



**СБЕРБАНК ТЕХНОЛОГИИ**

# **Java Serialization**

- Что такое Serialization в Java
- Как работает сериализация
- Что такое версия класса
- Как сериализовать объект и восстановить его состояние

- *Сериализация* – это процесс сохранения состояния объекта в последовательность байт;
- *Десериализация* – это процесс восстановления объекта из этих байт.

## Примеров много

- Осуществление передачи данных между процессами
- Сохранение пользовательских настроек при перезапуске программы
- Создавать «глубокую» копию объекта
- ...

Неужели каждый раз нужно заново разрабатывать механизм сохранения и считывания состояния объекта?

- **interface** Serializable { }
- **interface** Externalizable **extends** java.io.Serializable {  
    **void** writeExternal(ObjectOutput out) **throws** IOException;  
    **void** readExternal(ObjectInput in)  
                                    **throws** IOException, ClassNotFoundException;  
}
- **class** ObjectOutputStream **extends** OutputStream ...
- **class** ObjectInputStream **extends** InputStream ...
- **transient** keyword
- «Магия»

Для встроенного механизма сериализации необходимо чтобы

1. Класс реализовывал интерфейс `java.io.Serializable`;
2. Были определены поля, участвующие в сериализации
  - Стандартно это все поля не помеченные ключевым словом `transient`
  - Можно явно указать список полей для сериализации

```
private static final ObjectOutputStream[] serialPersistentFields =  
    { new ObjectOutputStream("name", String.class) };
```
3. Был доступен конструктор без аргументов первого не сериализуемого родительского класса

```
class SerializableObject implements Serializable {  
    private Student[] students = {new Student("Name")};
```

```
    // ...  
}
```

```
class Student implements Serializable {  
    private String name;
```

```
    Student(String name) {  
        this.name = name;  
    }
```

```
    // ...  
}
```

```
void serialize(String filename, Student student) throws IOException {  
    try (FileOutputStream fos = new FileOutputStream(filename);  
        ObjectOutputStream out = new ObjectOutputStream(fos))  
    {  
        out.writeObject(student);  
    }  
}
```

```
Student deserialize(String filename) throws IOException,  
ClassNotFoundException {  
    try (FileInputStream fis = new FileInputStream(filename);  
        ObjectInputStream in = new ObjectInputStream(fis))  
    {  
        return (Student) in.readObject();  
    }  
}
```



1. запись метаданных о классе ассоциированном с объектом
2. рекурсивная запись описания **суперклассов**, до тех пор пока не будет достигнут *java.lang.Object*
3. запись фактических данных ассоциированных с экземпляром начинается с самого верхнего суперкласса

```
00000000 ac ed 00 05 73 72 00 0e 72 75 2e 73 62 74 2e 53
00000010 74 75 64 65 6e 74 a9 73 e0 6c ef d1 4b ba 02 00
00000020 01 4c 00 04 6e 61 6d 65 74 00 12 4c 6a 61 76 61
00000030 2f 6c 61 6e 67 2f 53 74 72 69 6e 67 3b 78 70 74
00000040 00 04 4e 61 6d 65
```

AC ED: STREAM\_MAGIC. Говорит о том, что используется протокол сериализации.

00 05: STREAM\_VERSION. Версия сериализации.

73: TC\_OBJECT. Обозначение нового объекта. 72: TC\_CLASSDESC. Обозначение нового класса.

00 0e: Длина имени класса. 72 75 2e 73 62 74 2e 53 74 75 64 65 6e 74: ru.sbt.Student.

a9 73 e0 6c ef d1 4b ba: serialVersionUID, идентификатор класса.

02: Различные флаги. Этот специфический флаг говорит о том, что объект сериализуемый.

00 01: Число полей в классе. 0x4c: Идентификатор типа – Строка 00 04 – длина имени поля

6e 61 6d 65: name, 74: Новая строка, 00 12: длина строки, Ljava/lang/String;

0x78: TC\_ENDBLOCKDATA, конец опционального блока данных для объекта.

0x70: TC\_NULL, обозначает то что больше нет суперклассов. мы достигли верха иерархии классов.

74 00 04 4e 61 6d 65: 4-х символьная строка «Name»

```
class User implements Serializable {  
    String login;  
    transient String password;  
  
    User(String login, String password) {  
        this.login = login;  
        this.password = password;  
    }  
    // ...  
}
```

```
ObjectInputStream objectInputStream = ...
```

```
user = (User) objectInputStream.readObject();
```

```
System.out.println(user.getPassword()); // ?
```

```
private void writeObject(ObjectOutputStream out) throws IOException {  
    out.defaultWriteObject();  
    out.writeLong(Calendar.getInstance().getTimeInMillis());  
}
```

```
private void readObject(ObjectInputStream in) throws IOException, ClassNotFoundException {  
    in.defaultReadObject();  
    long writeTime = in.readLong();  
    System.out.println(writeTime);  
}
```

```
class NotSerializableParent {  
    String superData;
```

```
    ...
```

```
}
```

```
class SerializableChild extends NotSerializableParent implements Serializable {
```

```
    String data;
```

```
    ...
```

```
}
```

```
class SerializableParent implements Serializable {  
    String department;  
    // ...  
}
```

```
class NotSerializableChild extends SerializableParent {  
    String securedData;  
  
    // ...  
}
```

```
class SerializableParent implements Serializable {  
    String department;  
    // ...  
}
```

```
class NotSerializableChild extends SerializableParent {  
    String securedData;  
  
    private void writeObject(ObjectOutputStream stream) throws IOException {  
        throw new NotSerializableException();  
    }  
    // ...  
}
```

