2
Programski jezik Objective-C

iOS aplikacije se pišu na jeziku Objective-C pomoću *radnih okvira* (engl. *frameworks*) Cocoa Touch. Objective-C proširuje mogućnosti jezika C, a radni okviri Cocoa Touch su kolekcije klasa jezika Objective-C.

U ovom poglavlju naučićete osnovne elemente jezika Objective-C i napraviti aplikaciju RandomItems. Čak i ako poznajete Objective-C, pročitajte ovo poglavlje kako biste napravili klasu **`BNRItem`** koju ćete kasnije koristiti u ovoj knjizi.

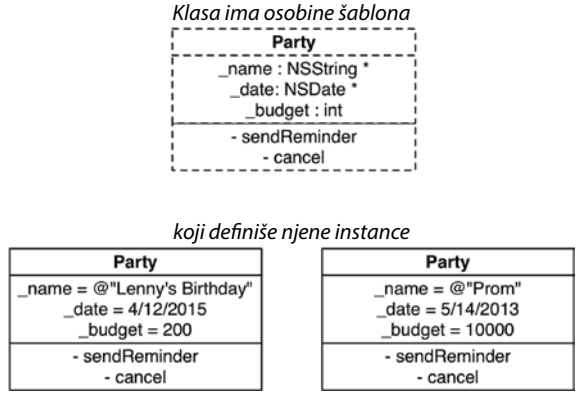
Da biste razumeli sadržaj ove knjige, trebalo bi da poznajete programski jezik C i osnovne principe objektno orijentisanog programiranja. Ako nije tako, preporučujemo da prvo pročitate knjigu *Objective-C Programming: The Big Nerd Ranch Guide*.

Objekti

Pretpostavimo da hoćete da organizujete proslavu (klasa Party), i da ta proslava ima nekoliko atributa koji su za nju jedinstveni – na primer, ime, datum i listu zvanica. Možete definisati i zadatke koje klasa Party treba da obavi, kao što su slanje podsetnika svim zvanicama putem elektronskih poruka, štampanje oznaka sa imenima zvanica, ili otkazivanje cele proslave.

Da radite u jeziku C, definisali biste strukturu s podacima koji opisuju proslavu. Ta struktura bi imala podatke članove (po jedan za svaki atribut proslave), a svaki podatak član imao bi ime i tip. Za definisanje jedne proslave upotrebili biste funkciju **`malloc`** i pomoću nje rezervisali jedinicu memorije, dovoljno veliku da se u nju smesti cela struktura.

Umesto strukture koja predstavlja proslavu, u jeziku Objective-C koristićete klasu. Klasa je poput modle za isecanje testa koja prozvodi objekte. Klasa **`Party`** definiše objekte, a ti objekti su njene instance. Svaka instanca klase **`Party`** može da sadrži podatke za jednu proslavu (slika 2.1).



Slika 2.1 Klasa i njene instance.

Jedna instanca klase **`Party`**, kao i svaki objekat, predstavlja određen deo podataka u memoriji, a vrednosti svojih atributa skladišti u *promenljivama instance* (engl. *instance variables*, ponegde se nazivaju *ivars*). U jeziku Objective-C, ispred imena promenljive instance najčešće se stavlja donja crta. Tako instanca klase **`Party`** može imati promenljive instance koje se zovu `_name`, `_date` i `_budget`.

Struktura u jeziku C i objekat u jeziku Objective-C predstavljaju jedinicu memorije. Struktura u jeziku C sadrži podatke članove, a svaki od njih ima ime i tip. Slično tome, objekat sadrži promenljive instance, i svaka od njih ima ime i tip.

Važna razlika između strukture u jeziku C i klase u jeziku Objective-C jeste to što klasa ima metode. Metoda je slična funkciji: ima ime, tip rezultata i listu očekivanih parametara. Metoda pristupa i promenljivama instance objekta. Ako želite da određeni objekat izvrši kôd svoje metode, tom objektu šaljete poruku.

