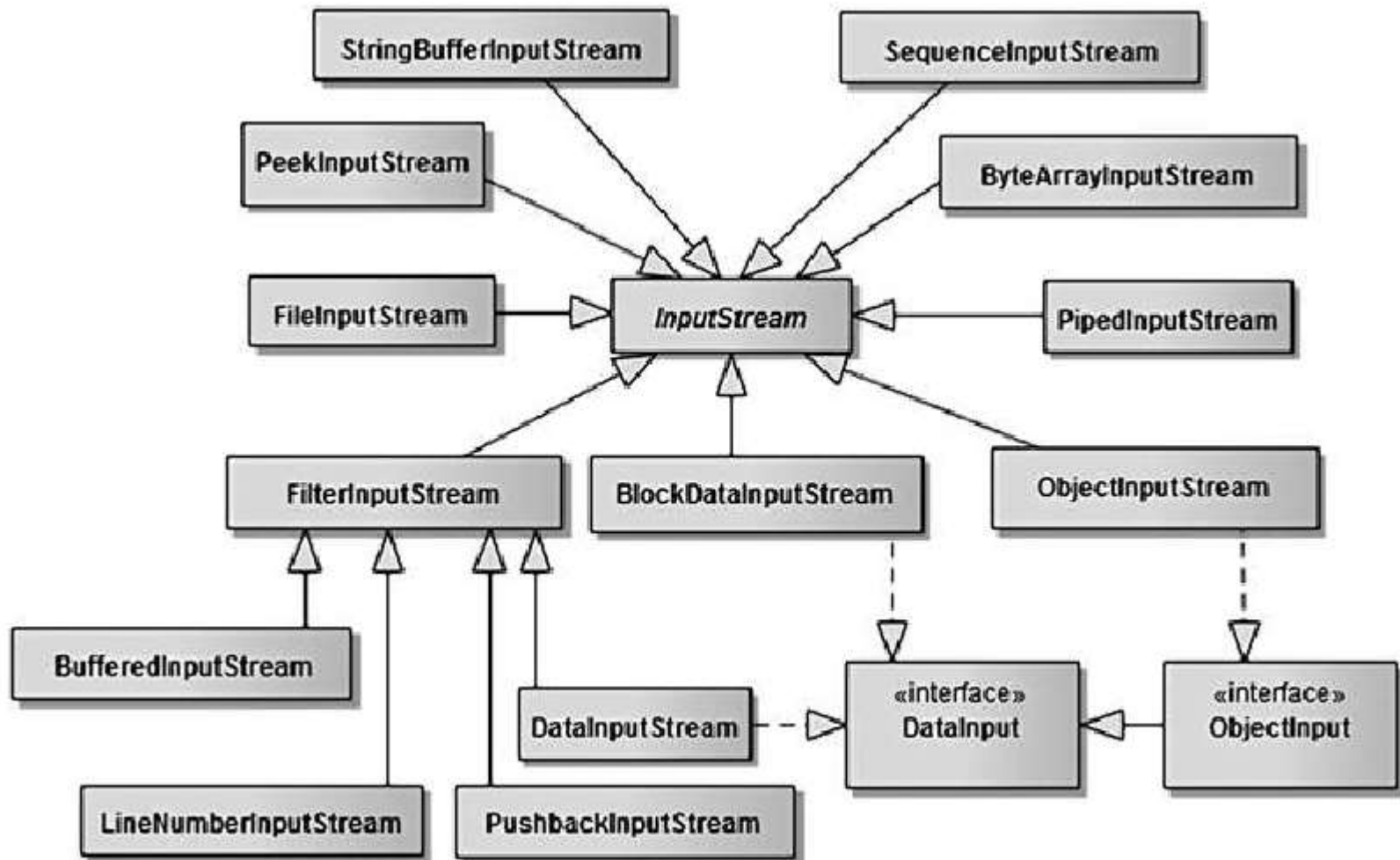


Потоки ввода/вывода. Сериализация Архивация

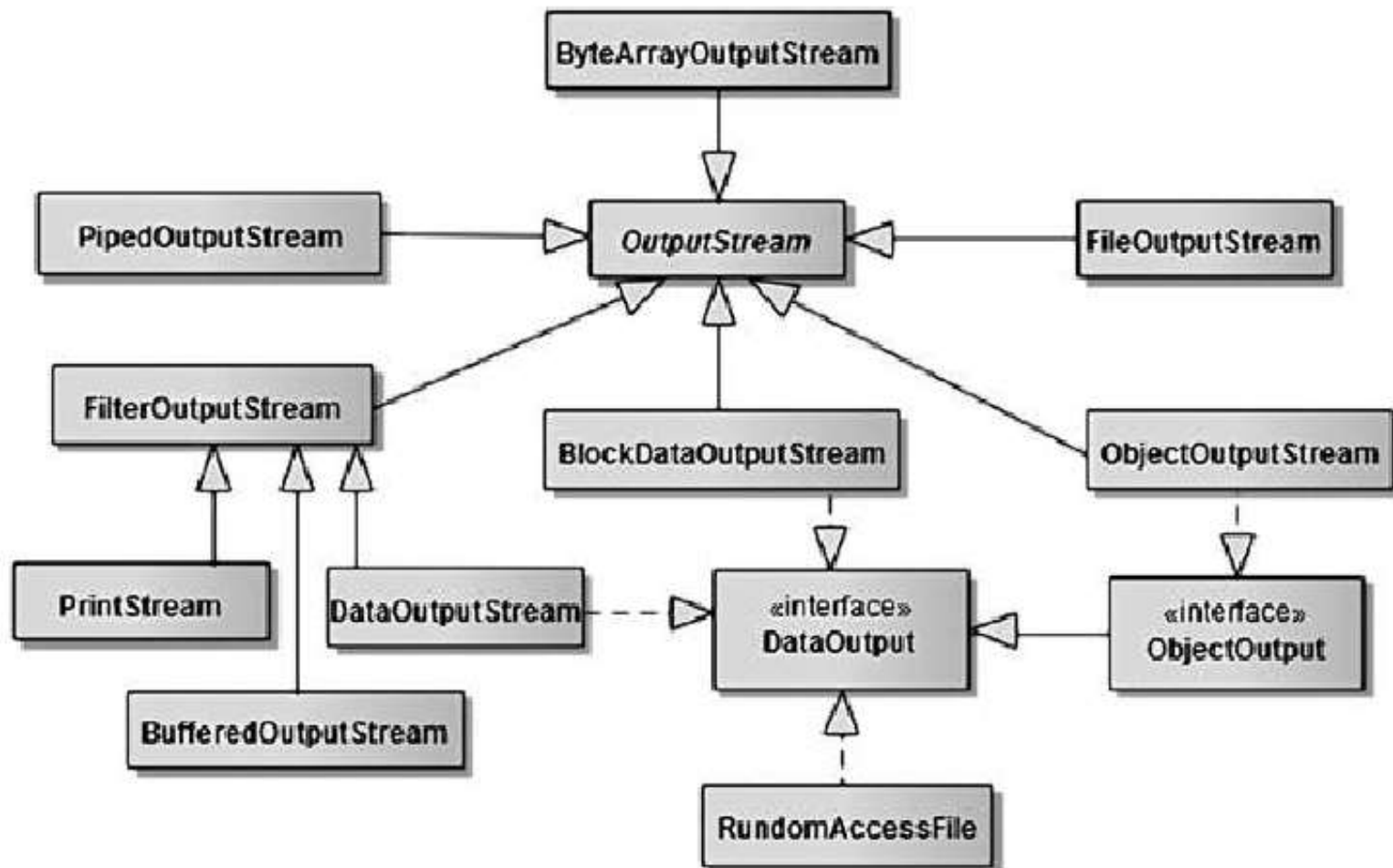
ПОТОКИ ВВОДА/ВЫВОДА

- ▶ java.io java.nio
- ▶ InputStream read
- ▶ OutputStream write

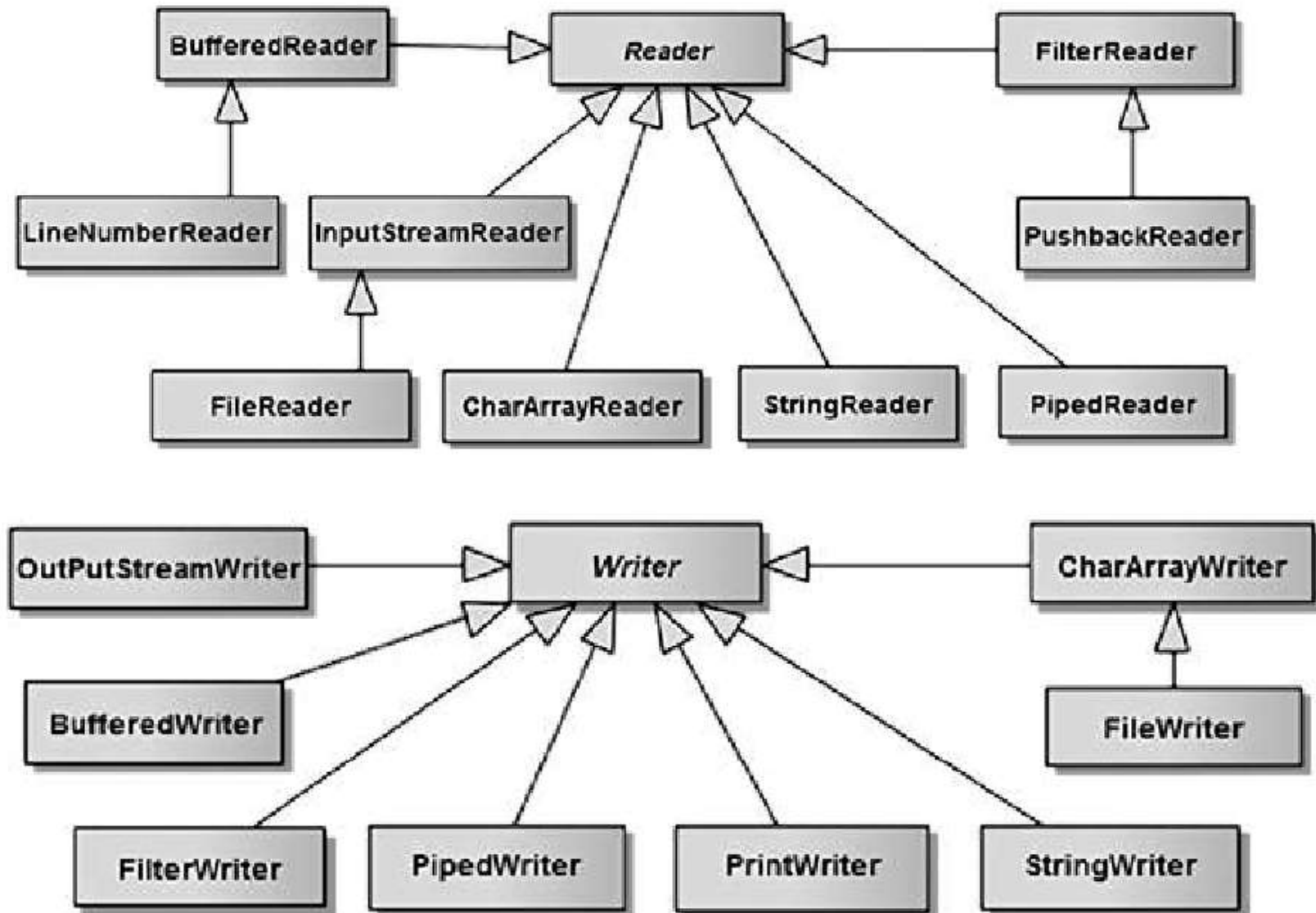
Байтовые потоки ввода



Байтовые потоки вывода

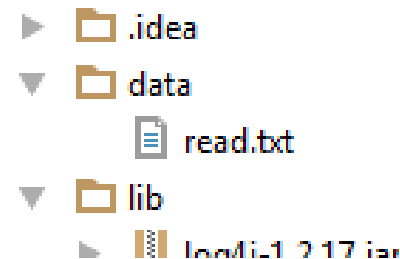


СИМВОЛЬНЫЕ ПОТОКИ



```
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
```

```
public class ReadDemo {
    public static void main(String[] args) {
        String file = "data/read.txt";
        File f = new File(file); // объект для связи с файлом на диске
        int b, count = 0;
        FileReader fileReader = null;
        try {
            fileReader = new FileReader(f);
            while ((b = fileReader.read()) != -1) { // чтение
                System.out.print((char) b);
                count++;
            }
            System.out.print("\n число байт = " + count);
        } catch (IOException e) {
            System.err.println("Ошибка файла: " + e.getMessage());
        } finally {
            if (fileReader != null)
                try {
                    fileReader.close(); // закрытие потока ввода
                } catch (IOException e) {
                    e.printStackTrace();
                }
        }
    }
}
```



"C:\Program ...
Для закрытия потока используется метод close(). Закр
произойти при любом исходе чтения: удачном или с ген
число байт = 143
Process finished with exit code 0

