PAKETI I STANDARDNE JAVINE BIBLIOTEKE

2016/17

PRIRODNO-MATEMATIČKI FAKULTET, UNIVERZITET U KRAGUJEVCU

PAKETI

- **Paket je kolekcija srodnih tiopova (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.
- 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.Navođenje imena paketa kome klasa pripada

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.

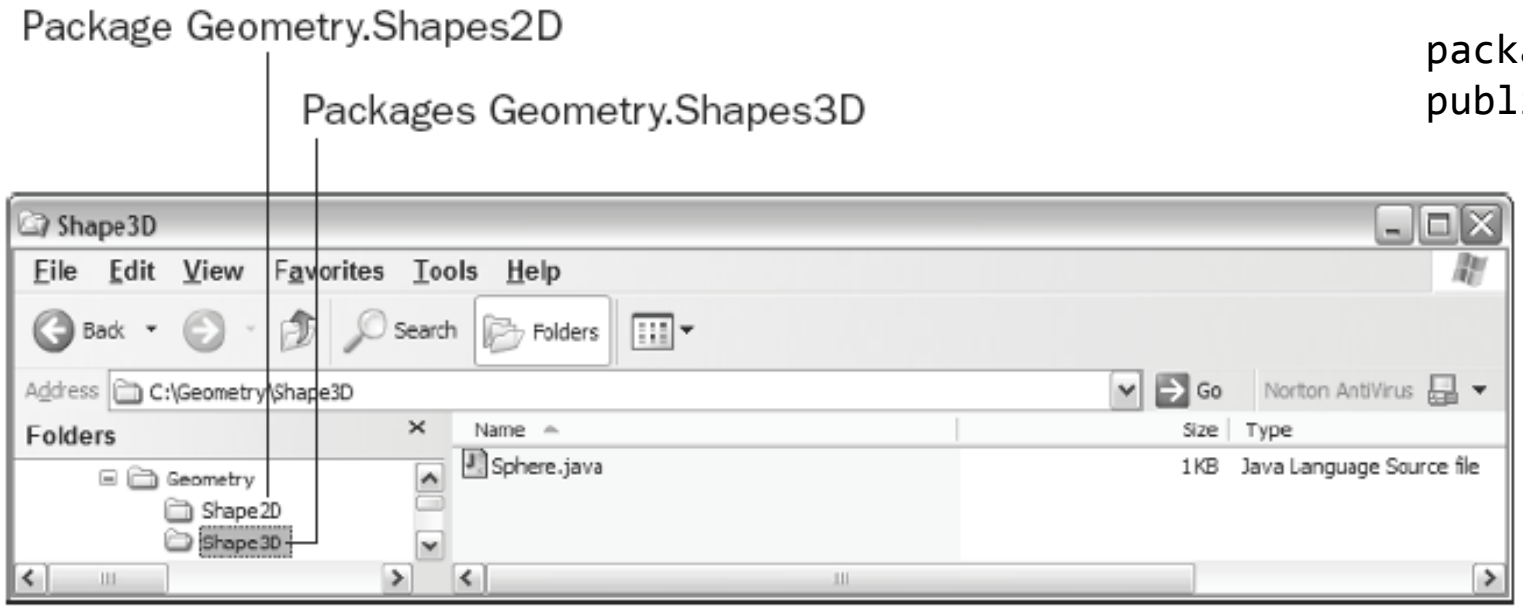
```
package geometry.shapes3D;
   public class Sphere {    // class definition }

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 direktorijuma.




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

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

POTRAGA ZA DEFINICIJOM TIPA

- 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).
- 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
```

