PREDMET

Osnove Java Programiranja

Čas3-4

PRVI PROGRAM I UNOS PODATAKA U PROGRAM

Oktobar 2011.

SADRŽAJ

Čas 3-4

Prvi program i unos podataka u program

Tipovi podataka

Java je tipski programski jezik. Za svaki podatak koji korstimo u Javi, moramo da znamo kog je tipa. Svi podaci mogu da se podele u dve grupe. Prva grupa su prosti tipovi, kod kojih se tačno zna koliko prostora zauzimaju. U tabeli možemo da vidimo proste tipove koji se koriste u Javi. Kao što vidite, u drugoj koloni dat je tačan iznos u bitovima koliko taj tip podataka zauzima prostora. Drugi tip podataka u Javi su klase. Kod klasa ne možemo unapred da kažemo koliko prostora će klasa da zauzme. Klase su dinamički tipovi.

Prosti tipovi	bitovi	Klase
boolean	1	Boolean
char	16 (32)	Character
byte	8	Byte
short	16	Short
int	32	Integer
long	64	Long
float	32	Float
double	64	Double

Tip podatka **boolean** je logička promenljiva, i on u sebi sadrži informaciju da li je nešto tačno ili nije. Za smeštanje tog podatka dovoljan mu je jedan bit. Pomoću jednog bita možemo da sačuvamo dve informacije - u ovom slučaju to su podaci tačno i ne tačno.

Tip podatka **char** predstavlja pojedinačne karaktere. U jednoj promenljivoj tipa **char** može da se smesti samo jedan karakter. U većini programskih jezika **char** je veličine 8 bita, što predstavlja samo osnovni skup karaktera koji je obuhvaćen engleskim jezikom. Java je od početka pravljena sa idejom da postane multinacionalni programski jezik, i iz tog razloga karakteri u Javi podržavaju Unicode standard. Izlaskom Jave 5 (JDK 1.5), Java je preuzela novi Unicode 4 standard u kome je svaki karakter definisan pomoću 32 bita. Starije verzije Jave podržavaju stari 16 bit-ni Unicode standard.

Tip podataka **byte** pedstavlja osnovnu reč i retko se korsti, uglavnom za raw obradu podataka.

Tipovi podataka **short**, **int** i **long** predstavljaju cele brojeve. Osnovna razlika među njima je u količini prostora koji zauzimaju, a samim tim i u veličini broja koji može sa svakim od njih da se predstavi. Iako nam je često dovoljno da promenljivu deklarišemo kao **short**, programeri skoro uvek koriste za te svrhe promenljivu tipa **int**. Tip podataka **short** se retko koristi. Tip podataka **long** je nezamenljiv u nekim situacijama. Dobar primer za to je vreme. Računar vreme i datum pamti kao broj milisekundi od

01.01.1900. S obzirom da je od tada prošlo više od sto godina broj milisekundi može da se prikaže samo pomoću promenljive tipa **long**.

Tipovi podataka **float** i **double** služe za predstavljanje brojeva u pokretnom zarezu. Obično za proračune i rad sa pokretnim zarezom koristimo tip **double**, dok se tip **float** ređe koristi.

Kao što vidite u tabeli, svaki prost tip podataka ima odgovarajuću klasu koja je zadužena za isti tip podatka, ali u obliku klase. Klase mogu da imaju i metode koje obrađuju te podatke, tako da odgovarajuće klase imaju extra mogućnosti koje prosti tipovi ne mogu da imaju.

Davanje naziva promenljivima

Postoje izvesna pravila prilikom imenovanja promenljivih:

- Pravilo 1 – promenljiva mora da bude jedna reč. U promenljivoj ne sme da postoji razmak.
- Pravilo 2 – promenljiva ne sme da počne brojem
- Pravilo 3 – promenljiva može da sadrži broj u svom nazivu
- Pravilo 4 – promenljiva može da sadrži sva slova (velika i mala), ali od specijalnih simbola dozvoljena su samo dva (_ i $).

