

Programmazione I-B 2020-21

Laboratorio T2

Attilio Fiandrotti

attilio.fiandrotti@unito.it

15 Ottobre 2020

Outline

- Soluzione e analisi esercizio ordinamento con scambio
- Modello di Memoria JVM
- I metodi in linguaggio Java
- Esercizi con booleani

Soluzione e analisi esercizio ordinamento con scambio

ESERCIZIO – Ordinamento con Scambio

- Ispirandovi all'esercizio *emersione massimo* visto in aula e utilizzando il costrutto *while*, scrivere un programma che date quattro variabili a, b, c, d esegua le opportune permutazioni a coppie di variabili adiacenti per cui infine $a \leq b \leq c \leq d$
- Suggerimento: iterare su *emersione massimo*
 - Quante volte iterare ?
 - Quale variabile ausiliaria ?
 - Come uscire dal ciclo ?

ESERCIZIO – Ordinamento con Scambio

```
1  /** OBIETTIVO.  
2   Date quattro variabili a, b, c, d, scrivere un algoritmo che  
3   riorganizzi i valori in esse contenuti, in modo che, al termine,  
4   la variabile d contenga il valore massimo, inizialmente in a, b ,c ,d.  
5  
6   ESEMPIO.  
7   Date le assegnazioni  
8       a = 3; b = 11; c = 8; d = 2;  
9   occorre produrre una configurazione finale tale che:  
10      d==massimo{3, 11, 8, 2}==11  
11   in cui non si mettono vincoli su cosa a, b, c debbano  
12   contenere. */  
13  
14  if (a > b) {  
15      tmp = a; a = b; b = tmp;  
16  }  
17  if (b > c) {  
18      tmp = b; b = c; c = tmp;  
19  }  
20  if (c > d) {  
21      tmp = c; c = d; d = tmp;  
22  }  
23  
24  /* DISPENSE  
25  Sezione 2.2. */
```

ESERCIZIO – Ordinamento con Scambio

```
1 public class OrdinamentoConScambio {
2     public static void main(String[] args) {
3
4         // Valori iniziali delle 4 variabili da ordinare, ignote al momento della scrittura di programma
5         int a = 12; int b = -6; int c = 3; int d = -1;
6         // Variabile di appoggio per lo scambio
7         int tmp;
8         // La variabile binaria di permanenza nel loop si chiamerà eseguiLoop
9         // e indica la necessità di eseguire un loop addizionale
10        // DOMANDA: a quale valore dovrà essere inizializzata tale variabile e perché ?
11        boolean eseguiLoop = ???;
12        // Contatore di iterazioni strumentale alla stampa di debug senza funzionalità algoritmiche
13        int contaIterazioni = 1;
14        // DOMANDA: Quale sarà la condizione di permanenza nel loop relativamente alla variabile eseguiLoop
15        while (eseguiLoop == ???) {
16            // Stampo la configurazione delle variabili all'inizio del ciclo
17            System.out.println(contaIterazioni + " ) " + a + " " + b + " " + c + " " + d);
18            // Aggiorno il contatore per un'eventuale stampa nel ciclo successivo
19            contaIterazioni++;
20
21            // DOMANDA: che operazione devo effettuare sul flag di permanenza del
22            // loop ad inizio ciclo in relazione alla necessità di restare nel
23            // ciclo unicamente se necessario ?
24            eseguiLoop = ???;
25
26            // DOMANDA: in che caso scambio a e b ?
27            if (???) {
28                // DOMANDA: Come eseguo lo scambio fra a e b ?
29                ???
30                // DOMANDA: come impostare il flag eseguiLoop in caso di scambio ?
31                eseguiLoop = ???;
32            }
33
34            // DOMANDA: Si applichino considerazioni analoghe al caso sopra per (b,c) e (c,d)
35
36        } // Fine del ciclo while
37    } // Fine del metodo main
38 }
```

