1. Zapouzdření

# Zapouzdření v souvislosti s Modifikátory přístupu

* Umožňuje skrýt detaily implementace objektu a nabídnout pouze určené rozhraní pro manipulaci s objektem
* To zajišťuje, že objekt bude používán správným způsobem a snižuje pravděpodobnost chyb při manipulaci s objektem
* Pro implementaci zapouzdření se používají modifikátory přístupu, jako jsou "private", "protected" a "public". Tyto modifikátory určují viditelnost daného prvku pro jiné části programu
* Modifikátor "private" označuje prvky jako viditelné pouze uvnitř dané třídy
* Modifikátor "protected" označuje prvky jako viditelné pouze uvnitř dané třídy a potomků této třídy
* Modifikátor "public" označuje prvky jako viditelné pro všechny části programu

# Zapouzdření v souvislosti s Algoritmy, Datovými strukturami a abstraktními datovými typy

* V souvislosti s algoritmy
  + Používá se k oddělení algoritmu od datových struktur
  + To znamená, že algoritmus nemusí znát detaily implementace datové struktury, ale pouze používá rozhraní, které tato datová struktura nabízí
  + To umožňuje změnit implementaci datové struktury, aniž by to mělo vliv na algoritmus
* V souvislosti s datovými strukturami
  + Umožňuje skrýt detaily implementace datové struktury a nabídnout pouze rozhraní pro manipulaci s daty
  + To znamená, že uživatelé datové struktury mohou manipulovat s daty pouze pomocí definovaných metod a nemohou přistupovat k datům přímo
  + Tímto způsobem se zajišťuje, že data budou vždy v konzistentním stavu a snižuje se pravděpodobnost chyb při manipulaci s daty
* V souvislosti s abstraktními datovými typy
  + Používá se k oddělení abstraktního datového typu od jeho implementace
  + To umožňuje uživatelům abstraktního datového typu používat pouze definované metody, aniž by se museli zabývat implementačními detaily
  + Toto umožňuje snadněji změnit implementaci abstraktního datového typu, aniž by to mělo vliv na uživatele tohoto datového typu

# Bezpečnostní význam zapouzdření

* Díky zapouzdření mohou být citlivá data a operace s nimi skryta před neoprávněnými uživateli a chráněna tak před útoky, které by mohly vést ke ztrátě dat, porušení integrity dat nebo přístupu neoprávněných osob k datům
* Zapouzdření také umožňuje snadnější správu programu a jeho údržbu

# Metody pro nastavení/získání zapouzdřených dat – implementace

* Gettery
  + Slouží k získání hodnoty zapouzdřeného datového atributu
  + Tyto metody mají obvykle název, který odpovídá názvu datového atributu, a jejich návratovou hodnotou je hodnota datového atributu

Obsah obrázku text, interiér, ve tmě

Popis byl vytvořen automaticky

* Settery
  + Slouží k nastavení hodnoty zapouzdřeného datového atributu
  + Tyto metody obvykle mají název, který začíná slovem "set", následovaným názvem datového atributu, a jako argument přijímají novou hodnotu pro tento atribut

Obsah obrázku text

Popis byl vytvořen automaticky

* Použití getterů a setterů umožňuje zapouzdřit data a skrýt je před uživatelem třídy. Uživatel třídy tak nemůže přistupovat k datům přímo, ale pouze pomocí definovaných metod. Tímto způsobem se minimalizuje riziko nesprávné manipulace s daty a zvyšuje se bezpečnost a správnost programu

# Zapouzdření a modifikátory přístupu v souvislosti s dědičností

* V dědičnosti jsou třídy organizovány do hierarchie, kde jsou třídy potomky jiných tříd (rodičů). Potomci dědí vlastnosti a metody svých rodičů a mohou přidávat nebo přepisovat další metody a vlastnosti
* V kontextu zapouzdření znamená, že potomci mohou mít přístup ke všem vlastnostem a metodám svých rodičů, které jsou public nebo protected
* Modifikátory přístupu v tomto kontextu určují, zda jsou vlastnosti a metody děděny potomkům a zda mohou být potomci tyto vlastnosti a metody přistupovat nebo je přepisovat

# Prakticky:

* Připravená třída, pomocí zapouzdření doplnit metody pro práci s atributy, nastavení přístupu atributů