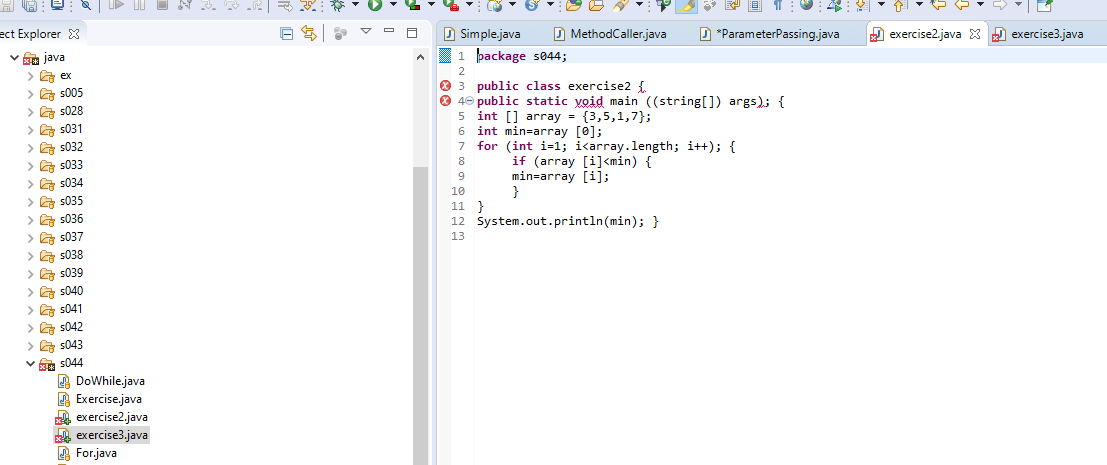
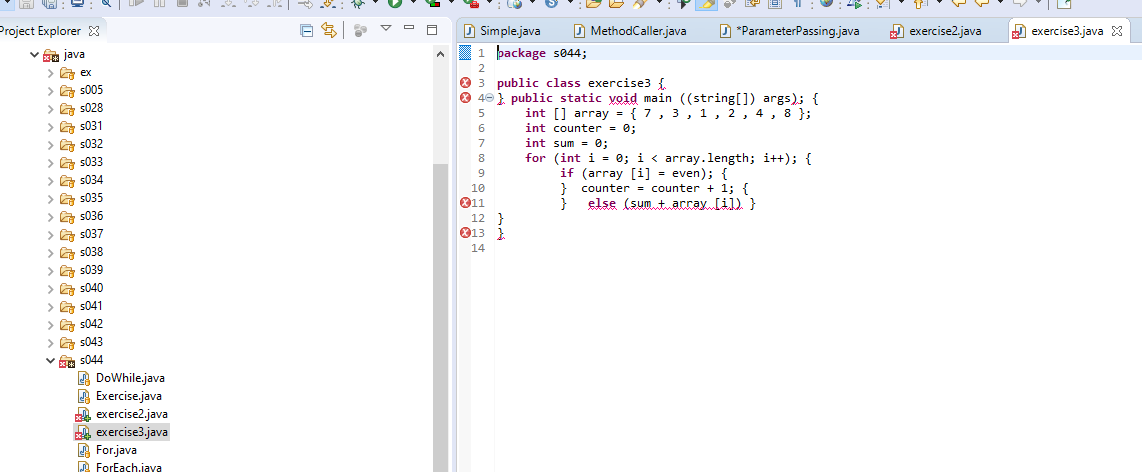
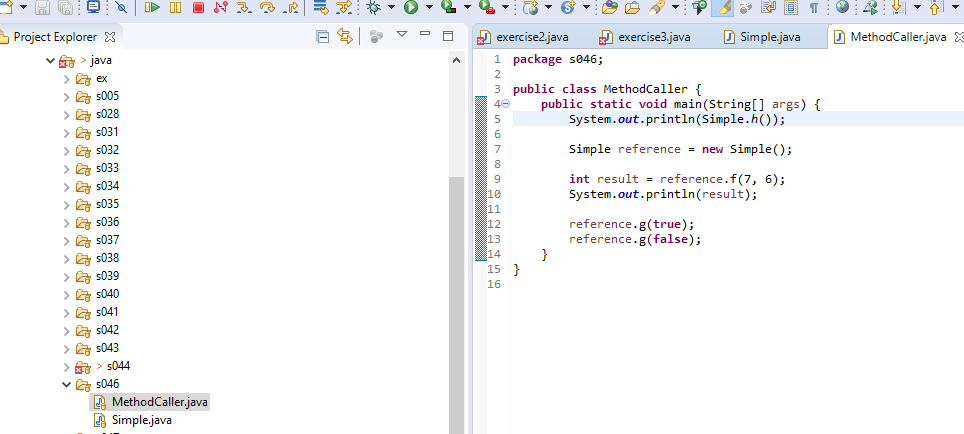
Es. del 10.12

