Identifikacija tipova tokom izvršavanja programa

Sadržaj

Određivanje tipova tokom izvršavanja

- Dva tipa:
  - **RTTI (run time type identification):**
    - poznate su sve klase kod prevođenja i izvršavanja.
  - **Refleksija:**
    - ne mogu se odrediti tipovi tokom prevođenja programa.
- Kada ih je potrebno koristiti ?
  - kada se iz objekta žele pročitati metode, konstruktori i atributi.
  - RMI – distribuirani objekti (Java viši kursevi).

Primer za RTTI

- Ako u nizu (`niz`) imamo niz tipa `Object`:
  - Ne može se pozvati `niz[i].prikazi();`
  - Jer `Object` klasa nema metodu `prikazi();`
  - Ali može: `((String) niz[i]).prikazi();`
- Ovo omogućava baš RTTI !
- A šta ako `niz[0]` nije tipa String ?
- Ako se ovo pretvaranje ne može izvršiti onda će se desiti `ClassCastException`
- Može se pitati:
  - `if (niz[i] instanceof String) ...`
- Pa onda ga pretvoriti.

RTTI

- Vidi se da je RTTI upravo suprotan od polimorfizma.
- RTTI koristiti samo kada je nužno.
- Polimorfizam koristiti za uopšten i elegantan kôd.

Kako radi RTTI ?

- Prvo, način kreiranja objekata u Javi:
- Klasa `java.lang.Class` učita u memoriju objekt za svaki tip (klasu) koja se koristi.
- I baš taj objekat (tipa `Class`) kreira objekte te klase.
- To se krije iza operatora `new`.
- Ali on se može i zaobići!
- Kada se kreira objekat tipa `Class`?
- Odgovor: baš onda kada je potrebno!

Kako radi RTTI ?

- Delovi programa koji se ne koriste, ne učitavaju se pre nego što su potrebni.
- Ako se definiše samo: `VektorStek s;`
- Tada se ne kreira objekat, niti se učitava sadržaj iz .class datoteke.
- Međutim, kada se uradi: `s = new VektorStek();`
- Kod izvršavanja ove linije prvo se učita u memoriju klasa `Object.class` (nalazi se u rt.jar)
- Probati sa: **java -verbose:class StekTest > a.txt**
- Pokazuje koju su sve sistemske klase učitane.

Detalji oko klase `java.lang.Class`

- Reprezentuje klase u Javi.
- Čak su i primitivni tipovi (int, double, char, ...) reprezentovani `Class` objektima.
- Ima privatni konstruktor, jer su objekti kreirani od strane JVM-a, koja poziva metodu `defineClass` apstraktne klase `java.lang.ClassLoader`.
- JVM može imati više podklasa ove klase koji predstavljaju **class loadere**, koje JVM koristi pri učitavanju klasa.
- Class loadere ćemo raditi na višim kursevima.

Neke metode klase `java.lang.Class`

- `public static Class forName(String className)`
  - vraća `Class` objekat za naziv klase ili baca izuzetak
  - poziv ove metode inicijalizuje objekat

```
Class a = Class.forName( args[0] );
```

  - I pozove se sa: **java Test VektorStek**
  - Ovaj kôd se ne izvršava kada se kreira objekt tipa `VektorStek`, već kada program kreira u memoriji objekat koji reprezentuje sam tip.
  - To je objekat klase `Class`.
  - To pokazuje da se kreira objekat klase `Class` za VektorStek.class kada se pozove metoda `forName()`.

Neke metode klase `java.lang.Class`

- `public ClassLoader getClassLoader()`
  - vraća referencu na `ClassLoader` koji se koristi.
  - sistemske klase kao `Object`, `System`, ... koriste podrazumevani, tj. "bootstrap" `ClassLoader`.
  - on je reprezentovan sa null.
- `public Object newInstance()`
  - kreira objekat za tip koji predstavlja.
- `public Class getSuperclass()`
  - vraća nadklasu.
  - ako je primitivni tip, polje ili `Object` vraća se null.

