

NOVU **GENERACIJU** TELEFONA **MOŽETE** PROGRAMIRATI  
**I TAKO IH** PRILAGODITI **SPECIFIČNIM** POTREBAMA.  
**EVO** KRATKOG **VODIČA** ZA PRIMENU **JAVE** NA **TELEFONIMA.**

# JAVA ZA mobilne TELEFONE

SVE JE VEĆI BROJ MOBILNIH TELEFONA S naprednim opcijama, što je uz pad cena doprinelo da se u ekonomskoj klasi nađu mnogi modeli sa ekranima u boji i memorijom solidne veličine. Gotovo svi novi telefoni podržavaju mikro izdanje Jave J2ME (Java 2 Micro Edition), a to omogućava da se aplikacije i igre lako instaliraju, preuzimanjem s Interneta ili sa stonog računara. ▶

**DEJAN  
MRATINKOVIĆ**

J2ME je verzija Java koja je namenjena mobilnim telefonima, pejdžerima, LDP i sličnim uređajima. Brojne kompanije su se pridružile njenom razvoju. Razvojne alate možete preuzeti s adrese <http://java.sun.com/j2me/download.html>.

J2ME pruža kompletna rešenja za mrežne aplikacije visokog nivoa, omogućava proizvođačima uređaja, davaocima usluga kao i programerima da razvijaju namenske aplikacije za svoje korisnike. Usaglašena je sa osnovnim smernicama u razvoju Java, kao što su bezbednost i nezavisnost od platforme.

Po čemu se ovo izdanje Java razlikuje od standardnog J2SE (Java 2 Standard Edition)?

## CDLC DEFINIŠE STANDARDNI SKUP

Što se koncepta jezika tiče, razlike uopšte nema, ali je ima u broju osnovnih ugrađenih klasa. Mnoštvo raznovrsnih klasa koje postoje u standardnom izdanju Java omogućavaju da se pri razvoju aplikacija pažnja obrati na detalje algoritma, jer je većina tehničkih pitanja već rešena. Što se mikro izdanja Java tiče, situacija je znatno drugačija. U ovom smanjenom izdanju nalazi se veoma mali broj klasa na koje su programeri standardne Java navikli, a i one su uglavnom siromašnije.

S gledišta visokog nivoa, J2ME definiše izvršno okruženje (runtime environment) koje se sastoji od niza Javinih virtuelnih mašina za različite uređaje, grupe biblioteka i API-ja koje se mogu izvršavati na svim od virtuelnim mašinama. Najvažniji deo je Javina virtuelna mašina koja se izvršava na operativnom sistemu uređaja. Iznad toga je specifična J2ME konfiguracija, skup biblioteka koje pružaju osnovnu funkcionalnost zasnovanu na mogućnostima uređaja. Na vrhu je jedan J2ME profil ili pak više njih koji se sastoji od dodatnih biblioteka što koriste neke od specifičnosti pojedinih uređaja. J2ME definiše i razne alatke za izvoz programa i konfigurisanje uređaja.

Vrlo je važno naglasiti razliku između konfiguracija i profila.

### KONFIGURACIJE

MOBILNI TELEFONI, pejdžeri i drugi mali uređaji razlikuju se po mnogo čemu, ali ipak, koriste slične procesore i imaju slične kapacitete memorije. Iz tih razloga postoje konfiguracije. One definišu horizontalno grupisanje proizvoda na nivou procesorske snage i raspoložive memorije. Na osnovu te informacije definišu se elementi jezika Java i biblioteke što će biti podržani i koja će se virtuelna mašina koristiti. Danas, postoje dve standardne konfiguracije: Konfiguracija za ograničene povezane uređaje (Connected Limited Device Configuration, CLDC) i Konfiguracija za povezane uređaje (Connected Device Configuration, CDC).

