JAVA 2

Serijalizacija

Teme

- Serijalizacija i deserijalizacija
- *Serializable interface*
- Verzioniranje – *serialVersionUID*
- Prilagođavanje serijalizacije i *transient*
- Odgovornosti kod serijalizacije
- Problemi sa *Singleton patternom*
- *Externalizable interface*

Serijalizacija i deserijalizacija

- serijalizacija – zapisivanje stanja objekata u binarne tokove radi:
  - pohrana na podatkovnom sustavu
  - dijeljenje objekata mrežnom komunikacijom – web services, RMI
  - . . .
- deserijalizacija – obrnut proces regeneriranja objekata iz binarnih tokova
- *static* ne odražava stanje objekta pa se ne pohranjuje
- klase stvorene za nasljeđivanje ne bi treble biti serijalizabilne, kao ni *inner* klase!

Serializable interface

- *marker interface* – ne sadrži metode, već deklarativno nagovješta sustavu da se objekt može serijalizirati
- *mixin interface* – implementacija opcionalnih ponašanja klase
- serijalizira se cijeli *object graph* - sve reference na objekte koje referencira instanca moraju biti *Serializable* – u suprotnom *java.io.NotSerializableException*
- ne shvaćati olako – implementacija smanjuje fleksibilnost budućih izmjena, ali i otvara javnim privatne dijelove implementacije
- **.ser** ekstenzija je konvencionalno preporučena

Verzioniranje - *serialVersionUID*

- *private static final long serialVersionUID*
- ako nije definiran, JVM ga automatski definira – gubimo kontrolu verzioniranja
- ovaj *static* se zapisuje, ali ne kao stanje objekta već kao *metadata*
- objekti se mogu deserijalizirati jedino ako im je *serialVersionUID* klase jednak onomu u binarnom toku
- ako promijenimo definiciju klase (varijable instance), slijedno bi trebali promijeniti i *serialVersionUID* – osiguranje ispravnosti objekta

Prilagođavanje serijalizacije i *transient*

- Java specificira predodređeni način zapisivanja objekata
  - *ObjectOutputStream, ObjectInputStream* koji se *Decorator Patternom* uokviruju u specifičnije (npr. *FileOutputStream, FileInputStream*)
- prilagodba serijalizacije – implementacija eksplicitnih metoda
  - *private void writeObject(ObjectOutputStream oos) throws IOException*
  - *private void readObject(ObjectInputStream ois) throws IOException, ClassNotFoundException*
  - *private* – sprečavanje premošćivanja i samo JVM provjerava i poziva
- *transient* – označavamo varijable koje ne želimo serijalizirati pa će prilikom deserijalizacije imati predodređeno stanje tog tipa

Odgovornosti kod serijalizacije

- serijalizabilna klasa nasljeđuje klasu koja to nije
  - serijalizacija – sprema predodređeno stanje *parenta*
  - deserijalizacija – poziva se *defaultni* konstruktor *parenta* – *parent* mora imati *default* konstruktor!
- deserijalizacija – tretiramo ju kao i konstruktor
  - ako postoji validacija u konstruktorima, i ovdje mora biti osigurana
  - ako konstruktor kreira *defensive copy* (sprečavanje izmjene prosljeđenog objekta), ovdje također mora biti osigurano

Problemi sa *Singleton Patternom*

- deserijalizacija kreira novu instancu objekta!
- Singleton koji implementira *Serializable* - onemogućen *singleton pattern* u kojem smo potrebiti samo jedne instance
- rješenje – implementacija metode

  private Object readResolve()
  - ovdje možemo vratiti vlastitu instancu, a instanca izgrađena deserijalizacijom je *eligible for garbage collection*
- koristiti enum za *Singleton pattern!*
  - enum serijalizira samo svoje ime, na temelju kojeg se metodom *Enum.valueOf(name, type)* deserijalizacijom izgradi jedinstvena instanca

Externalizable interface

- Predodređena serijalizacija u Javi nije pretjerano efikasna

- *Externalizable interface* omogućuje, ali i uvjetuje implementaciju vlastitih metoda serijalizacije i time veću kontrolu:

```
@Override
 public void writeExternal(ObjectOutput out) throws IOException
@Override
 public void readExternal(ObjectInput in) throws IOException,
                                                 ClassNotFoundException
```

Demo

- Project

Izvor:http://www.jnhsolutions.com/contact-us/request-a-demo/

Hvala na pažnji!