интерфейс AutoCloseable

► Реализуется потоковыми классами

Метод `close()` вызывается неявно для всех потоков, открытых в инструкции

`try(Поток1: Поток2:....: ПотокN)`

```
try (FileReader iis = new FileReader(new File("data/file.txt")))
    int byteR = 0;
    /* чтение */
    while ((b = iis.read()) != -1)
        System.out.print((char) byteR);
} catch (IOException e) {
    System.err.println( e.getMessage());
}
```

Класс File

- ▶ не содержит методы для работы с содержимым файла
- ▶ Управление свойствами файла
 - право доступа
 - дата и время создания
 - путь в иерархии каталогов
 - создание и удаление файла,
 - изменение имени и каталога и т. д

```
File obF = new File("data\\read.txt"); //файл
File obDir = new File("c:/temp/");
File obF1 = new File(obDir, "File.java");
File obF2 = new File("c:\\temp", "read.txt");
File obF3 = new File(new URI("www.belstu.by"));

File obF4 = new File(File.separator + "data" +
                    File.separator + "read.txt" );
```



```
File fileInfo = new File("data/read.txt");
if(fileInfo.exists()) {
    System.out.println(fileInfo.getName());
    if(fileInfo.isFile()) { // если объект - дисковый файл
        System.out.println("Путь : \t" + fileInfo.getPath());
        System.out.println("Размер : \t" + fileInfo.length());
        System.out.println("Модификация : \t"
            +new Date(fileInfo.lastModified())
            +new Time(fileInfo.lastModified()));
        System.out.println("Для чтения: \t" + fileInfo.canRead());
        System.out.println("Для записи: \t" + fileInfo.canWrite());
    }
} else
```

```
число байт = 143read.txt
Путь : data\read.txt
Размер : 257
Модификация : Wed Mar 09 12:01:52 EET 201612:01:52
Для чтения: true
Для записи: true
```

```
File dir = new File("C:"+File.separator+"office");  
File[] files = dir.listFiles();
```

```
for(int i = 0; i < files.length; i++) {  
    Date date = new Date(files[i].lastModified());  
    System.out.print("\n"+files[i].getPath()+" \t|  
                    "+files[i].length()+"\t|"+date);  
}
```

```
File root = File.listRoots()[0];  
System.out.printf("\n%s %,d из %,d свободно.",  
root.getPath(), root.getUsableSpace(),  
root.getTotalSpace());  
}
```

```
C:\office\Crack      | 0 |Thu Mar 27 09:51:31 EET 2014  
C:\office\m0nkrus.nfo | 2471 |Tue Nov 13 21:34:56 EET 2012  
C:\office\readme_eng.htm | 427 |Mon Jul 30 22:36:52 EEST 2012  
C:\office\readme_rus.htm | 827 |Sun Aug 19 20:54:18 EEST 2012  
C:\office\setup.dll    | 647816 |Mon Oct 01 05:16:04 EEST 2012  
C:\office\setup.exe    | 207496 |Mon Oct 01 05:13:16 EEST 2012  
C:\office\x64         | 0 |Thu Mar 27 09:54:32 EET 2014  
C:\office\x86         | 0 |Thu Mar 27 09:55:40 EET 2014  
C:\ 140 018 085 888 из 500 000 878 592 свободно.  
Process finished with exit code 0
```

Предопределенные потоки

► Класс System

- in - ссылка на объект класса InputStream,
 - out, err — ссылка на объекты PrintStream
- ввода, вывода и вывода ошибок
связаны с консолью (могут быть
переназначены)

Сериализация объектов

- ▶ Сериализация - процесс преобразования объектов в потоки байтов для хранения и передачи.
- ▶ Десериализация - процесс извлечения объекта из потока байтов.

интерфейс Serializable

► java.io.Serializable

- 1) transient и static - не сериализуются
 - a) transient - значение по умолчанию (deser)
 - b) static - значение по умолчанию в случае отсутствия в области видимости объектов своего типа или значение которое определено для существующего объекта
- 2) Вложенный тип - д.б. сериализуемый

