Stringovi

Tekstualni tipovi podataka su u Java programskom jeziku predstavljeni tipom **`String`**. Iako se String ponaša kao primitivan tip, u stvari je realizovan kao interna Java klasa. Svako navođenje tekstualnog tipa u javi (vrednost u dvostrukim navodnicima) kreira jednu instancu tipa String, iako se ne koristi operator **`new`**.

U Java programskom jeziku, String vrednosti se navode unutar dvostrukih navodnika, dok su jednostruki rezervisani za karaktere (tip **`char`**).

Svaki String je niz karaktera, tako da se elementima može pristupati po indeksu, takođe se može koristiti **`for`** petlja za iteraciju po karakterima u stringu.

Korisne metode

Primeri stringova:

```
String string1 = "Dobar dan.";

String string2 = "dan";

String string3 = "Crvena,Zelena,Plava";
```

Naziv	Opis	Primer	Rezultat
`length()`	Dužina stringa.	`string1.length();`	`10`
`toUpperCase()` **`toLowerCase()`**	Prevodi sva slova u stringu u velika (mala).	`string1.toUpperCase();` `string1.tolowerCase();`	`DOBAR DAN.` `dobar dan.`
`charAt()`	Vraća karakter na zadatoj poziciji u stringu.	`string1.charAt(4);`	`r`
`indexOf()`	Vraća prvi indeks na kojem se pojavljuje zadani string ili karakter (-1 ako se ne pojavljuje).	`string1.indexOf('a')` `string1.indexOf(string2)`	`3` `6`
`substring()`	Vraća karaktere koji se nalaze između zadanih pozicija. Ako se prosledi samo jedan parametar uzima se od tog indeksa do kraja stringa.	`string1.substring(3, 7)`	`ar d`
`split()`	Vrši "isecanje" teksta po zadanom karakteru. Rezulate smešta u niz.	`string3.split(",")`	`[Crvena, Zelena, Plava]`
`trim()`	Izbacuje razmake sa početka i kraja stringa. Vrlo korisno za vrednosti preuzete od korisnika.	`string1.trim()`	
`replace()`	Menja prosledjeni stari string sa novim.	`string1.replace(".", "!")`	`Dobar dan!`
`contains()`	Proverava da li string sadrži prosleđeni string.	`string1.contains(string2)`	`true`

Poređenje Stringova

U Java programskom jeziku, poređenje stringova korišćenjem operatora == ne vrši leksičko poređenje teksta, već proverava da li su operandi referenca na isti objekat. Ukoliko želimo da vršimo leksičko poređenje stringova (da li su svi karakteri u dva teksta isti), moramo da koristimo funkcije **`equals`** ili **`equalsIgnoreCase`**.

Primer	Rezultat
`"Dobar dan".equals("dobar dan")`	`false`
`"Dobar dan".equalsIgnoreCase("dobar dan")`	`true`

Preuzimanje teksta od korisnika

Za preuzimanje teksta sa standardnog ulaza (tastature) koristićemo Java klasu **Scanner**.

```
Scanner scanner = new Scanner(System.in);

System.out.println("Molimo unesite tekst: ");
String input = scanner.nextLine();

System.out.println("Molimo unesite celi broj: ");
int inputNum = scanner.nextInt();

scanner.close();
```

U zavisnosti od toga koji tip podatka očekujemo od korisnika, klasa Scanner obezbeđuje odgovarajuće metode koje vrše konverziju tipa (nextInt(), nextBoolean, nextDouble(), ...).

Zadaci

1. Kreirati program koji proverava da li je zadati tekst palindrom. Tekst je palindrom ako je isti pročitan unapred i unazad. Tekst za proveru preuzeti sa tastature.
2. Kreirati program za analizu jedinstvenog matičnog broja građana.

