# 5. Ввод-вывод

ИТМО ВТ

# Пакет java.io

- Байтовые и символьные потоки данных (I/O streams)
  - Поток — последовательность данных (байтов, символов, примитивных типов, объектов)
  - Входной поток — для чтения данных из источника
  - Выходной поток — для записи данных в приемник
- Старый интерфейс работы с файлами

# Потоки ввода-вывода

- Байтовые и символьные
- Входные и выходные
- Базовые абстрактные классы для потоков

| | Байтовые | Символьные |
|---|---|---|
| Входные | `java.io.InputStream` | `java.io.Reader` |
| Выходные | `java.io.OutputStream` | `java.io.Writer` |

# java.io.InputStream

- абстрактный метод int read()
  - возвращает байт от 0 до 255, если доступен
  - если недоступен — блокируется и ждет
  - если поток завершился — возвращает -1
  - при ошибке — IOException
- реализованные методы:
  - int read(byte[] buffer, int len, int offset)
  - int read(byte[] buffer)
  - int available()
  - long skip(long n)
  - boolean markSupported()
  - void mark(int limit)
  - void reset()

# java.io.OutputStream

- абстрактный метод void write(int b)
  - записывает байт
  - при ошибке — IOException
- реализованные методы:
  - void write(byte[] buffer, int len, int offset)
  - void write(byte[] buffer)
  - void flush()
  - void close()

# java.io.Reader

- абстрактный метод int read(char[] buffer, int len, int offset)
  - читает символы в массиа, возвращает число прочитанных (-1 при завершении)
  - если недоступен — блокируется и ждет
  - при ошибке — IOException
- abstract void close()
- реализованные методы:
  - int read()
  - int read(char[] buffer)
  - boolean ready()
  - int available()
  - long skip(long n)
  - boolean markSupported()

# java.io.Writer

- абстрактный метод void write(char[] buffer, int len, int offset)
  - записывает символы из массива
  - при ошибке — IOException
- abstract void flush()
- abstract void close()
- реализованные методы:
  - void write(byte[] buffer, int len, int offset)
  - void write(char[] buffer)
  - void write(String str, int len, int offset)
  - void write(String str)
  - Writer append(char c)
  - Writer append(CharSequence cs)

# Специализированные потоки

- Работают с определенным источником/приемником
- Обычно источник/приемник указывается в конструкторе
  - чтение/запись файла
    - File (InputStream, OutputStream, Reader, Writer)
  - чтение/запись массива
    - ByteArray (InputStream, OutputStream)
    - CharArray (Reader, Writer)
  - чтение/запись в конвейер (взаимодействие потоков выполнения)
    - Piped (InputStream, OutputStream, Reader, Writer)
    - соединение потоков в конвейер
      - передача парного потока в конструктор
      - вызов метода connect()
  - чтение/запись строк (только символьные потоки)
    - String (Reader, Writer)

# Потоки-фильтры

- Filter (InputStream, OutpurStream, Reader, Writer)
  - принимают исходный поток как аргумент конструктора
  - получившийся поток-фильтр преобразует исходный поток
  - Buffered (InputStream, OutpurStream, Reader, Writer)
  - LineNumber (~~InputStream~~, Reader)
  - Print (Stream, Writer)
  - ZIP (InputStream, OutputStream)
  - Data (InputStream, OutpurStream)
- Object (InputStream, OutpurStream)
- байты ↔ символы
  - InputStreamReader (байты в символы)
  - OutputStreamWriter (символы в байты)

# PrintStream, PrintWriter

- методы print, println — один аргумент примитивного типа, строка или объект
- методы printf, format — много аргументов, первый из которых — форматная строка
- Синтаксис форматной строки
  - `%[индекс$][флаги][размер][.точность]формат`
  - `%%` - символ процента
  - `%n` – перевод строки
  - индекс – порядковый номер аргумента или < (предыдущий)
  - флаги – зависят от формата
  - размер – количество символов для представления числа
  - точность – цифры после запятой для дробных форматов
- `format("%c = %2$+9.7f", 'π', Math.PI);`
- π = +3,1415927

# Нечисловые форматы

- Общие - любой тип аргумента
- Заглавная буква формата - toUpperCase()):
- %b, %B - boolean
  - boolean, Boolean → значение
  - null → false
  - все остальное → true
- %h, %H — hashcode
  - null → null
  - все остальное — 16-ричный хэш-код
- %s, %S — строка
  - Formattable → formatTo()
  - все остальное → toString()
- %c, %C — символ Unicode