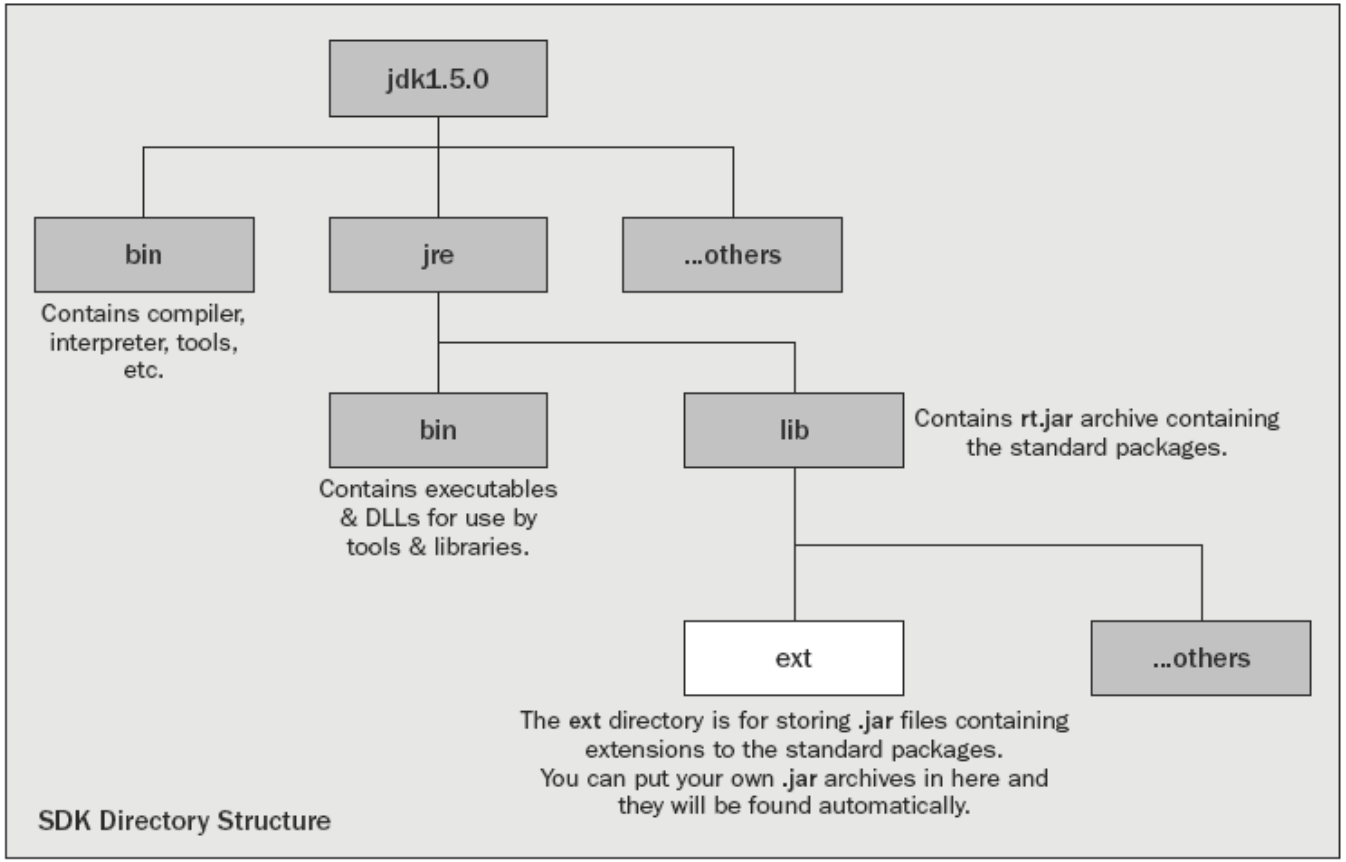
- `jar [options] [manifest] destination input-file [input-files]`

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`

jar FAJLOVI

- Ekstenzije su jar fajlovi smešteni u ext direktorijum kreiran pri instalaciji JRE-a.
- Ovaj direktorijum je standardno uvršten u listu direktorijuma za pretragu pri pronalaženju class fajlova od strane javinih komajlera i interpretera.



MODIFIKATORI VIDLJIVOSTI

2016/17

PRIRODNO-MATEMATIČKI FAKULTET, UNIVERZITET U KRAGUJEVCU

UČAURIVANJE ENKAPSULACIJA

- Učaurivanje predstavlja mehanizam sakrivanja informacija.
- Zašto?

