Predavanje 5.

OBJEKTNO-ORIJENTISANO PROGRAMIRANJE

2024/25.

Agenda

Paketi i standardne Javine biblioteke

PAKETI

- **Paket je kolekcija srodnih tipova (klasa i interfejsa)**

  srodnost je funkcionalnog karaktera (npr. `java.sql` sadrži API za pristup i procesiranje podataka smeštenih u nekom izvoru, tipa relacione baze, upotrebom Java programskog jezika)

- Razlozi za korišćenje paketa:
  - naznačavanje srodnosti određenih tipova
  - olakšavanje pronalaženja željenog tipa (fokusiranjem na samo jedan paket)
  - otklanjanje potencijalnih duplikata u nazivima (jedinstveni prostor imena)
  - kontrolisanje pristupa (klase u okviru istog paketa mogu da imaju neograničen pristup jedna drugoj, a spoljne ne)
- Svaka klasa pisana u Javi je deo nekog paketa.

  U primerima koje smo do sada radili nismo navodili kom paketu pripadaju klase, jer su pripadale podrazumevano uključenom paketu **`java.lang`**, npr. **`String`** i **`Math`** imaju puna imena **`java.lang.String`** i java.lang.Math.

Deklaracija pripadnosti paketu

- Navođenje imena paketa kome klasa pripada

```
package geometry;
public class Sphere {
// Details of the class definition
}
```

- `package` mora biti prva naredba u fajlu (ne računajući prazne linije i komentare).
- U fajlu može postojati samo jedna deklaracija paketa.
- Jedan tip može pripadati samo jednom paketu.
- Unutar jednog paketa ime tipa je jedinstveno, npr. u paketu geometry pomenutog u primeru, može postojati samo jedna klasa sa imenom Sphere.

Deklaracija pripadnosti paketu

- Ime paketa može biti složeno, npr.
  - `geometry.shapes3D`
  - sadržaj ovog paketa ne mora da ima veze sa sadržajem paketa
    - `geometry`
  - koji se nezavisno definiše i njegovo postojanje ne uslovljava postojanje paketa `geometry.shapes3D`.

Upotreba paketa

- Dva načina upotreba tipova definisanih u nekom paketu:

  Navođenjem punog imena tipa: `<ime paketa>.<ime tipa>`

```
public class Ball {
        geometry.Sphere  b = new geometry.Sphere();
        ...
}
```

  Izvršiti uvoz tipa ili svih tipova paketa

```
import  geometry.Sphere; // ili   import geometry.*;
public class Ball {
        Sphere  b = new Sphere();
        ...
}
```

- Paket java.lang se implicitno uvozi.

Konflikti imena tipova

- Mehanizam paketa i uvoženja daje kontrolu nad potencijalnim konfliktima imena tipova. Do konflikta imena tipova dolazi u slučaju da je u nekom trenutnku u istoj klasi potrebno koristiti dva tipa koji imaju ista imena i pripadaju različitim paketima.
- Ako su u dva tipa pod istim imenom deklarisana u različitim paketima,
- npr.

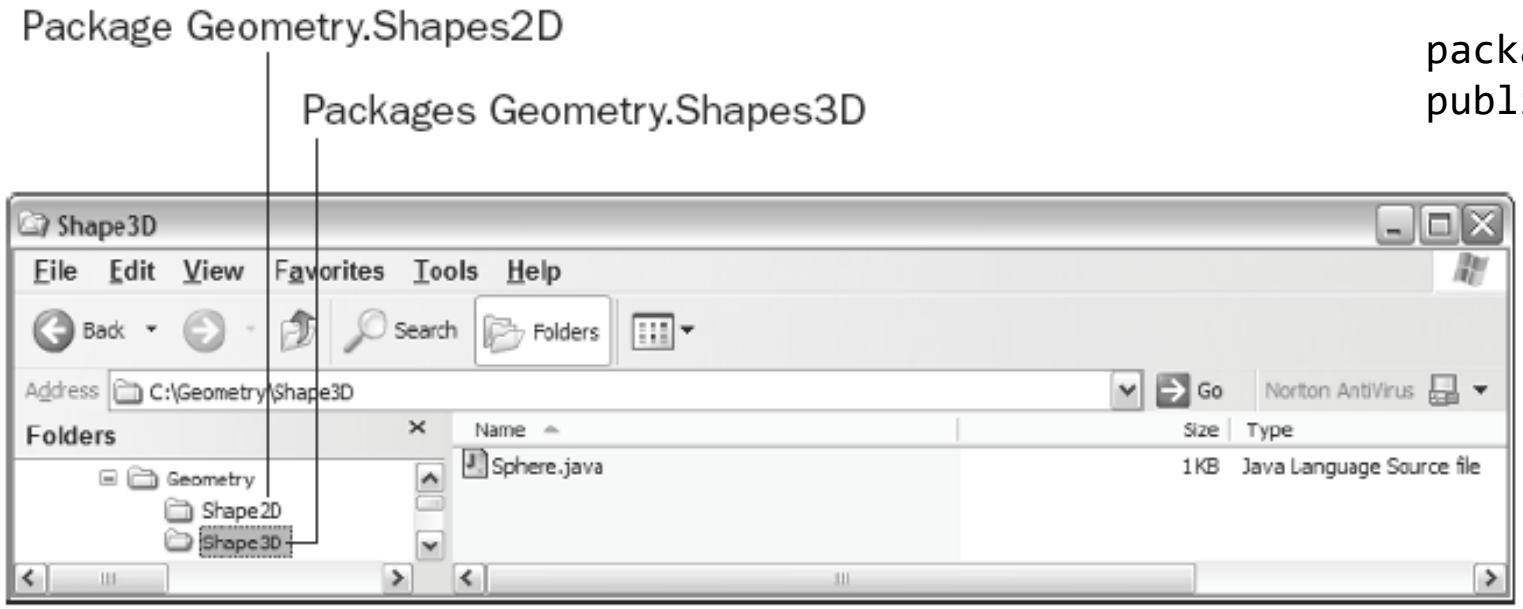
```
package geometry.shapes3D;
   public class Sphere {    // class definition }

package seometry.shapes2D;
   public class Sphere {    // class definition }
```

