Maturitní témata 2020/21

Otázka č. 26 – Počítačové sítě a programování

OOP:

- specifické programovací paradigma (základní programovací styl, liší se v pojmech a abstrakcích), které ho odlišilo od původního imperativního (procedurálnímu), zdrojový kód je v OOP přidružen k datům (metody jsou zapouzdřeny v objektech), což umožňuje snadnější přenos kódu mezi různými projekty (abstrakce a zapouzdření), propojení umožnilo zavést dědičnost ale kvůli zjednodušení si vyžádalo zavedení polymorfismu

Rozdíly oproti procedurálnímu (imperativnímu) paradigmatu:

- u procedurálního programování je program jeden celek maximálně rozdělený do procedur pro zpřehlednění a opakované použití

- u OOP je program tvořený víceméně samostatnými objekty, cílem OOP je vytvářet malé relativně samostatné a v různých aplikacích opakovaně použitelné jednotky (objekty

- u OOP se jako výhody uvádějí snazší a rychlejší vývoj aplikací, snazší údržba a menší chybovost, nevýhodou je že správa objektů zabere určité systémové prostředky → dokonale napsaný objektový program bude pomalejší než dokonale napsaný procedurální program (prakticky žádná aplikace nebude dokonale napsána, díky OOP se k dokonalosti lze snáze přiblížit)

Návrh objektů:

- třídy slouží jako plán pro tvorbu objektů (atribut objektu je definovaný ve třídě objektu)

- objekt vznikne naplněním konkrétních hodnot do vlastností třídy

- každý objekt lze popsat mnoha způsoby:

  - slovy
  - diagramem (diagram tříd jazyka UML)
  - zápisem v programovacím jazyku

- každý objekt musí patřit k určité třídě, nelze vytvořit objekt aniž by nebylo definováno jak má vypadat

- ze třídy může být vytvořen libovolný počet objektů

Zapouzdření:

- jeden ze tří pilířů OOP (Zapouzdření, Dědičnost a Polymorfismus), umožňuje skrýt některé metody a atributy tak aby zůstaly použitelné jen pro třídu zevnitř, objekt si můžeme představit jako černou skříňku která má určité rozhraní (interface), přes které ji předáváme instrukce/data o ona je zpracovává

- nevíme jak to uvnitř funguje ale víme jak se navenek chová a používá, nemůžeme tedy způsobit nějakou chybu protože využíváme a vidíme jen to co tvůrce třídy zpřístupnil

- např. u třídy člověk by mohl být problém ve chvíli když by měla mít atribut $datumNarozeni a na jeho základě další atributy: $plnolety a $vek, kdyby někdo objektu zvenčí změnil $datumNarozeni přestaly by platit proměnné plnolety a vek (vnitřní stav objektu by byl nekonzistentní), toto se nám může ve strukturovaném programování stát, v OOP ale objekt zapouzdříme a atribut $datumNarozeni označíme jako privátní, zvenčí tedy nebude viditelný, naopak ven vystavíme metodu zmenDatumNarozeni() která dosadí nové datum narození do proměnné datumNarozeni a zároveň provede potřebný přepočet věku a přehodnocení plnoletosti, použití objektu je bezpečné a aplikace stabilní

Polymorfismus:

- jeden z pilířů OOP, umožňuje používat jednotné rozhraní pro práci s různými typy objektů, vlastnost programovacího jazyka

- umožňuje:

- jednomu objektu volat jednu metodu s různými parametry
- objektům odvozeným z různých tříd volat tutéž metodu se stejným významem v kontextu jejich třídy často pomocí rozhraní
- přetěžování operátorů neboli provedení rozdílné operace v závislosti na typu operandů
- jedné funkci dovolit pracovat s argumenty různých typů

- např. máme mnoho objektů které reprezentují nějaké geometrické útvary (kruh, čtverec, trojúhelník), abychom s nimi komunikovali jednotně ačkoli se liší můžeme zavést třídu GeometrickyUtvar která by obsahovala atribut barva a metodu Vykresli(), všechny geometrické tvary by potom dědily z této třídy její interface (rozhraní), objekty kruh a čtverec se vykreslují jinak, polymorfismus nám umožňuje přepsat si metodu Vykresli() u každé podtřídy tak aby dělala co chceme, rozhraní tak zůstane zachováno a my nebudeme muset přemýšlet jak se to u onoho objektu volá

- podstatou polymorfismu je tedy metoda nebo metody které mají všichni potomci definované se stejnou hlavičkou ale jiným tělem