Paketi i standardne biblioteke Jave

Paketi

Essentially, a package is a uniquely named collection of classes.

Packages are implicit in the organization of the standard classes as well as your own programs. The primary reason for grouping classes in packages is to avoid possible name clashes with your own classes when you are using prewritten classes in an application.

(preuzeto sa slajdova prof. dr V. Devedžića)

- Paket je kolekcija srodnih klasa i interfejsa, srodnost je funkcionalnog karaktera (npr. `java.lang`)
- Razlozi za korišćenje paketa:
  - naznačavanje da su neke klase i interfejsi srodni
  - olakšavanje pronalaženja željene klase/interfejsa (fokusiranje samo na 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 a gde nismo navodili kom paketu pripadaju klase su pripadale podrazumevanom `default` paketu, a sve Javine gotove klase koje smo koristili, nr. `String` pripadaju `java.lang` paketu koji je porazumevano uključen u svaki fajl.

“Pakovanje” klase

- Navođenje imena paketa kome klasa pripada

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

- package mora da bude prva naredba u fajlu (ne računajući prazne linije i komentare).
- Svi fajlovi koji sadrže definicije klasa koje pripadaju paketu Geometry **moraju biti smešteni u direktorijumu koji nosi ime samog paketa**, dakle, Geometry.

Note the use of the public keyword in the definition of the Sphere class. This makes the class accessible generally. If you omit the public keyword from the class definition, the class would be accessible only from methods in classes that are in the Geometry package.

Note that you would also need to declare the constructors and methods in the class as public if you want them to be accessible from outside of the package.

Paketi i smeštanje `.java` i `.class` fajlova

- Package is intimately related to the directory structure in which it is stored.
- A package can have a composite name (example, `java.lang`) that is a combination of two or more simple names.

For example, you might have developed several collections of classes dealing with geometry, perhaps one that works with 2D shapes and another with 3D shapes. In this case you might include the class Sphere in a package with the statement:

```
package Geometry.Shapes3D;
```

and the class for circles in a package using the statement:

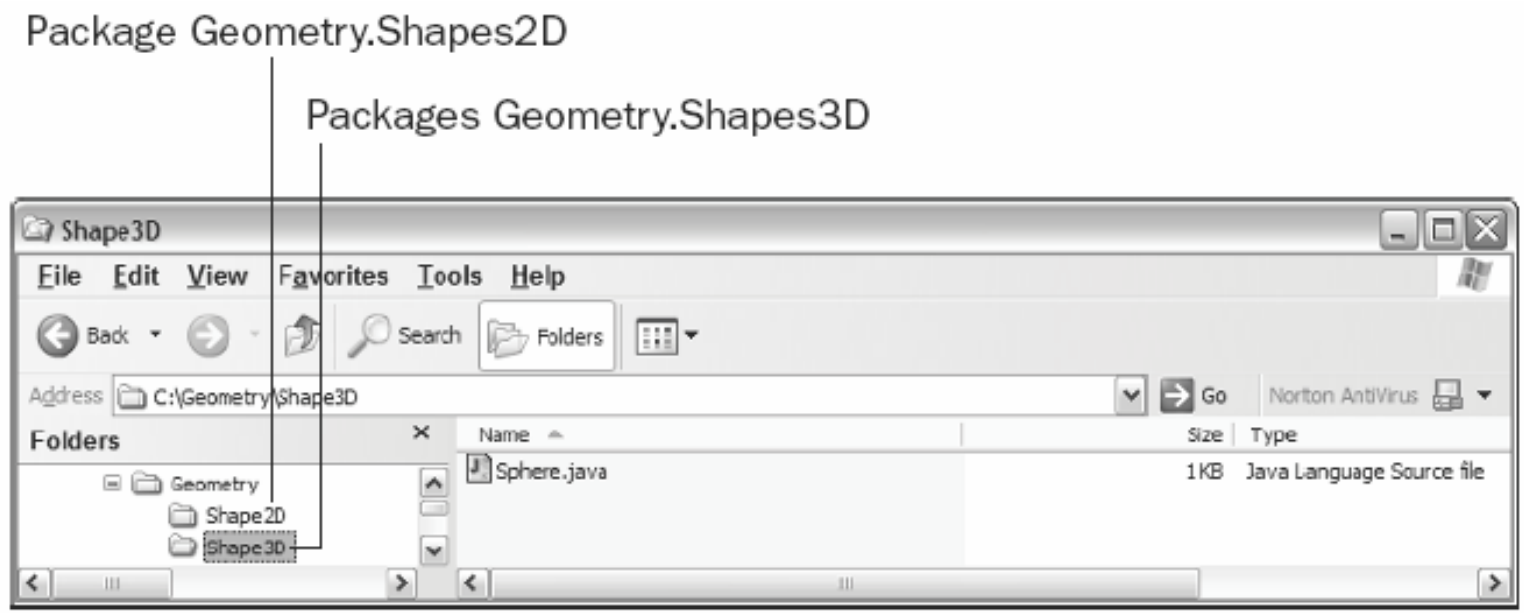
```
package Geometry.Shapes2D;
```

- package name MUST REFLECT THE DIRECTORY STRUCTURE in which the package is stored

Paketi i smeštanje `.java` i `.class` fajlova

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

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



- You can compile the source for a class within a package and have the `.class` file that is generated stored in a different directory, but the directory name must still be the same as the package name.