Korišćenje instanci

Da biste upotrebili instancu klase, morate imati promenljivu koja pokazuje na taj objekat. Pokazivačka promenljiva u memoriji skladišti lokaciju objekta, a ne sâm objekat. (Ona „pokazuje“ na objekat.) Promenljiva koja pokazuje na objekat **`Party`** deklariše se na sledeći način:

```
Party *partyInstance;
```

Definisanjem ovog pokazivača ne pravi se objekat **`Party`**, nego samo promenljiva koja može da pokaže na taj objekat.

Promenljiva se zove `partyInstance`. Njeno ime ne započinje donjom crtom, jer to nije promenljiva instance nego promenljiva koja ima ulogu pokazivača na instancu klase **`Party`**.

Definisanje objekata

Objekat ima svoj životni vek: on se definiše, šalju mu se poruke i uklanja se kada više ničemu ne služi.

Da biste definisali objekat, klasi šaljete poruku **`alloc`**. Kao odgovor na tu poruku, klasa definiše objekat u memoriji (u hip memoriji, kao što bi to uradila funkcija **`malloc()`**), i daje vam adresu tog objekta koja se čuva u promenljivoj:

```
Party *partyInstance = [Party alloc];
```

Pokazivač na instancu definišete kako biste mogli da joj šaljete poruke. Prva poruka koja se uvek šalje alociranoj instanci jeste inicijalizaciona poruka. Iako slanje poruke **`alloc`** klasi definiše instancu, ta instanca nije spremna za rad dok se ne inicijalizuje.

```
Party *partyInstance = [Party alloc];
[partyInstance init];
```

Budući da se objekat pre upotrebe mora alocirati i inicijalizovati, ove dve poruke uvek se navode zajedno u jednom redu programskog koda.

```
Party *partyInstance = [[Party alloc] init];
```

Spajanje dveju poruka u jednom redu koda naziva se *slanje ugnežđenih poruka* (engl. *nested message send*). Unutrašnje zagrade se izračunavaju prve, pa se poruka **`alloc`** prva šalje klasi **`Party`**. Rezultat ove operacije biće pokazivač na novu, neinicijalizovanu instancu klase **`Party`**, kojoj se zatim šalje poruka **`init`**. Rezultat je pokazivač na inicijalizovanu instancu koja se skladišti u pokazivačkoj promenljivoj.

Slanje poruka

Šta se radi sa instancom koja je inicijalizovana? Šalju joj se dodatne poruke.

Pogledajmo prvo građu poruke. Poruka se uvek nalazi u uglastim zagradama i sastoji se od tri dela:

prijemnik (engl. *receiver*)	pokazivač na objekat koji treba da izvrši metodu
selektor (engl. *selector*)	ime metode koja se izvršava
argumenti (engl. *arguments*)	vrednosti koje se kao parametri prosleđuju metodi

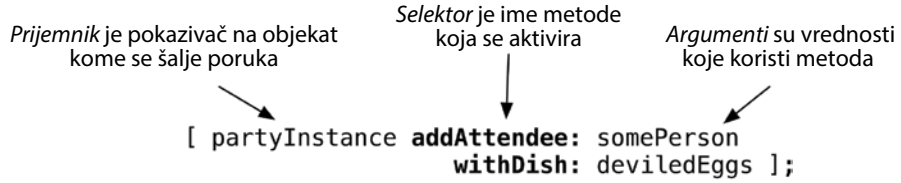
Na primer, proslava može imati listu učesnika, koje možete dodavati tako što ćete objektu proslave slati poruku **`addAttendee:`**.

```
[partyInstance addAttendee:somePerson];
```

Slanje poruke **addAttendee:** objektu partyInstance (prijemnik) aktivira metodu **addAttendee:** (koju imenuje selektor) i prosleđuje vrednost somePerson (argument).

Poruka **addAttendee:** ima jedan argument. Metode u jeziku Objective-C mogu imati više argumenata, ili ih uopšte ne moraju imati. Na primer, poruka **init**, nema nijedan argument.

