

# I/O (STRUMIENIE, PLIKI, ...)

## ZAGADNIENIA:

- pakiet `java.nio.file`,
- operacje na plikach,
- swobodny dostęp do plików,
- katalogi,
- archiwa.
- jar i manifest.

## MATERIAŁY:

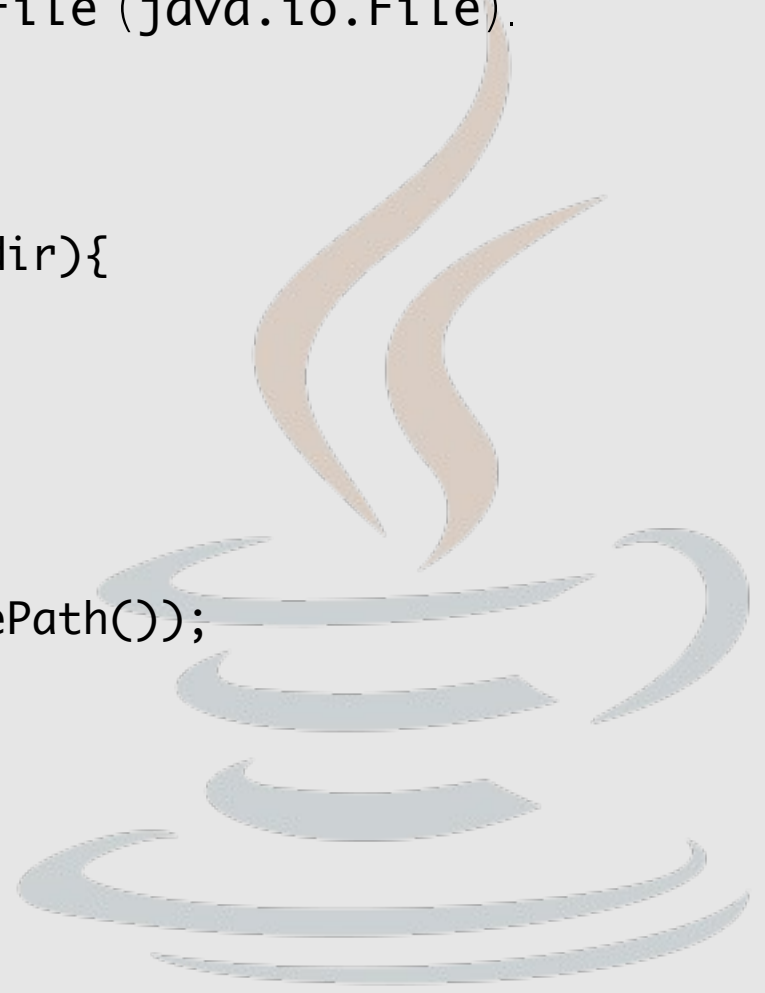
<http://docs.oracle.com/javase/tutorial/essential/io/>



# OPERACJE NA PLIKACH

Podstawowa klasa reprezentująca plik to **File** (`java.io.File`).

