## Serialisierung eines Objekts in Java – Interface Serializable

Das Interface Serializable erlaubt es bei der Ausführung, dass ein Objekt beim Senden (egal ob zu einem Permanentspeicher oder über eine Datenleitung) in eine Folge von Bytes verwandelt wird und beim Empfangen/Lesen daraus wieder ein Objekt erzeugt werden kann.

Ein Objekt, das dieses Interface implementiert, sieht aus wie die nebenstehende Klasse **BikeData**.

**public class BikeData implements Serializable {**

**private String description, type;**

**private int bId, wheelSize, frameHeight;**

**public BikeData(int bId, String description,**

**String type, int wheelSize, int frameHeight) {**

**...**

**}**

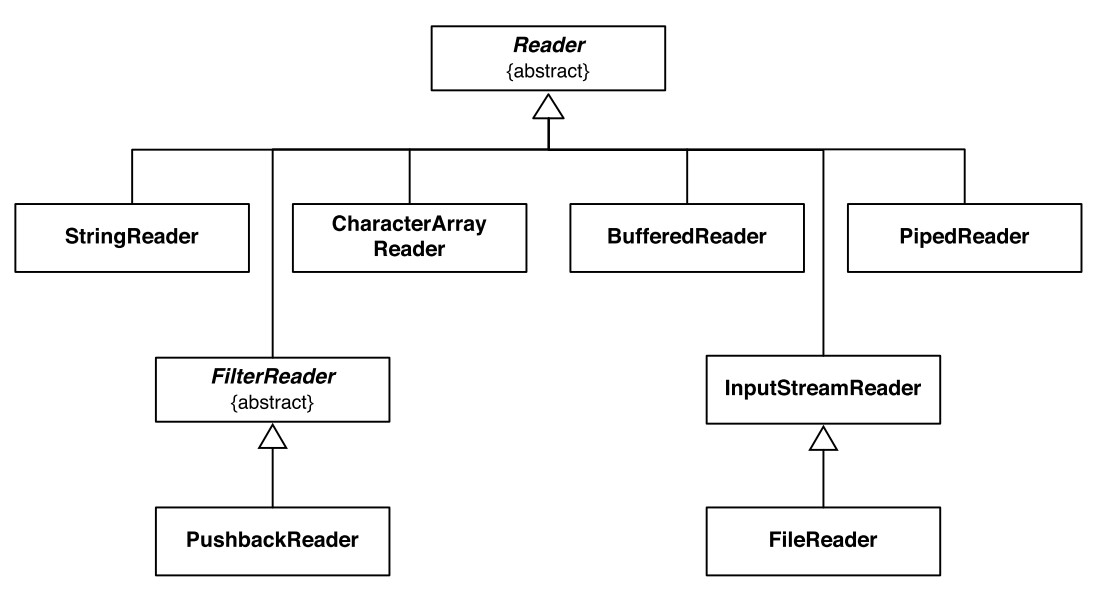
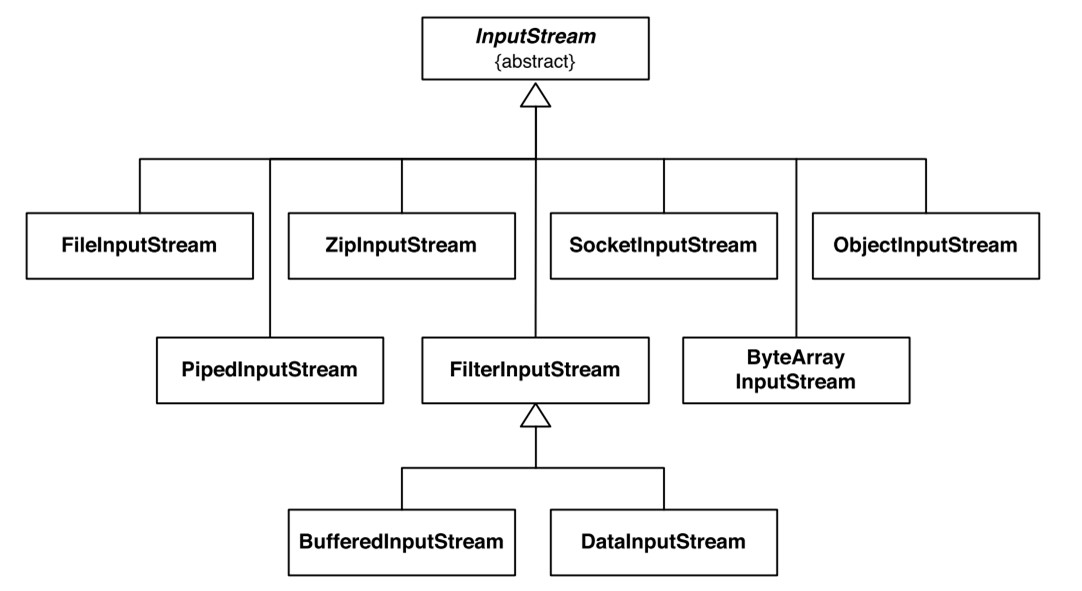
**/\* nur Getter ggfs. Setter \*/**

**}**

Im Interface gibt es keine Methoden, die zu implementieren wären.