Učesnici proslave bi trebalo da obaveste domaćina o tome koje će jelo doneti. Klasa **Party** može imati još jednu metodu, **addAttendee:withDish:**. Ova poruka ima dva argumenta: učesnika i jelo koje on donosi. Svaki argument se uparuje sa natpisom u selektoru, a svaki natpis završava dvotačkom. Selektor se sastoji od natpisa koji se obrađuju zajedno (slika 2.2).



Slika 2.2 Slanje delova poruke.

Uparivanje natpisa i argumenata važna je osobina jezika Objective-C. Ova metoda u drugim jezicima mogla bi se izraziti na sledeći način:

```
partyInstance.addAttendeeWithDish(somePerson, deviledEggs);
```

U tim drugim jezicima nije sasvim očigledno šta je svaki od argumenata koji se šalju navedenoj funkciji. U jeziku Objective-C, svaki argument se uparuje sa odgovarajućim natpisom.

```
[partyInstance addAttendee:somePerson withDish:deviledEggs];
```

Kada se naviknete na ovu novu osobinu programskog jezika Objective-C, shvatićete koliko je sve jasnije kada se argumenti umetnu u selektor. Zapamtite da se za svaki par uglastih zagrada šalje samo jedna poruka. Iako metoda **addAttendee: withDish:** ima dva natpisa, ona i dalje predstavlja jednu poruku, a kao rezultat slanja te poruke izvršava se samo jedna metoda.

U jeziku Objective-C ime metode je jedinstveno. Stoga klasa ne može imati dve metode sa istim imenom. Dva imena metoda mogu da sadrže iste pojedinačne natpise, ukoliko se ta imena kao celine razlikuju. Na primer, naša klasa **Party** ima dve metode – **addAttendee:** i **addAttendee:withDish:**. To su dve različite metode, s potpuno različitim programskim kodom.

Vodite računa i o razlikama između poruke i metode: metoda je odlomak koda koji se može izvršiti, a poruka je postupak u kome se od klase ili objekta traži da izvrše metodu. Ime poruke se uvek podudara sa imenom metode koja se izvršava.

Uništavanje objekata

Da biste uništili objekat, za promenljivu koja na njega pokazuje zadajte vrednost `nil`.

```
partyInstance = nil;
```

Ovaj kôd uništava objekat na koji pokazuje promenljiva `partyInstance` i kao vrednost ove promenljive zadaje `nil`. (Sve je u stvarnosti zapravo mnogo složenije, i zato ćete o upravljanju memorijom naučiti više u narednom poglavlju.)

Vrednost `nil` je prazan pokazivač. (Programeri na jeziku C znaju ovu vrednost kao `NULL`, a Java programeri kao `null`.) Pokazivač koji ima vrednost `nil` najčešće predstavlja odsustvo objekta. Na primer, proslava se uvek odvija na određenom mestu (Venue). I dok organizator (Organizer) ne odluči gde će ugostiti učesnike proslave, promenljiva `venue` može da pokazuje na `nil`. To omogućava da uradite sledeće:

```
if (venue == nil) {
    [organizer remindToFindVenueForParty];
}
```

Programeri na jeziku Objective-C obično koriste kraći način kako bi odredili da li pokazivač ima vrednost `nil`:

```
if (!venue) {
    [organizer remindToFindVenueForParty];
}
```

Budući da operator `!` znači „not“, ovaj uslov može da glasi „ako ne postoji mesto proslave“, i on će biti ispunjen ako promenljiva `venue` ima vrednost `nil`.