Neke metode klase `java.lang.Class`

- `public String getName()`
  - vraća naziv `Class` objekta.
- `public Class getClass()`
  - vraća `Class` objekat bilo kojeg objekta u programu.
  - svaka klasa nasleđuje `Object`, pa može da se napiše

```
Test test = new Test();
test.getClass().getName();
Class.forName("java.lang.Object").getName();
```


Tri načina dolaska do `Class` objekta

1. Pomoću statičke metode `forName` klase `Class`:
   ```
   Class a = Class.forName("Test");
   ```
2. Pomoću metode `getClass` nasleđene iz klase `Object`:
   ```
   Class a = ts.getClass();
   ```
3. Ako je `T` bilo koji tip u Javi, onda je `T.class` njegov `Class` objekat:
   ```
   Class c1 = Test.class;
   Class c2 = int.class;
   Class c3 = Double[].class;
   ```
   - Ovde se vidi `Class` ne reprezentuje klasu već tip podatka.
   - Metapodatak!

Refleksije

- Ako pokušamo:

```
Class cl = Class.forName( args[0] );
```

- Želimo kreirati objekat klase `args[0]` koristeći podrazumevani konstruktor, ali ne koristeći operator `new` jer to ne možemo u ovoj situaciji.
- TIP NIJE POZNAT TOKOM PREVOĐENJA PROGRAMA!

```
Object obj = cl.newInstance();
System.out.println( obj );
```


Refleksije

- Ako klasa B ima privatni podrazumevani konstruktor:

c:\>java Test A
Objekat klase A.

c:\>java Test B
Klasa ili podrazumevani konstruktor nije dostupan: Class Test can not access a member of class B with modifiers "private"

c:\> java Test java.lang.Class
Klasa ili podrazumevani konstruktor nije dostupan: Can not call newInstance() on the Class for java.lang.Class

- Ovo nije do kraja zadovoljavajuće jer mi želimo kreirati objekte za dati tip (možda nepoznat kod prevođenja).
- Ne samo za podrazumevani konstruktor, nego pristupati i njegovim metodama i atributima.

Refleksije

- To omogućava mehanizam refleksije.
- U paketu `java.lang.reflect` postoje ove tri klase:
  - **java.lang.reflect.Field** - repezentuje atribute članice,
  - **java.lang.reflect.Method** - reprezentuje metode članice,
  - **java.lang.reflect.Constructor** - reprezentuje konstruktore.
- Kad imamo objekat klase `Class`, onda se objekti tipa `Field`, `Method` i `Constructor` dobijaju pomoću sledeće tri metode klase:
  - **getFields()** - daje atribut javnih atributa članica;
  - **getMethods()** - daje atribut javnih metoda članica;
  - **getConstructors()** - daje atribut javnih konstruktora.

Precizniji opis metoda klase `Class`

- **public** Constructor**[] getConstructors()** - vraća atribut javnih konstruktora;
  - to je atribut objekata klase **java.lang.reflect.Constructor** koja ima ove interesantne metode:
    - **public** String **toString()** - vraća deklaraciju konstruktora kao string.
    - **public** Object **newInstance(**Object**[] arg) -** kreira objekat koristeći ovaj poseban konstruktor, argument je ili null ili polje koje reprezentuje formalne argumente, s tim da se primitivni tipovi "obmotaju" u odgovarajuće objekte.
- **public** Method**[] getMethods() -** vraća atribut javnih metoda;
  - to je atribut objekata klase **java.lang.reflect.Method**;
  - neke metode:
    - **public** String **toString()** - vraća deklaraciju metode kao string.
    - **public** Object **invoke(**Object **obj,** Object**[] args**) - izvršava specificiranu metodu kao metodu objekta `obj` sa listom argumenata `args`, argument je ili null ili atribut koje reprezentuje formalne argumente.

Precizniji opis metoda klase Class

- **public** Field**[] getFields()** - vraća atribut javnih varijabli članica;
- to je atribut objekata klase **java.lang.reflect.Field;**
- neke metode:
  - **public** String **toString()** - vraća deklaraciju varijable članice kao string.
  - sadrži niz `set` i `get` metoda koje omogućavaju da se zadaju ili vraćaju vrednosti pojedinih varijabli članica.