Kompajliranje

- 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
- The compiler needs type information for every class or interface used, extended, or implemented in the source file. This includes classes and interfaces not explicitly mentioned in the source file but which provide information through inheritance.
- When the compiler needs type information, it looks for a source file or class file which defines the type. The compiler searches first in the bootstrap and extension classes (`lib`), then in the user class path. The user class path is defined by setting the `CLASSPATH` environment variable or by using the `-classpath` command line option.

```
javac -classpath “C:\moji paketi” Line.java
javac -classpath “C:\Beg Java Stuff” *.java
```

Kompajliranje

- A successful type search may produce a class file, a source file, or both. Here is how javac handles each situation:
  - *Search produces a class file but no source file*: `javac` uses the class file.
  - *Search produces a source file but no class file*: `javac` compiles the source file and uses the resulting class file.
  - *Search produces both a source file and a class file*: `javac` determines whether the class file is out of date. If the class file is out of date, `javac` recompiles the source file and uses the updated class file. Otherwise, javac just uses the class file.

  `javac` considers a class file out of date only if it is older than the source file.
- If you use the `-sourcepath` option, the compiler searches the indicated path for source files; otherwise the compiler searches the user class path both for class files and source files.

`http://java.sun.com/j2se/1.3/docs/tooldocs/win32/javac.html#examples`

Kompajliranje

- The easiest way to specify `CLASSPATH` is by using the `-classpath` option when you invoke the compiler. The path to the package directory is the path to the directory that contains the package directory. For example, if you have stored the source files for classes that are in the `Geometry` package in the directory with the path `C:\Beg Java Stuff\ Geometry`, then the path to the Geometry directory is `C:\Beg Java Stuff`, not `C:\Beg Java Stuff\Geometry`, in which case the package will not be found.

Upotreba paketa

- Tri načina:
  - leave the `.class` files for the classes in the package in the directory with the package name and use the `-classpath` option on the command line when you invoke the compiler or the interpreter. This overrides the CLASSPATH environment variable if it happens to be set.

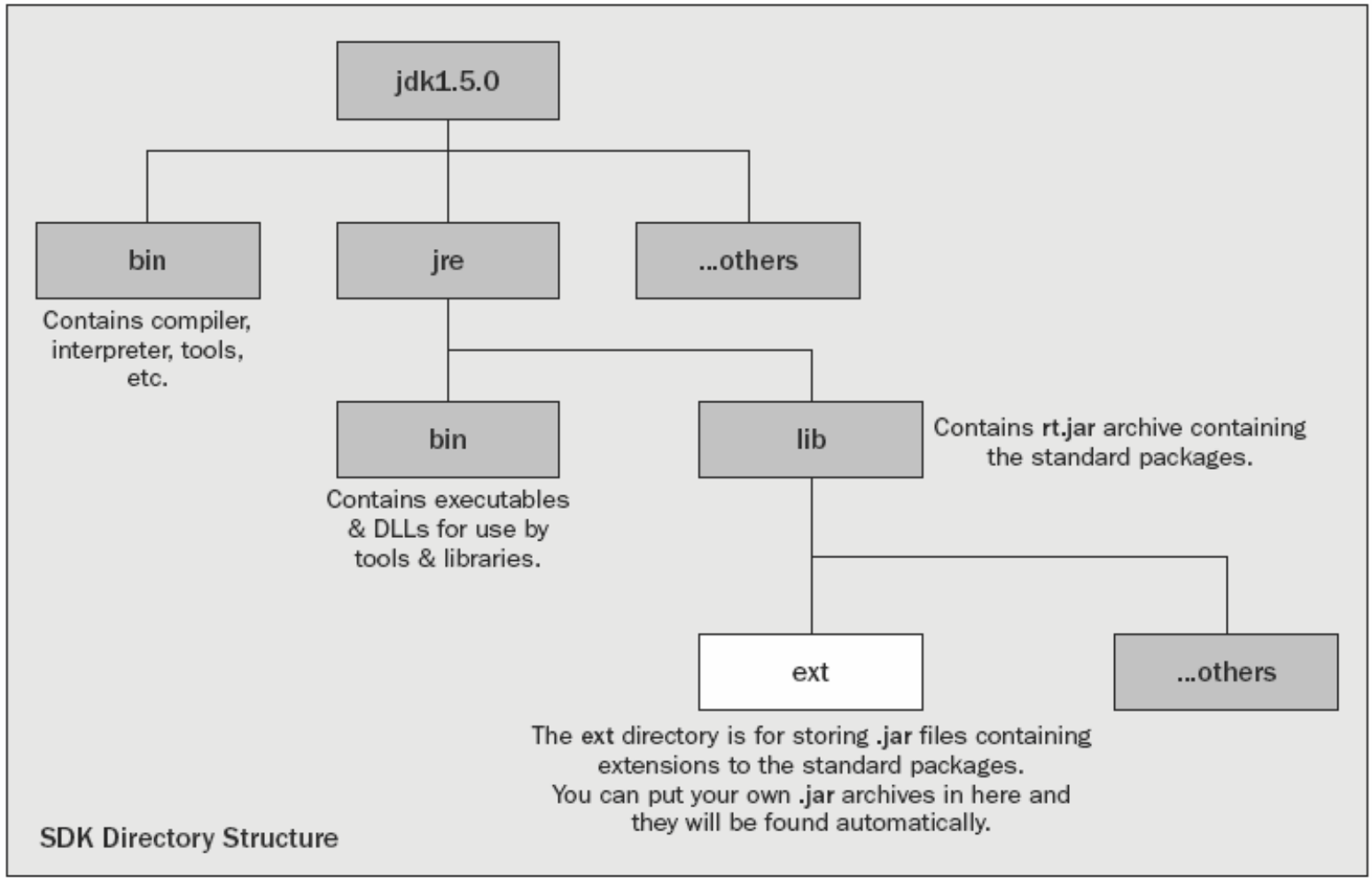
    Note that it is up to you to make sure that the classes in your package are in the right directory.