Evo nekliko primera ispravnog imenovanja metoda:

```
int i; //Promenljiva može da se sastoji od jednog simbola
float prosecan_broj; //Promenljiva može u sebi da ima simbol _
char oVoJePrOmEnLJiVa; //Promenljiva može da se sastoji od malih i velikih slova
String str2; //Promenljiva može da sadrži broj u sebi
String _str; //Promenljiva može da počne specijalnim simbolom _ ili $
Double PDV; //Promenljiva može da ima i sva velika slova
```

Evo nekoliko primera nepravilnog imenovanja promenljivih. Dakle, ovako ne sme!

```
float prosečan broj; //Ne sme promenljiva da ima razmak u sebi
String 2str; //Promenljiva ne sme da počne brojem
double pdv%; // Promenljiva ne sme da ima nikakve simbole u svom nazivu.
```

Mađarska konvencija

Iako nam sam programski jezik ne određuje kakve ćemo nazive davati promenljivima, ne bi bilo korisno kada bi svaki programer imao sopstveni stil nazivanja promenljivih, metoda, klasa, paketa...

Programeri su se dogovorili da naprave jedan zajednički sistem za davanje naziva i da se onda svi pridržavaju tih preporuka. Ovo je naravno na nivou preporuka, pošto sam jezik vas ne ograničava po tom pitanju. Mi ćemo na ovom predmetu poštovati Mađarsku konvenciju, koja daje neke preporuke kako se imenuju paketi, klase, metode i promenljive.

- Generalna napomena

Ako želimo da promenljivoj damo naziv koji se sastoji od više reči, to radimo tako što svaka sledeća reč počinje prvim velikim slovom (dok su sva ostala slova u nazivu mala).

U nazivima se ne pojavljuju dopušteni specijalni simboli _ i $.

- Paketi

Po mađarskoj konvenciji naziv paketa mora da počne malim slovom (preporuka je da paketi nemaju više od jedne reči u sebi):

sistem

util

- Klase

Klase po međarskoj konvenciji počinju velikim slovom:

```
Ucenik
Faktura
UlaznaFaktura
FactoryButton
```

- Metode

Sve metode počinju malim slovom:

```
izracunaj()
napuniMagacin()
setBroj()
```

- Promenljive

Nazivi promenljivih treba da počinju malim slovom

```
int brojac; :
String glavniNaziv;
double pdvPosto;
```

Metode

U objektno orijentisanom programiranju metode su jako važne. Možemo da definišemo nekoliko osnovnih tipova metoda, ali pre toga hajde da vidimo kako se deklariše neka metoda:

```
public void prikazi()
{
      //nekakav kod metode
}
```

Ovo je najjednostavniji oblik neke metode. Ova metoda nema povratnu vrednost i nema parametre. Kao što vidimo iz ovog primera, metoda može da nema parametre. To ne znači da ona ne koristi neke podatke, već samo znači da joj se ne prosleđuje ni jedan podatak van klase u kojoj se nalazi. Poljima klase, podacima u okviru same klase, može da se pristupi iz metoda (ne prosleđuju se kao parametri). Ključna reč void nam označava da metoda nema povratnih vrednosti. Povratna vrednost je rezulatat rada metoda koji treba da se dostavi nekom ko poziva taj metod, a koji se nalazi van klase. Ako metoda treba samo da promeni vrednost nekom polju klase, onda metoda može da bude tipa void (odnosno da ništa ne vraća).

Evo primera metode koja ima parametre i vraća rezulatat pozivaru:

```
public int funkcija( int prvi, int drugi, String opis)
{
      if(opis != "")
            return prvi * drugi;
      return prvi + drugi;
}
```

U ovom primeru vidimo da metoda prima tri promenljive i vraća proizvod prva dva parametra u slučaju da ima nekakav opis, a u slučaju da je opis prazan onda će vratiti zbir prva dva parametra. Parametri funkcije, odnosno metode, se zovu još i argumenti.

Argument funkcije može biti bilo koji objekat ili osnovni tip podataka.

Ako je argumet osnovni tip podataka, onda se šalje kopija tog podatka. Ako je argument objekat ,onda se šalje referenca na objekat. Kada se šalje kopija, kao u slučaju osnovnog tipa, to znači da sve izmene na toj promenljivoj neće uticati na originalni podatak.

Svi objekti u svojim promenljivima sadrže samo reference na objekte, tako da kad se prosledi kopija te reference, nova promenljiva pokazuje na isti fizički objekat. To znači da će se bilo kakva promena na objektu odraziti i na originalan objekat.

