2. Java osztályok

Az elterjedt nyelvekkel (C, C++, Delphi, stb.) ellentétben a Java már csak objektum alapokon mködik: a legkisebb fordítási egység az *osztály* lett. Mivel az objektum orientált programozás (*OOP*) témaköre nem túl könny - st mondhatjuk igen összetett: nézzük meg, mit is jelent objektum orientált módszertan szerint programot írni.

2.1. Elmélet

Az objektum alapú programozás közelebb hozta egymáshoz a programnyelv és a való élet fogalmait, így OOP alapon egyszerbb és átláthatóbb programok születhetnek, amelyek karbantartása is egyszerbb *lehet*, *ha* a programban használt fogalmak megközelítleg fedik a valóságot. A *jól megírt* OOP szemlélet program a valóságot próbálja modellezni, ehhez több fogalmat is tisztázni kell.

Az egyik legfontosabb fogalom maga az *objektum*, egy-egy objektum a való élet egy-egy részét *modellezi* a programunkban: egy objektum lehet például egy üveg, egy kis sör vagy egy pohár. Minden objektumnak van állapota, vannak lehetséges mveletei és képes más objektumokkal együttmködni. Ez azt jelenti, hogy egy üveg sört ki tudunk nyitni, ezzel a *"zárt"* és *"teli"* állapotát a *"nyitás"* mvelettel *"nyitott"* és *"teli"* állapotba hozni. Ha fogunk egy üres poharat, amelybe beletöltjük a nyitott üveg tartalmát, akkor a *"kitöltés"* mvelet eredménye egy *"teli"* pohár illetve egy *"üres"* és *"nyitott"* sörösüveg. Természetesen a sörrel teli poharat az *"ivás"* mvelettel üzemszer módon tudjuk használni... :)

Ha kerítünk egy újabb üveg sört, akkor az már egy *új objektum*, de észre kell vennünk a két üveg sör közötti *hasonlóságot*. Ha szólunk a csinos pincércsajnak, hogy még egy *üveg sört* szeretnénk, akkor szinte biztosak lehetünk benne, hogy nem egy üveg bambit fog hozni. Ehhez *mindkettnknek* tisztában kell lennünk az *üveg sör* fogalom jelentésével: a való életben a feltn hasonlóságok mentén - jobban mondva a közös jellemzk alapján - osztályokba tudjuk sorolni az objektumokat, egy-egy szavunk a dolgok egy-egy osztályát írja le. Az osztály határozza meg, hogy egy objektum milyen állapotokat vehet fel, milyen mveleteket lehet végezni vele, és milyen kölcsönhatások képzelhetk el más objektumokkal kapcsolatban. Az osztály neve egyedi kell legyen abban a környezetben, ahol hivatkozunk rá, ám fel kell készülnünk arra, hogy egy adott név másik környezetben mást jelent.

Mint mindent, a Java nyelv osztályait is példákon át ismerhetjük meg legjobban, ezért vegyük példának egy *kocsma absztakcióját*, hiszen oda mindenki ~~na ponta~~ idnként eljár – tehát a továbbiakban egy kocsmát fogunk ~~meglátogatni~~ absztrahálni.

Nézzünk körül egy kocsmában, mit is látunk: alapveten vannak *dolgok*. Ez egy igen *magas absztrakciós szint*, mivel teljesen elvonatkoztattunk attól, hogy a *dolgok* sörök, borok, zsíroskenyerek, üvegek, korsók, illetve poharak is lehetnek. Egy ilyen absztakció során az három dolgot kell megfigyelnünk a *dolog* osztályba sorolt *dolgokról*:

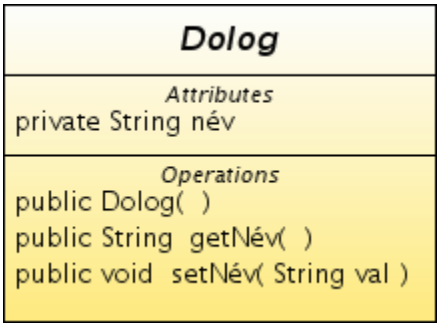
- milyen *számunkra lényeges* tulajdonságai vannak a *dolgoknak*
- milyen *számunkra lényeges* mveleteket végezhetünk a *dolgokon*
- milyen *számunkra lényeges* kölcsönhatások lehetnek a *dolgok* között

