Datoteke upis I čitanje

```
package podaci;

import java.io.*;

public class Datoteke {

	public static void main(String[] args) throws IOException {
		/* jednostavan upis/citanje u datoteku */

		/*upis, ako ne postoji kreira; postojeću sa write prepisuje*/
		System.out.println("Upis....");
		DataInputStream dis = new DataInputStream(System.in);
		FileOutputStream fos = new FileOutputStream("datoteka.txt");
		char c; /*za unos znak po znak; staje na prelazak u novi red*/
		while( (c = (char) dis.read()) != '\n') {
			fos.write(c);
		}

		fos.close();

		/*čitanje postojeće datoteek, -1 ako je kraj datoteke*/
		System.out.println("\nČitanje....");
		FileInputStream fis = new FileInputStream("datoteka.txt");
		int ch;
		while( (ch = fis.read()) != -1 ) {
			System.out.print( (char) ch);
		}

		fis.close();

		/*opšti rad sa postojećo datotekom*/
		System.out.println("\n\nOpšti rad sa datotekom");
		File f = new File("datoteka.txt");
		System.out.println("Dužina datoteke (by): " + f.length());
		System.out.println("Datoteka spremna za izvršavanje? - " +
f.canExecute());
		System.out.println("Datoteka spremna za upisivanje? - " +
f.canWrite());
		System.out.println("Datoteka spremna za čitanje? - " + f.canRead());
		System.out.println("Poslednja promena datoteke (ms): " +
f.lastModified());

		String st = new String("Hello, žaba!");
		System.out.println(st);

	}

}
```

Lambda izraz

funkcijski interfejs

```
package lambda;

public interface Funkcija {

	int kvadrat(int x);

}
```

primena

```
package lambda;

public class MainLambda {

	public static void main(String[] args) {

		int y = 10;
		Funkcija rezultat = (z) -> z*z;

		System.out.println(rezultat.kvadrat(y));

	}
}
```

Generička klasa

generička klasa

```
package generik;

public class Generik<T> {

	T obj;

	Generik(T obj) {
		this.obj = obj;
	}

	public void info() {
		System.out.println("Parametar: " + obj);
		System.out.println("Tip generičke klase (parametar): " +
obj.getClass().getName());
		System.out.println("Tip generičke klase: " + getClass().getName());
	}

}
```

primena + generički metod

```
package generik;

public class MainGenerik {

	//generički metod koji vrši iteraciju kroz niz
	public static<T> void stampaj(T[] n) {
		for(T el : n)
			System.out.print(el + " ");
		System.out.println(" ");
	}

	public static void main(String[] args) {
		//generička klasa, demonstracija rada

		Generik<Integer> gi = new Generik<Integer>(10);
		gi.info();

		//generički metod, primena
		Integer[] ni = {1,2,3,4,5};
		stampaj(ni);

	}
}
```

Enumeracijski tip – nabarajanje

definicija 1

```
package enumi;

public enum Boje {

		BELA, PLAVA, CRVENA, ZELENA, CRNA, MAGNETA, ZUTA;

}
```

definicija 2

```
package enumi;

public enum Boje2 {
	BELA(30), PLAVA(25), CRVENA(10), ZELENA(10), CRNA(5), MAGNETA(10),
ZUTA(10);

	private int udeo;

	Boje2(int udeo) {
		this.udeo = udeo;
	}

	int getUdeo() {
		return udeo;
	}


}
```