Ako želimo da osnovni tip podatka šaljemo kao referencu onda ga predstavimo kao niz dimenzije 1, pa ga posaljemo. S obzirom da je niz objekat, biće prosleđena kopija reference, pa ćemo moći u funkciji da menjamo vrednost objekta (u ovom slučaju osnovnog tipa).

Isto tako, ako ne želimo da nam se objekat menja u metodi koju pozivamo, možemo pre nego što prosledimo objekat da napravimo novi objekat koji je kopija originalnog objekta, i tada izmene nad tim novim objektom naravno neće uticati na originalan objekat.

U Javi imamo nekoliko karakterističnih metoda.

Prva od njih je konstruktor. Metoda konstruktor mora da postoji u svakoj klasi. Ako programer ne napiše konstruktor kompajler će automatski da generiše prazan konstruktor prilikom kompajliranja te klase. Bez obzira na činjenicu što kompajler kreira prazan konstruktor preporučeno je da programeri ipak napišu ovaj prazan konstruktor kada je potrebno. Evo primera praznog konstruktora:

```
class MojaKlasa
{
	int broj;
	MojaKlasa() {}
}
```

U predhodnom primeru smo napravili klasu koja se zove MojaKlasa, i u njoj kreirali prazan konstruktor. Iz ovog primera možemo da vidimo da konstruktor malo odstupa od do sada naučenog o metodama. Kao prvo, konstruktor nema tip podatka koji vraća. Ovo je očekivano s obzirom da je njegova uloga da kreira objekat, i uvek vraća objekat koji je napravio. Zbog toga konstruktor ne može da vraća ništa drugo. Kao što možete da vidite, konstruktor ima identično ime imenu klase, po tome se i zna da je to konstruktor te klase. Kao što primećujete, naziv konstruktora odstupa od mađarske konvencije za metode, jer nema prvo slovo malo već veliko (isto kao i klasa). Ovo je dobro, jer ćemo lako u programu uvek da znamo šta nam je konstruktor. Gore pokazani konstruktor ima prazan spisak argumenata (ovo nije pravilo i možemo da napravimo i neki ne prazan konstruktor). Obično se pravi konstruktor koji prima sve podatke koje ima klasa. Za predhodnu klasu to bi ovako izgledalo:

```
class MojaKlasa
{
	int broj;
	MojaKlasa() {}
	MojaKlasa(int broj)
	{
		this.broj = broj;
	}
}
```

Ključna reč **this** ima značenje: ova ista klasa (instanca klase) u kojoj se nalazimo. Kao što vidite, dali smo isti naziv atributu funkcije kao što je naziv polju klase. Oba se zovu **broj**.

Kada pristupimo promenljivoj **broj** u okviru konstruktora, mi uvek pristupamo atributu funkcije, jer je on preklopio promenljivu klase. Ovo možemo da prevaziđemo tako što pomoću ključne reči **this** pristupimo promenljivoj klase, kao što je prikazano u ovom primeru.

U ovom primeru vidimo da možemo da imamo dva metoda, konstruktora u ovom slučaju, koji se isto zovu. Ovo je jedna od mogućnosti objektno orijentisanog programiranja koju ćemo detaljnije objasniti

tokom ovog predmeta, a zove se overloading. Kada imamo dva ili više konstruktora, pokernuće se onaj koji se pozove. U predhodnom slučaju možemo da pokernemo prazan konstruktor i da dobijemo objekat tipa MojaKlasa, koji nema nikakav podatak u promenljivoj broj.

MojaKlasa mk = new MojaKlasa();

Drugi slučaj je kad pozovemo drugi konstruktor i prosledimo parametar. Tada će konsktuktor napuniti promenljivu broj, i vratiti nam objekat koji u sebi ima već napunjenu vrednost pormenljive broj. Evo i tog primera:

MojaKlasa mk = new MojaKlasa(8);

Postoji još jedan karakterističan konstruktor, a to je copy konstruktor. Taj konstruktor pravi novi objekat tako što primi kao parametar konstruktora objekat istog tipa, i svaki podatak iz prosleđene instance objekta iskopira u taj novi objekat. Nakon toga pozivaru vrati objekat, koji je kopija objekta čiju instancu je dobio. Evo primera sa dodatim copy konstruktorom:

```
class MojaKlasa
{
	int broj;
	MojaKlasa() {}
	MojaKlasa(int broj)
	{
		this.broj = broj;
	}
	MojaKlasa(MojaKlasa mk)
	{
		this.broj = mk.getBroj();
	}
}
```

Neki drugi objektno orijentisani programski jezici, kao što je C++, imaju destruktor. Destruktor je metoda koja uništava ono što je konstruktor napravio. U Javi ne postoji destruktor, već JVM sam uništava objekte. S vremena na vreme sakupljač đubreta prođe memorijom, virtuelnom mašinom, i počisti klase koje više niko ne koristi, i time oslobodi memoriju računara.

