- **`deleteCharAt(poz)`** … izbriše znak na poziciji *poz*

```
StringBuffer sb = new StringBuffer("Brat");
sb.deleteCharAt(1);
System.out.println(sb);  // izpis: Bat
```

- **`indexOf(niz2)`** … vrne položaj, kjer se *niz2* pojavi prvič. Če niza ne najde, vrne –1
- **`indexOf(niz2,poz)`** … išče od pozicije *poz* dalje. Z metodo lahko v nizu odkrijemo več pojavitev iskanega podniza .

```
StringBuffer sb = new StringBuffer("Janez Vajkard Valvasor");
System.out.println(sb.indexOf("Va"));        // izpiše se: 6
System.out.println(sb.indexOf("Va",7));      // izpiše se:14
```

8.3 Delo z nizi znakov in razred *StringTokenizer*

Objekti tega razreda znajo niz razbiti ("razsekati") na mestih v nizu, kjer je določen ločilni *znak* ali *niz* in vračati dobljene podnize (t. i. žetone, angl. *tokens*). Za uporabo razreda *StringTokenizer* moramo **uvoziti** razred *java.util.StringTokenizer* .

Razred *StringTokenizer* ima samo nekaj metod. Za našo nalogo potrebujemo dve:

- **`hasMoreTokens()`** … vrne *true*, če je v nizu ostal še kakšen del, žeton, sicer vrne *false*
- **`nextToken()`** … vrne naslednji 'odrezek' (žeton) začetnega niza

Primer: Imamo niz *imenaOtrok*, ki vsebuje imena otrok, ločena z vejicami. Imena otrok želimo izpisati na zaslon eno za drugim (navpično). Kako naj jih izluščimo iz niza?

Izdelamo objekt vrste *StringTokenizer*, mu priredimo vrednost našega niza in določimo, da bomo podnize (žetone) 'rezali' pri vejicah. Nato v zanki 'odrezujemo' del za delom, dokler je kaj odrezati.

Najprej ustvarimo objekt vrste *StringTokenizer*, nato z uporabo teh dveh metod izpišemo imena:

```
import java.util.StringTokenizer;

class Salama{
   public static void main(String[] args){

   String imenaOtrok = "Jan,Ana,Peter,Miha";
   StringTokenizer seznam = new StringTokenizer(imenaOtrok,",");

   while (seznam.hasMoreTokens()){
      System.out.println(seznam.nextToken()); }
   }
}
```

Izpiše se:

```
Jan
Ana
Peter
Miha
```

9 PRESTREZANJE IZJEM

V programih se seveda lahko zgodi in tudi *se dogaja*, da pride do takšne ali drugačne neželene ali nepričakovane situacije, zaradi katere se naš program ustavi in izvajanje se lahko prekine sredi dela.

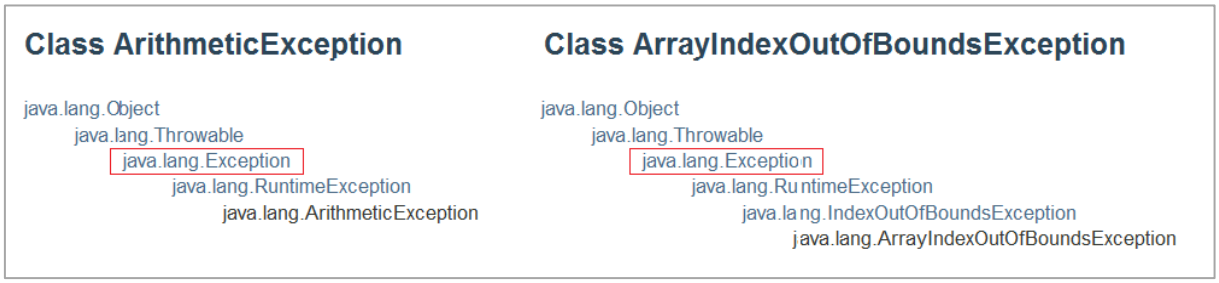
Pogosto pa lahko predvidimo, kje bi lahko prišlo do "napak", ki bi naš program pripeljale v težave in do ustavitve. Primer takšne napake bi lahko bil napačen vnos številske vrednosti – uporabnik se zatipka in pomotoma pripiše še kakšno črko. Ali pa vpiše decimalno vejico namesto decimalne pike.

Takšnim neljubim dogodkom ali situacijam, zaradi katerih bi bilo delovanje programa nepravilno ali bi se ustavil, pravimo **izjeme** (angl. *exception*). Java ima za to področje vpripravljen mehanizem, s katerim skušamo preprečiti sesutje programa: **prestrezanje izjem**.

9.1 Izjeme

Izjema je torej neljub *dogodek* v življenju programa. Izjeme se v Javi obravnavajo kot predmeti, objekti posebnih razredov. Še več, obstaja razvejana hierarhija razredov izjem in vsaka posamezna izjema (ko se zgodi) je objekt določenega razreda. Imena teh razredov opisujejo vrsto izjeme, končajo pa se z besedo *Exception*, npr. *IOException*, *DataFormatException* itd. Podrobnosti najdemo v dokumentaciji teh razredov, enako kot za ostale razrede.

Poglejmo primer. V dokumentaciji razreda *ArithmeticException* (glej sliko spodaj) najdemo 'rodovnik' tega razreda. Med predniki je tudi razred *Exception*, ki je nadrazred vseh razredov izjem. Izjema *ArithmeticException* se zgodi npr. pri deljenju z nič. Takrat se samodejno ustvari objekt razreda *ArithmeticException*. Preko njegovih metod lahko o nastali izjemi izvemo nekatere podrobnosti.



Slika 58: Rodovnika izjem ArithmeticException in ArrayIndexOutOfBoundsException

Podobno velja tudi za razred *ArrayIndexOutOfBoundsException*, izjemo, ki smo jo (skoraj gotovo) srečali, ko smo hoteli vpisati podatek v neveljavno celico tabele. Tudi ta razred ima istega skupnega prednika – razred *Exception.*

Kako lahko nadzorujemo izvajanje programa, ko pride do izjeme?

Java ima za takšne situacije vgrajen mehanizem za *prestrezanje izjem* in tudi programsko strukturo, v katero lahko vprogramiramo obravnavo takšnih situacij. Ko se pri izvajanju programa zgodi ***izjema,*** "napaka", Java preko **nadzornika napak** (an. *Exception Handler*) sporoči napako svojemu notranjemu mehanizmu. Če napake ne **prestrežemo** (an. *catch*) in ustrezno ukrepamo, se delovanje programa prekine. Lahko pa jih **prestrežemo** in ustrezno 'obdelamo' s programsko strukturo ***try-catch-finally*.**