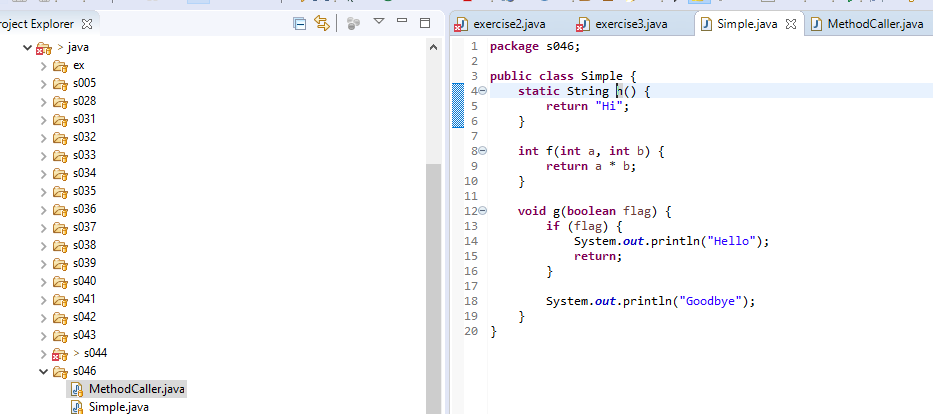




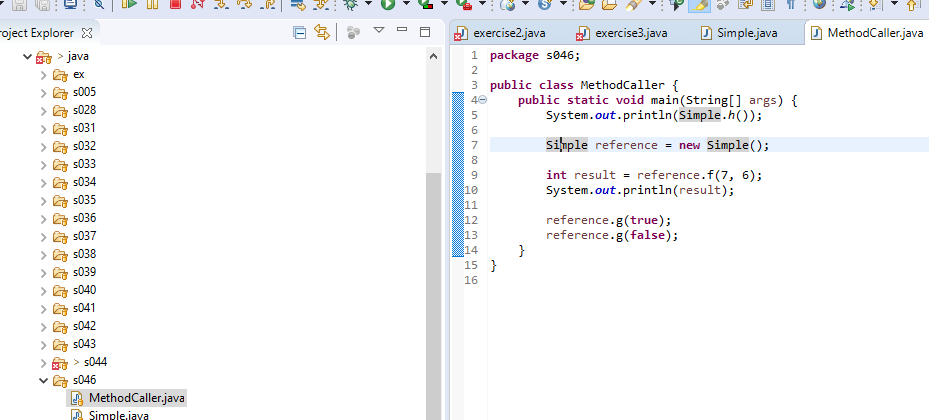
11.12.19



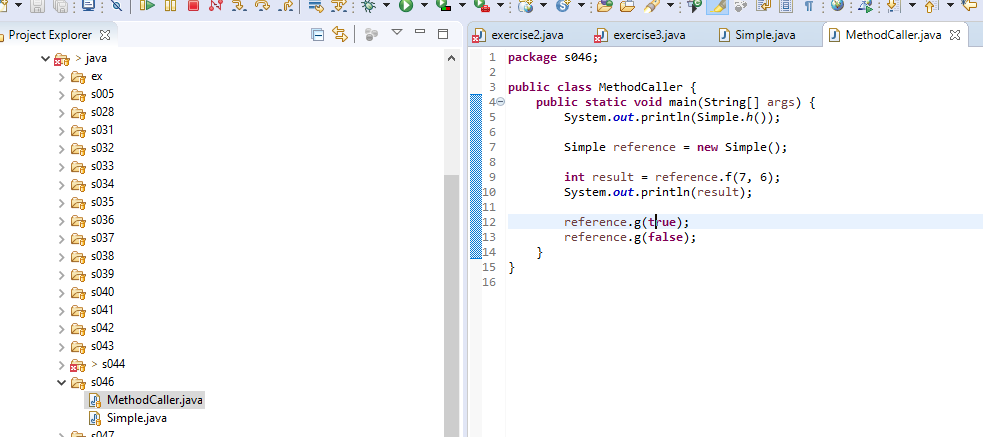
Leggi da dx a sx: Chiama funzione h di simple in riga 5. Funzione che è statica. Invoco il metodo h della classe simple. H appartiene a simple



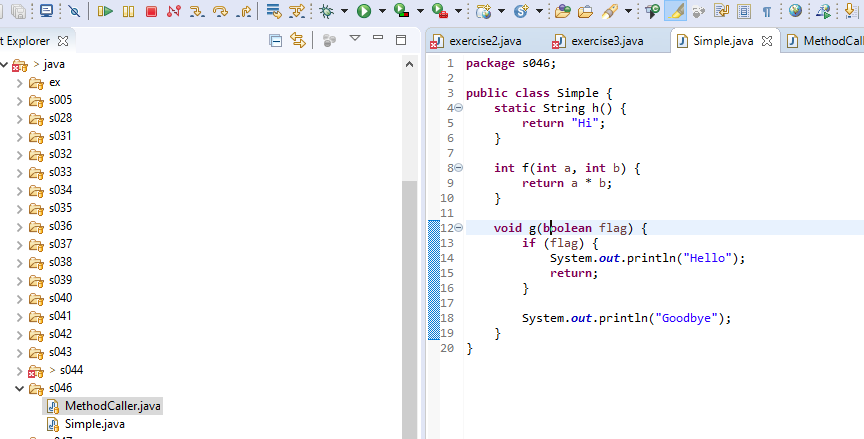
In simple.java il metodo h è statico, posso invocarlo. Cosa deve fare il metodo h? quando chiamo h, h, che è una funzione, esegue il blocco di codice che c’è dentro le graffe (return hi, ritorna una stringa)

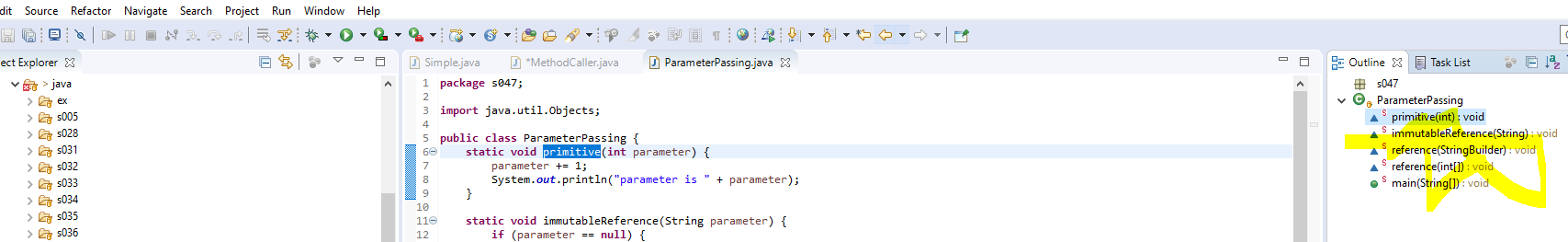


Stanzia oggetto di tipo simple (che va nello heap). Jvm alloca nello stack uno spazio per la variabile di tipo reference che chiamerà “reference” dove mette il rif. All’oggetto simple