Ciljna grupa CDC-a su procesorski relativno jaki uređaji direktno povezani na neku vrstu mreže, kao što su Internet TV prijemnici i uređaji za navigaciju u automobilu. Takvi uređaji su veoma retki na našem tržištu. CDC sadrži potpunu Javinu virtuelnu mašinu, vrlo sličnu onoj koja se koristi u J2SE.

Prema zvaničnoj specifikaciji, to su uređaji s 32-bitnim procesorom, bar 2 MB slobodne memorije za Javu, povezani na neku vrstu mreže do 9600 b/s. Detaljnije informacije o tome potražite

na adresi <http://www.jcp.org/jsr/detail/36.jsp>.

J2ME je više usmerena ka drugom tipu konfiguracije, CLDC. Ona je mnogo manje zahtevna od CDC-a. Napravljena je sa idejom da bude najveći zajednički deo za sve uređaje s Java platformom, pre svega u oblasti rada na mreži, bezbednosti i ulazno-izlaznog interfejsa. Najzanimljiviji uređaji koji podržavaju CLDC jesu mobilni telefoni, a tu su i dvosmerni pejdžeri, LDP uređaji i sl.

Najznačajnija razlika u odnosu na CDC jeste specifikacija memorije koja zahteva od 160 KB do 512 KB. Ta granica nije previše stroga i očita je konvergencija jednoj zajedničkoj specifikaciji.

## FUNKCIJA ZA OGRANIČENE UREĐAJE

### VIRTUELNE MAŠINE

SVAKA KONFIGURACIJA podržava određene mogućnosti Javine virtuelne mašine. Virtuelna mašina za CLDC je KVM (Kilo Virtual Machine). KVM je neuporedivo manja od CVM, virtuelne mašine za CDC budući da podržava daleko manje osobina standardne Javine virtuelne mašine.

KVM je celovito izvršno okruženje za male uređaje. To je prava virtuelna mašina, po specifikacijama Javine virtuelne mašine izuzev što ima neke osobine usko vezane za rad uređaja s ograničenom memorijom i procesorskom snagom.

CVM je pravljen za moćnije uređaje. Podržava sve osobine virtuelnu mašinu 1.3 za Javu 2, bezbednosne biblioteke, Javin lokalni interfejs JNI i daljinsko pozivanje metoda RMI.

### PROFILI

JAVA PLATFORME su vertikalno grupisane po profilima. U suštini, profil je skup API-ja koje se nalaze na vrhu konfiguracija i omogućavaju programski pristup funkcijama koje zavise od mašine.

Profil mobilnih informacionih uređaja MIDP (Mobile Information Device Profile) dizajniran je tako da se koristi sa CLDC-om. To je skup API-ja za prenosive uređaje poput mobilnih telefona i dvosmernih pejdžera. MIDP sadrži klase za korisničko okruženje, rad u mreži i trajno skladištenje, sadrži i elemente koji omogućavaju snimanje novih aplikacija na uređaje. Aplikacije na tom profilu zovu se MIDlet.

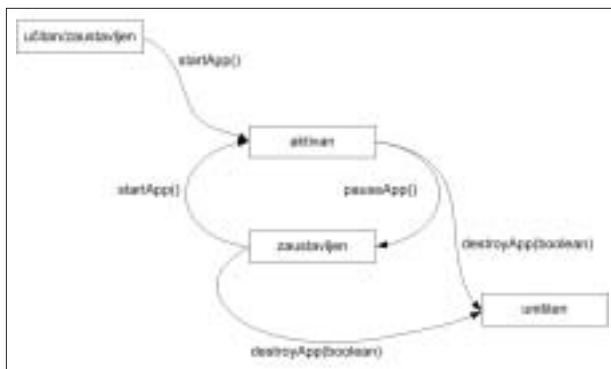
Profil PDA je takođe namenjen CLDC-u. Tu je, zatim Osnovni profil za CDC, karakterističan po tome što nema API-ja za korisničko okruženje, kao i profili Personal i RMI koji ga nasleduju. Ovde ćemo razmotriti samo nama najzanimljiviji profil MIDP konfiguracija za ograničene povezane uređaje.