primena

```
package enumi;

public class MainBoje {

	public static void main(String[] args) {

		Boje plava = Boje.PLAVA;
		Boje crvena = Boje.CRVENA;
		Boje bela = Boje.BELA;

		System.out.println(plava);


		//iteracija
		for(Boje b : Boje.values()) {
			System.out.print(b + " ");
		}
		System.out.println("");
		//primena metoda nad enum tipom, ugrađeni metodi
		// poređenje po vrednosti: false ako su različite vrednosti
```

```
            System.out.println("Plava Vs Bela (jednako): " +
plava.equals(bela));
            // poređenje po lokaciji: 0 ista lokacija, 1 - nalazi se iza, -1
nalazi se ispred
            System.out.println("Plava Vs Bela (poređenje lokacije): " +
plava.compareTo(bela)); // 1 plava je iza bele
            System.out.println("Plava Vs Crvena (poređenje lokacije): " +
plava.compareTo(crvena)); // -1 plava je ispred crvene



            //enum sa parametrom
            Boje2 plava2 = Boje2.PLAVA;
            System.out.println("\nprocenat učešća "+ plava2 + " boje: " +
plava2.getUdeo());

            //iteracija kroz enum sa parametrima
            System.out.println("Pozicija Boja Udeo");
            for(Boje2 b : Boje2.values()) {
                  System.out.println("    " + b.ordinal() + ".   " + b + " " +
b.getUdeo() + "% ");
            }
      }

}
```

Liste – osnovno

```
package liste;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
import java.util.ListIterator;

public class MainListe {

	public static void main(String[] args) {
		// rad sa listama

		List<Integer> alista = new ArrayList<Integer>();
		List<Integer> llista = new LinkedList<Integer>();

		//punjenje liste, ovde o jednom trošku obe liste
		for(int i=1; i<=5; i++) {
			alista.add(i*10);
			llista.add(i*10);
		}

		//itaracija kroz liste, for napredna
		for(int a : alista) {
			System.out.print(a + " "); //arraylist
		}
		System.out.println("");
		for(int a : llista) {
			System.out.print(a + " "); //linkedlist
		}
		System.out.println("");

		//iterator, demonstracija nad arraylist
		Iterator<Integer> itr = alista.iterator();
		while(itr.hasNext())
			System.out.print(itr.next() + " ");
		System.out.println("");

		//list iterator, demonstracija nad linkedlist
		ListIterator<Integer> ltr = llista.listIterator();
		while(ltr.hasNext())
			System.out.print(ltr.next() + " ");
		System.out.println("");

		//list iterator, demonstracija nad arraylist uz promenu konekcije; vožnja unazad
		ltr = alista.listIterator(alista.size());
		while(ltr.hasPrevious())
			System.out.print(ltr.previous() + " ");
		System.out.println("");


		//array list to array
		Object[] niz = alista.toArray();
		for(Object n : niz)
			System.out.println(n);
	}

}
```

Višenitno i konkuretno programiranje

I deo: ovo je OK

kreiranje niti1

```
package niti;

public class Arun implements Runnable {

	@Override
	public void run() {
		// ono što nit radi
		for(int i=0; i<5; i++)
			System.out.println(i + ".Sporedna nit A - implementacija
interfejsa Runnable .... ");
		System.out.println(Thread.currentThread().getName()); //vraća
podatke trenutne niti, ne mora da bude glavna
	}

}
```

kreiranje niti2

```
package niti;

public class Athread extends Thread {

		public Athread(String s) {
			super(s); //s = ime niti; nasleđivanje Thread konstruktora
		}

		public void run() {
			// ono što nit radi
			for(int i=0; i<5; i++)
				System.out.println(i + ".Sporedna nit A - proširenje
klase Thread .... ");
			System.out.println(Thread.currentThread().getName()); //vraća
podatke trenutne niti, ne mora da bude glavna
		}
}
```

primena niti (bez postavljanja prioriteta i usporavanja)

```
package niti;

public class MainSve {

	public static void main(String[] args) throws Exception {
		// rekapitulacija niti

		//objekat glavne niti
		Thread tMain = Thread.currentThread();
		System.out.println("Glavna nit:            " + tMain);
		System.out.println("Ime glavne niti:       " +
Thread.currentThread().getName());
```

```
            System.out.println("Prioritet glavne niti: " +
Thread.currentThread().getPriority());
            System.out.println("Pripadajuća grupa:     " +
Thread.currentThread().getThreadGroup());

            //sporedna nit implementacija Runnable interfejsa
            Arun a = new Arun(); // kreiranje "Runnable" objekta
            Thread ta = new Thread(a,"nit A"); // kreiranje sporedne niti
            System.out.println("\nSporedna nit:          " + ta);
            ta.start(); //pokreće nit

            //sporedna nit proširenje Thread klase
            Athread at = new Athread("nit At"); // kreiranje niti
            System.out.println("\nSporedna nit:          " + at);
            at.start(); // pokretanje niti


      }

}
```

niti razno, nesređeno (prijavljuje grešku)

