PJV

6. Programování v jazyce JAVA: vlastnosti a koncepce jazyka. Principy objektového programování.

Vývojové prostředí (JDK), JVM. Kompilace a běh programu.

- JDK (Java Development Kit) – základní vývojové prostředí, knihovny funkcí, překladač zdrojových souborů `javac`. Jeho součástí je i JRE.
- JRE (Java Runtime Environment) – základ prostředí Java pro spouštění programů, obsahuje virtuální stroj java.
- JVM (Java Virtual Machine) – interpreter java byte code, který za běhu java byte code kompiluje, optimalizuje a hlavně převádí na instrukce, virtuální stroj pro spouštění Java programů (java).
- kompilace pomocí javac, vstup `.java` výstup `.class` - na platformě nezávislý byte code pro JVM
- JAR (Java ARchive) – archív Java souborů, typicky množiny zkompilovaných `.class` (tříd) doplněných textovým popisem (Manifest), kterou třídu spustit. Slouží pro snadnější spouštění programů o více souborech.

Správa paměti, GC. Profilování a optimalizace.

- Heap
  - Společný paměťový prostor
  - Ukládání objektů a dat
  - Instanční proměnné
  - Spravován GC
- Stack
  - Dočasný paměťový prostor
  - Lokální proměnné, reference a argumenty funkce
  - Pro každé volání metody vlastní stack, který je po návratu uvolněn jako celek
- GC
  - jakmile na objekt už neukazuje reference garbage collector ho sežere a uvolní místo na haldě
  - automaticky běží na pozadí
  - 1. Nejdříve jsou objekty marknuty k likvidaci
  - 2. Pak jsou uvolněny všechny označené objekty
  - 3. Zkompaktnění heapu - odstranění mezer mezi objekty
  - Na Heap uloženy objekty do generací podle životnosti a postupně stárnou (když se naplní jejich generace) a přesouvají se do starších generací, což urychluje zkompaktnění
  - paralelní - vlastní, vlákno, nízká latence
- Profilování
  - vytížení procesoru, využití paměti, počet vláken, GC
  - výpis na příkazovou řádku stav haldy
  - escape analysis - ovlivňuje zda bude objekt alokován na haldě, cílem udžet co nejvíce objektů jako no escape
    - No Escape - objekt není viditelný mimo aktuální metodu a vlákno
    - ArgEscape – objekt je viditelný pouze jako argument metody
    - GlobalEscape – objekt uniká z metody (návratový argument, statická proměnná, apod.)

Objekty, třídy a jejich vztahy, princip abstrakce a zapouzdření, modifikátory přístupu.

- **Třída** - Představuje abstraktní definici datové struktury. Popisuje, jaké vlastnosti daná struktura má a jakým způsobem na ní lze operovat (v Javě říkáme, že definuje metody). Třída může vycházet z jiné, nadřazené třídy a upřesňovat originální definici. Třída rovněž může implementovat návrh, v Javě zvaný interface, a poskytovat API pro určitý seznam operací předem navržených v interface. Třída sama nenese žádná data. (pomíjím teď statické vlastnosti a metody v Javě, to s obecným OOP zas tak nesouvisí)
- **Objekt** - Objekt je konkrétní datová reprezentace třídy. Objekt je představován blokem dat a označen typem (názvem třídy), kterou reprezentuje. Na objektu můžeme operovat pomocí metod definovaných třídou, podle které byl vytvořen.
- **Abstrakce** - koncepty (šablony) organizujeme do tříd, objekty jsou pak instance tříd
- **Zapouzdření** - Objekty mají svůj stav skrytý, poskytují svému okolí jen rozhraní, komunikace s ostatními objekty zasíláním zpráv (volání metod)
- **Přístupy:**
  - private - přístup ze třídy ve které byly deklarovány
  - protected - přístup ze třídy a z odvozených tříd
  - public - přístup z libovolné třídy

- bez - přístup povolen v rámci stejného balíčku package

Interface a abstraktní třída.