CLDC definiše standardni, najveći zajednički deo za male uređaje ograničenih resursa. Na tom nivou ne postoji razlika između pejdžera i mobilnog telefona. Korisničko okruženje, životni ciklus aplikacije, interakcija s korisnikom i upravljanje događajima nisu rešeni u CLDC već se to definiše na nivou profila. Ove uređaje povezuju mala memorija i moć procesora. Svi oni podržavaju kompletan CDLC što znači da nema opcionih delova. Osnovni razlozi za celovitu podršku su prenosivost i komunikacija raznih sličnih uređaja.

Upravo zato što mora da podrži presek osobina svih malih uređa-



# PROGRAMIRANJE MIDP PROFILA ▶



## ŽIVOTNI VEK MIDLETA ▲

ja, JVM ne sadrži izvestan broj elemenata Javine virtualne mašine, iako je potpuno izvršno okruženje. Nekih nema zbog nedostatka hardvera, velike potrošnje memorije ili smanjene bezbednosti (na primer, nije moguće koristiti realne brojeve). Metoda `Object.finalize()` nije podržana, a podržane su samo tri klase za greške: `java.lang.Error`, `java.lang.OutOfMemoryError` i `java.lang.VirtualMachineError`.

Provera ispravnosti klasa u vreme izvršavanja, koja postoji u J2SE, izbačena je zbog velike potrošnje memorije. To je prebačeno na razvoju platformu, pa se nakon prevođenja proverava i ispravnost. Izvršne dataoteke se zbog toga povećavaju, ali se provera u vreme izvršavanja sprovodi vrlo efikasno.

Budući da sve greške koje izazivaju originalne klase moraju podržavati i nasleđene klase, pored osnovnih klasa grešaka izvedene su i neophodne klase `Exception`. Osim očigledno veoma uskog skupa klasa, mnoge su klase siromašnije od standardnih. Navedimo kao primer `java.lang.Math`. Pošto da nema realnih ni kompleksnih brojeva, izbačene su sve metode koje ih koriste, tako da je broj metoda sasvim smanjen. Budući da na nekim uređajima ne postoji sistem datoteka, nema ni klasa za rad s datotekama. Rad na mreži je prilično promenjen i pojednostavljen klasama iz paketa `javax.microedition.io`.

Nova generacija CLDC-a se razvija i pretpostavlja se da će unaprediti aktuelni CDLC. Očekuje se približavanje standardnoj Javi dodavanjem realnih brojeva i većih mogućnosti za obradu grešaka, naravno uz kompatibilnost sa starijom verzijom.

## SPECIFIKACIJA

SPECIFIKACIJA MIDP 2.0 je najnovija verzija MIDP-a. Ona nasleđuje i proširuje funkcionalnost izdanja MIDP 1.0. Pre te verzije, mnogi proizvođači su sopstvenim bibliotekama proširivali MIDP 1.0, budući da napredniji telefoni pružaju mnogo više mogućnosti od osnovnih modela i programiranje s minimalnim skupom komandi postaje previše komplikovano. MIDP 2.0 smanjuje potrebu za namenskim proširenjima. Zasad mali broj uređaja podržava verziju 2.0 MIDP-a.

Pre svega, u novoj verziji MIDP-a pojednostavljena je realizacija aplikacija. Uređaj može pokrenuti MIDlet da bi obradio određeni događaj poput prijema poruke ili sličnog. Značajan je broj novih API-ja. Podržani su i HTTPS, Socket API i dodat-



## MIDP 2.0

na funkcionalnost. Tu su klase za rad sa zvukom, kao i nove komponente grafičkog okruženja. Najznačajnije su klase `Container`, za pravljenje komponenata za obrasce i `GameCanvas`, koji proširuje klasu `Canvas` specifičnim elementima u vezani sa igricama. Klasa `Graphics` ima nove mogućnosti, od kojih se izdvaja kopiranje dela površine s jednog područja ekrana na drugi. Svi API-ji visokog nivoa su unapređeni i prošireni dodatnim elementima. Klasa `Canvas` podržava prikazivanje na čitavom ekranu (preko pokazivača napunjenosti baterije i signala mreže) što u prethodnoj verziji nije bio slučaj.

