Day15 Java NIO Features

**📘 Java IO, NIO.2, and HttpClient**

**Part 1: Java I/O**

**🔹 Core Concepts**

* Stream-based I/O
* Byte vs Character streams
* Buffered I/O
* Serialization & Deserialization

**🔹 Code Example – File Copy (Classic I/O)**

import java.io.\*;

public class FileCopyIO {

public static void main(String[] args) {

try (FileInputStream fis = new FileInputStream("input.txt");

FileOutputStream fos = new FileOutputStream("output.txt")) {

byte[] buffer = new byte[1024];

int bytesRead;

while ((bytesRead = fis.read(buffer)) != -1) {

fos.write(buffer, 0, bytesRead);

}

System.out.println("File copied using Java IO.");

} catch (IOException e) {

e.printStackTrace();

}

}

}

**🔹 Real-world Use Case**

* Reading log files, writing reports
* File import/export functionality

**Part 2: Java NIO.2**

**🔹 Core Concepts**

* Non-blocking I/O
* Channels, Buffers, Selectors
* File operations using java.nio.file package
* AsynchronousFileChannel

**🔹 Code Example – File Copy (NIO.2)**

import java.nio.file.\*;

public class FileCopyNIO2 {

public static void main(String[] args) {

Path source = Paths.get("input.txt");

Path target = Paths.get("output.txt");

try {

Files.copy(source, target, StandardCopyOption.REPLACE\_EXISTING);

System.out.println("File copied using NIO.2");

} catch (Exception e) {

e.printStackTrace();

}

}

}

**🔹 Example – Asynchronous Read**

import java.nio.ByteBuffer;

import java.nio.channels.AsynchronousFileChannel;

import java.nio.file.\*;

import java.util.concurrent.Future;

public class AsyncReadExample {

public static void main(String[] args) throws Exception {

Path path = Paths.get("input.txt");

AsynchronousFileChannel channel = AsynchronousFileChannel.open(path);

ByteBuffer buffer = ByteBuffer.allocate(1024);

Future<Integer> result = channel.read(buffer, 0);

while (!result.isDone()) {

System.out.println("Reading asynchronously...");

}

buffer.flip();

while (buffer.hasRemaining()) {

System.out.print((char) buffer.get());

}

channel.close();

}

}

**🔹 Real-world Use Case**

* Large file processing (logs, media)
* Concurrent file access (async read/write)

**Part 3: Java HttpClient**

**🔹 Core Concepts**

* Introduced in Java 11 (java.net.http)
* Supports synchronous & asynchronous requests
* Handles HTTP/1.1, HTTP/2
* Built-in JSON handling with BodyHandlers

**🔹 Code Example – Simple GET Request**

import java.net.http.\*;

import java.net.URI;

public class SimpleHttpGet {

public static void main(String[] args) throws Exception {

HttpClient client = HttpClient.newHttpClient();

HttpRequest request = HttpRequest.newBuilder()

.uri(new URI("https://jsonplaceholder.typicode.com/posts/1"))

.build();

HttpResponse<String> response = client.send(request, HttpResponse.BodyHandlers.ofString());

System.out.println("Status Code: " + response.statusCode());

System.out.println("Response: " + response.body());

}

}

**🔹 Example – Async POST Request**

import java.net.URI;

import java.net.http.\*;

import java.net.http.HttpRequest.BodyPublishers;

public class AsyncPostExample {

public static void main(String[] args) throws Exception {

HttpClient client = HttpClient.newHttpClient();

String json = "{\"title\":\"Java\",\"body\":\"HttpClient Example\",\"userId\":1}";

HttpRequest request = HttpRequest.newBuilder()

.uri(new URI("https://jsonplaceholder.typicode.com/posts"))

.header("Content-Type", "application/json")

.POST(BodyPublishers.ofString(json))

.build();

client.sendAsync(request, HttpResponse.BodyHandlers.ofString())

.thenApply(HttpResponse::body)

.thenAccept(System.out::println)

.join();

}

}

**🔹 Real-world Use Case**

* REST API integration
* Microservice communication
* Async calls for performance

**Part 4: Quiz + Hands-on Lab**

**🔹 Quiz (5 Questions)**

1. Difference between FileInputStream and BufferedInputStream?
2. What is the advantage of NIO.2 over traditional I/O?
3. What are Channels and Buffers in NIO?
4. Which class is used for asynchronous file read/write?
5. Which package provides HttpClient in Java 11+?

