# Reti e Laboratorio III
# Modulo Laboratorio III
# AA. 2023-2024

docente: Laura Ricci

laura.ricci@unipi.it

# Correzione Assignment 8
## "Valutazione strategie I/O bufferizzato"
## 23/11/2023

# ASSIGNMENT 8: I/O BUFFERIZZATO

- scopo dell'assignment è dare una valutazione delle prestazioni di diverse strategie di bufferizzazione di I/O offerte da JAVA
- scrivere un programma che copi un file di input in un file di output, utilizzando le seguenti modalità alternative di bufferizzazione, valutando il tempo impiegato per la copia del file in ognuna delle seguenti strategie:
  - `FileChannel` con buffer indiretti
  - `FileChannel` con buffer diretti
  - `FileChannel` utilizzando l'operazione transferTo()
  - `Buffered Stream di I/O`
  - stream letto in un byte-array gestito dal programmatore

- confrontare le prestazioni delle diverse soluzioni, variando la dimensione del file (da qualche kbyte fino ad almeno una decina di Megabyte) e la dimensione del buffer

# ASSIGNMENT 8: I/O BUFFERIZZATO

```
/**
 * Reti e Laboratorio III - A.A. 2023/2024
 * Soluzione dell'ottavo assignment
 *
 * Questo programma valuta le prestazioni di diversi metodi di copia di file
 * basati su Java NIO e IO tradizionale.
 *
 * Il programma prende in input tre parametri:
 * (1) il percorso di una directory da cui prelevare i file da copiare;
 * (2) il nome di un file di testo su cui scrivere i risultati;
 * (3) la dimensione del buffer da utilizzare (in byte).
 *
 */
```

# ASSIGNMENT 8: LETTURA INPUT E FILE

```
public class Main {
    public static int bufSize;
    public static final int numTrials = 5;
    public static final String tempFile = "temp";

    public static void main(String[] args) {
        if (args.length < 3) {
            System.err.println("Usage: Main <inputDir> <outputFile> <bufSize>");
            System.exit(1);
        }
        String inputDir = args[0], outputFile = args[1];
        bufSize = Integer.parseInt(args[2]);
        File dir = new File(inputDir);
        if (!dir.exists() || !dir.isDirectory()) {
            System.err.printf("%s non e' una directory valida.\n", dir.getName());
            System.exit(1);
        }
```

# ASSIGNMENT 8: LETTURA INPUT E FILE

```
try (PrintWriter out = new PrintWriter(outputFile)) {
    out.printf("filename,size,NIO_ind,NIO_dir,NIO_transfer,IO_buf,IO_custom\n");
    // Leggo tutti i file contenuti nella directory di input
    // e li ordino in base alla loro dimensione (in modo crescente).
    File[] files = dir.listFiles();
    Arrays.sort(files, (x, y) -> Long.compare(x.length(), y.length()));
```

# ASSIGNMENT 8: COPIE CON NIO

```
for (int i = 0; i < files.length; i++) {
    String filename = files[i].getName();
    long size = files[i].length();
    System.out.printf("Sto copiando il file: %s\n", filename);
    long nioInd = 0, nioDir = 0, nioTransfer = 0, ioBuf = 0, ioCustom = 0;
    // Primo metodo: NIO indirect buffer.
    for (int j = 0; j < numTrials; j++) nioInd += nioIndirectCopy(files[i]);
    nioInd /= numTrials;

    // Secondo metodo: NIO direct buffer.
    for (int j = 0; j < numTrials; j++) nioDir += nioDirectCopy(files[i]);
    nioDir /= numTrials;

    // Terzo metodo: NIO transferTo().
    for (int j = 0; j < numTrials; j++) nioTransfer += nioTransferCopy(files[i]);
    nioTransfer /= numTrials;
```

# ASSIGNMENT 8: COPIE CON I/O CLASSICO

```
    // Quarto metodo: I/O tradizionale con buffered stream.
    for (int j = 0; j < numTrials; j++) ioBuf += ioBufferedCopy(files[i]);
    ioBuf /= numTrials;
    // Quinto metodo: I/O tradizionale con file stream e array di byte.
    for (int j = 0; j < numTrials; j++) ioCustom += ioCustomCopy(files[i]);
    ioCustom /= numTrials;
    // Scrivo la riga con i tempi sul file di output.
    out.printf("%s,%d,%d,%d,%d,%d,%d\n", filename, size, nioInd, nioDir, nioTransfer,
ioBuf, ioCustom);
    }
    System.out.printf("Ho finito!");
}
catch (Exception e) {
    System.err.printf("Errore: %s\n", e.getMessage());
    e.printStackTrace();
    System.exit(1);
    }
}
```