    ```
    javac -classpath ".;C:\MySource;C:\MyPackages" MyProgram.java
    ```
  - leave the `.class` files for the classes in the package in the directory with the package name and use the `CLASSPATH` environment variable, which needs to contain the paths to your packages.

    `CLASSPATH=.;C:\MySource;C:\MyPackages` (Control Panel -> Advanced tab in the System Properties)
  - Another way to make your packages available once you have compiled them is by making them extensions to the set of standard packages, i.e. make `.jar` file.

.jar fajlovi i ekstenzije

- `.jar` – java archive
- Extensions are `.jar` files stored within the ext directory that is created when



.jar fajlovi i ekstenzije

- The classes and packages in the `.jar` archives that you place in the ext directory will automatically be accessible when you compile or run your Java programs, without the need to set the CLASSPATH environment variable or use the –classpath command-line option.
- When you create a `.jar` file for a package, you need to make sure that you add the `.class` files with the directory structure corresponding to the package name—you can't just add the `.class` files to the archive.
- This will create the archive `Geometry.jar`, and add all the `.class` files that are in the Geometry directory to it.

```
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`

  …

import i import static

- uvoz svih* klasa paketa

```
import Geometry.Shapes3D.*;
```

* Ne zaboraviti na vidljivost tipova (public).

- uvoz jedne klase

```
import Geometry.Shapes3D.Sphere;
```

Note that the * can be used only to select all the classes in a package. You can't use Geometry.* to select all the packages in the Geometry directory.

- Dozvoljeno je, naravno, i ovo

```
Geometry.Shapes3D.Sphere ball=new Geometry.Shapes3D.Sphere(10.0,1.0,1.0,1.0);
```

ali nije pregledno i praktično ako se ova klasa često koristi.

- uvoz static člana

```
import static java.lang.Math.PI;
class Sphere {
double volume() {return 4.0/3.0*PI*radius*radius*radius;}
}
```

- uvoz svih static članova