A *számunkra lényeges* azért van kiemelve, mivel ez az absztrakció lényege: csak azon tulajdonságokkal *kell* foglalkozunk, amelyek segítenek egymástól megkülönböztetni az osztályba sorolt különféle dolgokat, a többit figyelmen kívül *kell* hagynunk. Nézzünk újfent körül a kocsmában, és szedjük össze a *dolgok* **közös-de-megkülönböztet** jellemzit. Az osztályt úgy célszer elnevezznünk, hogy a neve a fogalmakat vagy a tárgyakat egy-két szóban jellemezze, és lehetleg ne használjunk többes számot. Mivel a leírásnak *minden egyes dologra* illenie kell a kocsmában, ezért az osztály neve az lesz, hogy **Dolog**.

2.1.1. Az osztály

Soroljuk fel a kocsmában található összes kézzelfogható dolog *lényeges* és *közös* jellemzjét. Mi az a tulajdonság, ami egyaránt jellemzi a sörösrekeszt, a sört, a zsíroskenyeret, a sörcsapot, a pultot és a kocsmárost? *Hát... hm... öööö... izé.*Az absztrakció nem is olyan egyszer, nézzük pár ötletet:

- Minden *dolog* kézzelfogható - nem jó, pont ezért elhagyható, mert minden *dologra* igaz.
- Minden *dolognak* neve van - ez jó, a név megkülönbözteti a *dolgokat*.
- Minden *dolognak* van mérete - nem jó, ez is elhagyható, mert minden *dologra* igaz.
- Minden *dolognak* van kinézete - nem jó, egyrészt túl összetett, másrészt pedig nem szükséges nekünk a *dolog* kinézete.



Maradjunk annyiban, hogy a dolgoknak egyelre csak neve van, ez kezdetnek egyszer és jó. Az osztályokat legjobban és legtömörebben UML osztálydiagrammon lehet leírni, amint az a jobb oldali ábrán is egy ilyen látszik. Az UML ábra tetején láthatjuk az osztály nevét, amelyet egyszeren úgy hívjuk, hogy *Dolog*. Az *Attributes* jelzi a tulajdonságokat, amelyek jellemzik az osztályt, jelen esetben egyetlen egy *String* típusú és *név* megnevezés tulajdonságunk van csak - ahogy ezt már szövegesen le is írtuk. Az osztállyal tudunk mveleteket végezni, jelen esetben semmi különös mveletünk nincs, van egy *konstruktor*, illetve egy *getter* (lekérdez) és egy *setter* (beállító) metódus a névhez. Az attribútumok és a mveletek eltt látható *private* és *public* szó jelzi, hogy az adott jellemz vagy mvelet elérhet-e az osztályon kívülrl - értelemszeren a public jelenti a publikus elérhetséget, a private pedig a privát használatot - de ezekrl késbb ejtünk több szót.

A *Dolog* osztály célja, hogy alapot adjon további osztályok definiálásához, ugyanis az objektum orientált programozás (és gondolkodás!) lényege az, hogy egyes osztályokból leszármaztatunk újabb osztályokat, amely esetben a leszármaztatott osztály specializáltabb lesz, mint az az osztály, amelybl leszármaztattuk (*figyelem*: nem szül és gyermek osztály a nevük, célszerbb az s és a leszármazott szó használata). Az UML ábrán az osztály neve dlt betvel van szedve, amely azt jelenti, hogy ez az osztály absztrakt osztály lesz, így ezt nem lehet példányosítani, csak azokat az osztályokat, amelyeket ebbl származtatunk le. A leszármaztatásról egy késbbi fejezetben lesz szó, egyelre tegyük félre ezt a fogalmat.