Klasa **Display** i sve druge klase korisničkog okruženja MIDP-a nalaze se u paketu **javax.microedition.lcdui**. Klasa **Display** sadrži metodu **setCurrent()** koja prikazuje sadržaj MIDleta. Ekran uređaja ne mora da prikaže ono što MIDlet zahteva – metod **setCurrent()** utiče samo na unutrašnje stanje interfejsa MIDleta i obaveštava upravljača aplikacije da bi MIDlet želeo objekat tipa **Displayable** bude prikazan. Razlika između **Display** i **Displayable** je da klasa **Display** predstavlja sadržaj ekrana uređaja, dok **Displayable** predstavlja nešto što se može prikazati na tom ekranu. MIDlet može pozvati metodu **isShown()** klase **Displayable** da utvrdi bi utvrdio da li je njen sadržaj zaista prikazan na ekranu.

## API KORISNIČKOG OKRUŽENJA ZA MIDP1.0

API KORISNIČKOG OKRUŽENJA ZA MIDP je podeljen u dva dela, API niskog i visokog nivoa. API visokog nivoa sadrži elemente kao što su dugmad, liste i polja za unos. Za razliku od AWT-a iz J2SE nije moguće kombinovati elemente visokog i niskog nivoa.

Postoje samo dva fiksna podnivoa elemenata API-ja visokog nivoa: **Screen** i **Item**. **Screen** je natklasa svih elemenata visokog nivoa koji popunjavaju čitav ekran, a **Item** je natklasa svih elemenata koji se mogu postaviti na **Form** ili **Alert**, specijalizovane naslednike klase **Screen**.

Klase visokog nivoa kao što je **Screen** i klasa niskog nivoa **Canvas** imaju zajedničku osnovnu klasu **Displayable**. Sve potklase **Displayable** popunjavaju čitav ekran uređaja. Potklase klase **Displayable** se mogu prikazati pozivanjem metoda **setCurrent()** objekta **Display**. Ekranu MIDleta se može pristupiti pozivanjem metode **getDisplay()**, kojoj se MIDlet prosleđuje kao parametar.

**Alert** je najjednostavniji naslednik klase **Screen**, a služi za prikazivanje dijaloga na izvesno vreme. Sastoji se od naslova, teksta i slike opciono. Moguće je podešiti period prikazivanja objekta toga tipa pre prikazivanje sledećeg ekrana. Evo kratkog primera koda:

```
Alert alert = new Alert
("HelloAlert!");
alert.setTimeout (Alert.FOREVER);
display.setCurrent (alert);
```

Najvažnija potklasa klase **Screen** je **Form**. Ona može sadržati proizvoljan broj elemenata, naslednika klase **Item** kao što su **StringItem**, **TextField** i **ChoiceGroup**. Sve klase sadrže natpis kome se pristupa pomoću metoda **setLabel()** i **getLabel()**. Klasa **Item** je apstraktna i ne možete praviti izvedene klase. Naslednici te klase se ne mogu ređati proizvoljno, već ih klasa **Form** raspoređuje po unutrašnjoj logici.

## UPRAVLJANJE PUTEM KOMANDI

SUPROTNO OD stonih računara, koji na velikom ekranu prikazuju kontrole i menije, u interakciji s korisnicima mobilnih telefona primenjen je drugačiji pristup. Većina uređaja podržava „meke tastere“, koji nemaju fiksnu funkcionalnost, već se ona određuje dinamički prema potrebama aplikacije. Neki uređaji uopšte i nemaju te tastere, a sve komande se izvršavaju kretanjem kroz menije. MIDP mora da apstrahuje sve te uređaje u jedinstveni interfejs primenljiv na sve njih, nezavisno od postojanja i broja mekih tastera. Zbog toga paket **lcdui** nema komande za rad sa tasterima ili menijima, već ima apstraktnu klasu **Command**. Realizacija tog interfejsa se može dodati bilo kom nasledniku klase **Displayable**. Neki od njih su klase **Form**, **List** i **TextBox** visokog nivoa API-ja ili **Canvas** u API-ju niskog nivoa.

