// I/O - kopiowanie OS do IS

**public** **static** **void** copy(InputStream in, OutputStream out) **throws** IOException {

**final** **int** BLOCKSIZE = 1024;

**byte**[] bytes = **new** **byte**[BLOCKSIZE];

**int** len;

**while** ((len = in.read(bytes)) != -1)// odczytuje kolejne porcje wielkości BLOCKSIZE

out.write(bytes, 0, len); // zapisueje do stream całą tablicę (od 0 do len)

}

// I/O - odczytanie wszystkich bajtów z pliku

**public** **static** **byte**[] readAllBytes(InputStream in) **throws** IOException {

ByteArrayOutputStream out = **new** ByteArrayOutputStream();

*copy*(in, out);

out.close();

**return** out.toByteArray();

}

// Budowa obiektu path

**public** **static** **void** IOPath() {

// Ĺšcieżka nie jest walidowana

Path p = Paths.*get*("...");

// Budowanie ścieżki

Path absolute = Paths.*get*("/", "home", "cay1");

Path relative = Paths.*get*("myapp", "conf", "user.properties");

// Dokleja ścieżkę, czyli będzie /home/cay/myapp/work

Path workPath1 = absolute.resolve("myapp/work");

// Dokleja sibling /home/myapp/work

// UWAGA! Dal ścieżek Linux jest problem z rozpoczynającym "\" i poniższe da

// błąd

// Paths.get("/", "sampleFolder").resolveSibling("myapp/work")

Path workPath2 = absolute.resolveSibling("myapp/work");

// Relatywizujemy 2 wzglęcem 1 - ../freud/myapp - pastępije cześć wspólną

// (obu ścieżek) wielokotopkami

// Mówiąc inaczej opisuje jak ze ścięzki A dośc do ścieżki B

Path workPath3 = Paths.*get*("/home/cay1/cay2/cay3").relativize(Paths.*get*("/home/cay1/fred/myapp"));

**int** brk = 0;

}

// Przelanie bajtow z InputStram do tablicy

**public** **static** **void** IOBytesToArr() **throws** IOException {

// wczytywanie bez dodatkowych opcji

**byte**[] byteArr1 = *readAllBytes*(Files.*newInputStream*(Paths.*get*("C:/2.txt")));

// dodanie opcji (czy zakładać nowe pliki etc)

OpenOption[] oos = **new** OpenOption[] { /\* tu można wstawić opcje \*/ };

**byte**[] byteArr2 = *readAllBytes*(Files.*newInputStream*(Paths.*get*("C:/2.txt"), oos));

// można skrócić implementację

**byte**[] byteArr3 = *readAllBytes*(Files.*newInputStream*(Paths.*get*("C:/2.txt"), **new** OpenOption[] {}));

}

// Odczytanie pojedynczego znaku z IS (w tym przypadku z pliku)

**public** **static** **void** IOByteToVar() **throws** IOException {

**int** inp;

InputStream inStream = Files.*newInputStream*(Paths.*get*("C:/2.txt"), **new** OpenOption[] { ***CREATE\_NEW*** });

// Stworzenie readera dla inputStream

Reader in = **new** InputStreamReader(inStream);

inp = in.read(); // Read robi jednocześnie Next

**char**[] chars = **new** **char**[5];

in.read(chars, 2, 3); // tablica docelowa, index od którego zacząć wstawianie, ile wstawić -

// [,,1,2,]

}

// Wczytanie całego pliku do Stringa -

// UWAGA!! Plik nie może być duży

**public** **static** **void** IOToString() **throws** IOException {

String content = **new** String(Files.*readAllBytes*(Paths.*get*("C:/2.txt")));

}

// Wczytanie pojedynczych słów z pliku

**public** **static** **void** IOWordToVar() **throws** IOException {

**int** intWord;

Scanner scn = **new** Scanner(Paths.*get*("C:/2.txt"));

scn.next();// pomija słowo

intWord = scn.nextInt();// to da Exception jeśli kolejne "słowo" nie będzie liczbą

}

// Zapis gotowego stringa (cały tekst) do pliku

**public** **static** **void** IOStringToFile() **throws** IOException {

String content2 = ".......\n%%%%%";

Files.*write*(Paths.*get*("D:/6.txt"), content2.getBytes(StandardCharsets.***UTF\_8***));

}

// Wczytanie wszystkich linii pliku do listy Stringów

**public** **static** **void** IOLinesToStringList() **throws** IOException {

// Za pomocą Files

List<String> lines1 = Files.*readAllLines*(Paths.*get*("C:/2.txt"));

// Za pomocą streama 1

List lineList1;

**try** (Stream<String> lines2 = Files.*lines*(Paths.*get*("C:/2.txt"))) {

lineList1 = Arrays.*asList*(lines2.toArray());

}

// Za pomocą streama 2

List lineList2;

**try** (Stream<String> lines2 = Files.*lines*(Paths.*get*("C:/2.txt"))) {

lineList2 = lines2.collect(Collectors.*toList*());

}

// Za pomocą BufferedReader

// try (BufferedReader reader = new BufferedReader(new

// InputStreamReader(url.openStream()) )) {

**try** (BufferedReader reader = **new** BufferedReader(

**new** InputStreamReader(Files.*newInputStream*(Paths.*get*("C:/2.txt"))))) {

Stream<String> lines = reader.lines();

// lines.collect(...)