2.1.1.1. A Java osztályok

Tekinsük meg az UML ábra osztályleírását Java nyelven is:

Dolog.java

```
package hu.javakocsma;

public abstract class Dolog
{
}
```

Haladjunk végig a fenti osztályon:

- *package* ***csomagnév***: a csomag neve, ahol az osztály található. Nem kötelez, de elvárt a megadása. A csomagnév meghatározza, hogy a fájlban lév osztály melyik csomag része. Csomagokba vannak szervezve a Java környezet osztályai és az általunk írt osztályok is. Mivel a sokszázsokezer osztály között elfordulhatnak névazonosságok: a csomag neve különbözteti meg az azonos nev osztályokat. A csomag neve mindig utal a benne lév osztályok funkciójára, a *java.io* a be- és kimenet kezeléséért (fájlkezelés, írás, olvasás) felels osztályok gyjthelye, a *java.net* pedig a hálózati funkciók helye. Saját programok esetén általában a projekt neve a csomagok kiinduló pontja, másokkal (vagy a világgal) megosztott programok esetén pedig a gyártó cég vagy közösség internet nevét (és a program funkcióját) megfordítva kapjuk a csomag nevét (*hu.javaforum.csomagnév*). A csomagszerkezet teljesen azonos módon kezelend, mint a háttértáron lév könyvtárstruktúra; s ez olyannyira analóg, hogy csomagok gyakorlatilag meghatározzák, hogy a bennük foglalt osztályok melyik könyvtárban vannak. Egy csomag tartalmazhat alcsomagot, amelynek szintén lehet alcsomagja; ahogy a csomag mappa tartalmazhat almappát, amelynek szintén lehet almappája. Vagyis a csomagok fa struktúrát alkotnak.
- *class* ***OsztályNév***: az osztály neve. Minden osztályt névvel kell ellátnunk, amely név a Java nyelv elírásait teljesíti. Ennek a névnek azonosnak kell lennie (kisbetvel és nagybetvel írt név különböz!) a forrást tartalmazó fájl nevével, egy osztályt hordozó állomány nevét az osztály neve után írt *.java* kiterjesztéssel képezzük. Ha az osztály neve több szóból áll, akkor minden szót nagybetvel kezdünk, és egybeírjuk - szóköz nélkül (például: *OsztályNév*). Bár a nyelv megengedi a Unicode karakterek használatát osztályok esetén is, a fájlrendszerek nem mindig teszik lehetvé az ékezetes betk korrekt kezelését, ezért lehetleg csak az angol karaktereket használjuk. A *class* kulcsszó elé kerülhetnek különféle módosítók (használatuktól késbb):
  - *abstract*: az osztály közvetlenül nem példányosítható, csak leszármazottai által.
  - *final*: az osztálynak nem lehetnek leszármazottai.
  - *private*: az osztály elérése privát, ez csak beágyazott osztályok esetén használható.
  - *protected*: az osztály elérése korlátozott, ez is csak beágyazott osztályok esetén használható. A *protected* megengedi, hogy a leszármazott osztályok is elérhessék ezeket a beágyazott osztályokat.
  - *public*: az osztály nyilvános, bárki számára elérhet.
  - *static*: az osztály csak egy példányban jön létre, ez szintén csak beágyazott osztályok esetén használható.
  - *strictfp*: az osztály törzsében a lebegpontos mveletek IEEE 754 szerint futnak le.
  - Szabályok:
    - Az *abstract* és a *final* nem használható együtt.
    - A *public*, a *private* és a *protected* közül csak egyet használhatunk. Ha egyiket se adjuk meg, akkor az osztályt csak az azonos csomagban lév osztályok látják, mások számára "láthatatlan".

Az osztály törzse kapcsos zárójelek közé kerül, ha az osztály üres, akkor is ki kell tennünk a nyitó és záró kapcsos zárójelet.