  Class creators vs. client programmer

  zaštita podataka i metoda

- Učaurivanje se izvodi upotrebom modifikatora vidljivosti/pristupa
- Modifikatorima pristupa (vidljivosti) se određuje opseg dostupnosti elemenata koda (polja, metoda, tipova). Svaki element ima definisanu vidljivost, implicitno ili eksplicitno definisanu.
- Modifikatori pristupa se mogu primeniti na:
  - Tipove
  - Elemente tipova – polja i metodi

  ne mogu na varijble definisane unutar metoda.

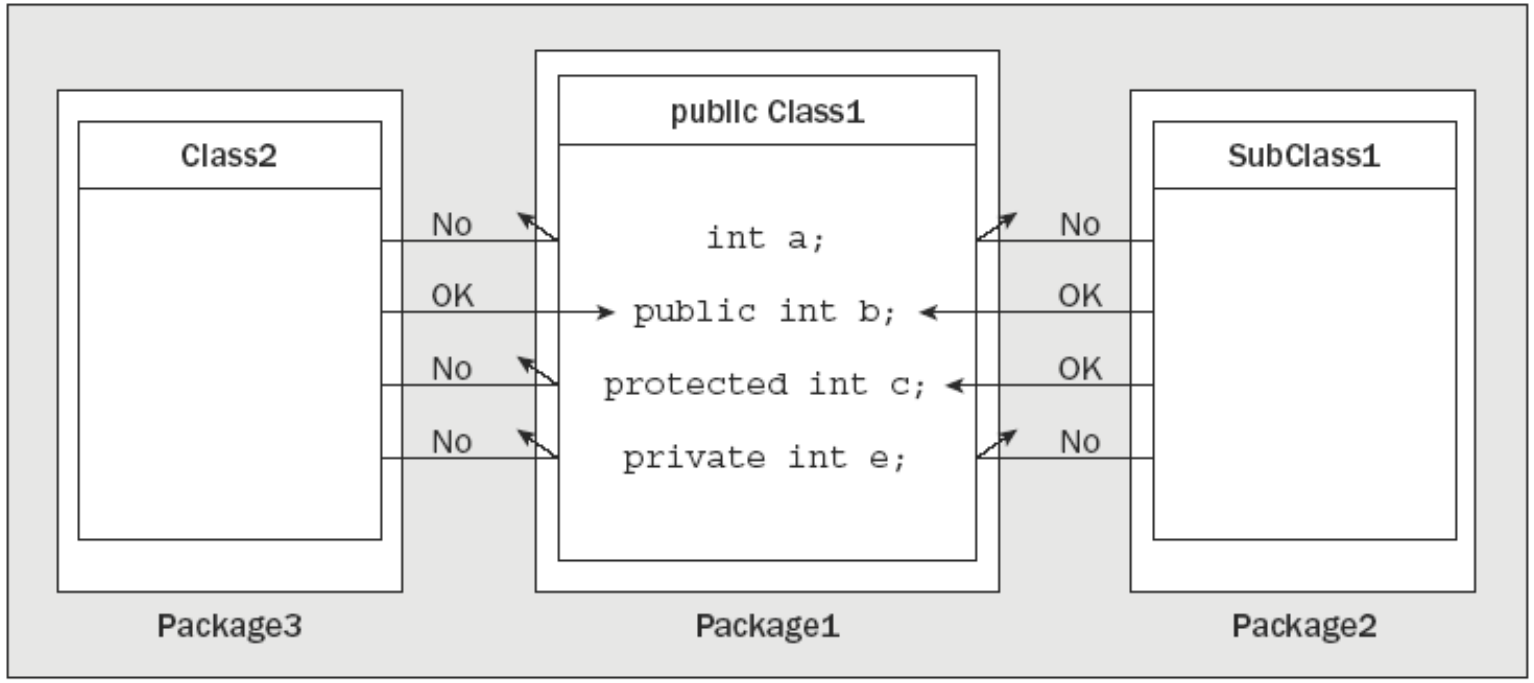
NASLEĐIVANJE - REVIDIRANO

- Objekat podklase uvek sadrži kompletan objekat superklase klase, ali to ne znači da su svi članovi superklase dostupni metodama koje su specifične samo za podbklasu!
- **nasleđivanje**: uključivanje članova bazne klase u izvedenu klasu na način da su dostupni (accessible) u izvedenoj klasi
- nasleđeni član bazne klase je onaj koji je dostupan u izvedenoj klasi