// lub

reader.readLine();

}

}

// Wczytanie pliku jako mapy z listami słów

**public** **static** **void** IOWordsToMap() **throws** IOException {

**try** (BufferedReader reader = **new** BufferedReader(

**new** InputStreamReader(Files.*newInputStream*(Paths.*get*("C:/2.txt"))))) {

Stream<String> lines = reader.lines();

AtomicInteger lpline = **new** AtomicInteger();

Map<Integer, List<String>> map = lines.peek(System.***out***::println).

// zbudowanie mapy. Key to rosnący licznik a value to lista stringów

// stanowiących słowa w danej liście

collect(Collectors.*toMap*(s -> lpline.getAndIncrement(), s -> Arrays.*asList*(s.split("[^\\S]+"))));

}

}

// Writer do pliku - proste metody

**public** **static** **void** IOBufferedWriter() **throws** IOException {

Writer out1 = Files.*newBufferedWriter*(Paths.*get*("D:/3.txt"), StandardCharsets.***UTF\_8***, **new** OpenOption[] { ***CREATE*** });

out1.write("1 111");

out1.write("1 222");

out1.flush();

}

// Writer do pliku - dodatkowo ma print i println i pozwala formatować tekst

**public** **static** **void** IOPrintWriter() **throws** IOException {

PrintWriter out2 = **new** PrintWriter(

Files.*newBufferedWriter*(Paths.*get*("D:/4.txt"), StandardCharsets.***UTF\_8***, **new** OpenOption[] { ***CREATE*** }));

out2.println("2 111");

out2.println("2 222");

out2.flush();

}

// Writer do streama - w tym przypadku stream do pliku

**public** **static** **void** IOOutputStreamWriter() **throws** IOException {

// Wybieramy zapis do pliku

OutputStream outStream = Files.*newOutputStream*(Paths.*get*("D:/5.txt"), **new** OpenOption[] { ***CREATE*** });

// Tu mamy papis na konsolę

// OutputStream outStream2 = System.out;

Writer out2 = **new** OutputStreamWriter(outStream, StandardCharsets.***UTF\_8***);

out2.write("3 111\n");

out2.write("3 222");

out2.flush();

}

// Zapis kolecji linii do pliku - to może być cokolwiek co jest

// Collection<String> lub nawet Iterable<? extends CharSequence>.

**public** **static** **void** IOManyStringsToFile() **throws** IOException {

List<String> lines = Arrays.*asList*("1\n2", "3"); // zapisza się 3 linie

Files.*write*(Paths.*get*("D:/7.txt"), lines, StandardCharsets.***UTF\_8***);

}

// IO danych inaczej niż jako String - jest szybszy

**public** **static** **void** IODataInputStream() **throws** IOException {

DataInput in2 = **new** DataInputStream(Files.*newInputStream*(Paths.*get*("D:/7.txt")));

DataOutput out2 = **new** DataOutputStream(Files.*newOutputStream*(Paths.*get*("D:/7.txt")));

}

// Dostęp do dowolnej cześci pliku

**public** **static** **void** IORandomAccessFile() **throws** IOException {

**int** i = 1;

RandomAccessFile file = **new** RandomAccessFile("D:/7.txt", "rw");

// file.readLine();//wczytanie całej linii (i robi next)

// file.read();//oddaje kod znaku (i robi next), np 1 da 49

// file.readInt();//oddaje dziwne rzeczy - nie rozumiem tego

// file.seek(i);//przesówa karetę na wskazaną pozycję po początku pliku

file.writeChar('a');

// Poniższy przyklad nie dział. może kodowanie jest nierodpowiednie, albo

// cioś innego.

// mechanizm jest mało potrzebny, więc nie będę się w to zagłębiał

// Przykład ++ dla integera w pliku

// int value = file.readInt();

// file.seek(file.getFilePointer() - 4);

// file.writeInt(value + 1);

}

// Dostęp do dużych plików - możemy zmapować fragment pliku i tylko w tym

// fragmencie wprowadać zmiany

**public** **static** **void** IOBigFileFileChannel() **throws** IOException {

// Totalnie nie wiem jednak jak tego używać ???

FileChannel channel = FileChannel.*open*(Paths.*get*("D:/3.txt"), ***READ***, StandardOpenOption.***WRITE***);

//Poniższe działa poprawnie, czyli 2 opuszcza znak o indeksach 0 i 1 i zaczyna zapis na znaku o indeksie 2

ByteBuffer buffer = channel.map(FileChannel.MapMode.***READ\_WRITE***, /\* start \*/0, /\* długość bloku \*/channel.size());

**int** offset = 0;

**byte** value = buffer.get(offset); // oddaje bajtową rezprezentrację znaku, czyli dla 0 odda "49"

// nie wiem co dokładnie oddaje get/putInt etc

// int value = buffer.getInt(offset);

// buffer.putInt(offset, 3);

}

// Blokada - czeka dopoki nie można założyć blokady

**public** **static** **void** BlokadaStop() **throws** IOException {

// Jeśli program (np Office) zablokował plik, to ta linia rzuci

// java.nio.file.FileSystemException

// Jest możliwość, że pewne aplikacje (np natywne) mają możliwość na

// ostrzejsze.