- 3) Не требует реализации методов
- 4) запись объектов в поток - класс `ObjectOutputStream`.
`writeObject(Object ob)`
- 5) чтение - класс `ObjectInputStream`
`.readObject()` + преобразование к типу
- 6) При десериализации конструктор не вызывается (память + запись полей)

```
class BankCard implements Serializable {

    protected static String bankName;
    private int id;
    private transient int password;
    private static final long serialVersionUID = 1L;
    static {
        bankName = "BBB";
    }

    @Override
    public String toString() {
        return "BankCard{" +
            "id=" + id +
            ", password=" + password +
            ", Bank=" + bankName +
            '}';
    }

    public BankCard(int id, int password) {
        this.id = id;
        this.password = password;
    }
}
```

```
class Serializator {
    public boolean serialization(BankCard card, String fileName) {
        boolean flag = false;
        File f = new File(fileName);
        try (FileOutputStream fos = new FileOutputStream(f)) {
            if (fos != null) {
                ObjectOutputStream ostream =
                    new ObjectOutputStream(fos);
                ostream.writeObject(card); // сериализация
                flag = true;
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        return flag;
    }
}
```



```
public BankCard deserialization(String fileName)
    throws InvalidObjectException {
    File fr = new File(fileName);

    try ( FileInputStream fis =
        new FileInputStream(fr)) {
        ObjectInputStream istream =
            new ObjectInputStream(fis);
        // десериализация
        BankCard card =
            (BankCard) istream.readObject();
        return card;
    }
    catch ( IOException | ClassNotFoundException e) {
        e.printStackTrace();
    }
    throw new InvalidObjectException
        ("объект не восстановлен");
}
}
```

```

public static void main(String[] args) {
    // создание и запись объекта
    BankCard somecard = new BankCard(1234, 1111);
    System.out.println(somecard);
    String file = "data/cards.data";
    Serializator sz = new Serializator();
    boolean b = sz.serialization(somecard, file);
    // BankCard.bankName = "SSS"; // изменение значения static-поля
    // чтение и вывод объекта
    BankCard res = null;
    try {
        res = sz.deserialization(file);
    } catch (InvalidObjectException e) {
        // обработка
        e.printStackTrace();
    }
    System.out.println(res);
}

```

```

BankCard{id=1234, password=1111, Bank=BBB}
BankCard{id=1234, password=0, Bank=BBB}

```

► Уникальный идентификатор версии сериализованного класса

```
class BankCard implements Serializable {  
private static final long serialVersionUID = 1L;  
}
```

записывается в поток при сериализации класса

если значения не совпадают, иницируется исключение `java.io.InvalidClassException`

интерфейс Externalizable

```
void writeExternal (ObjectOutput out)  
void readExternal (ObjectInput in)
```

Особенности:

- 1) Реализация методов в классе
- 2) Нужен конструктор по умолчанию
- 3) Для чтения и записи полей

ObjectInputStream.GetField

ObjectOutputStream.PutField

Класс Scanner

интерфейс для извлечения информации из любых источников (читает лексемы)

```
Scanner con = new Scanner(System.in);
```

Чтение:

```
String str1 = con.next();  
String str2 = con.nextLine();  
  
if (con.hasNextInt()) {  
    int n = con.nextInt();  
}
```

boolean hasNextТип(.)
useDelimiter (Pattern pattern)

```
FileReader fr = new FileReader(filename);  
scan = new Scanner(fr);
```

Архивация

JAR-архивы — файлы с упакованными данными (текстовые файлы, class-файлы, файлы с графической информацией и пр.), которые обычно используются для хранения частей java-программ.

► `java.util.zip` и `java.util.jar`

Класс *JarEntry* - доступ к записям jar-файла

`void setMethod(int method)`

`void setSize(long size)`

`long getSize()`

`long getCompressedSize()`

- ▶ Класс *JarOutputStream* - возможность записи данных в поток вывода в jar-формате
- ▶ Класс *JarFile* - доступ к записям, хранящимся в jar файле.
- ▶ Класс *JarInputStream* - читает данные в jar-формате из потока ввода

Создание jar архивов

файл описания (**manifest file**)

- ▶ номер версии стандарта JAR (**Manifest-Version**);
- ▶ минимальный номер версии утилиты JAR, которая сможет прочитать этот архив (**Required-Version**);
- ▶ отдельная запись для любого, помещённого в архив файла.

JDK (Java Developer Kit) - jar

C:\Users>jar

Usage: jar {ctxui}[vfm0Me] [jar-file] [manifest-file] [entry-point] [-C dir] files ...

Options:

- c create new archive
- t list table of contents for archive
- x extract named (or all) files from archive
- u update existing archive
- v generate verbose output on standard output
- f specify archive file name
- m include manifest information from specified manifest file
- n perform Pack200 normalization after creating a new archive
- e specify application entry point for stand-alone application bundled into an executable jar file
- 0 store only; use no ZIP compression
- M do not create a manifest file for the entries
- i generate index information for the specified jar files
- C change to the specified directory and include the following file

If any file is a directory then it is processed recursively.