Ukoliko šaljete poruku promenljivoj čija je vrednost `nil`, ništa se neće desiti. U drugim jezicima, slanje poruke praznom pokazivaču nije dozvoljeno, pa ćete videti nešto poput sledećeg:

```
// Da li je mesto proslave različito od vrednosti nil?
if (venue) {
    [venue sendConfirmation];
}
```

U jeziku Objective-C ova provera je suvišna, jer se zanemaruje poruka poslata vrednosti `nil`. Zato poruku možete poslati i bez proveravanja ove vrednosti:

```
[venue sendConfirmation];
```

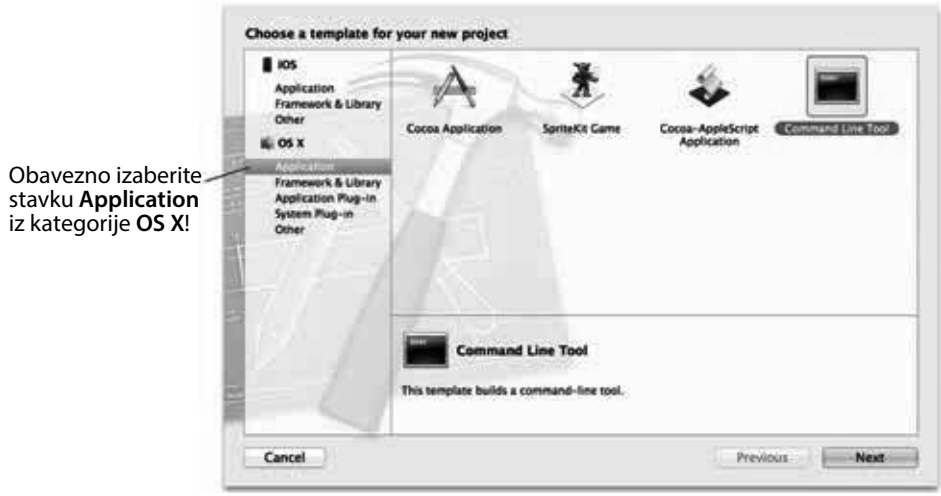
Ako mesto odvijanja proslave još nije izabrano, nećete nigde poslati potvrdu. (Znači, ukoliko vaš program ne radi ništa, a trebalo bi da radi, krivac bi mogao biti neočekivani pokazivač `nil`.)

Ostavimo teoriju po strani. Vreme je za praktičnu primenu i izradu novog projekta.

Izrada projekta RandomItems

Novi projekat nije iOS aplikacija, nego program komandne linije. On omogućava da se usredsredite na Objective-C, a da vam korisnički interfejs ne odvlači pažnju. Koncepti koje ćete naučiti u ovom poglavlju i u poglavlju 3 veoma su važni za razvoj aplikacija u operativnom sistemu iOS. Na izradu iOS aplikacija vratićete se u poglavlju 4.

Otvorite Xcode i izaberite File → New → Project... Na levoj strani ekrana, u odeljku OS X, pritisnite Application i iz gornjeg panoa izaberite Command Line Tool, kao što je prikazano na slici 2.3. Pritisnite Next.