# Целые числа

- %d — десятичное целое
- %o — 8-ричное целое
- %x, %X — 16-ричное целое

*Флаги*

- '-' - левое выравнивание
- '#' - для недесятичных чисел — добавить 0 или 0x
- '+' - обязательно использовать знак
- ' ' - пробел на месте знака для положительных
- '0' — использовать ведущие нули
- ',' - использовать групповой разделитель для десятичных
- '(' - отрицательные — в скобках

# Числа с плавающей запятой

- %e, %E — научная нотация с мантиссой и порядком
- %f — десятичная нотация
- %g, %G — научная или десятичная (зависит от точности)
- %a, %A — 16-ричная с мантиссой и порядком

*Флаги* — такие же, как для целых

*Точность* — количество цифр после запятой

- для %g/G — общее количество значащих цифр и $10^{\text{точность}}$ — максимальное число, представляемое в десятичной форме (минимальное — $10^{-4}$). По умолчанию =6, то есть как десятичные дроби представляются числа от 0,0001 до 999999

# DataInputStream, DataOutputStream

- Преобразование примитивных типов и строк в последовательность байтов
- Последовательность при чтении должна быть такой же как и при записи
- Методы:
  - writeBoolean(boolean), writeByte(int), writeShort(int), writeInt(int), writeLong(long), writeFloat(float), writeDouble(double), writeChar(int), writeUTF(String)
  - bolean readBoolean(), byte readByte(), int readUnsignedByte() short readShort(), int readUnsignedShort(), int readInt(), long readLong(), float readFloat(), double readDouble(), char readUTF()

# Сериализация объектов

- Сериализация — запись объектов в виде потока байтов
- Классы ObjectOutputStream, ObjectInputStream
- Интерфейс Serializable - метка
- При записи объекты записываются с порядковым номером (serial number)
- Объект записывается в поток только один раз, потом используется ссылка на номер объекта
- При чтении одного и того же объекта из одного потока, он восстанавливается один раз
- При чтении объекта из двух потоков, объект восстанавливается дважды

# ObjectOutputStream, ObjectInputStream

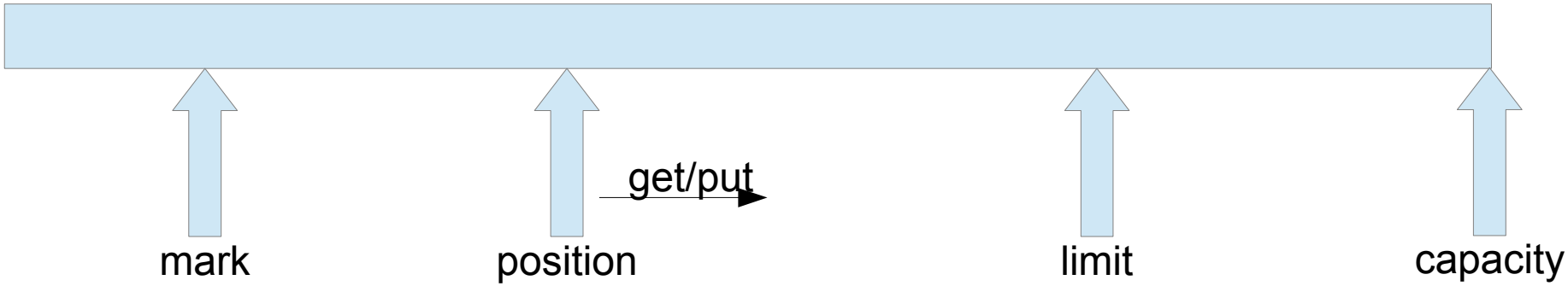
- Те же методы, что и у DataOutputStream, DataInputStream
- + writeObject(Object)
- + Object readObject()
- Методы записывают/читают:
  - класс объекта
  - сигнатуру класса (вычисляется как хэш-функция первых двух элементов дайджеста SHA-1 из имени и модификаторов класса, интерфейсов, типов и имен полей, имен и сигнатур конструкторов и методов)
  - сигнатура класса может быть записана в поле serialVersionUID
  - значения нестатических и непереходных полей данного класса, и всех его суперклассов
  - поля с модификатором transient (переходные) не сериализуются
  - поля записываются в поток иерархически