# ASSIGNMENT 8: NIO - BUFFER INDIRETTI

```java
/**
 * Copia un file usando NIO e buffer indiretti.
 * @param inputFile file di input (da copiare)
 * @return tempo impiegato dalla procedura
 * @throws IOException in caso di errore in lettura/scrittura
 */
public static long nioIndirectCopy(File inputFile) throws IOException {
    long start = System.nanoTime();
    // NOTA: in questo metodo e nei successivi il file di input
    // verra' copiato in un file temporaneo, il quale a propria volta
    // sara' automaticamente cancellato al termine della JVM.
    File outputFile = File.createTempFile(tempFile, null);
    outputFile.deleteOnExit();
```

# ASSIGNMENT 8: NIO - BUFFER INDIRETTI

```
    FileChannel in = FileChannel.open(inputFile.toPath(), StandardOpenOption.READ);
    FileChannel out = FileChannel.open(outputFile.toPath(), StandardOpenOption.WRITE);
    ByteBuffer buffer = ByteBuffer.allocate(bufSize);
    while (in.read(buffer) > 0) {
        buffer.flip();
        out.write(buffer);
        buffer.clear();
    }
    in.close();
    out.close();
    long end = System.nanoTime();
    return end-start;
}
```

# ASSIGNMENT 8: NIO - BUFFER DIRETTI

```java
/**
 * Copia un file usando NIO e buffer diretti.
 * @param inputFile file di input (da copiare)
 * @return tempo impiegato dalla procedura
 * @throws IOException in caso di errore in lettura/scrittura
 */
public static long nioDirectCopy(File inputFile) throws IOException {
    long start = System.nanoTime();
    // NOTA: in questo metodo e nei successivi il file di input
    // verra' copiato in un file temporaneo, il quale a propria volta
    // sara' automaticamente cancellato al termine della JVM.
    File outputFile = File.createTempFile(tempFile, null);
    outputFile.deleteOnExit();
```

# ASSIGNMENT 8: NIO - BUFFER DIRETTI

```
    FileChannel in = FileChannel.open(inputFile.toPath(), StandardOpenOption.READ);
    FileChannel out = FileChannel.open(outputFile.toPath(), StandardOpenOption.WRITE);
    ByteBuffer buffer = ByteBuffer.allocateDirect(bufSize);
    while (in.read(buffer) > 0) {
        buffer.flip();
        out.write(buffer);
        buffer.clear();
    }
    in.close();
    out.close();
    long end = System.nanoTime();
    return end-start;
}
```

# ASSIGNMENT 8: NIO - transferTo()

```
/**
 * Copia un file usando NIO e il metodo transferTo()
 * @param inputFile file di input (da copiare)
 * @return tempo impiegato dalla procedura
 * @throws IOException in caso di errore in lettura/scrittura
 */
public static long nioTransferCopy(File inputFile) throws IOException {
    long start = System.nanoTime();
    File outputFile = File.createTempFile(tempFile, null);
    outputFile.deleteOnExit();
    FileChannel in = FileChannel.open(inputFile.toPath(), StandardOpenOption.READ);
    FileChannel out = FileChannel.open(outputFile.toPath(), StandardOpenOption.WRITE);
    in.transferTo(0, in.size(), out);
    in.close();
    out.close();
    long end = System.nanoTime();
    return end-start;
}
```

# ASSIGNMENT 8: I/O + BUFFEREDSTREAM

```
public static long ioBufferedCopy(File inputFile) throws IOException {
    long start = System.nanoTime();
    File outputFile = File.createTempFile(tempFile, null);
    outputFile.deleteOnExit();
    BufferedInputStream in = new BufferedInputStream(new FileInputStream(inputFile));
    BufferedOutputStream out = new BufferedOutputStream(
        new FileOutputStream(outputFile) );
    byte[] buf = new byte[bufSize];
    int numRead = 0;
    while ((numRead = in.read(buf)) > 0) out.write(buf, 0, numRead);
    in.close();
    out.close();
    long end = System.nanoTime();
    return end-start;
}
```

# ASSIGNMENT 8: I/O + STREAM / BYTE ARRAY

```java
/**
 * Copia un file usando I/O tradizionale, file stream e un array di byte.
 * @param inputFile file di input (da copiare)
 * @return tempo impiegato dalla procedura
 * @throws IOException in caso di errore in lettura/scrittura
 */
public static long ioCustomCopy(File inputFile) throws IOException {
    long start = System.nanoTime();
    File outputFile = File.createTempFile(tempFile, null);
    outputFile.deleteOnExit();
    FileInputStream in = new FileInputStream(inputFile);
    FileOutputStream out = new FileOutputStream(outputFile);
    byte[] buf = new byte[bufSize];
    int numRead = 0;
    while ((numRead = in.read(buf)) > 0) out.write(buf, 0, numRead);
    in.close();
    out.close();
    long end = System.nanoTime();
    return end-start;
}
```