tree **school**

Corso Java Backend
Modulo 9
Package Java principali

Andrea Gasparini

Package principali

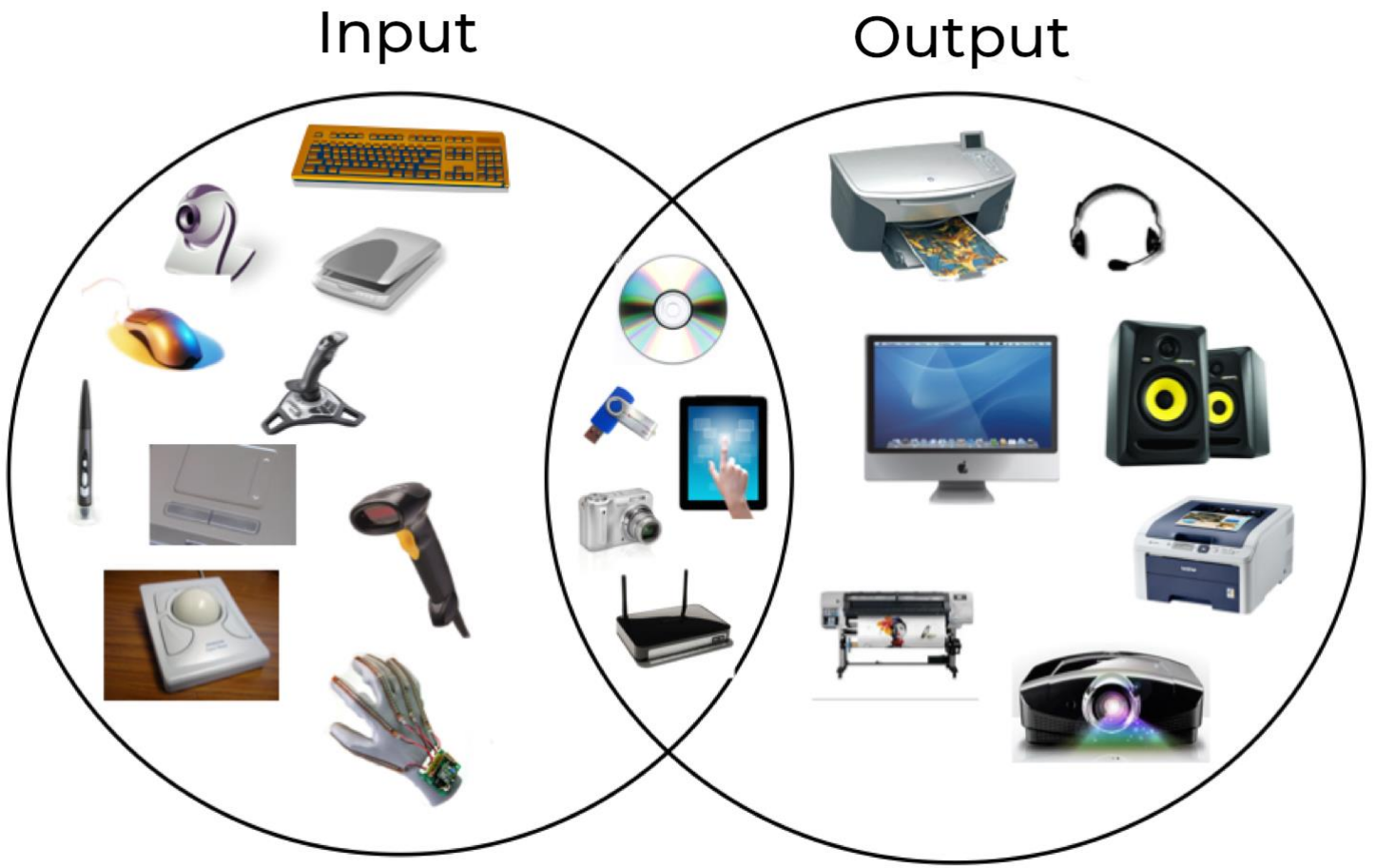
java.io – Gestione Input e Output

java.lang – Classe System, String e Math

java.util – Collections, Random, Scanner, UUID

java.time – Package per il ”tempo”, data, orario, timestamp

java.io



Output su console

System.out.println()

out è un campo statico final della classe System di tipo PrintStream

```
public static final PrintStream out
```