# Переопределение стандартной сериализации

- Методы в сериализуемом классе
  - private void writeObject(ObjectOutputStream os)
  - private void readObject(ObjectInputStream is)
  - в методах вызываются методы класса Object...Stream:
    - os.defaultWriteObject()
    - is.defaultReadObject()
- интерфейс Externalizable
  - методы:
    - writeExternal(ObjectOutput o)
    - readExternal(ObjectInput i)

# java.nio

- java.nio.Buffer — контейнер для хранения данных




- Создание буфера: allocate(capacity), wrap(array[])
- Методы:
  - limit(lim) и position(pos)
  - mark() и reset()  mark <-> position
  - clear() - позиция = 0, граница = емкость
  - flip() - граница = позиция, позиция = 0
  - rewind() - позиция = 0

# Запись и чтение данных

- методы get и put
- Относительная индексация (по текущей позиции)
  - позиция смещается после операции
  - одиночные и групповые операции
- Абсолютная индексация (явное указание индекса)
  - позиция не меняется
  - только одиночные
- Стандартное использование буфера — заполнение (запись), потом получение данных (чтение)

# Запись и чтение данных

- clear();

```
while () {
    put(byte);
}
```

- flip();

```
while(hasRemaining()) {
    get();
}
```

- rewind();

```
while(hasRemaining()) {
    get();
```

# Классы-потомки Buffer

- ByteBuffer
- CharBuffer
- IntBuffer, ShortBuffer, LongBuffer, FloatBuffer, DoubleBuffer

# java.nio.charset

- Класс Charset
  - методы
  - CharBuffer decode(ByteBuffer b)
  - ByteBuffer encode(CharBuffer c)

# Каналы

- java.nio.channels
- Файловые каналы и сетевые каналы
- FileChannel
  - FileChannel.open(), FileInputStream.getChannel()
  - write(ByteBuffer b) — запись в данный канал данных из буфера
  - read(ByteBuffer b) — чтение из канала данных в буфер
  - write для канала = get для буфера
  - read для канала = put для буфера
  - map() - получение MappedByteBuffer

# Работа с путями (java.nio.file.*)

- Класс Path — путь к файлу в файловой системе
- Paths — класс-помощник
- Path p1 = Paths.get("/home/s888888");
- Path p2 = Paths.get("C:\\Users\s888888");
- Path p = Paths.get(System.getProperty("user.home"));

# Работа с файлами: проверки

- Класс Files
  - booelan Files.exists(Path)
  - booelan Files.notExists(Path)
  - booelan Files.isReadable(Path)
  - booelan Files.isWritable(Path)
  - booelan Files.isExecutable(Path)

# Работа с файлами: создание и удаление

- Path Files.createFile(Path)
- Path Files.createDirectory(Path)
- Path Files.createDirectories(Path)
- Path Files.createTempFile(String prefix, String suffix)
- Path Files.createLink(Path, Path)
- Path Files.createSymbolicLink(Path, Path)
- void Files.delete(Path)
- boolean Files.deleteIfExists(Path)

# Работа с файлами: копирование и перемещение

- Path Files.copy(Path, Path)
- Path Files.move(Path, Path)

# Работа с файлами: чтение и запись

- byte[] Files.readAllBytes(Path)
- List<String> Files.readAllLines(Path)
- Stream Files.lines(Path)
- DirectoryStream<Path> Files.newDirectoryStream(Path)
- Stream Files.list(Path)
- Path Files.write(Path, byte[])
- Path Files.write(Path, Iterable<CharSequence>)

# Сетевое взаимодействие

- Соединение клиент-сервер
  - Сервер предоставляет данные по запросу клиента
  - IP-адрес и порт
- Протоколы транспортного уровня — TCP и UDP
  - TCP
    - устанавливается соединение
    - подтверждение доставки
    - надежность передачи данных
  - UDP
    - без установление соединения
    - без подтверждения доставки
    - скорость передачи данных
- Протокол прикладного уровня - HTTP

# IP-адреса и DNS

- IPv4 и IPv6
- Класс InetAddress (Inet4Address, Inet6Address)
- DNS — Domain Name Service
- Методы InetAddress (статические)
  - InetAddress getLocalHost()
  - InetAddress getByAddress(byte[] addr)
  - InetAddress getByName(String name) — обращение к DNS
- Нестатические методы
  - byte[] getAddress
  - String getHostName()

ИТМО ВТ

# Обмен по протоколу UDP

- java.net.DatagramPacket — дейтаграмма (передаваемые данные + служебная информация)
- java.net.DatagramSocket — сокет для обмена
- java.nio.channels.DatagramChannel — сетевой канал

