**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 (unico) → 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 |
| 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 |

|  |
| --- |
| System.*out*.println(card2); |
| System.*out*.println(“ciao"); |
| System.*out*.println(new Formation433(players).isValid()); System.*out*.println(new Formation433(players)); |
| catch (IllegalArgumentException e) {  System.*out*.println(e);  } |
| System.*out*.println(Arrays.*toString*(array)); |

Lettura dell'input da tastiera

|  |
| --- |
| import java.util.Scanner; |
| 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() |

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

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 |
| String(String other) → costruttore di copia: crea un clone  char charAt(int index) → ritorna il char all’indice passatogli  int compareTo(String other) → ritorna 1 (this > other), 0 (this = other), -1 (this < other) per ordinare  int compareToIgnoreCase(String other) → ritorna negativo, zero, oppure positivo  ignora maiuscole/minuscole  String concat(String other) → implicitamente usato per la concatenazione con +  boolean endsWith(String end)  boolean equals(Object other) → controlla se 2 oggetti stringa sono uguali  (non se puntano allo stesso spazio in memoria → == con gli oggetti)  boolean equalsIgnoreCase(String other)  static String format(String format, Object... args)  int indexOf(int character) → posizione della prima cella che contiene il carattere passato (?)  int indexOf(String what) → posizione della cella dove inizia la sottostringa (?)  boolean isEmpty() → stringa vuota  int length() → lunghezza stringa (dimensione vera, non da 0)  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() → rende la stringa tutta minuscola  String toUpperCase() → rende la stringa tutta maiuscola  String trim() → rimuove caratteri vuoti da inizio e fine stringa (“\n”,”\t”,’ ‘, …)  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 +  toString() (?) |

Mettere esempi

numeri Random

|  |
| --- |
| import java.util.Random; |
| boolean nextBoolean()  double nextDouble()  float nextFloat()  int nextInt()  int nextInt(int bound) → restituisce un numero casuale tra 0 e bound escluso  long nextLong() |

Mettere esempi

Operazioni aritmetiche

* Per operazioni tra i tipi primitivi → +, -, \*, /, ==, != …
* Per operazioni che comprendono oggetti → si passa per un metodo (es. this.value.equals(other.value))

equals() → controlla se un oggetto (this) è uguale a un altro oggetto (other)

== → controlla se 2 oggetti puntano alla stessa cella di memoria

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 e  static double PI → costante π  static int abs(int i) → modulo, esiste anche per altri tipi numerici  static double cos(double d) → coseno  static double log(double d) → logaritmo in base e  static double log10(double d) → logaritmo 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) → seno  static double sqrt(double d) → radice quadrata  static double tan(double d) → tangente  static double toDegrees(double radiants)  static double toRadiants(double degrees) |

Array

Mettere esempi

Dichiarazine array

Inizializzato

Dimensione inizializzata

Fuori dal metodo

Static

new

Utilizzo

Length

For each

Inizializzazione for normale

Altro

L’esempio della matrice, array di array (soccerplayer)

Ritorna puntatore

|  |
| --- |
| 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. |

Mettere esempi classe Arrays

Costruttore

Mettere esempi

Costruttore

Costruttore vuoto

Più costruttori con stesso nome

Utilizzo

Enumerazione

Classe di costanti

Mettere esempi

Classe enum

Utilizzo

Come chiamare classe

Alti esempi, tipo ordinal()

**Metodi di uso frequente delle classi E definite tramite enum**

*nome\_classe\_enumerazione*.

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

Ereditarietà

Extends

Superclasse - sottoclasse

Super

Costruttori non si ereditano

super()

super.

@Override

Sovrascrive il metodo della superclasse con la stessa firma

(tipo di ritorno – nome metodo – lista di input (quantità, tipo e ordine)),

quando il metodo viene chiamato, viene chiamato il metodo della classe più vicina

sottoclasse può implementare suoi metodi

se non si fa l’@Override, la classe eredita i metodi public (o protected se nella stessa cartella),

i metodi private ci sono ma non si vedono

java può estendere una sola classe ma implementare più interfacce

riceve in input l’interfaccia o la superclasse, gli viene passata la sottoclasse che può usare solo metodi definiti della superclasse, perché di tipo della superclasse, ma usa l’implementazione della sottoclasse (classe più vicina). Per ovviare si usa il casting dopo aver controllato (instanceof)

private non può essere usato al di fuori del file in cui si trova

interfaccia → implements

classi → extends

toString()

equals() e @Override equals()

java.lang.Object

equals() e compareTo() implements Comparable<T>

Interfacce

Vengono specificati i metodi da implementare nella sua sottoclasse

La visibilità delle interfacce e dei loro metodi è implicitamente public.

Classi abstract

Ibrido tra classe concreta e interfaccia, contiene sia metodi già implementati,

sia metodi da implementare, contrassegnati con abstract

Parametri varargs

Passati in come ultimo input un numero variabile di parametri, poi trasformati in array

Vincoli sulle variabili di tipo ???

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

Integer

Character

…

Boxing e unboxing automatico da tipi primitivi a classi wrapper e viceversa ???

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)

Eccezioni

schema eccezioni (tablet → gerarchia delle eccezioni)

throw new eccezione()

throws per costruttori e metodi

try/catch/finally

definizione di nostre classi di eccezione

costruttore senza input

costruttore con input

???

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.FileOutputStream 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)

…

Istruzioni if, while, do, for e switch

Quasi identiche a quelle del linguaggio C.