Самарский государственный аэрокосмический университет
имени академика С.П. Королёва

Объектно-ориентированное программирование

Механизмы ввода и вывода информации

Понятие сериализации

Занятие 5

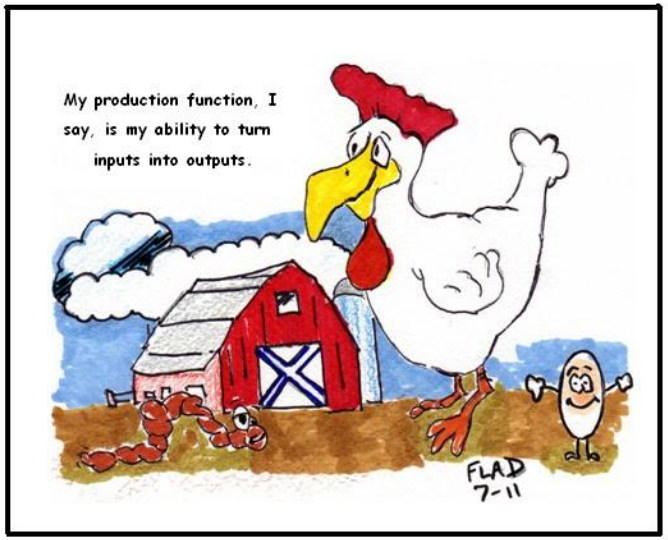
Самара
2015

План лекции

- Потоки данных и их виды
- Иерархия и разновидности потоков данных
- Понятие сериализации
- Особенности сериализации и десериализации

Традиционно, проблемы!

- Система ввода/вывода не должна зависеть от платформы!



- Применяется модель потоков данных:
  - упорядоченная последовательность данных,
  - которой соответствует определенный источник (потоки ввода) или получатель (потоки вывода)

Разновидности потоков

	Ввод	Вывод
Байтовые	Потоки ввода	Потоки вывода
Символьные	Потоки чтения	Потоки записи

Структура пакета java.io

- Типы общего назначения
- Классы разновидностей потоков
- Специализированные классы и интерфейсы для ввода и вывода значений простых типов
- Классы и интерфейсы работы с файлами
- Классы и интерфейсы механизма сериализации

Разновидности потоков

	Ввод	Вывод
Байтовые	Потоки ввода `InputStream`	Потоки вывода `OutputStream`
Символьные	Потоки чтения `Reader`	Потоки записи `Writer`

Класс InputStream

- `abstract int read()`
  `throws IOException`
- `int read(byte[] b, int off, int len)`
  `throws IOException`
- `int read(byte[] b)`
  `throws IOException`
- `long skip(long n)`
  `throws IOException`
- `int available()`
  `throws IOException`
- `void close()`
  `throws IOException`

Класс OutputStream

- `abstract void write(int b)`
  `throws IOException`
- `void write(byte[] b, int off, int len)`
  `throws IOException`
- `void write(byte[] b)`
  `throws IOException`
- `void flush()`
  `throws IOException`
- `void close()`
  `throws IOException`

Класс Reader

- `int read()`
  `throws IOException`
- `abstract int read(char[] b, int off, int len)`
  `throws IOException`
- `int read(char[] b)`
  `throws IOException`
- `long skip(long n)`
  `throws IOException`
- `boolean ready()`
  `throws IOException`
- `abstract void close()`
  `throws IOException`

Класс Writer

- `void write(int ch) throws IOException`
- `abstract void write(char[] b, int off, int len) throws IOException`
- `void write(char[] b) throws IOException`
- `void write(String str, int off, int len) throws IOException`
- `void write(String str) throws IOException`
- `abstract void flush() throws IOException`
- `abstract void close() throws IOException`

Занятная особенность

- Уже знакомые потоки:

`System.out`	`PrintStream`
`System.in`	`InputStream`
`System.err`	`PrintStream`

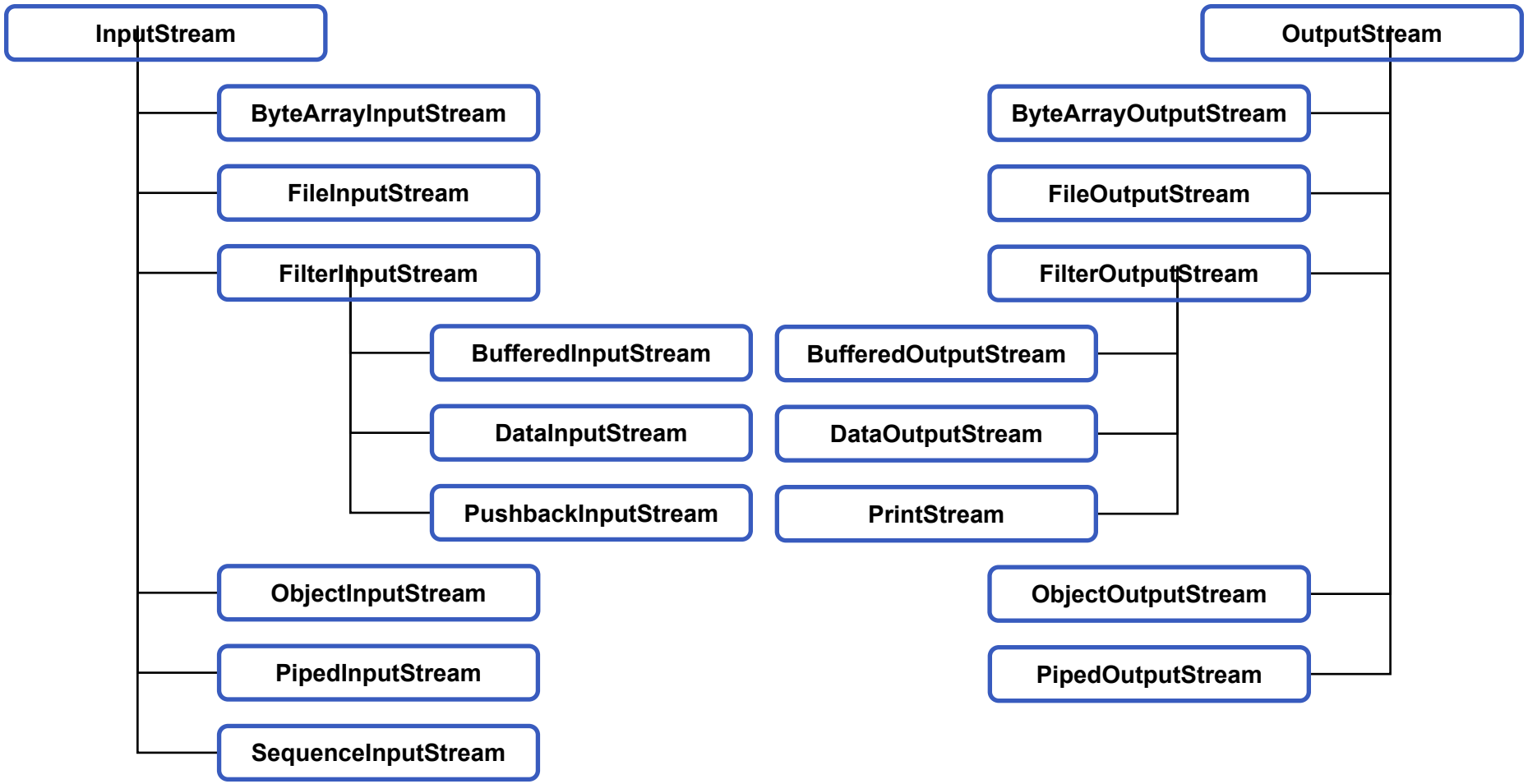
- Какого они типа?

- Байтового!!! (для совместимости с версиями Java 1.0 и 1.1)

