НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Object-Oriented Programming in the Java language

Part 5. Exceptions. I/O in Java

Yevhen Berkunskyi, NUoS
eugeny.berkunsky@gmail.com
http://www.berkut.mk.ua

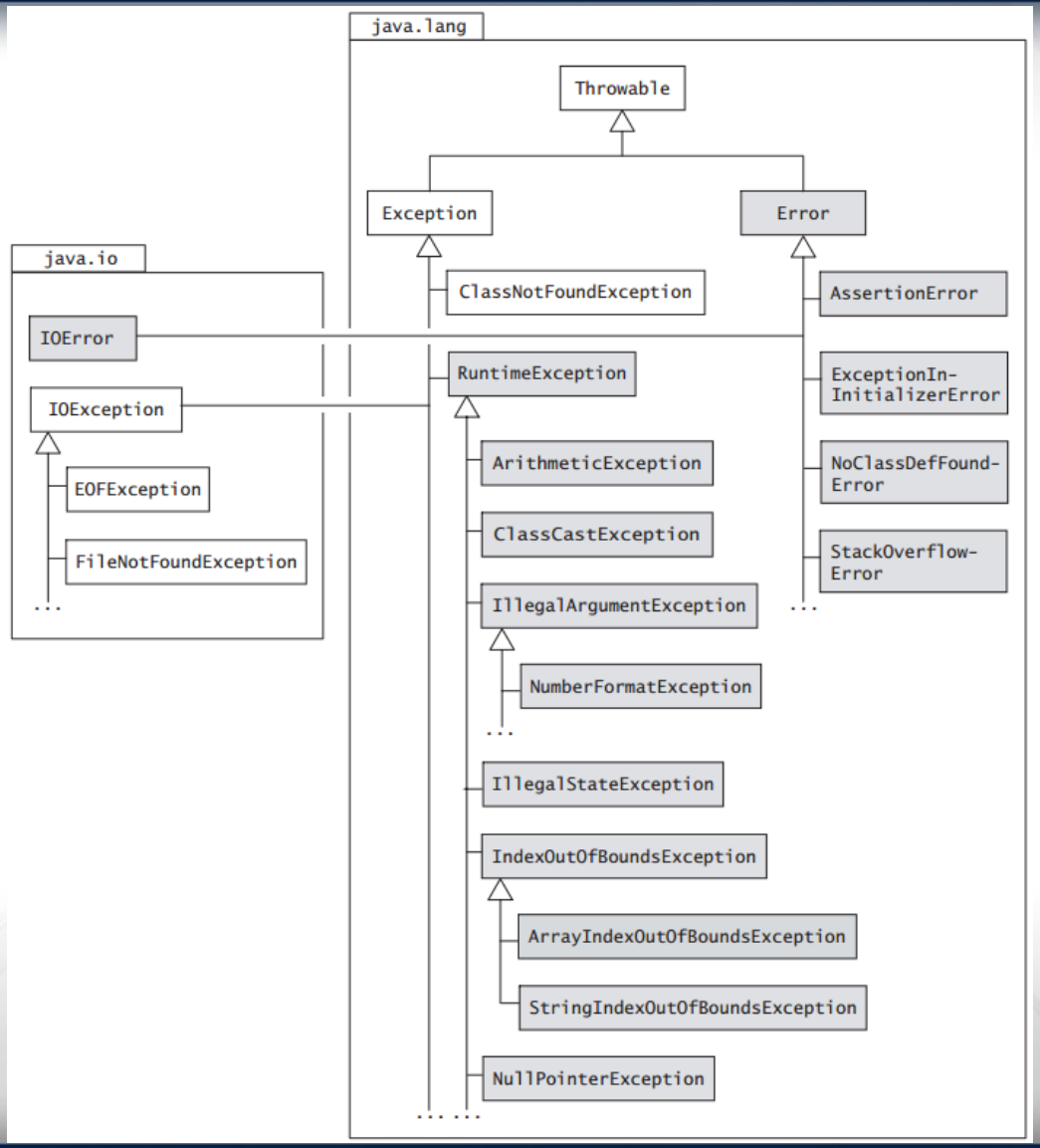
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Exceptions

- Exceptions in Java are objects. All exceptions are derived from the java.lang.Throwable class.
- The two main subclasses Exception and Error constitute the main categories of throwables, the term used to refer to both exceptions and errors.
- The **Exception** Class represents exceptions that a program would normally want to catch. Its subclass **RuntimeException** represents many common programming errors that can manifest at runtime (see the next subsection). Other subclasses of the Exception class define other categories of exceptions, e.g., I/O-related exceptions in the java.io package (IOException, FileNotFoundException, EOFException, IOError)

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Exceptions



НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

The RuntimeException Class

- Runtime exceptions are all subclasses of the java.lang.RuntimeException class, which is a subclass of the Exception class.
- As these runtime exceptions are usually caused by program bugs that should not occur in the first place, it is usually more appropriate to treat them as faults in the program design and let them be handled by the default exception handler
- *ArithmeticException*, *ArrayIndexOutOfBoundsException*, *ClassCastException*, *IllegalArgumentException*, *NumberFormatException*, *IllegalStateException*, *NullPointerException*

The Error Class

- The class Error and its subclasses define errors that are invariably never explicitly caught and are usually irrecoverable.
- *AssertionError*, *ExceptionInInitializerError*, *IOError*, *NoClassDefFoundError*, *StackOverflowError*,

Checked and Unchecked Exceptions

- Except for **RuntimeException**, **Error**, and ***their subclasses***, all exceptions are called *checked* exceptions.
- The compiler ensures that if a method can throw a checked exception, directly or indirectly, the method must explicitly deal with it.
- The method must either catch the exception and take the appropriate action, or pass the exception on to its caller

Checked and Unchecked Exceptions

- Exceptions defined by Error and RuntimeException classes and their subclasses are known as *unchecked* exceptions, meaning that a method is not obliged to deal with these kinds of exceptions.
- They are either irrecoverable (exemplified by the Error class) and the program should not attempt to deal with them, or
- They are programming errors (exemplified by the RuntimeException class) and should usually be dealt with as such, and not as exceptions.