ESERCIZIO – Ordinamento con Scambio

```
1 public class OrdinamentoConScambio {
2     public static void main(String[] args) {
3
4         // Valori iniziali delle 4 variabili da ordinare, ignote al momento della scrittura di programma
5         int a = 12; int b = -6; int c = 3; int d = -1;
6         // Variabile di appoggio per lo scambio
7         int tmp;
8         // La variabile binaria di permanenza nel loop si chiamerà eseguiLoop
9         // e indica la necessità di eseguire un loop addizionale
10        // DOMANDA: a quale valore dovrà essere inizializzata tale variabile e perché ?
11        boolean eseguiLoop = ???;
12        // Contatore di iterazioni strumentale alla stampa di debug senza funzionalità algoritmiche
13        int contaIterazioni = 1;
14        // DOMANDA: Quale sarà la condizione di permanenza nel loop relativamente alla variabile eseguiLoop
15        while (eseguiLoop == ???) {
16            // Stampo la configurazione delle variabili all'inizio del ciclo
17            System.out.println(contaIterazioni + " ) " + a + " " + b + " " + c + " " + d);
18            // Aggiorno il contatore per un'eventuale stampa nel ciclo successivo
19            contaIterazioni++;
20
21            // DOMANDA: che operazione devo effettuare sul flag di permanenza del
22            // loop ad inizio ciclo in relazione alla necessità di restare nel
23            // ciclo unicamente se necessario ?
24            eseguiLoop = ???;
25
26            // DOMANDA: in che caso scambio a e b ?
27            if (???) {
28                // DOMANDA: Come eseguo lo scambio fra a e b ?
29                ???
30                // DOMANDA: come impostare il flag eseguiLoop in caso di scambio ?
31                eseguiLoop = ???;
32            }
33
34            // DOMANDA: Si applichino considerazioni analoghe al caso sopra per (b,c) e (c,d)
35
36        } // Fine del ciclo while
37    } // Fine del metodo main
38 }
```

ESERCIZIO – Ordinamento con Scambio (1)

```
1 // DOMANDA: modificare il programma sottostante per stampare a schermo anche il numero tot
2 public class OrdinamentoConScambio {
3     public static void main(String[] args) {
4
5         // Valori iniziali delle 4 variabili da ordinare, ignote al momento della scrittura
6         int a = 12; int b = -6; int c = 3; int d = -1;
7         // Variabile di appoggio per lo scambio
8         int tmp;
9         // La variabile binaria di permanenza nel loop si chiamerà eseguiLoop
10        // DOMANDA: perché è inizializzata a true ?
11        boolean eseguiLoop = true;
12        // Contatore di iterazioni strumentale alla stampa di debug senza funzionalità a
13        int contaIterazioni = 1;
14        // La condizione di permanenza nel ciclo while() è eseguiLoop == true
15        while (eseguiLoop == true) {
16            // Stampo la configurazione delle variabili all'inizio del ciclo
17            System.out.println(contaIterazioni + " ) " + a + " " + b + " " + c + " " + d);
18            // Aggiorno il contatore per un'eventuale stampa nel ciclo successivo
19            contaIterazioni ++;
```


ESERCIZIO – Ordinamento con Scambio (2)

```
20
21 //DOMANDA: perché il flag eseguiLoop viene impostato a false incondizionatamen
22 eseguiLoop = false;
23
24 // Scambio a e b nel caso in cui a > b
25 if (a>b) {
26     // Eseguo lo scambio fra a e b
27     tmp = a; a = b; b = tmp;
28     // Imposto il flag a true in caso di scambio effettuato
29     // DOMANDA: perché il flag viene impostato a true solo in caso di scambio
30     eseguiLoop = true;
31 }
32
33 // Valgono considerazioni analoghe al caso sopra
34 if (b>c) {
35     tmp = b; b = c; c = tmp; eseguiLoop = true;
36 }
37 if (c>d) {
38     tmp = c; c = d; d = tmp; eseguiLoop = true;
39 }
40
41 }
42 }
43 }
44 }
```

ESERCIZIO – Ordinamento con Scambio

Stato <u>inizio</u> iterazione #	a	b	c	d	esegui ulteriore loop
0	12	-6	3	1	y
1					
2					
3					
4					

Modello di memoria JVM

Un semplice esempio

- Scrivere un programma Java che dato il raggio di un cerchio ne calcoli la circonferenza

```
1 public class CirconferenzaMain {  
2     public static void main (String[] args) {  
3         float pi = (float)3.1415;  
4         float raggio = (float)5.0;  
5         float circ = 2 * raggio * pi;  
6         System.out.println("Raggio " + raggio + " -> circonferenza " + circ);  
7     }  
8 }
```