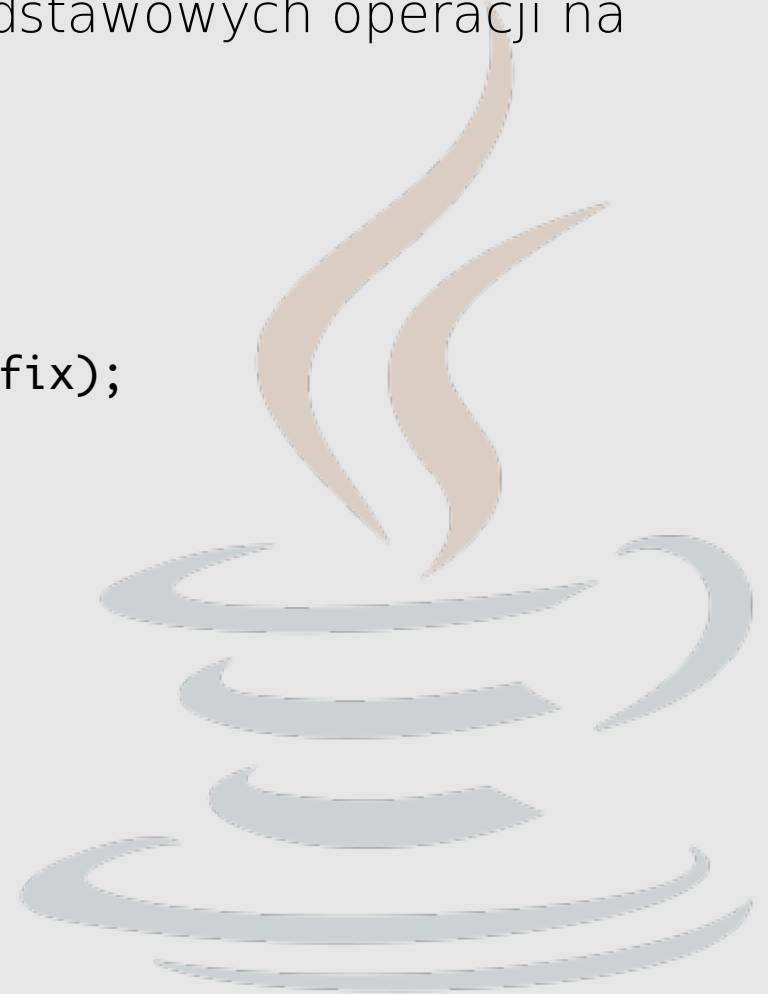
```
import java.io.File;
public class FileTest {
    public static void listDirectory(File fdir){
        File[] fa = fdir.listFiles();
        if (fa==null) return;
        for(File f: fdir.listFiles()){
            if (f.isDirectory())
                listDirectory(f);
            System.out.println(f.getAbsolutePath());
        }
    }
    public static void main(String[] args){
        File f = new File("/");
        listDirectory(f);
    }
}
```



# OPERACJE NA PLIKACH

Klasa **File** umożliwia wykonanie wielu podstawowych operacji na plikach np.:

- `exists();`
- `createNewFile();`
- `createTempFile(String prefix, String suffix);`
- `delete();`
- `deleteOnExit();`
- `renameTo(File dest);`
- `mkdirs();`
- `getParentFile();`
- `toPath();`



# OPERACJE NA PLIKACH

Począwszy od Javy 1.7, bardziej zaawansowane operacje plikowe są dostępne poprzez klasy pakietu **java.nio.file**. Plik jest tutaj reprezentowany przez interfejs **Path**.

```
Path p1 = Paths.get("/tmp/file");
```

```
Path p2 = FileSystems.getDefault().getPath("/tmp/file");
```

```
Path p3 = Paths.get(URI.create("file:///tmp/file"));
```

lub

```
Path p = (new File("/tmp/file")).toPath();
```

# OPERACJE NA PLIKACH

Podstawowe operacje na plikach wykonujemy używając metod statycznych klasy **Files**. Przykładowy dostęp atrybutów pliku:

```
Path file = ...;
BasicFileAttributes attr =
    Files.readAttributes(file, BasicFileAttributes.class);
System.out.println("creationTime: " + attr.creationTime());
System.out.println("lastAccessTime: " + attr.lastAccessTime());
System.out.println("lastModifiedTime: " + attr.lastModifiedTime());
System.out.println("isDirectory: " + attr.isDirectory());
System.out.println("isOther: " + attr.isOther());
System.out.println("isRegularFile: " + attr.isRegularFile());
System.out.println("isSymbolicLink: " + attr.isSymbolicLink());
System.out.println("size: " + attr.size());
```

