ABSTRAKTNÍ DATOVÉ TYPY (ADT)

hierarchie abstrakcí:

nejvyšší úroveň ZOO

↓

DruhZvirat

↓

celá čísla, řetězce

↓

nejnižší úroveň bity

Abstrahujeme od

- reprezentace (implementace) dat
- realizace (implementace) operací nad hodnotami

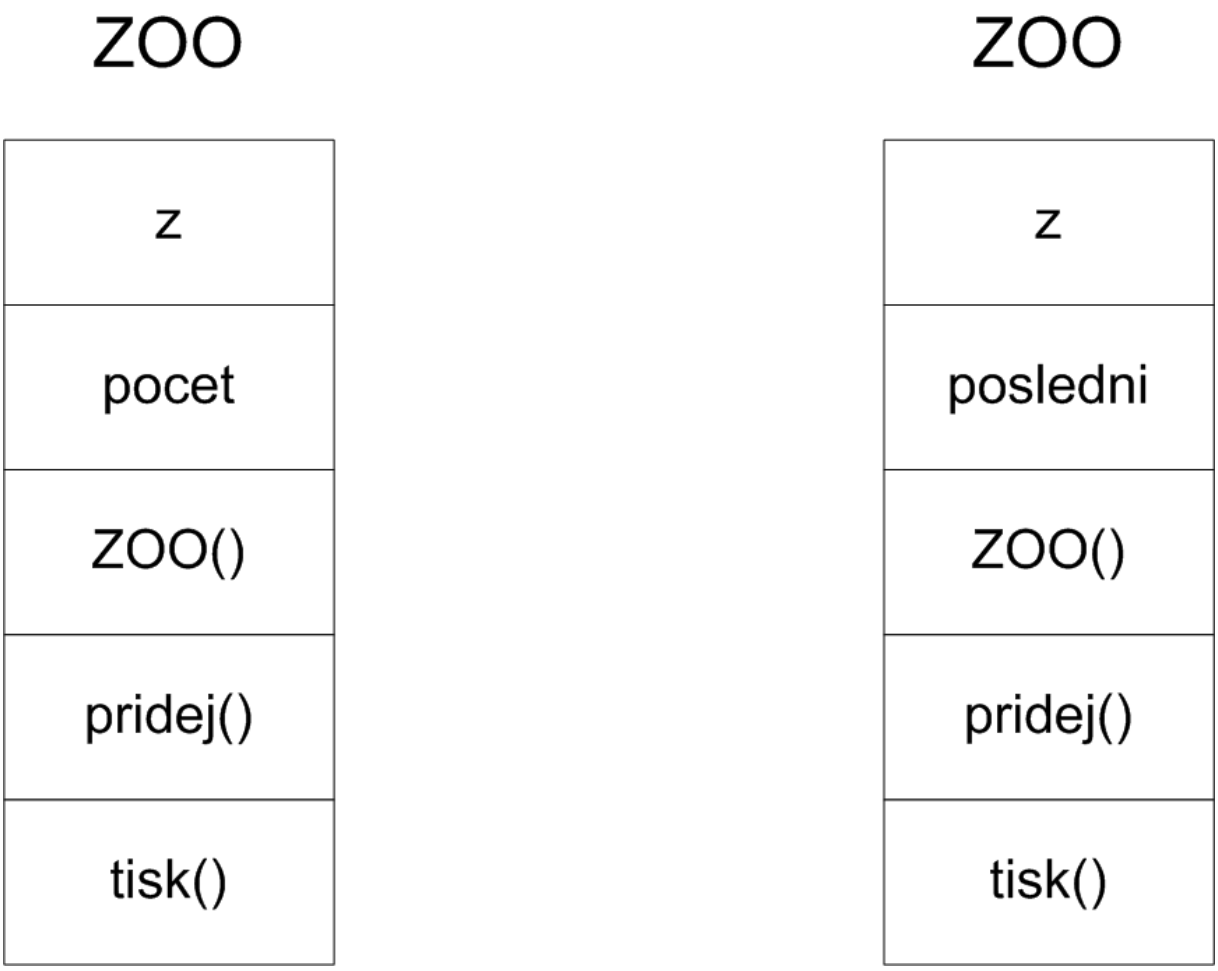
ZOO – *třída, pole, spojové struktury*

DruhZvirat - *třída*

celá čísla, řetězce - *kódování*

bity – *fyzikální jevy*

implementace (`ZOO.class`) -



klientský program (`ZOOHlavni.class`) –

pracuje s daty jenom metodami

`ZOO()`
`pridej()`
`tisk()`

rozhraní (interface) – deklarace používaných metod

smlouva mezi klientem a implementací:

Klient souhlasí s přístupem k datům pouze použitím metod definovaných rozhraním.

Implementace souhlasí s poskytnutím slíbených metod.

ADT je datový typ (množina hodnot a souhrn operací nad nimi) k jehož datům přistupujeme jenom přes rozhraní.

Program, který používá ADT - **klient**

Program, který je realizací ADT - **implementace**

Deklarace **rozhraní** - definice metod

signatura (definice) metody = typ návratové hodnoty, jméno metody a typ parametrů

V Javě získáme signaturu metody z její deklarace vynecháním jmen formálních parametrů v hlavičce

V jazyku Java zabráníme k přístupu členům třídy specifikátorem **`private`**

ZOO – ADT

hlavička:

```
  void pridej (String jaky, int kolik)
```

signatura:

```
  void pridej (String, int)
```

rozhraní:

```
class ZOO {
//ADT ZOO
  ZOO()
  void pridej (String, int)
  void tisk()
}
```

Kód klienta i implementace ADT jsou nezávislé !!!

1. Můžeme psát klienty před implementací ADT

2. Můžeme měnit implementaci ADT beze změny klienta

Různé implementace mohou mít různé vlastnosti

`String pridanXtyDruh(int)`

vrátí druh zvířat, který byl do ZOO přidán v pořadí daném parametrem typu **`int`**, tj. jako první, druhý, atd.

implementace polem

```