   Jedinstveni matični broj građana sastoji od 13 cifara u obliku DDMMGGGRRBBBK, gde su:

   DD - dan rođenja
   MM - mesec rođenja
   GGG - zadnje tri cifre godine rođenja
   RR - region rođenja, odn. prebivalište za građane rođene pre 1976. godine.
   BBB - jedinstveni broj, dodeljen prema

   - 000-499 - muški
   - 500-999 - ženski

   K - kontrolna cifra

   Na osnovu ovih podataka, tražiti od korisnika da unese JMBG i ispisati datum rođenja i pol.

Izuzeci

Izuzeci (eng. Exceptions) predstavljaju greške koje se javljaju u toku rada programa (run-time errors). Za razliku od sintaksnih grešaka, koje se otkrivaju kompajliranjem programa (compile-time errors), izuzeci mogu a ne moraju da prekinu rad programa.

Kada se desi izuzetak, Java program završava sa svojim izvođenjem, iako mi to nismo predvideli, što uzrokuje gubitak nesačuvanih podataka i narušava korisničko iskustvo.

Primer

Sledeći kod će izazvati izuzetak jer je umesto broja, metodi za konverziju prosleđeno slovo.

```
int ocena = Integer.parseInt("a");
```

Izuzetak koji se ovom prilikom javlja je tipa **NumberFormatException** i manifestuje se prekidom programa I sledećim tekstom u konzoli:

```
Exception in thread "main" java.lang.NumberFormatException: For input string: "a"
	at java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
	at java.lang.Integer.parseInt(Integer.java:580)
	at java.lang.Integer.parseInt(Integer.java:615)
	at Izuzeci.main(Izuzeci.java:6)
```

U ovakvim slučajevima, kada znamo koji deo programa može uzrokovati grešku možemo reagovati na željeni način uz pomoć **try catch** bloka:

```
try {
	int ocena = Integer.parseInt("a");
} catch (NumberFormatException e) {
	System.out.println("Prosledjeni tekst ne može biti pretvoren u broj.");
}
```

Na ovaj način, umesto da se program neprirodno završi u slučaju pojave izuzetka, izvršava se programski kod koji smo definisali u **catch** delu. **catch** deo može da sadrži proizvoljno složen java kod, ne samo obaveštenje o grešci.

S obzirom da svi Java izuzeci nasleđuju klasu **Exception**, ukoliko ne znamo unapred koji tip izuzetka očekujemo, u **catch** blok možemo staviti **Exception** objekat:

```
try {
	int ocena = Integer.parseInt("a");
} catch (Exception e) {
	System.out.println("Došlo je do greške");
}
```

Ako očekujemo nekoliko vrsta izuzetaka, možemo napisati kondicionalni **catch** blok:

```
	try {
		...
	} catch (NumberFormatException | NullPointerException e ) {
		...
	}
```

Ili možemo napisati posebne **catch** blokove za svaki tip izuzetka. Ova opcija je bolja ukoliko želimo da imamo različite reakcije za različite tipove izuzetaka.

```
	try {

		// ...
	}catch (NumberFormatException e ) {
		// ...
	}catch (NullPointerException e) {
		// ...
	}
```

Osim **catch** blokova, **try** može da ima i jedan **finally** blok. Finally blok uvek sledi **try** blok (**catch** nije obavezan) i izvršava se uvek posle **try** bloka. Ovi blokovi su korisni ukoliko želimo da se kod koji sledi nakog obrade izuzetka bude u svakom slučaju izvršen.

```
Scanner scanner = new Scanner(System.in);
System.out.println("Uneste broj: ");

try {
	int broj = scanner.nextInt();
}catch(InputMismatchException e) {
	System.out.println("Niste uneli broj.");
}finally {
	scanner.close();
}
```

Na ovaj način omogućavamo da se objekat **scanner** uvek zatvori, iako možda u **catch** bloku imamo naredbe koje bi preskočile izvršavanje koda posle obrade izuzetka (**return** , **break** isl.).