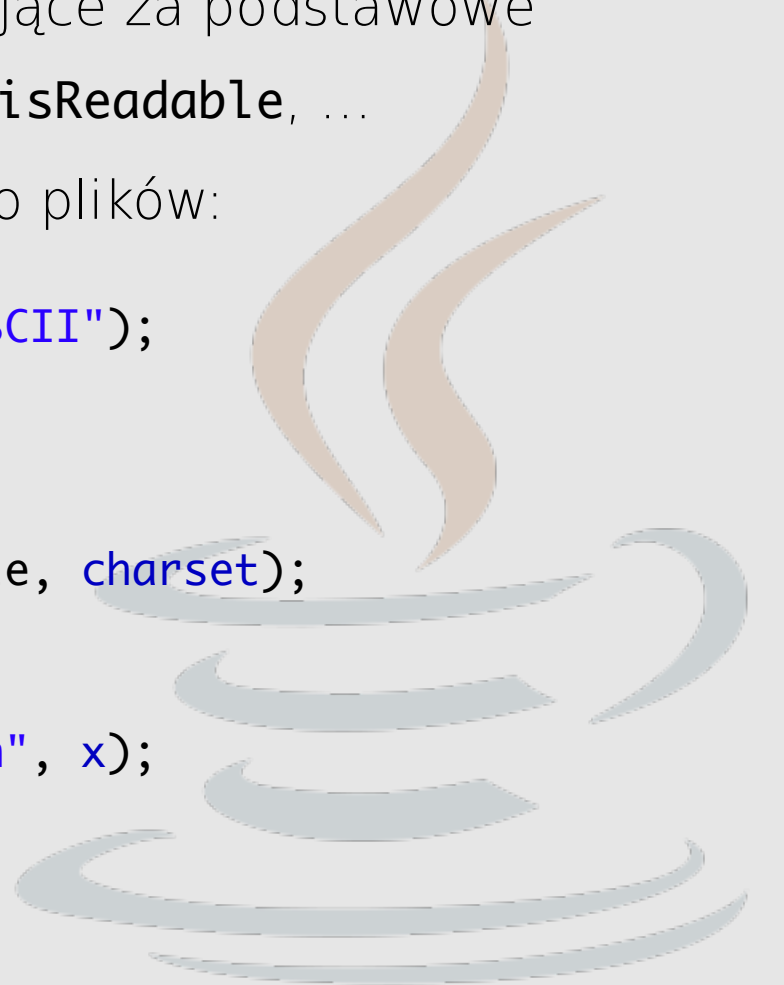
Można także używać **DosFileAttributes** lub **PosixFileAttributes**.

# DOSTĘP DO PLIKÓW

Klasa posiada również metody odpowiadające za podstawowe operacje na plikach: **exists**, **move**, **delete**, **isReadable**, ...

Można ją wykorzystać także do dostępu do plików:

```
Charset charset = Charset.forName("US-ASCII");
String s = ...;
BufferedWriter writer = null;
try {
    writer = Files.newBufferedWriter(file, charset);
    writer.write(s, 0, s.length());
} catch (IOException x) {
    System.err.format("IOException: %s%n", x);
} finally {
    if (writer != null) writer.close();
}
```



# DOSTĘP DO PLIKÓW

Dostęp do niewielkich plików:

- odczyt

```
Path file = ...;  
byte[] fileArray;  
fileArray = Files.readAllBytes(file);
```

- zapis:

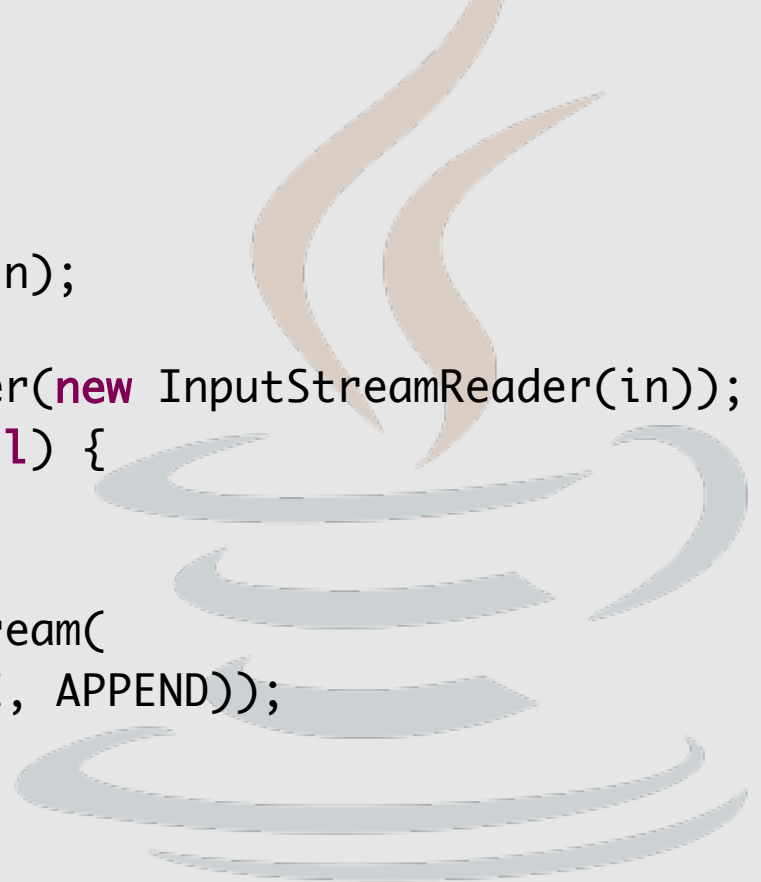
```
Path file = ...;  
byte[] buf = ...;  
Files.write(file, buf);
```