? Сможем ли мы восстановить объект если после его сериализации

- изменится имя поля сериализованного уже в файл класса
- поменялся только тип
- всего лишь изменился модификатор доступа к полю
- Изменился как-либо метод
- JVM, сериализовавшая объект отличается от считывающей его JVM

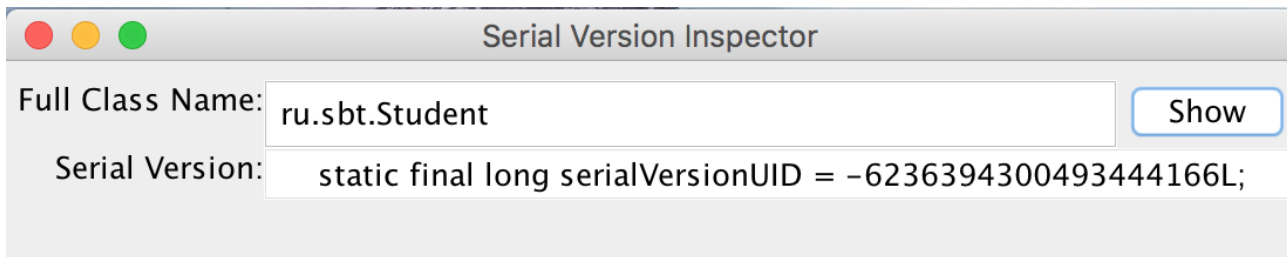


**private static final long *serialVersionUID* = 773530713475795375L;**

Если явно не указан в классе то он вычисляется на основе хэш SHA-1 используя

- Имя класса
- Имена интерфейсов
- Имена всех методов и полей
- ... см. спецификацию

Может быть рассчитан с помощью утилиты из JDK – **serialver**



Это изменения класса после сериализации его предыдущей версии, гарантирующие совместимость данных после десериализации в обновленный класс.

- Добавление новых полей
- Добавление и удаление классов в иерархию класса
- Добавление и удаление методов `writeObject/readObject`
- Реализация `java.io.Serializable` - аналогично добавлению нового класса
- Изменение модификаторов доступа к полям
- Изменение поля `static` -> `nonstatic` или `transient` -> `nontransient` – аналогично добавлению нового поля

```
package java.io;

interface Externalizable extends Serializable
{
    public void writeExternal(ObjectOutput out)
        throws IOException;

    public void readExternal(ObjectInput in)
        throws IOException, java.lang.ClassNotFoundException;
}
```

## **Externalizable** объект

- Реализовывает интерфейс `java.io.Externalizable`
- Отвечает за сериализуемыми данными своего класса и всех его родительских классов
- Имеет конструктор без аргументов

# EXTERNALIZABLE VS SERIALIZABLE

Externalizable	Serializable
Свои правила и свой механизм сериализации объектов	Предоставляет механизм сериализации и стандартный протокол «из коробки»
Наследуется от Serializable	Маркер-интерфейс
Ответственность сохранения состояния родительского класса на реализующем классе	Поддерживается сериализации состояния всех сериализуемых родительских классов
writeExternal & readExternal заменяют магические writeObject & readObject	Расширение стандартного механизма с помощью writeObject\readObject
Объект создается публичным конструктором без аргументов	Рефлексивное наполнение без вызова конструктора
Требование иметь конструктор	Конструктор без аргументов не обязателен

```
class GraphNode implements Serializable {  
    GraphNode nextNode;  
    //...  
}
```

```
GraphNode cyclicGraph = new GraphNode();  
cyclicGraph.nextNode = cyclicGraph;
```

```
SerializableUtils.serialize(filename, cyclicGraph);  
GraphNode deserialize = (GraphNode) SerializableUtils.deserialize(filename);
```

```
System.out.println(deserialize == deserialize.nextNode);
```

- Java Serialization – простой механизм сериализации данных
- Встроенная поддержка безопасной типизации объектов и их свойств в сериализованном виде
- Обеспечена возможность расширения протокола сериализации для поддержки маршалинга и демаршалинга удаленных объектов при RPC вызовах
- Обеспечена возможность расширения протокола сериализации для поддержки персистентности объектов
- Особенности сериализации добавляются только в классах, хранящих эти особенности
- Объекты могут сами определять свой протокол сериализации

- <https://m.habrahabr.ru/post/60317/>
- <http://www.javaworld.com/article/2076120/java-se/flatten-your-objects.html>
- <http://www.javaworld.com/article/2072752/the-java-serialization-algorithm-revealed.html>
- <https://docs.oracle.com/javase/8/docs/technotes/guides/serialization/relnotes.html>
- <http://javapapers.com/core-java/externalizable-vs-serializable/>
- <http://docs.oracle.com/javase/6/docs/platform/serialization/spec/serialTOC.html>