String pridanXtyDruh (int j) {
   if (j <= pocet )
     return z[j-1].druh;
   else
     return "neni tolik druhu";
 }
```

složitost

n – stávající počet druhů

$T(n) = c_{porovnání} + c_{návratu} = c$

$c \leq cf(n)$ pro $f(n) = 1$ (n^0) a $n_0 \geq 1$

metoda **`pridanXtyDruh`** v implementaci polem je O(1)

implementace spojovou strukturou

```
String pridanXtyDruh(int k) {
    DruhZvirat x = z;
    int i = 1;
    while (x != null) {
      if (i == k ) return x.druh;
      x = x.dalsi;
      i++;
    }
    return "neni tolik druhu";
}
```

složitost

n – stávající počet druhů

$T(n) = 2*c_{\text{přiřazení}} + 2*k*c_{\text{porovnání}} + 2*(k-1)*c_{\text{přiřazení}} + c_{\text{návratu}} = 2*k*(c_{\text{přiřazení}} + c_{\text{porovnání}}) + c_{\text{návratu}} = ak + b$

metoda **`pridanXtyDruh`** v implementaci spojovou strukturou je:

nejlepší případ k = 1, O(1)
nejhorší případ k = n, O(n)
průměrný případ k = n/2, O(n)

Dynamické množiny [Cormen et al.]

V matematice množina jako souhrn nějakých prvků je po její specifikaci neměnná.

Dynamická množina - souhrn prvků, který se může zvětšovat, zmenšovat nebo jinak měnit.

Dvě základní operace

- vlož prvek
- vyber prvek

Jednotlivé prvky jsou identifikovány hodnotou položky **klíč - `druh`** pro prvky **`DruhZvirat`**

Další typickou operací nad dynamickými množinami potom může být operace

- najdi prvek, kterého klíč má zadanou hodnotu

ADT ZÁSOBNÍK A FRONTA
STACK and QUEUE

Operace vyber ze zásobníku vyjme prvek, který byl operací vlož do zásobníku vložen jako poslední.

last-in, first-out LIFO

Operace vyber z fronty vyjme prvek, který je ve frontě vložený nejdéle, jinými slovy z prvků, které jsou ve frontě vybere ten, který byl do fronty vložen jako první.

first-in, first-out FIFO

Obě tyto ADT jsou používány mnoha algoritmy (klienty) na dalších úrovních abstrakce.