sposób działania metody write można ustawić za pomocą dodatkowych flag. Domyślnie plik jest tworzony lub nadpisywany.

# DOSTĘP DO PLIKÓW

Pliki i strumienie bajtowe:

```
Path fin=..., fout=...;
String line = null;
byte data[] = "Ala ma kota".getBytes();
try{
    InputStream in = Files.newInputStream(fin);
    BufferedReader reader =
        new BufferedReader(new InputStreamReader(in));
    while ((line = reader.readLine()) != null) {
        System.out.println(line);
    }
    OutputStream out = new BufferedOutputStream(
        fout.newOutputStream(CREATE, APPEND));
    out.write(data, 0, data.length);
} catch (IOException x) {
    System.err.println(x);
}
```

Stylizowane logo Java, składające się z trzech falistych linii w kolorze szarym i brązowym, które tworzą kształt przypominający parującą kawę.



# DOSTĘP DO PLIKÓW - KANAŁY

Strumienie odczytują/zapisują "na raz" bajt lub znak. Kanały (Channels) odczytują/zapisują "na raz" bufor, co umożliwia swobodny dostęp do pliku.

Path file;

// domyślny kanał ma flagę READ

```
try (SeekableByteChannel sbc = Files.newByteChannel(file)) {  
    ByteBuffer buf = ByteBuffer.allocate(10);  
    // odczyt zgodny z domyślnym kodowaniem  
    String encoding = System.getProperty("file.encoding");  
    while (sbc.read(buf) > 0) {  
        buf.rewind();  
        System.out.print(Charset.forName(encoding).decode(buf));  
        buf.flip();  
    }  
} catch (IOException x) { System.out.println("caught exception: " + x); }
```

# SWOBODNY DOSTĘP DO PLIKÓW

Interfejs **SeekableByteChannel** udostępnia metody pozwalające na swobodny dostęp do plików:

- **position()** - zwraca bieżącą pozycję,
- **position(long)** - ustawia pozycję,
- **read(ByteBuffer)** - odczytuje dane z kanału,
- **write(ByteBuffer)** - zapisuje dane do kanału,
- **truncate(long)** - nadpisuje plik połączony do kanału.

