

    - min() - vrátí nejnižší hodnotu
    - sum() - vrátí součet všech číselných hodnot
- Skupiny
    - Group by - používá se např. získat data ze 2 řádků z jedné tabulku a vytvořit skupinu
    - Having - používá se k zadání podmínky na základě agregační fce např. k omezení řádků
        - přes where se to řešit nedá -> nepodporuje agregační fce
        - musí být uvedeno za *group by*

20. Skriptovací jazyky – PHP

- skriptovací jazyk - přesná definice neexistuje ale jsou to programovací jazyk, který je interpretovaný, nevyžaduje deklarování proměnných, používá dynamickou typovou kontrolu, Podporuje práci se složitějšími datovými typy jako jsou seznamy a asociativní pole, bez potřeby starat se o uvolňování paměti
- je určný k manipulaci s prostředky daného systému, dokáže se přizpůsobit k potřebám uživatele
- nejčastěji se používají pro vývoj webových aplikací a dynamických stánek
- řadíme mezi ně jazyky PHP, Python, Type/Javascript, Ruby, Go
- dynamická stránka - stránka, jenž je schopna změnit svůj obsah v závislosti na různé situace - načtení z databáze, přihlášení uživatele
- dynamický web je realizován skriptovacím jazykem vkládaným mezi HTML tagy
- př. dynamického webu - blog, eshop
- **PHP**
    - používá se na straně serveru, vyvinul ho Rasmus Lerdorf roku 1995, syntaxe obdobná C/C++, nabízí OOP
    - momentálně ve verzi 8.x.x
    - nejrozšířenějčší skriptovací jazyk pro web, až 80% je v něm napsána
    - funguje tak, že stránka je poslána na server, kde se vykoná PHP skript a výsledek je potom poslán zpět userovi
    - PHP oddělujeme tagy <?php a ukončujeme ?>
    - podporuje přístup k databázím(MySQL, Oracle, SQLite, PostgreSQL)
    - echo - příkaz pro vypsání čehokoli do stránky, třeba dotaz který jsme poslali do databáze
    - proměnné se deklarují dolarem($) před názvem a název, deklarování typu není povinně
    - pomocí API můžeme posílat data do databáze a zpět nebo požadavky na server pomocí metod:
        - GET - data se stanou součástí URL adresy a jsou vidět v adresním řádku
        - POST - data se posídají odděleně od URL adresy