**Java zadaci**

**Zadaci**

[1. Priprema novog projekta 1](#_Toc371582988)

[2. Organizacija klasa u fajlove 2](#_Toc371582989)

[3. Ispis vrijednosti u konzolni prozor 3](#_Toc371582990)

[4. Učitavanje vrijednosti sa konzolnog prozora 4](#_Toc371582991)

[4.1 Učitavanje podatka tipa string 4](#_Toc371582992)

[4.2 Učitavanje brojčanih vrijednosti 6](#_Toc371582994)

[5. Rad sa dinamičkim objektima 7](#_Toc371582996)

[5.1 Početna vrijednost varijable 7](#_Toc371582997)

[6. Pravo pristupa 8](#_Toc371582998)

[7. Jednodimenzionalni nizovi 8](#_Toc371582999)

[8. Petlja foreach 9](#_Toc371583000)

[8.1 Primjer korištenja foreach-petlje sa nizom 9](#_Toc371583001)

[8.2 Primjer korištenja foreach-petlje sa kolekcijom 9](#_Toc371583002)

[8.3 Primjer korištenja foreach-petlje sa parametrom funkcije 10](#_Toc371583003)

[8.4 Napomene za foreach-petlju 10](#_Toc371583004)

[8.5 Zadaci sa nizovima 10](#_Toc371583005)

[9. Kolekcije 12](#_Toc371583006)

[9.1 Zadaci sa kolekcijama 13](#_Toc371583007)

# Priprema novog projekta

Novi Java projekt možete pripremiti na slijedeći način:

1. **Otvorite razvojno okruženje Eclipse.**

*Pokrenite eclipse.exe (napravite shortcut na desktop-u)*

1. **Odaberite jedan folder kao *workspace-folder***

U ovom folderu će se nalaziti postavke Eclipse-a i radni projekti

1. **Otvorite novi projekat**

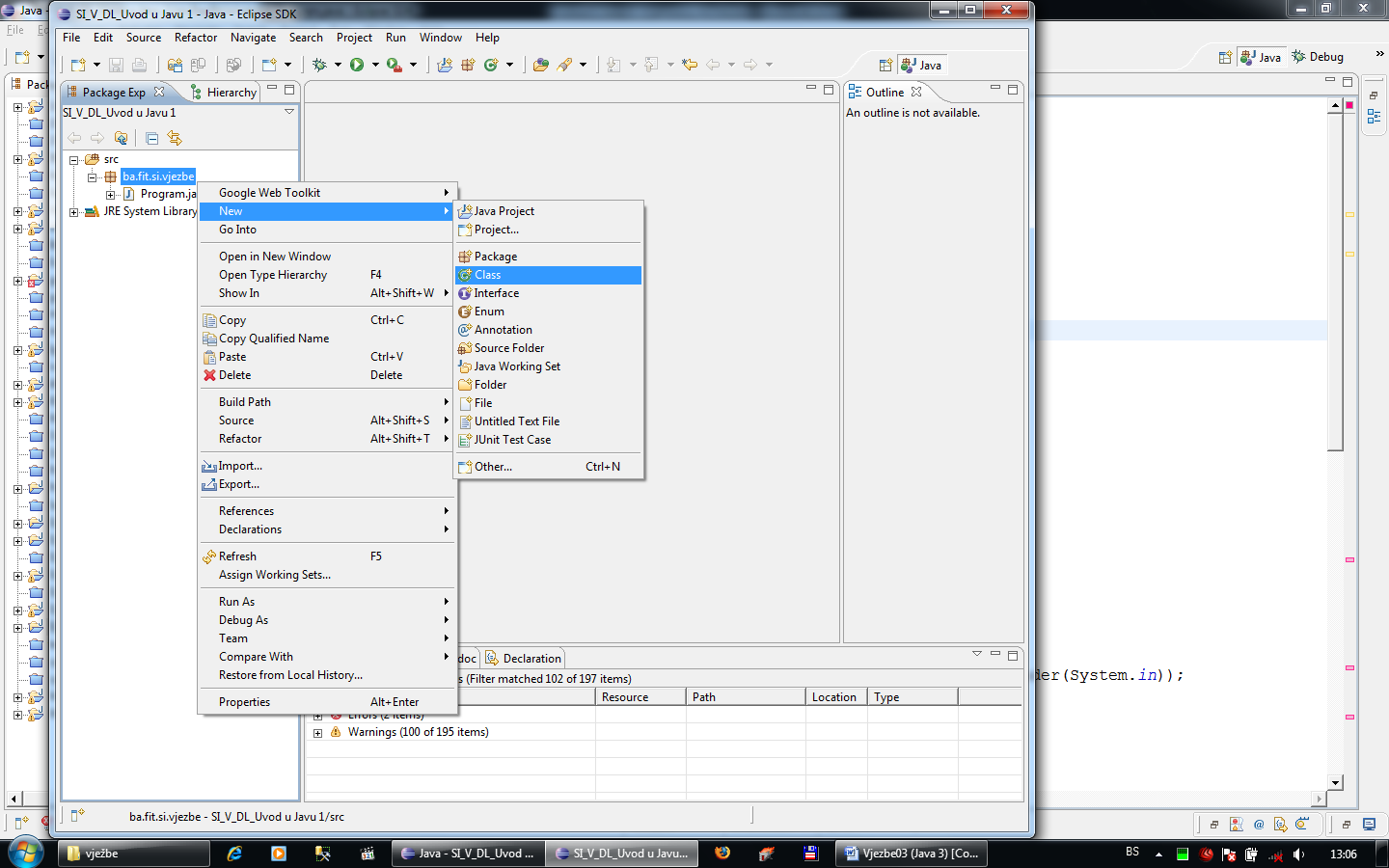
File -> New -> Java Project

1. **U folder *src* dodajte novi *package*-folder sa nazivom "si.vjezbe"**

Otvorite menij na src folderu pomoću miša (desni klik) i odaberite *New -> Package*

1. **U package "si.vjezbe" dodajte novu klasu sa imenom "TestKlasa"**

Otvorite menij na "si.vjezbe" folderu pomoću miša (desni klik) i odaberite *New -> Class*



Slika 1. Primjer dodavanja nove klase u *Eclipse*-u

1. **Isprobajte code slijedećeg programa.**

Program pokrenite sa opcijom *Run as -> Java Application.*

Nakon što prvi put pokrenete funkciju *main* sa opcijom *Run as -> Java application* onda je moguće istu funkciju pokretati sa CTRL + F11.

<https://www.jetbrains.com/help/idea/creating-running-and-packaging-your-first-java-application.html>

# Organizacija klasa u fajlove

U programskom jeziku Java jedna javna klasa se implementiraju u zaseban istoimeni fajl. Klasa NekaKlasa se smješta u fajl NekaKlasa.java (kompajlirana klasa će se zvati NekaKlasa.class), dok u programskim jezicima C++ i C# jedan fajl može imati više klasa.

Klase se smještaju u *package*-e, koji predstavljaju logičku grupu klasa, koji su na fajl-sistemu čine foldere i podfoldere.

Package možete porediti sa putanjom foldera za neki fajl. Pogledajte sljedeći primjer.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **putanja foldera** | **+** | **naziv fajla** | **=** | **puna putanja do fajla** |
| c:\folder1\folder2\ | fajl.txt | c:\folder1\folder2\fajl.txt |
|  | | | | |
| **package** | **+** | **naziv klase** | **=** | **puni naziv klase** |
| Java.util. | ArrayList | Java.util.ArrayList |

Slika 2. Analogija između package-a i putanje foldera

Dodavanje *package*-a se vrši na sljedeći način:

1. desni klik na folder „src“
2. New -> Package
3. Upišite ime *package*-a -> Finish

Dodavanje klasa se vrši na sljedeći način:

1. desni klik na ime *package*
2. *New* -> *Class*
3. Upišite ime nove klase -> *Finish*

# Ispis vrijednosti u konzolni prozor

Ispis na konzolu se vrši pomoću statičkih funkcija **print** i **println** koje se nalaze u klasi **Console**. Slijedi primjer ispisa stringova.

|  |  |
| --- | --- |
| 1:  2:  3:  4:  5:  6: | **public** **static** **void** main(String[] args)  {  String naziv = "FIT";  String grad = "Mostar";  System.*out*.**println**("Fakultet " + naziv + " se nalazi u gradu " + grad + "!");  } |

