Programmazione II – Java

Ogni file contiene una classe → nome classe = nome file

Definire una classe → campi (proprietà), costruttore (non obbligatorio) e metodi

Variabili

- tipi primitivi (iniziale minuscola) → `boolean`, `char`, `byte`, `short`, `int`, `long`, `float`, `double`
  variabili (C) → proprietà di una classe (Java)
- oggetti → `String`, ..., o altri oggetti creati da altre classi tramite il costruttore della classe,
  comprende il costruttore e i relativi metodi di quell'oggetto,
  se un metodo è `static` si chiama tramite il nome della classe (*nome_classe.nome_metodo_static*)

`final` (costante) → quando viene assegnato un valore a una proprietà, questo non cambia più,
se il metodo è final, nelle classi ereditate non si può sovrascrivere.

`static` → alla compilazione del codice viene istanziato una sola volta
Se si creano più oggetti a partire dallo stesso "stampino" (classe), se un metodo (o una proprietà) è static, viene creato una vota sola, e se si modifica, si modifica per tutti gli oggetti che lo comprendono.
Può essere chiamato solo dalla classe o da un altro metodo static presente nella classe.

Visibilità

`public` → visibile ovunque

`private` → visibile solo all'interno della classe, (incapsulamento)
classi esterne non possono modificare direttamente il valore, si passa attraverso metodi pubblici, quindi anche attraverso controlli all'interno della classe:
getter → per vederlo all'esterno della classe
setter → per modificarlo

`protected` → visibile solo all'interno della cartella stessa (neanche in sottocartelle)

`package` → percorso di dove si trova il file della classe rispetto a `src` (cartella source che contiene il codice)

`this.` → sostituto dell'oggetto dentro sé stesso (segnaposto)

Stampa su terminale

```
System.out.println("Hello world");
```
il println chiama già il .toString() dell'oggetto "più vicino"

```
import java.lang.System;
```
java.lang già importata di default

metodi

`err` → costante della classe che fa riferimento allo standard error
`in` → costante della classe che fa riferimento allo standard input
`out` → costante della classe che fa riferimento allo standard output
`static long currentTimeMillis()` → ritorna il numero di millisecondi passati dalla mezzanotte dell'1 gennaio 1970 UTC

Lettura dell'input da tastiera

```
import java.util.Scanner;

Scanner scanner = new Scanner(System.in);  Scanner(source)
int numero = scanner.nextInt();
```

```
import java.util.Scanner;
```

metodi

`Scanner(source)` → costruttore, che crea uno Scanner legato alla sorgente indicata
`void close()` → chiude lo Scanner: dopo non può più essere usato
`double nextDouble()`
`float nextFloat()`
`int nextInt()`
`String nextLine()`
`long nextLong()`

Stringhe

caso particolare delle stringhe

Creare nuova stringa inizializzata:

`String string = new String("ciao");` → `String string = "ciao";`

Creare nuova stringa vuota:

`String string = new String();` → `String string = "";`

Identità delle stringhe (`==`) → puntano allo stesso oggetto
Uguaglianza delle stringhe (`equals()`) → controlla se un oggetto è uguale a un altro

Concatenazione fra stringhe → `+` (uso implicito dei metodi concat() e valueOf() di java.lang.String)

```
import java.lang.String;
```

java.lang già importata di default
classe immutabile → i suoi oggetti non possono più essere modificati dopo la creazione

metodi

`String(String other)` → costruttore di copia: crea un clone
`char charAt(int index)`
`int compareTo(String other)` → ritorna negativo, zero, oppure positivo
`int compareToIgnoreCase(String other)` → ritorna negativo, zero, oppure positivo
`String concat(String other)` → implicitamente usato per la concatenazione con +
`boolean endsWith(String end)`
`boolean equals(Object other)`
`boolean equalsIgnoreCase(String other)`
`static String format(String format, Object... args)`
`int indexOf(int character)`
`int indexOf(String what)`
`boolean isEmpty()`
`int length()`
`boolean startsWith(String what)`
`String substring(int start)` → da start incluso, estrae una stringa

`String substring(int start, int end)` → da start incluso ad end escluso
`String toLowerCase()`
`String toUpperCase()`
`String trim()`
`static String valueOf(int i)` → esegue una conversione esplicita di tipo;
esiste per tutti i tipi primitivi, non solo per int;
implicitamente usato per la concatenazione con +

numeri Random

```
import java.util.Random;
```

metodi

```
boolean nextBoolean()
double nextDouble()
float nextFloat()
int nextInt()
int nextInt(int max) → restituisce un numero casuale tra 0 e max escluso
long nextLong()
```