Pomoću naredbe System.gc() i Runtime.gc() možemo da sugerišemo JVM da pokrene skupljač đubreta. Ako objektima koji nam više ne trebaju dodelimo vrednosti **null**, onda ćemo obezbediti da ih skupljač đubreta pokupi kad sledeći put bude prolazio.

Prva klasa

Osnovno o klasama:

Klasa je šablon za pravljenje objekata. Objekat koji je nastao po šemi neke klase zove se instanca klase. Svaki source fajl mora da ima bar jednu javnu klasu, a naziv tog fajla mora da se pokalpa sa nazivom te klase. Source fajl može da ima tri dela:

- Package deklaracija
- Importi
- Class definicija

Source fajl je običan tekstulni fajl ,koji ima ekstenziju java.

Tipična klasa u javi treba da ima sledeće delove:

Podatke

Konstruktor prazan

Konstruktor koji prima sve podatke klase

Copy konstruktor

Setere i getere za podatke

Override metode toString

Metode specifične za tu klasu

Počećemo od podataka. Napravićemo klasu koja će nam biti zadužena za vođene kredita. Podaci koji nam trebaju su nam naziv kredita, broj mesečnih rata i preostalo do kraja.

private String kredit;

private int brojRata;

private double ostalo;

ključna reč je modifikator i on obezbeđuje da promenljivoj ne može da pristupa niko van same klase.

Kako se prave konstruktori, već smo objasnili.

Seteri i geteri su metode koje nam obezbeđuju pristup promenljivim van klase u kojoj se nalazimo. U nekom od narednih predavanja ćete učiti o osnovnim principima objektno orijentisanog programiranja, gde će biti više reči o prednostima pristupa promenljivima preko setera i getera. Evo kako izgleda jedan tipičan seter i geter. U ovom primeru smo napravili seter i geter za podatak **kredit**:

```
public String getKredit()
{
	return kredit;
}
public void setKredit(String kredit)
{
	this.kredit = kredit;
}
```

Geteri služe da nekom spolja dostave vrednost promenljive. Iz tog razloga metoda mora da ima isti potpis (tip koji vraća), kao što je tip promenljive koju vraća.

Seter služi da bi vrednost promenljive postavio na neku novu vrednost. Da bi to moglo da se ostvari, argument metode mora da bude istog tipa kao i podatak koji se setuje. Seter ništa ne vraća.

Evo još jedne metode koja se često pravi u klasama. Metoda toString je zadužena za tekstualnu reprezentaciju instance klase. Ova metoda već postoji u korenu svakog objekta, ali mi često želimo da tekstualna reprezentacija bude prilagođena našim potrebama. Iz tog razloga pravimo novu metodu koja ima isti naziv kao i osnovna metoda, i time gazimo osnovnu metodu našom:

```
@Override public String toString()
{
	return "Kredit: " + krediti + "\t ostalo jos: "+ ostalo;
}
```

@Override je anotation i to je novost u jeziku Java od verzije 5. Pomoću anotation-a mi govorimo razvojnom okruženju da želimo da pregazimo korenu metodu. Zahvaljujući anonation-u, razvojno okruženje može da proverava da li smo napravili neku grešku. Što se same metode tiče, ona uvek mora da vrati String. U samoj metodi mi pakujemo taj string kako nama odgovara, odnosno kako želimo da se prikazuje.