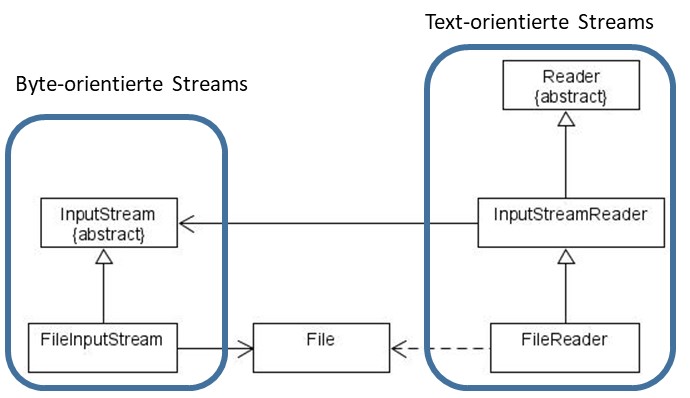
Die eigentliche Umwandlung des Objekts erfolgt durch Klassen der Bibliothek java.io. Nachfolgend sind die Übersichten über die Klassenhierarchien der Streams und der Reader/Writer dargestellt.

Zur Vereinfachung wurde jeweils nur die Hierarchie der InputStreams und der Reader dargestellt. Mit wenigen Ausnahmen existiert zu jedem InputStream/Reader eine entsprechende OutputStream/Writer-Klasse.



Die Klassen der Streams sind zum Lesen/Schreiben von byte-orientierten Daten vorgesehen, die Reader/Writer für zeichenorientierte Ein-/Ausgabe (Text).

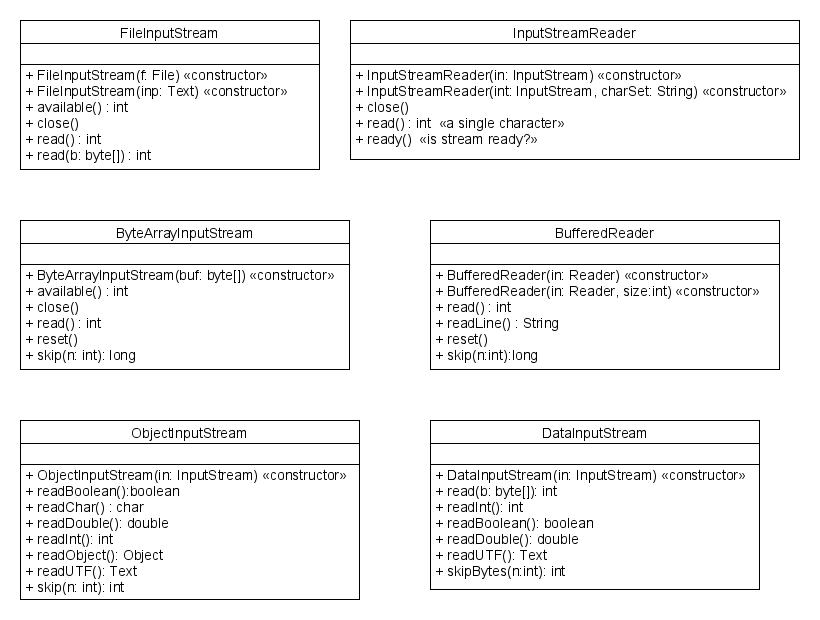
Im Folgenden sind Klassendiagramme einiger ausgewählter InputStream/OutputStream sowie Reader/Writer-Subklassen dargestellt.

Die Klassendiagramme sind nicht vollständig, sondern auszugsweise dargestellt. Sofern Attribute in den Klassen existieren, wurden sie weggelassen, da sie für die Aufgabenstellung ebenso wenig relevant sind wie die ausgelassenen Methoden.

Letztlich zeigen diese Konstruktoren bereits einen Weg auf, wie die einzelnen Klassen miteinander zu benutzen sind.

Ein Zusammenhang zwischen den Byte-orientierten Stream-Klassen und den zeichenorientierten Streams zeigt die nebenstehende Skizze. Der FileReader verwendet intern die Klasse File.

Einige Klassen zur Eingabe (InputStream / Reader), mit denen die gestellten Aufgaben gelöst werden können.



**Ein Beispiel zum Lesen von Textdateien.**

File f = **new** File("sicherung.txt");

**try** {

FileInputStream fis = **new** FileInputStream(f);

InputStreamReader isr = **new** InputStreamReader(fis);

**while** (isr.ready()) {

System.***out***.print((**char**) isr.read());

}

isr.close();

} **catch** (IOException e) {

System.***err***.println(e);

}

Wie man an dem Konstruktor von InputStreamReader erkennen kann, akzeptiert er als Parameter alle Subklassen von InputStream. Letztlich liegt es beim Benutzer, diese sinnvoll zu kombinieren.

Einige Klassen zur Ausgabe (OutputStream / Writer), mit denen die gestellten Aufgaben gelöst werden können.