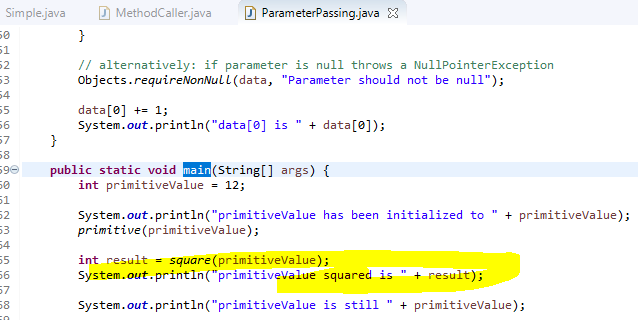
Jvm arriva a riga 12: cos’è reference? È una variabile istanziata sull’oggetto simple. Reference è true. In oggetto simple trovo il metodo g? sì, quindi vado a vedere cosa fa il metodo g: ora ho flag che è true🡪if flag true print hello e return zero quindi giusto perché 0=void. Se non avessi return dopo, andrebbe in fondo (riga 18) e stamperebbe anche goodbye:

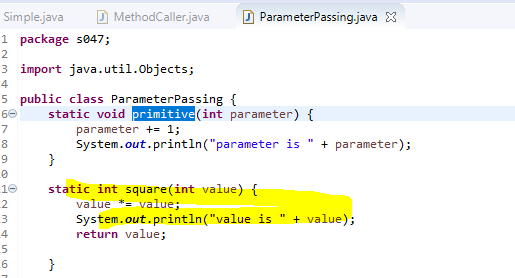




La jvm

Esercizio



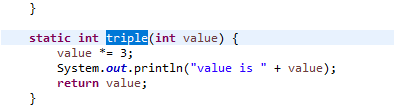


Es. 3