Un semplice esempio

- Scrivere un programma Java che dato il raggio di un cerchio ne calcoli la circonferenza

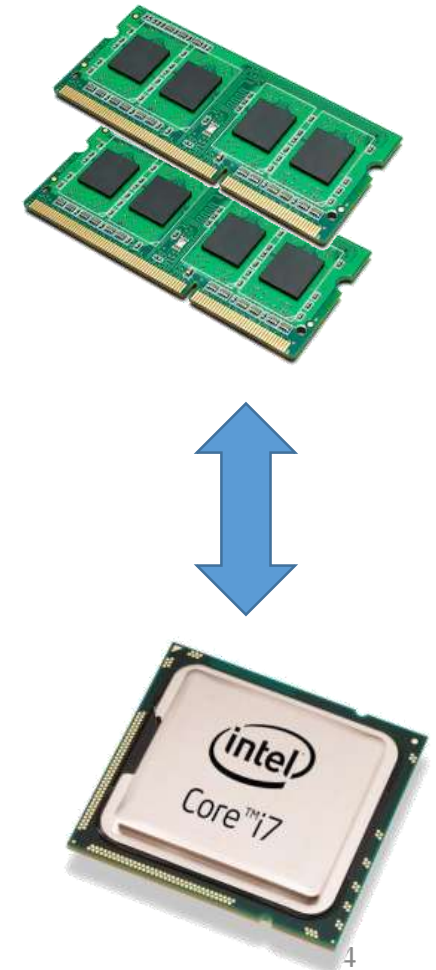
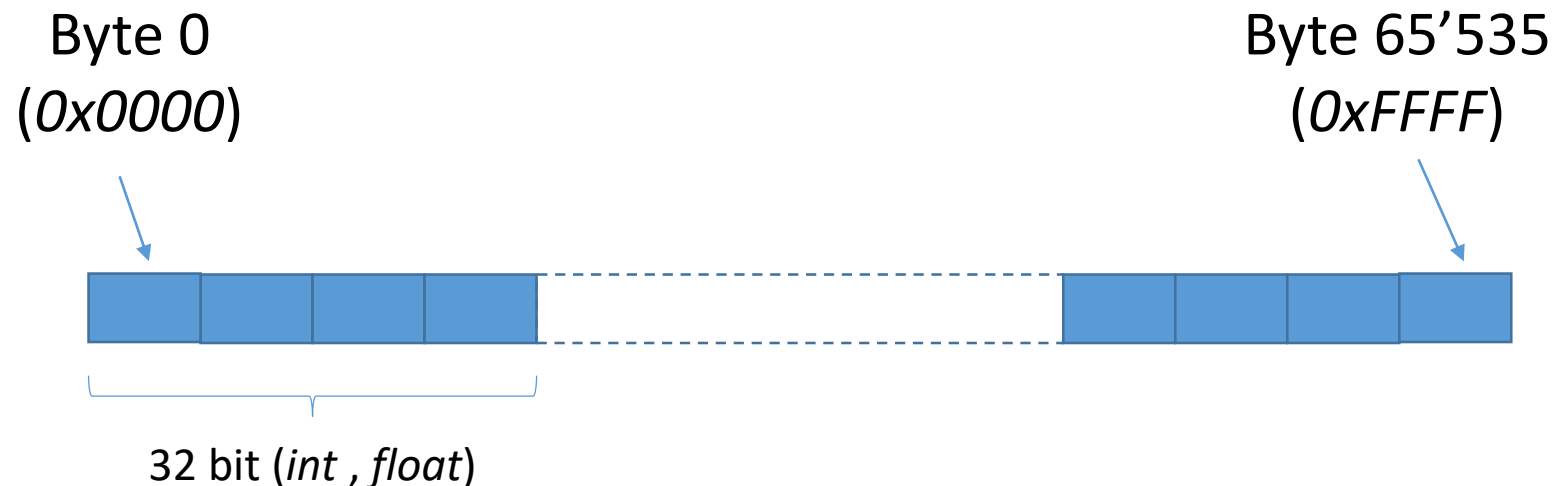
Queste variabili risiedono in memoria,
ma come sono organizzate ?



```
1 public class CirconferenzaMain {  
2     public static void main (String[] args) {  
3         float pi = (float) 3.1415;  
4         float raggio = (float) 5.0;  
5         float circ = 2 * raggio * pi;  
6         System.out.println("Raggio " + raggio + " -> circonferenza " + circ);  
7     }  
8 }
```