Slijedi primjer ispisa brojčanih podataka.

|  |  |
| --- | --- |
| 1:  2:  3:  4:  5:  6:  7: | **public** **static** **void** main(String[] args)  {  String naziv = "FIT";  Integer godina = 2007;  System.*out*.println("Copyright by " + naziv + ", " + godina**.toString()**);  } |

U prethodnom primjeru možemo izostaviti poziv funkcije **toString**. Sljedeći primjer je ekvivalentan kao prethodni.

|  |  |
| --- | --- |
| 1:  2:  3:  4:  5:  6:  7: | **public** **static** **void** main(String[] args)  {  String naziv = "FIT";  Integer godina = 2007;  System.*out*.println("Copyright by " + naziv + ", " + godina);  } |

Šta će se ispisatu u sljedećem programu?

|  |  |
| --- | --- |
| 1:  2:  3:  4:  5:  6:  7:  8:  9: | **public** **static** **void** main(String[] args)  {  String naziv = "FIT";  Integer godina = 2007;  System.*out*.println("Copyright by " + naziv + ", " + godina.toString());  System.*out*.println("Copyright by " + naziv + ", " + (godina + 1));  System.*out*.println("Copyright by " + naziv + ", " + godina + 1);  } |

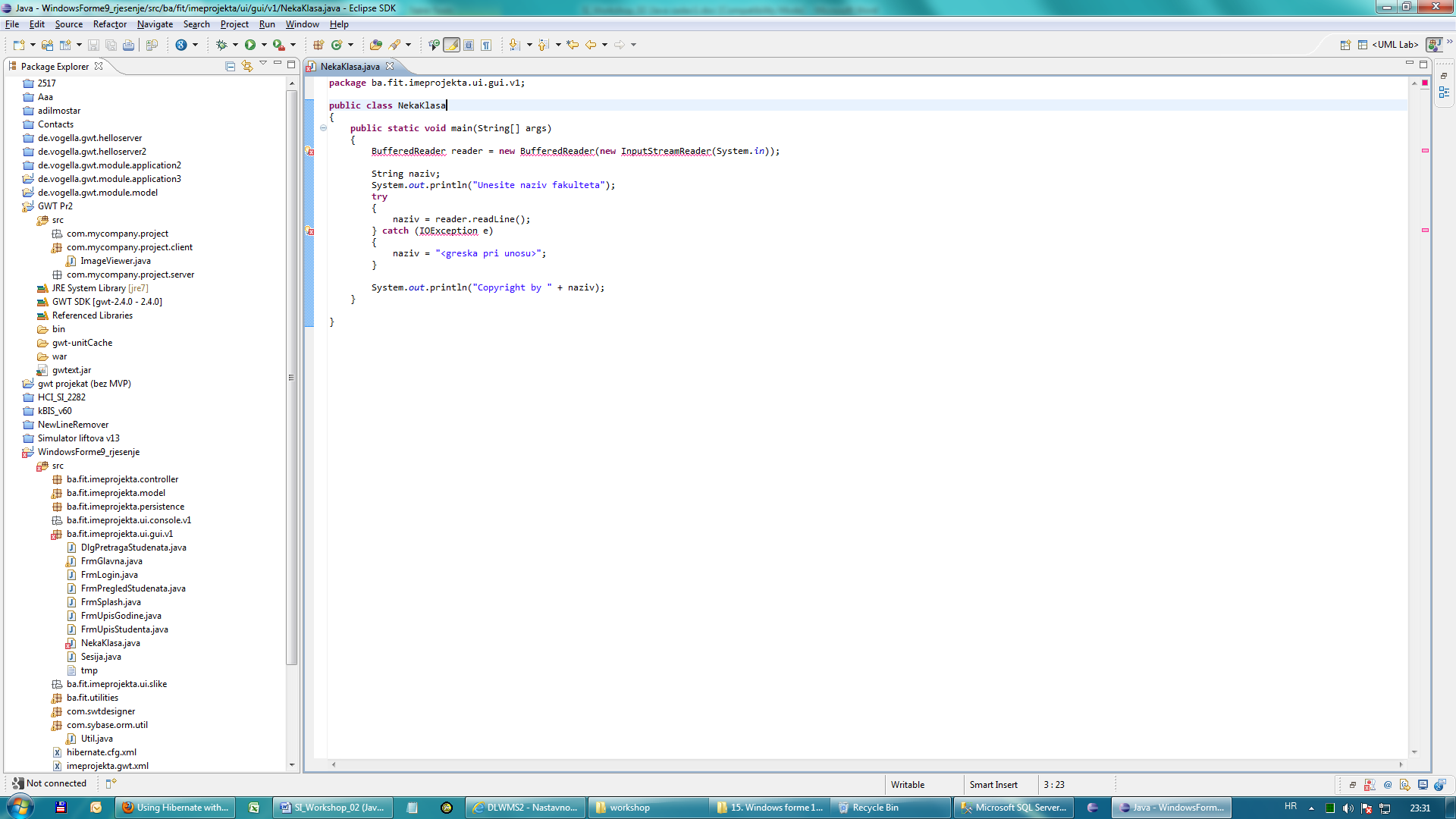
# Učitavanje vrijednosti sa konzolnog prozora