Operazioni aritmetiche

- Per le operazioni tra i tipi primitivi → `+`, `-`, `*`, `/` ...
- Per le operazioni comprendono oggetti → si passa per un metodo (es. `this.value.equals(other.value)`)

La libreria Math non ha costruttore, per chiamare un metodo che esegue un'operazione → `Math.`*nome_metodo*

```
import java.lang.Math;
```

`static double E` → costante della classe
`static double PI` → costante della classe
`static int abs(int i)` → fa il modulo, esiste anche per altri tipi numerici
`static double cos(double d)`
`static double log(double d)` → in base e
`static double log10(double d)` → in base 10
`static int max(int a, int b)` → esiste anche per altri tipi numerici
`static int min(int a, int b)` → esiste anche per altri tipi numerici
`static double sin(double d)`
`static double sqrt(double d)`
`static double tan(double d)`
`static double toDegrees(double radiants)`
`static double toRadiants(double degrees)`

Costruttore di default. Attivazione di un metodo o costruttore. Parametri formali e attuali. Record e stack di attivazione. Modificatore final per i campi e le variabili locali. Creazione di array per enumerazione degli elementi. Definizione di campi static. Controllo di legalità delle date. Overloading dei costruttori. Concatenazione dei costruttori con la sintassi this().

Definizione automatizzata di un tipo enumerato tramite la parola chiave enum. Definizione di metodi di confronto fra oggetti.

Enumerazione delle stagioni, metodo delle Date che ritorna la stagione di una data. Dichiarazione degli array, inizializzazione esplicita per enumerazione degli elementi oppure creazione con new. Lo pseudocampo length degli array. Ciclo for each sugli array. Creazione dell'array delle stagioni di un array di date.

Metodi di uso frequente delle classi E definite tramite enum

- static E[] values() (ritorna l'array di tutti gli elementi dell'enumerazione)
- static E valueOf(String name) (ritorna l'elemento dell'enumerazione che ha il nome indicato)
- int compareTo(E other) (determina chi viene prima nell'enumerazione)
- int ordinal() (ritorna il numero d'ordine di un elemento dell'enumerazione)

Array

```
import java.util.Arrays;
```

- static int binarySearch(int[] arr, int key) (ritorna la posizione di key dentro arr, oppure un numero negativo se arr non contiene key. Assume che l'array arr sia ordinato. Questo metodo esiste anche per gli altri tipi primitivi numerici e per i tipi riferimento, nel qual caso chiama compareTo() per decidere l'ordine)
- static boolean equals(int[] arr1, int[] arr2) (controlla che arr1 e arr2 abbiano stessa lunghezza e contengano gli stessi elementi nello stesso ordine. Questo metodo esiste anche per gli altri tipi primitivi nonché per array di tipi riferimento, nel qual caso chiama equals() fra tutte le coppie di oggetti da confrontare)
- static void fill(int[] arr, int val) (assegna val a tutti gli elementi di arr. Questo metodo esiste anche per tutti gli altri tipi primitivi e per array di tipi riferimento)
- static void sort(int[] arr) (ordina arr in senso crescente, in tempo O(n log n). Questo metodo esiste anche per tutti gli altri tipi primitivi numerici e per i tipi riferimento, nel qual caso chiama compareTo() per decidere l'ordine)
- static String toString(int[] arr) (ritorna una stringa che riporta gli elementi di arr, nel loro ordine. Questo metodo esiste anche per gli altri tipi primitivi e per array di tipi riferimento, nel qual caso chiama toString() sugli elementi dell'array e concatena il risultato.

Assegnamento fra variabili di tipo array. Array multidimensionali e loro struttura in memoria. Array sparsi. Definizione di metodi static. Esempio del contatore di oggetti creati di una classe. Esempio di un metodo che ritorna il primo gennaio di un anno e di un metodo che ritorna il 31 dicembre di un anno. Interfaccia pubblica di una classe. Classi e oggetti mutabili e immutabili. Aliasing fra variabili e rischi in caso di condivisione di oggetti mutabili. JavaDoc. Aggiunta dei commenti JavaDoc nel codice sorgente e generazione delle pagine HTML da Eclipse. Definizione di classi per estensione, ereditando campi e metodi. I costruttori non si ereditano. Notazione super() per la concatenazione dei costruttori da sottoclasse a superclasse. Esempio delle date generiche e poi italiane, americane e australiane.

