Objektno orijentisano programiranje 2

Izuzeci

Uvod

- Za vreme izvršenja aplikacije javljaju se greške različitih nivoa ozbiljnosti
- Kada se metod nekog objekta pozove on može:
  - da otkrije probleme internog stanja objekta
    - nekonzistentne vrednosti polja
  - da otkrije greške na resursima kojima manipuliše
    - problem u fajlu ili na mrežnoj adresi
  - da otkrije da njegov poziv narušava neki protokol
    - čitanje iz zatvorenog fajla
- Dolazi se u priliku da treba praviti kompromis između:
  - korektnosti (proverom na sve moguće greške) i
  - čistote (nezamagljivanjem osnovnog toka kontrole proverom grešaka)
- Izuzeci pružaju elegantan način da se provere greške bez zamagljivanja koda osnovne obrade
- To je mehanizam koji greške signalizira direktno
  - bez flegova i/ili bočnih efekata

Ugovor i implementacija

- Izuzeci koje metod može da signalizira su deo ugovora tog metoda
  - zato je potrebno navesti njihove tipove u zaglavlju metoda
- Listu navedenih izuzetaka koje metod može da signalizira može da:
  - vidi programer
  - proveri prevodilac
- Izuzetak se:
  - baca kada se otkrije izuzetna situacija
  - hvata i obrađuje u kontekstu u kojem se mogao dogoditi (npr. u kontekstu iz kojeg je pozvan metod koji je bacio izuzetak)
  - prosleđuje pozivajućem bloku ako se ne obradi

Tipovi izuzetaka

- Izuzeci u Javi su isključivo objekti (ne mogu biti primitivni tipovi)
- Svi tipovi izuzetaka moraju da budu izvedeni iz `Throwable` ili potklasa
  - klasa `Throwable` sadrži nisku koja se koristi da opiše izuzetak
- Izuzeci u Javi su primarno provereni izuzeci (*checked exceptions*):
  - moraju se navoditi u `throws` klauzuli svakog metoda koji ih baca
  - prevodilac proverava da li metod baca samo one izuzetke koje je deklarisao
- Klasa `Exception` se izvodi iz `Throwable` i opisuje proverene izuzetke
- Svi izuzeci koje programer kreira treba da se izvode iz klase `Exception`
  - tako će biti u kategoriji proverenih izuzetaka
- Problem sa proverenim izuzecima:
  - ako se modifikuje neki metod daleko u lancu poziva da baci novi tip izuzetka
  - a obrada izuzetka je moguća tek negde pri početku lanca
  - moraju se modifikovati svi metodi u lancu dodavanjem izuzetka u `throws`
- Neprovereni izuzeci:
  - standardni izuzeci izvedeni iz klasa `RuntimeException` i `Error`

Razlozi za novi tip izuzetka

- Razlozi za kreiranje novog tipa izuzetka su:
  - dodavanje informacija o podacima koji su doveli do pojave greške (uz nisku opisa)
  - specifični tip izuzetka može biti specifično uhvaćen i obrađen (izuzeci se hvataju prema njihovom tipu)
- Primer
  - reprezentuje uslov "pokušaj smeštanja elementa u pun bafer"

```
class BaferPun extends Exception {
        BaferPun(String s){super(s);}
                        // s sadrži nisku koja opisuje objekat
                        // čije smeštanje nije uspelo;
                        // može se dohvatiti sa getMessage()
}
```

Naredba `throw`

- Izuzeci se bacaju naredbom `throw`
- Nije operator kao u jeziku C++
- Parametar naredbe je referenca na objekat izuzetka
- Primer izuzetaka u klasi `Stek`:

```
public void stavi(Object o) throws BaferPun{
 if (sp>=velicina) throw new BaferPun(o.toString());
 niz[sp++]=o;
}
```

Klauzula `throws`

- Programeri treba da znaju za izuzetke koje metod može da baca
  - isto kao što treba da znaju tip rezultata pri očekivanom ponašanju
- U klauzuli `throws`:
  - lista zarezima odvojenih tipova izuzetaka
- Legalno je baciti i izuzetke koji su izvedeni iz izuzetaka navedenih u `throws` klauzuli
  - razlog:
    klasa se može koristiti polimorfno gde god se njena natklasa očekuje
- Bacanje nedeklarisanog tipa izuzetka je pogrešno:
  - bilo direktno koristeći `throw` ili indirektno pozivajući drugi metod
  - otkriva prevodilac i javlja grešku