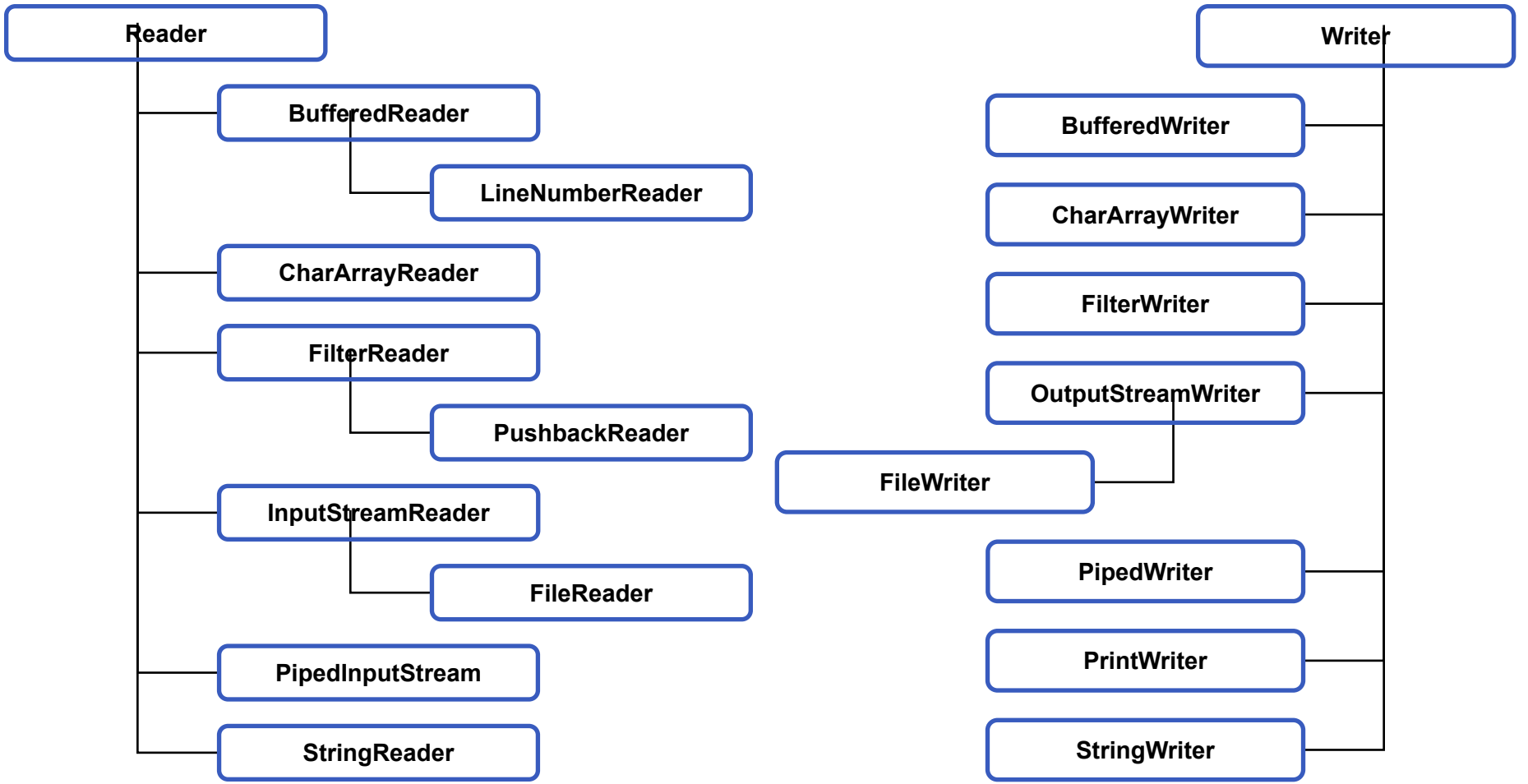
Классы потоков ввода и вывода

- Образуют 4 иерархии, в основе которых лежат базовые абстрактные классы
- Имя любого дочернего класса в иерархии имеет суффикс, совпадающий с именем корневого класса
- По сути делятся на 2 вида:
  - «Реальные» потоки: источник (получатель) данных реален
  - Потоки-обертки: источником (получателем) данных является другой поток

Иерархия байтовых потоков



Иерархия символьных потоков



Классы-трансляторы

- Позволяют читать из байтового как из символьного и записывать в байтовый поток как в символьный (с учетом кодировки)

- **InputStreamReader**
  - `InputStreamReader(InputStream in)`
  - `InputStreamReader(InputStream in, String encoding) throws UnsupportedEncodingException`

- **OutputStreamWriter**
  - `OutputStreamWriter(OutputStream out)`
  - `OutputStreamWriter(OutputStream out, String encoding) throws UnsupportedEncodingException`

Группа потоков Filter

`FilterInputStream`, `FilterReader`

`FilterOutputStream`, `FiltrerWriter`

- Обертки, позволяют объединять потоки в цепочки для получения сложных потоков, обладающих расширенным набором функций
- Обладают дополнительными защищенными конструкторами
  `protected FilterInputStream(InputStream in)`
- В наследниках обычно переопределяются методы чтения/записи с добавлением новой функциональности

Группа потоков Buffered

`BufferedInputStream`, `BufferedReader`

`BufferedOutputStream`, `BufferedWriter`

- Обертки, осуществляют буферизацию данных на программном уровне
- Размер буфера можно задать в конструкторе
- Символьные версии имеют методы чтения и записи строк

Группа потоков Piped

`PipedInputStream`, `PipedReader`

`PipedOutputStream`, `PipedWriter`

- Используются в виде пар ввода-вывода
- Данные, переданные в поток вывода, служат источником для потока ввода
- Например, реализуют механизм обмена данными между нитями
- Поток-пара задается параметром конструктора либо с помощью метода `connect()`

Группа байтовых потоков ByteArray

`ByteArrayInputStream`, `ByteArrayOutputStream`

- В качестве источника и получателя данных используются массивы байт
- В потоке вывода размер буфера может меняться динамически
- В потоке вывода существуют методы преобразования:
  - к массиву байт
    `byte[] toByteArray()`
  - к строке
    `String toString()`
  - вывода в другой поток
    `void writeTo(OutputStream out)`

