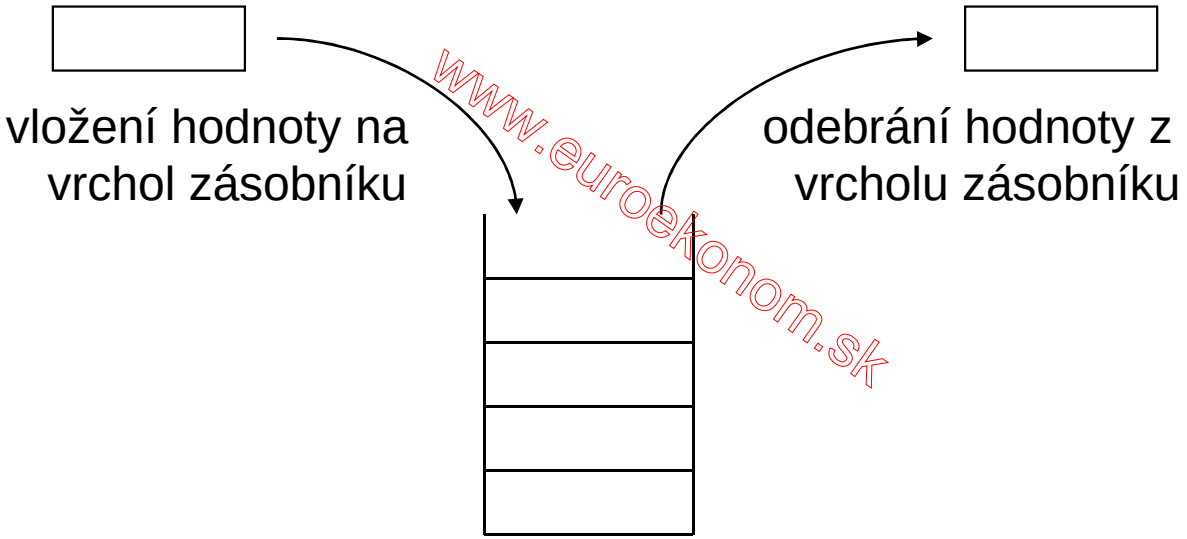
Datové struktury

- Jedna z klasických knih o programování (autor prof. Wirth) má název

  Algorithms + Data structures = Programs
- Datová struktura je množina dat (prvků, složek, datových objektů), pro kterou jsou definovány určité operace a stanoven způsob reprezentace (implementace)
- Datová struktura je statická, jestliže počet složek datové struktury je neměnný
- Statické datové struktury jsou ve vyšších programovacích jazycích přímo podporovány strukturovanými datovými typy (pole, řetězec, třída)
- Datová struktura je dynamická, jestliže počet jejích složek je proměnný (mezi operace patří např. vložení nového prvku, odebrání určitého prvku, atd.)
- Dynamickou datovou strukturou je např. zásobník, fronta, tabulka apod.
- V knihovně jazyka Java najdeme mnoho tříd pro práci s různými typy datových struktur
- V každé třídě realizující nějaký typ datové struktury jsou pro operace s datovou strukturou zavedeny veřejné instanční metody, implementace je však neveřejná
- Stejným způsobem budeme postupovat při návrhu vlastních datových struktur

Zásobník

- Zásobník (stack) je datová struktura, která umožňuje vkládání a odebírání hodnot, přičemž naposledy vložená hodnota se odebere jako první
- Zásobník je paměť typu LIFO (zkratka z angl. last-in first-out, poslední dovnitř, první ven)



- Základní operace:
  - vložení hodnoty na vrchol zásobníku
  - odebrání hodnoty z vrcholu zásobníku
  - test na prázdnost zásobníku

Zásobník

- Příklad třídy realizující zásobník znaků:

```
class ZasobnikZnaku {
  public ZasobnikZnaku() {...}
  public void vloz(char z) { ... }
  public char odeber() { ... }
  public boolean jePrazdny() { ... }
  ...
}
```

