Objektno orijentisano programiranje 2

Pregled jezika Java

Primitivni tipovi podataka

- Primitivni (ugrađeni, ne-klasni, vrednosni) tipovi:
- `boolean` - vrednosti `true` ili `false`
- `char` - znak iz *Unicode* skupa (kodiranje UTF-16)
- `byte` - 8-bitni označeni ceo broj (*signed integer*)
- `short` - 16-bitni označeni ceo broj (*signed integer*)
- `int` - 32-bitni označeni ceo broj (*signed integer*)
- `long` - 64-bitni označeni ceo broj (*signed integer*)
- `flaot` - 32-bitni floating point (IEEE 754-1985)
- `double` - 64-bitni floating point (IEEE 754-1985)
- Stroga provera tipova, ali dozvoljene bezbedne implicitne konverzije
- Za svaki primitivan tip postoji i klasa-omotač (npr. `Integer` za `int`)
- Automatsko pakovanje/raspakivanje

Literali i konstante

- Literali – leksički simboli koji bukvalno predstavljaju napisano
- Primeri literala:
  `true, 1_000_000, 6.28, 6.28d, 3.14f, 'a', '\n', "Zdravo"`
- Imenovane konstante: modifikator `final`, inicijalizacija konstantnim izrazom
- Polja klasa kao simboličke konstante: modifikatori `static final`
- Primer:
  ```
  static final double pi=3.14;
  ```
- Logički povezane konstante mogu da budu grupisane unutar klase
- Primer:
  ```
  class BojeKarata{
       final static int PIK = 2;
       final static int KARO = 3;
       final static int HERC = 4;
       final static int TREF = 5;
  };
  ```
- Pristup:
  `BojeKarata.HERC`, `BojeKarata.TREF`, itd.

Unicode skup znakova

- Tradicionalni jezici koriste ASCII skup znakova
- Program na Javi koristi Unicode skup (UTF-16)
- Unicode je internacionalni standard
- UTF-16, način kodiranja
  - kodovi koriste 16 ili 32 bita da predstave Unicode znake
- Prethodni primer sa konstantom `pi`:
  ```
  static final double π = 3.14;
  ```
- Kod na Javi pisan pomoću ASCII (7-bit) skupa se translira u Unicode pre prevođenja

Operatori

- Po opadajućim prioritetima:
  - ind., pristup, poziv: `[]` `.` `(argumenti)`
  - unarni postfiksni: *izraz*`++` *izraz*`--`
  - unarni prefiksni: `++`*izraz* `--`*izraz* `+`*izraz* `-`*izraz* `~` `!`
  - kast: `(tip)`*izraz*
  - multiplikativni: `*` `/` `%`
  - aditivni: `+` `-`
  - pomerački: `<<` `>>` `>>>`
  - relacioni: `<` `>` `>=` `<=` `instanceof`
  - jednakost: `==` `!=`
  - logičko ili bitsko AND: `&`
  - logičko ili bitsko XOR: `^`
  - logičko ili bitsko OR: `|`
  - logičko uslovno AND: `&&`
  - logičko uslovno OR: `||`
  - uslovni: `?:`
  - dodela: `=` `+=` `-=` `*=` `/=` `%=` `>>=` `<<=` `>>>=` `&=` `^=` `|=`
- Svi binarni osim dodela su levo-asocijativni

Komentari

- Prevodilac ih ignoriše
- Tri stila:
  - `//` komentar u liniji – proteže se do kraja linije
  - `/*` komentar koji može da obuhvati više linija `*/`
  - `/**` dokumentacioni komentar
    - namenjen opisu entiteta (klase, metoda,...) koji sledi `*/`
- *javadoc* alat na osnovu dokumentacionih komentara generiše HTML dokumentaciju