Metode tipične za klasu su metode koje nam pomažu da data klasa ima željeno ponašanje. U ovom slučaju pravimo metodu koja oduzima ratu i menja preostali dug na novu vrednost.

```
public void umanjiZaRatu()
{
```

```
        double tempRata;
        tempRata = ostalo / brojRata;
        ostalo -= tempRata;
}
```

Evo kako izgleda celokupna naša klasa. Ova klasa mora da se nalazi u fajlu koji se zove Krediti.java :

```
public class Krediti
{
        private String krediti;
        private int brojRata;
        private double ostalo;
        public Krediti(){ }
        public Krediti(String krediti, int brojRata, double ostalo)
        {
                this.setKrediti(krediti);
                this.setBrojRata(brojRata);
                this.setOstalo(ostalo);
        }
        public Krediti(Krediti kredit)
        {
                this.setKrediti(kredit.getKrediti());
                this.setBrojRata(kredit.getBrojRata());
                this.setOstalo(kredit.getOstalo());
        }
        public String getKrediti()
        {
                return krediti;
        }
        public void setKrediti(String krediti)
        {
                this.krediti = krediti;
        }
        public int getBrojRata()
        {
                return brojRata;
        }
        public void setBrojRata(int brojRata)
        {
                this.brojRata = brojRata;
        }
        public double getOstalo()
```

```
	{
		return ostalo;
	}
	public void setOstalo(double ostalo)
	{
		this.ostalo = ostalo;
	}
	@Override public String toString()
	{
		return "Kredit: " + krediti + "\t ostalo jos: "+ ostalo;
	}
	public void umanjiZaRatu()
	{
		double tempRata;
		tempRata = ostalo / brojRata;
		ostalo -= tempRata;
	}
}
```

Ovde vidimo kako treba da izgleda jedna tipična klasa u Javi. Ono što početnike obično brine je što se odavde ne vidi kako program radi. Objektno orijentisano programiranje se bavi pojedinačnim objektima i njihovim međusobnim odnosima, a funkcionalnost aplikacije će proizaći od toga.

Metoda main

Aplikacija mora da počne od negde. U slučaju Java programa to je metoda **main**. Ova metoda treba teoretski da postoji samo u jednoj klasi. Klasu u kojoj se nalazi **main** metoda zovemo startnom klasom:

```
public static void main(String[] args)
{
	new StartKlasa();
}
```

Kao što vidite u ovom primeru u okviru **main** metode smo samo konstruisali prvu klasu. Ovo je osnovni zadatak main metode da pokrene program, i da dalje prepusti tok programa objektima.

Metoda **main**, kao što vidite, može da ima i niz stringova kao ulaz. Mnogi savremeni programi korste tu mogćnost, kako bi obezbedili pokretanje programa sa unpred pripremljenim podacima, ili na primer pokretanje određenog dela aplikacije. U sledećem programu vidimo kako bi se startovala Java aplikacija sa dva parametra. Zamislite da smo napravili aplikaciju koja prikazuje mapu Srbije i na mapi možemo da odaberemo gradove, pa se onda iscrtava najkraći put između ta dva grada i prikazuje se kilometraža tog puta. Kada program pokrenemo on verovatno prvo ulazi na neki ekran odakle biramo mapu, pa onda odabiremo gradove itd... Ako iskoristimo mogućnost prosleđivanja parametra u **main** metodi, u mogućnosti smo da našu hipotetičku aplikaciju usavršimo time što ćemo obezbediti da se sa prosleđenim parametrima aplikacija pokreće sa već iscrtanim potrebnim podacima. Ovo je samo jedna od mogućih primena parametara u okviru **main** metode:

java gradovi Beograd Niš

args[0] sadrži naziv startne klase

args[1] sadrži String Beograd

args[2] sadrži String Niš

Prvi parametar se automatski popunjava, tako da je prvi član niza args uvek naziv startne klase, dok je svaka sledeća reč (odvaja ih razmak) sledeći član niza args.

Prvi program

Da bi naš program radio moramo da imamo startnu klasu. Startnu klasu za ovu aplikaciju ćemo nazvati **Main**. Ovo pišemo u tekstualnom fajlu koji se zove isto kao i klasa **Main,** i ima extenziju **java** (dakle, fajl se zove **Main.java)**. U okviru startne klase imamo samo dve metode. Metoda **main** koja nam služi za startovanje aplikacije i koja u ovom slučaju samo pokreće konstruktor startne klase, i konstruktor startne klase. U okviru konstruktora smo napravili instancu naše klase **Krediti**. Ovu instancu smo nazvali **kreda,** i kroz konstruktor je napunili početnim podacima. Iskoristili smo metodu **toString** da bi prikazali početno stanje klase. Nakon toga smo pokrenuli metodu **umanjiZaRatu**, koja će da izvrši umanjivanje kredita za jednu ratu. Ponovnim pokretanjem **toString** metode prikazujemo novo stanje instance **kreda**. U programu smo koristili metodu **System.out.println** koja ispisuje string na komandnoj liniji razvojnog okruženja:

```
public class Main
{
	Main()
	{
		Krediti kreda = new Krediti("Kola", 12, 10000);
		System.out.println(kreda.toString());
		kreda.umanjiZaRatu();
		System.out.println(kreda.toString());
	}
	public static void main(String[] args)
	{
		new Main();
	}
}
```