- Poznámky
  - v angličtině se operace nad zásobníkem obvykle jmenují *push*, *pop* a *isEmpty*
  - pro zásobník může být definována ještě operace čtení hodnoty z vrcholu zásobníku bez jejího odebrání (v angl. *top*)
- Implementací zásobníku se prozatím nebudeme zabývat, ukážeme si nejprve jeho použití

Příklad použití zásobníku

- Program, který přečte výraz obsahující tři druhy závorek a zkontroluje jejich správné párování
  - příklad správného párování: { [ ] ( ) }
  - příklad nesprávného párování: { [( ]) }
- Nástin řešení:
  1. Postupně projdeme všechny znaky tvořící výraz. Pro každý znak mohou nastat tři případy:
     - jestliže znak je otevírací závorka, pak jej uložíme do zásobníku
     - jestliže znak je zavírací závorka, pak ze zásobníku (musí být neprázdný) odebereme naposledy uloženou otevírací závorku a zjistíme, zda odpovídá zavírací závorce
     - jestliže znak není závorka, pak nás nezajímá
  2. Po zpracování všech znaků výrazu musí být zásobník prázdný (pokud není, pak chybí zavírací závorky)

Párování závorek

- Pomocné funkce:

```
static boolean jeOteviraci(char z) {
  return z=='(' || z=='[' || z=='{';
}

static boolean jeZaviraci(char z) {
  return z==')' || z==']' || z=='}';
}

static char zaviraciK(char z) {
  if (z=='(') return ')';
  if (z=='[') return ']';
  if (z=='{') return '}';
  return z;
}

static void chyba(String str) {
  Sys.pln(" chyba: "+str);
  System.exit(0);
}
```

Párování závorek

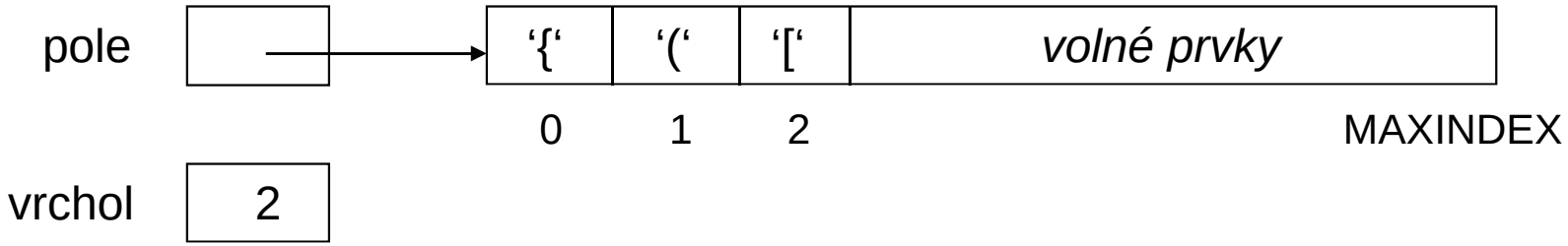
- Hlavní funkce:

```
public class Zavorky {
  public static void main(String[] args) {
    ZasobnikZnaku zasobnik = new ZasobnikZnaku();
    Sys.pln("zadejte řádek se závorkami");
    String str = Sys.readLine();
    int i;
    for (i=0; i<str.length(); i++) {
      char znak = str.charAt(i);
      Sys.p(znak);
      if (jeOteviraci(znak))
        zasobnik.vloz(znak);
      else if (jeZaviraci(znak)) {
        if (zasobnik.jePrazdny())
          chyba("k této závorce chybí otevírací");
        char ocekavany = zaviraciK(zasobnik.odeber());
        if (znak!=ocekavany)
          chyba("očekává se "+ocekavany);
      }
    }
```

Párování závorek

```
if (!zasobnik.jePrazdny()) {
     String chybi = "";
     do {
       chybi = chybi + zasobnik.odeber();
     } while (!zasobnik.jePrazdny());
     chyba("chybí závírací závorky k "+chybi);
   }
   Sys.pln();
 }
```

