elnevezése a `Listener` posztfixet fogja kapni, például egy `et` eseménytípus esetén a név `EtListener` lesz.

- A komponens a lényegi változásokról eseményekben értesíti a hozzá kapcsolódó komponenseket, így egyrészt lehetővé teszi a figyelők kapcsolódását és esetleges lekapcsolódását, valamint az események bekövetkezésekor értesíti azokat. Egy `et` eseménytípus esetén a figyelő hozzákapcsolódása az `addEtListener`, lekapcsolódása a `removeEtListener` elnevezésű metódus segítségével történik, amelyek egy `etListener` típusú paramétert fogadnak. A komponens `TooManyListenersException` kivétellel jelzi a hozzákapcsolódás során, ha csak adott számú (pl. egyetlen) figyelő kapcsolódhat hozzá.
- A komponens tetszőleges módszert alkalmazhat az esemény bekövetkeztekor a figyelők értesítésére, azonban ezt a kódrészt általában kiemelik egy adott (pl. `protected`) metódusba (pl. `ee` esemény esetén `fireEe`), hogy az több helyről is hivatkozható legyen.
- Ha egy komponens értesítést kíván kapni adott típusú eseményekről, akkor implementálja a megfelelő interfészt és regisztrálja magát az alapkomponens eseményfigyelőjeként.

Objektum-szerializáció

Összetett alkalmazások esetén lényeges, hogy az alapvető, módosított értékek, adatok a program leállításakor ne vesszenek el, és a következő indításkor is rendelkezésre álljanak. A programvégrehajtás során a csak időlegesen szükséges, ún. tranziens („átmeneti”) adatoktól így megkülönböztetjük azokat, amelyek külsőleg is tárolandók, „tartósak”, „állandók”, azaz perzisztensek.

A külső tárolás lehetősége központi szerepű az alkalmazások esetén. A Java erre egy beépített mechanizmussal rendelkezik, az ún. szerializációval.

A szerializáció műveleteit az `InputStream`, illetve az `OutputStream` egy-egy leszármazottja, az `ObjectInputStream`, illetve az `ObjectOutputStream` biztosítja. Az ezekben található `readObject`, illetve `writeObject` metódusokkal egyetlen utasítással kitárolhatjuk vagy visszatölthetjük az objektumok teljes rendszerét. Mindehhez mindössze annyit kell tennünk, hogy a tárolandó objektumok esetén engedélyezzük a szerializációt, azaz implementáljuk a `Serializable` interfészt.

```
…
  ObjectOutputStream os=new ObjectOutputStream(
                            new FileOutputStream("dvdshop.ser")
                       );
  os.writeObject(shop);
  os.close();
…
```

Az így kapott állomány ugyancsak néhány sorral visszatölthető:

```
…
  ObjectInputStream is=new ObjectInputStream(
                            new FileInputStream("dvdshop.ser")
                         );
  DVDShop shop2=(DVDShop)is.readObject();
  is.close();
…
```

A szerializált állomány bináris formában tárolja az objektumot, illetve az ahhoz kapcsolódó további objektumokat és elemi értékeket.

A szerializációt mi is vezérelhetjük. Abban az esetben, ha bizonyos mezők tárolása nem szükséges (pl. ideiglenes tárak, „cache”-ek esetén), azok szerializációját lekapcsolhatjuk a mező definíciójában a mezőtípus elé a `transient` kulcsszót írva.

A szerializáció azt is lehetővé teszi, hogy átvegyük a tárolásnál alkalmazott kódolás teljes kontrollját. Ehhez mindössze a következő két metódust kell implementálnunk:

```
private void readObject(java.io.ObjectInputStream stream)
               throws IOException, ClassNotFoundException;
private void writeObject(java.io.ObjectOutputStream stream)
               throws IOException
```

Ha léteznek a megfelelő metódusok, akkor a szerializáció alapértelmezett mechanizmusa helyett ezek műveletei határozzák meg az objektum kódolását. Mivel az `ObjectInputStream` implementálja a `DataInput`, az `ObjectOutputStream` pedig a `DataOutput` interfészt, ezért a kiírások/beolvasások műveleteit visszavezethetjük a részobjektumok, illetve elemi értékek írásának-olvasásának a műveleteire. Természetesen a beolvasás során a kiírásnak megfelelő sorrendet kell követni.