**🔹 Hands-on Lab**

👉 **Problem Statement:**  
Build a program that:

1. Reads a text file (input.txt) line by line (Java I/O).
2. Copies it to another file (backup.txt) using NIO.2.
3. Makes a **POST request** to a REST API (https://jsonplaceholder.typicode.com/posts) sending the file contents as JSON using HttpClient.
4. Prints the response from the API.

Day14 Revisit

Executor Framework – It’s a way to implement Concurrency in JAVA.

Parallel processing with the help of multiple Threads is called Concurrency.

Executor Framework API (Set of Interfaces, Abstract Classes & Impl. Classes)

Java.util.concurrent (package)

Executor

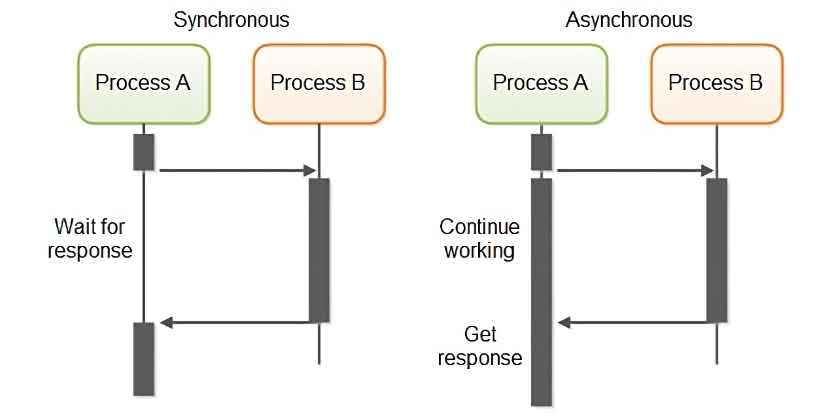
ExecutorService

1. SingleThreaded
2. FixedThreadPool
3. CachedThread
4. ScheduledThread

Threads enable Parallel processing. (Async Operation) – Non-Blocking Operations (Reactive)

Serial Operation (Sync Operation) – Blocking Operations.

Blocking vs Non-Blocking Operation



Web based Applications (Client – Server) [Request & Response Objects]

IO Operations (Input/Output Operations)

Java is running inside a Virtual Machine. This JVM is relaying on underlying System Architecture.

Java is platform independent whereas JDK, JRE & JVM are all platform dependent.

Two way Data flow in JVM

1. From User/File to the Program (Input)
2. From Program/Application to the User/Console/File (Output)

IO Related API are available in java.io apackage

Three Stream with respect to IO operations

Input Stream – System.in

Output Stream – System.out

Error Stream – System.err

Byte & Character (2 bytes) .

1 byte = 8 bits. [0-255] [-128 – 127] - 27

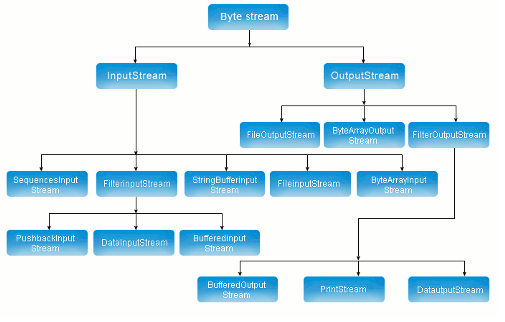
2 bytes = 16 bits [ 215]