La memoria vista dalla CPU

- I dati (variabili) di un programma risiedono in memoria
- La memoria é «vista» dalla CPU come un vettore di bytes
 - Ogni byte é leggibile/scrivibile indipendentemente
 - Ad ogni byte corrisponde un *indirizzo* (es., 16 bit in JVM)
 - Q: quanta memoria é possibile *indirizzare* ?



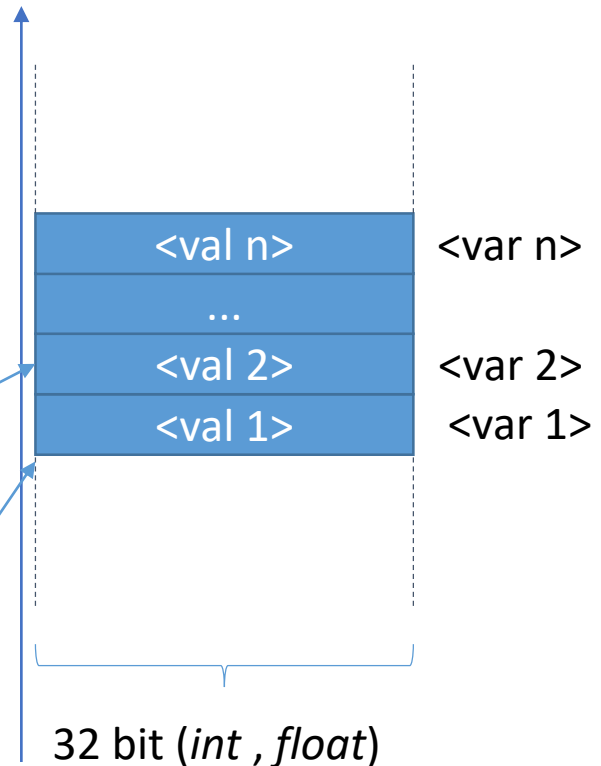
Il modello di memoria JVM (semplificato)

- Due (tre) aree di memoria per JVM
 - Frame (+operand) stack : useremo questa
 - Heap: per il momento non la useremo
- Un'area di memoria (*frame*) per metodo
 - Il primo indirizzo in tale area é la *base* del frame
- Variabili allocate come una *pila/stack/LIFO*
 - N variabili -> N *push* sullo stack

Prima variabile (0x1028)

Base del frame (es, 0x1024)

Indirizzi crescenti




Il modello di memoria JVM (semplificato)

Prossima linea da eseguire
(non ancora eseguita)

Base del frame di
main() (es, 0x1024)



32 bit (*int* , *float*)

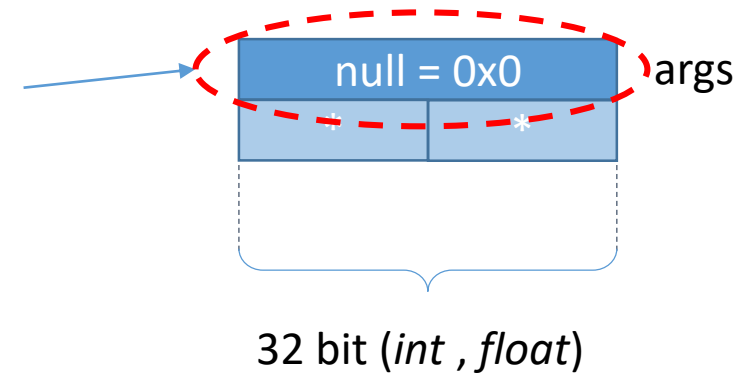


```
1 public class CirconferenzaMain {  
2     public static void main (String[] args) {  
3         float pi = (float)3.1415;  
4         float raggio = (float)5.0;  
5         float circ = 2 * raggio * pi;  
6         System.out.println("Raggio " + raggio + " -> circonferenza " + circ);  
7     }  
8 }
```