```
import static java.lang.Math.*;
```

Konvencije davanja imena

- nazivi klasa (`MojaKlasa`)
- nazivi metoda (`mojaMetoda`)
- nazivi atributa (`mojAtribut`)
- nazivi paketa (`mojpaket.drugipaket`)
- set/get metode (`setAtribut/getAtribut`)
- user biblioteke se imenuju obrnutim redosledom domena
  `edu.firma.autor.utility.imebiblioteke`

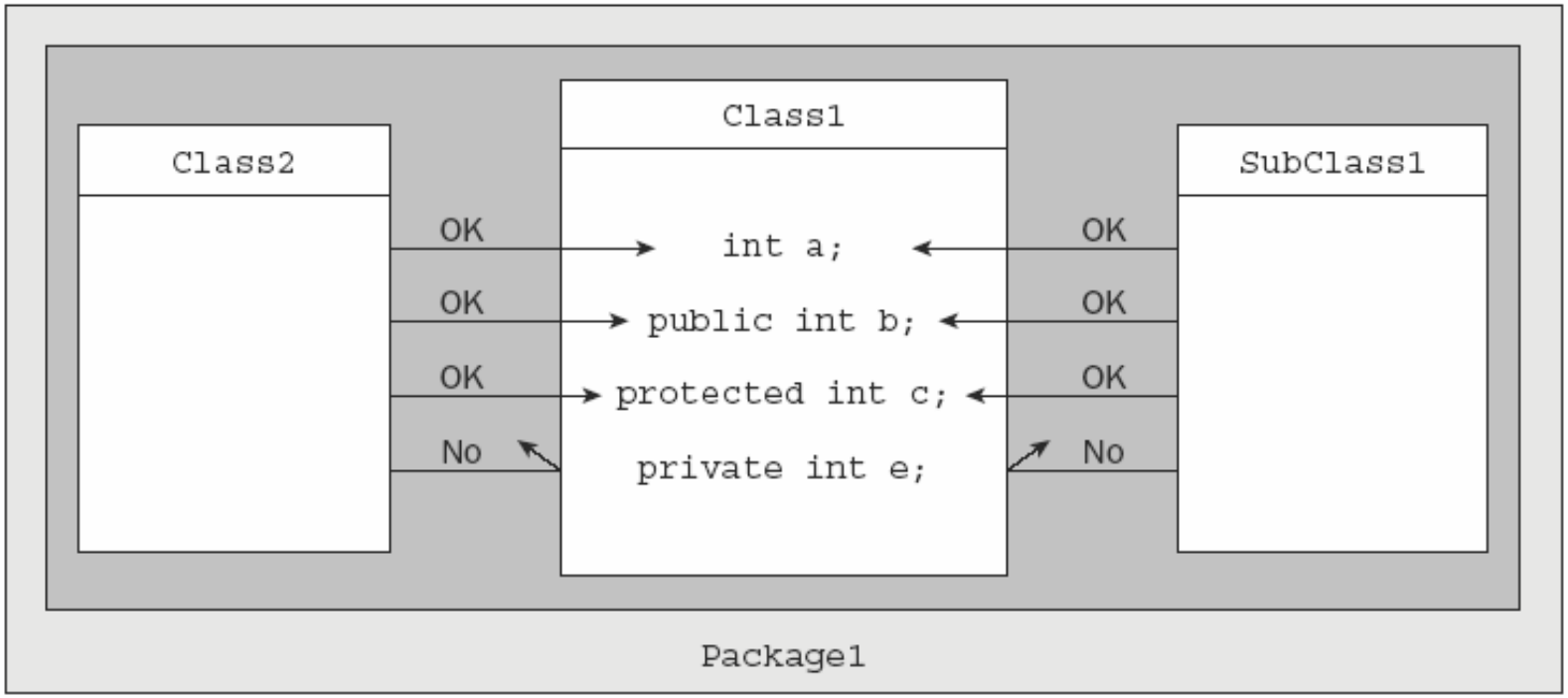
Konvencije davanja imena

- nazivi klasa (`MojaKlasa`)
- nazivi metoda (`mojaMetoda`)
- nazivi atributa (`mojAtribut`)
- nazivi paketa (`mojpaket.drugipaket`)
- set/get metode (`setAtribut/getAtribut`)
- user biblioteke se imenuju obrnutim redosledom domena
  `edu.firma.autor.utility.imebiblioteke`

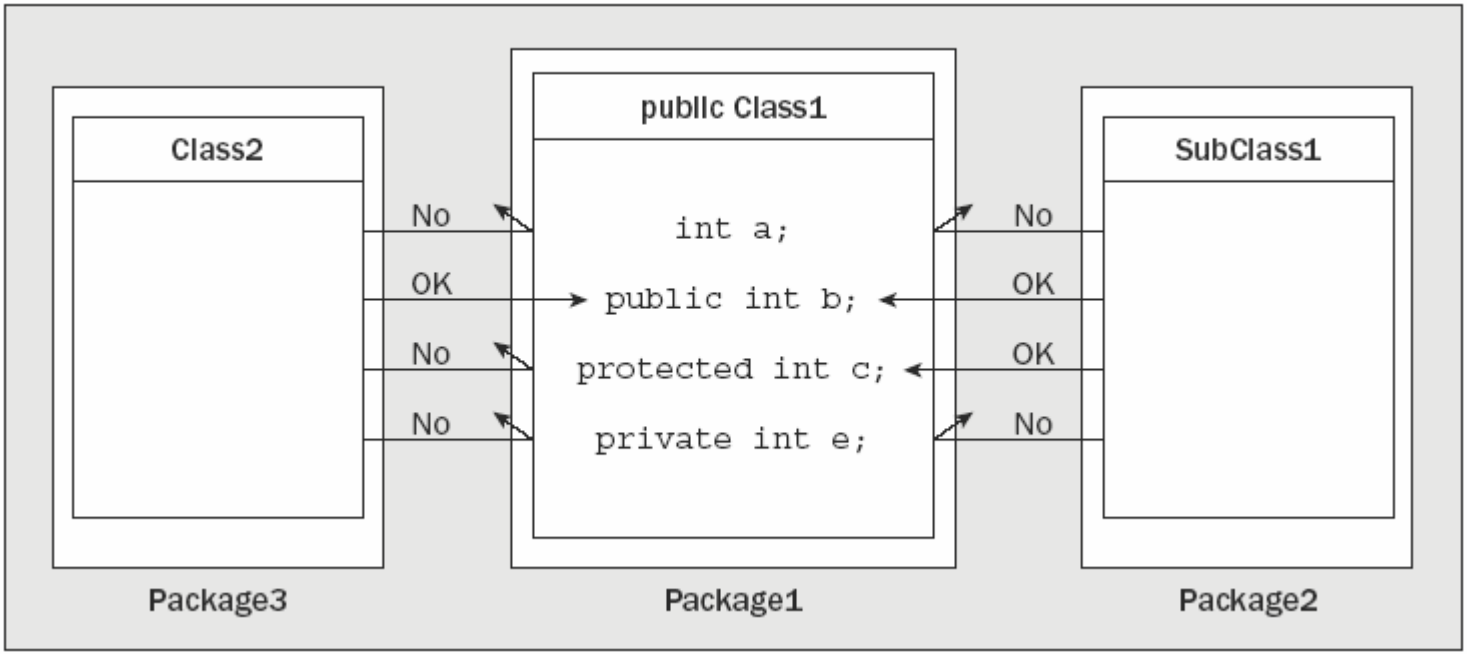
Još malo o access modifikatorima