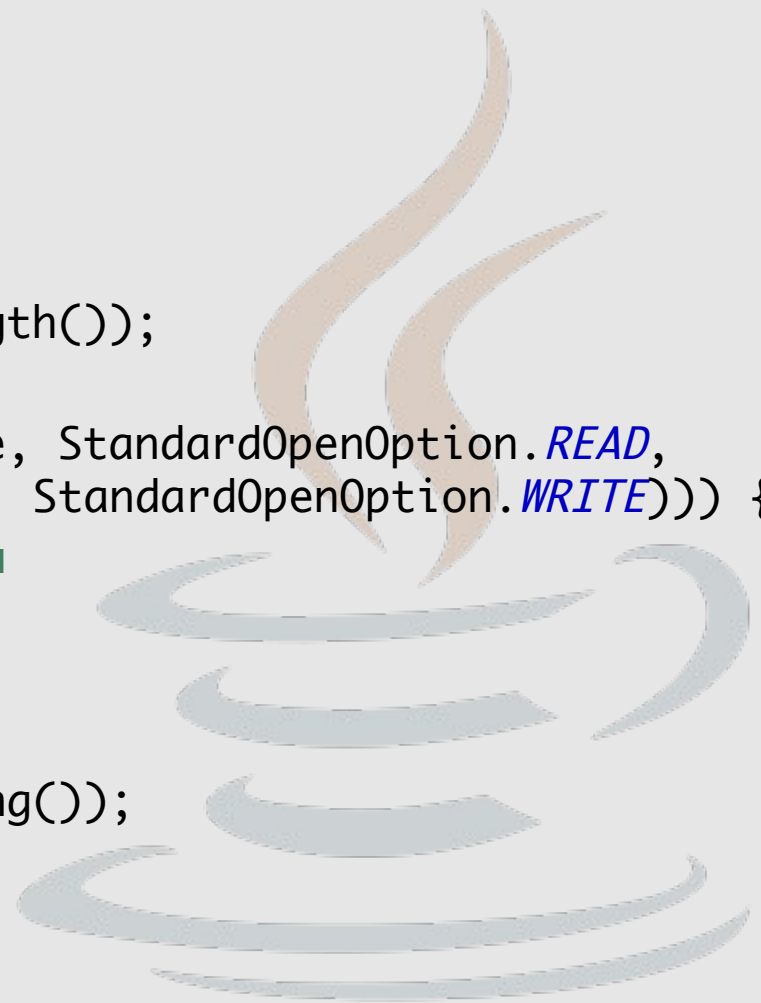
Kanał utworzony metodą **Path.newByteChannel()** można zrzutować na **FileChannel** dający więcej możliwości (np. odwzorowania pliku w pamięci w celu np. przyspieszenia dostępu)

# SWOBODNY DOSTĘP DO PLIKÓW

```
Path file = Paths.get("plik.txt");  
String s = "tu bylem!";  
byte data[] = s.getBytes();
```


```
ByteBuffer out = ByteBuffer.wrap(data);  
ByteBuffer copy = ByteBuffer.allocate(s.length());
```

```
try (FileChannel fc = (FileChannel.open(file, StandardOpenOption.READ,  
                                         StandardOpenOption.WRITE))) {  
    // Odczyt początkowych 10 bajtów z pliku  
    int nread;  
    do {  
        nread = fc.read(copy);  
    } while (nread != -1 && copy.hasRemaining());
```



# SWOBODNY DOSTĘP DO PLIKÓW

```
// zapis "tu bylem!" na początku pliku (nadpisuje pierwsze 10 bajtów)
fc.position(0);
while (out.hasRemaining())
    fc.write(out);
out.rewind();
// przesunięcie na koniec pliku.
fc.position(fc.size() - 1);
// zapisanie kopii na końcu pliku
copy.flip();
while (copy.hasRemaining())
    fc.write(copy);
// zapisanie "tu bylem!" na końcu pliku
while (out.hasRemaining())
    fc.write(out);
} catch (IOException x) {
    System.out.println("I/O Exception: " + x);
}
}
```