## Učitavanje podatka tipa string

Učitavanje sa tastature se vrši pomoću funkcija **readLine** koje se nalaze u klasi **BufferedReader**. Slijedi primjer učitavanja stringova.

|  |  |
| --- | --- |
| 1:  2:  3:  4:  5:  6:  7:  8:  9:  10:  11:  12:  13:  14:  15:  16: | **public** **static** **void** main(String[] args)  {  BufferedReader reader = **new** BufferedReader(**new** InputStreamReader(System.*in*));  String naziv;  System.*out*.println("Unesite naziv fakulteta");  **try**  {  naziv = reader.readLine();  } **catch** (IOException e)  {  naziv = "<greska pri unosu>";  }    System.*out*.println("Copyright by " + naziv);  } |

*Napomena:* Ako kopirate ovaj dio koda u novu klasu, pojavit će greške kao na sljedećoj slici.

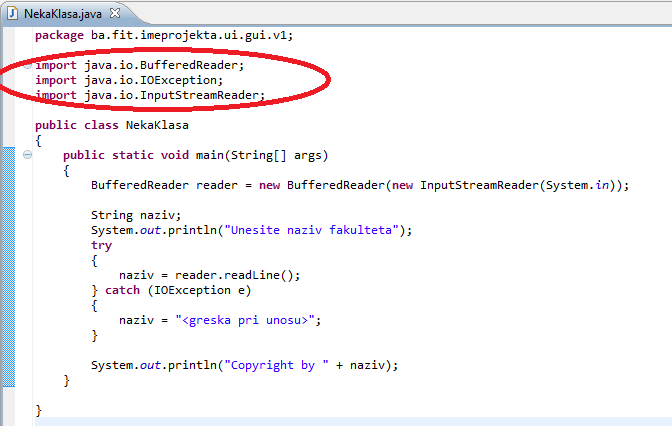


Slika 3. Java program bez importovanih potrebnih paketa

Problem nastaje jer nisu importovani potrebni paketi ili klase. Njihovo uključivanje je neophodno kako bi kompajler mogao pronaći klase. Ovaj problem se može automatski riješiti pomoću opcije *Organize packages*, koja se može pozvati na jedan od sljedeća dva načina:

* (desni klik na kode) -> Source -> Organize packages
* kombinacija tipki CTRL + SHIFT + O

Nakon korištenja ove opcije, Eclipse je automatski dodao linije koda koje uključuju potrebne pakete (pogledati sliku 4)



Slika 4. Java program (importovani paketi)

Funkcija **readLine** se mora nalaziti u **try-catch** blocku.

### Implementacija „pomoćne“ klasa za čitanje stringa

S obzirom da je veoma često potrebu učitati neki string sa tastature, preporučuje se implementacija jedne posebne klase sa statičkom funkcijom za učitavanje stringa.