// W takim przypadku nie można liczyc na to że dojdziemy do locka, ale trzeba

// badać rzycane wyjątki

FileChannel channel2 = FileChannel.*open*(Paths.*get*("D:/3.txt"), ***READ***, StandardOpenOption.***WRITE***);

FileLock lock1 = channel2.lock(); // Tutaj wątek stanie i będzie czekać dopóki nie założy blokady

**int** i = 0;

}

// Blokada - czeka dopoki nie można założyć blokady

**public** **static** **void** BlokadaTry() **throws** IOException {

FileChannel channel = FileChannel.*open*(Paths.*get*("D:/3.txt"), ***READ***, StandardOpenOption.***WRITE***);

FileLock lock2;

**try** {

lock2 = channel.tryLock();

} **catch** (OverlappingFileLockException e) {

// Ten wyjątek wystapi jeśli ten program już zablokował ten plik

}

**int** i = 0;

**try** (FileLock lock3 = channel.lock()) {

i = 0;

}

}

// Z path dostajemy obiekt File

**public** **static** **void** IOPathToFile() {

File file = (Paths.*get*("D:", "3.txt")).toFile();

**if** (file.exists()) {

/\* .... \*/}

;

}

// Tworzenie plików i katalogów

**public** **static** **void** IOCreate() **throws** IOException {

/\*

\* W java Directory i File mają jedną klasę. Odróżniają się po

\* isDirectory()

\*/ **if** (((Paths.*get*("D:", "test2")).toFile()).isDirectory())

((Paths.*get*("D:", "test2")).toFile()).delete();

Path dirPath = Files.*createDirectories*(Paths.*get*("D:", "test1") /\* , Attributes \*/);

File dir = (Paths.*get*("D:", "test1")).toFile();

/\*\*/ **if** (((Paths.*get*("D:", "test2")).toFile()).isFile())

((Paths.*get*("D:", "test2")).toFile()).delete();

Files.*createFile*(Paths.*get*("D:", "test2"));

// Tworzenie tymczasowego pliku i katalogi

Path tempFile = Files.*createTempFile*(Paths.*get*("D:/"), "MY\_", ".log"); // D:/MY\_462324563250.log

Path tempDir = Files.*createTempDirectory*(Paths.*get*("D:"), "MY\_");

Path tempDir2 = Files.*createTempDirectory*("MY\_"); // katalog tymczasowy w domyślnej lokalizacji

// (...)/Local/Temp/MY\_(...)

**int** brk = 0;

}

// Kopiowanie i przenoszenie

**public** **static** **void** IOCopyMove() **throws** IOException {

// Kopiowanie

Files.*copy*(Paths.*get*("D:/3.txt"), Paths.*get*("D:/33.txt"), StandardCopyOption.***REPLACE\_EXISTING***,

StandardCopyOption.***COPY\_ATTRIBUTES***);

/\* Przeniesienie \*/ Files.*copy*(Paths.*get*("D:/3.txt"), Paths.*get*("D:/34.txt"), StandardCopyOption.***REPLACE\_EXISTING***,

StandardCopyOption.***COPY\_ATTRIBUTES***);

Files.*move*(Paths.*get*("D:/34.txt"), Paths.*get*("D:/35.txt"), StandardCopyOption.***REPLACE\_EXISTING***);

// Prosty delete - rzuca wyjątkiem

Files.*delete*(Paths.*get*("D:/33.txt"));

// Delete - nie rzyca wyjątkiem, ale oddaje info o wykonaniu działania

**boolean** deleted = Files.*deleteIfExists*(Paths.*get*("D:/35.txt"));

**int** brk = 0;

}

// Parsowanie struktury katalogów (pliki i katalogi)

**public** **static** **void** IOParseDir() **throws** IOException {

// Płaska lista pliów i katalogów

**try** (Stream<Path> entries1 = Files.*list*(Paths.*get*("C:\\"))) {

entries1.forEach(System.***out***::println);

}

// Wgłąb lista pliów i katalogów

**try** (Stream<Path> entries2 = Files.*walk*(Paths.*get*("C:", "Situ"), 2/\* max głębokość \*/)) {

entries2.forEach(System.***out***::println);

}

}

// URL wymaga innego Źródła informacji niż książka - tu jest bieda

// Tu tylko podstawy pobierania informacji

**public** **static** **void** IOURL() **throws** IOException {

URL google = **new** URL("http://www.google.com");

URLConnection connection = google.openConnection();

// connection.setRequestProperty("Accept-Charset", "UTF-8, ISO-8859-1");//

// można pominąć

// connection.setReadTimeout(1000);

// connection.setConnectTimeout(1000);

BufferedReader reader = **new** BufferedReader(**new** InputStreamReader(google.openStream()));

reader.lines().forEach(System.***out***::println);

}