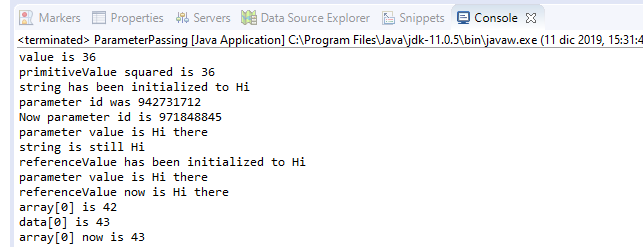
Nel chiamante

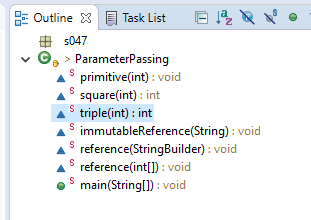


Nel chiamato

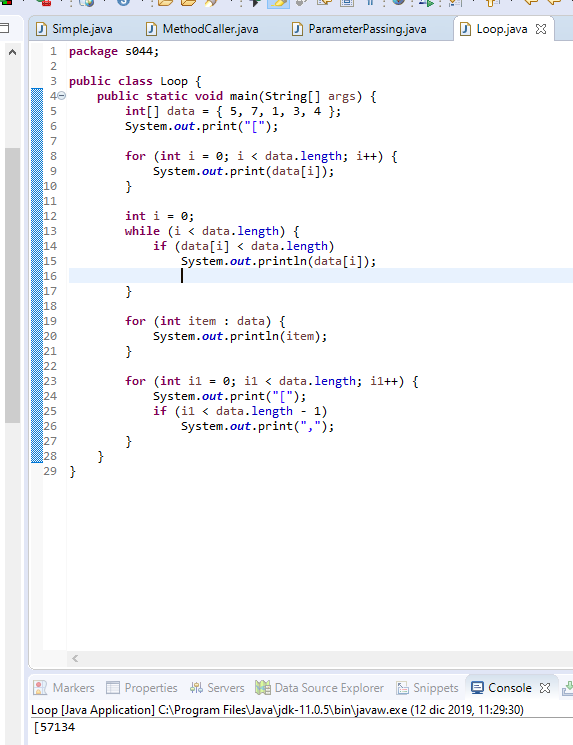


Risultato





12.12.19

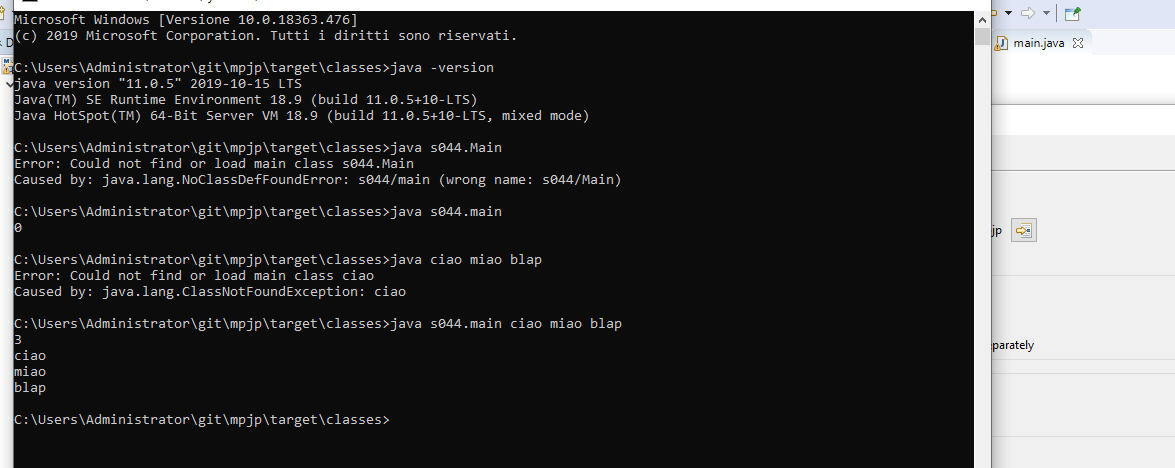


X eseguire la classe s.044.main. gli passo dei parametri,

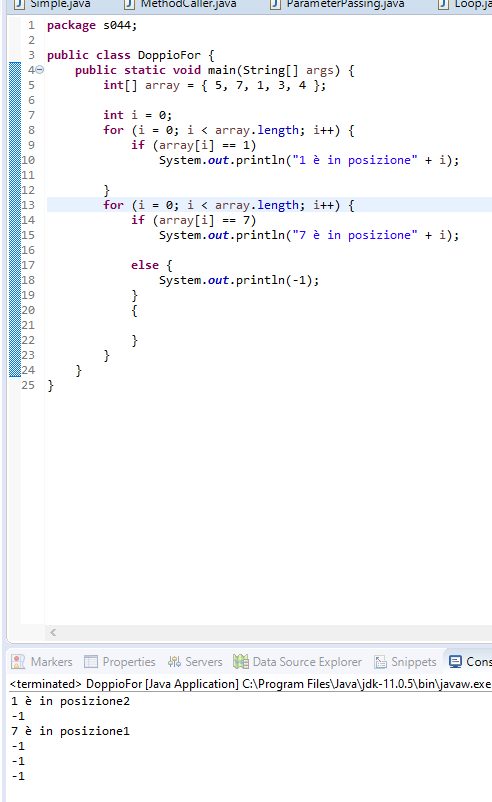
jvm va a vedere se c’è la classe,

va a vedere se c’è il metodo main,

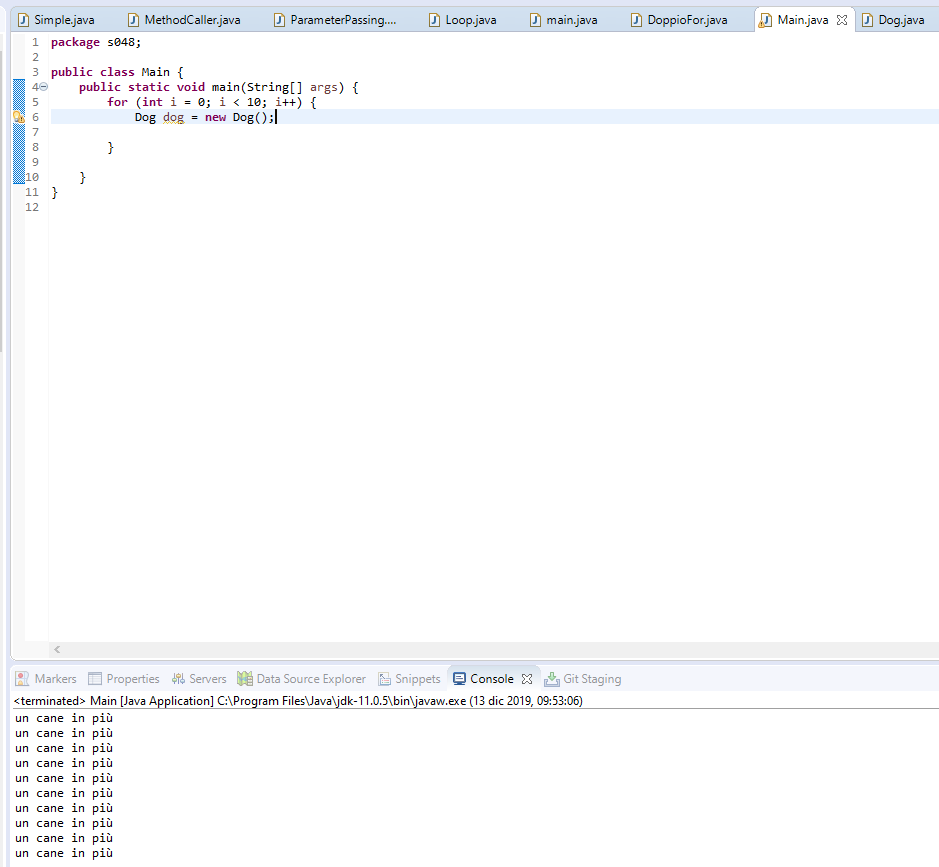
invoca la classe passando un array di stringhe (ciao miao blap), lei stampa il numero dei parametri (3) e poi le stringhe