A metódusok mindig az aktuális (`this`) objektumra vonatkoznak. További két metódus, a `writeReplace`, illetve a `readResolve` definiálásával a ténylegesen betöltendő, illetve kiírandó objektumon is módosíthatunk.

A `readObject` és `writeObject` metódusokkal megvalósított szerializáció során elegendő csak az osztályban definiált mezőket kezelni. A szuperosztályok, illetve az alosztályok értékei vagy az alapértelmezett, vagy az ott definiált saját szerializációval íródnak ki.

Az is lehetséges, hogy teljesen átvegyük a szerializálás feladatát. Ehhez a `Serializable` interfész kiterjesztését, az `Externalizable` interfészt kell implementálnunk, illetve az abban deklarált két, `readExternal`, illetve `writeExternal` elnevezésű metódust kell megvalósítanunk.

A szerializáció nagyon függ az osztály felépítésétől, így ha pl. a fejlesztés során módosítunk az osztály definícióján, akkor már valószínűleg nem tudjuk visszatölteni a korábban elmentett objektum-rendszert. A szerializáció ezért csak az úgynevezett rövid-idejű perzisztencia (`short-term persistence`) megoldására ajánlott, amikor pl. a hálózaton átküldünk egy objektumot (vagy objektum-rendszert), vagy azt a külső tárolóra időlegesen kitároljuk, majd visszatöltjük.

„Hosszú-idejű” szerializáció XML dokumentumként

Ahogy említettük, az alap szerializációs technika használata csak rövid-idejű perzisztencia esetén ajánlott. (Részmegoldás lehet, ha saját szerializációt valósítunk meg.)

A hosszú idejű perzisztencia (`long-term persistence`) rendkívül fontos a komponensek esetén. Egy kimentett komponens-rendszert akkor is fel kell tudnunk használni, ha a komponensek definícióin jelentős bővítéseket hajtottunk végre.

A Java a Beanek hosszú idejű perzisztenciájára azt a megoldást kínálja, hogy lehetővé teszi az adatoknak egy adott szerkezetű XML-be történő kitárolását, illetve az abból történő betöltést.

Az XML betöltés és tárolás alapvetően a `java.beans` csomag `XMLEncoder` és `XMLDecoder` osztályainak a segítségével történik.

A kiírás csak néhány sor:

```
…
  XMLEncoder out=new XMLEncoder(
                              new BufferedOutputStream(
                                new FileOutputStream("dvd1001.xml")
                              )
                            );
  out.writeObject(shop.getDVDForCode("1001"));
  out.close();
…
```

Hasonlóan, a beolvasás is:

```
…
  XMLDecoder in=new XMLDecoder(
                              new BufferedInputStream(
                                new FileInputStream("dvd1001.xml")
                              )
                            );
  DVD dvd2=(DVD)in.readObject();
  out.close();
…
```

A bináris szerializációhoz hasonlóan, az XML előállítást és beolvasást is vezérelhetjük, így pl. létezik megoldás bizonyos tulajdonságok tranziensként („nem mentendőként”) való bejelölésére. A kódoló/dekódoló a működése során úgynevezett perzisztencia-megvalósításokat (`PersistenceDelegate`) használ; saját XML-részletekkel ezeken keresztül egészíthetjük ki az alap-mechanizmust.

A `XMLEncoder` segítségével megvalósított hosszú-idejű perzisztencia maga is egy időigényes művelet, ezért, ha csak rövid idejű tárolásról van szó, akkor célszerűbb a bináris szerializációt választani.