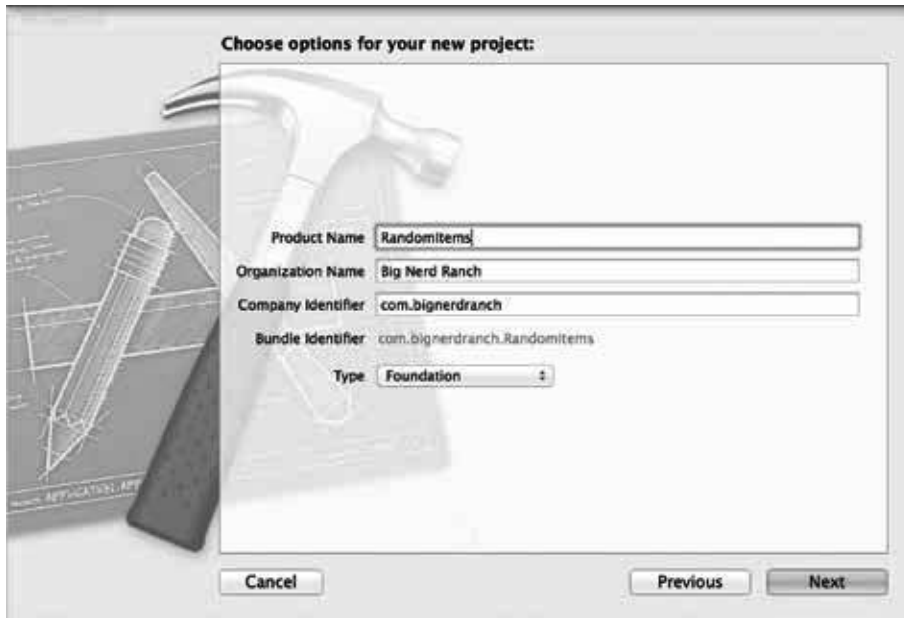


Slika 2.3 Izrada alatke komandne linije.

Na sledećoj strani prozora, dodelite projektu ime RandomItems i kao tip projekta zadajte Foundation (slika 2.4).

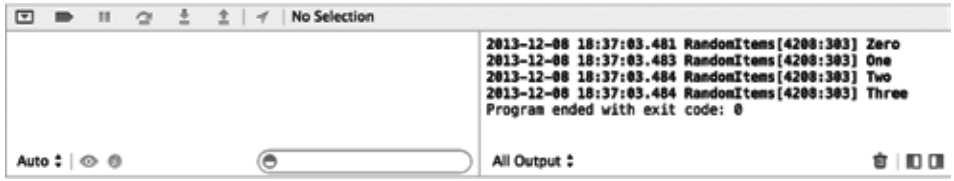
Kada pritisnete Next, od vas će se tražiti da snimite projekat. Čuvajte ga na bezbednom mestu, jer ćete delove njegovog koda koristiti u narednim projektima.

U početnoj verziji projekta RandomItems napravićete *niz* (engl. *array*) koji sadrži četiri *znakovna niza* (engl. *string*). Niz je uređena lista pokazivača na druge objekte. Pokazivačima u nizu pristupa se pomoću indeksa. (U drugim programskim jezicima, niz se može nazivati lista ili vektor.) Vrednost indeksa za prvi član niza uvek je nula.



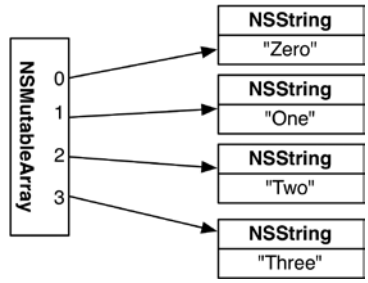
2.4 Dodeljivanje imena projektu.

Nakon što definišete niz, prolazićete kroz njega u petlji i ispisivati svaki znakovni niz. Rezultat će se prikazati na konzoli u okruženju Xcode:



Slika 2.5 Prikaz rezultata na konzoli.

Za program će vam biti potrebno pet objekata: jedna instanca klase **`NSMutableArray`** i četiri instance klase **`NSString`** (slika 2.6).

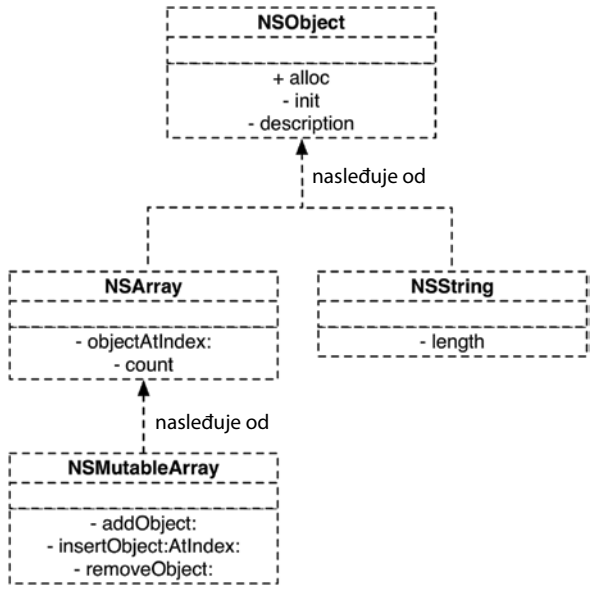


Slika 2.6 Instanca klase **`NSMutableArray`** sa pokazivačima na instance klase **`NSString`**.