PrintStream è una classe del package java.io che estende OutputStream, la classe astratta che rappresenta un output di byte

```
PrintStream output = System.out;
output.println("ciao");
```

Scorciatoia intellij ‘sout’

Output su console

java.io.OutputStream

Constructors

Constructor and Description
`OutputStream()`

Method Summary

All Methods | **Instance Methods** | **Abstract Methods** | **Concrete Methods**

Modifier and Type	Method and Description
`void`	`close()` Closes this output stream and releases any system resources associated with this stream.
`void`	`flush()` Flushes this output stream and forces any buffered output bytes to be written out.
`void`	`write(byte[] b)` Writes `b.length` bytes from the specified byte array to this output stream.
`void`	`write(byte[] b, int off, int len)` Writes `len` bytes from the specified byte array starting at offset `off` to this output stream.
`abstract void`	`write(int b)` Writes the specified byte to this output stream.

Input da tastiera

```
Scanner s = new Scanner(System.in);
```

Scanner è una classe di java.util che si costruisce a partire da un InputStream

InputStream è una classe astratta del package java.io che rappresenta un input di byte

come ‘’out’’, anche ‘’in’’ è un campo static final di System ma di tipo InputStream

```
InputStream input = System.in;
Scanner s = new Scanner(input);
```

Input da tastiera

java.io.InputStream

Constructors

Constructor	Description
`InputStream()`	

Method Summary

All Methods | **Static Methods** | **Instance Methods** | **Abstract Methods** | **Concrete Methods**

Modifier and Type	Method	Description
int	`available()`	Returns an estimate of the number of bytes that can be read (or skipped over) from this inpu
void	`close()`	Closes this input stream and releases any system resources associated with the stream.
void	`mark(int readlimit)`	Marks the current position in this input stream.
boolean	`markSupported()`	Tests if this input stream supports the `mark` and `reset` methods.
static `InputStream`	`nullInputStream()`	Returns a new `InputStream` that reads no bytes.
abstract int	`read()`	Reads the next byte of data from the input stream.
int	`read(byte[] b)`	Reads some number of bytes from the input stream and stores them into the buffer array `b`.
int	`read(byte[] b, int off, int len)`	Reads up to `len` bytes of data from the input stream into an array of bytes.
byte[]	`readAllBytes()`	Reads all remaining bytes from the input stream.
int	`readNBytes(byte[] b, int off, int len)`	Reads the requested number of bytes from the input stream into the given byte array.
byte[]	`readNBytes(int len)`	Reads up to a specified number of bytes from the input stream.

java.util.Scanner

La classe scanner rappresenta la ”scansione” di un testo, che può arrivare dall’InputStream della tastiera, da un altro InputStream, da una stringa, da un oggetto File, dal Path di un file etc...

Constructors

Constructor and Description
`Scanner(File source)` Constructs a new `Scanner` that produces values scanned from the specified file.
`Scanner(File source, String charsetName)` Constructs a new `Scanner` that produces values scanned from the specified file.
`Scanner(InputStream source)` Constructs a new `Scanner` that produces values scanned from the specified input stream.
`Scanner(InputStream source, String charsetName)` Constructs a new `Scanner` that produces values scanned from the specified input stream.
`Scanner(Path source)` Constructs a new `Scanner` that produces values scanned from the specified file.
`Scanner(Path source, String charsetName)` Constructs a new `Scanner` that produces values scanned from the specified file.
`Scanner(Readable source)` Constructs a new `Scanner` that produces values scanned from the specified source.
`Scanner(ReadableByteChannel source)` Constructs a new `Scanner` that produces values scanned from the specified channel.
`Scanner(ReadableByteChannel source, String charsetName)` Constructs a new `Scanner` that produces values scanned from the specified channel.
`Scanner(String source)` Constructs a new `Scanner` that produces values scanned from the specified string.

