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, 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 biti 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 obuhvatati više linija `*/`
  - `/**` dokumentacioni komentar
    - namenjen opisu deklaracije koja 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`
- *foreach* petlja (Java 5.0) za iteriranje kroz zbirke (kolekcije) i nizove
  - bez korišćenja iteratora i indeksa
- Instrukcije skoka
  - `break` i `continue` koje mogu imati i labelu naredbe iz koje se iskače
  - `goto` instrukcija ne postoji
- Labele služe samo za iskakanje iz petlji pomoću `break` i `continue`

Primer *foreach* petlje

- Konvencionalna `for` petlja:

```
public int sumaNiza(int niz[]){
     int suma=0;
     for (int i=0; i<niz.length; 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 pristupiti članu
  - ostala prava pristupa umanjuju pristupačnost člana

Stvaranje i uništavanje objekata

- Objekti se stvaraju korišćenjem ključne reči `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 se ukloniti
  - sakupljač đubreta je posebna programska nit (radi u pozadini)

Metodi

- Metodi pristupaju implementacionim detaljima klase (objekta) koji su, po pravilu, sakriveni od drugih klasa
- 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 se mogu imenovati direktno (bez reference na objekat), kao i u C++
- Objekat čiji se metod poziva se naziva primalac poruke (*receiver*)
- 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
- Objekti tipa niske se jednostavno stvaraju i inicijalizuju:

```
String ime="Petar"
```

- Objekti tipa niske se mogu samo čitati
  - na primer: `ime += " Petrović"`
    formira novi objekat niske `"Petar Petrović"`
- Postoje `StringBuffer` i `StringBuilder` za promenljive niske
- U svakoj klasi se može napisati metod `toString()`
  - konvertuje dati objekat u nisku na željeni način

Nizovi

- Niz je objekat koji predstavlja seriju objekata nekog tipa
- Nizovski objekti imaju 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 kreiranju niza objekata svi elementi niza (reference) su inicijalizovani na `null`
- Mogu se odmah kreirati i objekti na koje pokazuju reference u nizu:

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

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 mu se proslediti argument tipa izvedene klase
- Polimorfizam
  - objekat na koji upućuje referenca može ispoljavati 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 se može proširiti

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 bitan njihov potpis
- Primer: metodi sa istom deklaracijom koji se mogu primeniti 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 čistog ugovora (u praksi – samo do verzije Java 8)
  - deklariše koji metodi su podržani klasom koja će implementirati taj interfejs
- Interfejs je tip
  - mogu se definisati 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
- Klase izuzetaka se izvode iz klase `Throwable`
  - klasa `Throwable` direktno nasleđuje klasu `Object`
  - klasa `Throwable` sadrži polje tipa niske koje se može koristiti za opis izuzetka
- Svi novi izuzeci treba da se izvode iz klase `Exception`
  - klasa `Exception` je izvedena iz `Throwable`
  - koren hijerarhije klasa proverenih izuzetaka
- Paradigma za obradu izuzetaka: `try-catch-finally` sekvenca:
  - prvo pokušaj (`try`) da uradiš nešto
  - ako to nešto baci (`throw`) izuzetak, uhvati ga (`catch`)
  - konačno (`finally`), obavi završne aktivnosti, dogodio se izuzetak ili ne
- Neuhvaćeni izuzeci se obrađuju podrazumevanom rutinom za obradu
  - podrazumevana rutina (*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 se uvoziti (`import`) pojedini ili svi tipovi iz nekog paketa:

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

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

Niti

- Podrška za konkurentno izvršavanje programskih niti (*threads*)
  - mogu se kreirati aplikacije sa više niti programske kontrole
- 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
- 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