public	Classes Interfaces Constructors Inner Classes Methods Field variables	A Class or Interface may be accessed from outside its package. Constructors, Inner Classes, Methods and Field variables may be accessed from wherever their class is accessed.
protected	Constructors Inner Classes Methods Field variables	May be accessed by other classes in the same package or from any subclasses of the class in which they are declared.
private	Constructors Inner Classes Methods Field variables	May be accessed only from within the class in which they are declared.
no modifier	Classes Interfaces Constructors Inner Classes Methods Field variables	May only be accessed from within the package in which they are declared.

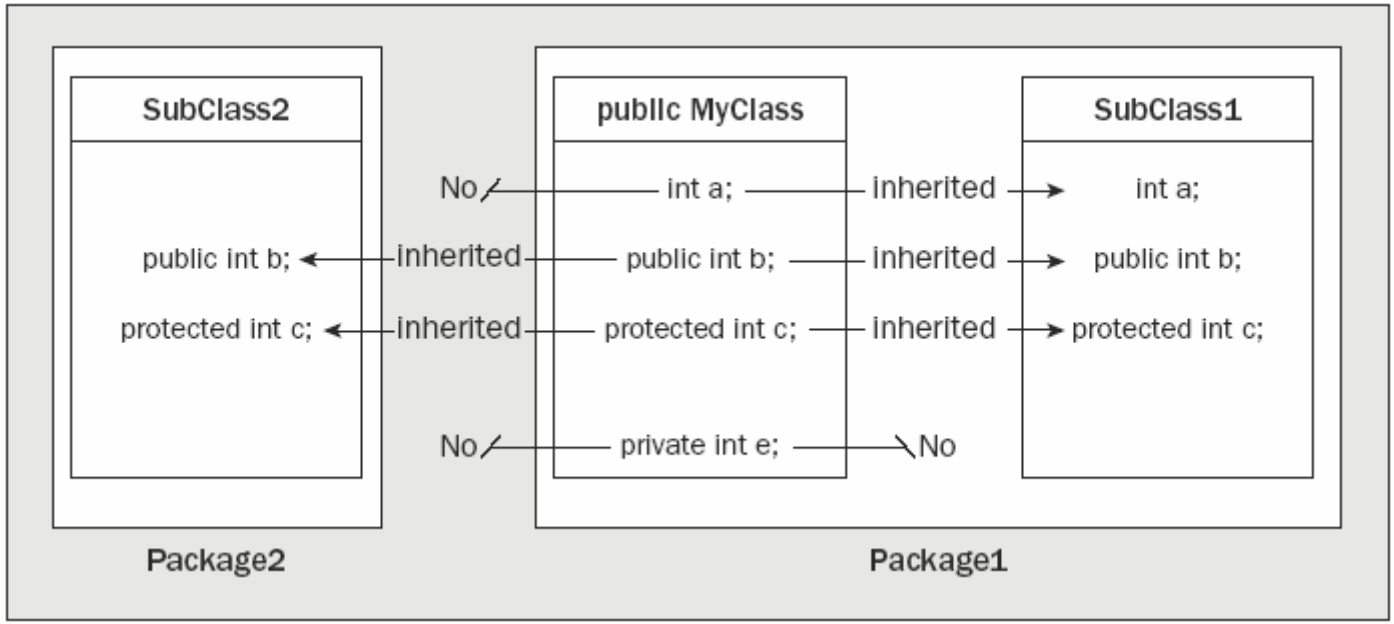
Još malo o access modifikatorima



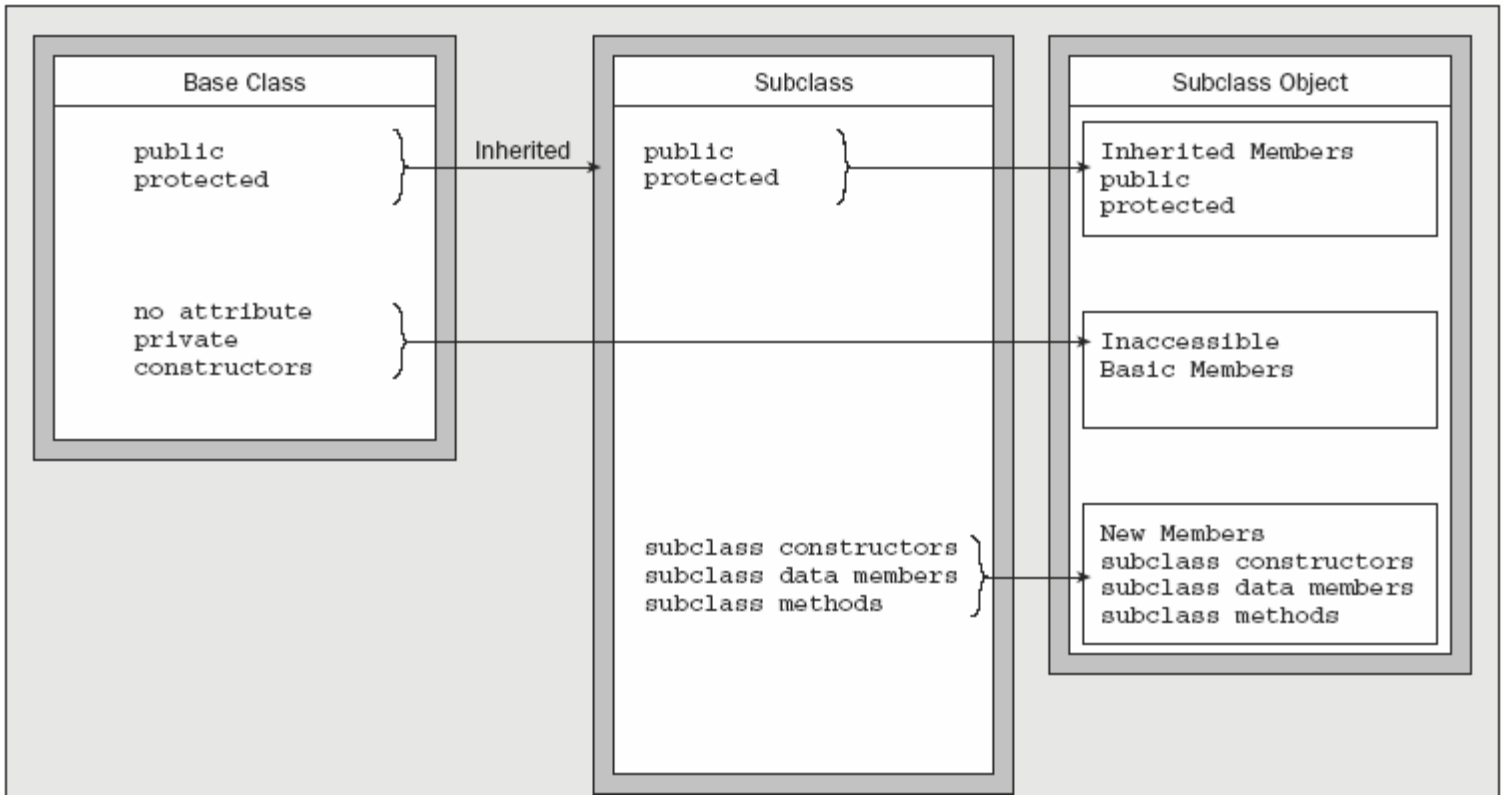
Još malo o access modifikatorima



Još malo o access modifikatorima



Još jednom - nasleđivanje



- 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!
- nasljeđivanje: uključivanje članova bazne klase u izvedenu klasu na način da su dostupni (acessible) u izvedenoj klasi
- nasleđeni član bazne klase je onaj koji je dostupan u izvedenoj klasi

Još jednom - nasleđivanje

- Unutar paketa svi članovi su dostupni i nasleđuju se osim članova označenih s `private`
- Klase van paketa mogu pristupiti članovima u drugom paketu ako je klasa kojoj pristupamo `public` i ako je član kom pristupamo označen s `public`
- Klase van paketa mogu naslediti članove klase iz drugog paketa ako je klasa kojoj pristupamo označena s `public` i ako su članovi `public` ili `protected`
- Moguće je i ovo