Inicijalizacija i izuzeci

- Konstruktor sme da baca izuzetak
  - pod uslovom da je izuzetak naveden u `throws` klauzuli
- Inicijalizator i nestatički inicijalizacioni blok sme da baca izuzetak
  - samo ako svi konstruktori klase deklarišu da ga bacaju
- Statički blok ne sme da baci izuzetak
  - nema ko da uhvati izuzetak
- Rešenje za “inicijalizaciju” polja metodom koji baca izuzetak:
  - konstruktor ili inicijalizacioni blok, unutar njih se može obraditi izuzetak
- Rešenje za “inicijalizaciju” statičkih polja:
  - statički blok – unutar njega se može obraditi izuzetak

Obrada izuzetaka

- Java je striktna u forsiranju proverenih izuzetaka
- Ako se pozove metod koji navodi izuzetak u njegovoj `throws` klauzuli – 3 mogućnosti:
  - uhvatiti i obraditi izuzetak
  - uhvatiti izuzetak i preslikati ga u neki drugi tip izuzetaka
    - bacanjem izuzetka tog drugog tipa
  - deklarisati izuzetak u vlastitoj `throws` klauzuli i proslediti izuzetak dalje (na pozivajući nivo)
    - implicitno (automatski), ako ne postoji odgovarajuća `catch` grana
    - eksplicitno, pomoću `throw` naredbe u nekoj `catch` grani

`try`, `catch` i `finally` (1)

- Izuzeci se dešavaju u `try` blokovima
- Hvataju se u `catch` blokovima
- U svakom slučaju se izvršava `finally` blok

```
try
     blok
catch (TipIzuzetka1 identifikator)
     blokObradeIzuzetka1
catch (TipIzuzetka2 identifikator)
     blokObradeIzuzetka2
  ...
finally
     zavrsniBlok
```

`try`, `catch` i `finally` (2)

- Telo `try` naredbe se izvršava
  - dok se ne baci izuzetak
  - ili dok se blok ne završi uspešno
- Ako se u `try` bloku ne baci izuzetak:
  - blok se uspešno završi
- Ako se baci izuzetak:
  - rukovaoci izuzecima (`catch` grane) se ispituju po redu
  - traži se odgovarajuća grana sa parametrom tipa
    - klase bačenog izuzetka
    - natklase bačenog izuzetka
- Java 7 dopušta i obradu više tipova izuzetka u jednoj `catch` grani
  - navode se u listi parametara `catch` grane

`try`, `catch` i `finally` (3)

- Ako se u `try` bloku desi izuzetak
  - kontrola se nepovratno prenosi u neki `catch` blok (ako postoji odgovarajući)
- Ako se ne pronađe odgovarajući `catch`
  - izuzetak se propagira spoljašnjem nivou `try`
  - ako se u metodu ne pronađe odgovarajući `catch`
    - izuzetak se propagira pozivajućem metodu
- Kada se `catch` blok završi
  - kontrola se prenosi iza `try` bloka (u `finally`, ako postoji)
- Primer

```
public static void main(String [] arg){
  Stek s= new Stek(10);
  try{for (int i = 0; i<11; i++)s.stavi(new Integer(i)); }
  catch (BaferPun e){System.out.println(e); }
}
```

Obrada supertipova izuzetaka

- Nije dozvoljeno da se rukovalac (hendler) nadtipa izuzetka nalazi ispred rukovaoca podtipa izuzetka
- Primer

```
class SuperIzuzetak extends Exception{}
class SubIzuzetak extends SuperIzuzetak{}
class LoseHvatanje{
  public void DobarPokusaj(){
    try { throw new SubIzuzetak();
    } catch (SuperIzuzetak i) {//...
    } catch (SubIzuzetak i) {/* ! Greska */}
  }
}
```

`finally`

- Klauzula `finally` se izvršava u svakom slučaju
  - bez obzira na to da li je izuzetak bačen ili ne
- Bez `finally` klauzule,
  - blok iza `try-catch` sekvence se ne mora izvršiti ako se baci izuzetak iz `catch` bloka
- Svrha `finally`:
  - sređivanje internog stanja i/ili
  - oslobađanje neobjektnih resursa (kao što su otvorene datoteke)

Primer finally

- Primer:

```
public boolean traziRec(String imeRecnika, String rec)
                                        throws GreskaRecnika{
  Recnik recnik;
  try {
    recnik = new Recnik(imeRecnika);
    while (!recnik.kraj())
      if (rec.equals(recnik.sledecaRec())) return true;
    return false;
  } finally { if (recnik != null) recnik.zatvori(); }
}
```