Input & Output

- Creating a good input/output (I/O) system is one of the more difficult tasks for a language designer.
- This is evidenced by the number of different approaches.

The File class (java.io)

- The **File** class has a deceiving name; you might think it refers to a file, but it doesn't.
- In fact, "FilePath" would have been a better name for the class.
- It can represent either the *name* of a particular file or the *names* of a set of files in a directory. If it's a set of files, you can ask for that set using the **list( )** method, which returns an array of **String**.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Interoperability with java.nio.file package

- java.nio.file package defines interfaces and classes for the Java virtual machine to access files, file attributes, and file systems.
- This API may be used to overcome many of the limitations of the java.io.File class. The toPath method may be used to obtain a Path that uses the abstract path represented by a File object to locate a file.
- The resulting Path may be used with the Files class to provide more efficient and extensive access to additional file operations, file attributes, and I/O exceptions to help diagnose errors when an operation on a file fails.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Example

Input and output

- The Java library classes for I/O are divided by input and output, as you can see by looking at the class hierarchy in the JDK documentation.
- Through inheritance, everything derived from the **InputStream** or **Reader** classes has basic methods called **read( )** for reading a single **byte** or an array of **byte**s.
- Likewise, everything derived from **OutputStream** or **Writer** classes has basic methods called **write( )** for writing a single **byte** or an array of **byte**s.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

InputStream & OutputStream

- In **java.io**, the library designers started by deciding that all classes that had anything to do with input would be inherited from **InputStream**, and all classes that were associated with output would be inherited from **OutputStream**.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Types of InputStream

InputStream's job is to represent classes that produce input from different sources.

These sources can be:

1. An array of bytes.
2. A **String** object.
3. A file.
4. *A "pipe," which works like a physical pipe: You put things in at one end and they come out the other.*
5. *A sequence of other streams, so you can collect them together into a single stream*
6. *Other sources, such as an Internet connection.*

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Types of InputStream

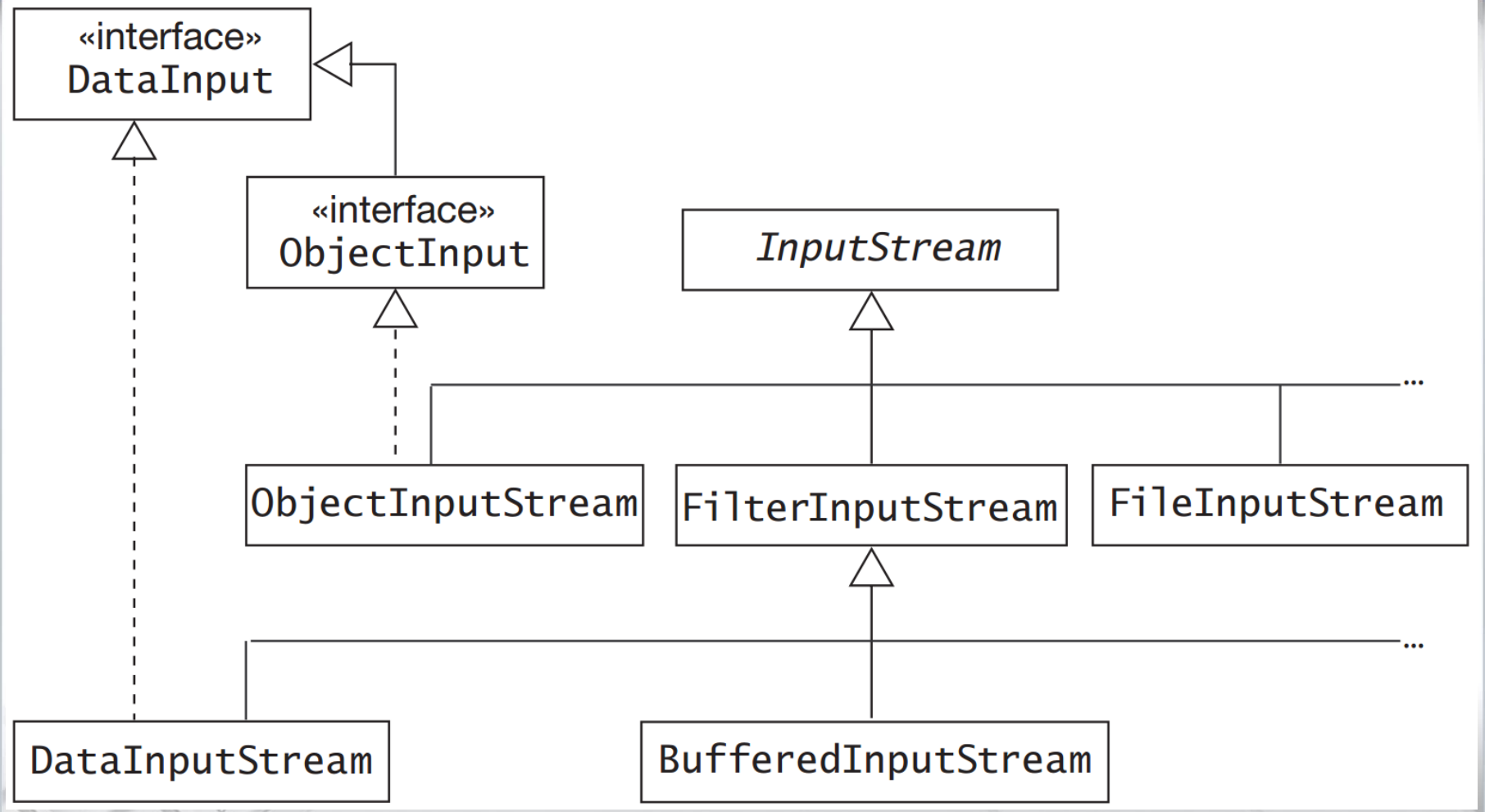
Class	Function	Constructor arguments How to use it
ByteArray-InputStream	Allows a buffer in memory to be used as an **InputStream.**	The buffer from which to extract the bytes. As a source of data: Connect it to a **FilterInputStream** object to provide a useful interface.
StringBuffer-InputStream	Converts a **String** into an **InputStream.**	A **String**. The underlying implementation actually uses a **StringBuffer**. As a source of data: Connect it to a **FilterInputStream** object to provide a useful interface.
File-InputStream	For reading information from a file.	A **String** representing the file name, or a **File** or **FileDescriptor** object. As a source of data: Connect it to a **FilterInputStream** object to provide a useful interface.