The manifest file name, the archive file name and the entry point name are specified in the same order as the 'm', 'f' and 'e' flags.

Example 1: to archive two class files into an archive called classes.jar:

```
jar cvf classes.jar Foo.class Bar.class
```

Example 2: use an existing manifest file 'mymanifest' and archive all the files in the foo/ directory into 'classes.jar':

```
jar cvfm classes.jar mymanifest -C foo/ .
```

C:\Windows\system32\cmd.exe

```
C:\Temp>jar -cf sm.jar Demo.class Bill.class
```

```
C:\Temp>jar -tf sm.jar
```

```
META-INF/
```

```
META-INF/MANIFEST.MF
```

```
Demo.class
```

```
Bill.class
```

```
C:\Temp>
```

C:\Windows\system32\cmd.exe

```
C:\Temp>
```

```
C:\Temp>jar cf sm2.jar *.class
```

► Архивирование пакетов

C:\Windows\system32\cmd.exe

```
C:\Temp>jar cvf sm3.jar *class -C . \base *.class
added manifest
adding: Runner.class(in = 1523) (out= 725)(deflated 52%)
adding: SimpleAct.class(in = 656) (out= 364)(deflated 44%)
adding: base/(in = 0) (out= 0)(stored 0%)
adding: base/AirDemo.class(in = 1476) (out= 779)(deflated 47%)
adding: base/AirType.class(in = 1788) (out= 879)(deflated 50%)
adding: base/bill/(in = 0) (out= 0)(stored 0%)
adding: base/bill/Bill.class(in = 941) (out= 610)(deflated 35%)
```

External Libraries

< 1.8 (1) > (C:\Program Files\Java\jdk1.8.0_25)

sm3

sm3.jar (library home)