Ne postoji mogućnost da se odrede parametri prikazivanja elemenata klase **Command** na ekranu. Klasa **Displayable** je zadužena za celokupno predstavljanje elemenata klase **Command** na ekranu – možete uticati samo na natpis komande, i na semantičke informacije. Semantičke informacije se sastoje od tipa i prioriteta. Prioritet dopušta uređaju da odredi koje su komande prikazane kao meki tasteri ako ima više komandi nego takvih tastera; za takve komande se automatski pravi poseban meni. Tip informacije je dodatni podatak koji pomaže uređaju pri određivanju načina prikazivanja komande. Na primer, ako je komanda **Exit** uvek na krajnjem levom mekom tasteru u standardnim aplikacijama, MIDP će podržati takvo prikazivanje i nadalje, da bi se zadržao ujednačen izgled i utisak.

## API NISKOG NIVOA

ZA RAZLIKU OD API-ja visokog nivoa, API niskog nivoa omogućava detaljno upravljanje prikazom sve do piksela.

Za tu namenu paket **lcdui** ima specijalan ekran, **Canvas**. Klasa **Canvas** ne sadrži metode za crtanje, ali sadrži metodu **paint()**, sličnu istoimenoj metodi AWT komponentenata. Kad god upravljač programa zaključa da je neophodno nešto nacrtati na ekranu, poziva se metoda **paint()** klase **Canvas**. Jedini parametar metode **paint()** je objekat klase **Graphics**. Za razliku od klase visokog nivoa, postoje mnoge sličnosti između AWT-a i API-ja niskog nivoa.

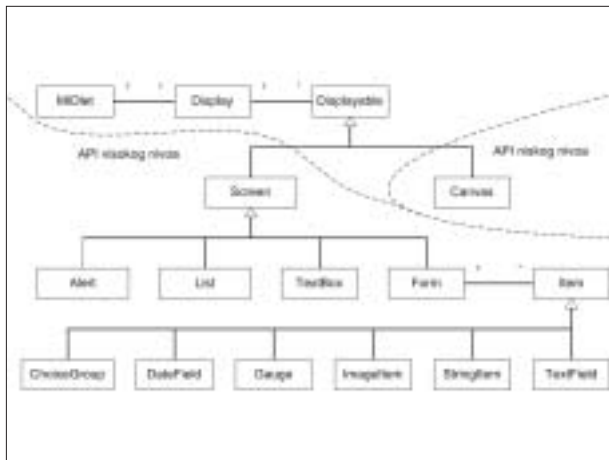
Objekat klase **Graphics** sadrži sve neophodne metode za crtanje po ekranu, kao što su **drawLine()** za crtanje linija, **fillRect()** za crtanje popunjenog pravougaonika ili **drawString()** za ispisivanje teksta.

Ipak, za razliku od AWT-a, **lcdui** ne dozvoljava mešanje grafike niskog i visokog ▶

```
PRIMER
MIDLETA

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class MIDletPrimerKlasa extends MIDlet {
    private CanvasCrttaElipsu canvasCrttaElipsu; // Platno
    private MenuList menuList; // Meni
    public MIDletPrimerKlasa() {}
    public void nacrtajElipsu() {
        canvasCrttaElipsu.start();
        // Postavimo izgled ekrana
        Display.getDisplay(this)
            .setCurrent (canvasCrttaElipsu);
    }
    void nazad() {
        // Vratimo na ekran meni
        Display.getDisplay(this).setCurrent (menuList);
    }
    public void izlaz() {
        destroyApp(true); // Unistimo aplikaciju
        notifyDestroyed(); // Obavestimo upravljača
    }
    protected void startApp() {
        menuList = new MenuList(this); // Pravimo meni
        canvasCrttaElipsu = new CanvasCrttaElipsu(this);
        // Postavljamo meni na ekran
        Display.getDisplay(this).setCurrent (menuList);
    }
    protected void pauseApp() {}
    protected void destroyApp(boolean promenljiva) {}
}
```