Types of OutputStream

Class	Function	Constructor arguments How to use it
ByteArray-OutputStream	Creates a buffer in memory. All the data that you send to the stream is placed in this buffer.	Optional initial size of the buffer. To designate the destination of your data: Connect it to a **FilterOutputStream** object to provide a useful interface.
File-OutputStream	For sending information to a file.	A **String** representing the file name, or a **File** or **FileDescriptor** object.

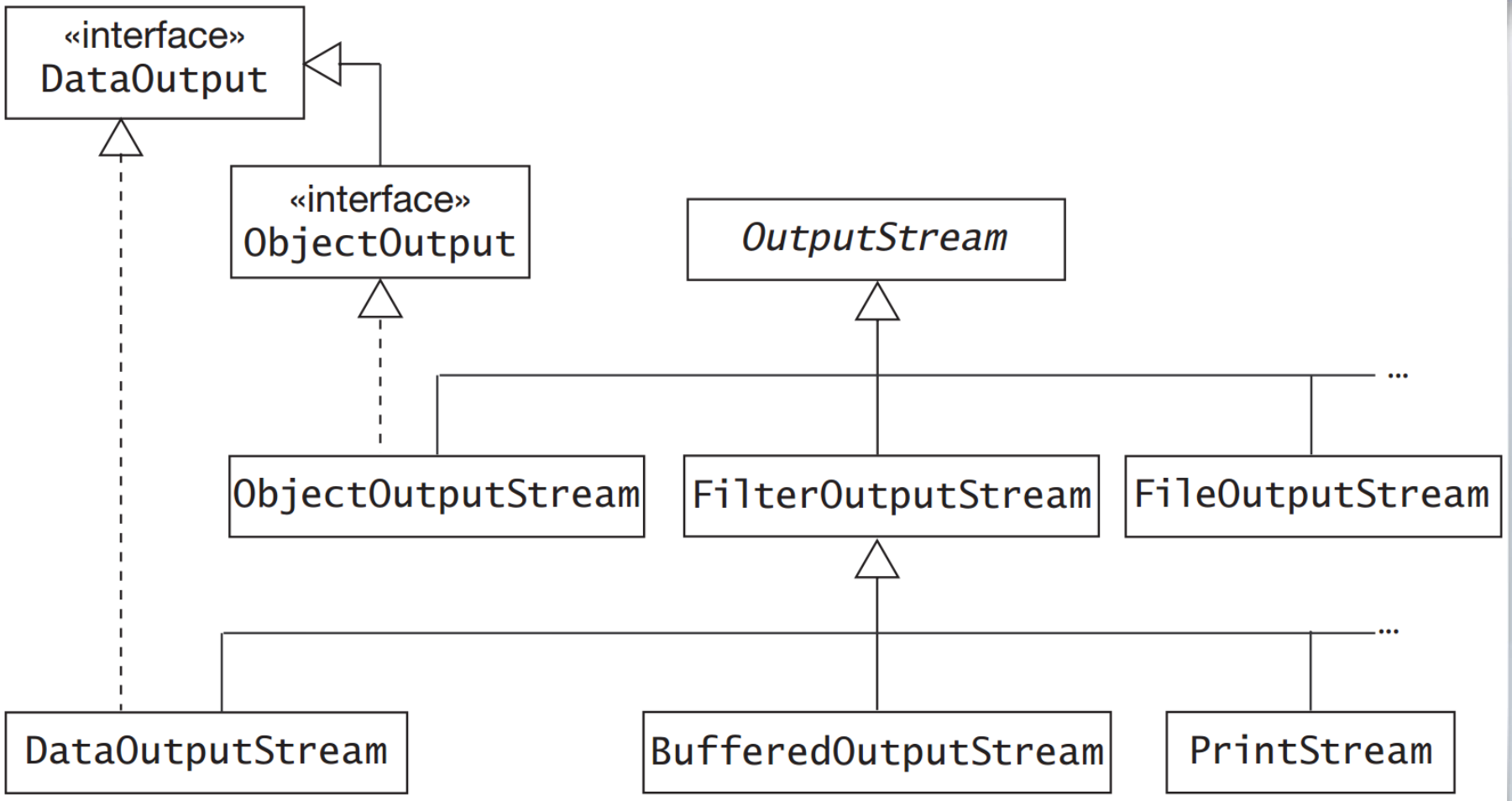
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Hierarchy of InputStreams