Implementace zásobníku polem




```
class ZasobnikZnaku {
  static final int MAXINDEX = 99;
  private char[] pole;
  private int vrchol;

  public ZasobnikZnaku() {
    pole = new char[MAXINDEX+1];
    vrchol = -1;
  }
```

Implementace zásobníku polem

```
  public void vloz(char z) {
    if (vrchol==MAXINDEX)
      throw new RuntimeException
                    ("vložení do plného zásobníku");
    pole[++vrchol] = z;
  }

  public char odeber() {
    if (vrchol<0)
      throw new RuntimeException
                    ("odebrání z prázdného zásobniku");
    return pole[vrchol--];
  }

  public boolean jePrazdny() {
    return vrchol<0;
  }
}
```


- Poznámka: *RuntimeException* je třída výjimek, jejichž šíření z funkce není třeba deklarovat v hlavičce funkce

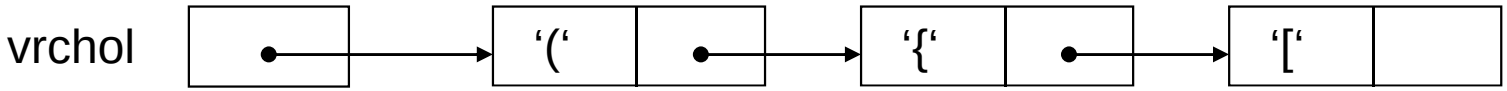
Implementace zásobníku rozšiřitelným polem

- Při vkládání do plného zásobníku lze vytvořit nové, větší pole a původní pole do něj zkopírovat

```
public class ZasobnikZnaku2 {
  private char[] pole;
  private int vrchol;

  public ZasobnikZnaku2() {
    pole = new char[2];
    vrchol = -1;
  }

  public void vloz(char z) {
    if (vrchol==pole.length-1) {
      char[] nove = new char[2*pole.length];
      System.arraycopy(pole,0,nove,0,pole.length);
      pole = nove;
    }
    pole[++vrchol] = z;
  }
  ...
}
```

Implementace zásobníku spojovým seznamem

- Spojový seznam je datová struktura, jejíž prvky tvoří posloupnost a každý prvek obsahuje odkaz na další prvek seznamu



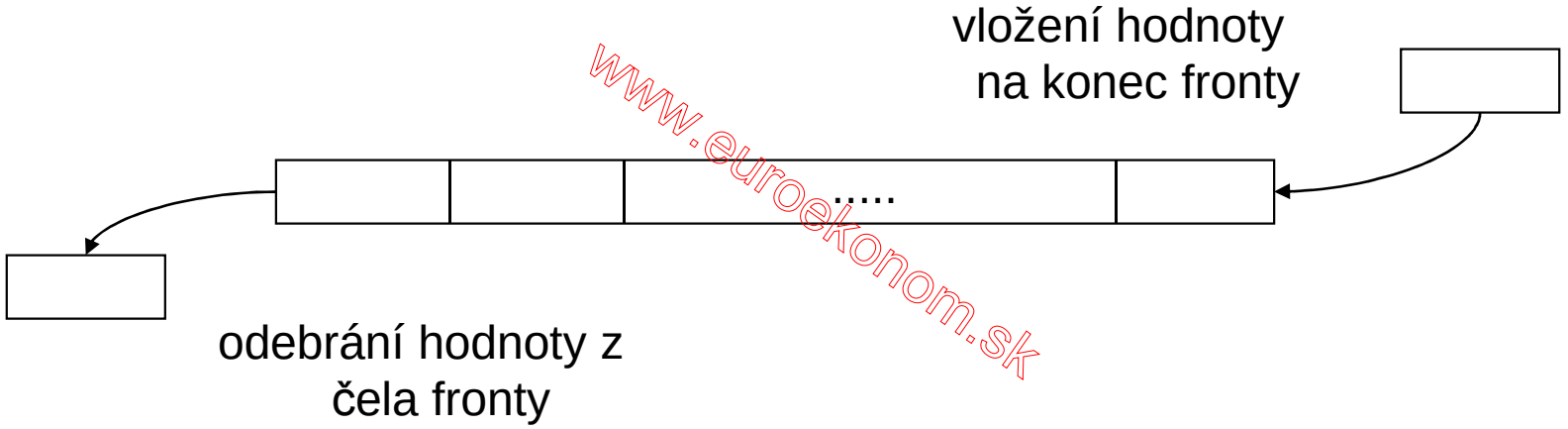

```
class Prvek {
  char hodn;
  Prvek dalsi;

  public Prvek(char h, Prvek p) {
    hodn = h; dalsi = p;
  }
}

class ZasobnikZnaku {
  private Prvek vrchol;

  public ZasobnikZnaku() {
    vrchol = null;
  }
```