Implementace:

- polem
- spojovou strukturou

Zásobník

JVM je zásobníkový stroj - ukázali jsme si jeho použití na vyhodnocování výrazů.

Zpracování řádky textu na obrazovce

- píšeme = přidáváme znak na konec řádky
- mažeme - „backspace“ = odebereme poslední napsaný znak

Vyvážené použití závorek “{“ a “}” v deklaraci třídy

1.Čti znaky textu.

2. Je-li znakem otevírací závorka, vlož ji do zásobníku.

3. Je-li přečtený znak zavírací závorka, je-li zásobník prázdný ohlas chybu „chybějící otvírací závorka“, jinak vyber prvek ze zásobníku.

4. Jsou-li přečteny všechny znaky, je-li zásobník prázdný závorky jsou vyváženy, jinak ohlaš chybu „chybějící zavírací závorka“.

vlož - *push*

vyber - *pop*

Implementace zásobníku celých čísel

třída **`IntZasobnik`**

```
class IntZasobnik {
  // ADT rozhrani
  IntZasobnik()           // vytvoření prázdného zásobníku
  boolean jePrazdny()     // test je-li prázdný
  void push(int)          // vložení prvku
  int pop()               // výběr prvku
}
```

http://www.cs.usask.ca/resources/tutorials/csconcepts/index.html

Implementace pomocí pole

```
class IntZasobnik {

  private int[] z;
  private int vrchol;
  final int maxN=10;

  IntZasobnik() {
    z = new int[maxN];
    vrchol = 0;
  }

  boolean jePrazdny() {
    return (vrchol == 0);
  }

  void push(int klic) {
    z[vrchol++] = klic;
  }

  int pop() {
    return z[--vrchol];
  }
}
```

Čas každé z operací nad zásobníkem implementovaným pomocí pole je ***O(1).***

Vykonání operace **`pop`** nad prázdným zásobníkem - **podtečení** *underflow* zásobníku. V Javě lze na něj reagovat použitím mechanizmu výjimek.

Vykonání operace **`push (vrchol = maxN)`** - **přetečení** *overflow* zásobníku.

Velikost vytvářeného zásobníku - parametr konstruktoru.

```
private int[] z;
private int vrchol;

IntZasobnik(int maxN) {
  z = new int[maxN];
  vrchol = 0;
}
```

Technika dynamického rozšiřování pole

```
class DynPole {
  public static void main (String[] arg) {

    int maxN = 4;
    int n;

    int[] a = new int[maxN];
    for(n = 0; n < a.length; n++)
      a[n] = n;

    int[] x = a;

    a = new int[2*a.length];
    for(n = 0; n < x.length; n++)
      a[n] = x[n];

    for(n = maxN; n < a.length; n++)
      a[n] = n;

    for(n=0; n < a.length; n++)
      System.out.println(n);
 }
}
```

Čas vkládání při použití dynamického pole

agregovaná analýza

čas vyjádříme počtem přiřazení při vkládání i-tého prvku – $\mathbf{c_i}$
(začínáme s polem velikosti 1)

je-li pole plné, obsahuje **i -1** prvků, $\mathbf{i - 1 = 2^j,\ j=0,1,2,...}$
přiřadíme jeho prvky do nového pole a přiřadíme vkládanou i-tou hodnotu
$\mathbf{c_i = i}$

jinak, přiřadíme vkládanou i-tou hodnotu
$\mathbf{c_i = 1}$

amortizovaná analýza – průměrný čas operace
pro posloupnost **n** vykonaných operací

pole bylo plné $\mathbf{\lfloor \log_2 n \rfloor}$ krát
průměrný čas operace

$$\mathbf{1/n \sum_{i=1}^{n} c_i \le 1/n \left(n + \sum_{j=0}^{\lfloor \log_2 n \rfloor} 2^j\right) < 1/n\ (n + 2n) = 3}$$

operace vkládání (push) je tedy O(1)