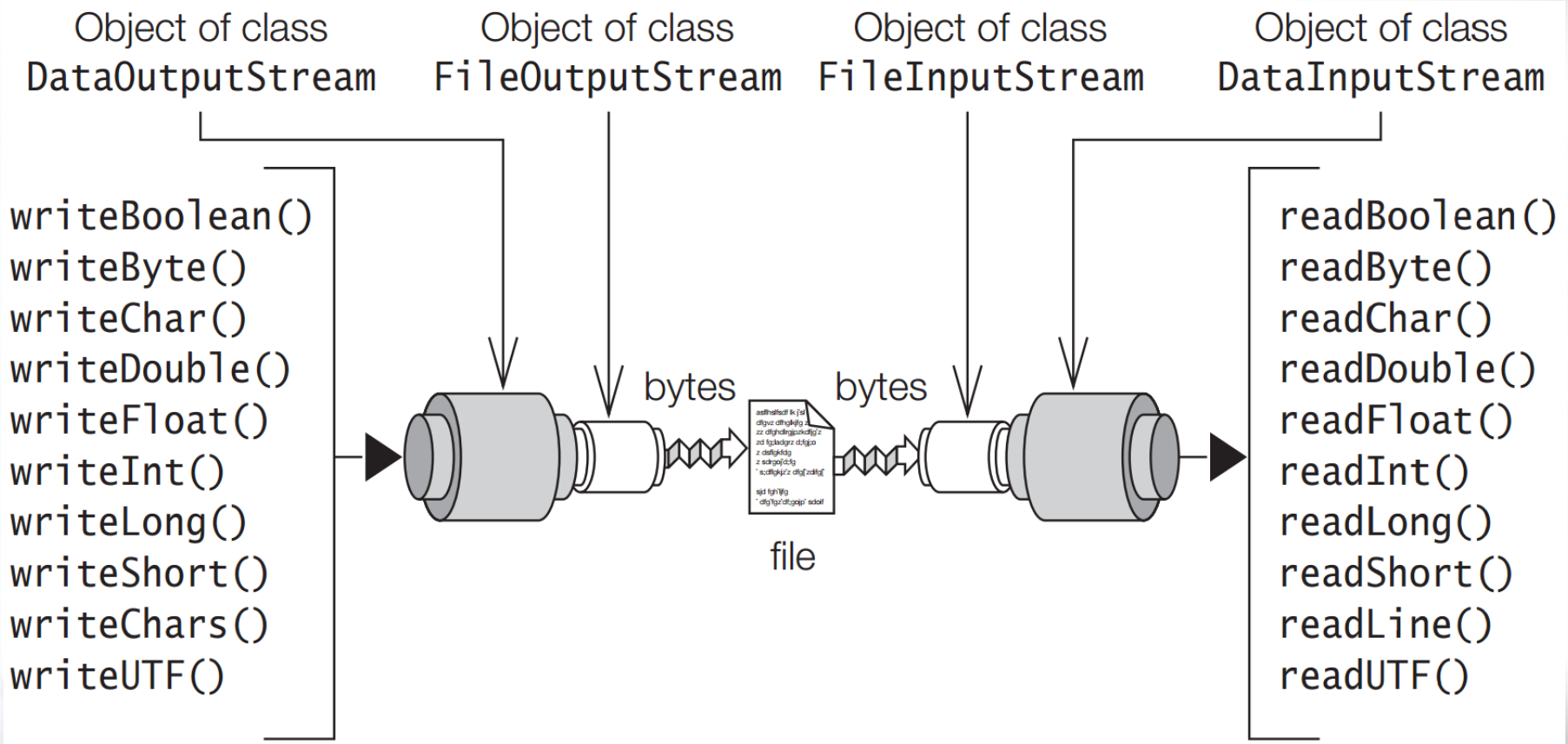
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Hierarchy of OutputStreams



НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

DataOutputStream & DataInputStream



НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Writing Binary Values to a File

1. Create a FileOutputStream:

```
FileOutputStream outputFile =
        new FileOutputStream("primitives.data");
```

2. Create a DataOutputStream which is chained to the FileOutputStream:

```
DataOutputStream outputStream =
        new DataOutputStream(outputFile);
```

3. Write Java primitive values using relevant write*X*() methods:

```
outputStream.writeBoolean(true);
outputStream.writeChar('A');
outputStream.writeByte(Byte.MAX_VALUE);
outputStream.writeShort(Short.MIN_VALUE);
outputStream.writeInt(Integer.MAX_VALUE);
outputStream.writeLong(Long.MIN_VALUE);
outputStream.writeFloat(Float.MAX_VALUE);
outputStream.writeDouble(Math.PI);
```

4. Close the filter stream, which also closes the underlying stream:

```
outputStream.close();
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Reading Binary Values From a File

1. Create a FileInputStream:

```
FileInputStream inputFile =
        new FileInputStream("primitives.data");
```

2. Create a DataInputStream which is chained to the FileInputStream:

```
DataInputStream inputStream =
        new DataInputStream(inputFile);
```

3. Read the (exact number of) Java primitive values in the *same order* they were written out, using relevant readX() methods:

```
boolean v = inputStream.readBoolean();
char c = inputStream.readChar();
byte b = inputStream.readByte();
short s = inputStream.readShort();
int i = inputStream.readInt();
long l = inputStream.readLong();
float f = inputStream.readFloat();
double d = inputStream.readDouble();
```

4. Close the filter stream, which also closes the underlying stream:

```
inputStream.close();
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

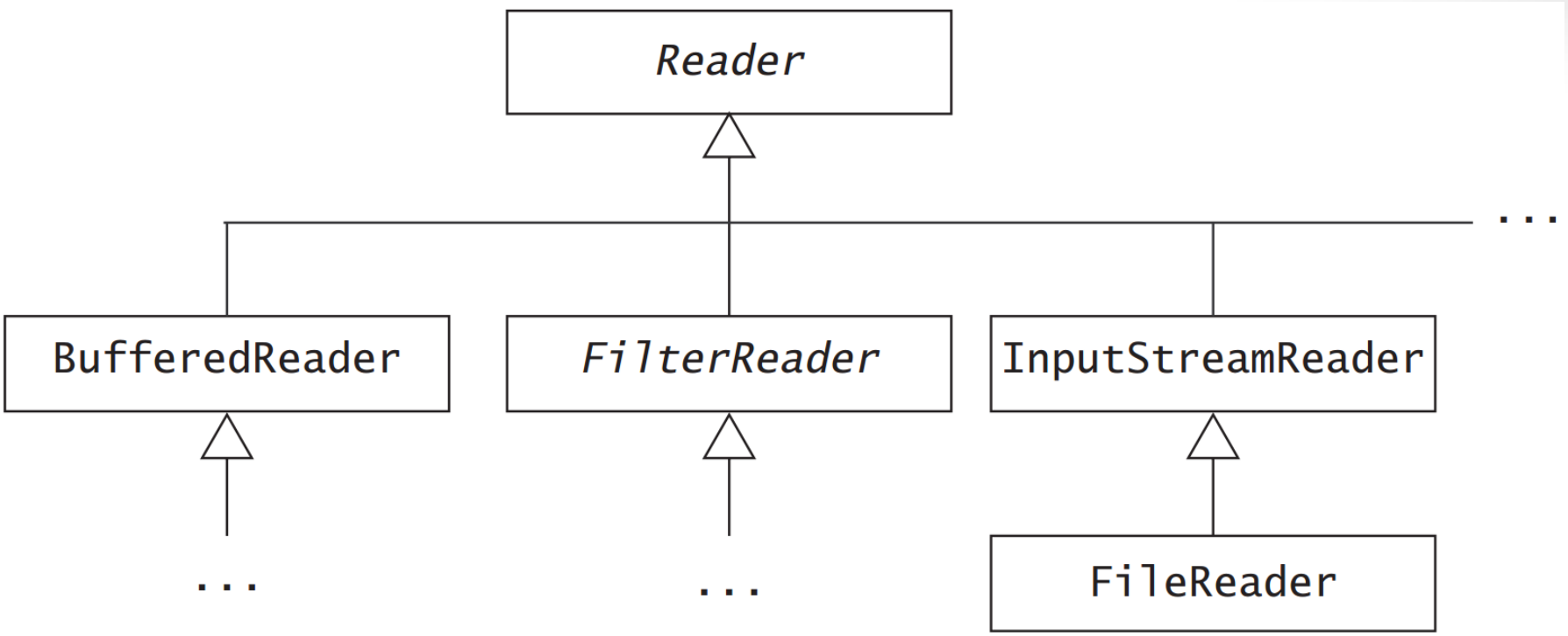
Character Streams: Readers and Writers

- A *character encoding* is a scheme for representing characters. Java programs represent values of the char type internally in the 16-bit Unicode character encoding, but the host platform might use another character encoding to represent and store characters externally.
- The abstract classes Reader and Writer are the roots of the inheritance hierarchies for streams that read and write *Unicode characters* using a specific character encoding
- A *reader* is an input character stream that reads a sequence of Unicode characters, and a *writer* is an output character stream that writes a sequence of Unicode characters.
- Character encodings are used by readers and writers to convert between external encoding and internal Unicode characters.

Selected Readers

BufferedReader	A reader that buffers the characters read from an underlying reader. The underlying reader must be specified and an optional buffer size can be given.
InputStreamReader	Characters are read from a byte input stream which must be specified. The default character encoding is used if no character encoding is explicitly specified.
FileReader	Reads characters from a file, using the default character encoding. The file can be specified by a File object, a FileDescriptor, or a String file name. It automatically creates a FileInputStream that is associated with the file.

