

Objektno orijentisano programiranje 2

Generici

Uvod

- Namena kao i u C++:
 - uopštavanje apstrakcija - parametrizovani tipovi i postupci
- Često korišćenje kod zbirki (Javina biblioteka)
- Sličnost sa C++ genericima – samo namena
- Implementacija u Javi je potpuno različita:
 - nezavisno prevođenje
 - samo jedan tip bez obzira na broj konkretizacija generika
 - veličina međukoda ne raste u zavisnosti od broja konkretizacija
 - parametrizacija primitivnim tipom nije moguća
- Generici: tipovi (klase i interfejsi), metodi i konstruktori

Motivacija

- Do Java 1.5:
 - tip `Object` korišćen kao opšti tip
- Problem kod dohvatanja elemenata iz heterogene zbirke
 - zbirka sadrži objekte koji pripadaju jednoj hijerarhiji tipova
 - rezultat dohvatanja iz zbirke se mora konvertovati naniže
 - nebezbedna konverzija
 - ako nije dohvaćen objekat očekivanog tipa – `BadCastException`
- Java 1.5 uvodi generike koji rešavaju problem
 - nebezbedna konverzija naniže više nije potrebna

Primer problema

- Pre Jave 1.5:

- klasa Stek koristi reference tipa Object
 - kako bi omogućila rad sa objektima proizvoljnog tipa
 - ako je u programu potreban stek diskova

```
Stek stek = new Stek();
```

```
...
```

```
Disk disk = new Disk()
```

```
String niska = "Disk";
```

```
stek.stavi(disk); stek.stavi(niska);
```

```
...
```

```
Disk disk = (Disk) stek.uzmi();
```

- problem je u nebezbednoj konverziji

Rešenje problema

- Od Jave 1.5:

- generička klasa Stek, zavisi od tipa elemenata T

```
class Stek <T> {  
    public void stavi(T t) {...};  
    public T uzmi () {...};  
}
```

- kreiranje objekta steka diskova:

```
Stek<Disk> stekDiskova = new Stek<Disk>();
```

Rešenje (2)

- Stavljanje na stek diskova:

```
Disk disk1 = new Disk();  
stekDiskova.stavi(disk1);
```

- prevodilac dodaje proveru tipa pri stavljaju na stek

- Uzimanje sa steka diskova:

```
Disk disk2 = stekDiskova.uzmi();  
  
– nepotrebna nebezbedna konverzija naniže  
– rezultat dohvatanja je sigurno tipa Disk
```

Priroda generika

- Kod generika se prevodi nezavisno
- Samo jedna klasa, bez obzira na broj poziva generika
- Konkretizovane klase: Stek<Disk> i Stek<String>, u stvari su ista klasa Stek
- Eksperiment koji potvrđuje navedenu činjenicu:

```
Stek<Disk> stekDiskova= new Stek<Disk>();  
Stek<String> stekNiski= new Stek<String>();  
boolean isto = (stekDiskova.getClass() ==  
                stekNiski.getClass());
```

– rezultat: isto==true

Provera i “brisanje” tipa

- Prevodilac ubacuje proveru tipova
 - neće dozvoliti da se na Stek stavi nešto što nije Disk
- Formalni parametar generičkog tipa,
u celom telu se zamjenjuje tipom Object
- Prevodilac ubacuje i konverziju tipa Object u ciljni tip
 - iako je u pitanju konverzija naniže, sada je bezbedna
- Nakon toga – brisanje informacija o tipu (*type erasure*)
- U bytecode – jedinstveni tip za sve pozive generika
 - dati tip se naziva “sirovim” tipom (*raw type*)

Sirovi tip

- Za klasu `class Stek <T> { ... }` sirovi tip je `Stek`
- Dozvoljeno bi bilo pisati:

```
Stek<String> stekNiski = new Stek<String>();  
Stek stek = stekNiski;
```
- Razlog za sirove tipove
 - kompatibilnost sa zatečenim bibliotečkim klasama (pre JDK 5.0), kao što su klase zbirki (*Collections*), koje su bile ne-generičke
 - kada se koriste sirovi tipovi generičkih tipova, staro ponašanje: dohvatanje sa steka: objekat `Stek` vraća objekat tipa `Object`