```
superclass
double value;
```

```
subclass
double value;
```

```
{
  value=10.25;
  super.value=11.25;
}
```

Još jednom - nasleđivanje

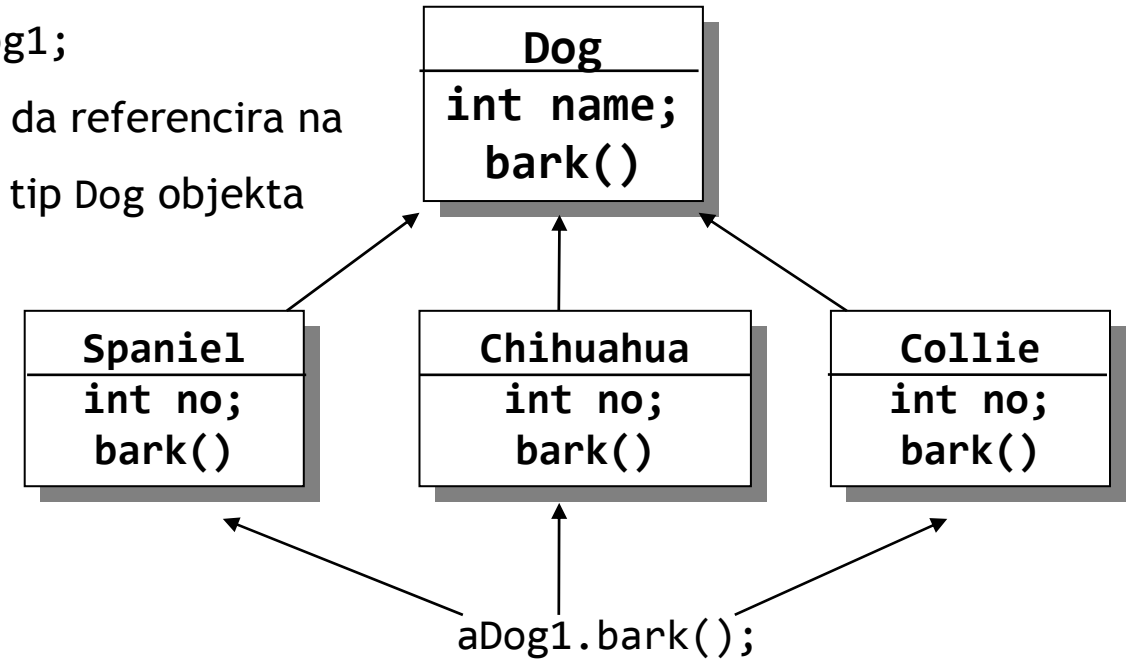
- Odabir atributa bazne klase
  - Metode koje sačinjavaju alat za komunikaciju sa spoljašnjim svetom/klasama se definišu kao `public`
  - Podaci članovi ne treba da budu `public` osim konstanti namijenjenih za opću upotrebu
  - Ako očekujete da će drugi ljudi koristiti vaše klase za izvođenje sopstvenih 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

Još jednom - polimorfizam

- mogućnost da s jednom varijablom određenog tipa referenciramo objekte različitih tipova i da automatski pozivamo metode koje su specifične za tip objekta na koji varijabla referencira
- uslovi
  - Poziv metoda podklase kroz varijablu bazne klase
  - Pozvana metoda mora biti i član bazne klase
  - Signatura metode i povratni tip moraju biti isti i u baznoj i u izvedenoj klasi
  - Atribut pristupa ne sme biti restriktivniji u izvedenoj klasi nego što je u baznoj klasi
  - Polimorfizam se odnosi samo na metode – reference baznog tipa mogu se koristiti samo za pristup podacima članovima baznog tipa

Još jednom - polimorfizam

```
Dog aDog1;
// može da referencira na
bilo koji tip Dog objekta
```




```
Dog  aDog1, aDog2, aDog3;
 aDog1 = new Dog(); aDog2 = new Spaniel(); aDog3 = new Collie();
 aDog1.bark();
 aDog2.bark();
 aDog3.bark();
 aDog1.no=2; //illegal
```

Object klasa

- Svaka klasa definisana u Javi dirktno ili indirektno nasleđuje klasu `Object`
- klasa `Object` se ne specificira eksplicitno

```
class HelloWorld extends Object {
  public static void main(String[] args) {
          System.out.println("Hello, World");}
  }
```

- Poseduje 7 `public` i 2 `protected` metoda
- Poziv metoda podklase kroz varijablu bazne klase
  - `protected  Object clone()`
  - `boolean equals(Object obj)`
  - `protected void finalize()`
  - `Class<?> getClass()`
  - `int hashCode()`
  - `void notify()`
  - `void notifyAll()`
  - `String  toString()`
  - `void wait()` sa jos dve oveloaded varijante

Object klasa

- `toString()`

  vraća `String` objekt s opisom trenutnog objekta. U nasleđenoj verziji ove metode to je naziv klase nakon koga sledi znak '@' i heksadecimalna reprezentacija objekta (hash code).

  Ovaj metod se poziva automatski pri konkatenaciji sa String varijablama korišćenjem '+'.

  Moguće je metod prepisati i dati drugačiji opis objekta.

  Očito je da metod `toString()` u izvedenoj klasi mora biti `public`
- `equals()` – pričali o tome - za većinu Javinih standardnih klasa poredi stanja objekata, ne reference (za konkretne klase proveriti API reference), inače ako nije prepisan poredi reference.
- `getClass()` – vraća objekt tipa `Class` koji nosi podatke o klasi kojoj pripada objekat čiji je `getClass()` pozvan
- `hashCode()` – vraća hash code objekta (koristi se za smeštanje objekata u `hash` tabele)
- `notify(), notifyAll(), wait()` – vezani su za niti, o tome kasnije
- `clone()` – kreira objekt koji je kopija trenutnog objekta, tzv. plitka kopija objekta
- `finalize()` – poziva se pri uništavanju objekta, tj. kad garbage collector zaključi da nema više referenci na objekat, ali se da prepisati i definisati dodatno ponašanje za definisanje "poslednjih želja" objekta.

Object klasa - getClass()

- getClass() – vraća objekt tipa Class koji nosi podatke o klasi kojoj pripada objekat čiji je getClass() pozvan

```
Animal pet= new Duck(...);
Class objectType = pet.GetClass(); // Uzmi tip klase
System.out.println(objectType.getname()); // Ispiši naziv klase
System.out.println(pet.getClass().getname()); // odjednom
```

- klasa Class
  - Kada se program izvršava JVM pravi instance klase Class za sve klase i interfejse tog programa
  - klasa Class nema public konstruktor tako da se ne možete instancirati od strane korisnika
  - Jedan od metoda ove klase je String getName() koji vraća String sa imenom klase (tačnije entiteta koji može biti - class, interface, array class, primitive type, or void)