Character Streams: Readers and Writers

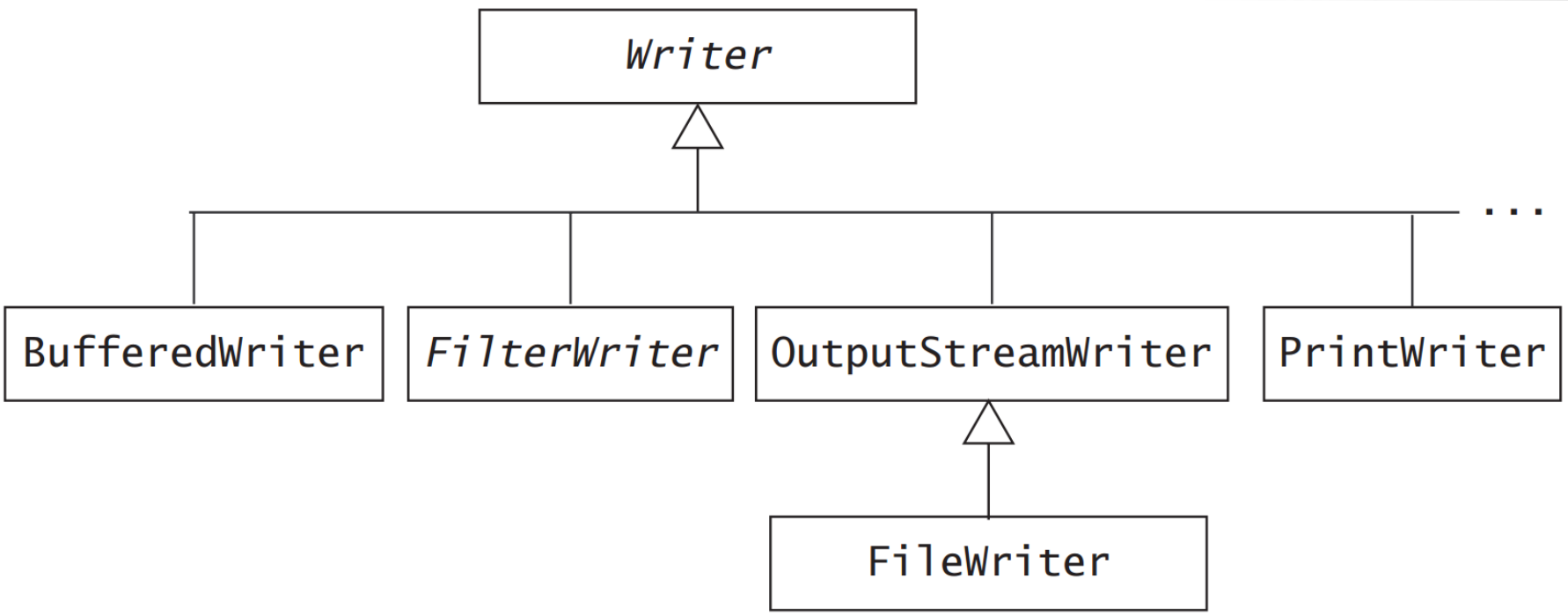


НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Selected Writers

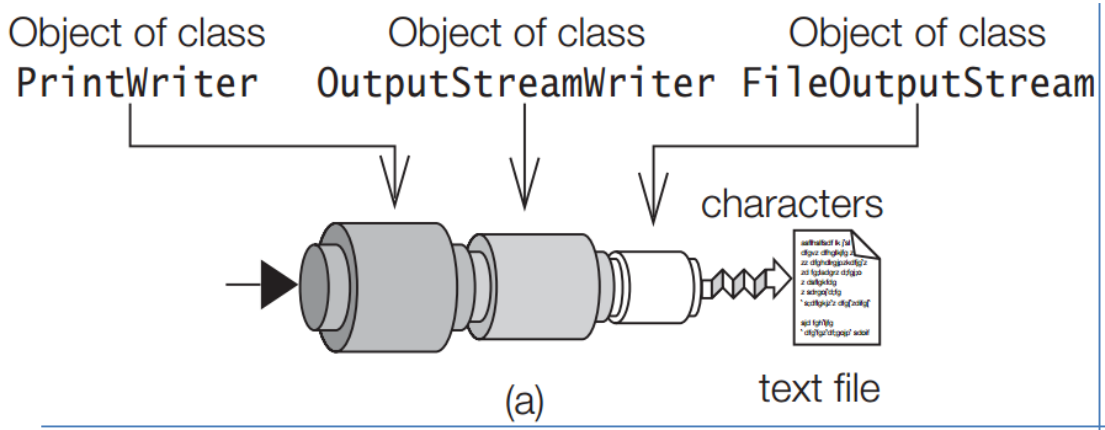
BufferedWriter	A writer that buffers the characters before writing them to an underlying writer. The underlying writer must be specified, and an optional buffer size can be specified.
OutputStreamWriter	Characters are written to a byte output stream which must be specified. The default character encoding is used if no explicit character encoding is specified.
FileWriter	Writes characters to a file, using the default character encoding. The file can be specified by a File object, a FileDescriptor, or a String file name. It automatically creates a FileOutputStream that is associated with the file
PrintWriter	A filter that allows text representation of Java objects and Java primitive values to be written to an underlying output stream or writer. The underlying output stream or writer must be specified.

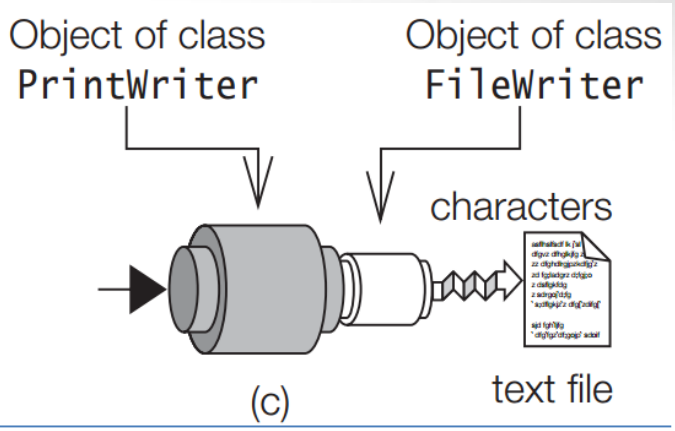
Character Streams: Readers and Writers

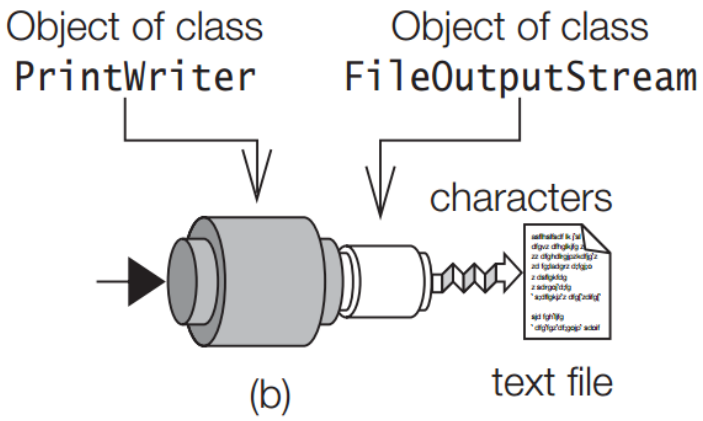


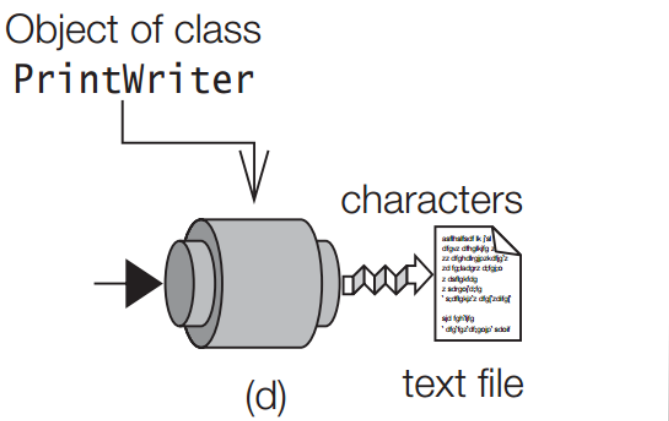
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Writing Text Files