Pogledajte slijedeću klasu **KonzolniAlati**.

|  |  |
| --- | --- |
| 1:  2:  3:  4:  5:  6:  7:  8:  9:  10:  11:  12:  13:  14:  15:  16:  17:  18:  19:  20:  21:  22: | **import** java.io.BufferedReader;  **import** java.io.IOException;  **import** java.io.InputStreamReader;  **public** **class** KonzolniAlati  {  **public** **static** String ucitajString()  {  BufferedReader reader = new BufferedReader(  new InputStreamReader(System.*in*));  String result;  **try**  {  result = reader.readLine();  } **catch** (IOException e)  {  result = "";  }  **return** result;  }  } |
| 1:  2:  3:  4:  5:  6:  7:  8:  9:  10:  11:  12:  13:  14:  15: | **public** **class** Program  {  **public** **static** **void** main(String[] args)  {  System.*out*.println("Unesite naziv fakulteta");  String naziv = KonzolniAlati*.ucitajString*();    System.*out*.println("Unesite naziv grada");  String grad = KonzolniAlati*.ucitajString*();  System.*out*.println("Copyright by " + naziv + ", " + grad);  }  } |

## Učitavanje brojčanih vrijednosti

Povratna vrijednost funkcije **ucitajString** je podatak tipa string. Ako želimo učitati brojčane vrijednosti moramo string pretvoriti (engl. „parse“) u neku brojčanu vrijednost. Slijedi primjer.

|  |  |
| --- | --- |
| 1:  2:  3:  4:  5:  6:  7:  8:  9:  10:  11:  12:  13: | **public** **class** Program  {  **public** **static** **void** main(String[] args)  {  **int** godina;  String s;    System.*out*.println("Unesite godinu:");  s = KonzolniAlati*.uctajString*();  godina = Integer.*parseInt*(s);  //...  }  } |

### Implementacija „pomoćne“ klasa za čitanje integera

S obzirom da je veoma često potrebu učitati neki integer sa tastature, preporučuje se implementacija statičke funkcije za učitavanje integera.

Pogledajte dorađenu klasu **KonzolniAlati**.

|  |  |
| --- | --- |
| 1:  2:  3:  4:  5:  6:  7:  8:  9:  10:  11:  12:  13:  14:  15:  16:  17:  18:  19:  20:  21:  22:  23:  24:  25:  26:  27:  28:  29: | **import** java.io.BufferedReader;  **import** java.io.IOException;  **import** java.io.InputStreamReader;  **public** **class** KonzolniAlati  {  **public** **static** String ucitajString()  {  BufferedReader reader = new BufferedReader(  new InputStreamReader(System.*in*));  String result;  **try**  {  result = reader.readLine();  } **catch** (IOException e)  {  result = "";  }  **return** result;  }  **public** **static** Integer ucitajInteger()  {  **final** String s = *ucitajString*();  **final** Integer i = Integer.*parseInt*(s);  **return** i;  }  } |
| 1:  2:  3:  4:  5:  6:  7:  8:  9: | **public** **class** Program  {  **public** **static** **void** main(**final** String[] args)  {  System.*out*.println("Unesite godinu:");  **int** godina = KonzolniAlati.*ucitajInteger*();  //...  }  } |

# Rad sa dinamičkim objektima

Svi objekti se alociraju u dinamičkoj memoriji. Dealociranje se u programskom jeziku Java vrši automatski.

Kopiranje objekata klase vrši se **uvijek po refernci**.

|  |
| --- |
| **public** **static** **void** main(String[] args)  {  NekaKlasa a = **new** NekaKlasa();  NekaKlasa b = a;  a.cijena = 20;  b.cijena = 23;  System.*out*.println(a.cijena); //ispis: 23  } |

* Ovdje se varijable a i b ponašaju kao **pokazivači** iz jezika C++
* U prethodnom primjer a i b su referenca na isti objekat. Tako će izmjena preko varijable b uticati i na varijablu a. (Program će ispisati vrijednost „23“.)

