6. Objektově orientované programování

Základy objektového programování (zapouzdření, dědičnost, polymorfismus)

Objektově orientované programování (OOP) je přístup k návrhu softwaru, kde je systém rozdělen do objektů. Každý objekt v sobě kombinuje data (atributy) a funkce (metody), které s těmito daty pracují. OOP umožňuje lepší organizaci kódu, opakované využití a snadnější údržbu složitých aplikací.

Zapouzdření (Encapsulation)

Zapouzdření znamená skrytí vnitřní implementace objektu před okolím a zpřístupnění pouze jasně definovaného rozhraní. Data objektu jsou obvykle privátní (private) a jsou přístupná či měnitelná jen prostřednictvím veřejných metod (gettery/settery). Tím se zvyšuje bezpečnost a modularita – interní změny neovlivní ostatní části systému.

Dědičnost (Inheritance)

Dědičnost umožňuje, aby nová třída (potomek) převzala (zdědila) vlastnosti a metody jiné třídy (rodiče). Potomek může chování předka rozšiřovat nebo upravovat. To podporuje opětovné použití kódu – například všechny třídy „Zvíře“ mohou dědit ze společné třídy „Organismus“.

Polymorfismus (Polymorphism)

Polymorfismus znamená, že objekty různých tříd mohou sdílet stejné rozhraní (například metodu `vypiš()`), ale chovat se odlišně podle svého konkrétního typu. Polymorfismus může být statický (přetížení metod – stejný název, různé parametry) nebo dynamický (přepsání metod – potomek poskytuje vlastní implementaci metody z předka).

Virtuální stroj (.NET Framework)

Virtuální stroj je softwarová vrstva, která umožňuje spouštění programů nezávisle na konkrétním hardwaru nebo operačním systému. V případě .NET Frameworku se programy nejprve přeloží do mezijazyka CIL (Common Intermediate Language) a ten je následně vykonáván tzv. CLR (Common Language Runtime), což je právě virtuální stroj.

Hlavní výhody tohoto přístupu jsou:

- **Přenositelnost:** Kód lze spustit na různých platformách bez úprav.
- **Bezpečnost:** CLR zajišťuje správu paměti a chrání před neautorizovaným přístupem.
- **Interoperabilita:** .NET Framework podporuje více jazyků (např. C#, Visual Basic .NET, F#), které se všechny překládají do CIL.
- **Bohatá knihovna tříd:** Vývojáři mohou využívat široké spektrum funkcí bez nutnosti vše psát od začátku.

Ukládání dat v paměti (halda, zásobník)

Při běhu programu se data ukládají do dvou hlavních částí paměti:

Zásobník (Stack)

Zásobník je rychlá paměťová oblast, kde se ukládají lokální proměnné, parametry funkcí a návratové adresy. Pracuje metodou LIFO (Last In, First Out), což umožňuje velmi rychlé přidávání a odebírání dat. Životnost dat na zásobníku je omezena na dobu trvání funkce/metody.

Halda (Heap)

Halda je paměťová oblast určená pro dynamickou alokaci objektů, které mohou žít nezávisle na rámci funkcí. Objekty na haldě vytváříme většinou pomocí klíčového slova `new`. Přístup k datům na haldě je pomalejší a jejich životnost řídí správce paměti (garbage collector).

Garbage Collector

Garbage Collector (GC) je automatický mechanismus pro správu paměti v prostředí .NET Framework (a podobných virtuálních strojích). Jeho hlavní úlohou je sledovat, které objekty na haldě již nejsou využívané (tj. neexistuje na ně žádná reference), a tuto paměť automaticky uvolnit.

Výhoda:
Minimalizuje riziko „memory leaků“ (úniků paměti) a zjednodušuje správu paměti.

Nevýhoda:
Nelze přesně ovlivnit, kdy bude paměť uvolněna (GC není deterministický), což může v některých případech způsobit krátké pauzy v běhu programu.

Moderní garbage collectory (např. v .NET) používají tzv. „generační“ přístup – rozdělují objekty do několika generací podle délky života, což zefektivňuje samotný proces sběru a minimalizuje výkonnostní dopady na běžící aplikaci.