## HIJERARHIJA MIDP KLASA ▲

nivoa, što znači da se ne mogu prikazati na istom ekranu.

Upravljač programa zna da mora pozvati metodu **paint()** klase **Canvas** kada je instanca te klase prikazana na ekranu. Ipak, ponovno crtanje se može zahtevati bilo kada. Pozivanjem metode **repaint()** klase **Canvas** odmah obaveštavate sistem da je ponovno iscrtavanje neophodno i pokreće se metoda **paint()**, ali ne odmah, već nakon obrade tekućeg događaja. Može se desiti da bude poslato više zahteva za ponovno iscrtavanje pre nego što se **paint()** zaista pozove. To najčešće ne predstavlja problem, ali ako se radi sa animacijama, najbezbedniji način da se ekran osveži jeste pozivanjem metode **Display.callSerially()** ili iscrtavanje iz posebne niti. Iscrtavanje se može zatražiti metodom service **Repaints()**.

Klasa **Canvas** sadrži metode za obradu korisničkih događaja. Za ulazne tastere, klasa sadrži metode: **keyPressed()**, **keyReleased()** i **keyRepeated()**. Metoda **keyPressed()** poziva se kada je pritisnut neki taster, **keyRepeated()** kada se taster drži neko vreme, a **keyReleased()** se poziva kada se taster otpusti. Sve tri sadrže celobrojni parametar koji predstavlja Unicode znak vezan za određeni taster. Ako za taster ne postoji Unicode znak, prosleđuje se negativna vrednost.

MIDP definiše sledeće konstante tastera sa standardne ITU-T tastature: **KEY\_NUM0**, **KEY\_NUM1**, **KEY\_NUM2**, **KEY\_NUM3**, **KEY\_NUM4**, **KEY\_NUM5**, **KEY\_NUM6**, **KEY\_NUM7**, **KEY\_NUM8**, **KEY\_NUM9**, **KEY\_POUND** i **KEY\_STAR**. Aplikacija ne bi smela da pretpostavlja postojanje nijedne dodatne vrednosti. Veličina slova ili

## UKRATKO

### Ukratko

višestruko pritiskanje tastera nisu podržani. „Ime“ tastera možete dobiti metodom **getKeyName()**.

Neki tasteri mogu imati dodatno značenje u igrama. MIDP za tu svrhu sadrži konstante: **UP**, **DOWN**, **LEFT**, **RIGHT**, **FIRE**, **GAME\_A**, **GAME\_B**, **GAME\_C** i **GAME\_D**. Značenje u igri se može dobiti metodom **getGameAction()**. Veza između kodova tastera i akcija u igri zavisi od platforme, tako da različiti tasteri mogu biti povezani na iste igračke tastere na različitim uređajima. Na primer, neki mogu imati zasebne tastere sa strelicama, dok drugi mogu povezati pokrete sa tasterima s brojevima. Više tastera može biti povezano na istu igračku komandu. Igračka komanda se može ponovo pretvoriti u kod tastera metodom **getKeyCode()**. Tako se, poimenice, mogu dobiti imena tastera predviđenih za akcije u igri. Na primer sledeća poruka prikazuje ime tastera:

```
„pritisni „ + getKeyname (getKeyCode (GAME_A))
```

