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

5. Modularita
    - Princip je přehledný a lze některou část snadno vyměnit. Je rozdělen do kompaktních celků

Typy operací

1. Konstruktor
    a. Ze zadaných parametrů vytváří novou hodnotu ADT. (Prostě konstruktor s parametrem velikosti pole)
    b. Sestavuje vnitřní reprezentaci hodnot dle parametrů (nastaví velikost pole kam se to ukládá)
2. Selektor

a. Získává hodnoty, které tvoří vlastnosti daného ADT (metoda get(parametr) do našeho ADT

3. Modifikátor
    a. Provádí změny hodnoty ADT
    b. (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)

```java
// Pushing element on the top of the stack
static void stack_push(Stack<Integer> stack)
{
    for(int i = 0; i < 5; i++)
    {
        stack.push(i);
    }
}

// Popping element from the top of the stack
static void stack_pop(Stack<Integer> stack)
{
    System.out.println("Pop Operation:");

    for(int i = 0; i < 5; i++)
    {
        Integer y = (Integer) stack.pop();
        System.out.println(y);
    }
}

// Displaying element on the top of the stack
static void stack_peek(Stack<Integer> stack)
{
    Integer element = (Integer) stack.peek();
    System.out.println("Element on stack top: " + element);
}
```

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. IS_EMPTY (dotaz, zda je fronta prázdná)

```java
public class Queue<T> {
    11 usages
    private T[] data;
    3 usages
    private int dataTop;

    1 usage
    public Queue(int size){
        this.data = (T[]) new Object[size];
        dataTop = 0;
    }

    3 usages
    public void enqueue(T item){
        T[] copy = (T[]) new Object[data.length];
        for (int i = 0; i < data.length; i++) {
            copy[i] = data[i];
            if (data[i] == null){
                copy[i] = item;
                break;
            }
        }
        data = copy;
        dataTop++;
    }

    2 usages
    public T dequeue(){
        if (!isEmpty()){
            T temp = data[0];
            for (int i = 0; i < data.length-1; i++) {
                data[i] = data[i + 1];
            }
            return temp;
        }
        return null;
    }
```

```java
3 usages
public boolean isEmpty(){
    if (dataTop == 0){
        return true;
    }
    return false;
}

1 usage
public T peek(){
    if (!isEmpty()){
        return data[0];
    }
    return null;
}
```

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
-

```java
public class NodeController {
    1 usage
    public void build(){
        Node RootNode = new Node( number: 25);
        System.out.println("Working...\n==========");


        insert(RootNode,  value: 70);
        insert(RootNode,  value: 5);
    }

    4 usages
    protected void insert(Node node, int value){
        if (value < node.value)
        {
            if (node.left != null)
            {
                insert(node.left, value);
            } else
            {
                System.out.println("  Inserted " + value + " to left of Node " + node.value);
                node.left = new Node(value);
            }
        }
        else if (value > node.value)
        {
            if (node.right != null)
            {
                insert(node.right, value);
            } else
            {
                System.out.println("  Inserted " + value + " to right of Node " + node.value);
                node.right = new Node(value);
            }
        }
    }
```

```java
class Node {
    5 usages
    int value;
    3 usages
    Node left;
    3 usages
    Node right;

    3 usages
    public Node(int number){
        this.value = number;
    }
}
```

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

```
public class Main {
  public static void main(String[] args) {
    // Create a HashMap object called capitalCities
    HashMap<String, String> capitalCities = new HashMap<String, String>();

    // Add keys and values (Country, City)
    capitalCities.put("England", "London");
    capitalCities.put("Germany", "Berlin");
    capitalCities.put("Norway", "Oslo");
    capitalCities.put("USA", "Washington DC");
    System.out.println(capitalCities);
  }
}
```

```
capitalCities.get("England");
```

```
capitalCities.remove("England");
```

To remove all items, use the `clear()` method:

Example

```
capitalCities.clear();
```

```
capitalCities.size();
```

Note: Use the `keySet()` method if you only want the keys, and use the `values()` method if you only want the values:

Example

```
// Print keys
for (String i : capitalCities.keySet()) {
  System.out.println(i);
}
```

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)

```
public class Main {
  public static void main(String[] args) {
    LinkedList<String> cars = new LinkedList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");
    System.out.println(cars);
  }
}
```

Method	Description
addFirst()	Adds an item to the beginning of the list.
addLast()	Add an item to the end of the list
removeFirst()	Remove an item from the beginning of the list.
removeLast()	Remove an item from the end of the list
getFirst()	Get the item at the beginning of the list
getLast()	Get the item at the end of the list

Dlužíte mi každý 50, nebo 2x párky v rohlíku