U jeziku Objective-C, niz ne sadrži objekte koji mu pripadaju nego pokazivače na objekte. Kada se nizu doda objekat, adresa tog objekta u memoriji skladišti se unutar niza.

Razmotrimo klase **`NSMutableArray`** i **`NSString`**. **`NSMutableArray`** je *potklasa* (engl. *subclass*) klase **`NSArray`**. Klase su deo hijerarhije, i svaka od njih ima jednu i samo jednu natklasu – osim klase **`NSObject`**, koja je osnovna klasa cele hijerarhije. Klasa nasleđuje karakteristike svoje natklase.

Na slici 2.7 prikazana je hijerarhija klasa **`NSMutableArray`** i **`NSString`**, zajedno s nekoliko metoda koje primenjuje (implementira) svaka od njih.



Slika 2.7 Hijerarhija klasa.

Uloga klase **`NSObject`** najvišeg nivoa jeste da preda osnovne karakteristike svih objekata u Cocoa Touch. Svaka klasa nasleđuje metode i promenljive instance definisane u klasi **`NSObject`**. Ova klasa primenjuje dve metode, **`alloc`** i **`init`** (slika 2.8), i zato se one mogu koristiti za definisanje instance svake klase.

Potklasa dodaje metode i promenljive instance da bi proširila karakteristike svoje natklase:

- **`NSString`** dodaje karakteristiku za skladištenje i obradu znakovnih nizova, uključujući metodu **`length`** koja daje broj znakova u znakovnom nizu.
- **`NSArray`** dodaje karakteristiku za uređene liste, uključujući pristup objektu na poziciji zadatoj vrednošću indeksa (**`objectAtIndex:`**) i izračunavanje broja objekata u nizu (**`count`**).
- **`NSMutableArray`** proširuje karakteristike klase **`NSArray`** mogućnostima za dodavanje i uklanjanje pokazivača.

Definisanje i popunjavanje niza

Navedene klase ćemo praktično upotrebiti u pravom programskom kodu. U navigatoru projekta pronađite i izaberite datoteku `main.m`. Kada se ona otvori, primetićete da je deo koda već napisan za vas, pre svega funkcija **`main`**, koja je ulazna tačka svake aplikacije pisane na jezicima C ili Objective-C.

U datoteci `main.m` obrišite red koda koji sadrži funkciju **`NSLog`** („Hello, World!“) i zamenite ga kodom koji definiše instancu klase **`NSMutableArray`**, dodaje ukupno četiri objekta promenljivom nizu, a zatim uništava niz.

```
#import <Foundation/Foundation.h>


int main (int argc, const char * argv[])
{
    @autoreleasepool {

        // umetni kôd ovde...
        NSLog(@"Hello, World!");

        // Definiši objekat promenljivog niza, sačuvaj njegovu
        // adresu u promenljivoj items
        NSMutableArray *items = [[NSMutableArray alloc] init];

        // Pošalji poruku addObject: objektu NSMutableArray na koji
        // pokazuje promenljiva items, i svaki put prosledi znakovni niz
        [items addObject:@"One"];
        [items addObject:@"Two"];
        [items addObject:@"Three"];

        // Pošalji drugu poruku, insertObject:atIndex:, tom istom
        // objektu niza
        [items insertObject:@"Zero" atIndex:0];

        // Uništi objekat promenljivog niza
        items = nil;
    }
    return 0;
}
```

Objekti koje ste dodali i umetnuli u niz predstavljaju instance klase **`NSString`**. Instancu klase **`NSString`** možete definisati tako što ćete kao prefiks ispred niza znakova dodati simbol `@`:

```
NSString *myString = @"Hello, World!";
```

Iteracija kroz niz

Napravili ste niz items čiji su članovi znakovni nizovi. Svakom znakovnom nizu ćete pristupiti i ispisati ga na konzoli u zasebnom ponovljenom koraku.

