Harjutustund 5

Klassid `String`, `StringBuilder`, `Character`.

Sõned (Klass `String`)

Erinevalt arvutüüpidest (täisarvutüübid `int`, `byte`, `short`, `long` ja reaalarvutüübid `double`, `float`), loogilisest tüübist (`boolean`) ja sümboltüübist (`char`) pole sõnetüüp keeles *Java* algtüüp, vaid iga sõne on klassi `String` isend. Kui vaataksime näiteks Java API-st, siis näeme, et klassi `String` isendi loomiseks on võimalik kasutada üle kümne erineva konstruktori. Näiteks on nende hulgas konstruktor, mis nõuab argumendiks sümbolite järjendit.

```
char[] symbolsArray = {'H','e','l','l', 'o', ',',
'w','o','r','l','d','!'};
String text = new String(symbolsArray);
System.out.println(text);
```

On aga ka lihtsam viis klassi `String` isendi loomiseks, nimelt kiirloome *sõneliteraali* abil:
`String text = "Hello, world!";`

Literaal on konkreetse väärtuse üleskirjutus programmis. Väärtuse tüüp on määratud kirjakujuga, nt. 15 on `int`-tüüpi, 15L aga `long`-tüüpi. Sõneliteraali tähistavad jutumärgid. Kui sõneliteraali sees on vaja jutumärke kasutada, siis saab seda teha langkriipsu abil, nt `"\""`.

Edasi, kui klassi `String` isend on loodud, saab kasutada klassis `String` defineeritud isendimeetodeid, näiteks `length`, `toLowerCase` jne. Nagu eelmises praktikumis õppisime, kasutatakse isendimeetodit koos konkreetse isendiga. Isendi nimi ja meetodi nimi ühendatakse punktiga, näiteks sõne `text` pikkus leitakse järgmiselt:

```
text.length()
```

Kui me soovime kontrollida, kas sõne on tühi, siis eeldades, et sõne muutuja ei ole olematu, on parim viis kontrollida sõne pikkust:

```
String str = "";
if (str.length() == 0) {
  System.out.println("Empty string");
}
```

Järgnev koodilõik demonstreerib klassi `String` meetodeid. Leidke nende meetodite täpsed kirjeldused Java API-st. Tutvuge API-s ka teiste klassi `String` meetoditega.

Sõne sümbolite nummerdamine algab nullist (analoogiliselt massiiviindeksiga).

```
String name = "Mart Mardikas";
System.out.println("Sõne pikkus on: " + name.length()); // 13
System.out.println(name.startsWith("Mart")); // true
System.out.println(name.endsWith("kas")); // true
System.out.println(name.endsWith("Mart")); // false
System.out.println("\'a\' esimene positsioon: " +
name.indexOf('a')); // 1
int rIndex = name.indexOf('r');
System.out.println("\'r\' esimene positsioon: " + rIndex); // 2
System.out.println("\'r\' jargmine positsioon: " +
name.indexOf('r', rIndex + 1)); // 7
int aIndex = name.lastIndexOf('a');
System.out.println("\'a\' viimane positsioon: " + aIndex); // 11
System.out.println("Alamsõne \'Mardi\' algus: " +
name.indexOf("Mardi")); // 5
System.out.println("4. täht on "+name.charAt(3)); // 't'

//Täpne võrdsuse kontroll:
System.out.println(name.equals("Mart Mardikas")); // true
System.out.println(name.equals("mart mardikas")); // false

//Suuri-väikesi tähti mitteeristav võrdsuse kontroll:
System.out.println(name.equalsIgnoreCase("mart mardikas")); //true

//Leksikograafiline võrdlemine:
System.out.println(name.compareTo("Jaan Jaaniste")); // >0
System.out.println(name.compareTo("Peeter Paan")); // <0
System.out.println(name.compareTo("Mart Mardikas")); // =0

System.out.println(name.replace('M', 'P')); // "Part Pardikas"
System.out.println(name.toUpperCase()); // "MART MARDIKAS"

System.out.println(name.substring(0,4)); // "MART"
System.out.println(name.substring(5)); // "MARDIKAS"
```

Näide:

```
String str = "abcdifghjkli";
//int index = str.indexOf('i');
char ch =str.charAt(4);

System.out.println(str);
System.out.println(ch);

//Asendame ‘i’ ‘e’-ga :
//1.
str = str.replace(ch,'e');
System.out.println(str);
```

```
//2.
str = "abcdifghjkli";
char[] charsArray = str.toCharArray();
charsArray[4]='e';
str = new String(charsArray);
System.out.println(str);
//3.
str = "abcdifghjkli";
str = str.substring(0,4)+"e"+ str.substring(5);
System.out.println(str);
```