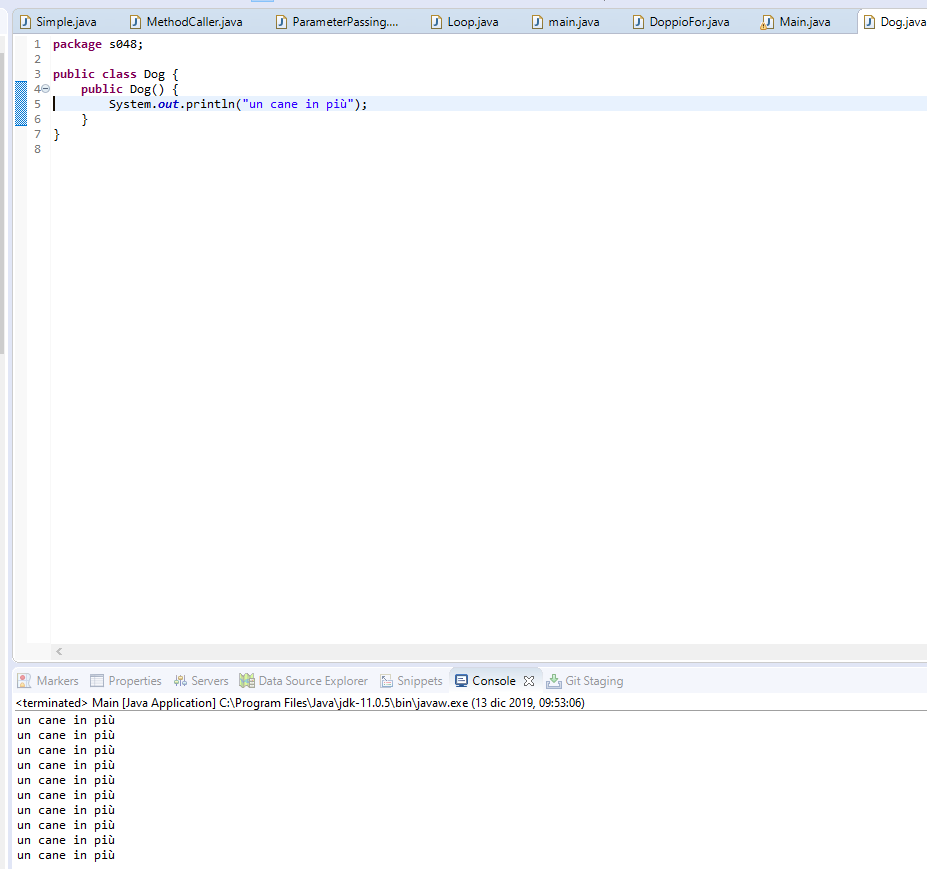


Esercizio doppio for



13.12.19

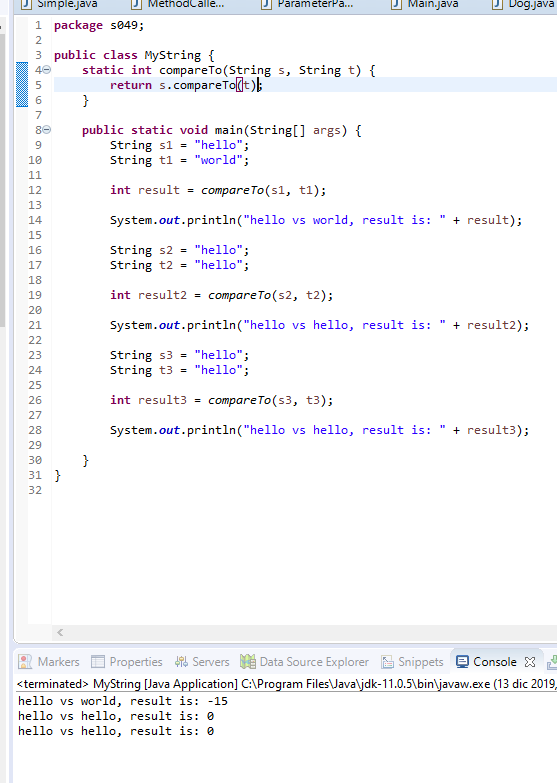




La variabile temporanea i serve per dire quante volte devo stampare che c’è un cane.

Ho creato 10 cani e ce l’ho nello heap ma non sono utilizzabili per questo.

Se creo un array (nello heap) di 10 cani , in posizione 0 ci sarà un NULL come reference a cane (reference perché sono primitivi ma reference).



**return** s.compareTo(t); scorciatoia per comparare s e t e mi da tutti e tre i risultati

**package** s049;

**public** **class** AboutString {

**public** **static** **void** main(String[] args) {

String s = "hello";

String t = "world";

String u = "or";

System.***out***.println("s, t, u: " + s + ", " + t + ", " + u);

System.***out***.println("char at position 1 in s: " + s.charAt(1));

System.***out***.println("s < t: " + s.compareTo(t));

System.***out***.println("t > s: " + t.compareTo(s));

System.***out***.println("concat s and t: " + s.concat(t));

System.***out***.println("t contains u? " + t.contains(u));

String u2 = t.substring(1, 3);

System.***out***.println("u2 = t.substring(1, 3): " + u2);

System.***out***.println("t.substring(3): " + t.substring(3));

System.***out***.println("u equals u2? " + u.equals(u2));

System.***out***.println("u == u2? " + (u == u2));

System.***out***.println("First index of 'l' is s: " + s.indexOf('l'));

System.***out***.println("Last index of 'l' is s: " + s.lastIndexOf('l'));

System.***out***.println("there is no 'x' in s: " + s.indexOf('x'));

System.***out***.println("in s \"ll\" starts at " + s.indexOf("ll"));

System.***out***.println("there is no \"lx\" in s: " + s.indexOf("lx"));

System.***out***.println("check if an empty string is empty: " + "".isEmpty());

System.***out***.println("s length: " + s.length());

String s2 = s.replace('l', 'q');

System.***out***.println("Replacing 'l' with 'q': " + s + " -> " + s2);

String[] splits = "one for me, one for you".split(" ");

System.***out***.println("Splitting: ");

**for** (String token : splits) {

System.***out***.println(token);

}

String joined = String.*join*(" ", splits);

System.***out***.println("Joining back [" + joined + "]");

System.***out***.println("upper: " + s.toUpperCase());

System.***out***.println("lower: " + "SHUT UP!".toLowerCase());

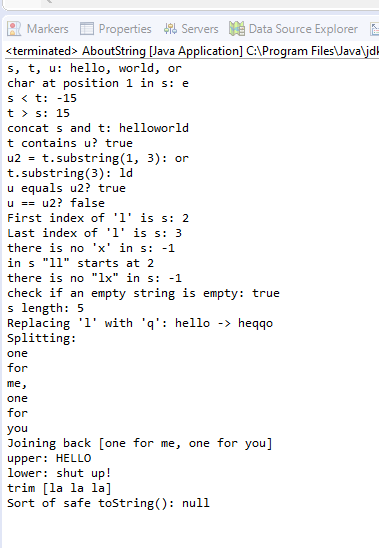
System.***out***.println("trim [" + " la la la ".trim() + "]");