2.1.1.2. Tulajdonságok

Haladjunk tovább, én egészítsük ki ezt az UML ábrán látott tulajdonságokkal:

Dolog.java

```
package hu.javakocsma;

public abstract class Dolog
{
  private String név;
}
```

A tulajdonságokat általában az UML leírás szerint soroljuk fel, egy tulajdonságot egy sorba - az UML tulajdonságot a Java programban már *változónak* jobban mondva *példányváltozónak* hívjuk, és az osztályba foglalását *deklarációnak*:

- *private String* ***név***: a változó deklarációja, lássuk visszafelé haladva:
  - *név*: a változó neve, egy osztályon belül a változó neve egyedi kell legyen, s a változók elnevezésének is vannak szabályai. Minden változónevet kisbetvel írunk, ha a név több szóból áll, akkor az els szót leszámítva minden további szót nagybetvel kezdünk, és egybeírjuk - szóköz nélkül (például: *változóNév*).
  - *String*: a változó típusa, amely azt határozza meg, hogy a változóba milyen típusú adatokat tudunk tenni. A változók típusáról a késbbiekben bvebben megemlékezünk.
  - *private*: az osztályhoz hasonlóan a változó mködését és elérését a módosítók befolyásolhatják:
    - *final*: a változó értéke végleges, az a program futása során már nem változhat (más néven konstans változó). A *final* módosítóval ellátott változó nevét másképp írjuk! A változó nevét nagybetvel írjuk, ha több szóból áll, akkor szóköz helyett aláhúzást teszünk (például: *VÁLTOZÓ_NÉV*).

- *private*: a változót csak az osztályon belül lehet elérni és használni, a leszármazott osztályok sem használhatják.
- *protected*: a változót az azonos csomagban lév illetve leszármazott osztályok is elérik.
- *public*: a változó nyilvános, bárki számára elérhet és módosítható.
- *static*: a változó egyetlen példányban létezik csak - ekkor *osztályváltozónak* nevezzük.
- *transient*: a változó nem határozza meg az osztály állapotát, értéket csak átmenetileg tárol.
- *volatile*: a változó nem kerül gyorstárba, mivel tartalma több szál által is változhat.
- Szabályok:
    - A *final* és a *volatile* nem használható együtt.
    - A *public*, a *private* és a *protected* közül csak egyet használhatunk. Ha egyiket se adjuk meg, akkor a változót csak az azonos csomagban lév osztályok látják, a leszármazottak számára már "láthatatlan".

2.1.1.3. Mveletek

Lássuk az UML ábra utolsó harmadát, a mveleteket is:

Dolog.java

```
package hu.javakocsma;

public abstract class Dolog
{

  private String név;

  public Dolog()
  {
  }

  public String getNév()
  {
    return this.név;
  }

  public void setNév(String név)
  {
    this.név = név;
  }
}
```

Mint látható, az osztályban már az UML ábra mveletei is fel vannak sorolva, Java nyelvben *konstruktor* és *metódus*néven hivatkozunk a továbbiakban ezekre a mveletekre.

Konstruktorok

Emeljük ki a konstruktort, hogy megvizsgálhassuk közelebbrl is:

Dolog.java

```
public Dolog()
{
}
```

A konstruktor legfbb ismérve, hogy a neve azonos az osztály nevével és a név után szorosan a metódusokra is jellemz kerek zárójelet találjuk meg, a zárójelek között pedig paramétereket - a jelenlegi példában nincs ilyen paraméter - a paraméter nélküli konstruktort *alapértelmezett konstruktornak*nevezzü k. A konstruktor neve eltt lehetnek módosítók:

- *private*: a konstruktor csak az osztályon belül hívható meg, ez akkor hasznos, ha limitálni szeretnénk a létrehozott példányokat.
- *protected*: a konstruktort az azonos csomagban lév illetve leszármazott osztályok képesek elérni.
- *public*: a konstruktor nyilvános, bárki számára elérhet.
- Szabályok:
    - A *public*, a *private* és a *protected* közül csak egyet használhatunk. Ha egyiket se adjuk meg, akkor a konstruktort csak az azonos csomagban lév osztályok látják, a leszármazottak számára már "láthatatlan".

Hasonlóan a osztályhoz - a konstruktornak is van törzse, amelyet kapcsos zárójelek zárnak közre. Ha a törzs üres, akkor is ki kell tennünk a nyitó és a záró kapcsos zárójelet.

A konstruktor feladata, hogy elkészítse az osztály egy új példányát. Egy osztálynak több példánya is lehet, a példányokat a példányváltozók különböztetik meg egymástól. Ha két példánynak azonos értékek vannak a példányváltozóikban - például két azonos nev dolog, akkor a két példány ugyan *egyenl egymással*, de mégis *két külön példánynak számítanak*.

