JAVA Persistence and Serialization (SLR 201)

Patrick Bellot & Ada Diaconescu

Télécom ParisTech

What is persistence ?

- It may often happen that data have to be saved and retrieved from one run of the application to another run of the application.
- This is true for all programming languages.
- You may save the data in a file with your own coding and decoding algorithm.
- However…

Serialization

- Most of JAVA data can be serialized.
- That means that objects can be translated into a sequence of bytes and saved somewhere.
- Given this sequence of bytes and the class of the object, JAVA is able to rebuild the object.

Serialization

- Object serialization is a process for saving an object's state to a sequence of bytes, as well as the process for rebuilding those bytes into a live object.
- The Java Serialization API provides a standard mechanism for developers to handle object serialization.
- The API is small and easy to use, provided the classes and methods are understood.

Serialization

- **The goals of serialization are:**
  - To be able to save an object in a file and to reread it.
  - To be able to transfer an object from one running program instance to another running program instance.
  - To support JAVA Remote Methods Invocation (RMI).

Serialization

- What can be serialized ?
- All ordinary data and data structures can be serialized.
- Special objects such as a network `Socket` or a `File` cannot be serialized. It would not make sense to serialize this kind of data.

Serialization

- To be serializable, a class must implement the `Serializable` interface:

```
import java.io.* ;

public class MyData implements Serializable
{
   …

   public static long serialVersionUID = 201509151636L ;
}
```

Serialization

- A serializable object may contain data that are not serializable. These data will *not* be serialized and must be tagged with the `transient` keyword:

```
import java.io.* ;

public class MyData implements Serializable
{
   …
   transient Thread thread ;
   …
}
```

Serialization : saving an object.

```
// Creating the serializable data
MyData data = new MyData(…)

// Opening an output stream
FileOutputStream  fout = new FileOutputStream("mydata.ser") ;
ObjectOutputStream out = new ObjectOutputStream(fout) ;

// Writing the serialized data
out.writeObject(data) ;

// Closing the output stream
out.close() ;
```

Serialization : saving an object.

- **The output stream can be :**
  - A file as in our example.
  - An array of bytes.
  - A socket output stream.

Serialization : saving an object.

```
// Declaring a variable
MyData data ;

// Opening the input stream
FileInputStream fin = new FileInputStream(“mydata.ser”) ;
ObjectInputStream in = new ObjectInputStream(fin) ;

// Reading the object
data = (MyData)in.readObject() ;

// Closing the stream
in.close() ;
```

Serialization and Exception

- **The use of methods `writeObject(Object obj)` and `readObject()` may raise exceptions if:**

  - The object to be written is *not* serializable.

  - The class used to cast the read object is *not* the good one.

Serialization

- Using serialization, you can save and retrieve your data ensuring the persistence of your data between two runs of your application.

- If input and output streams are sockets, your can transfer objects from one program to another (mobile data).

- JAVA/RMI uses serialization to implement *remote method invocation*.

- Serialized objects can be stored in data bases too.

Serialization problems

- A delicate problem occurs when an object is written several times in the output stream.
- By default, the `ObjectOutputStream out` maintains a reference to the written object.
- The second time, the object will *not* be written unless you call:

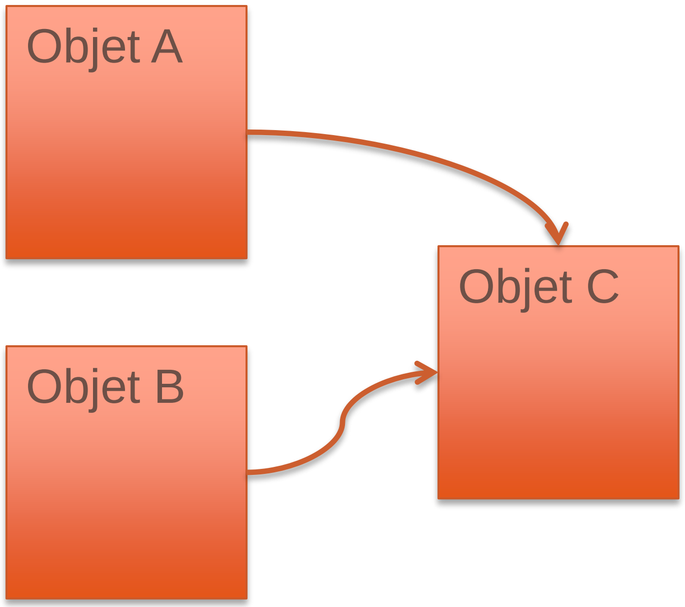
  ```
  out.reset() ;
  ```

  to releases the cache of written objects references.