base

bill

AirDemo

AirType

META-INF

MANIFEST.MF

Runner

SimpleAct

Project Structure

log4j-1.2.17

sm3

Project Settings

Project

Modules

Libraries

Facets

Artifacts

Platform Setting

SDKs

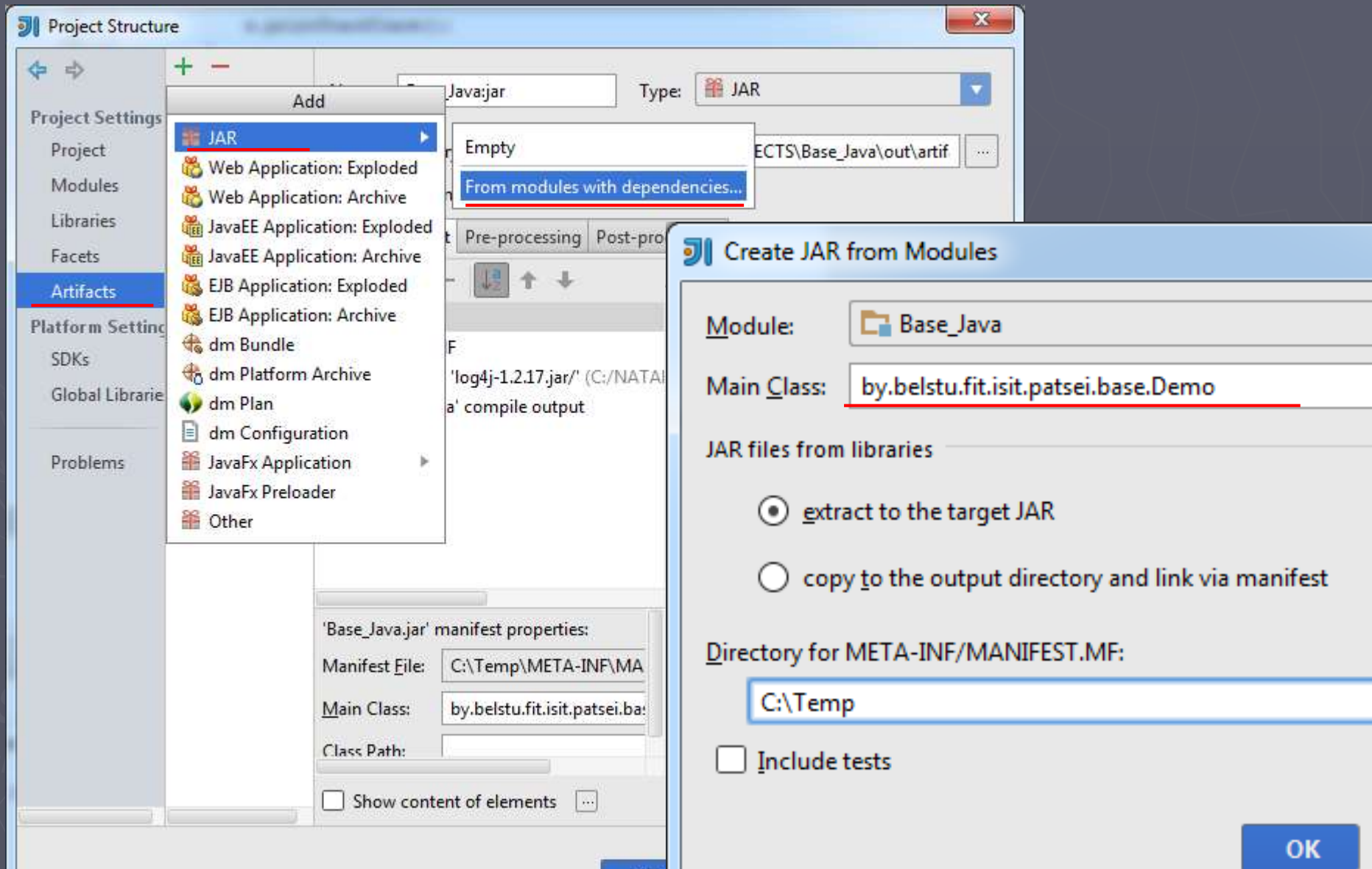
Global Libraries

Name: sm3

Classes

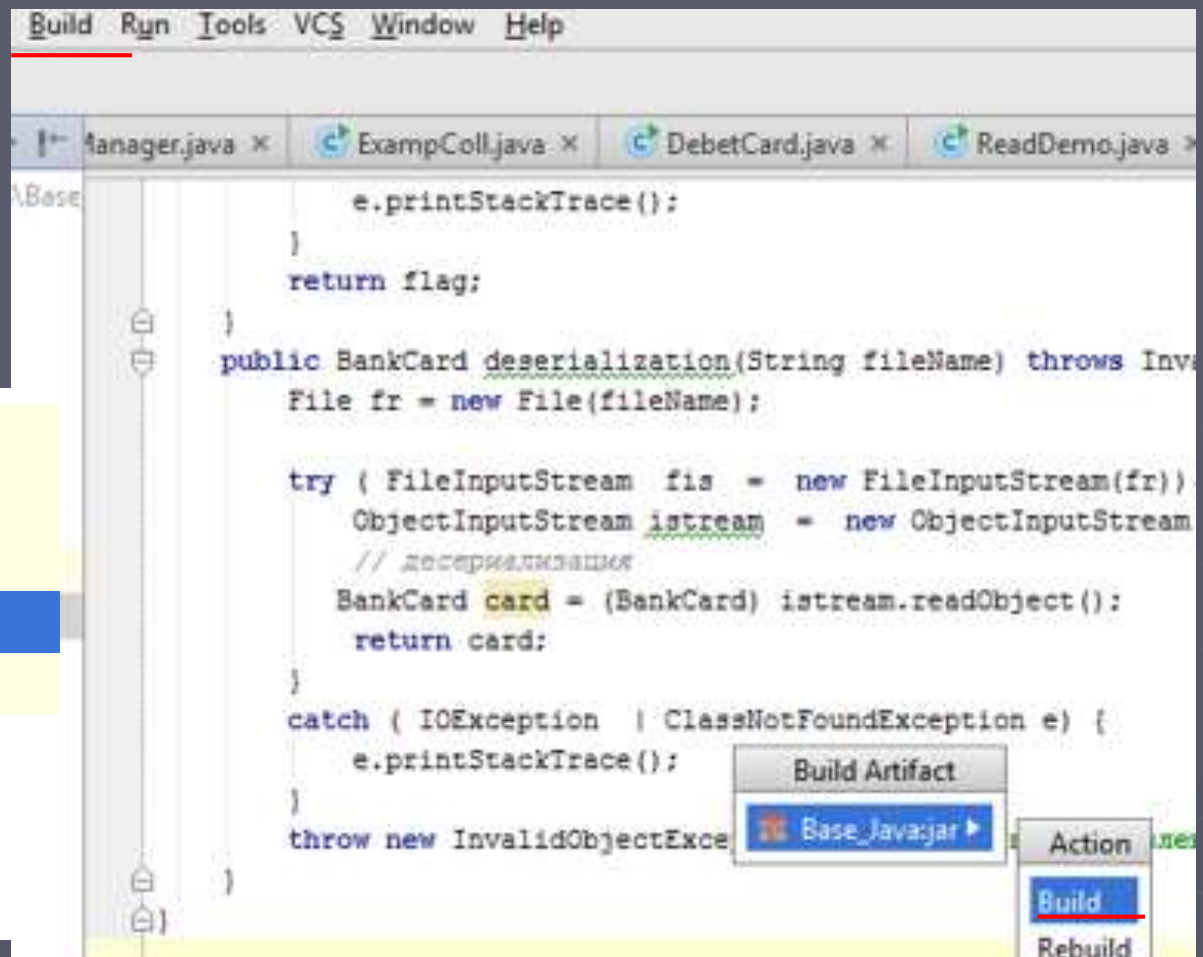
C:\Temp\sm3.jar

Project Structure -> Artifacts -> Jar -> From modules with dependencies



out/artifacts/
Base_java/

- ▼ out
 - ▼ artifacts
 - ▼ Base_Java_jar
 - Base_Java.jar
 - ▶ production
 - ▶ src
 - Base_Java.iml
 - External Libraries

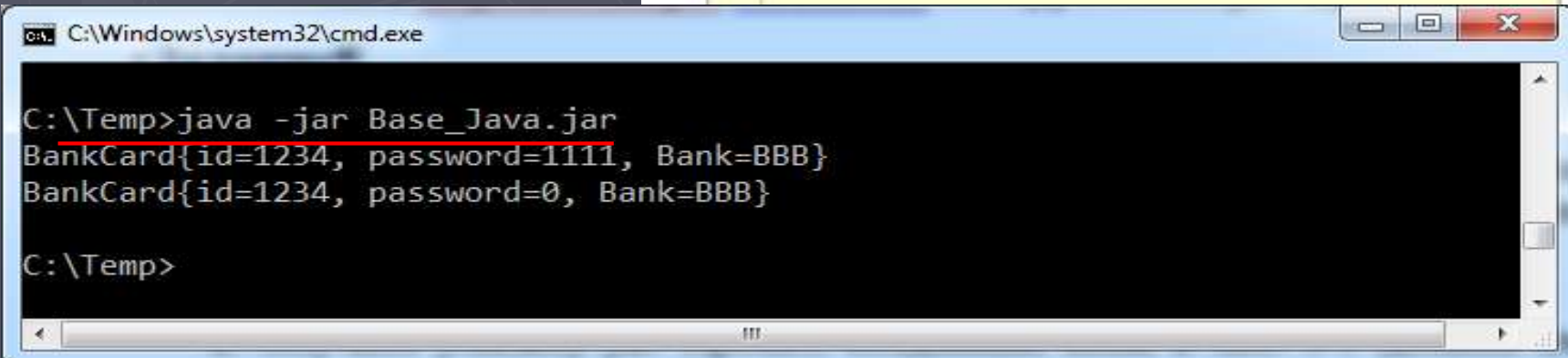


The screenshot shows an IDE window with several tabs: manager.java, ExampColl.java, DebetCard.java, and ReadDemo.java. The active tab is manager.java, which contains the following Java code:

```
    e.printStackTrace();
}
return flag;
}
public BankCard deserialization(String fileName) throws InvalidObjectException {
    File fr = new File(fileName);

    try {
        FileInputStream fis = new FileInputStream(fr);
        ObjectInputStream istream = new ObjectInputStream(fis);
        // десериализация
        BankCard card = (BankCard) istream.readObject();
        return card;
    }
    catch (IOException | ClassNotFoundException e) {
        e.printStackTrace();
    }
    throw new InvalidObjectException("Invalid object");
}
```

A context menu is open over the code, showing the "Build Artifact" option. The "Build Artifact" menu is open, showing the "Base_Java.jar" artifact. The "Action" menu is also open, showing the "Build" option.



The screenshot shows a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The command prompt shows the following commands and output:

```
C:\Temp>java -jar Base_Java.jar
BankCard{id=1234, password=1111, Bank=BBB}
BankCard{id=1234, password=0, Bank=BBB}

C:\Temp>
```