Ridefinizione o aggiunta di metodi. Relazione di sottoclasse e superclasse. Tipo statico delle espressioni e tipo dinamico dei valori. Late binding dei metodi. Il class tag degli oggetti, usato per il late binding. Ereditarietà singola e multipla (Java ha solo ereditarietà singola).

La notazione final per metodi e classi. Visibilità public, protected, package e private. La classe java.lang.Object e i suoi metodi toString() ed equals(). Il cast fra tipi riferimento e sua differenza con la conversione di tipi primitivi. L'operatore instanceof. Ridefinizione del metodo equals() di java.lang.Object. Aggiunta di campi nelle sottoclassi. Richiamo dell'implementazione di un metodo della superclasse tramite la notazione super.metodo(...). Esempio delle date e delle date con orario. Modifica conseguente ai metodi equals() e compareTo() delle date.

Modifica del metodo equals() delle date:

- se due date hanno giorno, mese o anno diversi, non sono equals;
- altrimenti, se una data ha anche il tempo e l'altra non ha il tempo, non sono equals;
- altrimenti, se nessuna delle due date ha il tempo, sono equals;
- altrimenti, le due date sono equals se e solo se hanno lo stesso tempo (secondi, minuti e ore uguali).

Modifica del metodo compareTo() delle date:

- due date si mettono prima in ordine per giorno/mese/anno
- altrimenti, se una ha il tempo e l'altra non ha il tempo, si mette prima la data senza tempo
- altrimenti, se nessuna delle due date ha il tempo, si considerano uguali
- altrimenti si mettono in ordine per tempo.

Problema: cosa accade se arrivano altre date con tempo, per esempio AmericanDateAndTime e/o AustralianDateAndTime ?

Interfacce. Esempio dell'interfaccia per le date con tempo (indipendentemente dalla nazione) e suo utilizzo per semplificare i metodi equals() e compareTo() delle date. La visibilità delle interfacce e dei loro metodi è implicitamente public.

Metodi e classi abstract. Esempi per le figure geometriche.

Parametri varargs. Esempio per i costruttori delle classi Formation e Formation433 dell'ultimo laboratorio. Tipi generici: variabili di tipo nella dichiarazione di una classe e di una interfaccia. L'interfaccia di libreria Comparable<T>. Vincoli sulle variabili di tipo. Esempio di un metodo per restituire il minimo di un array non vuoto di Comparable<T>.

```
import java.lang.Comparable<T>;
```

int compareTo(T other): ritorna un numero negativo se viene prima this, un numero positivo se viene prima other e 0 se this e other sono equivalenti.

Classi wrapper della libreria standard. Boxing e unboxing automatico da tipi primitivi a classi wrapper e viceversa. Eccezioni. Lancio delle eccezioni implicito dal linguaggio o esplicito tramite l'istruzione throw. Esempio per il test di legalità delle date. Intercettazione delle eccezioni tramite il costrutto try/catch/finally. Gerarchia delle eccezioni. Eccezioni controllate ed eccezioni non controllate. La clausola throws per costruttori e metodi. Definizione di nostre classi di eccezione.

```
import java.lang.Integer;
```

- static int MAX_VALUE (costante che contiene il massimo int utilizzabile in Java)
- static int MIN_VALUE (costante che contiene il minimo int utilizzabile in Java)
- Integer(int value) (deprecato!)
- Integer(String value) throws java.lang.NumberFormatException
- int intValue() (restituisce il valore int corrispondente)
- int compareTo(Integer other) (infatti Integer implementa Comparable<Integer>)
- static int parseInt(String s) throws java.lang.NumberFormatException (traduce la stringa s in int)
- static String toBinaryString(int i) (ritorna la rappresentazione binaria di i)
- static String toHexString(int i) (ritorna la rappresentazione esadecimale di i)
- static Integer valueOf(int i) (ritorna new Integer(i) ma usa una cache per chiamate ripetute)

Esistono altre classi wrapper corrispondenti agli altri tipi primitivi, con costanti, costruttori e metodi simili a quanto riportato sopra: java.lang.Short, java.lang.Long, java.lang.Float, java.lang.Double,java.lang.Byte e java.lang.Boolean.

```
import java.lang.Character;
```