Ovu operaciju biste mogli da napišete u obliku petlje `for`:

```
for (int i = 0; i < [items count]; i++) {
    NSString *item = [items objectAtIndex:i];
    NSLog(@"%@", item);
}
```

Budući da je vrednost indeksa za prvi član niza nula, brojač započinje vrednošću `0` i zaustavlja se kada vrednost postane za 1 manja od rezultata slanja poruke **`count`** članovima niza. Nakon toga, unutar petlje šaljete poruku **`objectAtIndex:`** kako biste dobili objekat s tekućom vrednošću indeksa pre nego što ga ispišete.

Informacija dobijena od poruke **`count`** važna je zato što će se, u slučaju da je vrednost indeksa traženog objekta veća ili jednaka broju objekata u nizu, generisati izuzetak. (U nekim programskim jezicima, izuzeci se stalno generišu i obrađuju. U jeziku Objective-C, izuzeci se posmatraju kao greške programera i ne obrađuju se u vreme izvršavanja programa nego se ispravljaju u kodu. O izuzecima ćemo govoriti na kraju ovog poglavlja.)

Navedeni kôd će pravilno raditi, ali Objective-C ima bolje rešenje za iteraciju kroz niz, koje se zove *brzo numerisanje* (engl. *fast enumeration*). Brzo numerisanje ima kraću sintaksu od klasične petlje `for` i manju verovatnoću pojave greške. U nekim slučajevim, takav kôd će biti i brži.

U datoteku `main.m` dodajte sledeći kôd, koji sadrži brzo numerisanje i omogućava iteraciju kroz niz `items`:

```
int main (int argc, const char * argv[])
{
    @autoreleasepool {

        // Definiši objekat promenljivog niza, sačuvaj njegovu
        // adresu u promenljivoj items
        NSMutableArray *items = [[NSMutableArray alloc] init];

        // Pošalji poruku addObject: objektu NSMutableArray na koji
        // pokazuje promenljiva items, i svaki put prosledi znakovni niz
        [items addObject:@"One"];
        [items addObject:@"Two"];
        [items addObject:@"Three"];

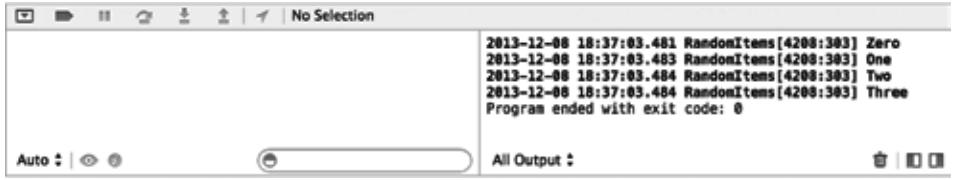
        // Pošalji drugu poruku, insertObject:atIndex:, tom istom
        // objektu niza
        [items insertObject:@"Zero" atIndex:0];
```

```
        // Za svaki član niza items...
        for (NSString *item in items) {
            // ispiši opis člana niza
            NSLog(@"%@", item);
        }

        items = nil;
    }
    return 0;
}
```

Brzo numerisanje ima svoje ograničenje: ne možete ga koristiti ako treba da dodate ili uklonite objekte unutar petlje. Ako to pokušate, generisaće se izuzetak. Za dodavanje ili uklanjanje objekata upotrebite običnu petlju `for` s promenljivom kao brojačem.

Prevedite i izvršite aplikaciju (Command-R). Na dnu prozora u okruženju Xcode, prikazaće se novo okno, koje se naziva *oblast za ispravljanje grešaka* (engl. *debug area*). Desno od ove oblasti nalazi se *konzola* (engl. *console*) s rezultatima.



Slika 2.8 Prikaz rezultata na konzoli.