Kreiranje novih tipova izuzetaka

Ukoliko je potrebno, moguće je kreirati naš tip izuzetka, nasleđivanjem klase Exception.

```
public class NeispravnaOcenaException extends Exception {

	public NeispravnaOcenaException() {
		super("Ocene moraju biti brojevi između 5 i 10");
	}
}
```

Kreiranje (bacanje) izuzetka se vrši ključnom rečju **throw**. Sve funkcije koje u svom telu imaju **throw**, moraju ili obaviti reakciju na pojavu izuzetka **try-catch** blokom, ili u nazivu proglasiti da je moguća pojava izuzetka dodavanjem ključne reči **throws**.

```
public void proveriOcene() throws NeispravnaOcenaException {
	int[] ocene = new int[] {10, 7, 4, 8};

	for (int i = 0; i < ocene.length; i++) {
		if(ocene[i] < 5  || ocene[i] > 10) {
			throw new NeispravnaOcenaException();
		}
	}
}
```

U ovom slučaju, potrebno je obaviti rukovanje izuzetkom u trenutku poziva metode.

```
try {
	proveriOcene();
} catch (NeispravnaOcenaException e) {
	e.printStackTrace();
}
```

Rad sa datotekama

Za upis i čitanje iz datoteka koristićemo odgovarajuće reader i writer Java klase. S obzirom da želimo da radimo nad podacima koji su sačuvani u datotekama na disku, koristićemo klase **`FileReader`** i **`FileWriter`** za pisanje odnosno čitanje.

Čitanje tekstualnih datoteka datoteka

Čitanje započinje kreiranjem objekta klase **`File`** (paket `java.io`) sa putanjom do željene datoteke. Ukoliko se datoteka nalazi u paketu **`txt`** unutar **`src`** repozitorijuma, **`File`** objekat možemo kreirati na sledeći način:

```
File file = new File("src/txt/datoteka.txt");
```

Nakon toga, potrebno je kreirati odgovarajuće **`BufferedReader`** i **`FileReader`** objekte za pristup kreiranoj datoteci.

```
BufferedReader reader = new BufferedReader(new FileReader(file));
```

Sada, uz pomoć metode **`readLine()`** možemo čitati liniju po liniju iz tekstualne datoteke:

```
String line;

while((line = reader.readLine()) != null) {
	System.out.println(line);

}
```

Na kraju je potrebno zatvoriti **`BufferedReader`** instancu:

```
reader.close();
```

Pisanje u datoteke

Upisivanje podataka u tekstualnu datoteku se vrši na sličan način kao I čitanje, sa tom razlikom što se za svrhe pisanja koriste klase **`BufferedWriter`** i **`FileWriter`**.

Primer:

```
try {
	File file = new File("src/txt/korisnici.txt");

	BufferedWriter writer = new BufferedWriter(new FileWriter(file));
	writer.write("Prva Linija\nDruga linija");
	writer.close();
} catch (IOException e) {
	e.printStackTrace();
}
```

NAPOMENA Metoda **`write`** će prebrisati ceo postojeći sadržaj datoteke novim. Ukoliko ovo nije željeno ponašanje, potrebno je omogućiti mehanizam čuvanja starih podataka.

Zadaci

1. Kreirati program koji će omogućiti prijavu (Login) korisnika na osnovu unetog korisničkog imena i šifre. Podaci o korisnicima koji mogu da se prijave na sistem se čuvaju u tekstualnoj datoteci koja sadrži sledeće podatke: ime, prezime, korisničko ime i šifru.

   Svaka linija u datoteci sadrži podatke o samo jednom korisniku, dok su posebni podaci razdvojeni specijalnim delimiterskim karakterom.

   Primer datoteke:

   ```
   Petar|Petrovic|petarp|12345
   Jovana|Jovanovic|jovanaj|54321
   ```

   Na početku rada, program od korisnika traži da unese svoje korisničko ime i šifru i nakon toga ispisuje poruku da li je prijava uspešno prošla u zavisnosti od toga da li korisnik sa unesenim korisničkim imenom i šifrom postoji u datoteci.

2. Kreirati program koji će od korisnika tražiti da unese svoje ime, prezime, korisničko ime I šifru I preuzete podatke zapisuje u tekstualnu datoteku u formatu datom u zadatku 1.