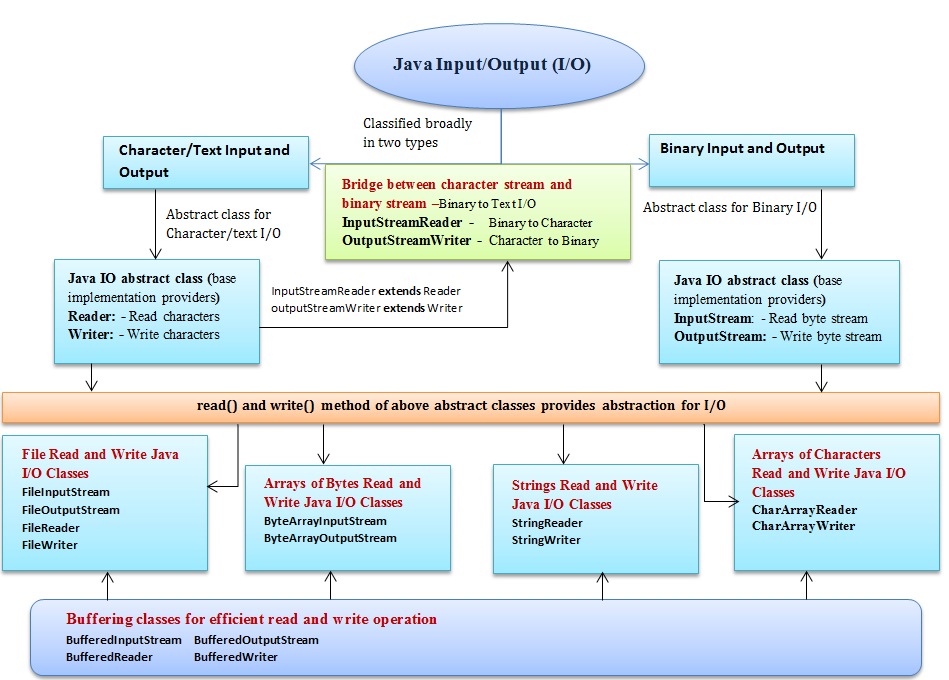
Basic IO operations are done by two methods.

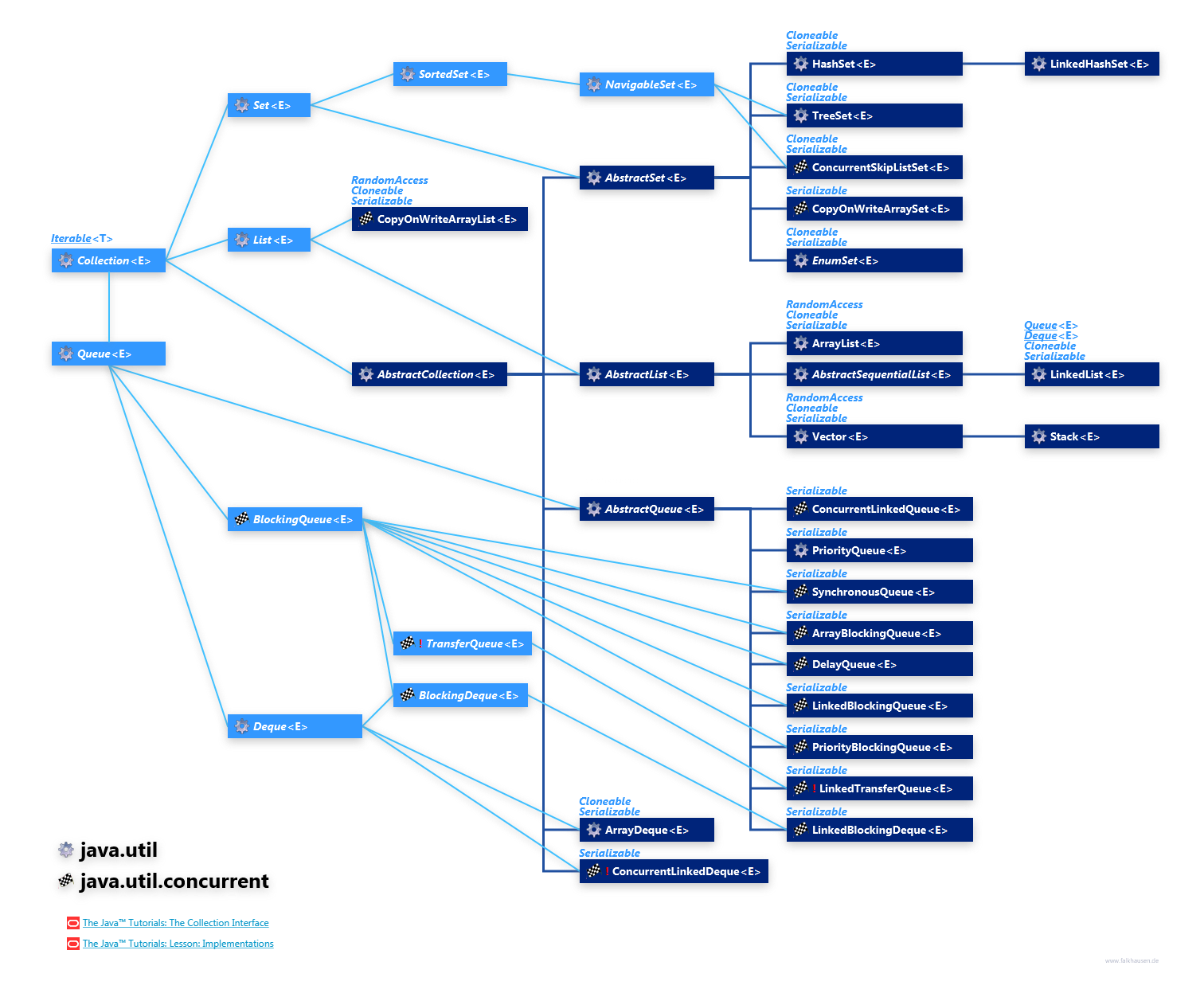
1. Byte Stream based approach (Input/Output Stream)
2. Character Stream based approach. (Reader/Writer)