Veličinu oblasti za ispravljanje grešaka i njenih panoa možete menjati povlačenjem ivica njihovih okvira. (Tako možete promeniti dimenzije svake oblasti u prozoru radnog prostora.)

Završili ste prvu fazu programa RandomItems. Pre nego što nastavite rad, reći ćemo nešto o funkciji **`NSLog`** i znakovnim nizovima za formatiranje.

Znakovni nizovi za formatiranje

Funkcija **`NSLog()`** ima promenljiv broj argumenata i upisuje znakovni niz u *dnevnik* (engl. *log*). U okruženju Xcode, dnevnik se prikazuje na konzoli. Prvi argument funkcije **`NSLog()`** obavezan je. On je instanca klase **`NSString`** i zove se *znakovni niz za formatiranje* (engl. *format string*).

Znakovni niz za formatiranje sadrži tekst i *tokene* (engl. *tokens*). Ispred svakog tokena (koji se naziva i specifikacija formata), navodi se znak za procenat (`%`). Svaki dodatni argument prosleđen funkciji zamenjuje po jedan token u znakovnom nizu za formatiranje.

Pomoću tokena se zadaju tipovi argumenata koji im odgovaraju. Navešćemo primer:

```
int a = 1;
float b = 2.5;
char c = 'A';
NSLog(@"Integer: %d Float: %f Char: %c", a, b, c);
```

Rezultat će biti

```
Integer: 1 Float: 2.5 Char: A
```

Znakovni nizovi za formatiranje u jeziku Objective-C rade na isti način kao u jeziku C. Međutim, Objective-C ima dodatni token, `%@`. On odgovara argumentu tipa „pokazivač na svaki objekat."

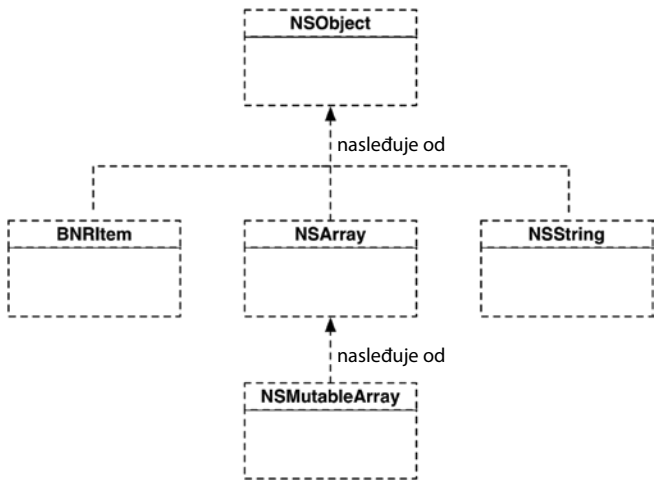
Kada se token `%@` nađe u znakovnom nizu za formatiranje, token se ne zamenjuje odgovarajućim argumentom, nego se tom argumentu šalje poruka **`description`**. Metoda **`description`** vraća instancu klase **`NSString`** koja zamenjuje token.

Budući da se poruka šalje argumentu koji odgovara tokenu `%@`, taj argument mora biti objekat. Ako ponovo pogledate sliku 2.7, videćete da je **`description`** metoda klase **`NSObject`**. To znači da svaka klasa primenjuje metodu **`description`**, i zato token `%@` možete koristiti sa svakim objektom.

Definisanje potklase u jeziku Objective-C

U ovom odeljku definisaćete novu klasu **`BNRItem`**, koja će biti potklasa klase **`NSObject`**.

Instanca klase **`BNRItem`** predstavljaće nešto što neka osoba ima u stvarnom svetu – na primer, računar, bicikl, ranac i slično. U kontekstu obrasca Model-View-Controller, **`BNRItem`** je klasa modela. Instanca klase **`BNRItem`** skladišti informacije o predmetima u vlasništvu date osobe.



Slika 2.9 Hijerarhija klasa s novom klasom **`BNRItem`.**