Sõnede tükeldamine

Sõnede tükeldadamiseks on mitu võimalust (nt. klass `StringTokenizer`). Siin praktikumis kasutame tükeldamiseks klassi `String` isendimeetodit `split`. Selle meetodi tagastustüübiks on `String[]`, mis tähendab, et tulemuseks saadakse sõnede massiiv. Argumendiks on regulaaravaldis, mis määrab, milliseid sümboleid lugeda eraldajateks. Järgmises näites on eraldajaks tühik

```
String[] pieces = name.split(" ");
for (int i = 0; i< pieces.length; i++)
    System.out.println(pieces[i]);
```

Antud juhul saaksime siis ekraanile

```
Mart
Mardikas
```

Kui aga määrata eraldajaks täht "a"

```
String[] pieces = name.split("a");,
```

siis saame ekraanile

```
M
rt M
rdik
s
```

Regulaaravaldis võib olla ka keerulisem, näiteks `"[art]"` korral loetakse eraldajaks nii tähte `"a"` kui ka tähti `"r"` ja `"t"`.

Veel üks näide:

```
String names = "Mari Maasikas, Johan Juurikas, Mart Mardikas";
String[] pieces = name.split(", ");
for (int i = 0; i< pieces.length; i++)
```

```
    System.out.println(pieces[i]);
```

Antud juhul saaksime siis ekraanile

```
Mari Maasikas
Johan Juurikas
Mart Mardikas
```

Unicode

Java kasutab tekstitöötlusel kooditabelit Unicode, mis sisaldab tähti jm. sümboleid erinevate tähestike jaoks. Iga sümbol on *Javas* esitatud 16-bitise märgita täisarvuna, seega võimaldab kooditabel esitada $2^{16} = 65\ 536$ erinevat sümbolit. Paljud teised programmeerimiskeeled kasutavad tekstitöötlusel ASCII kooditabelit, milles sümbolite kodeerimiseks kasutatakse 7-bitiseid arve (128 sümbolit). Samas on ASCII Unicode-i alamhulk – kooditabeli Unicode esimesed 128 sümbolit. Esimesed 256 moodustavad aga kooditabeli ISO-Latin-1 extended ASCII. Sõnede võrdlemisel, näiteks meetodiga `compareTo` võrreldakse tegelikult paarikaupa nende sümbolite arvulisi koode antud kooditabelis.

Sõnede võrdlus

Ülaltoodud meetodite hulgas on mitmeid, mis kaht klassi `String` isendit omavahel võrdlevad. Võrduse kontrolliks saab kasutada meetodit `equals`. Miks ei võiks kahe isendi võrdust aga kontrollida märgipaari == abil nagu arvude võrduse kontrollimiseks? Tegelikult ongi märgipaar == täiesti lubatud ja kompileerimisel veateadet ei tule. Võrdusmärkide puhul võrreldakse isendite viitasid, aga meetod `equals` arvestab isendite sisu.

```
char[] symbolsArray = {'H','e','l','l','o'};
String text1 = new String(symbolsArray);
String text2="Hello";
String text3="Hello";

System.out.println(text1.equals(text2));
System.out.println(text2.equals(text3));
System.out.println(text1==text2);
System.out.println(text2==text3);
```

Kuna sõne on *Javas* muutmatu ja sageli kasutatav, siis JVM (Java virtuaalmasin) säästab mälu ja kasvatab jõudlust sellega, et paneb kiirloome abil sama sõneliteraaliga loodud klassi `String` isendid ühte. Seda sõne nimetatakse *kanooniliseks sõneks*. Nii ongi viitade mõttes `text2` ja `text3` võrdsed, aga `text1` nendega mitte. Sisu mõttes on aga kõik kolm võrdsed. Kuna tavaliselt on meil just sisu mõttes võrdsust vaja kontrollida, siis kasutamegi meetodit `equals`.

Klass `StringBuilder`

Eespool nägime palju meetodeid, mida saab rakendada klassi `String` isendi puhul. Tähtis on aga silmas pidada, et need meetodid ei muuda isendit ennast. Sõne on *Javas* muutumatu – tema sisu ei saa muuta. Ka näiteks read

```
String s = "Soome"
s = "Poola";
```

ei muuda sõne `s` sisu. Esimene rida loob isendi, mille sisuks on `"Soome"` ja omistab selle viida `s`-ile. Teine rida loob isendi, mille sisuks on `"Poola"` ja omistab selle viida `s`-ile. Esimene objekt jääb tegelikult alles (ega muutu), aga tema poole ei saa enam pöörduda.