Definisanje generičke klase

- Generička klasa
 - može biti parametrizovana jednim ili pomoću više tipova koji se pojavljuju kao formalni argumenti (parametri) generika
`class ime <T1, T2, ..., Tn> { ... }`
- Parametri T_i se mogu pojaviti:
 - u deklaraciji polja
 - kao povratni tip metoda
 - kao tip parametra metoda ili konstruktora
 - u deklaraciji lokalne promenljive
 - u deklaraciji ugnezđenog tipa
- Doseg parametra:
 - do kraja klase (odosno generičkog tipa, metoda ili konstruktora)

Ograničenja u definicijama

- Formalni parametar generika se ne može koristiti kao:
 - tip statičkog polja generičke klase
 - u statičkim metodima za:
 - povratni tip, tip parametra metoda, tip lokalne promenljive
 - u statičkim inicijalizacionim blokovima date klase
 - razlog: postoji samo jedna klasa (sirovi tip)
- Statičkom članu klase se ne može pristupiti preko parametrizovanog imena tipa
 - na primer, ako je metod `m()` statički metod generičke klase `A<T>`: nije dozvoljeno `A<Integer>.m()`, dozvoljeno je `A.m()`
 - slično je sa literalom klase:
nije dozvoljeno `A<Integer>.class`, dozvoljeno je `A.class`
- `U A<T>` nije dozvoljeno kreirati objekat tipa `T`, niti niz objekata tipa `T`
 - na primer, nije dozvoljeno `new T()` niti `new T[100]`

Korišćenje generika

- Konkretizovani tip ili “poziv” generika:
 - generički tip kojem su formalni parametri tipa zamenjeni stvarnim argumentima tipa
- Primer:
 - za generički tip `class G<T> { ... }`,
`G<String>` je poziv za konkretan tip `String`
- Pozivi generika se koriste u definicijama objekata
 - primer: `G<String> gS = new G<String>();`
- Argumenti generika mogu biti
 - klase, interfejsi i nizovi (čak i sa elementima primitivnog tipa),
- Argumenti generika ne mogu biti
 - primitivni tipovi

Zaključivanje o tipu

- Problem:
 - u definiciji objekta – dva puta se navode argumenti generika
 - što više parametara generika, to veći problem
- Java 7 rešava problem “dijamant simbolom”
 - u pozivu konstruktora se mogu izostaviti argumenti tipa
 - primer: `G<String> gS = new G<>();`
- Prevodilac automatski zaključuje o tipovima argumenata
 - na osnovu deklaracije tipa reference sa leve strane =

Generici i nizovi

- Argument generika može biti niz
 - referenci na objekte
 - podataka primitivnog tipa
- Niz objekata konk. generičke klase se ne može kreirati:
 - `G<Arg> [] gNiz = new G<Arg> [n]; // ! GRESKA`
- Može:
 - `G<Arg> [] gNiz = (G<Arg> []) new Object [n];`
 - nije bezbedno, jer niz može sadržati objekte različitog tipa

Generici, izvođenje i ugnezđenje

- Ako je D podtip B , generički tip $G<D>$ nije podtip G
- Generički tip može biti izведен iz
 - negeneričkog tipa ili
 - konkretizacije generičkog tipa
 - konkretizacija generičke osnovne klase može imati kao argument neki parametar generika, na primer:
`class GD<T1, T2> extends GB<T1> { ... }`
- Generička klasa ne može biti potklasa `Throwable`
- Kod ugnezđenih tipova
 - i spoljašnji i ugnezđeni tip mogu da budu generički ili ne

Ograničenje parametra

- Parametar generika se može ograničiti sa gornje strane:
 - class G <T extends B & I1 & I2 & ...> {...}
- Argument koji menja T mora biti tipa koji je:
 - podtip ili tip svih navedenih tipova (klase B, interfejsa I1, I2,)
 - klasa B se može pojaviti samo ako je generik G klasa
 - u listi tipova se može naći i ranije navedeni parametar generika
- U protivnom ograničenje sa gornje strane je Object
 - class G <T extends Object> {...}
 - je isto što i class G <T> {...}