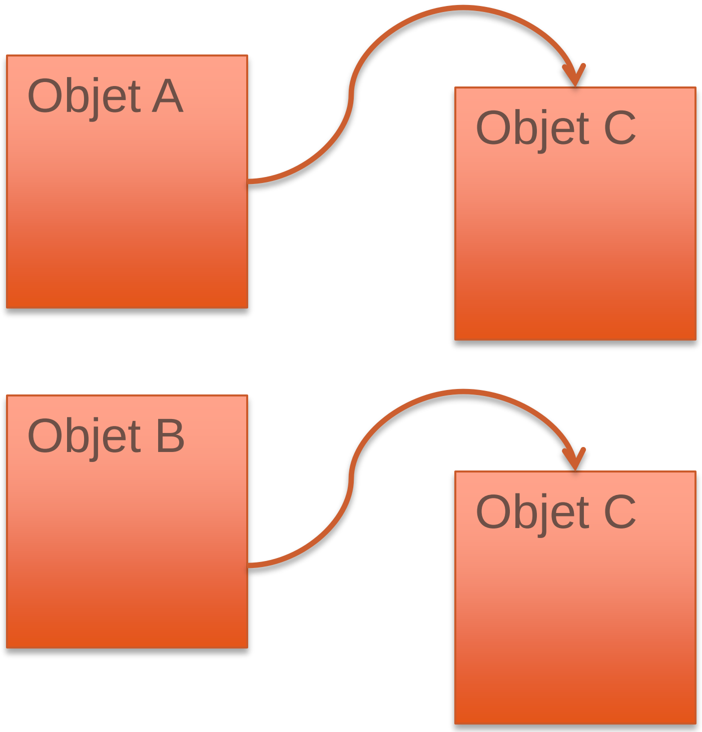
Serialization problems

- Also due to the cache of written objects, these objects will *not* be collected by the garbage collector.
- The solution is simply to close the stream.

Serialization problems

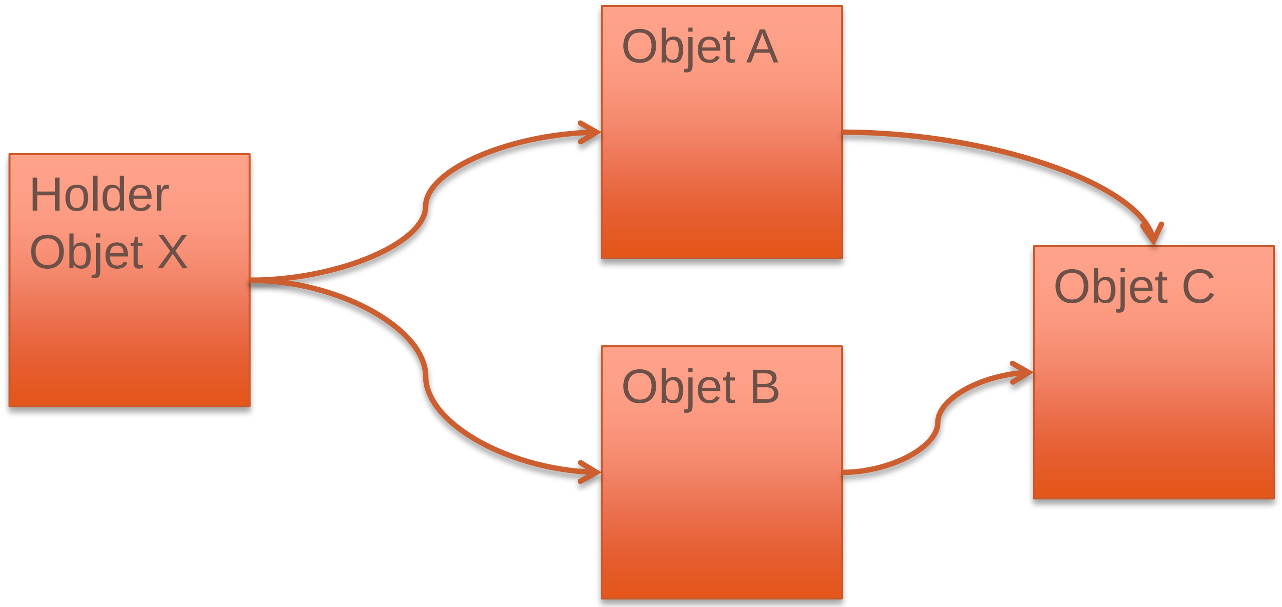


Save A
then
Save B



Load A
then
Load B

Serialization problems



Save X
then
Load X

Serialization problems

- A serializable class may evolve.
- If you add or remove attributes in the class declaration, you may not be able to reload objects.
- JAVA assumes that you maintain a serial number for serializable classes:

```
static final long serialVersionUID = 200812042336L ;
```

- If serial numbers of the class and the serialized object do not agree, it raises an exception.