- promena prioriteta
nit.setPriority(prioritet);
prioritet = 0-10 (min, max); Thread,MIN_VALUE, ..., Thread.NORMAL_VALUE, ...,
Thread.MAX_VALUE (podrazumevano za NORMAL_value je 5)

- privremeno zaustavljanje niti ("usporenje")
nit.sleep(ms);
ms = vrednost izražena u milisekundama
// ide u klasu niti
public class A extends Thread{ public void run() {
	int i = 1; while(i <=10) {
	System.out.println("i: " + i); // sleep(vrednost u milisekundama)
	try { Thread.sleep(100);}
	catch(InterruptedException e) {
		System.out.println("Nit je prekinuta");
		e.printStackTrace();
	}
	i++;
//metoda sleep(milisekunde) privremeno suspenduje ili zaustavlja izvrsavanje niti
//argument metode sleep() je interval cekanja u milisekundama
//metoda sleep moze da izazove izuzetak interruptedException
//desice se izuzetak ukoliko neka druga nit pozeli da prekine ovu suspendovanu nit
} } }

*** niit razbacano, nesređeno

***** nabacano niti

public class Main { public static void main(String[] args) { // TODO Auto-generated method stub A a = new A(); B b = new B(); // Kada odredjenoj niti zelite da dodate prioritet prilikom izvrsavanja pozovite metodu setPriority() koja je clan klase Thread //void setPriority(nivo); //parametrom nivo zadajete nivo prioriteta niti za koju je metoda pozvana //vrednost mora biti u opsegu izmedju MIN_PRIORITY i MAX_PRIORITY //a vrednosti tih konstanti su izmedju 1 i 10 //podrazumevana vrednost iznosi 5 a moze se zadati preko NORM_PRIORITY a.setPriority(Thread.MIN_PRIORITY+5);//1 6 b.setPriority(Thread.MAX_PRIORITY-7);//10-7 = 3 a.start(); b.start(); System.out.println("Kraj glavne niti."); } }

****

kage ana; public class A extends Thread{ public void run() { int i = 1; while(i <=10) { System.out.println("i: " + i); // sleep(vrednost u milisekundama) try { Thread.sleep(100);} catch(InterruptedException e) { System.out.println("Nit je prekinuta"); e.printStackTrace(); } i++; //metoda sleep(milisekunde) privremeno suspenduje ili zaustavlja izvrsavanje niti //argument metode sleep() je interval cekanja u milisekundama //metoda sleep moze da izazove izuzetak interruptedException //desice se izuzetak ukoliko neka druga nit pozeli da prekine ovu suspendovanu nit } } }

***

package ana; public class Main { public static void main(String[] args) { // TODO Auto-generated method stub A a = new A(); a.start(); } }

****

public class A implements Runnable { String ime;//ime niti Thread t; A(String name){ ime = name; t = new Thread(this,ime); t.start(); } public void run() { try { for(int i = 1; i<5; i++) { System.out.println(ime + " = " + i); Thread.sleep(500); } } catch (Exception e) {} } }