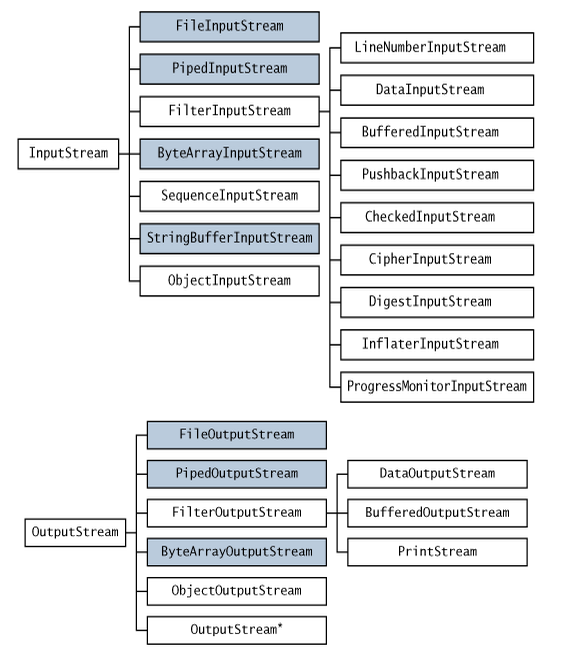
26\*2 (A-Z, a-z) – ASCII (8 bits)

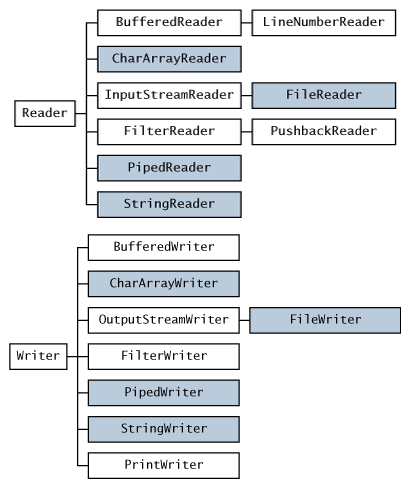
247 – Unicode Character (UTF – Unicode Text Format) (16-bits)











