Abstraktní datové typy

* Je to typ dat, které jsou nezávislé na vlastní implementaci
* Cílem je zjednodušit a zpřehlednit program, který provádí operace s daným datovým typem
* Umožňuje vytvářet i složitější datové typy (String, Object atd.)
* Všechny abstraktní datové typy lze realizovat pomocí algoritmických operací (přiřazení, sčítání, podmínky atd)
* Příklady: Stack, fronta, tree, hashmap, linkedlist
* ADT jsou reprezentovány rozhraním, skrývají vlastní implementaci. Důležité je použití, ne kód za ním
* Robustnost spočívá v tom, že programátor má přístup pouze k ovládání

## Vlastnosti abstraktního datového typu

* Pokud je ADT programované objektově, tak je většina splněna
  1. Všeobecná implementace
     + Jednou vytvořený ADT lze využít v jakémkoliv programu
  2. Přesný popis
     + Propojení mezi implementací a rozhraním musí být jednoznačné
  3. Jednoduchost
     + Uživatel ADT se nemusí starat o realizaci a správu paměti
  4. Zapouzdření
     + ADT má být uzavřený celek. Uživatel má vědět co dělá, ne jak to dělá.
* Integrita
  + - Uživatel nemůže sahat do vnitřní struktury dat. Výrazně se sníží riziko nechtěného smazání
  1. Modularita
     + Princip je přehledný a lze některou část snadno vyměnit. Je rozdělen do kompaktních celků

## Typy operací

1. Konstruktor
   1. Ze zadaných parametrů vytváří novou hodnotu ADT. (Prostě konstruktor s parametrem velikosti pole)
   2. Sestavuje vnitřní reprezentaci hodnot dle parametrů (nastaví velikost pole kam se to ukládá)
2. Selektor
   1. Získává hodnoty, které tvoří vlastnosti daného ADT (metoda get(parametr) do našeho ADT
3. Modifikátor
   1. Provádí změny hodnoty ADT
   2. (Vnitřní implementace jak funguje ADT)

## Zásobník (STACK)

* Používán pro dočasné ukládání dat
* Funguje podle LIFO (Last in first out)
* Funguje zde ukazatel zásobníku (vrchol zásobníku) = pole.length
* Příklad použití:
  1. V procesoru
     + Ukládá návratové adresy a stav procesoru při přerušení a skoků do podprogramu
* **MUSÍ OBSAHOVAT:**
  1. CREATE (inicializace zásobníku, může být v PUSH)
  2. PUSH (přidání položky na vrchol)
  3. POP (odebrání položky z vrchu, prostě get)
  4. TOP (dotaz na vrchol zásobníku)
  5. IS\_EMPTY (dotaz na prázdnost, může být součástí POP)

Obsah obrázku text

Popis byl vytvořen automaticky

Metody zásobníku 1

## Fronta (QUEUE)

* Používána pro meziprocesorovou komunikaci (výměna dat mezi více thready)
* Funguje na principu FIFO
* Opakem je Zásobník
* **MUSÍ OBSAHOVAT:**

1. CREATE (inicializace)
2. ENQUEUE (vložení položky na konec)
3. DEQUEUE (vybrání položky ze začátku fronty a odebrání)
4. PEEK (získání položky ze začátku bez jejího odebrání)
5. Obsah obrázku text

   Popis byl vytvořen automatickyIS\_EMPTY (dotaz, zda je fronta prázdná)

Obsah obrázku text

Popis byl vytvořen automaticky

## Tree (STROM)

* Prvky stromu
  + Kořen stromu
    - Nejvyšší uzel (root)
    - Jediný uzel bez rodiče, maximálně 1
  + Vnitřní uzly
    - Uzel, který není koncový ale ani ne root
    - Má potomka
  + Koncový uzel
    - Prvek, který nemá potomka
    - Pokud má strom jen 1 uzel, tak je to root i koncový
* Obsah obrázku text

  Popis byl vytvořen automaticky

Obsah obrázku text

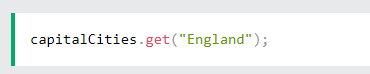
Popis byl vytvořen automaticky

## HashTable (HASHOVACI TABULKA)

* Vyhledávací datová struktura
* (Místo hashmap si predstav hashtable, spletl jsem si to)
* Asociuje hashovací klíče s odpovídajícími hodnotami
  + Klíč se spočítá z hodnoty
* Může dojít ke kolizi
  + Záznamy s různými klíči hashují na stejné místo

Obsah obrázku text

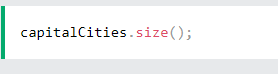
Popis byl vytvořen automaticky

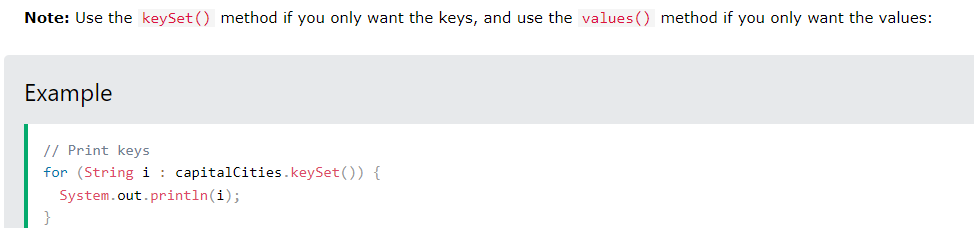




Obsah obrázku text

Popis byl vytvořen automaticky





## LinkedList

* Funguje tak, že znám někoho, kdo ví to co žádáš
* Je to seznam podobný poli, kde hodnoty mají stejný typ a obsahují referenci
* Jednosměrný nebo obousměrný (odkazuje buď jen na následující nebo i na předchozí)

Obsah obrázku text

Popis byl vytvořen automaticky

Obsah obrázku stůl

Popis byl vytvořen automaticky

**Dlužíte mi každý 50, nebo 2x párky v rohlíku**