Tok kontrole

- Instrukcije se završavaju `;`
- Jednostavne instrukcije – izrazi
- Sekvence instrukcija – blokovi u zagradama `{}`
- Kontrolne strukture kao na jezicima C i C++
  - selekcije: `if-else` i `switch`
  - iteracije: `while`, `do-while` i `for`
- Selektorska promenljiva naredbe `switch` može da bude tipa `String` (od v7)
- *foreach* petlja (Java 5.0) za iteriranje kroz zbirke (kolekcije) i nizove
  - bez korišćenja iteratora i indeksa (primer na sledećem slajdu)
- Instrukcije skoka
  - `break` i `continue` koje mogu da imaju i labelu naredbe iz koje se iskače
  - `goto` instrukcija ne postoji (rezervisana reč, kao i `const`, koja se ne koristi)
- Labele služe samo za iskakanje iz petlji pomoću `break` i `continue`

Primer *foreach* petlje

- Konvencionalna `for` petlja:

```
public int sumaNiza(int niz[], int n){
     int suma=0;
     for (int i=0; i<n; i++) suma+=niz[i];
     return suma;
}
```

- Nova *foreach* petlja:

```
public int sumaNiza(int niz[]){
     int suma=0;
     for (int e: niz) suma+=e;
     return suma;
}
```

Klase i objekti

- Klase definišu tipove (apstrakcije)
- Objekti su primerci klasa (pojave)
- Primer klase – tačka u 2D:

```
class Tacka{ public double x,y; }
```

  - ova klasa ima dva javna polja i ne sadrži metode
- Prava pristupa se deklarišu za svaki član klase
  - koriste se modifikatori `public`, `protected` i `private`
  - deklaracija `public` znači da svaki kod sa pristupom objektu može da pristupi članu
  - ostala prava pristupa umanjuju pristupačnost člana

Stvaranje i uništavanje objekata

- Objekti se stvaraju korišćenjem ključne reči, alokatora `new`
- Primer:

```
Tacka centar = new Tacka();
```

- Objekti su smešteni u memoriji za dinamičku alokaciju (*heap*)
- Objektima se pristupa preko referenci
  - reference su slične pokazivačima na C++
  - promenljiva tipa neke klase sadrži referencu na objekat ili `null`
  - referenca može da pokazuje na razne objekte u toku životnog veka
- Objekti se ne uništavaju eksplicitno, uklanja ih sakupljač đubreta
  - ako na objekat ne ukazuje ni jedna referenca – može da se ukloni
  - sakupljač đubreta je posebna programska nit (radi u pozadini)

Metodi

- Metodi pristupaju implementacionim detaljima klase (objekta)
- Potpis:
  - ime metoda, broj i tipovi parametara metoda
- Primer (u klasi `Tacka`):
  `public void inicijalizuj(){x=0;y=0;}`
  - metod `inicijalizuj()` nema parametre i nema rezultat
- Unutar metoda, članovi klase mogu da se imenuju direktno (bez reference na objekat), kao i u C++
- Objekat čiji se metod poziva se naziva primalac poruke (*receive*
- U metodima nisu dozvoljene statičke lokalne promenljive
- Statički metodi – kao u jeziku C++
- Parametri se prenose po vrednosti
  - ako je parametar referenca – sam objekat se prenosi po referenci

Objekti `String`

- Java obezbeđuje klasu `String`
  - podrška za podatke tipa niski (sekvenci znakova)
- Operator `+` se koristi za nadovezivanje (konkatenaciju) niski
- Niske mogu da se stvaraju u memoriji literala ili na hipu:

```
String ime="Petar";//u memoriji literala
String ime= new String("Petar");//na hipu
```

- Objekti tipa `String` mogu samo da se čitaju
  - na primer: `ime += " Petrović"`
    formira se novi objekat niske `"Petar Petrović"`
- Za promenljive niske se koriste klase:
  - `StringBuffer` (bezbedna za niti) i `StringBuilder` (nije bezbedna)
- U svakoj klasi može da se napiše metod `toString()`
  - konvertuje dati objekat u nisku na željeni način