Il modello di memoria JVM (semplificato)

Prossima linea da eseguire
(non ancora eseguita)

`*args == 0x1028`

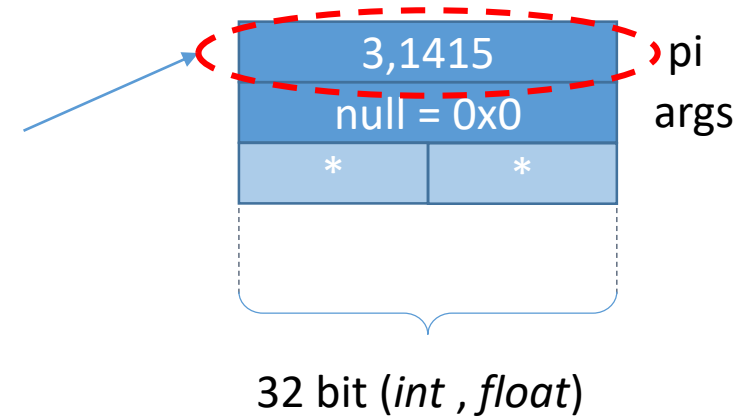


```
1 public class CirconferenzaMain {  
2     public static void main (String[] args) {  
3         float pi = (float)3.1415;  
4         float raggio = (float)5.0;  
5         float circ = 2 * raggio * pi;  
6         System.out.println("Raggio " + raggio + " -> circonferenza " + circ);  
7     }  
8 }
```

Il modello di memoria JVM (semplificato)

Prossima linea da eseguire
(non ancora eseguita)

*pi == 0x1032

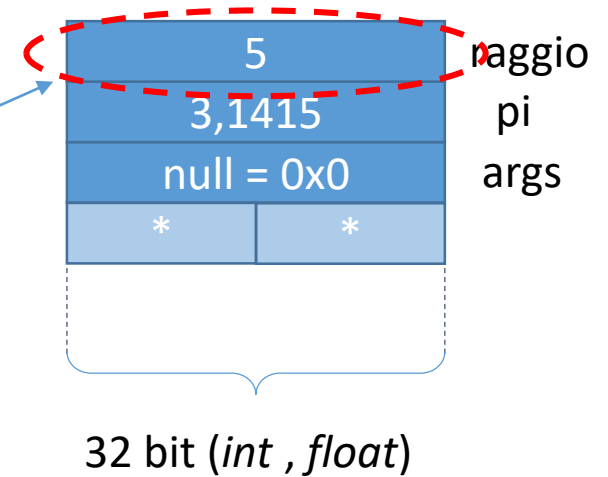


```
1 public class CirconferenzaMain {  
2     public static void main (String[] args) {  
3         float pi = (float)3.1415;  
4         float raggio = (float)5.0;  
5         float circ = 2 * raggio * pi;  
6         System.out.println("Raggio " + raggio + " -> circonferenza " + circ);  
7     }  
8 }
```

Il modello di memoria JVM (semplificato)

Prossima linea da eseguire
(non ancora eseguita)

`*raggio == 0x1036`



```
1 public class CirconferenzaMain {  
2     public static void main (String[] args) {  
3         float pi = (float)3.1415;  
4         float raggio = (float)5.0;  
5         float circ = 2 * raggio * pi;  
6         System.out.println("Raggio " + raggio + " -> circonferenza " + circ);  
7     }  
8 }
```

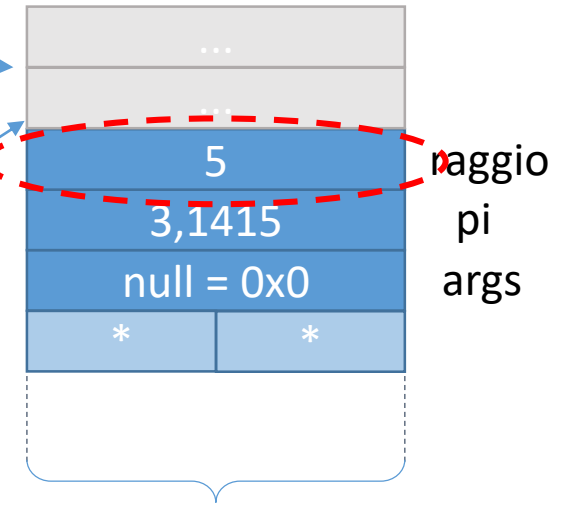
Il modello di memoria JVM (semplificato)