Unos podataka pomoću JOptionPane dijaloga

Konzolni ulazi su manje interesantni jer smo svi navikli na grafička okruženja. Java ima jednostavan način unosa podataka preko dijaloga.

```
String unos = JOptionPane.showInputDialog("Uneti neki tekst:");
```

Postoje nekoliko tipova **JOptionPane** dijaloga.

- InputDijalog
- ConfirmDialog
- MessageDialog
- OptionDialog

Svaki od ovih dijaloga ima svoju namenu. InputDialog nam služi za unos podataka u program. Podatak koji unosimo je tipa String. ConfirmDijalog nam služi da potvrdimo nešto obično (ima dva dugmeta OK i Cancel pomoću kojih prihvatamo ili odustajemo od nečega). MessageDijalog nam služi da o nečemu obavestimo korisnika. OptionDialog nam služi za pravljenje proizvoljnog dijaloga.

Evo izgleda jednog tipičnog MessageDialoga:

```
JOptionPane.showMessageDialog(null,"unos","Unet je sledeci tekst: " +
tekst, JOptionPane.INFORMATION_MESSAGE)
```

Ovaj MessageDialog ima 4 parametra. Prvi objekat predstavlja instancu koja je Frame roditeljskog prozora, pošto trenutno još uvek nemamo Prozor aplikacije, ovaj parametar ćemo postaviti na **null**, što označava da nema objekta. Drugi parametar je tekst u naslovu dijaloga. Treći parametar je tekst u dijalogu (pomoću plusa smo fiksnom stringu dodali String promenljivu), i sve to zajedno sada predstavlja tekst koji će se prikazati u dijalogu. Četvrti parametar je konstanta koja označava ikonicu koja će se prikazati sa leve strane teksta u dijalogu. Moguće vrednosti ove konstante su sledeće:

- - JOptionPane.ERROR_MESSAGE
- - JOptionPane.INFORMATION_MESSAGE
- - JOptionPane.QUESTION_MESSAGE
- - JOptionPane.WARING_MESSAGE

(Bez slike) - JOptionPane.PLAIN_MESSAGE

Da vidimo kako možemo da napravimo jedan proizvoljan dijalog. Evo primera:

```
Object[] opcije = {"Da", "Ne", "Odustujem"};
int n = JOptionPane.showOptionDialog(null,
"Da li želite nešto novo da naučite",
"Odlučujuće pitanje",
JOptionPane.YES_NO_CANCEL_OPTION,
JOptionPane.QUESTION_MESSAGE,
null, opcije, opcije [2]);
System.out.println("Rezultat upita je: " + n);
```

Napravili smo niz stringova **opcije** koji nam određuju tekst na dugmićima YES, NO i CANCEL resprektivno.

Prvi objekat predstavlja instancu koja je Frame roditeljskog prozora. Pošto trenutno još uvek nemamo Prozor aplikacije, ovaj parametar ćemo opet postaviti na **null,** što označava da nema objekta. Drugi parametar je tekst u dijalogu. Treći parametar je tekst u naslovu dijaloga. Četvrti parametar je tip dugmića u dijalogu, u ovom slučaju znači da imamo tri dugmeta: YES, NO i CANCEL, koje ćemo u predzadnjem parametru da pregazimo sa našim tekstom. Peti parametar je konstanta koja označava ikonicu koja će se prikazati sa leve strane teksta u dijalogu. Šesti parametar je proizvoljna ikona koja bi menjala default sličicu (u ovom slučaju null znači da će se koristiti default slika). Sedmi parametar je niz tekstova za dugmiće koji će menjati standardne YES, NO i CANCEL. Poslednji parametar nam određuje koje dugme će biti ponuđeno kao default.