Java Package

java.util.Scanner

next()
Finds and returns the next complete token from t

next(Pattern pattern)
Returns the next token if it matches the specified

next(String pattern)
Returns the next token if it matches the pattern c

nextBigDecimal()
Scans the next token of the input as a BigDecim

nextBigInteger()
Scans the next token of the input as a BigInteg

nextBigInteger(int radix)
Scans the next token of the input as a BigInteg

remove()
The remove operation is not supported by this i

reset()
Resets this scanner.

skip(Pattern pattern)
Skips input that matches the specified pattern,

skip(String pattern)
Skips input that matches a pattern constructed

toString()
Returns the string representation of this Scann

useDelimiter(Pattern pattern)
Sets this scanner's delimiting pattern to the spe

useDelimiter(String pattern)
Sets this scanner's delimiting pattern to a patte

useLocale(Locale locale)
Sets this scanner's locale to the specified locale

useRadix(int radix)
Sets this scanner's default radix to the specifie

close()
Closes this scanner.

delimiter()
Returns the Pattern this Scanner is currently using to match delimiters.

findInLine(Pattern pattern)
Attempts to find the next occurrence of the specified pattern ignoring delimi

findInLine(String pattern)
Attempts to find the next occurrence of a pattern constructed from the spec

findWithinHorizon(Pattern pattern, int horizon)
Attempts to find the next occurrence of the specified pattern.

findWithinHorizon(String pattern, int horizon)
Attempts to find the next occurrence of a pattern constructed from the spec

hasNext()
Returns true if this scanner has another token in its input.

hasNext(Pattern pattern)
Returns true if the next complete token matches the specified pattern.

hasNext(String pattern)
Returns true if the next token matches the pattern constructed from the spe

hasNextBigDecimal()
Returns true if the next token in this scanner's input can be interpreted as a

hasNextBigInteger()
Returns true if the next token in this scanner's input can be interpreted as a

hasNextBigInteger(int radix)
Returns true if the next token in this scanner's input can be interpreted as a

hasNextBoolean()
Returns true if the next token in this scanner's input can be interpreted as a

hasNextByte()
Returns true if the next token in this scanner's input can be interpreted as a

hasNextByte(int radix)
Returns true if the next token in this scanner's input can be interpreted as a

hasNextDouble()
Returns true if the next token in this scanner's input can be interpreted as a

hasNextFloat()
Returns true if the next token in this scanner's input can be interpreted as a

hasNextInt()
Returns true if the next token in this scanner's input can be interpreted as a

hasNextInt(int radix)
Returns true if the next token in this scanner's input can be interpreted as a

hasNextLine()
Returns true if there is another line in the input of this scanner.

File

Un altro tipo di I/O è effettuato sui file, che sono essenzialmente collezioni di dati salvate su un supporto di memoria

Possiamo distinguere due tipi di file: File di **testo** e File **binari**

I file di testo contengono linee di testo, delimitate da '\n' o '\r'

- Ad esempio file .txt, .html, .java, o file ad uso di altri programmi tipo .ics per i calendari

I file binari contengono informazioni sotto forma di concatenazione di byte.

- Ad esempio immagini, file audio, video o .class di java

java.io.File

La classe File rappresenta appunto un file

Constructors

Constructor and Description
`File(File parent, String child)` Creates a new `File` instance from a parent abstract pathname and a child pathname string.
`File(String pathname)` Creates a new `File` instance by converting the given pathname string into an abstract pathname.
`File(String parent, String child)` Creates a new `File` instance from a parent pathname string and a child pathname string.
`File(URI uri)` Creates a new `File` instance by converting the given `file:` URI into an abstract pathname.

`canExecute()`
Tests whether the application can execute the file denoted by this abstract pathname.

`canRead()`
Tests whether the application can read the file denoted by this abstract pathname.

`canWrite()`
Tests whether the application can modify the file denoted by this abstract pathname.

`createNewFile()`
Atomically creates a new, empty file named by this abstract pathname

`delete()`
Deletes the file or directory denoted by this abstract pathname.

`getAbsolutePath()`
Returns the absolute pathname string of this abstract pathname.

`getName()`
Returns the name of the file or directory denoted by this abstract pathname.

`getParent()`
Returns the pathname string of this abstract pathname's parent, or `null` if this

`toPath()`
Returns a **`java.nio.file.Path`** object constructed from the this abstract path.

`toString()`
Returns the pathname string of this abstract pathname.

`toURI()`
Constructs a `file:` URI that represents this abstract pathname.

Stream

Input e Output si gestiscono quindi con gli Stream (InputStream e OutputStream)

Gli Stream rappresentano l'astrazione di un flusso di dati sequenziali, quindi essenzialmente un carattere per volta.