****

public class Main { public static void main(String[] args) { // isAlive() utvrdjuje da li se nit jos izvrsava //join() ona ceka da se druga nit zavrsi A a1 = new A("Prva nit"); System.out.println("Prva nit se izvrsava: " + a1.t.isAlive()); A a2 = new A("Druga nit"); System.out.println("Prva nit se izvrsava: " + a2.t.isAlive()); A a3 = new A("Treca nit"); System.out.println("Prva nit se izvrsava: " + a3.t.isAlive()); try { a1.t.join(); a2.t.join(); a3.t.join(); }catch(Exception e) {} System.out.println("Prva nit se izvrsava: " +a1.t.isAlive()); System.out.println("Druga nit se izvrsava: " +a2.t.isAlive()); System.out.println("Treca nit se izvrsava: " +a3.t.isAlive()); } }

****

public class Main { public static void main(String[] args) { // prilikom pokretanja java programa 1 programska nit odmah pocinje da se izvrsava //to je glavna ili inicijalna nit //upravljanje glavnom niti radi se preko objekta tipa //Thread i to pozivom metode currentThread koja pripada klasi currentThread() Thread t = Thread.currentThread(); System.out.println("Trenutna nit: " + t); //getName() = vraca ime niti System.out.println("Naziv niti: " + Thread.currentThread().getName()); //getPriority()= vraca prioritet niti System.out.println("Prioritet niti: " + Thread.currentThread().getPriority()); //ime niti , prioritet, ime grupe kojoj pripada nit //podrazumevano ime glavne niti je main. njen prioritet je 5 po dafaultu sto je takodje

podrazumevana vrednost //treci parametar main je ime grupe niti kojoj ta nit pripada //grupa niti je struktura podataka koja upravlja stanjem kolekcije niti kao celinom

****

package ana; //Nit se moze napraviti za svaki objekat koji implementira interfejs Runnable //da bi klasa implementirala interfejs runnable potrebno je da ima tu jednu metodu koja se zove run() koja je deklarisana u sledecem obliku //public void run() //u metodu run se zadaje kod koji treba da se izrsava u novoj niti //kada se izvrsi metoda run nit se unistava public class A implements Runnable{ public void run() { for (int i = 1; i<=5; i++)
{ System.out.println(Thread.currentThread().getName()); } }

****

//pokretanje nove niti // novu nit pokrecemo tako sto pravimo instancu objekta thread //na 2 nacina //1.implementacija inerfejsa Runnable //2. prosirivajnem klase Thread //u main klasi: //nakon kreiranja klase koja implementira interfejs runnable, //potrebno je napraviti instancu objekta tipa Thread public class Main { public static void main(String[] args) { A a = new A();//** //*** //oblik konstruktora: //Thread(Runnable objekatNit, String imeNiti) //objekatNit je instanca klase koja implementira interfejs Runnable //definise mesto gde zapocinje nit //imeNiti predstavlja ime nove niti koju mi zadajemo Thread t = new Thread(a, "Nit1"); Thread t2 = new Thread(a, "Nit2"); Thread t3 = new Thread(a, "Nit3"); t.start();//metoda start poziva metodu run t2.start(); t3.start(); } }

***

package ana; //nit se moze napraviti i kada klasu thread prosirite novom nasledjenom klasom //a zatim napravimo instancu te klase //nova podklasa mora da redefinise metodu run() public class A extends Thread{ A(String s){ super (s);//poziva se konstruktor Thread sledeceg oblika // public Thread (String imeNiti) } public void run() { for (int i = 1; i<=10; i++)
{ System.out.println(getName()); } } }

***

public class Main { public static void main(String[] args) { // TODO Auto-generated method stub A a1 = new A("Nit1"); A a2 = new A("Nit2"); A a3 = new A("Nit3"); a1.start();//zapocinje izvrsavanje niti pozivanjem njene metode run a2.start(); a3.start(); } }

***

public class A extends Thread{ public void run() { for(int i = 1; i<=10; i++)
{ System.out.println("Nit A: " + i); } System.out.println("Kraj izvrsavanja niti A."); } }

***

public class B extends Thread{ public void run() { for(int i = 1; i<=10; i++)
{ System.out.println("Nit B: " + i); } System.out.println("Kraj izvrsavanja niti B."); } }

****