- **Interface** - Rozhraní definuje množinu metod, které třída musí implementovat, pokud implementuje (`implemets`) dané rozhraní, Třída může implementovat více rozhraní
  - Vhodné pro případy - Očekáváme, že rozhraní bude implementováno v jiných, nesouvisejících třídách, Chceme využít vícenásobnou dědičnost, Chceme specifikovat chování konkrétního datového typu, bez ohledu na konkrétní implementaci chování
- **Abstraktní třída** - klíčové slovo `abstract` , Abstraktní třída umožňuje deklarovat abstraktní metody, Abstraktní metody nemají implementaci a je nutné je definovat v odvozených třídách, Lze je využít například pro vytvoření společného předka hierarchie tříd, které mají mít společné vlastnost
  - Vhodné pro případy - Odvozené třídy sdílejí implementaci, Odvozené třídy vyžadují přístup na položky, které nejsou public

Dědičnost a kompozice, polymorfismus, dynamická vazba.

- **Polymorfismus** - Vlastnost, která nám umožňuje pojmenovat nějakou konkrétní schopnost (metodu) identickým jménem, přičemž její implementace se může v jednotlivých třídách hierarchie tříd lišit
  - realizace pomocí - dědičnosti, virtuální metody, rozhraní, abstraktní třídy, překrývání metod `override`
- **Dědičnost** - základ polymorfismu, Hierarchie tříd (konceptů) se společnými (obecnými) vlastnostmi, které se dále specializují, `extends` , příznak "is-a"
- **Kompozice** - V případě kompozice existence vnitřního objektu nemá smysl bez objektu vnějšího. Třeba pokud by došlo k zániku šachovnice, pak už je šachová figurka zbytečná, příznak "has-a"
- **Agregace** - Naproti tomu agregace znamená volnější vztah. Pokud by zaniklo město, tak to neznamená, že všechny automobily musí také nutně zaniknout, jelikož si na ně může držet referenci třeba firma, majitel atp.
- **Dynamická vazba** - určuje volanou metodu na objektu až v době výpočtu (například v kódu je definován jen parent type s nějakou funkcí a za runtime je funkce volána na child), na základě objektů, které jsou zpracovávány, pomalejší než statická

Výčtové typy, práce s kolekcemi, vzor iterátor, generické typy

- Výčtové typy - jsou speciální třídy zavedené pro větší bezpečí a pohodlí, typově bezpečné, lze získat jak hodnotu, tak jméno, lze použít v switch
  - `public enum Suit {CLUBS, DIAMONDS, HEARTS, SPADES}`
- Kolekce (Java Collection Framework) - Množina třídy a rozhraní implementující sadu obecných a znovupoužitelných datových struktur, množiny, fronty, mapy, tabulky, Umožňuje nejen ukládání objektů, získávání a jejich zpracování, ale také výpočet souhrnných údajů apod.
- Iterátor - objekt pomocí, kterého lze bezpečně procházet a odstraňovat prvky z kolekce, `hasNext()` , `Next()` , `remove()`
  1. vytvoří se kopie kolekce - bezpečné před upravováním za iterace, ale O(n)
  2. přímé využití kolekce - není bezpečné, ale O(1)
- Generické typy - Generický typ umožňuje určit typ instance tříd, které lze do kolekce ukládat, Generický typ tak poskytuje statickou typovou kontrolu během překladu, <>

Vnitřní a anonymní třídy. Imutabilita, vzor singleton. Proměnné a metody třídy vs. instance.

- Vnitřní třída se definuje uvnitř těla jiné třídy, syntaxe je stejná jako u jakékoliv třídy:
- Vnitřní třída je dynamická (pokud ji neoznačíme klíčovým slovem static), v každé instanci vnější třídy se tedy vztahuje ke kontextu dané instance.

```
public class A {
        public class AHelp { }
}
```

- Anonymní třída - deklarace třídy v rámci výrazu, příkladem použití jsou pomocné třídy, například privátní HashMap.Node, nebo implementace callbacků (implementace Runnable, ActionListener, MouseListener etc.).
- Immutable - Definice neměnitelného objektu, Všechny datové položky jsou final a private, Zákaz přepisu metod v potomcích, průběhu života nemění svůj stav
- Singleton - private konstruktor, existuje jen jedna instance (static), getter na instanci
- Třídní proměnné (statické) - dostupné bez instance, neměnné
- Třídní metody (statické) - dostupné bez instance, nelze používat instanční atributy (jedině předat pomocí parametru)
- Instanční proměnné - každá instance objektu je může mít jiné
- Instanční metody - stejné jako třídní, ale mohou pracovat s instančními atributy

Mechanismus výjimek (typy a jejich ošetření, vlastní výjimky)

- Výjimka je „nestandardní situace“:

1. Situace, které jsou nestandardní, či které my považujeme za nestandardní, měli bychom reagovat a můžeme a dokážeme reagovat (RuntimeException)
    1. Pokus o čtení z prázdného zásobníku EmptyStackException
    2. Dělení nulou, indexování mimo rozsah pole, špatný formát čísel AritmeticException, NumberFormatException
2. Situace, na které musíme reagovat, Java nás přinutí (Exception, IOException)
    1. Odkaz na chybějící soubor FileNotFoundException
3. Chyba v hardware, závažné chyby, nemůžeme reagovat (Error), (OutOfMemoryError, UnknownError)
    1. Chyba v JVM
    2. HW chyba

- Vytvoření výjimky `public class MyException extends Exception {…}`, Kromě z Exception lze dědit také z Throwable, Error, RuntimeException. Tyto tři zmíněné lze zachytit pomocí try … catch bloku, ale není nutné je deklarovat pomocí throws.
    - Throwable - nemusí se zachytávat
    - Exception - musí se zachytávat
    - RuntimeException - nemusí se zachytávat
- Všechny podtypy Exception kromě podtypů RuntimeException je třeba ošetřit strukturou try … catch. Chybám typu RuntimeException by mělo jít zabránit správnou implementací programu (kontrola na null, kontrola správného stavu nějaké třídy, tak aby nevyhodila (legalStateException). Nicméně I RuntimeException lze odchytit pomocí try … catch. Ve skutečnosti lze odchytit všechny typy výjimek a chyb jejichž nejvyšším supertypem je Throwable.
- Výjimky typu Error jsou výjimky, které zpravidla zanechají program v nedefinovaném/nefunkčním stavu a zachytávat by se neměly. Pokud ano, pak by po jejich zachycení mělo následovat uložení do logu a ukončení programu.
- Měli bychom nejprve zachytit konkrétnější výjimky, až poté obecné. Například nejdříve IOException a pak teprve Exception.
- Vlastní výjimka je potomkem třídy Exception. Jedná se o tzv. synchronní výjimku, vzniká na přesně definovaném místě. Většinou se jedná o výjimku, na kterou chceme reakci uživatele. Reakci na vlastní výjimky systém vyžaduje.

```
        void mojeMetoda() throws MojeVyjimka { … }

        class MojeVyjimka extends Exception
        {
                int n;
                int d;
                MojeVyjimka(int i, int j) {
                        n = i;
                        d = j;
                }

                public String toString() {
                        return "Hodnota " + n + "/" + d +" není integer.";
                }
        }

        //priklad pouziti
        throw new MojeVyjimka(numer\[i\], denom\[i\]);
```

Práce se soubory (přístup k souboru, textové vs. binární, proudy, ukládání dat)

- Binární soubor zpravidla parsujeme byte po bytu. Obvykle se skládá z bloků dat s definovanou délkou. Textový soubor je třeba nejdříve převést na text v závislosti na použitém kódování (kterých jsou mraky) a poté jej parsujeme znak po znaku. ASCII textové soubory lze rovněž rovnou číst byte po bytu, jelikož v ASCII neexistují vícebytové znaky.
- Přímý přístup představuje čtení z předem zadaných lokací v souboru (např. Od 100 bytu do 120 bytu). Toto praktikujeme zejména u binárních souborů. Sekvenční přístup spočívá ve čtení souboru od začátku a hledání dat, která chceme. Takhle parsujeme třeba XML, JSON či Javou serializované soubory.
- Soubor je množina údajů uložená ve vnější paměti počítače
- Proud je přístup (nástroj) k přenosu informaci z/do souboru, ale také z/do libovolného jiného média, které je schopné generovat nebo pojmout data jako posloupnost bytů - sítě, sériová linka, paměť, jiný program, atd.
- Proudy - bytové (FileInputStream/FileOutputStream, ObjectOutputStream), Znakové (FileReader/FileWriter)
- File třída - zpracování souborů/adresářů: test existence, oddělovač adresářů/souborů, vytvoření, mazání, atd.

Sokety (typy soketů, typy spojení, síťová komunikace).

