Računarske mreže i telekomunikacije

Skripta – Niti

Autor: Mihailo Stupar

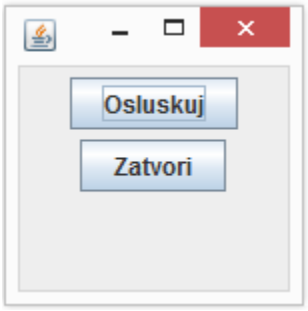
Sadržaj

Primer 1 – Problem jednonitnih programa

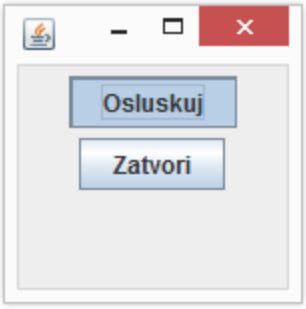
Najbolje je odmah početi sa primerima kako bi se uočio značaj niti. Na početku je dat jednostavan primer koji pokazuje problem kod programa koji imaju samo jednu nit.

```
public class Forma extends JFrame{

	JButton dugme_1;
	JButton dugme_2;

	public Forma() {
		setSize(150, 150);
		setLayout(new FlowLayout());

		dugme_1 = new JButton("Osluskuj");
		dugme_1.setSize(60, 20);
		dugme_1.addActionListener(new OsluskujListener());

		dugme_2 = new JButton("Zatvori");
		dugme_2.setSize(60, 20);
		dugme_2.addActionListener(new ZatvoriListener());

		add(dugme_1);
		add(dugme_2);

		setVisible(true);
	}
}
class OsluskujListener implements ActionListener {
	@Override
	public void actionPerformed(ActionEvent arg0) {
		try {
			ServerSocket serverSoket = new ServerSocket(8765);
			Socket soket = serverSoket.accept();
		} catch (IOException e) {}
	}
}
class ZatvoriListener implements ActionListener {
	@Override
	public void actionPerformed(ActionEvent arg0) {
		System.exit(0);
	}
}
public static void main(String[] args) {

	new Forma();

}
```

Primer1 je jednostavna forma koja sadrži dva dugmeta. Drugo dugme prekida rad programa. Kad se pokrene, forma izgleda kao na prvoj slici. Kada se pritisne dugme Osluskuj, izvšava se `actionPerformed()` koja čeka da se ostvari TCP konekcija na portu 8765. Kao rezultat je izgled forme sa druge slike.



Šta se zapravo dešava i zašto je sad nemoguće kliknuti na dugme Zatvori ili na close?



Kada se kreira GUI aplikacija, tačnije prilikom pozivanja metode `setVisible(true)`, kreira se jedna posebna nit koja se zove Event Dispatch Thread. Ova nit je zadužena za osluškivanje događaja kao što su pritiskanje dugmeta, klik mišem itd. U našem slučaju desio se klik mišem na dugme Osluskuj i `actionPerformed()` je počela sa izvršavanjem, i to u ovoj Event Dispatch niti. Metoda `.accept()` je tzv. "bloker" i kôd neće nastaviti da se izvršava sve dok ne dođe do TCP zahteva. Samim tim, ni metoda `actionPerformed()` se neće završiti, pa ni Event Dispatch nit nije u mogućnosti da osluškuje na nove događaje (klik na dugme Zatvori). Problem bi bio rešen ukoliko bismo Osluškivanje ostvarili u posebnoj niti, tako da nit Event Dispatch može nesmetano da nastavi sa osluškivanjem.

Primer 2 –Nit extends Thread

Niti (Threads) u Javi su objekti klase `Thread`. Klasa `Thread` nasleđuje interfejs `Runnable`, koji ima jednu jedinu metodu - `run()`. Kôd koji se nalazi u metodi `run()` je kôd koji se izvršava pokretanjem nove niti. Pošto niti nisu ništa drugo nego objekti, potrebno je kreirati klasu koja će ili da nasledi `Thread`, ili da implementira `Runnable`.