Группы символьных потоков CharArray и String

- `CharArrayReader` и `CharArrayWriter` аналогичны `ByteArrayInputStream` и `ByteArrayOutputStream`, но оперируют с массивом символов

- `StringReader` и `StringWriter` имеют аналогичную функциональность, позволяют считывать символы из строки и записывать данные в строковый буфер

Группа потоков Print

- Обертки `PrintStream` и `PrintWriter` содержат методы, упрощающие задачу вывода данных простых типов в текстовом виде
- Методы `print()` и `println()` не выбрасывают исключений
- `System.out` и `System.err` – единственные потоки `PrintStream`

Класс StreamTokenizer

- Не является потоком чтения, но позволяет обрабатывать информацию из них
- Содержит методы лексической обработки текста
- Ряд методов предназначен для настройки работы анализатора
- Метод `nextToken()` производит обработку очередной лексемы, после чего:
  - Поле `ttype` содержит константу типа лексемы
  - Поля `nval` и `sval` содержат числовое и строковое представление лексемы

Группа байтовых потоков Data

- Интерфейсы `DataInput` и `DataOutput` содержат объявления методов ввода и вывода значений простых типов
  `void writeLong(long v)`, `void writeFloat(float v)`
  `boolean readBoolean()`, `String readUTF()`
- Обертки `DataInputStream` и `DataOutputStream`, соответственно, реализуют эти интерфейсы
- Класс `RandomAccessFile` реализует оба интерфейса Data и позволяет работать с файлами в режиме произвольного доступа

Класс File

- Инкапсулирует платформенно-независимые методы работы с файлами и директориями:
  - создание
  - проверка атрибутов
  - удаление
  - переименование
- Позволяет создавать временные файлы, удаляемые при завершении работы программы
- API класса изучите самостоятельно

Группа потоков File

`FileInputStream`, `FileReader`
`FileOutputStream`, `FileWriter`

- Позволяют трактовать файл как поток, предназначенный для ввода и вывода данных
- Связаны с исключениями **`FileNotFoundException`** и **`SecurityException`**
- Конструкторы могут получать параметры:
  - Строку **`String`**, задающую имя файла
  - Объект класса **`File`**
  - Объект **`FileDescriptor`** (возвращается методом **`getFD()`** байтовых потоков)

Пример записи в текстовый файл

```
import java.io.*;

public class TextWrite {
  public static void main(String[] args) {
    int[] values = {1, 2, 3, 4, 5};
    try {
      PrintWriter out = new PrintWriter(new
        BufferedWriter(new FileWriter("out.txt")));
      for (int i = 0; i < values.length; i++) {
        out.println(values[i]);
      }
      out.close();
    }
    catch(IOException e) {
      System.out.println("Some error occurred!");
    }
  }
}
```

out.txt
Текстовая форма

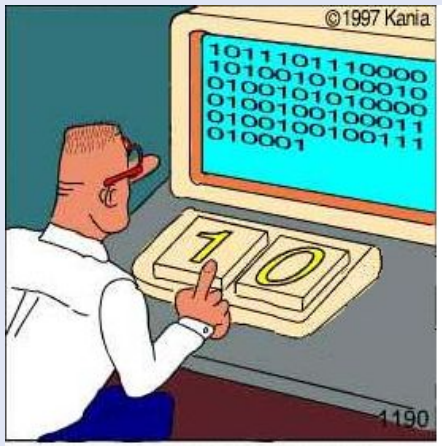
```
1
2
3
4
5
```

out.txt
Байтовая форма

```
31 0D 0A
32 0D 0A
33 0D 0A
34 0D 0A
35 0D 0A
```

Пример записи в байтовый файл

```
import java.io.*;

public class ByteWrite {
  public static void main(String[] args) {
    int[] values = {1, 2, 3, 4, 5};
    try {
      DataOutputStream out = new
DataOutputStream(new FileOutputStream("out.bin"));
      for (int i = 0; i < values.length; i++) {
        out.writeInt(values[i]);
      }
      out.close();
    }
    catch(IOException e) {
      System.out.println(
    "Some error occurred!");
    }
  }
}
```



Real programmers code in binary.

out.bin
Текстовая форма

☺ ☻ ♥
♦ ♣

out.bin
Байтовая форма

```
00 00 00 01
00 00 00 02
00 00 00 03
00 00 00 04
00 00 00 05
```

Пример чтения из текстового файла и из консоли

```
import java.io.*;

public class TextRead {
  public static void main(String[] args) {
    int[] values = new int[5];
    try {
      BufferedReader in = new BufferedReader(new
        FileReader("in.txt")); //InputStreamReader(System.in));
      for (int i = 0; i < values.length; i++) {
        values[i] = Integer.parseInt(in.readLine());
      }
      in.close();
    }
    catch(IOException e) {
      System.out.println("Some error occurred!");
    }
  }
}
```