# KATALOGI

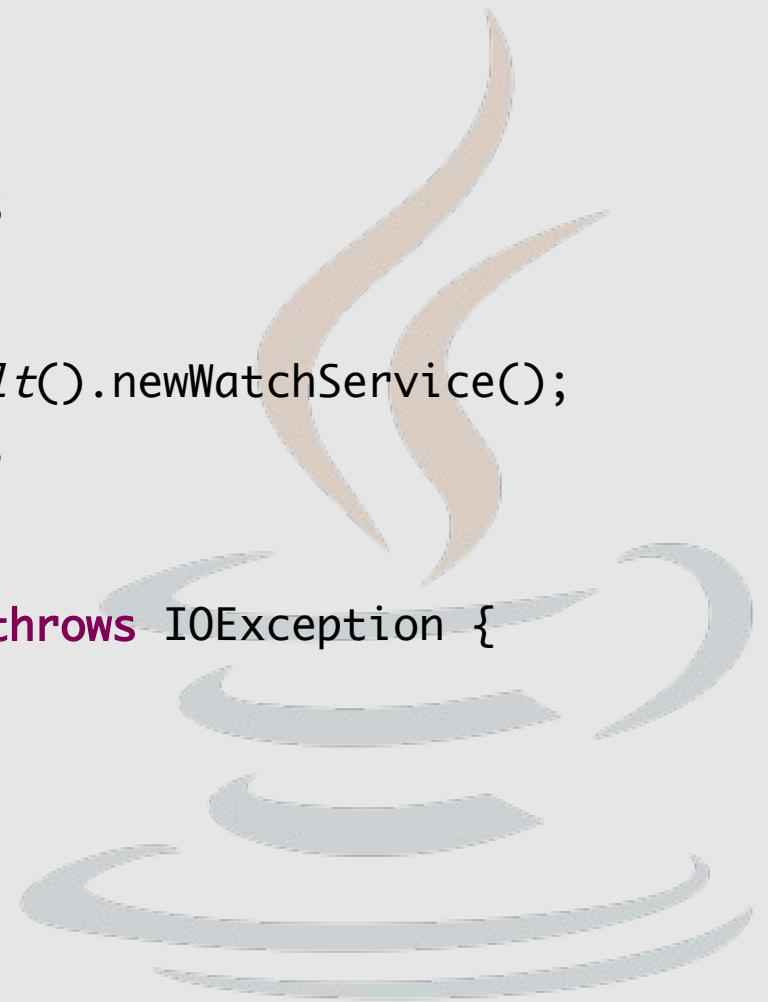
```
Iterable<Path> dirs = FileSystems.getDefault().getRootDirectories();  
for (Path name: dirs)  
    System.err.println(name);
```

```
Path dir = ...; // listujemy tylko pliki *.java, *.class i *.jar  
try (DirectoryStream<Path> stream =  
    Files.newDirectoryStream(dir, "*. {java,class,jar}")) {  
    for (Path entry: stream)  
        System.out.println(entry.getFileName());  
} catch (IOException x) { System.err.println(x); }
```

```
Path dir = ...;  
Files.createDirectory(dir); // tworzymy katalog  
Set<PosixFilePermission> perms =  
    PosixFilePermissions.fromString("rwxr-x---");  
FileAttribute<Set<PosixFilePermission>> attr =  
    PosixFilePermissions.asFileAttribute(perms);  
Files.createDirectory(file, attr); // tworzymy katalog i nadajemy mu prawa
```

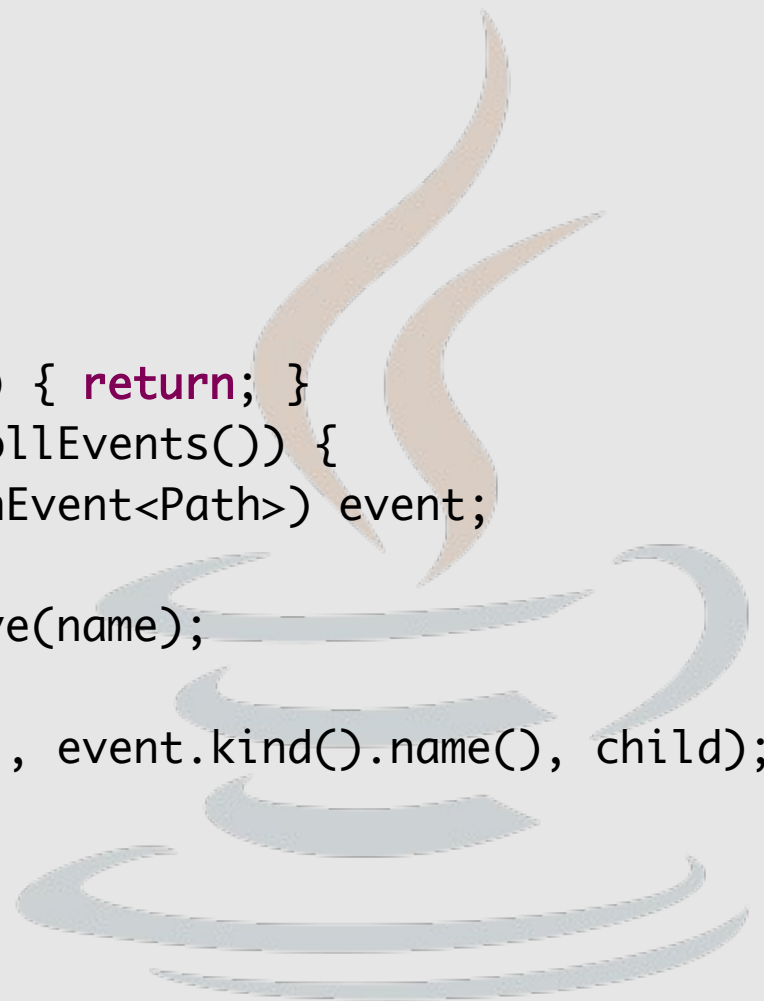
# MONITOROWANIE KATALOGÓW

```
public class WatchDir {  
  
    private final WatchService watcher;  
    private final Path dir = Paths.get(".");  
  
    public WatchDir() throws IOException {  
        this.watcher = FileSystems.getDefault().newWatchService();  
        dir.register(watcher, ENTRY_CREATE);  
    }  
  
    public static void main(String[] args) throws IOException {  
        // oczekujemy na zdarzenia  
        new WatchDir().processEvents();  
    }  
}
```



