Programowanie obiektowe

Wykład 11: Serializacja obiektów

Problem składowania/przesyłania danych

- Strumienie
  - Tylko typy proste
  - Konieczność „ręcznego” odtwarzania obiektów
- Serializacja
  - Zapis/odczyt całych obiektów
  - Konwersja do postaci binarnej lub tekstowej(XML)
  - Automatycza deserializacja

Strumienie do serializacji

- *ObjectOutputStream(OutputStream)*
  - *writeObject(Object o)*
- *ObjectInputStream(InputStream)*
  - *Object readObject()*

Klasy muszą implementować interfejs *Serializable*
(pakiet Java.io)

Przykład 1 (1)

```
import java.io.*;
class Osoba implements Serializable {
  String nazwisko;
  String imie;
  Adres adresZameldowania;
   Osoba(String nazwisko, String imie,
        Adres adresZameldowania) {
      this.nazwisko = nazwisko;
      this.imie = imie;
      this.adresZameldowania =
                    adresZameldowania;
       System.out.println("wywołanie
                konstruktora klasy Osoba");
   }
   public String toString() {
     String adrPamiec = super.toString();
    return adrPamiec+"(" + nazwisko + ", " +
      imie + ", " + adresZameldowania + ")";
   }
}
```

```
class Adres implements Serializable {
   String miasto;
   String ulica;
   String nrDomu;
   String nrLokalu;
   Adres(String miasto, String ulica, String nrDomu,
              String     nrLokalu) {
      this.miasto = miasto;
      this.ulica = ulica;
      this.nrDomu = nrDomu;
      this.nrLokalu = nrLokalu;
      System.out.println("wywołanie konstruktora klasy Adres");
    }
   public String toString() {
       String adrPamiec = super.toString();
       return adrPamiec + "(" + miasto + ", " + ulica + ", " +
           nrDomu + ", " + nrLokalu + ")";
}
}
```

Przykład 1 (2)

```
public class TestSerializacji {
   public static void main(String[] args) throws Exception {
      Adres alternatywy4 = new Adres("Warszawa", "Alternatywy", "4", "9");
      Osoba kotek = new Osoba("Kotek", "Zygmunt", alternatywy4);
      System.out.println(kotek);
            // wersja dla Linuxa
      //String nazwaPliku = "/tmp/lista.ser";
            // wersja dla Windows
      String nazwaPliku = "c:\\lista.ser";
      ObjectOutputStream out = new ObjectOutputStream(
                                       new BufferedOutputStream(
                                            new FileOutputStream(nazwaPliku)));
       out.writeObject("Lista lokatorów");
       out.writeObject(kotek);
       out.close();
       ObjectInputStream in = new ObjectInputStream(
                                      new BufferedInputStream(
                                           new FileInputStream(nazwaPliku)));
       String nagłówek = (String) in.readObject();
       kotek = (Osoba) in.readObject();
       in.close();
       System.out.println(kotek);
   }
}
```


Przykład 1 (3)

Wynik:

wywołanie konstruktora klasy Adres
wywołanie konstruktora klasy Osoba
Osoba@1372a1a(Kotek, Zygmunt, Adres@ad3ba4(Warszawa, Alternatywy, 4, 9))
Osoba@1c39a2d(Kotek, Zygmunt, Adres@bf2d5e(Warszawa, Alternatywy, 4, 9))

Kontrola serializacji (1)

- Serializacja poufnych danych
- Pominięcie serializacji obiektów dla których serializacja nie ma sensu (np. wątki, strumienie)
- Serializacja obiektów nie implementujących interfejs *Serializable*

Kontrola serializacji (2)

- interfejs *Externalizable*
  - ***writeExternal(ObjectOutput)***
  - ***readExternal(ObjectInput)***
- *transient*

  np. private transient String password;
- przesłanianie metod serializujących:
  - **private void writeObject(ObjectOutputStream stream) throws IOException;**
  - **private void readObject(ObjectInputStream stream) throws IOException, ClassNotFoundException;**
  - **defaultWriteObject(), defaultReadObject()**

Przykład 2

```
class Osoba implements Serializable {
String nazwisko;
String imie;
transient Adres adresZameldowania;
Osoba(String nazwisko, String imie, Adres adresZameldowania) {
  this.nazwisko = nazwisko;
  this.imie = imie;
  this.adresZameldowania = adresZameldowania;
  System.out.println("wywołanie konstruktora klasy Osoba");
  }
public String toString() {
  String adrPamiec = super.toString();
  return adrPamiec+"(" + nazwisko + ", " + imie + ", " +
  adresZameldowania + ")";
 }
}
```

wywołanie konstruktora klasy Adres
wywołanie konstruktora klasy Osoba
Osoba1@1372a1a(Kotek, Zygmunt, Adres@ad3ba4(Warszawa, Alternatywy, 4, 9))
Osoba1@750159(Kotek, Zygmunt, **null**)

Wersja klasy

- A co gdy będzie deserializacja do nowszej wersji klasy?
- *static final long serialVersionUID*
  - Obliczony na podstawie treści definicji klasy
- **serialver**

```
class KlasaOZmieniajacejSieDefinicji implements Serializable {
static final long serialVersionUID = 2L;
String staryAtrybut = "jakas wartosc";
}
```

Serializacja do XMLa

Pakiet java.beans:

- *XMLEncoder zamiast ObjectOutputStream*
- *XMLDecoder zamiast ObjectInputStream*
- *Klasa zgodna z wymaganiami Java Beans (metody getXxx, setXxx, itp.)*

Podsumowanie

- Serializacja całej struktury powiązań obiektów
- Deserializacja nie wywołuje konstruktorów!
- Klasa deserializowana musi być dostępna

Pytania

1. Na czym polega serializacja i deserializacja obiektów?
2. Jakie warunki musi spełniać klasa aby możliwa była jej serializacja?
3. Dla jakich obiektów serializacja nie ma sensu?

Koniec