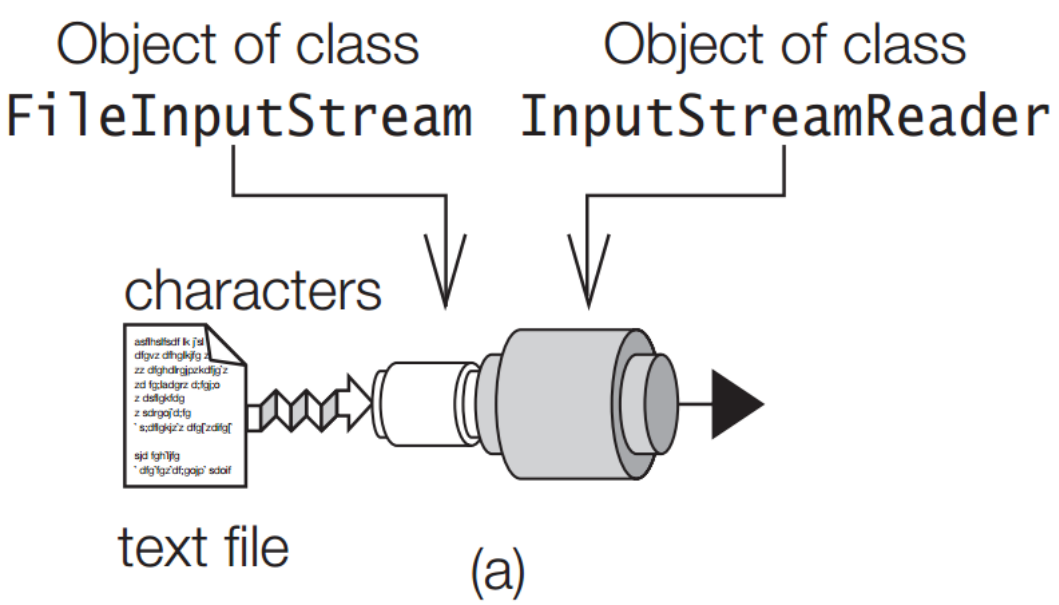


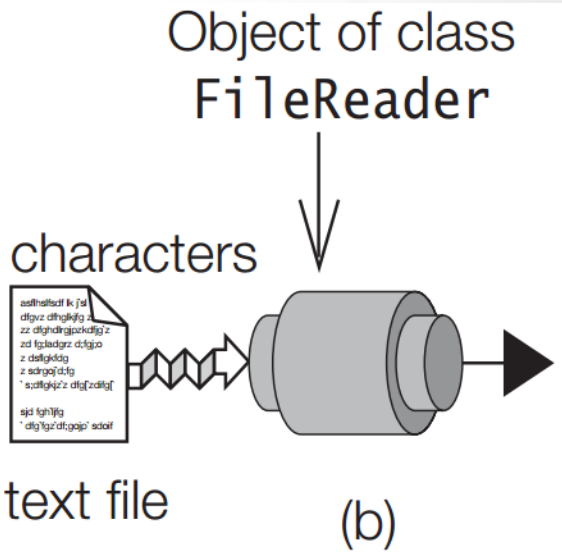




НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

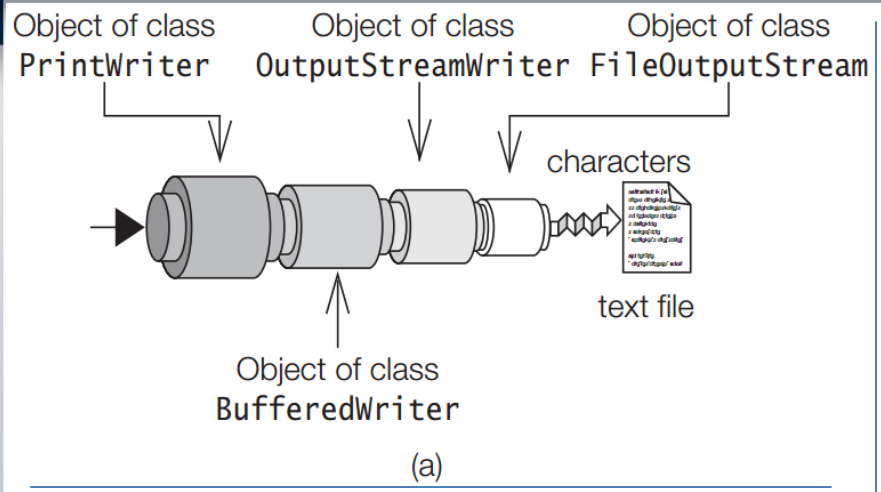
Reading Text Files



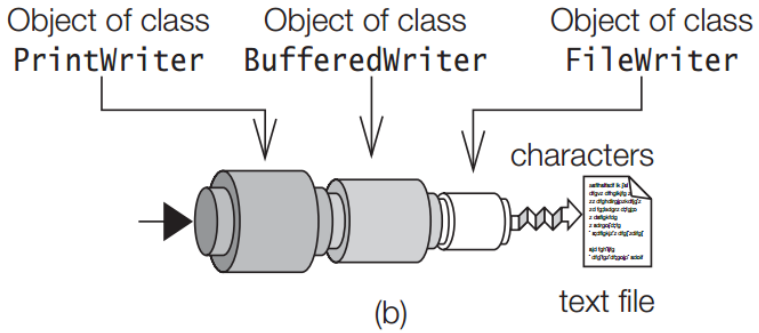


НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

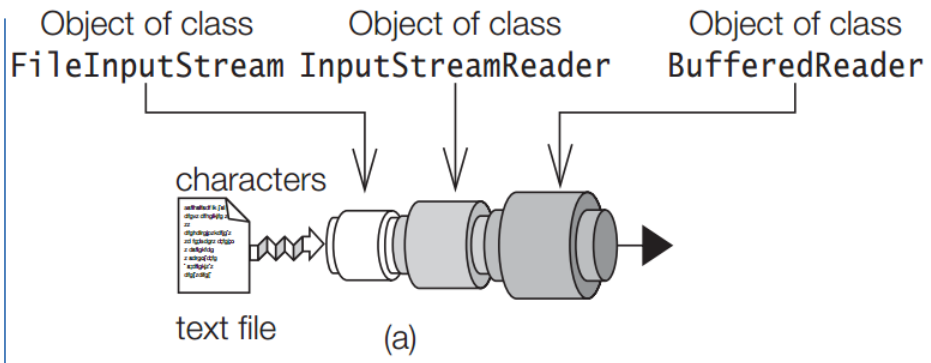
Using Buffered Writers & Readers



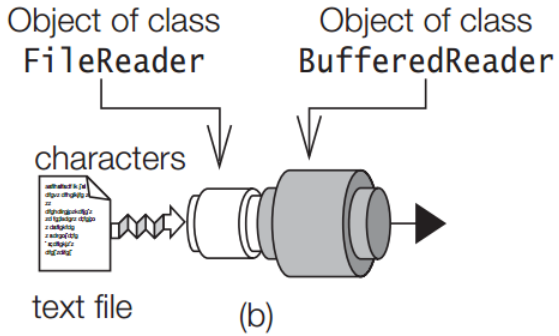
(a)