# MONITOROWANIE KATALOGÓW

```
void processEvents() {  
    while(true){  
        // wait for key to be signalled  
        WatchKey key;  
        try {  
            key = watcher.take();  
        } catch (InterruptedException x) { return; }  
        for (WatchEvent<?> event: key.pollEvents()) {  
            WatchEvent<Path> ev = (WatchEvent<Path>) event;  
            Path name = ev.context();  
            Path child = this.dir.resolve(name);  
  
            System.out.format("%s: %s\n", event.kind().name(), child);  
        }  
        key.reset();  
    }  
}
```



# ARCHIWA

Java implementuje strumienie realizujące kompresję danych:

```
GZIPOutputStream gos = new GZIPOutputStream(  
    new FileOutputStream("plik.gz"));  
for (int i = 0; i < 10000; i++)  
    gos.write("Ala ma kota".getBytes());  
gos.close();  
  
ZipOutputStream zos = new ZipOutputStream(  
    new FileOutputStream("plik.zip"));  
for(int i=0; i<5; i++){  
    ZipEntry ze = new ZipEntry("plik" + i);  
    zos.putNextEntry(ze);  
    for(int j=0; j< 1000; j++)  
        zos.write("Ala ma kota".getBytes());  
    zos.closeEntry();  
}  
zos.close();
```



# ARCHIWA JAR

Java wyróżnia także szczególny rodzaj archiwum ZIP: JAR (JarOutputStream, JarInputStream). Archiwa JAR zawierają pliki klas wraz z dodatkowymi zasobami potrzebnymi do działania aplikacji. Podstawowe zalety dystrybucji programów w postaci plików **jar** to:

- bezpieczeństwo: archiwa mogą być cyfrowo podpisywane,
- kompresja: skrócenie czasu ładowania apletu lub aplikacji,
- zarządzanie zawartością archiwów z poziomu języka Java,
- zarządzanie wersjami na poziomie pakietów oraz archiwów (Package Sealing, Package Versioning),
- przenośność.

# JAR

Archiwum jar tworzy się używając komendy jar, np:

```
jar cf archiwum.jar klasa1.class klasa2.class ...
```

Użyte opcje:

- **c** – tworzenie pliku (create),
- **f** – zawartość archiwum zostanie zapisana do pliku archiwum.jar zamiast do standardowego wyjścia (stdout);

Inne najczęściej używane opcje:

- **m** – do archiwum zostanie dołączony plik manifest z określonej lokalizacji, np: `jar cmf plik_manifest archiwum.jar *`,
- **C** – zmiana katalogu w trakcie działania archiwizatora, np: `jar cf ImageAudio.jar -C images * -C audio *`.

# MANIFEST

W archiwum jar znajduje się katalog **META-INF** a w nim plik **MANIFEST.MF** zawierający dodatkowe informacje o archiwum. Przykładowa zawartość:

```
Manifest-Version: 1.0  
Created-By: 1.5.0-b64 (Sun Microsystems Inc.)  
Ant-Version: Apache Ant 1.6.5  
Main-Class: pl.edu.uj.if.wyklady.java.Wyklad06
```

mówi, że po uruchomieniu archiwum wykonana zostanie metoda `main(String[] args)` zawarta w klasie **Wyklad06** znajdującej się w pakiecie `pl.edu.uj.if.wyklady.java`.

Uruchomienie pliku jar:

```
java -jar archiwum.jar
```

DZIĘKUJĘ ZA UWAGĘ