účtovací metoda

- cena (čas) přímého vložení (přiřazení) nechť je **1$**
- mějme pole o velikosti **2m** v okamžiku jeho rozšíření
- obsahuje **m** vložených prvků a **m** volných pozic
- přímé vložení dalších **m** prvků stojí **m$**
- před dalším rozšířením musíme kopírovat (vložit do nového rošířeného pole) **2m** prvků, což stojí **2m$**
- cena vložení dalších **m** prvků je **m$ + 2m$ = 3m$**

Implementace zásobníku spojovým seznamem

```
class IntZasobnik {

  private Prvek vrchol;

  private class Prvek {
    int klic;
    Prvek dalsi;

    Prvek (int klic, Prvek dalsi) {
      this.klic = klic;
      this.dalsi = dalsi;
    }
  }

  IntZasobnik() {
    vrchol = null;
  }

  boolean jePrazdny() {
    return (vrchol == null);
  }

  void push(int klic) {
    vrchol = new Prvek(klic, vrchol);
  }

  int pop() {
    int v = vrchol.klic;
    vrchol = vrchol.dalsi;
    return v;
  }
}
```

Čas každé z operací nad zásobníkem implementovaným pomocí spojového seznamu je **O(1).**

Implementace ADT **`IntZasobnik`** můžeme zaměnit bez jakékoliv změny klientského programu.

```
class Zasobnik {
  public static void main(String[] arg) {

    IntZasobnik Zasobnik = new IntZasobnik();

    if (Zasobnik.jePrazdny())
     System.out.println("zasobnik je prazdny");

    Zasobnik.push(4);
    System.out.println(Zasobnik.pop());

    if (Zasobnik.jePrazdny())
    System.out.println("zasobnik je prazdny");

    Zasobnik.push(4);
    Zasobnik.push(3);
    Zasobnik.push(2);
    Zasobnik.push(1);

    System.out.println(Zasobnik.pop());
    System.out.println(Zasobnik.pop());
    System.out.println(Zasobnik.pop());
    System.out.println(Zasobnik.pop());
  }
}
```

Zásobník objektů

rozhraní:

```
class ObjZasobnik {
  //ADT rozhrani
  ObjZasobnik()
  boolean jePrazdny()
  void push(Objekt)
  Objekt pop()
}


class Objekt {
  ...
}
```

```
class ObjZasobnik {

  private Prvek vrchol;

  private class Prvek {
    Objekt ref;
    Prvek dalsi;

    Prvek (Objekt ref, Prvek dalsi) {
      this.ref = ref;
      this.dalsi = dalsi;
    }
  }

  ObjZasobnik() {
    vrchol = null;
  }

  boolean jePrazdny() {
    return (vrchol == null);
  }

  void push(Objekt ref) {
    vrchol = new Prvek(ref, vrchol);
  }

  Objekt pop() {
    Objekt r = vrchol.ref;
    vrchol = vrchol.dalsi;
    return r;
  }
}
```

Méně vhodné řešení je rozšířit ukládaný objekt o členskou proměnnou **`dalsi (`**jako **`DruhZvirat)`**, protože vyžaduje zásah do deklarace třídy ukládaných objektů**.**

Porovnání implementací

Implementace pomocí pole

- vyžaduje po celou dobu výpočtu paměť pro předpokládaný maximální počet prvků (pokud neuvažujeme dynamické pole)

Implementace pomocí spojového seznamu

- vyžaduje paměťový prostor úměrný počtu uložených prvků
- potřebuje paměť pro uchovávání ukazatelů a čas na přidělení paměti při každé operaci *push* a nakonec čas na uvolnění paměti po každé operaci *pop*.

http://java.sun.com/j2se/1.5.0/docs/api/java/util/Stack.html

Fronta

Fronta je reprezentací každodenních situacích v bance, jídelně, šatně ap.