Input e Output con gli Stream non si limitano ai file, ma è possibile creare Stream di dati anche tramite internet o con dispositivi diversi, una videocamera o uno schermo ad esempio

I file vengono trattati come Stream sia di input che di output, rispettivamente per lettura e scrittura

Quali classi

Per leggere e scrivere caratteri (file di testo)

- java.io.Reader e java.io.Writer

Per leggere e scrivere byte (file binari)

- java.io.InputStream e java.io.OutputStream

La classe **java.util.Scanner** è stata introdotta in Java 1.5 , per permettere l'accesso input e output in modo più semplice ma meno efficiente perché mette a disposizione una maggior quantità di funzioni che impattano sull'efficienza

Reader

Per leggere in modo efficiente un file di medio-grandi dimensioni conviene usare la bufferizzazione tramite la classe **BufferedReader**, evitando di dover effettuare una lettura sequenziale da disco (altamente inefficiente!), ovvero leggere un byte alla volta, poi convertirlo in un carattere ed infine ritornarlo

```
BufferedReader br = null;
try {
    br = new BufferedReader(new FileReader( fileName: "/percorso/al/file"));

    while (br.ready()){
        String line = br.readLine();
    }

}
catch (IOException e)
{
    //gestisci errore
}
finally
{
    if (br != null){
        try {
            br.close();
        } catch (IOException ex){
            //gestisci errore
        }
    }
}
```

Reader

Uso del **try-with-resource**

La risorsa inizializzata nella parentesi del try viene chiusa in automatico alla fine del blocco try

```
try(BufferedReader br = new BufferedReader(new FileReader( fileName: "/percorso/al/file"))) {

    while (br.ready()){
        String line = br.readLine();
    }

}
catch (IOException e)
{
    //gestisci errore
}
```

Reader

Uso del **try-with-resource**

L'interfaccia **AutoCloseable** implementa il metodo close() e indica che una classe possiede la funzione di rilascio delle risorse, che viene usata dal try-with-resource

```
public interface Closeable extends AutoCloseable {

    /** Closes this stream and releases any system resources
    public void close() throws IOException;
}
```

Writer

Per scrivere su file si usa FileWriter, eventualmente supportato da un BufferedWriter in caso la quantità di dati da scrivere non sia contenuta

```
try(BufferedWriter br = new BufferedWriter(new FileWriter( fileName: "/percorso/al/file"))) {

    br.write( str: "ciao");

}
catch (IOException e)
{
    //gestisci errore
}
```

In un altro modo

Le classi Scanner e PrintWriter sono un'alternativa (meno efficiente) per lettura e scrittura di file di testo

```java
File f = new File( pathname: "/path/to/file");

try (Scanner s = new Scanner(f)) {

    while (s.hasNext()){
        String line = s.nextLine();
    }

} catch (IOException e) {
    e.printStackTrace();
}
```

```java
File f = new File( pathname: "/path/to/file");

try (PrintWriter pw = new PrintWriter(f)) {

    pw.print("ciao");
    pw.println(5);

} catch (IOException e) {
    e.printStackTrace();
}
```

[Approfondimento su BufferedReader/BufferedWriter vs Scanner/PrintWriter]

Java NIO

A partire da Java 7, java.nio è divento il package principale per gestire i file. Questo perché la classe *File* di java.io ha dei problemi di robustezza, ad esempio ci sono errori per cui non vengono lanciate eccezioni, rendendo la gestione di questi errori molto più complessa.

In più *java.io.File* non contiene i metadati dei file (permessi, proprietario), non supporta i symlink e contiene metodi che non sono efficienti su grandi quantità di file.

Essenzialmente il cambiamento è che la classe **java.io.File** è stata rimpiazzata dall'interfaccia **java.nio.Path** che rappresenta un percorso invece che un file.

Java NIO

Per ottenere un oggetto Path relativo ad un percorso si usa il metodo Paths.get()

```
Path p = Paths.get("percorso", "al", "file");
```

Non si usa il separatore di cartelle "/" esplicitamente, perché ci pensa il compilatore ad usare quello più appropriato per l'OS sottostante

Java NIO

Tutte le funzioni della classe java.io.File sono state raggruppate come metodi statici della classe java.nio.Files

delete(**Path** path)
Deletes a file.

isDirectory(**Path** path, **LinkOption**... options)
Tests whether a file is a directory.

isExecutable(**Path** path)
Tests whether a file is executable.

isHidden(**Path** path)
Tells whether or not a file is considered *hidden*.

isReadable(**Path** path)
Tests whether a file is readable.