```
if(pet.getClass()==Duck.class)
        System.out.println("It is a duck !");
```

(upotreba .class)
Referenca na Class objekt klase Duck

Object klasa - protected Object clone()

- clone() – kreira objekt koji je kopija trenutnog objekta, tzv. plitka kopija objekta
- objekti koji se kloniraju moraju naznačiti da je kloniranje nad njima prihvatljivo – implementiranjem interfejsa Cloneable

```
class Dog implements Cloneable
  { ... }
```

  Interfejs Cloneable deluje kao flag koji signalizira da je dozvoljeno kloniranje klase (marker interface)
- clone() metoda klonira objekt kreiranjem objekta istog tipa i postavljanjem svih polja u kloniranom objektu na istu vrednost kao i polja u kopiranom objektu
- Kada podaci članovi referenciraju na objekte, objekti se ne umnožavaju!
- Po konvenciji ovaj metod treba da bude prepisan i to public metodom

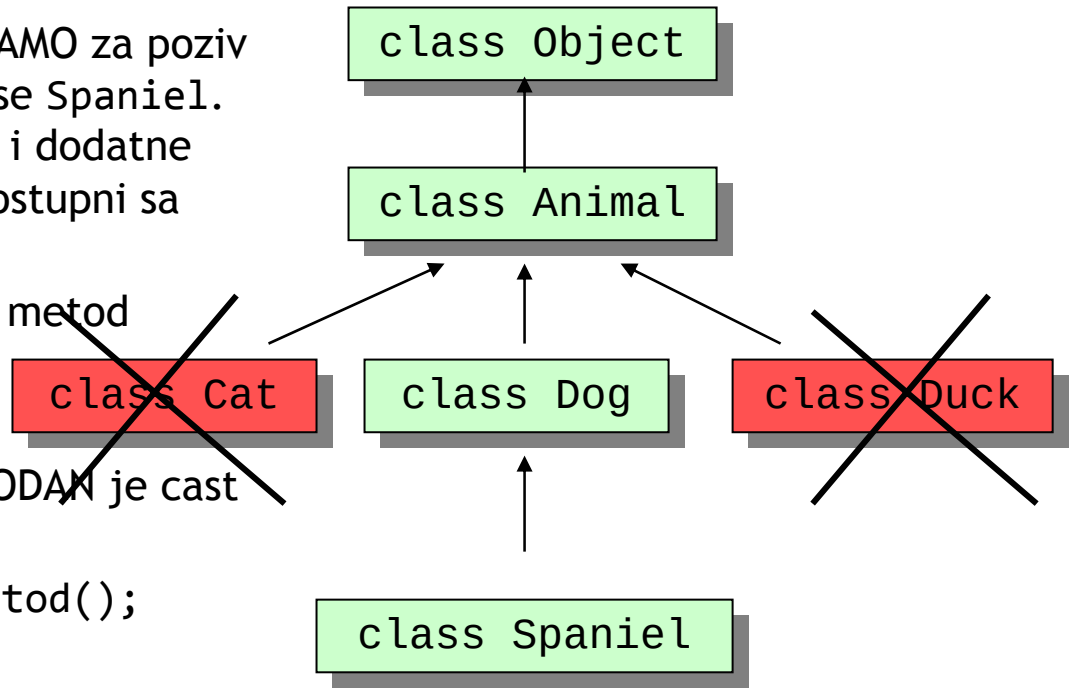
Cast objekata

- Cast objekata je moguć u slučaju da radimo cast izmedju klasa koje pripadaju istom lancu nasledjivanja (dakle, jedan tip mora da bude pottip drugog u bilo kom redosledu).
- Java zadržava sve informacije o originalnoj klasi kojoj objekat pripada !!!

```
Spaniel aPet = new Spaniel("Fang"); // Kreiraj objekat tipa Spaniel
Animal theAnimal = (Animal)aPet;
Animal theAnimal = aPet; // cast na gore - upward cast
Dog aDog = (Dog)theAnimal; // cast na dole (ekspl.) - downward cast
```

- aDog se može koristiti SAMO za poziv overrided metoda iz klase `Spaniel`. Dakle, ako `Spaniel` ima i dodatne metode, oni neće biti dostupni sa aDog refence.
- ako je potrebno pozvati metod specifičan za `Spaniel` klasu koristeći referencu aDog, NEOPHODAN je cast aDog-a na `Spaniel`.

```
((Spaniel)aDog).specmetod();
```



Kojoj klasi pripada? - instanceof

- operator instanceof vraća true ako je objekat referenciran levim operandom istog tipa kao i desni operand ili je bilo koji njegov podtip (tj. pripada podklasi klase kojoj pripada desni operand).

```
if(pet instanceof Duck) {
((Duck)pet).layEgg(); // It is a duck so You can have an egg
}
```

- razlika u odnosu na getClass()

```
Spaniel aPet = new Spaniel("Fang"); // Kreiraj objekat tipa Spaniel

    if (pet instanceof Dog) System.out.println("You have a dog!");
       else System.out.println("It's definitely not a dog!");

    if (pet.getClass() == Dog.class) System.out.println("You have a dog!");
       else System.out.println("It's definitely not a dog!");
```

izlaz u jednom i u drugom slučaju je obojen crvenom bojom.