- static char MAX_VALUE (costante che contiene il massimo char utilizzabile in Java)
- static char MIN_VALUE (costante che contiene il minimo char utilizzabile in Java)
- Character(char value) (deprecato!)
- char charValue() (restituisce il valore char corrispondente)
- int compareTo(Character other) (infatti Character implementa Comparable<Character>)
- static boolean isDigit(char c)
- static boolean isLetter(char c)
- static boolean isLetterOrDigit(char c)

- static boolean isLowerCase(char c)
- static boolean isUpperCase(char c)
- static boolean isWhitespace(char c)
- static char toLowerCase(char c)
- static char toUpperCase(char c)
- static Character valueOf(char c) (ritorna new Character(c) ma usa una cache per chiamate ripetute)

Rendiamo toString() astratto nella nostra superclasse Date, e quindi anche la classe Date stessa. Aggiunta di un metodo abstract per la creazione di una data dello stesso tipo dell'oggetto su cui è invocato (factory method). Utilizzo in getSeason(). Le interfacce java.lang.Closeable e java.lang.AutoCloseable. La clausola finally del try. Il costrutto try with resource. Utilizzo della classe java.util.Scanner per semplificare la lettura di un file testuale.

```
import java.lang.AutoCloseable;
```

- void close() throws java.lang.Exception

```
import java.io.Closeable;
```

- void close() throws java.io.IOException

Le classi astratte java.io.Reader e java.io.Writer per leggere e scrivere file di caratteri. Lettura e scrittura di file di caratteri tramite le classi concrete java.io.FileReader e java.io.FileWriter. Decorazione con aggiunta di buffer tramite java.io.BufferedReader e java.io.BufferedWriter. La classe java.io.PrintWriter per scrivere comodamente file di caratteri. Lettura e scrittura di file di byte (file raw):
classi java.io.InputStream, java.io.FileInputStream, java.io.BufferedInputStream, java.io.OutputStream, java.io.FileOutput Stream e java.io.BufferedOutputStream. Esempio di un metodo dumpAsText(String fileName) aggiunto alle date, per scriverle su un file testuale, e di un costruttore che riceve uno Scanner per rileggere una data precedentemente scritta su file.

```
import java.io.Reader;
```

- int read() throws java.io.IOException (blocca l'esecuzione finché non c'è un carattere da leggere; a quel punto ritorna il codice Unicode del prossimo carattere letto; ritorna -1 se la sorgente di lettura è terminata)
- int read(char[] buffer) throws java.io.IOException (blocca l'esecuzione finché non arriva qualche carattere da leggere; a quel punto scrive i caratteri nel buffer e ritorna il numero di caratteri letti; ritorna -1 se la sorgente di lettura è terminata)

```
import java.io.FileReader;
```

metodi oltre quelli ereditati da java.io.Reader

- FileReader(String fileName) throws java.io.FileNotFoundException (crea un lettore di file che legge i caratteri dal file di testo col nome indicato)

```
import java.io.BufferedReader;
```

metodi oltre quelli ereditati da java.io.Reader

- BufferedReader(Reader parent) (crea una vista bufferizzata di parent)

```
import java.io.Writer;
```
classe astratta

- void write(int c) throws java.io.IOException (interpreta i 16 bit meno significativi di c come codice Unicode di un carattere e lo scrive nel file)
- void write(char[] buffer) throws java.io.IOException (scrive nel file i caratteri contenuti nell'array buffer)
- void write(String s) throws java.io.IOException (scrive nel file i caratteri della stringa s)

```
import java.io.FileWriter;
```

metodi oltre quelli ereditati da java.io.Writer

- FileWriter(String fileName) throws java.io.IOException (crea uno scrittore di file che scrive i caratteri nel file di testo col nome indicato)

```
import java.io.BufferedWriter;
```

metodi oltre quelli ereditati da java.io.Writer

- BufferedWriter(Writer parent) (crea una vista bufferizzata di parent)

```
import java.io.PrintWriter;
```

metodi oltre quelli ereditati da java.io.Writer

- PrintWriter(String fileName) throws java.io.FileNotFoundException (crea uno scrittore di file che scrive i caratteri nel file di testo col nome indicato)
- void print(int i) (scrive i caratteri della rappresentazione decimale dell'intero i nel file. Questo metodo esiste anche per gli altri tipi primitivi)
- void println(int i) (scrive i caratteri della rappresentazione decimale dell'intero i nel file. Questo metodo esiste anche per gli altri tipi primitivi)
- void print(String s)
- void println(String s)
- PrintWriter printf(String format, Object... args) (scrive il formato nel file, in stile linguaggio C)

...

Quasi identiche a quelle del linguaggio C.