Võime öelda, et klass `String` käsitleb sõne staatiliselt. Kui aga tahame sõne dünaamiliselt käsitleda (nt. muuta), siis on sobiv klass `StringBuilder`.
Klassis `StringBuilder` (vt. API) on neli konstruktorit, millest meie vaatleme kolme. Parameetriteta konstuktori abil moodustub klassi `StringBuilder` isend, milles on kohti 16 sümboli jaoks, aga ühtegi sümbolit (esialgu) pole. Kui isendi loomisel anda ette täisarv, siis tekibki kohti niipaljude sümbolite jaoks. Kui argumendiks on sõne, siis vastava sõne sümbolid "puhvrisse" pannaksegi.

```
StringBuilder()
StringBuilder(int)
StringBuilder(String)
```

Meetodeid on mõnikümmend, mitmed neist on korduva nimega. Meie jaoks olulisemad on

- `append` (lõppu lisamine)
- `capacity` (maht)
- `charAt` (sümbol vastaval kohal)
- `delete` (kustutamine)
- `insert` (lisamine vastavale kohale)
- `length` (pikkus)
- `setCharAt` (sümboli asendamine vastaval kohal)
- `setLength` (pikkuse muutmine)
- `replace` (asendamine)
- `reverse` (transponeerimine)

Toomegi mõningad näited nende kasutamisest. Kõigepealt mõned meetodid, mille väärtuseks on täisarv ja mis veel isendi sisu ei muuda.

```
StringBuilder sb = new StringBuilder("Suusalumi-suusalumi sadas
õhtust");
System.out.println(sb.capacity());
System.out.println(sb.length());
System.out.println(sb.indexOf("sada"));
```

Järgmine meetod tagastab `char`-tüüpi väärtuse.

```
System.out.println(sb.charAt(10));
```

Järgmised meetodid on `void`-tüüpi ja muudavad vastavalt sisu ja pikkust.

```
sb.setCharAt(10, 'p');
sb.setLength(20);
System.out.println(sb);
```

Järgmisedki meetodid muudavad sisu, aga nad tagastavad ka viida isendile endale. Neid saab kasutada kahel moel. Kui me pole tagastatavast viidast huvitatud, siis võime neid meetodeid ilma omistamata kasutada.

```
sb.append(" ei sadanud");
sb.delete(10,18);
sb.insert(10, "lum");
sb.replace(0, 5, "Uisu");
sb.reverse();
```

Seda omadust saab kasutada näiteks nii:

```
StringBuilder tervitusSB = new StringBuilder();
String tervitatav = "Peeter";
tervitusSB.append("Hello, ").append(tervitatav).append("!");
```

Klass `Character`

Mähisklass (wrapper `class`) tüübile **`char`**.

```
Character co = new Character('a');
System.out.println(co.compareTo(new Character('a'))); // 1
System.out.println(co.compareTo(new Character('b')));// 0
System.out.println(co.compareTo(new Character('c')));// –1
System.out.println(co.compareTo(new Character('d')));// –2
System.out.println(co.equals(new Character('b'))) ;// true
System.out.println(co.equals(new Character('d')));// false
char c = co.charValue(); // c = 'b'

System.out.println(Character.isLetter(c1));// true
System.out.println(Character.isDigit(c1));// false
System.out.println(Character.isLetterOrDigit(c1));// true
System.out.println(Character.isUpperCase(c1));// false
System.out.println(Character.isLowerCase(c1));// true
```

Ülesanded

1. Kirjutada meetod, mis leiab isiku sugu isikukoodist. Tagastab `'N'` või `'M'`.

2. Kirjutada meetod, mis kontrollib parooli korretksust parooli loomisel. Paroolis peab olema vähemalt 1 väiketäht, 1 suurtäht ning 1 number. Näiteks, “asdFg1” on korrektne.

```
isPasswordCorrect ("asdFg1")          true
isPasswordCorrect ("ABCDF1")          false
isPasswordCorrect ("asdFghjk")        false
isPasswordCorrect ("asdhg1")          false
```

Ülesanded koju

1. Leida lõigus [5..300] kõigi paaritu arvude aritmeetiline keskmine ning tulemus trükkida ekraanile.
   Abi: Paaritu arv on arv, mis ei jagu arvuga 2 (või mis jagamisel 2-ga annab jäägi 1).
   x_1,x_2, ..., x_n aritmeetiline keskmine on (x_1,x_2, ..., x_n)/n.

2. Kirjutada meetod findEvenNumbers(numbers), mis tagastab etteantud arvuloendis loend sisaldunud paarisarvude loendi. Näiteks:

   ```
   findEvenNumbers ([0, 1, 40, 60])     [0, 40, 60]
   findEvenNumbers ([7, 43, 99])        []
   findEvenNumbers ([1, 2, 3, 4, 5])    [2, 4]
   ```

3. Kirjutada meetod, mis kontrollib parooli korretksust parooli loomisel. Parool peab olema vähemalt 8 sümbolit pikk ja sisaldama suurtähti ja numbreid.

   ```
   isPasswordCorrect ("asdFg1io")        true
   isPasswordCorrect ("asdFg1ioPP")      true
   isPasswordCorrect ("asdFg1i")         false
   isPasswordCorrect ("asdFghjk")        false
   isPasswordCorrect ("asdhg111")                false
   ```

4. Võtke sõnena kasutusele üks lause, milles on vähemalt 5 sõna. Kirjutage meetod, mis leiab kõik lauses esinevad sõnad ja mõõdab, kui pikad need on. Ekraanile peab väljastatama alglause ning sõnad ja nende pikkused.

Kirjutada testklass, mis testib meetodeid ülaltoodud näidetes antud väärtustega.