# Пример обмена по протоколу UDP

```
// клиент

byte b[] = {0,1,2,3,4,5,6,7,8,9};
```

```
SocketAddress a =
   new InetSocketAddress(ADDR, PORT);
DatagramSocket s =
   new DatagramSocket();
```

```
DatagramPacket o =
  new DatagramPacket(b, b.length, a);
s.send(o);
```

```
DatagramPacket i =
  new DatagramPacket(b, b.length);
s.receive(i);
```

```
for (byte i : b) {
    System.out.println(b);
}
```

```
// сервер

byte b[] = new byte[10];
```

```
SocketAddress a =
   new InetSocketAddress(PORT);
DatagramSocket s =
   new DatagramSocket();
```

```
DatagramPacket i =
  new DatagramPacket(b, b.length);
s.receive(i);
```

```
for (int i = 0; i < 10; i++) {
    b[i] *= 2;
}
```

```
DatagramPacket o =
  new DatagramPacket(b, b.length,
  i.getAddress(), i.getPort());
s.send(o);
```

# Пример обмена по протоколу UDP

```
// клиент

byte b[] = {0,1,2,3,4,5,6,7,8,9};
```

```
// сервер

byte b[] = new byte[10];
```

```
SocketAddress a =
   new InetSocketAddress(ADDR, PORT);
DatagramChannel s =
   DatagramChannel.open();
s.bind(null);
ByteBuffer f = ByteBuffer.bind(b);
```

```
SocketAddress a =
   new InetSocketAddress(PORT);
DatagramChannel s =
   DatagramChannel.open();
s.bind(a);
ByteBuffer f = ByteBuffer.bind(b);
```

```
f.flip();
s.send(f, a);
```

```
f.clear();
a = s.receive(f);
```

```
for (int i = 0; i < 10; i++) {
    b[i] *= 2;
}
```

```
f.clear();
a = s.receive(f);
```

```
f.flip();
s.send(f, a);
```

```
for (byte i : b) {
    System.out.println(b);
}
```

# Обмен по протоколу TCP

- java.net.Socket — сокет для обмена
- java.net.ServerSocket — фабрика сокетов
- обмен данными через потоки ввода-вывода
- java.nio.channels.SocketChannel — сетевой канал

# Пример обмена по протоколу TCP

```
// клиент

byte b[] = {0,1,2,3,4,5,6,7,8,9};
```

```
// сервер

byte b[] = new byte[10];
```

```
Socket s = new Socket(ADDR, PORT);
```

```
ServerSocket ss =
    new ServerSocket(PORT);
Socket s = ss.accept();
```

```
OutputStream o = s.getOutputStream();
o.write(b);
```

```
InputStream i = s.getInputStream();
i.read(b);
```

```
for (int i = 0; i < 10; i++) {
    b[i] *= 2;
}
```

```
InputStream i = s.getInputStream();
i.read(b);
```

```
OutputStream o = s.getOutputStream();
o.write(b);
```

```
Arrays.stream(b)
      .forEach(System.out::println());
```

```
i.close(); o.close(); s.close();
```

```
i.close(); o.close(); s.close();
ss.close();
```

# Пример обмена по протоколу TCP (NIO)

```
// клиент

byte b[] = {0,1,2,3,4,5,6,7,8,9};
```

```
// сервер

byte b[] = new byte[10];
```

```
SocketAddress a =
   new InetSocketAddress(ADDR, PORT);



SocketChannel s =
   SocketChannel.open(a);
ByteBuffer f = ByteBuffer.bind(b);
```

```
SocketAddress a =
   new InetSocketAddress(PORT);
ServerSocketChannel ss =
   ServerSocketChannel.open();
ss.bind(a);
SocketChannel s = ss.accept();

ByteBuffer f = ByteBuffer.bind(b);
```

```
f.flip();
s.write(f, a);
```

```
f.clear();
a = s.read(f);
```

```
for (int i = 0; i < 10; i++) {
    b[i] *= 2;
}
```

```
f.clear();
a = s.read(f);
```

```
f.flip();
s.write(f, a);
```

```
s.close();
```

```
s.close();
ss.close();
```

# Протокол HTTP

```
URL url = new URL("http://www.google.com");
InputStream is = url.openStream();
// is.read();
is.close();

URLConnection uc = url.getConnection();
uc.connect();
InputStream is = uc.getInputStream();
// is.read();
is.close();
uc.close()
```