String x = **null**;

System.***out***.println("Sort of safe toString(): " + String.*valueOf*(x));

}

}



Il metodo equals va a vedere il contenuto (quindi è true perché contengono entrambi “or”)

Il metodo == va a vedere la reference (quindi è false perché la reference è diversa, t e u son diverse)

First index of indica in che posizione (indice) si trova la lettera richiesta o da che posizione parte la serie di caratteri richiesti

Lo split usa lo spazio come separatore (possono essere usate anche le virgole) per ottenere un array (gruppo di celle) di stringhe (contenute nelle celle dell’array) e stampo ogni riga dell’array separatamente

Il metodo join è un metodo statico perché alla sua sinistra c’è un nome di una classe (“STRING.”join)

Upper/lower case sono metodi che ridanno una copia della stringa modificata in maiuscolo o minuscolo (la stringa originale non viene toccata perché le stringhe sono immutabili)

Il metodo trim elimina gli spazi bianchi all’inizio e alla fine della stringa

valueOf è un altro metodo statico che crea una stringa a partire dall’oggetto che gli passo (in questo caso ho convertito la x in stringa e la macchina invece di bloccarsi ha ridato un null perché accetta i null)

**package** s050;

**public** **class** AboutStringBuilder {

**public** **static** **void** main(String[] args) {

StringBuilder sb = **new** StringBuilder();

StringBuilder sb2 = **new** StringBuilder(20);

StringBuilder sb3 = **new** StringBuilder("hello");

System.***out***.println("capacity 1: " + sb.capacity());

System.***out***.println("capacity 2: " + sb2.capacity());

System.***out***.println("capacity 3: " + sb3.capacity());

System.***out***.println("appending: " + sb3.append(' ').append("world"));

System.***out***.println("char at position 1 in sb3 is '" + sb3.charAt(1) + "'");

System.***out***.println("deleting: " + sb3.delete(5, 10).deleteCharAt(2));

System.***out***.println("index of \"elo\": " + sb3.indexOf("elo"));

System.***out***.println("inserting \"lo c\" at 3: " + sb3.insert(3, "lo c"));

System.***out***.println("current sb3 length: " + sb3.length());

System.***out***.println("replace section 2-7 with \"r r\": " + sb3.replace(2, 7, "r r"));

stampa r r nell'intervallo tra 2 e 7 escluso

System.***out***.println("reversing: " + sb3.reverse());

sb3.setCharAt(5, 'a');

System.***out***.println("setCharAt: " + sb3);

x fare editing di un solo carattere di una stringa

sb3.setLength(3);

System.***out***.println("setLength: " + sb3);

butta via tutto ciò che viene dopo i primi 3 caratteri

16.12.19

package ex;

import static org.hamcrest.CoreMatchers.is;

import static org.hamcrest.MatcherAssert.assertThat;

import static org.junit.jupiter.api.Assertions.assertEquals;

import org.junit.jupiter.api.Test;

class S57Test {

@Test

void speedPlain() {

double actual = S57.speed(100, 9.58);

assertEquals(actual, 10.438, 0.001);

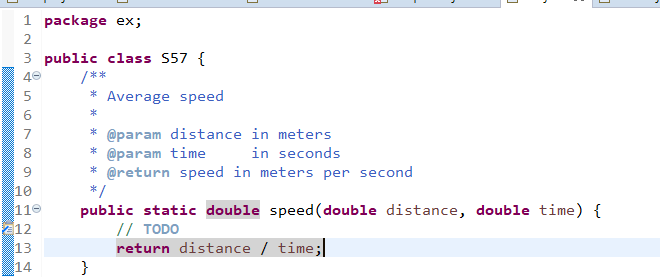
}

@Test

void speedInfinity() {

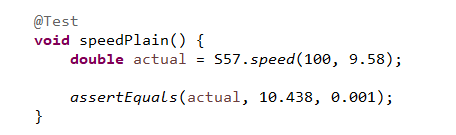
double actual = S57.speed(100, 0);

assertThat(actual, is(Double.POSITIVE\_INFINITY));

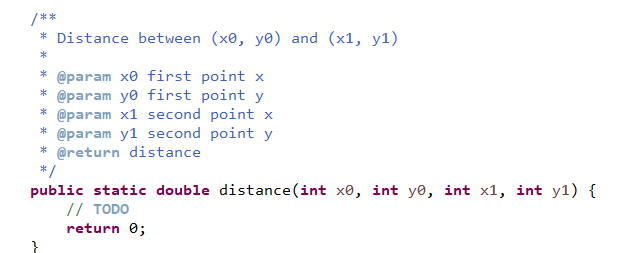


Invece di return 0,..

Parametri distanza e tempo sono double, li prendo per calcolare la velocità.



Prende due interi e ritorna un double facendo un widening (quindi non dà problemi).



Uso metodo Math sqrt

**package** ex;

**public** **class** S58 {

/\*\*

\* Check if the parameter is positive, negative, or zero

\*

\* **@param** value an integer

\* **@return** "positive", "negative", or "zero"

\*/

**public** **static** String checkSign(**int** value) {

**int** value;

{

**if**(value < 0); {

**return** "negative";

} **else** **if**(value > 0); {

**return** "positive";

}

**return** "zero";

}

}

/\*\*

\* Check if the parameter is odd

\*

\* **@param** value an integer

\* **@return** "odd" or "even"

\*/

**public** **static** **boolean** isOdd(**int** value) {

// **TODO**

**int** value = 0; {

**if** **int** value = 0; {

**return** **false**;

}

**return** **true**;

}

}

/\*\*

\* Convert the parameter to its English name, if it is in [0..9]

\*

\* **@param** value an integer

\* **@return** "zero" for 0 ... "nine" for 9, or "other"

\*/

**public** **static** String asWord(**int** value) {

// **TODO**

**return** "";

}

/\*\*

\* Convert a value in the interval [0, 100] to a letter in [A, F]