Пример чтения из байтового файла

```
import java.io.*;

public class ByteRead {
  public static void main(String[] args) {
    int[] values = new int[5];
    try {
      DataInputStream in = new DataInputStream(new
        FileInputStream("out.bin"));
      for (int i = 0; i < values.length; i++) {
        values[i] = in.readInt();
      }
      in.close();
    }
    catch(IOException e) {
      System.out.println("Some error occurred!");
    }
  }
}
```

Форматированные ввод и вывод (Java 5)

- **`java.util.Formatter`**
  Обеспечивает преобразования формата, позволяющие выводить числа, строки, время и даты в практически любом нужном вам формате

- **`java.util.Scanner`**
  Позволяет использовать форматированный ввод и преобразовывать значения к нужным типам

java.util.Formatter
Конструкторы

- Имеет множество конструкторов, позволяющих задать следующие параметры (либо, если они не заданы, использовать значения по умолчанию)
- Объект вывода
  - `Appendable a`
  - `File file`
  - `String fileName`
  - `OutputStream os`
  - `PrintStream ps`
  - по умолчанию – без автоматического вывода
- Кодовая таблица
  - `String charSet`
  - по умолчанию – текущая таблица
- Параметры локализации
  - `Locale locale`
  - по умолчанию – текущие параметры

java.util.Formatter
Важные методы

- `Formatter format(String fmtString, Object ... args)`

  Форматирует указанные аргументы в соответствии со строкой форматирования

- `Formatter format(Locale loc, String fmtString, Object ... args)`

  Форматирует указанные аргументы в соответствии со строкой форматирования и указанной локализацией

java.util.Formatter
Важные методы

- `IOException ioException()`
  Возвращает объект исключения, генерируемый объектом-приемником, иначе `null`
- `Appendable out()`
  Возвращает ссылку на объект-приемник выходных данных
- `Locale locale()`
  Возвращает ссылку на объект локализации

java.util.Formatter
Важные методы

- `String toString()`
  Возвращает объект типа `String`, содержащий отформатированный вывод
- `void flush()`
  Переносит информацию из буфера форматирования
- `void close()`
  Закрывает объект форматировщика, освобождает ресурсы

Строка форматирования

- Строка форматирования состоит из:
  - простых символов
    Просто копируются в вывод
  - спецификаторов формата
    Определяют способ отображения аргументов
- Спецификатор формата:
  - знак процента (%)
  - преобразующий спецификатор формата

```
Formatter fmt = new Formatter();
fmt.format("Formatting %s is easy! %d %f",
           "with Java", 15, 12.3);
```

Преобразующие спецификаторы формата

Спецификатор	Преобразование
%a %A	Шестнадцатеричное значение с плавающей точкой
%b %B	Булево значение
%c %C	Символьное значение
%d	Десятичное целое значение
%h %H	Хэш-код аргумента
%e %E	Экспоненциальное представление аргумента
%f	Десятичное значение с плавающей точкой
%g %G	Выбирает более короткое представление из двух: %f и %e
%o	Восьмеричное целое значение аргумента
%n	Символ новой строки
%s %S	Строковое представление аргумента
%t %T	Время и дата
%x %X	Шестнадцатеричное целое значение аргумента
%%	Знак процента

Возможности форматирования

- Порядковый номер аргумента
  Позволяет использовать не текущий аргумент, а заданный

```
fmt.format("%3$d %2$d %1$d", 1, 2, 3); // 3 2 1
```

- Относительный номер
  Позволяет несколько раз вывести одно и то же значение без явной нумерации

```
Calendar c = Calendar.getInstance();
fmt.format("Today is day %te of %<tB, %<tY", c);
// Today is day 4 of December, 2006
```

Возможности форматирования

- Управление регистром вывода

```
fmt.format("Some %s", "String"); //Some String
fmt.format("Some %S", "String"); //Some STRING
```

- Сложное форматирование времени и даты

```
fmt.format("Now is %tH:%<tS of %<td.%<tm.%<ty \n", c);
fmt.format("Now is %tl:%<tS%<tp of %<te %<tB %<tY \n", c);
// Now is 20:03 of 04.10.09
// Now is 8:03pm of 4 October 2009
```

- Задание минимальной ширины поля

```
fmt.format("%3s %3s %3s %3s ", "1", "22", "333", "4444");
//   1  22 333 4444
```

Возможности форматирования

- Задание точности вывода для вещественных значений

```
fmt.format("%10.2f %10.8f", Math.PI, Math.PI);
//       3.14 3.14159265
```

- Выравнивание вывода

```
fmt.format("%-10.2f|%10.2f", Math.PI, Math.PI);
// 3.14      |      3.14
```

- Разделение групп цифр и т.д.