## Početna vrijednost varijable

Početna vrijednost varijabli je „neinicijalizirana vrijednost“. Svakoj varijabli morate dodijeliti neku početnu vrijednost (npr. null). Slijedeći program će prijaviti grešku.

|  |
| --- |
| **public** **static** **void** main(String[] args)  {  NekaKlasa p;  **if** (p == **null**) // **error: The local variable p may not have been initialized**  System.*out*.println("p je null");  **else**  System.*out*.println("p nije null");  } |

# Pravo pristupa

U jeziku Java, kao i u jeziku C#, morate prije svakog člana klase ili funkcije napisati da li je riječ o javnom, privatnom, zaštićenom ili defaultnom članu.

|  |  |
| --- | --- |
| 1:  2:  3:  4:  5:  6:  7:  8:  9:  10:  11:  12:  13:  14:  15:  16: | **public** **class** NekaKlasa  {  **private** **int** NekiPrivatniClan;    **public** **int** NekiJavniClan;    **private** **void** NekaPrivatnaFunkcija()  {  //...  }  **public** **void** NekaJavnaFunkcija()  {  //...  }  } |

Ako nije definisan pristup za varijablu ili funkciju, on će se smatrati kao *default*, a to znači da je moguće pristupati članu samo iz istog *package*-a.

U programskom jeziku Java, za razliku od C++-a, se ne stavlja tačka-zarez (;) na kraj klase.

Ako definišite funkciju kao statičku onda toj funkciji pristupamo preko klase a ne preko objekta. Primjer statičke funkcije su funkcije printLine koje se nalaze u klasi System.out.

# Jednodimenzionalni nizovi

Slijedi primjer deklarisanja i korištenja niza za primitivne podatke: **char, byte, short, int, long, boolean, float, double, char**

|  |  |
| --- | --- |
| 1:  2:  3:  4:  5: | **int**[] n = **new** **int**[5];  n[0] = 123;  n[1] = 124;  n[2] = 125;  //... |

Slijedi primjer deklarisanja niza objekata neke klase.

*Primjer 1: Alokacija objekata niza bez konstruktora*

|  |  |
| --- | --- |
| 1:  2:  3:  4:  5: | NekaKlasa[] n = **new** NekaKlasa[10];  n[0] = **new** NekaKlasa("FIT");  n[1] = **new** NekaKlasa("Biblioteka");  //... |

*Primjer 2: Alokacija objekata niza bez konstruktora*

|  |  |
| --- | --- |
| 1:  2:  3:  4:  5:  6:  7:  8: | NekaKlasa[] n = **new** NekaKlasa[10]; //U C++ ovo predstavlja niz od 10 pokazivača na NekaKlasa  n[0] = **new** NekaKlasa(); //<-- Nemojte zaboraviti alocirati objekat u dinamičkoj memoriji !!!  n[0].nekiClan = "FIT";  n[1] = **new** NekaKlasa(); //<-- Nemojte zaboraviti alocirati objekat u dinamičkoj memoriji !!!  n[1].nekiClan = "Biblioteka";  //... |

Svaki niz ima javni član length, koji predstavlja veličinu niza. Evo jednog primjera.

|  |  |
| --- | --- |
| 1:  2:  3:  4:  5:  6:  7: | **void** nekaFunkcija(NekaKlasa[] n)  {  **for** (**int** i = 0; i < n.length; i++)  {  //...  }  } |

# Petlja *foreach*

## Primjer korištenja *foreach*-petlje sa nizom

|  |  |
| --- | --- |
| 1:  2:  3:  4:  5:  6:  7:  8:  9:  10:  11: | **public** **static** **void** main(String[] args)  {  **int**[] n = **new** **int**[30];  n[0] = 11;  n[1] = 12;    **for** (**int** i : n)  {  System.*out*.println(i);  }  } |

## Primjer korištenja *foreach*-petlje sa kolekcijom

|  |  |
| --- | --- |
| 1:  2:  3:  4:  5:  6:  7:  8:  9:  10:  11: | **public** **static** **void** main(String[] args)  {  NekaKlasa[] n = **new** NekaKlasa[30];  //ovdje je potrebno alocirati elemente niza    **for** (NekaKlasa i : n)  {  i.nekiClan = "neka vrijednost";  }  } |

Tijelo petlje (linija br. 9) će se pozvati automatski za svaki član niza n (tj. 30 puta). Varijabla i će poprimiti vrijednost svakog član niza n.

Varijabla i mora biti isti tip podatka kao niz n.