(b)



(a)



(b)

Object Serialization

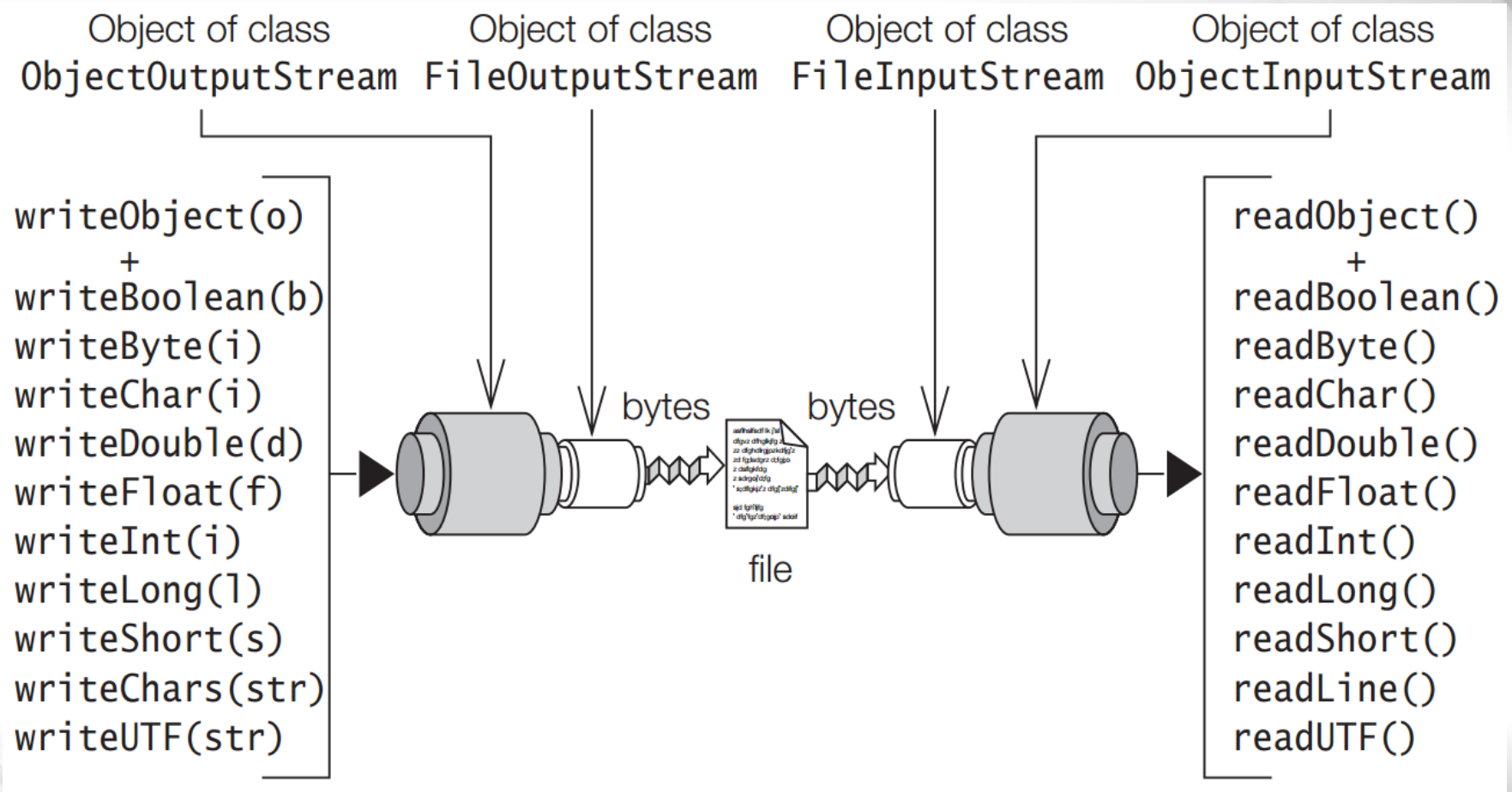
- *Object serialization* allows an object to be transformed into a sequence of bytes that can later be re-created (*deserialized*) into the original object.
- After deserialization, the object has the same state as it had when it was serialized, barring any data members that were not serializable.
- This mechanism is generally known as *persistence*.
- Java provides this facility through the ObjectInput and ObjectOutput interfaces, which allow the reading and writing of objects from and to streams.

Object Serialization

- The ObjectOutputStream class and the ObjectInputStream class implement the ObjectOutput interface and the ObjectInput interface, respectively, providing methods to write and read binary representation of objects as well as Java primitive values.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Object Stream Chaining



Marker Interface

- The marker interface pattern used with languages that provide run-time type information about objects. It provides a means to associate metadata with a class where the language does not have explicit support for such metadata.
- To use this pattern, a class implements a marker interface, and methods that interact with instances of that class test for the existence of the interface.
- Whereas a typical interface specifies functionality (in the form of method declarations) that an implementing class must support, a marker interface need not do so.

Serializable

- An example of the application of marker interfaces is the Serializable interface.
- A class implements this interface to indicate that its non-transient data members can be written to an ObjectOutputStream.
- The ObjectOutputStream method writeObject() contains a series of instanceof tests to determine writability, one of which looks for the Serializable interface.
- If any of these tests fails, the method throws a NotSerializableException

Example

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Questions?

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Object-Oriented Programming in the Java language

Part 5. Exceptions. I/O in Java

Yevhen Berkunskyi, NUoS
eugeny.berkunsky@gmail.com
http://www.berkut.mk.ua