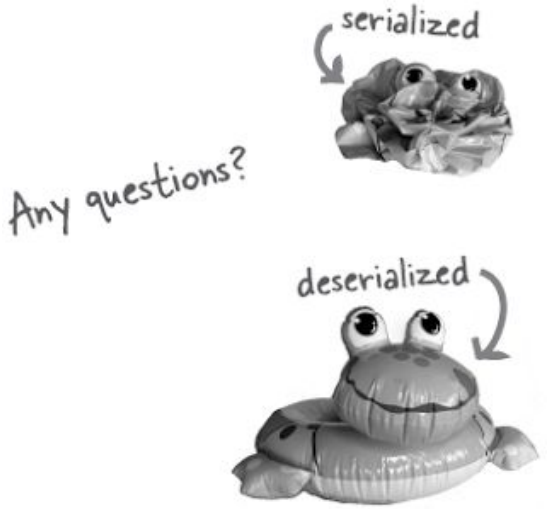
```
fmt.format("%,d", Integer.MAX_VALUE);
// 2,147,483,647
```

Метод printf()

- Использует автоматически создаваемый экземпляр класса `Formatter`
- Объявлен в классах:
  - `java.io.PrintWriter`
  - `java.io.PrintStream`
- Имеет такие же параметры, что и метод `Formatter.format()`

Сериализация объектов

- **Сериализация** – преобразование состояния объекта в поток байтов
- **Десериализация** – восстановление состояния объекта из данных потока
- Не все объекты могут быть сериализованы
- Класс должен быть подготовлен к сериализации



Группа байтовых потоков Object

- Класс `ObjectOutputStream` реализует сериализацию
- Класс `ObjectInputStream` реализует десериализацию
- Классы позволяют выводить и вводить графы объектов с сохранением структуры
- Результатом десериализации является объект, равнозначный исходному

Пример сериализации в файл

```
import java.io.*;

public class SerializationWrite {
  public static void main(String[] args) {
    int[] values = {1, 2, 3, 4, 5};
    try {
      ObjectOutputStream out = new
        ObjectOutputStream(new
        FileOutputStream("out.bin"));
      out.writeObject(values);
      out.close();
    }
    catch(IOException e) {
      System.out.println("Some error occurred!");
    }
  }
}
```

out.bin
Текстовая форма

```
┐нur☻
[IMє`&vк__☻  xp
♣   ☺   ☻   ♥
♦   ♣
```

out.bin
Байтовая форма

```
AC ED 00 05 75 72
00 02 5B 49 4D BA
60 26 76 EA B2 A5
02 00 00 78 70 00
00 00 05 00 00 00
01 00 00 00 02 00
00 00 03 00 00 00
04 00 00 00 05
```

Сериализация в Javahttp://habrahabr.ru/post/60317/

Пример десериализации из файла

```
import java.io.*;
public class SerializationRead {
  public static void main(String[] args) {
    int[] values;
    try {
      ObjectInputStream in = new ObjectInputStream(new
        FileInputStream("out.bin"));
      values = (int[])in.readObject();
      in.close();
    }
    catch(IOException e) {
      System.out.println("Some error occurred!");
    }
    catch(ClassNotFoundException e) {
      System.out.println("Wrong object type");
    }
  }
}
```

Подготовка классов к сериализации

- Должен реализовываться интерфейс-маркер `java.io.Serializable`
- Все сериализуемые поля должны иметь сериализуемый тип
- Родительский класс должен иметь конструктор по умолчанию (без параметров) или быть подготовленным к сериализации
- Сериализуются поля объекта, не обозначенные как `transient` или `static`