Prossima linea da eseguire
(non ancora eseguita)

Evoluzione operand stack

$2 * pi * raggio$
omessa

0x1036



32 bit (int , float)

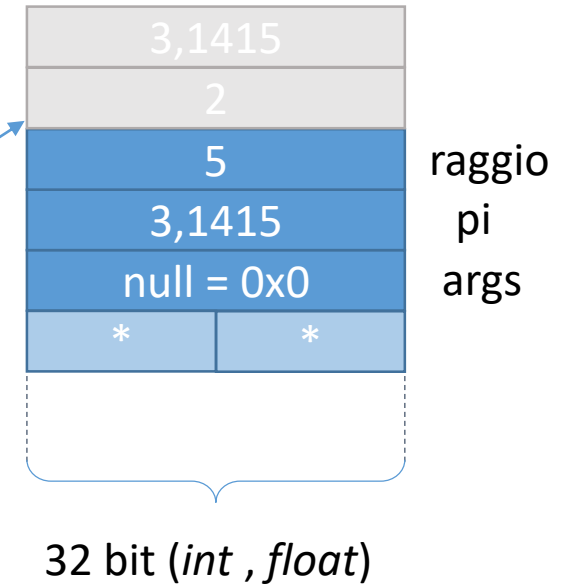
```
1 public class CirconferenzaMain {  
2     public static void main (String[] args) {  
3         float pi = (float)3.1415;  
4         float raggio = (float)5.0;  
5         float circ = 2 * raggio * pi;  
6         System.out.println("Raggio " + raggio + " -> circonferenza " + circ);  
7     }  
8 }
```

Il modello di memoria JVM (semplificato)

Prossima linea da eseguire
(non ancora eseguita)

Evoluzione operand stack
per $2 * pi$

0x1036



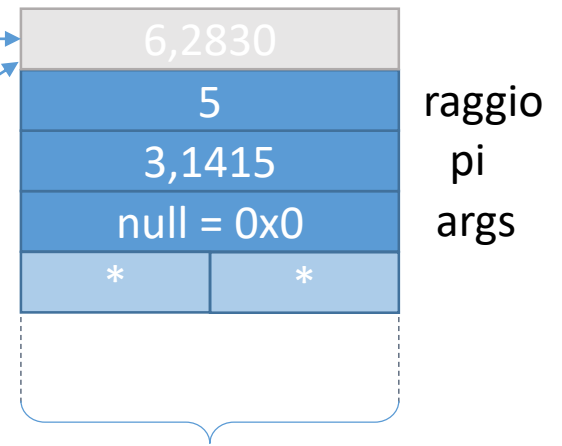
```
1 public class CirconferenzaMain {  
2     public static void main (String[] args) {  
3         float pi = (float)3.1415;  
4         float raggio = (float)5.0;  
5         float circ = 2 * raggio * pi;  
6         System.out.println("Raggio " + raggio + " -> circonferenza " + circ);  
7     }  
8 }
```

Il modello di memoria JVM (semplificato)

Prossima linea da eseguire
(non ancora eseguita)

Evoluzione operand stack
per $2 * pi$

0x1036



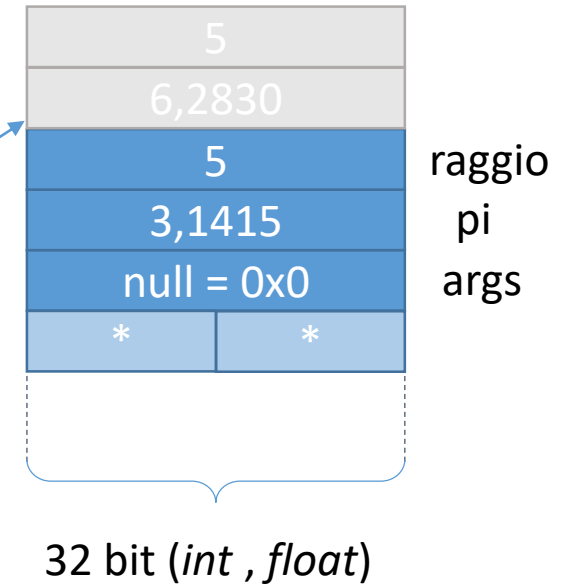
```
1 public class CirconferenzaMain {  
2     public static void main (String[] args) {  
3         float pi = (float)3.1415;  
4         float raggio = (float)5.0;  
5         float circ = 2 * raggio * pi;  
6         System.out.println("Raggio " + raggio + " -> circonferenza " + circ);  
7     }  
8 }
```