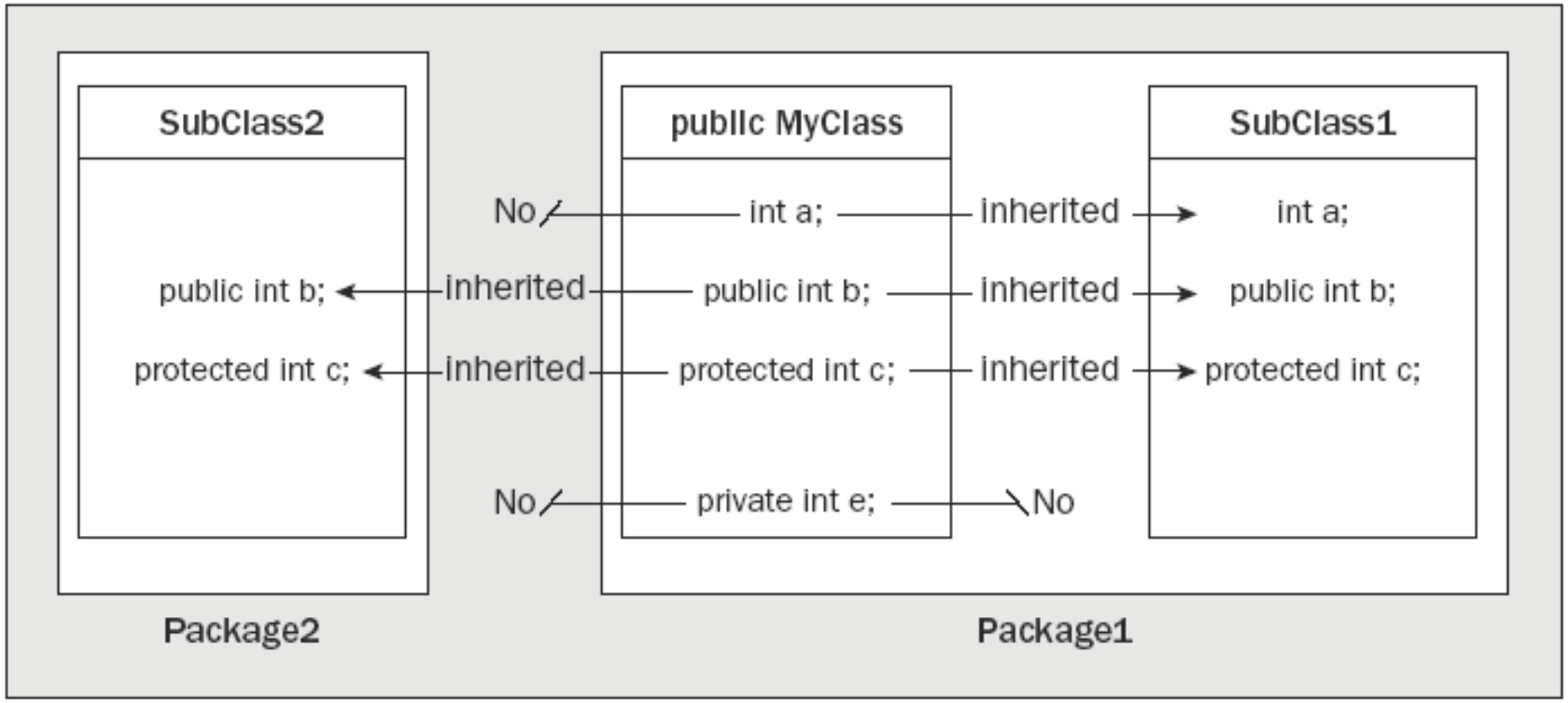
VRSTE MODIFIKATORA VIDLJIVOSTI

Tip \ vidljivost	Dostupno (vidljivo)			
	klasi	podklasi	paketu	bilo kome
Private	X			
protected	X	X	X	
public	X	X*	X	X
Nothig(default) Package Friendly	X		X	

VIDLJIVOST ČLANOVA



DOSTUPNOST NASLEĐENIH ČLANOVA



PRIVATE

```
class Alpha {
    private int iamprivate;
    private void privateMethod() {
        System.out.println("privateMethod");
    }
}
class Beta {
    void accessMethod() {
        Alpha a = new Alpha();
        a.iamprivate = 10;      // illegal
        a.privateMethod();      // illegal
    }
}
Beta.java:9: Variable iamprivate in class Alpha not accessible from class Beta.
        a.iamprivate = 10;     // illegal
          ^
1 error
Beta.java:12: No method matching privateMethod() found in class Alpha.
        a.privateMethod();         // illegal
1 error
```

PRIVATE

```
class Alpha {
    private int iamprivate;
    boolean isEqualTo(Alpha anotherAlpha) {
      if (this.iamprivate==anotherAlpha.iamprivate) //legal
            return true;
        else
            return false;
    }
}
```

PROTECTED

```
package Greek;
public class Alpha {
    protected int iamprotected;
    protected void protectedMethod() {
        System.out.println("protectedMethod");
    }
}
```

```
package Greek;
public class Gamma {
   void accessMethod(Alpha a) {
     a.iamprotected = 10;    // legal
     a.protectedMethod();    // legal
   }
}
```

```
package Latin;
import Greek.*;
public class Delta extends Alpha {
    void accessMethod(Alpha a, Gamma g) {
        iamprotected = 10;    // legal
        a.iamprotected = 10;  // illegal
        protectedMethod();    // legal