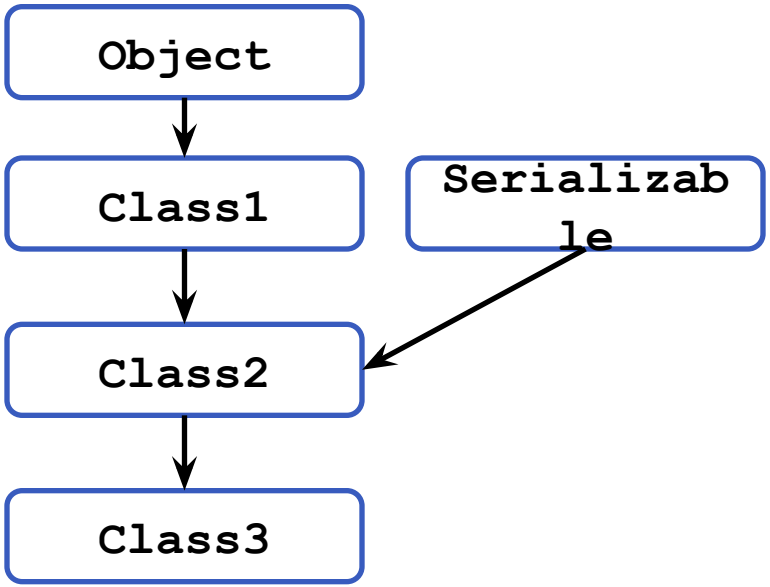
Порядок сериализации и десериализации

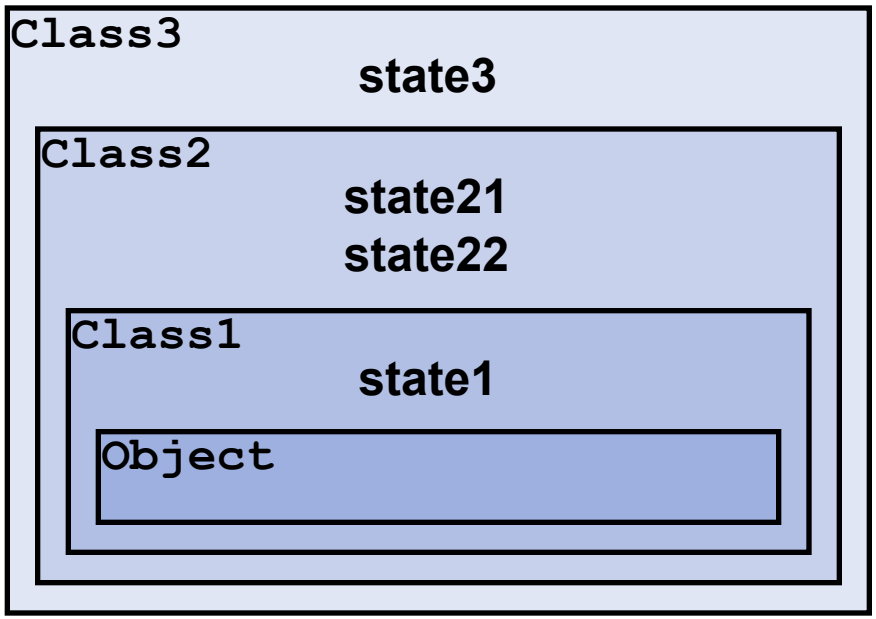
- В нисходящем порядке по древовидной иерархии типов: от первого сериализуемого класса до частного типа
- Объекты, на которые ссылаются поля, сериализуются в порядке обнаружения
- Перед десериализацией выполняется загрузка участвующих классов (возможен выброс исключения `ClassNotFoundException`)

Пример иерархии классов

```
class Class1 extends Object {
  private int state1 = 1;
}

class Class2 extends Class1 implements java.io.Serializable {
  protected int state21;
  private int state22;

  public Class2(int s1, int s2) {
    state21 = s1 + 15;
    state22 = s2 - 1;
  }
}

class Class3 extends Class2 {
  public int state3 = 3;
}
```