\*

\* A if value > 90 B if value in (80, 90] ... F if value <= 50

\*

\* **@param** percentile in [0, 100]

\* **@return** a letter in [A, F]

\*/

**public** **static** **char** vote(**double** percentile) {

// **TODO**

**return** 'F';

}

/\*\*

\* Leap year checker

\*

\* **@param** year

\* **@return** true if leap year

\*/

**public** **static** **boolean** isLeapYear(**int** year) {

// **TODO**

**return** **false**;

}

/\*\*

\* Sort the input parameters

\*

\* **@param** a

\* **@param** b

\* **@param** c

\* **@return** a sorted array

\*/

**public** **static** **int**[] sort(**int** a, **int** b, **int** c) {

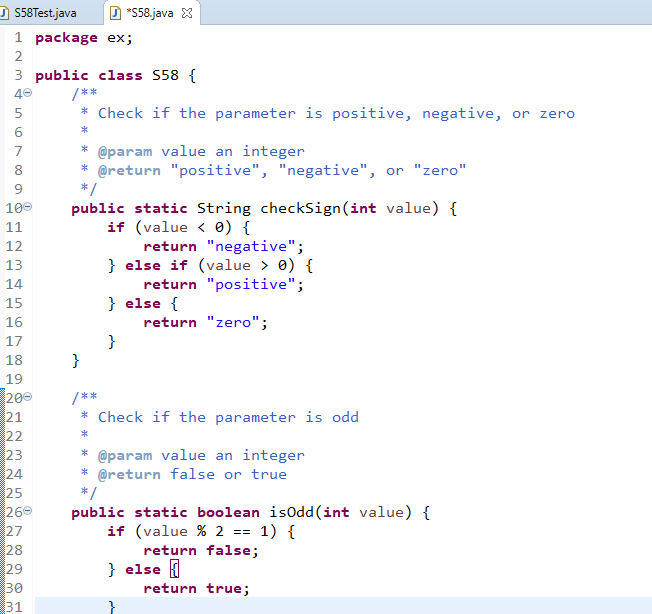
**int**[] result = **new** **int**[3];

// **TODO**

**return** result;

}

}



17.12.19

/\*\*

\* Convert a value in the interval [0, 100] to a letter in [A, F]

\*

\* A if value > 90 B if value in (80, 90] ... F if value <= 50

\*

\* **@param** percentile in [0, 100]

\* **@return** a letter in [A, F]

\*/

**public** **static** **char** vote(**double** percentile) {

**if** (percentile > 90) {

**return** 'A';

} **else** **if** (percentile > 80) {

**return** 'B';

} **else** **if** (percentile > 70) {

**return** 'C';

} **else** **if** (percentile > 60)

**return** 'C';

**else** **if** (percentile > 50) {

**return** 'E';

} **else**

**return** 'F';

}

/\*\*

\* Leap year checker

\*

\* **@param** year

\* **@return** true if leap year

\*/

**public** **static** **boolean** isLeapYear(**int** year) {

**if** (year % 400 == 0) {

**return** **true**;

} **else** **if** (year % 4 == 0 && !% year % 100 == 0) {

} **return** **true**;

} **else** {

**return** **false**;

}

**public** **static** **int**[] sort(**int** a, **int** b) {

**int**[] result = **new** **int**[2];

**if** (a < b) {

result[0] = a;

result[1] = b;

} **else** {

result[0] = b;

result[1] = a;

}

**public** **static** **int**[] sort(**int** a, **int** b, **int** c) {

**int**[] result = **new** **int**[3];

**if** (a < b) {

**public** **static** **int**[] sort(**int** a, **int** b, **int** c) {

**int**[] result = **new** **int**[3];

**if** (a < b) {

**if**(c < a) {

result[0] = c;

result[1] = a;

result[2] = b;

} **else** **if** (c > b) {

result[0] = a;

result[1] = b;

result[2] = c;

} **else** {

result[0] = a;

result[1] = c;

result[2] = b;

}

}

**return** result;

}

}

Però manca la parte in cui a>b e c>b perché i due if non sono necessariamente veri.

**package** ex;

**public** **class** S59 {

/\*\*

\* Add up all the numbers in the passed closed interval

\*

\* **@param** first the left limit

\* **@param** last the right limit

\* **@return** the sum of all the numbers, or zero

\*/

**public** **static** **long** sum(**int** first, **int** last) {

**long** result = 0;

**for** (**int** cur = first; cur <= last; cur++)

result += cur;

// result = result++cur;

**return** result;

}

/\*\*

\* Add up only the even numbers in the passed closed interval

\*

\* **@param** first the left limit

\* **@param** last the right limit

\* **@return** the sum of all the even numbers, or zero

\*/

**public** **static** **long** evenSum(**int** first, **int** last) {

**long** result = 0;

**for** (**int** cur = first; cur <= last; cur++) {

**if** (cur % 2 == 0) {

result += cur;

// result = result++cur;

}

}

**return** result;

}

Non ho nessun array, per cui scrivo int cur invece di int[].

**public** **static** **int**[][] multiplicationTable(**int** value) {

**int**[][] result = **new** **int**[0][0];

**for** (**int** i = 0; i < value; i++) {

**for** (**int** j = 0; j < value; j++)

result[i][j] = (i + 1) \* (i + 1);

}

**return** result;

}

}

Fibonacci parte da zero

Reverse:

**package** ex;

**public** **class** S60 {

/\*\*

\* Reverse a string

\*

\* **@param** s

\* **@return** the input reversed

\*/

**public** **static** String reverse(String s) {

StringBuilder sb = **new** StringBuilder();

**for** (**int** i = s.length() - 1; i >= 0; i--) { // s.length-1 perchè parto dal fondo, i-- perchè devo decrescere e

// non// incrementare

sb.append(s.charAt(i)); // "prendimi l'ultimo carattere e mettilo in prima posizione, ma come fa la

// macchina a capire che deve metterlo in prima posizione???"