Implementace zásobníku spojovým seznamem

```
  public void vloz(char z) {
    vrchol = new Prvek(z, vrchol);
  }

  public char odeber() {
    if (vrchol==null)
      throw new RuntimeException
                ("odebrani z prazdneho zasobniku");
    char vysl = vrchol.hodn;
    vrchol = vrchol.dalsi;
    return vysl;
  }

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

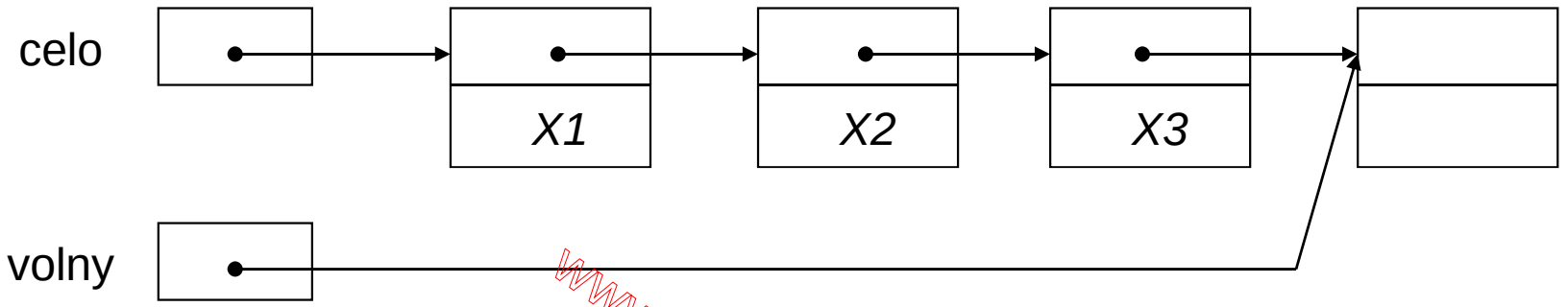
Fronta

- Fronta je datová struktura podobná zásobníku, hodnoty se však odebírají v tom pořadí, v jakém byly vloženy
- Fronta je paměť typu FIFO (zkratka z angl. first-in first-out, první dovnitř, první ven)



- Implementace fronty:
  - pomocí pole
  - pomocí spojového seznamu

Implementace fronty spojovým seznamem



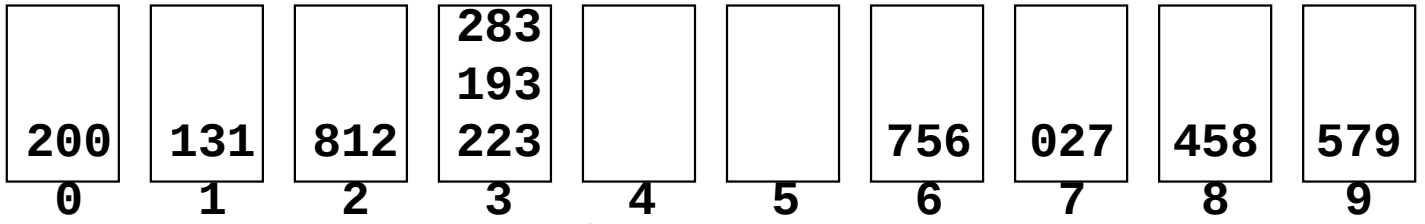