## Primjer korištenja foreach-petlje sa parametrom funkcije

|  |  |
| --- | --- |
| 1:  2:  3:  4:  5:  6:  7: | **void** nekaFunkcija(NekaKlasa[] m)  {  **for** (NekaKlasa i : m)  {  i.nekiClan = "neka vrijednost";  }  } |

## Napomene za foreach-petlju

*Foreach*-petlja predstavlja „*forward-only*“ i „*read-only*“ pristup. Petlju *foreach* ne možete koristiti za deklarisanje objekata u dinamičkoj memoriji preko interativne varijabli (u ovim primjerima to je varijabla i). Alokacija objekta u narednom primjeru (linija broj 5) nije ispravna. Varijabla (pokazivač) **i** se ne može usmjerili na objekat koji je tek alociran.

|  |  |
| --- | --- |
| 1:  2:  3:  4:  5:  6:  7: | NekaKlasa[] n = **new** NekaKlasa[30];  **for** (NekaKlasa i : n)  {  i = **new** NekaKlasa(); **// "i = nesto..." nije dozvoljeno**  i.nekiClan = "neka vrijednost"; **// "i.nekiClan = nesto..." je uredu**  } |

Prethodnu *foreach*-petlju je potrebno zamijeniti običnom *for*-petljom. Slijedi ispravak prethodnog programa.

|  |  |
| --- | --- |
| 1:  2:  3:  4:  5:  6:  7: | NekaKlasa[] n = **new** NekaKlasa[30];  **for** (**int** i = 0; i < n.length; i++)  {  n[i] = **new** NekaKlasa();  n[i].nekiClan = "neka vrijednost";  } |

## Zadaci sa nizovima

**Zadatak 1**

Napravite program sljedeći navedene zahtjeve:

1. neka korisnik sa tastature vrši unos cijelog broja x
2. deklarišite niz n tipa *int* veličine x
3. učitajte vrijednost niza sa tastature koristeći *for* petlju
4. ispišite vrijednost niza koristeći *foreach* petlju. *(Objašnjenje foreach petlje slijedi u nastavku.)*

*Rješenje se nalazi u Java projektu* ***\_code.zip*** *kao prilog ovog dokumenta). Importujte projekat u Eclipse: File -> Import -> General -> Existing Project into Workspace -> Archive file -> odabrati* ***\_code.zip*** *-> Finish . Otvorite projekt -> package zadatak1*

**Zadatak 2**

Napravite prazan projekat i dodajte sljedeće:

1. Definišite klasu Kupac:

* Dodajte privatne članove **ime** i **email** (neka svi članovi budu tipa String)
* Dodajte javne *gettere* i *settere* za sve članove.
* Prilikom inicijalizacije vrijednosti e-mail adrese potrebno je odsjeći prazne razmake prije i poslije adrese i postaviti vrijednost „malim“ slovima. (pročitajte pomoć na kraju zadatka)

1. Definišite novu klasu Zadatak2

* Dodajte statičku funkciju ucitaj
  + 1. funkcija treba da ima jedan parametar (koji predstavlja niz tipa Kupac)
    2. funkcija treba učitati podatke sa tastature u niz.
    3. koristite *for*-petlju (ovdje nije moguće koristiti petlju *foreach*)
* Definišite statičku funkciju ispis
  + 1. funkcija treba da ima jedan parametar (koji predstavlja niz tipa Kupac)
    2. funkcija treba ispisati podatke o kupcima na ekran (koristiti *foreach* petlju)
* Definište funkciju main
  + 1. definišite niz od 5 kupaca
    2. učitajte podatke za kreirani niz pomoću funkcije ucitaj
    3. ispišite podatke za kreirani niz pomoću funkcije ispis

*Rješenje se nalazi u Java projektu* ***\_code.zip*** *-> package zadatak2*

*Pomoć:*

* Funkcija trim nad Stringom vraća string sa uklonjenim praznim razmacima sa početka i kraja stringa.
* Funkcija toLowerCase nad *stringom* vraća *string* čiji su svi karakteri „mala“ slova.
* Slijedi primjer poziva funkcija.