Порядок сериализации

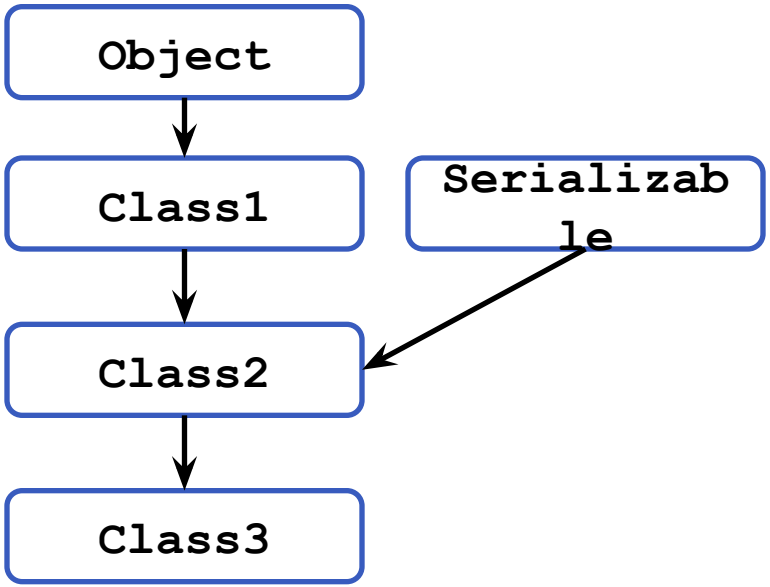


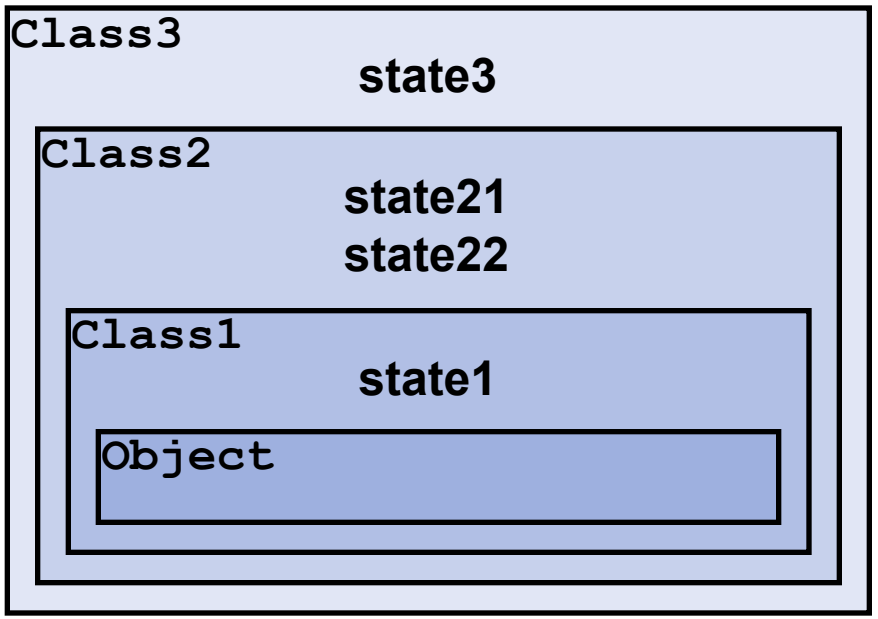


Сериализованное состояние объекта класса Class3

Class3: (Class2) state21 state22 (Class3) state3

Порядок десериализации





Сериализованное состояние объекта класса Class3

Class3: (Class2) state21 state22 (Class3) state3

Настройка сериализации

- Для изменения работы механизма сериализации на уровне вашего класса в самом классе надо описать методы:
  - реализация сериализации
    ```
    private void writeObject(ObjectOutputStream out) throws
    IOException
    ```
  - реализация десериализации
    ```
    private void readObject(ObjectInputStream in) throws
    IOException, ClassNotFoundException
    ```
- Уровень доступа методов позволяет им независимо существовать в различных классах в иерархии наследования
- Можно не переписывать чтение/запись полностью, а лишь изменить порядок записи полей и их формат (см. методы `ObjectOutputStream.writeFields()` и `ObjectInputStream.readFields()`)

Контроль версий

- Каждый класс имеет уникальный идентификатор номера версии – 64 битовое значение `long`
- По умолчанию значение рассчитывается как функция от кода класса (включая методы)
- Несовпадение версий при десериализации объекта выбрасывает исключение `InvalidClassException`
- Проблему можно обойти, явно введя в класс поле

```
private static final long
        serialVersionUID = ...;
```

Интерфейс Externalizable

- «Ручная» сериализация:
  - реализация сериализации
    ```
    public void writeExternal(ObjectOutputStream out) throws
    IOException
    ```
  - реализация десериализации
    ```
    private void readExternal(ObjectInputStream in) throws
    IOException, ClassNotFoundException
    ```
- Требует наличия конструктора по умолчанию у класса.
  JVM сначала вызывает конструктор без параметров, и только потом на уже созданном объекте вызывает метод readExternal
- Выигрыш в производительности при грамотной реализации
- Нарушение целостности графа

Спасибо за внимание!

Дополнительные источники

- Арнолд, К. Язык программирования Java [Текст] / Кен Арнолд, Джеймс Гослинг, Дэвид Холмс. – М. : Издательский дом «Вильямс», 2001. – 624 с.
- Вязовик, Н.А. Программирование на Java. Курс лекций [Текст] / Н.А. Вязовик. – М. : Интернет-университет информационных технологий, 2003. – 592 с.
- Хорстманн, К. Java 2. Библиотека профессионала. Том 1. Основы [Текст] / Кей Хорстманн, Гари Корнелл. – М. : Издательский дом «Вильямс», 2010 г. – 816 с.
- Эккель, Б. Философия Java [Текст] / Брюс Эккель. – СПб. : Питер, 2011. – 640 с.
- JavaSE at a Glance [Электронный ресурс]. – Режим доступа: http://www.oracle.com/technetwork/java/javase/overview/index.html, дата доступа: 21.10.2011.
- JavaSE APIs & Documentation [Электронный ресурс]. – Режим доступа: http://www.oracle.com/technetwork/java/javase/documentation/api-jsp-136079.html, дата доступа: 21.10.2011.