- Sériová komunikace – jedná se o obousměrnou komunikaci po dvou kanálech, z nichž každý je jednosměrný. Komunikace je asynchronní (kanály na sobě nejsou závislé).
- TCP socketová komunikace – komunikace se navazuje mezi zařízeními pomocí síťové IP adresy a portu. Je o dost komplexnější než sériová, disponuje automatickou opravou chyb (chybně přijatá data nemusíte řešit – buď přijdou správná data, nebo se socket odpojil a nepřijde nic.
- UDP komunikace – od socketové se liší tím, že nemá zpětnou vazbu ani opravu chyb. To z ní dělá komunikaci méně spolehlivou, ale nesrovnatelně rychlejší. Je rovněž méně bezpečná.
- Pipe – komunikace v rámci jednoho zařízení poskytovaná operačním systémem. Umožňuje posílat data mezi běžícími programy. Je na ní založen příkazový řádek.
- Síťová komunikace - slouží k přenosu informace pomocí výměny zpráv, jasná pravidla
  - zahájení komunikace, předání zprávy, reakce na zprávu, ukončení komunikace.
- Protokol - definuje: formát zpráv, pořadí výměny zpráv, syntaxi zpráv, sémantiku zpráv, chování při přijmu a vyslání zprávy
- Server - reprezentuje služby
- Klient - reprezentuje poptávku po službě
- klient/server - klient žádá o službu server
- peer-to-peer (P2P) - každý účastník vystupuje jako klient i jako server
- spojovaná - Obě strany nejdříve navazují spojení
- nespojovaná - nenavazují spojení
- Soket - Soket je objekt, který propojuje aplikaci s nějakým „síťovým” protokolem, connect(), bind(), close()

Paralelismus, vícevláknové aplikace, problém souběhu a zastavení.

- paralelismus - vyšší výpočetní výkon, efektivní využívání strojového času, zpracování více požadavků najednou
- Vlákno je samostatně prováděný výpočetní tok
- Proces je spuštěný program ve vyhrazeném prostoru paměti, executing - běží, blocked - čeká na periferie, waiting - čeká na procesor, má PID
- Aplikace jsou interaktivnější

Proces	**Vlákno procesu**
výpočetní tok	výpočetní tok
vlastním paměťovém prostoru	společném paměťovém prostoru
Synchronizace entitami OS	Synchronizace exkluzivním přístupem k proměnným
Přidělení CPU, rozvrhovačem OS	Přidělení CPU, v rámci časového kvanta procesu
Časově náročnější	Méně časově náročné

- jenom jeden procesor - stále má smysl paralelizovat protože, operační systém přepíná sám mezi vlákny a simuluje chování víceprocesorového systému
- Kdy má smysl použít paralelismus:
  - Je možné provádět paralelně více relativně náročných úloh na více procesorovém systému (kompilace například).
  - Aplikace provádí blokující IO operace (čtení velkého souboru, síťová komunikace), ale zároveň má GUI, které musí zůstat responzivní.
  - Aplikace něco kontroluje na pozadí (daemon thread) v pravidelných intervalech, zatímco na popředí s ní uživatel normálně interaguje.
- Problémy paralelismu:
  - Zdroj/konzument - Problém dvou vláken, kdy jedno vlákno data produkuje a zařazuje do fronty, druhé data z fronty odebírá a zpracovává. Je třeba zabránit cpaní dat do plné fronty a tahání dat z prázdné fronty.
  - Čtení/zápis - Problém, kdy mutexem rezervujeme proměnnou pro jedno vlákno, i když obě vlákna se z proměnné snaží pouze číst. Je ale pořád třeba zabránit kombinaci zápis+čtení.
  - Obědvající filozofové (deadlock) - Problém spočívá ve zdrojích sdílených vlákny. Ve chvíli kdy každé vlákno (ze dvou) disponuje polovinou potřebných zdrojů a čeká na tu druhou, program nikdy neskončí. Řeší se zpravidla nadřazeným algoritmem pro rozdělování zdrojů.
  - Problém souběhu (race condition) - dochází pokud dvě vlákna začnou měnit stejná data v paměti ve stejnou chvíli. Výsledná hodnota paměti je nedefinovaná. Souběhu se v Javě předchází synchronized blokem a klíčovým slovem volatile.

Tvorba vláken a jejich ukončení, threadpool, synchronizace, volatilita.

- extends Thread, implements Runnable
- `public void run()`
- start() metodou se vlákno spustí
- ukončení vlákna pomocí .join()
- pokud proměnnou může měnit více vláken - keyword volatile
- kritická sekce - keyword synchronized, Vstup do kritické sekce je umožněn pouze jedinému vláknu
- wait - vlákno čeká na notify

- threadpool - místo vytváření hodně vláken, je vytvořen seznam vláken, které jsou znovupoužity pro určené úkoly, lze tím limitovat počet vláken, aby nepřehltili systém, spravuje je thread executor