In aggiunta ci sono anche i metodi di comodo per poter ottenere Input/OutputStream e BufferedReader/Writer

static **InputStream**	**newInputStream**(**Path** path, **OpenOption**... options) Opens a file, returning an input stream to read from the file.
static **OutputStream**	**newOutputStream**(**Path** path, **OpenOption**... options) Opens or creates a file, returning an output stream that may be used to write bytes to the file.
static **BufferedReader**	**newBufferedReader**(**Path** path, **Charset** cs) Opens a file for reading, returning a `BufferedReader` that may be used to read text from the file in an efficient manner.
static **BufferedWriter**	**newBufferedWriter**(**Path** path, **Charset** cs, **OpenOption**... options) Opens or creates a file for writing, returning a `BufferedWriter` that may be used to write text to the file in an efficient manner.

Java NIO

Ottenere la lista dei file in una directory

```
List<File> files = new ArrayList<>();

try (DirectoryStream<Path> stream = Files.newDirectoryStream(Paths.get( first: "/cartella"))) {

    for (Path path : stream)
        if (!Files.isDirectory(path))
            files.add(path.toFile());

} catch (IOException e) {
    //gestire errore
}
```

java.lang

Il package java.lang contiene le classi principali del linguaggio java

- String, StringBuilder
- Wrapper dei primitivi (Integer, Float, Character, etc)
- Math
- Object
- Thread

java.lang.String

length()	Restituisce la lunghezza della stringa	”ciao”.length() == 4
subString(int from, int to)	Restituisce una stringa che contiene i caratteri della stringa originaria dall’indice from all’indice to-1	”ciao”.subString(1, 3) == “ia”
charAt(int index)	Restituisce il carattere all’indice index	”ciao”.charAt(1) == ‘i’
indexOf(String str)	Restitiusce l’indice al quale compare la prima occorrenza della stringa str	“ciao ciao”.indexOf(”ao”) == 2

java.lang.String

replace(char old, char new)	Rimpiazza tutte le occorrenze del carattere old con il carattere new	”ciao”.replace(‘c’, ‘i’) == ”iiao”
replaceAll(String regex, String replacement)	Rimpiazza tutte le occorrenze della stringa regex (rispettando la regex se presente) con la stringa replacement	”ciao ciao come va”.replaceAll(‘’ciao”, ‘’ola”) == ‘’ola ola come va”
split(String regex)	Restituisce un array di sottostringhe, dividendo la stringa ogni volta che la regex matcha	”ciao come va”.split(“ “) == [‘’ciao”, ”come”, ‘’va”]
startsWith(String prefix)	Restituisce true se prefix è prefisso della stringa	‘’ciao”.startsWith(‘’ci”) == true
toLowerCase() toUpperCase()	Restituiscono la stringa rispettivamente tutta in minuscolo o maiuscolo	‘’ciao”.toUpperCase() == ‘’CIAO” ‘’CIAO”.toLowerCase() == ”ciao”

java.lang.StringBuilder

L'oggetto String in java è **immutabile**, ovvero non è possibile modificarne un'istanza. Quando si usano i metodi della classe String oppure si concatena una stringa con un'altra, java sta creando un nuovo oggetto String che contiene la stringa modificata. Questo processo è molto poco efficiente.

StringBuilder

La classe StringBuilder rende queste operazioni efficienti, andando a creare l'oggetto String solo alla fine delle modifiche

```
String s = "ciao";
s += " come";
s += " va";
```

```
StringBuilder sb = new StringBuilder("ciao");
sb.append(" come");
sb.append(" va");

String s2 = sb.toString();
```

java.lang - Wrapper

Nel package java.lang sono presenti le classi wrapper dei primitivi, ovvero la rappresentazione come oggetto dei tipi primitivi, che forniscono metodi di accesso e visualizzazione dei valori

Primitive Types	Wrapper Class
int	Integer
char	Character
float	Float
short	Short
long	Long
double	Double
byte	Byte

java.lang - Wrapper

Alcuni campi e metodi statici delle classi wrapper

- Integer.*MIN_VALUE*, Integer.*MAX_VALUE*
- Double.*MIN_VALUE*, Double.*MAX_VALUE*
- I metodi Integer.parseInt(), Double.parseDouble() ecc.
- Il metodo toString() fornisce una rappresentazione di tipo stringa per un tipo primitivo
- Character.isLetter(), Character.isDigit(), Character.isUpperCase(), Character.isLowerCase(), Character.toUpperCase(), ecc.