Ukoliko se odlučimo za nasleđivanje, naša klasa neće moći da nasledi nijednu drugu klasu osim Thread. U Javi je moguće naslediti samo jednu klasu. Sa druge strane, dobijamo implementirane metode klase `Thread`. Ako se odlučimo za implementaciju interfejsa `Runnable`, dobijamo kao zadatak da redefinišemo metodu `run()`, i naša nit može da nasledi bilo koju klasu i da implementira još interfejsa. U Javi je moguće implementirati više interfejsa.

Sledeći primer, pokazuje kreiranje niti pomoću nasleđivanja. Kreiraćemo 5 objekata koji predstavljaju niti i pokrenućemo ih.

```
public class Nit extends Thread {
```

```
	String ime;
	Random random;
	public Nit(String imeNiti) {
		this.ime = imeNiti;
		random  = new Random();
	}
	@Override
	public void run() {
		try {
			while (true) {
				System.out.print(ime + " - ");
				int pauza = random.nextInt(5) * 1000;
				sleep(pauza);
			}
		} catch (InterruptedException e) {
			e.printStackTrace();
		}
	}
}

public static void main(String[] args) {
	new Nit("A").start();
	new Nit("B").start();
	new Nit("C").start();
	new Nit("D").start();
	new Nit("E").start();
}
```

Klasa `Nit` nasleđuje `Thread` i potrebno je da redefinišemo metodu `run()`. Ta metoda već postoji u klasi Thread, pa moramo eksplicitno zatražiti njenu reimplementaciju. To se radi kucanjem `run` i pritiskom ctrl+space, biće vam ponuđeno da je redefinišete. U `main` metodi je kreirano pet objekata klase Nit (tj. Thread) i pozivom metode .start() počinje izvršavanje nove niti. Važno je napomenuti da se pozivom metode start(), izvršava kôd koji se nalazi u run() metodi u posebnoj odvojenoj niti. Ono što se nalazi u run() je beskonačna petlja koja u svakoj iteraciji ispiše ime niti, i zatim ode da ''spava'' na par sekundi. Nit se uspavljuje metodom sleep(), klase Thread. Ta metoda može da baci izuzetak i zbog toga je korišćen try/catch blok. Deo ispisa u konzoli je:

```
- A - B - D - D - C - E - A - D - B - C - C - E - A - B - D - A - E - C - B –
```

Primer 3 – Nit implements Runnable

Treći primer je identičan drugom, samo smo umesto nasledjivanja, koristili implementaciju interfejsa.

```
public class Nit implements Runnable{
	String ime;
	Random random;
	public Nit(String imeNiti) {
		this.ime = imeNiti;
		random  = new Random();
	}
	@Override
	public void run() {
		try {
			while (true) {
				System.out.print(ime + " - ");
				int pauza = random.nextInt(5) * 1000;
				Thread.sleep(pauza);
			}
		} catch (InterruptedException e) {}
	}
}
public static void main(String[] args) {

	new Thread(new Nit("A")).start();
	new Thread(new Nit("B")).start();
	new Thread(new Nit("C")).start();
	new Thread(new Nit("D")).start();
	new Thread(new Nit("E")).start();

}
```

Za kreiranje niti, prvo instanciramo objekat klase `Thread`. Potrebno je kao ulazni parametar u konstruktor ubaciti interfejs `Runnable` (u našem slučaju klasu koja je implementirala taj interfjes), a zatim nad tim objektom pozvati metodu `start()`. Metoda `sleep()` je statička, pa smo mogli da je pozovemo samo nad klasom `Thread`.