Primer sa niskama

```
public class Niske {
  public static void main(String args[]) {
    String s1="Petar";
    String s2="Petar";
    String s3=new String("Petar");
    String s4="Petar Petrović";
    String s5="Petar"+" Petrović";
    String s6=s1+" Petrović";
    String s7=s3+" Petrović";
    boolean b1=s1==s2, b2=s1==s3, b3=s4==s5, b4=s4==s6, b5=s4==s7;
    System.out.println(b1+","+b2+","+b3+","+b4+","+b5);
  }
}
```

Ispisuje se: `true, false, true, false, false`

Nizovi

- Niz je objekat koji predstavlja seriju referenci ili vrednosti nekog tipa
- Nizovski objekti imaju javno polje `length` koje može samo da se čita
  - polje daje informaciju o broju elemenata niza
- Indeksi su celi brojevi u opsegu između `0` i `length-1`
- Kontrola proboja opsega indeksa:
  - izuzetak `IndexOutOfBoundsException` ukazuje da je indeks van opsega
- Primer:

```
class Špil{
  final static int VELIČINA_ŠPILA = 32;
  Karta[] karte=new Karta[VELIČINA_ŠPILA];
  ...
  public void print(){for(Karta k: karte) System.out.println(k);}
}
```

- Po stvaranju niza objekata svi elementi niza (reference) imaju vrednost `null`
- Mogu odmah da se kreiraju i objekti na koje pokazuju reference u nizu:

```
X[] x2= new X[]{new X(1), new X(2)};
```

Višedimenzioni nizovi

- 2D matrica `m2` i 3D matrica `m3`: `X[][] m2; X[][][] m3;`
  - `m2` je referenca na niz referenci na vrste koje su nizovi referenci na objekte klase `X`
- Višedimenzioni nizovi ne moraju da budu „pravougaoni“
  - mogu da imaju vrste različitih dužina
- Primer

```
public class Matrica2D {
    public static void main(String[] args) {
        int[][] m = { {1, 2, 3}, {4, 5}, {7, 8, 9} };
        for (int[] n: m) { for (int e: n) System.out.print(e + " ");
            System.out.println();
        }
    }
}
```

```
Izlaz:
1 2 3
4 5
7 8 9
```

Izvođenje klasa (proširivanje)

- Jedna od glavnih dobiti OO programiranja
  - proširenje ponašanja postojeće klase
- Nova (proširena) klasa nasleđuje sva polja i metode originalne klase
- Izvedena klasa može da:
  - proširi strukturu podataka osnovne klase dodavanjem polja
  - proširi ugovor osnovne klase dodavanjem novih metoda
  - redefiniše nasleđeno ponašanje redefinisanjem metoda osnovne klase
- Objekat izvedene klase može da zameni objekat osnovne klase
  - npr. ako metod očekuje parametar tipa osnovne klase može da mu se prosledi argument tipa izvedene klase
- Polimorfizam
  - objekat na koji upućuje referenca može da ispoljava više(poly-) oblika(-morph)
  - metod se poziva na osnovu stvarnog tipa objekta, a ne tipa reference
- Java podržava samo jednostruko nasleđivanje implementacije
  - samo jedna klasa može da se proširi

Klasa `Object`

- Klase koje ne proširuju eksplicitno druge klase, implicitno proširuju klasu `Object`
- `Object` se nalazi u korenu hijerarhije klasa
- `Object` je najopštija klasa za reference koje mogu da upućuju na objekat proizvoljne klase
- Primer:

```
Object o = new Tacka();
o = "Petar Petrović"
```

  - legalno je referencu `o` postaviti da upućuje na objekat tipa `Tacka` i na objekat tipa `String`