**package** com.example;

**import** java.io.FileInputStream;

**import** java.io.IOException;

**import** java.io.InputStream;

**public** **class** IOStreamsDemo {

**public** **static** **void** main(String[] args) **throws** IOException {

// **TODO** Auto-generated method stub

InputStream ios = **new** FileInputStream("d:\\input.txt");

**byte**[] data = ios.readAllBytes();

System.***out***.println("File contents");

**for** (**byte** b : data)

System.***out***.print((**char**) b+" ");

}

}

**package** com.example;

**import** java.awt.Taskbar.Feature;

**import** java.io.FileReader;

**import** java.io.IOException;

**public** **class** FileReaderDemo {

**public** **static** **void** main(String[] args) **throws** IOException {

// **TODO** Auto-generated method stub

FileReader fr = **new** FileReader("d:\\input.txt");

**int** d = 0;

**do** {

d = fr.read();

**if** (d != -1)

System.***out***.print((**char**) d);

} **while** (d != -1);

}

}

**package** com.example;

**import** java.io.FileInputStream;

**import** java.io.FileNotFoundException;

**import** java.io.FileOutputStream;

**import** java.io.IOException;

**import** java.io.InputStream;

**import** java.io.OutputStream;

**public** **class** OutputStreamDemo {

**public** **static** **void** main(String[] args) **throws** IOException {

// **TODO** Auto-generated method stub

InputStream ios = **new** FileInputStream("d:\\input.txt");

**byte**[] data = ios.readAllBytes();

OutputStream os = **new** FileOutputStream("d:\\output.txt");

os.write(data);

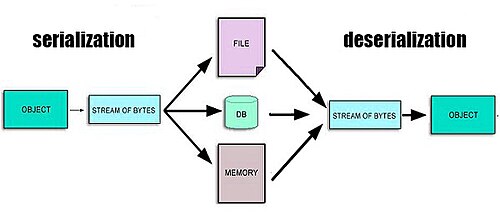
System.***out***.println("File contents are copied!!!");

}

}

Async Operations Example ( Using NIO – New IO features)

Serialization - Process of converting the state of an object to Stream of strings/character so that it can be stored in flat file.



**package** com.example;

**import** java.io.Serializable;

**public** **class** Employee **implements** Serializable{

**private** **int** id;

**private** String name;

**private** String email;

**public** Employee(**int** id, String name, String email) {

**super**();

**this**.id = id;

**this**.name = name;

**this**.email = email;

}

**public** Employee() {

**super**();

}

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getEmail() {

**return** email;

}

**public** **void** setEmail(String email) {

**this**.email = email;

}

@Override

**public** String toString() {

**return** "Employee [id=" + id + ", name=" + name + ", email=" + email + "]";

}

}

**package** com.example;

**import** java.io.File;

**import** java.io.FileNotFoundException;

**import** java.io.FileOutputStream;

**import** java.io.IOException;

**import** java.io.ObjectOutputStream;

**public** **class** SerializationDemo {

**public** **static** **void** main(String[] args) **throws** IOException {

// **TODO** Auto-generated method stub

Employee emp = **new** Employee(100, "Test","test@gmail.com");

System.***out***.println(emp);

FileOutputStream fos = **new** FileOutputStream("d:\\emp\_object.ser");

ObjectOutputStream oos = **new** ObjectOutputStream(fos);

oos.writeObject(emp);

System.***out***.println("Emp object is serialized!!!");

}

}

**package** com.example;

**import** java.io.FileInputStream;

**import** java.io.IOException;

**import** java.io.ObjectInputStream;

**public** **class** DeserializationDemo {

**public** **static** **void** main(String[] args) **throws** IOException, ClassNotFoundException {

// **TODO** Auto-generated method stub

FileInputStream fis = **new** FileInputStream("d:\\emp\_object.ser");

ObjectInputStream ois = **new** ObjectInputStream(fis);

System.***out***.println((Employee) ois.readObject());

System.***out***.println("Successfully deserialised Employee Object and reconstructed it using serialised file");

}

}

HttpClient – It’s a program helps to interact with any Web Server using Http (hypertext transfer protocol)