Primer 4 – Interrupt

Nit se gasi onda kada se završi metoda `run()`. To je jednostavno učiniti iz same metode, samo je potrebno napisati `return` unutar `run()`. Međutim, ako je potrebno neku nit prekinuti iz neke druge klase ili metode, to se čini pomoću jednog `boolean`-a, koji označava stanje niti. Sledeći primer pokazuje prekidanje niti:

```
public class Nit implements Runnable {
	String ime;
	Random random;
	public Nit(String ime) {
		this.ime = ime;
		random = new Random();
	}
	@Override
	public void run() {
		try {
			while (!Thread.currentThread().isInterrupted()) {
				System.out.print(ime + "-");
				int pauza = random.nextInt(5) * 1000;
				Thread.sleep(pauza);
			}
		} catch (InterruptedException e) {}
	}
}
```

```
public static void main(String[] args) throws Exception{

	Thread A = new Thread(new Nit("A")); A.start();
	Thread B = new Thread(new Nit("B")); B.start();
	Thread C = new Thread(new Nit("C")); C.start();
	Thread D = new Thread(new Nit("D")); D.start();
	Thread E = new Thread(new Nit("E")); E.start();
	Thread.sleep(3000);     // uspavaj glavnu nit 3s
	C.interrupt(); System.out.print("|C|"); // prekini nit "C"
	Thread.sleep(3000);     // uspavaj glavnu nit 3s
	D.interrupt(); System.out.print("|D|"); // prekini nit "D"
	Thread.sleep(3000);     // uspavaj glavnu nit 3s
	E.interrupt(); System.out.print("|E|"); // prekini nit "E"

}
```

Pogledajmo prvo `main` metodu. Kreirali smo pet niti i sve ih startovali. Zatim smo glavnu nit uspavali na 3 sekunde i tako pustili sve niti da se nesmetano izvršavaju. Posle te tri sekunde smo pozvali metodu `interrupt()` nad niti C i opet uspavali glavnu nit. Kasnije smo pozvali istu metodu nad niti D i E. Metoda `interrupt()` ne radi ništa drugo, osim što postavlja `boolean` promenljivu `interrupted` klase Thread na true. Za sada nit još nije prekinuta. Zbog toga smo u

klasi Nit, while(true) petlju zamenili petljom sa uslovom. Pozivom Thread.currentThread() dobijamo referencu na konkretan Thread koji se izvršava (iako smo koristili interfejs na ovaj način, dobijamo metodu koja ima `Thread`), i zatim metodom `isInterrupted()` proveravamo vrednost booleana interrupted. Ako je on postavljen na true, iskočiće iz petlje i sledeći korak je završetak metode `run()`, tj. gašenje niti. Ovde treba biti pažljiv, jer ukoliko pokušamo da postavimo vrednost `interrupted` na `true` ž, dok se nit nalazi u stanju spavanja (nad njom je pozvana metoda `sleep()`), baciće se `InterruptedException`. Zbog toga se kôd u while petlji nalazi unutar `try/cacth` bloka. Mi samo uhvatimo izuzetak (ali izvan `while` petlje), tako da se posle toga metoda run završava i nit gasi. Ispis u konzoli izgleda:

```
B-E-D-C-A-E-D-B-E-|C|A-B-D-D-E-B-|D|B-A-E-B-|E|A-B-A-B-B-B-A-A-
```

Prvo se pojavljuju sva slova, zatim se nit C gasi i to slovo se više ne pojavljuje. Zatim, nit D i E se gase i na kraju ostaju samo niti A i B žive, zato se samo on i ispisuju.

Primer 5 – Više niti, jedan resurs

Često je potrebno da više niti koriste jedan isti resurs, što pokazuje sledeći primer.