Ha egy osztályban nincs konstruktor, akkor azt úgy kell vennünk, mintha a fentebb említett minimális konstruktor lenne benne - ez azt jelenti, hogy a *Dolog* osztályból a jelenlegi konstruktort el is hagyhatnánk, nem történne érdemi változás a program mködésében. Természetesen egy osztályban lehet több konstruktor is - de erre visszatérünk a késbbiekben.

Metódusok

Egy metódus sok dologban hasonlít a konstruktorhoz, azonban a metódus neve eltt szerepeltetni kell egy típust, amely típusú értéket a metódus visszaad. Egy metódusra úgy is gondolhatunk, mint egy matematikai függvényre, például az *y = sin(x)* esetén a színusz függvény az *x* érték színuszát adja értékül az *y* változónak. Nézzük meg egy átlagos metódust közelebbrl:

Dolog.java

```
public String getNév()
{
  return this.név;
}
```

A metódus feje távolról hasonlít a példányváltozók deklarálásához, attól *"csak"*a kerek zárójelek és a metódus törzsét jelent kapcsos zárójelek különböztetik meg:

- *public String getNév()*: a metódus feje, haladjunk a végétl az eleje felé:
  - *()*: a kerek zárójelek közé átadott paramétereket tudunk felsorolni, jelen esetben nem adunk át paramétert.
  - *getNév*: a metódus neve, amelyre ugyan azok a szabályok érvényesek, mint a példányváltozókra - leszámítva, hogy egy osztályon belül lehet több azonos nev metódus, ha eltér paramétereik vannak.
  - *String*: a metódus átlal visszaadott érték típusa, ha nem akarunk értéket visszaadni, akkor *void* szerepelhet ezen a részen.
  - *public*: az változókhoz hasonlóan a metódus mködését és elérését a módosítók befolyásolhatják:
    - *abstract*: a metódus törzs nélküli, a törzset a leszármazott osztályok tartalmazzák.
    - *final*: a metódus törzse végleges, a leszármazott osztályban nem lehet azt megváltoztatni felüldefiniálással.
    - *native*: a metódus törzs nélküli, a törzse nem Java nyelven van megvalósítva.
    - *private*: a metódust csak az osztályon belül lehet elérni és használni, a leszármazott osztályok sem használhatják.
    - *protected*: a metódust az azonos csomagban lév illetve leszármazott osztályok is elérik.
    - *public*: a metódus nyilvános, bárki számára elérhet és módosítható.
    - *static*: a metódus csak statikus példányváltozókkal tud dolgozni, s használatához nem kell az osztályt példányosítani.
    - *synchronized*: a metódus törzsében egyszerre csak egy szál tartózkodhat, erre a környezet ügyel.
    - *strictfp*: a metódus törzsében a lebegpontos mveletek IEEE 754 szerint futnak le.
    - Szabályok:
      - Az *abstract* metódus nem lehet *private*, *static*, *final*, *native*, *strictfp* vagy *synchronized*.
      - Egy *abstract* metódus csak olyan osztályban lehet deklarálva, amelyik maga is *abstract*.
      - Az *abstract* vagy a *native* metódusnak nem lehet törzse, a kapcsos zárójelek helyett pontosvesszvel kell lezárni.
      - A *public*, a *private* és a *protected* közül csak egyet használhatunk. Ha egyiket se adjuk meg, akkor a metódust csak az azonos csomagban lév osztályok látják, a leszármazottak számára már "láthatatlan".
      - A *native* és a *strictfp* nem használható együtt.

A metódus törzsét a kapcsos zárójelek között találjuk meg - a példában ez üres, ide kerülnek a Java nyelv utasítások, amelyek lépésrl-lépésre elvégzik a kért mveletet:

Dolog.java

```
return this.név;
```

Jelen esetben a *getNév* (publikus) metódus visszaadja a privát elérés *név*példányváltozó által hordozott értéket. Nézzük meg a következ metódust is:

Dolog.java

```
public void setNév(String név)
{
  this.név = név;
}
```

Ennek a metódusnak már van egy paramétere, amely *String* típusú és neve egyszeren *név*. A metódus törzse egy értékadás, ahol a paraméterben kapott értéket adjuk át a példányváltozónak.