        a.protectedMethod();  // illegal
    }
}
```

PUBLIC

```
package Greek;
public class Alpha {
    public int iampublic;
    public void publicMethod() {
        System.out.println("publicMethod");
    }
}
package Roman;
import Greek.*;
class Beta {
    void accessMethod() {
        Alpha a = new Alpha();
        a.iampublic = 10;        // legal
        a.publicMethod();        // legal
    }
}
```

DEFAULT - PACKAGE

```
package Greek;
public class Alpha {
    public int iampublic;
    public void publicMethod() {
        System.out.println("publicMethod");
    }
}
package Roman;
import Greek.*;
class Beta {
    void accessMethod() {
        Alpha a = new Alpha();
        a.iampublic = 10;       // illegal
        a.publicMethod();       // illegal
    }
}
```

```
package Greek;
class Gama {
    void accessMethod() {
        Alpha a = new Alpha();
        a.iampackage = 10;      // legal
        a.packageMethod();      // legal
    }
}
```

ODABIR VIDLJIVOSTI ATRIBUTA BAZNE KLASE

- Metode koje sačinjavaju alat za komunikaciju sa spoljašnjim svetom/klasama se definišu kao `public`.
- Podaci članovi ne treba da budu `public` osim konstanti namenjenih za opštu upotrebu.
- Ako očekujete da će drugi ljudi koristiti vaše klase za izvođenje sopstvenih tada podatke članove definišite kao `private`, ali obezbedite `public` get-ere i set-ere.
- Koristite `protected` kada želite neometan pristup od strane klasa u istom paketu, a za klase iz drugih paketa dozvoljen samo ako su podklase.
- Izostavljanje modifikatora pristupa omogućava vidljivost člana klase u svim klasama paketa, dok je za klase van paketa to isto kao i upotreba private atributa.