Často je využívána v operačních systémech

Rozhraní

```
class IntFronta {
  // ADT rozhrani
  IntFronta()             // vytvoření prázdné fronty
  boolean jePrazdna()     // test je-li prázdná
  void vloz(int)          // vložení prvku
  int vyber()             // výběr prvku
}
```

Implementace fronty pomocí pole

- neposouváme ukládané prvky v poli
- udržujeme hodnoty dvou indexů

`zacatek` – index odkud vybereme prvek
`konec` –index kam uložíme prvek

```
zacatek = 0;
konec = 0;
```

po uložení prvku na konec:

```
konec++;
konec = konec % maxN;
```

po uložení maxN-tého prvku na index maxN – 1:

```
konec == 0;
```

V případě, že fronta je prázdná nebo plná jsou si tyto indexy rovny !

Vytvoříme pole pro implementaci fronty o 1 větší než maximální počet prvků fronty a rovnost indexů bude znamenat prázdné pole.

```
class IntFronta {

  private int[] f;
  private int zacatek, konec;
  final int maxN=5, n;

  IntFronta() {
    n = maxN + 1;
    f = new int[n];
    zacatek = 0;
    konec = 0;
  }

  boolean jePrazdna() {
    return (zacatek == konec);
  }

  void vloz(int klic) {
    f[konec++] = klic;
    konec = konec % n;
  }

  int vyber() {
    int v = f[zacatek++];
    zacatek = zacatek % n;
    return v;
  }
}
```

Čas každé z operací nad frontou implementovanou pomocí pole je ***O(1).***

Alternativa

můžeme použít proměnnou pro počet prvků ve frontě, která se inkrementuje v operaci **`vloz`** a dekrementuje v operaci **`vyber`** a podle její hodnoty určit, je-li fronta prázdná nebo plná.

Implementace fronty pomocí spojového seznamu

Použijeme ukazatel na konec seznamu, kam prvky vkládáme a ukazatel na začátek, odkud prvky vybíráme.

```
class IntFronta {

  private class Prvek {
    int klic;
    Prvek dalsi;

    Prvek(int klic) {
      this.klic = klic;
      this.dalsi = null;
    }
  }

  private Prvek zacatek, konec;

  IntFronta() {
    zacatek = null;
    konec = null;
  }

  boolean jePrazdna() {
    return (zacatek == null);
  }

  void vloz(int klic) {
    Prvek k = konec;    //ulozime konec pred vloz
    konec = new Prvek(klic);
    if (jePrazdna())
      zacatek = konec;
    else
      k.dalsi = konec;
  }
```

```
  int vyber() {
    int v = zacatek.klic;
    zacatek = zacatek.dalsi;
    return v;
  }
}
```

Čas každé z operací nad frontou implementovanou pomocí spojového seznamu je *O(1).*

Porovnání implementací

stejné jako pro zásobník

http://www.cs.usask.ca/resources/tutorials/csconcepts/index.html

http://java.sun.com/j2se/1.5.0/docs/api/java/util/Queue.html

```
class Fronta {

  public static void main(String [] arg) {

  IntFronta fronta = new IntFronta();
  if (fronta.jePrazdna())
    System.out.println("fronta je prazdna");

  fronta.vloz(4);
  System.out.println(fronta.vyber());

  if (fronta.jePrazdna())
    System.out.println("fronta je prazdna");

  fronta.vloz(1);
  fronta.vloz(2);
  fronta.vloz(3);
  fronta.vloz(4);
  fronta.vloz(5);

  System.out.println(fronta.vyber());
  System.out.println(fronta.vyber());
  System.out.println(fronta.vyber());
  System.out.println(fronta.vyber());
  System.out.println(fronta.vyber());

  if (fronta.jePrazdna())
    System.out.println("fronta je prazdna");

  }
}
```

http://www.kiv.zcu.cz/~netrvalo/vyuka/ppa2/portal/cviceni/cv04.pdf