Razlog za džokere

- Problem:
 - $G<IT>$ nije podtip tipa $G<T>$, gde je IT tip izведен iz tipa T
- Različito od nizova:
 - niz $IT[]$ je podtip tipa $T[]$
- Primer: za generički stek $Stek<T>$
 - $Stek<String>$ nije jedna vrsta $Stek<Object>$
 - $String[]$ jeste jedna vrsta $Object[]$
- U praksi ovo predstavlja prilično ograničenje
- Način da se ono prevaziđe – džokeri

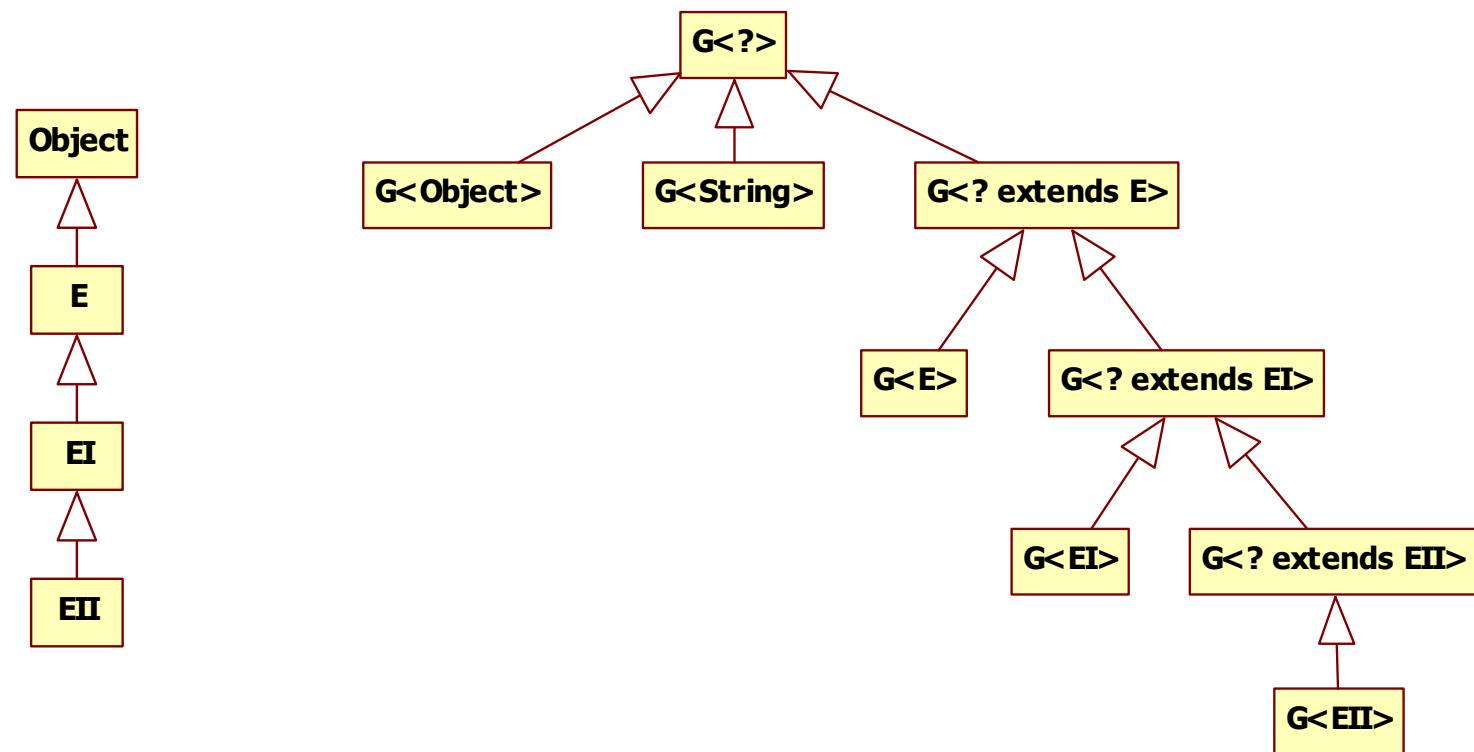
Neograničeni džoker

- Džoker '?' na mestu stvarnog argumenta generika
 - parametrizovani tip sa kojim će biti kompatibilni kao podtipovi svi parametrizovani tipovi sa prozvoljnim tipom argumenta generika
 - primer: ako je formalni parametar nekog metoda tipa `Stek<?>`, metodu se mogu proslediti stvarni argumenti tipa `Stek<String>` ili `Stek<Integer>` ili `Stek<Object>`
- Ponekad je neograničeni džoker previše opšti
 - postoje slučajevi kada se želi ograničiti kompatibilnost parametrizovanog tipa sa njegovim podtipovima
 - postoje dve vrste ograničenja tipa:
ograničenje gornje i ograničenje donje granice tipa

Ograničenje gornje granice tipa

- Ograničenje gornje granice tipa se postiže pozivom:
 $G<? extends E>$
 - E – najviši tip u hijerarhiji tipova sa kojim postoji kompatibilnost
- Primer: $Stek<? extends Disk>$
 - kompatibilan kao nadtip sa stekovima elemenata tipa $Disk$ i podtipova $Disk$ npr. $Stek<Disk>$, $Stek(HD)$
 - nije kompatibilan sa stekovima elemenata drugih tipova, npr. $Stek<String>$ ili $Stek<Integer>$
- Tip $G<?>$ je ekvivalentan sa $G<? extends Object>$

Hijerarhija tipova



Ograničenje donje granice tipa

- Ograničenje donje granice tipa se postiže pozivom:
 $G<? super E>$
 - E – najniži tip u hijerarhiji tipova sa kojim postoji kompatibilnost
- Primer: $Stek<? super Disk>$