Bean pattern

A két metódus közül az els a *getter*, a második a *setter*; a kett publikus metódus és a privát példányváltozó együtt valósítják meg a *bean pattern* nev fogalmat, amely a Java egyik alapfogalma. Java nyelven bizonyos osztályokat úgy nevezünk, hogy *bean* (mint kávébab). Egy ilyen bean olyan osztály, amelynek van egy alapértelmezett konstruktora és példányai állapotát csak olyan példányváltozók határozzák meg, amelyekhez egy getter és egy setter metódus tartozik az alábbi módon:

Java

```
private String név;

public String getNév()
{
  return this.név;
}

public void setNév(String név)
{
  this.név = név;
}
```

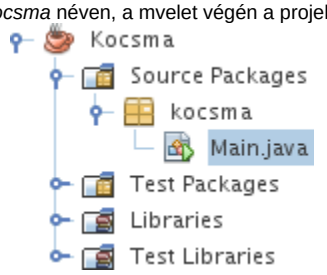
Nézzük részletesen:

- A példányváltozó privát, a két metódus publikus.
- A példányváltozó típusa azonos a getter metódus által visszaadott típussal és a setter metódus egyetlen paraméterének típusával.
- A getter metódus neve a példányváltozó nevébl képzdik: egy *get* szót illesztünk elé és a név els betjét nagybetssé tesszük.
- A setter metódus neve a getterhez hasonlóan képzdik, csak a *set* szót illesztjük a példányváltozó neve elé.
- A getter metódus visszaadja a példányváltozó értékét.
- A setter metódus beállítja a példányváltozó értékét a paraméterben kapott értékére.

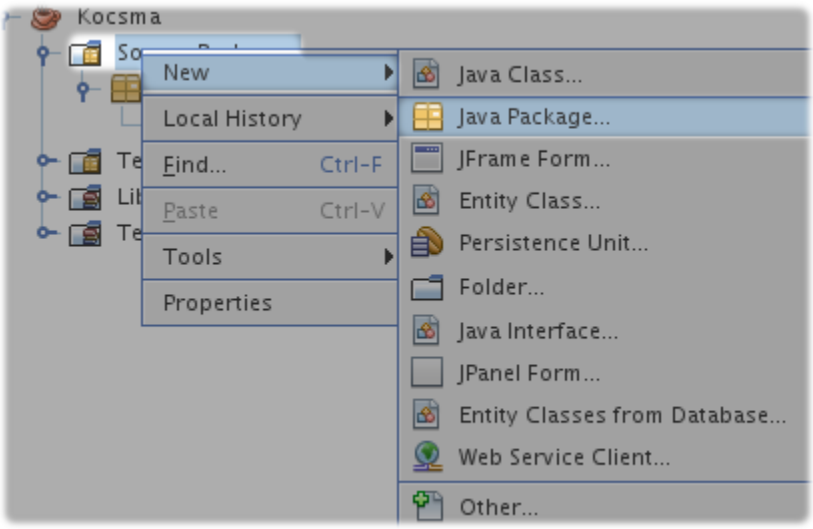
Vonjuk le a következtetést: a *Dolog* osztályunk egy Java bean.

2.2. Gyakorlat

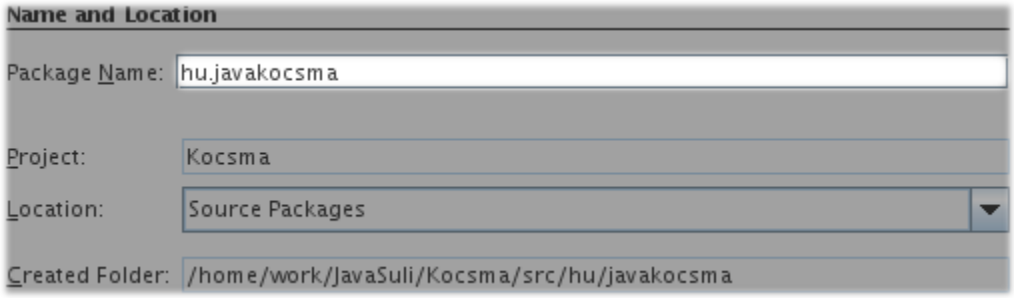
Hozzunk létre (a már megismert módon) egy új projektet *Kocsma* néven, a mvelet végén a projekt nézetben az alábbit kell látnunk:



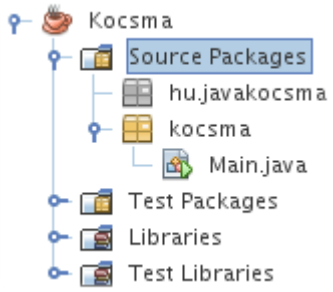
Ügyeljünk arra, hogy a projekt neve nagy betvel kezddjön, és ne tartalmazzon szóközt. Az elméleti részben hivatkozott *hu.javakocsma* csomagot kell elször létrehoznunk, ehhez jobb egérgombbal kattintani kell a *Source Packages* feliraton, majd a menübl a bal egérgomb segítségével ki kell választanunk a *New*, illetve a felbukkanó újabb menübl a *Java Package...* menüpontot:



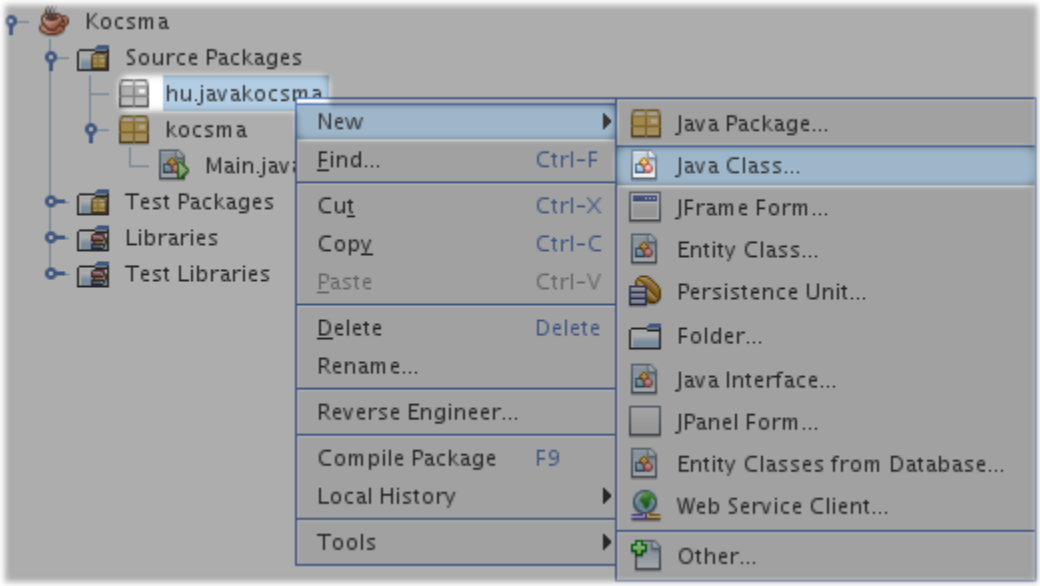
A dialógus ablakban a csomag nevét kell csak megadnunk, majd a *Finish* gombon kell kattintsunk:



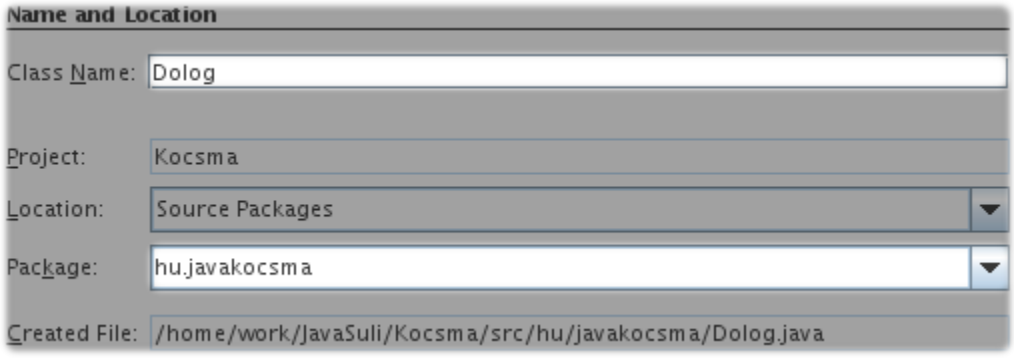
Ezek után az új csomagnak meg kell jelennie a projekt nézetben:



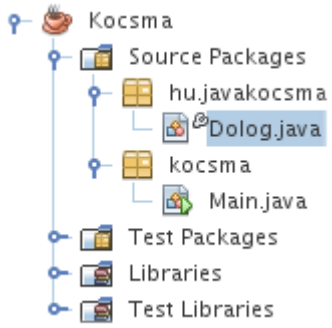
Az újonnan létrehozott csomagon jobb egérgombbal kattintva létre kell hoznunk egy új osztályt:



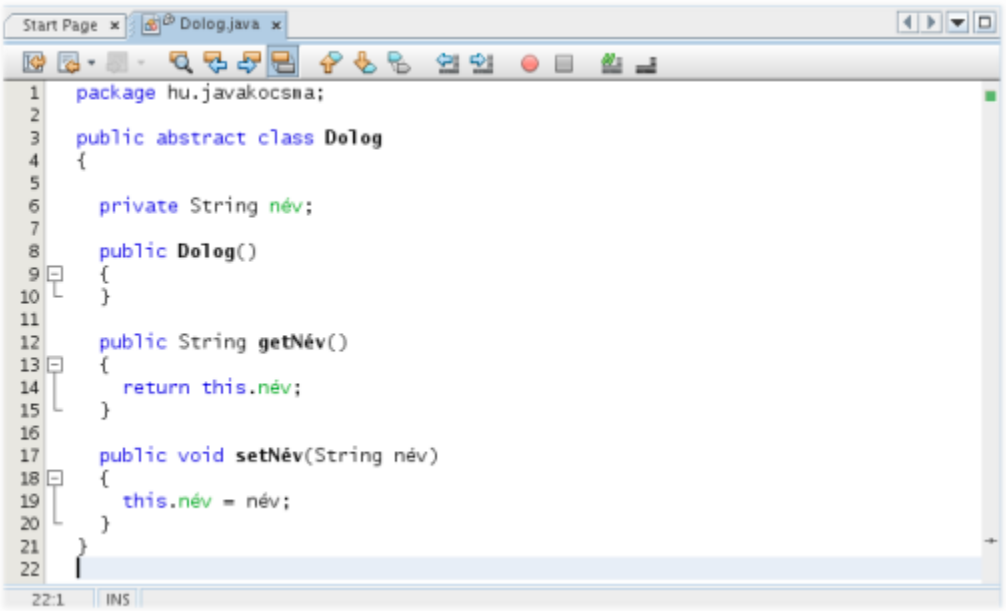
Nevezzük el az új osztályt *Dolog* néven, és ellenrizzük, hogy a *hu.javakocsma* csomagban található:



A projekt nézetben ezek után látnunk kell az új osztályt az új csomag alatt:



A NetBeans szerkeszt panelben pedig megjelenik egy új fülön a *Dolog*osztály tartalma, ezt írjuk át arra, amit az elméleti részben láttunk:



Ezek után nincs más hátra, minthogy az *F11*gomb megnyomásával a projektet futtatható állapotba hozzuk:

Result

```
init:

deps-jar:
compile:
Building jar: /home/work/JavaSuli/Kocsma/dist/Kocsma.jar
Not copying the libraries.
To run this application from the command line without Ant, try:
java -jar "/home/work/JavaSuli/Kocsma/dist/Kocsma.jar"
jar:
BUILD SUCCESSFUL (total time: 0 seconds)
```

Nos, létrehoztuk az els saját osztályt NetBeans alatt, veregessük meg a vállunkat - jó munkát végeztünk... 🙂