```
class PrvekFronty {
  PrvekFronty dalsi;
  int hodn;
}

public class FrontaCisel {
  private PrvekFronty celo;
  private PrvekFronty volny;
```

Implementace fronty spojovým seznamem

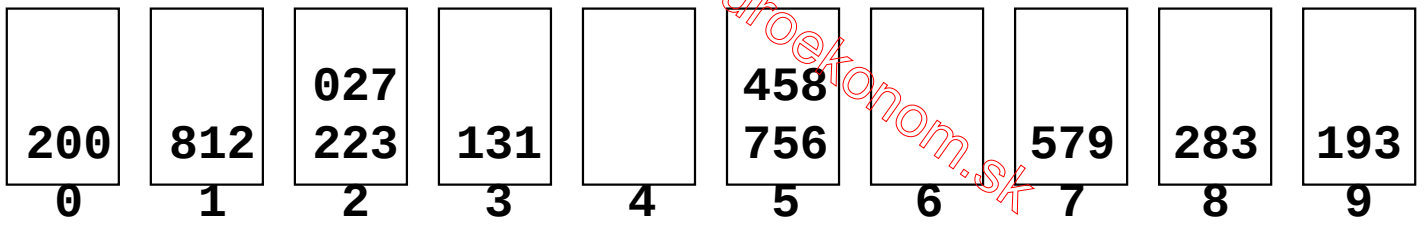
```
  public FrontaCisel() {
     celo = new PrvekFronty();
     volny = celo;
   }

   public void vloz(int x) {
     volny.hodn = x;
     volny.dalsi = new PrvekFronty();
     volny = volny.dalsi;
   }

   public int odeber() {
     if (jePrazdna())
       throw new RuntimeException("odebrání z prázdné fronty");
     int vysl = celo.hodn;
     celo = celo.dalsi;
     return vysl;
   }

   public boolean jePrazdna() {
     return celo == volny;
   }
 }
```

Použití fronty: přihrádkové řazení

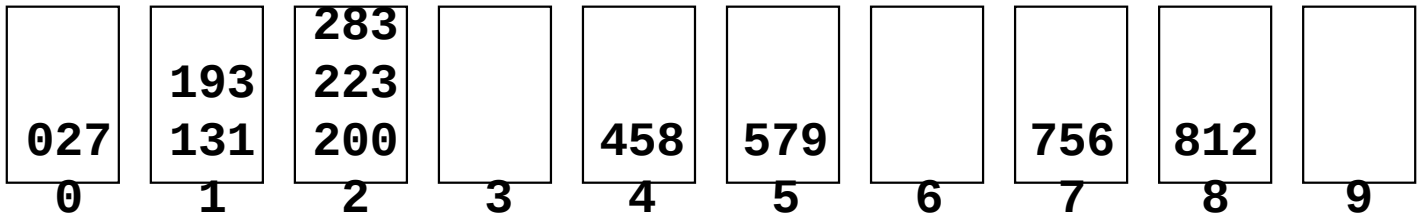
- 1. průchod 223 131 458 193 756 812 027 579 283 200
  rozdělení podle poslední číslice



- 2. průchod 200 131 812 223 193 283 756 027 458 579
  rozdělení podle druhé číslice



- 3. průchod 200 812 223 027 131 756 458 579 283 193
  rozdělení podle první číslice



Výstup: 027 131 193 200 223 283 458 579 756 812

Použití fronty: přihrádkové řazení

```
static int hexCislice(int x, int r) {
  return (x >> 4*r) & 0xF;
}

static void caseSort(int[] pole) {
  FrontaCisel[] prihradky = new FrontaCisel[16];
  int r, j, c;
  for (c=0; c<16; c++)
    prihradky[c] = new FrontaCisel();
  for (r=0; r<8; r++) {
    for (j=0; j<pole.length; j++)
      prihradky[hexCislice(pole[j],r)].vloz(pole[j]);
    j = 0;
    for (c=0; c<16; c++)
      while (!prihradky[c].jePrazdna())
        pole[j++] = prihradky[c].odeber();
  }
```