Persistence

- We have seen that we can manage persistence of data using serialization.

- It is a convenient and easy (but inefficient) way to do things.

- However, JAVA also proposes a persistence API named Java Persistence API (JPA).

JAVA Persistence API (JPA)

- **JPA is linked to Enterprise JavaBeans (EJB 3.0) which are standardized software components.**

- **Java Persistence API:**
  - simplifies the entity persistence model,
  - stores objects in a relational database so that they can be accessed at a later time,
  - ensures the continued existence of an entity's state even after the application that uses it ends.

JAVA Persistence API (JPA)

- **Preparing a class for persistence:**

```
import java.io.* ;
import javax.persistence.* ;

@Entity
public class Student implements Serializable
{
   @id
   private int number ; // primary key

   private String first_name ;
   private String last_name ;
}
```

- **There are many types of annotations…**

JAVA Persistence API (JPA)

- **Configuration file: `persistence.xml`**

```
<?xml version="1.0"  encoding="UTF-8"?>
<persistence  xmlns="http://java.sun.com/xml/ns/persistence"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              version="1.0"
              xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
              http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">

    <persistence-unit  name= "StudentDB">
         <provider>oracle.toplink.essentials.PersistenceProvider</provider>
         <class>MonPackage.Student</class>
    </persistence-unit>

</persistence>
```

JAVA Persistence API (JPA)

- **Create the corresponding table in `StudentDB`:**

```
CREATE TABLE Student
{
   id           INTEGER    PRIMARY KEY ;
   first_name CHAR(256)  ;
   last_name  CHAR(256)  ;
}
```

JAVA Persistence API (JPA)

- **Add new mapping declarations:**

```
@Entity
public class Student implements Serializable
{
   @id
   @column(name=“id” table=“Student”)
   private int number ; // primary key

   @column(name=“first_name” table=“Student”)
   private String first_name ;

   @column(name=“last_name” table=“Student”)
   private String last_name ;
}
```

JAVA Persistence API (JPA)

- Add getter and setter:

```
@Entity
public class Student implements Serializable
{
  private String last_name ;

  public String getLast_name()
  {
        return last_name ;
  }

  public void setLast_name(String last_name)
  {
        this.last_name = last_name ;
  }
}
```

JAVA Persistence API (JPA)

- Persistent objects are managed by an `EntityManager`.

```
EntityManagerFactory emf
       = Persistence.createEntityManagerFactory(“StudentDB”) ;
EntityManager em = emf.createEntityManager() ;
EntityTransaction et = em.getTransaction() ;
et.begin() ;

Student s = new Student(12345,”Elton”,”John”) ;
em.persist(s) ;

et.commit() ;
em.close() ;
emf.close() ;
```

JAVA Persistence API (JPA)

- Finding persistent objects.

```
EntityManagerFactory emf
       = Persistence.createEntityManagerFactory("StudentDB") ;
EntityManager em = emf.createEntityManager() ;

// find by primary key, null if not found
Student s = em.find(Student.class,12345) ;

em.close() ;
emf.close() ;
```

JAVA Persistence API (JPA)

- Finding persistent objects.

```
EntityManagerFactory emf
       = Persistence.createEntityManagerFactory(“StudentDB”) ;
EntityManager em = emf.createEntityManager() ;

// find by primary key, raises EntityNotFoundException if not found
Student s = em.getReference(Student.class,12345) ;

em.close() ;
emf.close() ;
```

JAVA Persistence API (JPA)

- Finding persistent objects.

```
EntityManagerFactory emf
               = Persistence.createEntityManagerFactory(“StudentDB”)
  ;
EntityManager em = emf.createEntityManager() ;

Query query = em.createQuery("select s from Student p where … ") ;

Student s =(Student)query.getSingleResult() ;

List<Student> s =(List<Student>)query.getResultList() ;

em.close() ;
emf.close() ;
```

JPA Conclusions

- This is only a superficial view of JPA.
- JPA is a complex technology covering all aspects of data base technology.
- JPA is better used in Application Servers such as SUN GlassFish where everything is automated, including:
  - Declarations
  - Table creations
  - Class generations
  - ...