## EKRANSKO

## PLATNO

```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;

public class CanvasCrtaElipsu extends Canvas
    implements CommandListener {
    private MIDletPrimerKlasa midlet; // Aktivan MIDlet
    private Command komandaZaNazad; // Komanda za nazad

    public CanvasCrtaElipsu(MIDletPrimerKlasa midlet) {
        this.midlet = midlet;
        // Komanda za povratak na standardnoj poziciji
        komandaZaNazad = new Command(„Nazad“,
            Command.BACK, 1);
        addCommand(komandaZaNazad);
        setCommandListener(this);
    }

    // Osluskujemo komande
    public void commandAction(Command c, Displayable d)
    {
        if (c == komandaZaNazad) {
            midlet.nazad();
        }
    }

    public void start() {
        repaint(); // Jednostavna akcija
    }

    public void paint(Graphics g) {
        int sirina = getWidth();
        int visina = getHeight();
        // Prvo ocistimo pozadinu tako sto popunimo
        // pravougaonik velicine ekrana belom bojom
        g.setColor(0xFFFFFFFF);
        g.fillRect(0, 0, sirina, visina);
        // Pa iscrtamo elipsu
        g.setColor(0x0000FF);
        g.fillArc(0, 0, sirina, visina, 0, 360);
    }
}
```

Time biste obavestili korisnika da pritisne potrebno dugme iako unapred ne znate na koje dugme se to odnosi.

MIDP opciono podržava pokazivačke uređaje kakvih nema na mobilnim telefonima poput ekrana na dodir ili kuglice. Klasa **Canvas** sadrži tri metode za obradu tih događaja: **pointerPressed()**, **pointerDragged()** i **pointerReleased()**. Postupak obrade je sličan kao način reagovanja na događaj pritiskanja tastera, osim što se prosleđuju dva parametra – x i y koordinate pokazivača.

Prikazani objekat klase **Canvas** može, iz nekoliko razloga, otići u pozadinu – na primer, zato što je prikazan neki drugi objekat, ili pak uređaj prikazuje neki sistemski dijalog. U tim slučajevima objekat tipa **Canvas** dobija obaveštenje metodom **hideNotify()**. Kada se skriveni objekat (ponovo) prikaže, poziva se **showNotify()**. ▶

# RAZVOJ APLIKACIJA ZA NOKIJINE MODELE ▶

J2ME APLIKACIJE se mogu razvijati na više načina. Standardan razvojni alat firme Sun Microsystems možete preuzeti s adrese <http://java.sun.com/products/jtvi/>. To je verzija 2.1 Beta aktuelne J2ME palete alatki za bežične uređaje. Jednostavna je i sadrži veliki broj alatki za razvoj aplikacija za mobilne telefone i slične bežične uređaje. Pomoću nekoliko programčića lako se mogu razvijati MIDleti. Postoji nekoliko test uređaja, od onih koji prikazuju samo dve boje do ekrana u punom koloru. To nisu aparati koji postoje kao fizički uređaji, već prikazuju presek osobina sličnih naprava. Za razvoj programa koristi se J2SE, pa se zatim gotove klase testiraju korišćenjem tih alatki. Integrisana okruženja kao što su Sun One Studio, NetBeans IDE i JBuilderg podržavaju razvoj takvih aplikacija.

Vodeći proizvođači mobilnih telefona prave sopstvene alate za razvoj programa koji podržavaju specifičnosti njihovih modela. U njima se po pravilu nalazi veliki broj klasa koje odudaraju od standarda. One su uglavnom su nastajale zbog vrlo uskog opsega funkcionalnosti MIDP 1.0, ali i za MIDP 2.0 postoje mnoge dodatne biblioteke koje su u direknoj vezi s određenom platformom.

Sa adrese <http://www.forum.nokia.com/> možete preuzeti besplatne alate za razvoj aplikacija za uređaje ove firme.

Nokia Developer's Suite for Java 2 Micro Edition je skup alatki koje pomažu u pisanju Java aplikacija za Nokijine uređaje. To je modularna arhitektura, tako da podržava promene i dodavanje novih alatki. Podržava pisanje MIDleta u stilu alata za brz razvoj, od izrade klasa do aplikacije u celini. Podržani su bezbednosni mehanizmi MIDP 2.0.