java.lang - Math

La classe Math offre le costanti più comuni nel mondo della matematica e metodi di supporto per operazioni matematiche non banali

Math.*PI*	Pi greco
Math.*E*	Costante E del logaritmo naturale
Math.pow(x, y)	Restituisce il risultato di "x alla y"
Math.sqrt(x)	Restituisce la radice quadrata di x
Math.log(x)	Restituisce il logaritmo di x
Math.exp(x)	Restituisce "2 alla x"

java.lang - Math

Ottenere un valore decimale casuale in Java è possibile tramite il metodo statico random() della classe Math

Questo metodo restituisce un valore decimale maggiore o uguale di 0 e minore di 1; da cui possiamo facilmente ottenere valori interi o boolean

```
// Obtain a number between [0.0 - 1.0)
double randomValue = Math.random();

// Obtain a number between [0 - 49]
int randomInt = (int) (randomValue * 50);

// Obtain a boolean value
boolean randomBool = randomValue > 0.5;
```

java.util

Il package java.util contiene classi di utilità, tra cui tutte le collection (List, Map, Set, etc), la classe Random, per generare numeri randomici, la classe Scanner, la classe UUID

UUID - Universally Unique Identifier

7e8162ce-8f65-46c0-8fd4-d41f465399c5

Identificatore univoco con una probabilità di collisione quasi zero

Esistono **340 282 366 920 938 000 000 000 000 000 000 000 000** UUID diversi

Se generassi **1 miliardo** di UUID al **secondo** per **100 anni**, la probabilità di ottenere un duplicato sarebbe del **50%**

```
UUID id = UUID.randomUUID();
id.toString(); //7e8162ce-8f65-46c0-8fd4-d41f465399c5
UUID id2 = UUID.fromString("7e8162ce-8f65-46c0-8fd4-d41f465399c5");
```

java.util - Random

Il package java.util fornisce una classe Random per la generazione di valori casuali, che consente di creare direttamente valori casuali di tipo int, float, boolean e double

```
import java.util.Random;

Random rand = new Random();

// Obtain a number between [0.0 - 1.0)
double randomDouble = rand.nextDouble();

// Obtain a number between [0 - 49]
int randomInt = rand.nextInt(50);

// Obtain a boolean value
boolean randomBool = rand.nextBoolean();
```

java.time

Il package java.time contiene le classi per gestire date ed orari.

Le più comuni sono

LocalTime	Che rappresenta un orario
LocalDate	Che rappresenta una data
LocalDateTime	Che rappresenta una data ed un orario
Period	Indica un lasso di tempo in termini di anni, mesi, giorni
Duration	Indica un lasso di tempo in termini di secondi e nanosecondi
Instant	Indica un istante nel tempo con precisione al nanosecondo

java.time

LocalDate

```
LocalDate date = LocalDate.now();
int year = date.getYear();
Month month = date.getMonth();
int dayOfYear = date.getDayOfYear();
int dayOfMonth = date.getDayOfMonth();
DayOfWeek dayOfWeek = date.getDayOfWeek();
```

LocalTime

```
LocalTime time = LocalTime.now();
int hour = time.getHour();
int minute = time.getMinute();
int second = time.getSecond();
int nano = time.getNano();
```

LocalDateTime

```
LocalDateTime dateTime = LocalDateTime.now();
LocalDateTime dateTime2 = date.atTime(time);
LocalDateTime dateTime3 = date.atStartOfDay();
```

```
boolean after = dateTime.isAfter(dateTime2);
boolean before = dateTime.isBefore(dateTime2);

boolean afterDate = date.isAfter(date2);
boolean beforeDate = date.isBefore(date2);

boolean afterTime = time.isAfter(time2);
boolean beforeTime = time.isBefore(time2);
```

java.time

Period e Duration

```
Period p = Period.between(date, date2);
p.get(ChronoUnit.DAYS);
p.get(ChronoUnit.MONTHS);
p.get(ChronoUnit.YEARS);

Duration d = Duration.between(time, time2);
d.get(ChronoUnit.HOURS);
d.get(ChronoUnit.MINUTES);
d.get(ChronoUnit.SECONDS);
```

Instant

```
Instant i = Instant.now();
Instant i2 = dateTime.toInstant(ZoneOffset.UTC);
i.isAfter(i2);
i.isBefore(i2);
i.getEpochSecond();
i.getNano();
```

Domande?