}

**return** sb.toString(); // perchè sto lavorando su stringhe, e quindi devo convertire

}

Palindromo: Il programma deve indicare se una parola é palindrome oppure no. Ad  
esempio otto è una parola palindrome perché si legge allo stesso modo sia da  
destra che da sinistra. Per questo programma utilizzare il metodo chatAt della  
classe String;

**public** **static** **boolean** isPalindrome(String s) {

**int** len = s.length();

**for** (**int** i = 0; i < s.length() / 2; i++) {

**if** (s.charAt(i) != s.charAt(len - 1 - i)) {

**return** **false**;

}

}

**return** **true**;

}

18.12.19

**package** s070;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

Barker[] barkers = **new** Barker[2]; //faccio un array di barkers con spazio per 2 barker mettendoci NULL. se voglio metterci dentro qualcosa per non lasciare null e per implementare l'interfaccia, metto le classi dog o fox

// barker b = new Barker(); non posso farlo perchè l'interfaccia non ha costruttore (perchè non è una vera classe)

barkers[0] = **new** Fox();

barkers[1] = **new** Dog();

**for**(Barker barker: barkers) { //for each: per ogni elemento dell'array barker stampa

System.***out***.println(barker.bark());

}

BarkAndWag baw = **new** Dog();

System.***out***.println(baw.tailWaggingSpeed());

Dog dog = **new** Dog();

System.***out***.println(dog.bark(3));

}

}

**package** s070;

**public** **class** Fox **implements** Barker {

@Override

**public** String bark() {

**return** "yap!";

**package** s070;

**public** **class** Dog **implements** BarkAndWag {

@Override

**public** String bark() {

**return** "woof!";

}

**public** String bark(**int** count) {

StringBuilder sb = **new** StringBuilder();

**for** (**int** i = 0; i < count; i++) {

sb.append(bark());

}

**return** sb.toString();

}

@Override

**public** **int** tailWaggingSpeed() {

**return** BarkAndWag.***AVG\_WAGGING\_SPEED***;

}

}

**package** s070;

**interface** Barker {

String bark();

}

**package** s070;

**interface** BarkAndWag **extends** Barker {

**int** ***AVG\_WAGGING\_SPEED*** = 12;

**int** tailWaggingSpeed();

}

**package** s072;

// The type Concrete must implement the inherited abstract method

//public class Concrete extends Abstract {

//}

**public** **class** Concrete **extends** Abstract { //per creare il persiano devo estendere la classe astratta, poi devo fare override del metodo astratot di presente il cat

@Override

**public** **void** anAbstractMethod() {

System.***out***.println("not abstract anymore");

}

}

**package** s072;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

// Cannot instantiate

// Abstract ma = new Abstract();

// Abstract2 ma2 = new Abstract2();

Abstract ma = **new** Concrete();

ma.anAbstractMethod(); //chiamo metodo meow che è stato definito in class cat come astratto, ma con override l'ho ridefinito come concreto

Abstract2 ma2 = **new** Concrete2();

ma2.aConcreteMethod();

}

}

**package** s073;

**public** **abstract** **class** Mammal { //quindi non posso creare oggetti di tipo Mammal

**protected** **int** gestation; //cosa differenzia i mammiferi dagli altri animali

**public** Mammal(**int** gestation) { //è un metodo costruttore perchè ha lo stesso nome della classe. gestation=parametro

**this**.gestation = gestation; //this distingue il primo gestation (variabile) che appartiene all'oggetto mammal corrente (la proprietà) dal secondo (valore passato dall'utente)

}

@Override //se non è specificata una classe madre, sottintende che sia object

**public** String toString() { //metodo che mi ritorna una nuova stringa contenente

**return** "Mammal [gestation=" + gestation + "]";

}

}

**package** s073;

**public** **class** Cat **extends** Mammal { //posso creare oggetti di tipo cat

**private** Tail tail; //proprietà privata di tipo tail (classe tail)

**public** Cat(**int** gestationDays, **int** tailLen) { //met. costruttore (se fosse un metodo avrebbe un tipo, un return type) con 2 parametri

**super**(gestationDays); //super=serve a chiamare il costruttore della classe Cat.

**this**.tail = **new** Tail(tailLen);

}

@Override

**public** String toString() {

**return** "Cat [tail=" + tail + ", gestation=" + gestation + "]"; //tail è diventata stringa grazie al passaggio precedente, che è valido per ogni oggetto, gestation viene da classe madre e riesco a vederlo perchè nella classe madre è protected (quindi si può vedere da classe figlia a madre)

}

}

**package** s073;

**public** **class** Tail {

**private** **int** size;

**public** Tail(**int** size) { //metodo costruttore inizializza proprietà size

**this**.size = size; //se non mettessi this, direi che size è = a sè stessa

}

**public** **int** getSize() { //metodo che ritorna valore size. size non è modificabile, dovrei creare un metodo

**return** size;

}

@Override

**public** String toString() {

**return** "Tail [size=" + size + "]";

}

}

**package** s073;

**public** **class** Dog **extends** Mammal {

**private** Tail tail;

**public** Dog(**int** gestationDays, **int** tailLen) {

**super**(gestationDays);

**this**.tail = **new** Tail(tailLen);

}

@Override

**public** String toString() {

**return** "Dog [tail=" + tail + ", gestation=" + gestation + "]";

}

}

**package** s073;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

Mammal[] mammals = **new** Mammal[2]; //faccio un array di mammal

mammals[0] = **new** Dog(63, 40); //creo cane e gatto

mammals[1] = **new** Cat(63, 20);

**for** (Mammal mammal : mammals) { //per ogni elemento di mammal print

System.***out***.println(mammal.toString());

}

}

}