|  |  |
| --- | --- |
| 1:  2:  3:  4:  5:  6:  7:  8:  9:  10: | **public** **static** **void** main(String[] args)  {  String org = " fit Mostar ";  String a = org.trim();  String b = org.toLowerCase();  System.*out*.println("a[" + a + "]"); //a[fit Mostar]  System.*out*.println("b[" + b + "]"); //b[ FIT MOSTAR ]  } |

# Kolekcije

Prednosti korištenja kolekcija u odnosu nizove su sljedeće:

* nije potrebno prilikom deklaracije *liste* navoditi veličinu niza (lista se automatski širi), primjer deklaracije liste n tip String:

|  |
| --- |
| ArrayList<String> n = **new** ArrayList<String>(); |

* moguće je dodavanje i uklanjanje elemenata se bilo koje pozicije
  + funkcija n.**add**(i, "FIT") dodaje objekat na poziciju i (lista se širi)
  + funkcija n.**add**("FIT")dodaje objekat na zadnju poziciju (lista se širi)
  + funkcija n.**remove**(i) briše objekat sa pozicije i (lista se smanjuje)
  + funkcija n**.get(**i**)** pristupamo objektu na poziciji i
* pomoću funkcije subList moguće je izdvojiti dio niza
  + „noviNiz = n.subList(i, j)“ izdvaja (kopira) iz niza n objekte sa indeksom i, (i+1), (i+2), ..., (j-1), j
* pomoću funkcije n.contains("FIT") moguće je pretražiti listu
* pomoću funkcije n.size() možemo saznati broj elementa u listi

Slijedi primjer rada korištenja generičkih listi.

|  |  |
| --- | --- |
| 1:  2:  3:  4:  5:  6:  7:  8:  9:  10:  11:  12:  13:  14:  15:  16:  17: | **public** **static** **void** main(String[] args)  {  ArrayList<String> n = **new** ArrayList<String>();    **for** (**int** i = 0; i < 10; i++)  {  System.*out*.println("Unesite neku riječ");  String str = *ucitajString*();  n.add(str);  }    **for** (**int** i = 0; i < n.size(); i++)  {  String str = n.get(i);  System.*out*.println(str);  }  } |

Slijedi isti primjer, ali sa upotrebom *foreach*-petlje (u ispisu).

|  |  |
| --- | --- |
| 1:  2:  3:  4:  5:  6:  7:  8:  9:  10:  11:  12:  13:  14:  15:  16: | **public** **static** **void** main(String[] args)  {  ArrayList<String> n = **new** ArrayList<String>();    **for** (**int** i = 0; i < 10; i++)  {  System.*out*.println("Unesite neku riječ");  String str = *ucitajString*();  n.add(str);  }    **for** (String str : n)  {  System.*out*.println(str);  }  } |

Slijedi primjer koji ispisuje elemente sa indeksom 2,3,4,5 sa upotrebom *foreach*-petlje i funkcije subList.

|  |  |
| --- | --- |
| 1:  2:  3:  4:  5:  6:  7:  8:  9:  10:  11:  12:  13:  14:  15:  16: | **public** **static** **void** main(String[] args)  {  ArrayList<String> n = **new** ArrayList<String>();    **for** (**int** i = 0; i < 10; i++)  {  System.*out*.println("Unesite neku riječ");  String str = *ucitajString*();  n.add(str);  }    **for** (String str : n.subList(2, 5))  {  System.*out*.println(str);  }  } |

## Zadaci sa kolekcijama

**Zadatak 3**

Prepravite prethodni program tako da umjesto klasičnih nizova koristite generičke *liste* (klasa ArrayList <T>). (Klasa ArrayList je slična klasi vector u jeziku C++.)

*Rješenje se nalazi u Java projektu* ***\_code.zip*** *-> package zadatak3*

**Zadatak 4**

Prepravite prethodni program tako da se korisniku, nakon unosa svih podatak o jednom kupcu, omogući ponovni unos podataka o novom kupcu (npr. ako korisnik unese riječ „da“) .

Ako korisnik, nakon jednog unosa, ne želi unos novog kupca program treba, pomoću funkcije ispis, ispisati sve kupce.

U ovom programu veličina niza nije unaprijed poznata. Koristite, umjesto klasičnih nizova, generičku listu?

*Rješenje se nalazi u Java projektu* ***code.zip*** *-> package zadatak4*