Interfejsi

- Ponekad je korisno deklarisati ugovor klase - metode koje mora da podrži
- Implementacija metoda je irelevantna, za klijenta je bitna njihova deklaracija
- Primer: metodi sa istom deklaracijom mogu da se primene na razne zbirke
  - lančanu listu vrednosti
  - heš-tabelu vrednosti
- Interfejs je nalik klasi, ali sadrži (u principu) deklaracije metoda
  - sličan je apstraktnoj klasi sa svim apstraktnim metodima bez promenljivih polja
- Interfejs je stvar ugovora (čistog ugovora – do verzije Java 8)
  - deklariše koji metodi su podržani klasom koja će da implementira taj interfejs
- Interfejs je tip
  - mogu da se definišu reference tipa nekog interfejsa
  - takve reference mogu da upućuju na objekte onih klasa koje implementiraju taj interfejs
- Klase implementiraju interfejse, tj. ostvaruju njihove ugovore

Izuzeci

- Izuzetak u Javi je isključivo objekat neke klase
- Klase izuzetaka se izvode iz klase `Throwable`
  - klasa `Throwable` direktno nasleđuje klasu `Object`
  - klasa `Throwable` sadrži polje koje može da se koristi za opis izuzetka
- Izuzeci su provereni, osim izuzetaka klasa `RuntimeExcepton` i `Error`
- Korisnički provereni izuzeci treba da se izvode iz klase `Exception`
  - klasa `Exception` je izvedena iz `Throwable`
- Paradigma za obradu izuzetaka: `try-catch-finally` sekvenca:
  - prvo pokušaj (`try`) da uradiš nešto
  - ako to nešto baci (`throw`) izuzetak, uhvati ga (`catch`) i obradi
  - konačno (`finally`), obavi završne aktivnosti, dogodio se izuzetak ili ne
- Neuhvaćeni izuzeci se obrađuju podrazumevanim rukovaocem za obradu
  - podrazumevani rukovalac (*default handler*) prijavljuje grešku i prekida nit kontrole u kojoj se greška javila

Paketi

- Konflikti imena su čest problem u razvoju softvera
- Klase kapsuliraju polja i metode,
  pa se problem konflikta imena prenosi na imena klasa
- Java uvodi pojam paketa
  - donekle sličan pojmu prostora imena u C++
- Paket sadrži skup tipova i potpaketa
- Paket se imenuje navođenjem deklaracije na početku fajla

```
package rs.ac.bg.etf.igre;
class Karte { ... }
```

- Puno ime klase: `rs.ac.bg.etf.igre.Karte`
- Mogu da se uvoze (`import`) pojedini ili svi tipovi iz nekog paketa:

```
import rs.ac.bg.etf.igre.*
```

- Uvezena imena tipova mogu da se koriste u kratkom obliku: `Karte`

Niti

- Podrška za konkurentno izvršavanje programskih niti (*threads*)
  - mogu da se pišu aplikacije sa više niti programske kontrole
  - niti simultano (uporedno) napreduju sa izvršavanjem
- Aktivni objekti – vlastita nit kontrole definisana metodom `run()`
- Dva pristupa za definisanje aktivnih objekata:
  - proširiti klasu `Thread`
  - implementirati interfejs `Runnable`
- Sinhronizacija za konkurentni pristup podacima objekta ili klase
  - sinhronizovani metodi – upravljaju bravom za pristup objektu
  - suština – međusobno isključivanje pristupa aktivnih objekata
  - opasnost – uzajamno blokiranje (eng. *deadlock*)
- Mehanizam za komunikaciju između niti (`wait-notify`)
  - ugrađen u klasu `Object`

Grafički korisnički interfejs

- Paket `java.awt` (*Abstract Windowing Toolkit*)
- Prozori, dijalozi, meniji,...
- Rad sa komponentama (kontrole, *widgets*)
  - jednostavne i kontejnerske komponente
- Obrada događaja
  - delegirani model: izvori i osluškivači događaja
  - adapteri osluškivača
  - paradigma programiranja: programiranje vođeno događajima *(event-driven programming)*
- Crtanje