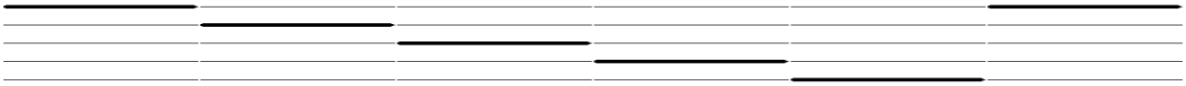
```
public class Broj {
	static int brojac = 0;
	public static void povecaj() {
		brojac++;
	}
}
```

```
public class Nit extends Thread{
	@Override
	public void run() {
		for (int i = 0; i < 10; i++) {
			Broj.povecaj();
		}
	}
}
```

```
public static void main(String[] args) throws InterruptedException {
	for (int i = 0; i < 1000; i++)
		new Nit().start();
	Thread.sleep(5000);
	System.out.println(Broj.brojac);
}
```

Ako pažljivije pogledamo kôd, vidimo da se u `main` metodi kreira 1000 niti i da se svaka startuje. Svaka od tih niti, tačno 10 puta pozove statičku metodu `povecaj()`, a ta metoda povećava za jedan `brojac`, koji je statički `intiger`. Nakon startovanja niti, uspavamo glavnu nit na 5 sekundi, kako bismo bili sigurni da će svih 1000 niti završiti posao, tj. povećati po 10 puta `brojač`. Zatim, taj `brojac` ispišemo u konzoli. Očekivano bi bilo da je vrednost `brojaca` 10*1000 = 10 000, međutim broj koji se ispiše je 9994.[1]

Problem je u tome što se povećavanje broja (iako jednostavna operacija), sastoji iz više koraka. Prvo se vrednost koju trenutno ima `brojac` prepiše u registar, zatim se u registru poveća, pa se onda ta povećana vrednost vrati u `brojac`. Kako smo mi pokrenuli 1000 niti, a nemamo 1000 jezgara na procesoru da se one izvršavaju potpuno paralelno, izvršava se multitasking. Procesor svakoj niti daje po mali deo vremena, tako da se nama čini da se sve niti izvršavaju paralelno.



Zbog toga, svaka nit može biti prekinuta u bilo kom trenutku, čak iako nije završila svoju metodu, metodu `povecaj()`.

Posmatrajmo dve niti, npr. nit A i nit B. Neka je vrednost `brojac` = 121. Trenutno se izvtšava A, a B čeka. A uzme vrednost upisanu u `brojac` i upiše je u registar, zatim poveća broj u registru za 1, znači na 122. Tu nastupa prekid niti A, i počinje izvršavanje niti B. Nit B, takođe uzima vrednost iz `brojac`-a koji je i dalje 121, upisuje ga u svoj registar, povećava na 122 i vraća ga u `brojac`. `brojac` sada ima vrednost 122. Prekida se nit B i počinje opet sa izvršavanjem nit A. A nastavlja tamo gde je stala. Da bi završila metodu, potrebno je da upiše vrednost iz registra u `brojac`, a to je 122. I opet je vrednost u `brojac`-u je 122. Dakle, dva puta se izvršila metoda `povecaj()`, a `brojac` se sa 121 povećao na 122.

Kako bismo se zaštitili od ovakve greške, potrebno je da nekako zaključamo promenljivu `brojac` za sve niti, osim za onu koja je započela rad sa njom. To postižemo ključnom reči syncronized, prilikom deklarisanja metode koje rade na deljenom resursu. Kod nas je to metoda povecaj(), a deljeni resurs je `brojac`.

```
public synchronized static void povecaj() {
      brojac++;
}
```

Izvršavanje ovakvog kôda je nešto sporije nego pre, ali barem smo sigurni da ćemo dobiti očekivan rezultat, u našem slučaju 10 000. Postoji još jedan mehanizam za zaključavanje, a to je klasa `ReentrantLock`.