 - kompatibilan kao nadtip sa stekovima elemenata tipa Disk i tipa Object
 - nije kompatibilan sa stekovima elemenata podtipova Disk, niti sa stekovima drugih tipova, npr. $Stek<String>$
- Tip $G<? super Object>$
 - kompatibilan kao nadtip samo sa $G<Object>$ pa nema svrhe

Ograničenja ograničenih džokera

- Nije moguće istovremeno ograničiti džoker donjom granicom i gornjom granicom tipa
 - primer: `G<? extends EG super ED>` nije ispravno
- Ne može se granica (gornja ili donja) sastojati od više tipova
 - primer: `G<? extends EG1 & EG2>` nije ispravno čak ni ako su `EG1` neka klasa, a `EG2` neki interfejs

Nizovi objekata generičke klase

- Problem: pri kreiranju niza objekata generičke klase se ne može navesti argument za konkretizaciju g. klase
 - jedno rešenje (ranije navedeno):
$$G<\text{Arg}>[] \ gNiz = (G<\text{Arg}>[]) \ \text{new Object}[n];$$
 - loše: elementi $gNiz$ mogu biti proizvoljnog klasnog tipa
 - drugo rešenje:
$$G<?>[] \ gNiz = \text{new } G<?>[n];$$
 - loše: i ovde elementi $gNiz$ mogu biti proizvoljnog klasnog tipa
 - najbolje rešenje:
$$G<\text{Arg}>[] \ gNiz = (G<\text{Arg}>[]) \ \text{new } G<?>[n];$$
 - preko $gNiz[i]$ se u niz mogu upisati samo objekti tipa $G<\text{Arg}>$

Generički metodi

- Nestatički i statički metodi mogu biti generički
- Definicija generičkog metoda:
`<param_gen> tip_rez ime_met (param_met) { ... }`
- Pozivanje konkretizacije generičkog metoda:
`ref.<arg_gen>ime_met (arg_met);`
`this.<arg_gen>ime_met (arg_met); // obavezno this`
`Klasa.<arg_gen>ime_met (arg_met); // za staticke`
- Argumenti generičkog metoda mogu da se određuju automatski
 - poziv izgleda kao poziv negeneričkog metoda
 - argumenti generika se određuju prema tipu argumenata metoda, odnosno prema tipu konteksta u koji se vraća rezultat metoda