Napisane aplikacije se mogu preneti na telefone korišćenjem RS232, IrDA i USB veza, a na servere putem FTP-a. Podržano je i pretvaranje MIDI i Ringing Tone XML datoteka u oblik za korišćenje u Java aplikacijama.

Za razvoj aplikacija neophodno je Java 2 izvršno okruženje 1.4.1 ili novije. Alatke možete ugraditi u postojeći razvojni alat – Sun One Studio Release 4 ili JBuilderg. Aplikacija se zatim može koristiti samostalno ili iz nekog od ta dva okruženja. Korišćenje iz okruženja podrazumeva naknadno uključivanje specifičnih Nokijinih API-ja da bi se aplikacije mogle prevesti. Svi alati će se u JBuilderu pojaviti u meniju Tools glavnog menija.



Programi se testiraju na emulatorima. ▲

Emulatori su, za razliku od Sunovog alata manje ili više realni telefoni. Nokia 7210 je potpuno funkcionalan telefon sa svim opcijama koje ima i fizički uređaj (jedino se ne može razgovarati preko njega). Drugi emulator predstavnik je serije 60, sa osobinama zajedničkim za tu seriju. Postoji veći broj primera sa izvornim kodom u kojima početnici tu mogu naći ideje za razvoj aplikacija.

Mnogi proizvođači telefona takođe omogućavaju besplatno preuzimanje razvojnih alata. Firma Siemens organizuje veliko takmičenje studenata u razvoju originalne aplikacija (igrica ili aplikacija drugog tipa).

Dovoljno je da ste student i da vaša aplikacija radi na Siemens telefonu. Konkurs se završava u martu, pa još nije kasno da se pridružite takmičenju. Detaljnije informacije o tome potražite na: [www.javamasters.org/](http://www.javamasters.org/).

Na Web lokaciji Nokije naći ćete detaljna uputstva za distribuiranje aplikacija, a slična mogućnost postoji i za aplikacije namenjene modelima drugih kompanija.

Budući da je proizvođačima u interesu da poboljšaju funkcionalnost svojih telefona, plasiraće vašu aplikaciju na tržište utvrđenim kanalima ako ona bude na zahtevanom nivou upotrebljivosti. Kvalitetni programi će bez problema će doći do velikog broja krajnjih korisnika. Ukoliko i ne zaradite, unapredićete svoj mobilni uređaj jedinstvenom aplikacijom, što je dovoljno za upuštanje u avanturu razvoja J2ME aplikacija. ■

## KOMANDNI MENI

```
import javax.microedition.lcdui.*;

class MenuList
    extends List
    implements CommandListener {
    private MIDletPrimerKlasa midlet;
    private Command komandaZaIzlaz;
    MenuList(MIDletPrimerKlasa midlet) {
        super(„Primer Midleta“, List.IMPLICIT);
        this.midlet = midlet;
        // Dodamo prvu stavku menija
        append(„Nacrtaj Elipsu preko celog ekrana“, null);
        append(„Prazna opcija“, null);
        komandaZaIzlaz = new Command(„Izlaz“,
            Command.EXIT, 1);
        addCommand(komandaZaIzlaz);
        setCommandListener(this);
    }

    public void commandAction(Command c, Displayable d) {
        if (c == List.SELECT_COMMAND) {
            // Ako je akcija biranje stavke iz liste
            int index = getSelectedIndex(); // Indeks opcije
            if (index != -1) { // Nikada ne bi smelo biti -1
                switch (index) {
                    case 0: // Izabrana je prva stavka u listi
                        midlet.nacrtajElipsu(); // Crta elipsu
                        break;
                    default:
                        break;
                }
            }
        }
        else if (c == komandaZaIzlaz) {
            midlet.izlaz();
        }
    }
}
```