[1] Neće se svaki put dobiti rezultat 9 994, ali očekivano je da se dobije broj manji od 10 000

Primer 6 – Wait i notify

Ključna reč `synchronized` ne garantuje koliko dugo jedna nit radi sa jednim resursom. Sledeći primer to pokazuje. Ideja je napraviti dve niti, koje će po 50 puta naizmenično da ispisuju `Cao!` pa `Zdravo!` u konzoli.

```
public class Konzola {
	public synchronized void cao() {
		System.out.println("CAO!");
	}
	public synchronized void zdravo() {
		System.out.println("\t\t ZDRAVO!");
	}
}
public class Cao extends Thread {
	Konzola konzola;
	public Cao(Konzola konzola) {
		this.konzola = konzola;
	}
	@Override
	public void run() {
		for (int i = 0; i < 50; i++)
			konzola.cao();
	}
}
public class Zdravo extends Thread{
	Konzola konzola;
	public Zdravo(Konzola konzola) {
		this.konzola = konzola;
	}
	@Override
	public void run() {
		for (int i = 0; i < 50; i++)
			konzola.zdravo();
	}
}
public static void main(String[] args) {
	Konzola konzola = new Konzola();
	new Cao(konzola).start();
	new Zdravo(konzola).start();
}
```

Iz main metode se pokreću dve niti. Obe niti koriste metode objekta klase `Konzola`. Te metode su sinhronizovane, što znači da su resursi u njoj zaključani za korišćenje drugih niti, dok tekuća ne završi posao. Kako procesor daje određeno vreme jednoj niti, pa drugoj, može se desiti da se izvrši više od jedne iteracije metode `run()` i na taj način ispiše više puta Cao! pre jednog Zdravo! i obrnuto. Zbog toga, deo ispisa na konzoli izgleda:

```
CAO!
CAO!
                ZDRAVO!
                ZDRAVO!
CAO!
```

Ono što bismo želeli da se ispiše je jedno Cao!, zatim jedno Zdravo!, pa opet Cao! itd. Zbog toga, jednu nit treba staviti na čekanje, dok je druga ne obavesti da može da počne sa radom. Kôd u konzoli treba izmeniti:

```
public class Konzola {
	boolean reciZdravo = false;
	public synchronized void cao() {
		try {
			if (reciZdravo)
				wait();
			System.out.println("CAO!");
			reciZdravo = true;
			notifyAll();
		} catch (InterruptedException e) {}
	}
	public synchronized void zdravo() {
		try {
			if (!reciZdravo)
				wait();
			System.out.println("\t\t ZDRAVO!");
			reciZdravo = false;
			notifyAll();
		} catch (InterruptedException e) {}
	}
}
```

Metoda za obaveštanje drugih niti je `notify()` ili `notifyAll()`. U ovom primeru smo koristili `notifyAll()`. Ukoliko postoji više niti, preporuka je koristiti `notifyAll()`, jer se na taj način obaveštavaju sve niti, dok se metodom `notify()` nasumično bira nit koja će prekinuti čekanje. Čekanje niti na poziv `notify()` se ostvaruje metodom `wait()`. Ova metoda baca `InterruptedException` ukoliko se nad njom pozove metode `interrupt()`, a nit se nalazi u fazi `wait()`. Sve metode se pozivaju samo iz `syncronized` metoda. Tako su resursi zaključani, ukoliko su zajednički. U našem slučaju, zajednički resurs je konzola po kojoj pišemo.

Redosled kojim će se niti izvršavati moramo mi da odredimo. To se radi uz pomoć booleana/flagova ili sličnih mehanizama. U ovom primeru, koristimo boolean reciZdravo, koji je na početku postavljen na false, znači prvo će se ispisati Cao!.

Iz `main` metode smo startovali obe niti. I jedna, i druga, pokušavaju da ispišu svoj tekst u konzoli. Zbog booleana reciZdravo, uslov u niti Zdravo je ispunjen i ta nit je pozvala metodu wait(). Za to vreme, ispisalo se Cao, zatim je promenjena vrednost reciZdravo i pozvana je metoda notifyAll(). To je obavestenje za metodu Zdravo da nastavi sa radom, dok je nit Cao

otišla na čekanje (zbog vrednosti booleana). Sada nit Cao čeka da neko pozove notifyAll(). I sve se dešava tako u krug dok se ne završi 50-ta iteracija.