- pri njihovoj upotrebi u istom tipu potrebno je naglasiti razliku, ali svakako konflikt imena je prebačen na nivo imena paketa.

Struktura direktorijuma

Struktura direktorijuma u koju su ‘članovi’ paketa smešteni mora da isprati strukturu naziva paketa.




```
package Geometry.Shapes3D;
public class Sphere {...}
```

```
package Geometry.Shapes2D;
public class Sphere {...}
```

`javac` u potrazi za tipovima

- Kompajler i interpreter intenzivno koriste informacije o razmeštaju datoteka (preuzeto sa slajdova prof. dr V. Devedžića)
  - kompajler mora da zna gde da nađe import-ovane klase
  - interpreter mora da zna gde da nađe neku klasu i njene metode

- Kompajleru su potrebne informacije o svakom tipu koji se koristi.
  - Ovde se podrazumevaju i tipovi koji se ne pominju eksplicitno, ali se nalaze u hijerarhiji tipa koji se koristi (npr. Predstavljaju indirektnu nadklasu klase koju deinišemo).

`javac` u potrazi za tipovima

- U trenutku kada naiđe na naziv tipa čiju definiciju nema u tekućem fajlu, kompajler traži **izvorni ili bajt kod** (dovoljno mu je da jednbo pronađe) u kome je definisan tip i to prvo trži u:
  - Tekućem direktorijumu, pa u
  - lib direktorijumu Java Runtime Enviroment-a, a zatim u
  - u tzv. **user class path**-u, tj. korisnički definisanim putanjama do korisničkih klasa; korisnički class path može biti pročitan na dva načina:
    - u **CLASSPATH** envoronment varijabli
    - direktno iz opcija pri kompajliranju

```
javac -classpath “C:\moji paketi” Line.java
```

`javac` u potrazi za tipovima

- Pri potrazi za definicijom tipa javac može pronaći:
  - class fajl, ali ne i izvorni (java) fajl: tada kompajler direktno koristi bajkod koji je pronašao
  - izvorni, ali ne i class fajl: tada kompajler kompajlira pronađeni izvorni fajl i koristi tako dobijeni bajt kod
  - nalazi i izvorni i class fajl: tada kompajler prvo utvrđuje da li je class fajl out of date (zastareo). Ako je class fajl stariji od izvornog koda, tada izvorni kod biva kompajliran i class fajl zamenjen novim. U suprotnom, kompajler koristi postojeći bajt kod.

`java` u potrazi za tipovima

- Pokretanje pri upotrebi korisnički defnisanih paketa je takođe drugačije:

```
java –classpath “.;C:\mojipaketi” TryPackage
```

- Česta greška:

```
java –classpath “C:\mojipaketi” TryPackage
Exception in thread "main" java.lang.NoClassDefFoundError: TryPackage
Caused by: java.lang.ClassNotFoundException: TryPackage
…
```

`jar` fajlovi

`.jar` – java archive

- Jar arhive sadrže kompresovane class fajlove sa kompletno sačuvanom direktorijumskom strukturom i obezbeđuju jednostavnost u prenosu i upotrebi većeg broja korisnički definisanih paketa
- Kreiranje jedne .jar arhive:

```
C:\Beg Java Stuff>jar cvf Geometry.jar Geometry\*.class
```

- `jar [options] [manifest] destination input-file [input-files]`

  `options`
  - `c` - Creates a new or empty archive on the standard output.
  - `t` - Lists the table of contents from standard output.
  - `x` file - Extracts all files, or just the named files, from standard input.
  - `f` - The second argument specifies a jar file to process.
  - `v` - Generates verbose output on stderr.
  - `u` - update an existing JAR file by adding files. For example,
    - `jar uf foo.jar foo.class`