Refleksije - primer

```
import java.lang.reflect.*;

public class C { ...
```

- pa u `main` metodi:
- ne zna se naziv klase koji se učitava sa komandne linije

```
Class cl = Class.forName( args[0] );

// ispis svih konstruktora
System.out.println("Konstruktori: ");
Constructor[] con = cl.getConstructors();

for(int i=0; i < con.length; i++)
System.out.println(i + ". " + con[i].toString() );
```

- *Rezultat:*

Konstruktori:
0. public C(int)
1. public C()

Refleksije – primer

```
// ispis svih metoda

System.out.println("Metode:");
Method[] met = cl.getMethods();
for(int i=0; i < met.length; i++)
System.out.println(i + ". " + met[i].toString() );
```

- *Rezultat:*

Metode:
0. public int C.getX()
1. public void C.setX(int)
2. public native int java.lang.Object.hashCode()
3. public final native java.lang.Class java.lang.Object.getClass()
4. public final void java.lang.Object.wait(long,int) throws java.lang.InterruptedException
5. ...

Refleksije – primer

```
// ispis svih atributa

System.out.println("Atributi:");
Field[] fil = cl.getFields();
for(int i=0; i < fil.length; i++)
System.out.println(i + ". " + fil[i].toString() );
```

- *Rezultat:*

Atributi:
0. public float C.y

Refleksije – primer

• Kreiramo objekat te klase koristeći konstruktor potpisa (int);
• Najpre potražimo konstruktor sa traženim potpisom koristeći metodu:
`public Constructor getConstructor(Class[] param)`
klase `Class`.

```
Class[] cll= { int.class }; // uočite klasu povezanu sa primitivnim tipom podatka!

Constructor cons = cl.getConstructor(cll);
```

• Kreiramo objekat za vrednost integera = 3. Tu koristimo `newInstance` metodu klase `Constructor`.

```
Object[] atr = { new Integer(3) }; // moramo koristiti obmotavajuće klase

Object obj = cons.newInstance(atr); // obj = instanca klase učitane na kom. liniji
```


Refleksije – primer

- Sada se poziva metoda "getX()" ako takva postoji na objektu `obj` koristeći metodu klase `Class`:

```
public Method getMethod(String name, Class[] param)
```

- Metoda `getX()` nema argumenata pa je drugi argument null:

```
Method met = cl.getMethod("getX", null);
```

- Izvršimo tu metodu na objektu **obj** pozivajući metodu:
`public Object invoke(Object obj, Object[] args),`

gde je,

`obj` = objekat na kojem će metoda biti pozvana, a
`args` = polje argumenata metode

Refleksije – primer

```
System.out.println("Izvrsavamo metodu getX(): ");

System.out.println( met.invoke(obj, null) );
```

- *Rezultat:*

Izvrsavamo metodu getX():
3

Zadatak

- Napišite program koji za datu klasu (naziv) učitanu sa komandne linije ispisuje sve javne konstruktore, metode i atribute.
- Onda korisnik treba da izabere konstruktor preko kojeg konstruiše objekat tako da učitava njegov redni broj sa komandne linije i redom defisane formalne argumente.
- Slično s metodama. Najpre rešite jednostavniji problem u kojem sve metode i konstruktori imaju kao formalne argumente primitivne tipove podataka ili ništa.
- Onda uopštite tako da formalni argumenti mogu biti i klase odnosno interfejsi.

Pitanja i odgovori

Pitanja?

Nadam se da imam odgovore…

Izvor

- Sun-ov sajt! http://java.sun.com, API, http://java.sun.com/docs/books/tutorial/reflect/.
- Računarski praktikum 2, Dr sc. Mladen Jurak, Prirodoslovno-matematički fakultet, Matematički odjel, Sveučilište u Zagrebu, http://web.math.hr/nastava/rp2/.
- The Java Programming Language, 4th Edition, Ken Arnold, James Gosling, David Holmes, Addison Wesley Professional, 2005.
- Java: The Complete Reference, J2SE 5 Edition, Herbert Schildt, McGraw-Hill Osborne Media, 6th Edition, 2004.
- Google!