Rezultat u konzoli sada izgleda:

```
CAO!
                ZDRAVO!
CAO!
                ZDRAVO!
```

Primer 7 – Wait i notify (više od dve niti)

Pored dve niti koje ispisuju Cao! i Zdravo!, ovde dodajemo i treću nit koja ispisuje DobarDan!. Ideja je ista, redosled ispisivanja da bude Cao! Zdravo! Dobar dan! Cao! Zdravo! …

Primer 6 je malo izmenjen:

```
public class Konzola {
	boolean isCao,isZdravo,isDobarDan;

	public Konzola() {
		isCao = true; isZdravo = false; isDobarDan = false;
	}
	public synchronized void cao() {
		try {
			if (isCao == false)
				wait();
			System.out.println("CAO!");
			isCao = false; isZdravo = true; isDobarDan = false;
			notifyAll();
		} catch (InterruptedException e) {}

	}
	public synchronized void zdravo() {
		try {
			if (isZdravo == false)
				wait();
			System.out.println("\t\t ZDRAVO!");
			isCao = false; isZdravo = false; isDobarDan = true;
			notifyAll();
		} catch (InterruptedException e) {}
	}
```

```
	public synchronized void dobarDan() {
		try {
			if (isDobarDan == false)
				wait();
			System.out.println("\t\t\t\t DOBAR DAN!");
			isCao = true; isZdravo = false; isDobarDan = false;
			notifyAll();
		} catch (InterruptedException e) {}
	}
}
```

Pored ovog kôda , dodata je klasa DobarDan, koja je implementirana na isti način kao i klase Cao i Zdravo. U `main` metodi je pokrenuta i treća nit: `new DobarDan(konzola).start();`

Sada nije dovoljna boolean vrednost da proveri sva tri slučaja, pa su uvedene 3 vrednosti. Kada je neka od njih `true`, to je sledeća nit koja treba da se izvrši. Kada se ispiše Cao!, `boolean` za nit Zdravo se postavi na `true`, ostale na `false` i ista logika je za druge niti.

Rezultat ovog programa je:

```
CAO!
		ZDRAVO!
CAO!
				DOBAR DAN!
CAO!
		ZDRAVO!
```

Rezultat nije očekivan, ispiše se Cao! , Zdravo!, pa opet Cao! itd. To je zbog toga što se provera za čekanje vrši `if` uslovom. Ono što se desi je da se ispiše Cao!, postave booleani da se sledeće ispiše Zdravo!. Nakon Zdravo! poziva se `notifyAll()`. Za to vreme i Cao i DobarDan su ušle u `if` uslov i čekaju da se prekine metoda `wait()`. Kada to uradi Zdravo, opet se ispiše Cao.

Ono što je dobra praksa i što bi uvek trebalo raditi, je da se `if` uslov zameni `while` uslovom.[2] Sve tri metode bi trebao da liče na sledeću.

```
public synchronized void cao() {
	try {
		while (isCao == false)
			wait();
		System.out.println("CAO!");
		isCao = false;
		isZdravo = true;
		isDobarDan = false;
		notifyAll();
	} catch (InterruptedException e) {}
}
```

[2] Efikasno programiranje na Javi (Joshua Bloch), savet 50

Sada, kada se ispiše Cao!, nameste booleani, ispiše Zdravo!, ponovo nameste booleani i pozove metoda `notifyAll()`, obavesti se i nit Cao da izađe iz `wait()` stanja. Međutim, kako se sad taj uslov nalazi u petji, opet se proveri da li je zaista nit Cao ta koja treba da nastavi sa izvršavanjem. Ako nije, ona opet ulazi u fazu `wait()`. Rezultat je:

```
CAO!
                ZDRAVO!
                                DOBAR DAN!
CAO!
                ZDRAVO!
                                DOBAR DAN!
```