Il modello di memoria JVM (semplificato)

Prossima linea da eseguire
(non ancora eseguita)

Evoluzione operand stack
per $(2 * \pi) * \text{raggio}$

0x1036



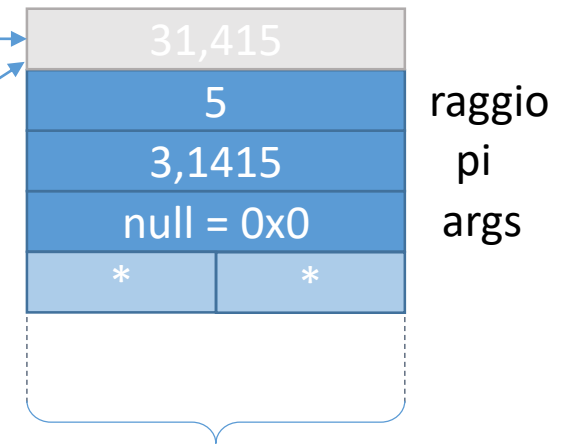
```
1 public class CirconferenzaMain {  
2     public static void main (String[] args) {  
3         float pi = (float)3.1415;  
4         float raggio = (float)5.0;  
5         float circ = 2 * raggio * pi;  
6         System.out.println("Raggio " + raggio + " -> circonferenza " + circ);  
7     }  
8 }
```

Il modello di memoria JVM (semplificato)

Prossima linea da eseguire
(non ancora eseguita)

Risultato $(2 * \pi) * \text{raggio}$

0x1036



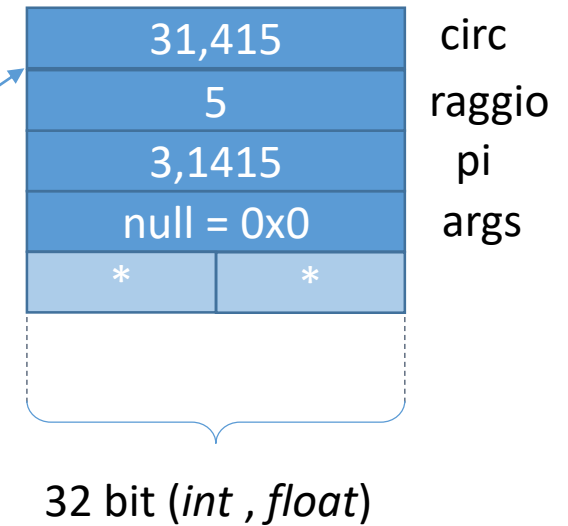
32 bit (int, float)

```
1 public class CirconferenzaMain {  
2     public static void main (String[] args) {  
3         float pi = (float)3.1415;  
4         float raggio = (float)5.0;  
5         float circ = 2 * raggio * pi;  
6         System.out.println("Raggio " + raggio + " -> circonferenza " + circ);  
7     }  
8 }
```


Il modello di memoria JVM (semplificato)

Prossima linea da eseguire
(non ancora eseguita)

`*circ == 0x1036`



```
1 public class CirconferenzaMain {  
2     public static void main (String[] args) {  
3         float pi = (float)3.1415;  
4         float raggio = (float)5.0;  
5         float circ = 2 * raggio * pi;  
6         System.out.println("Raggio " + raggio + " -> circonferenza " + circ);  
7     }  
8 }
```

Java Visualizer



Write your Java code here:

```
1 public class ClassNameHere {  
2     public static void main(String[] args) {  
3  
4     }  
5 }
```

options

args:

(also visualizes consumption of [StdIn](#))

Visualize execution

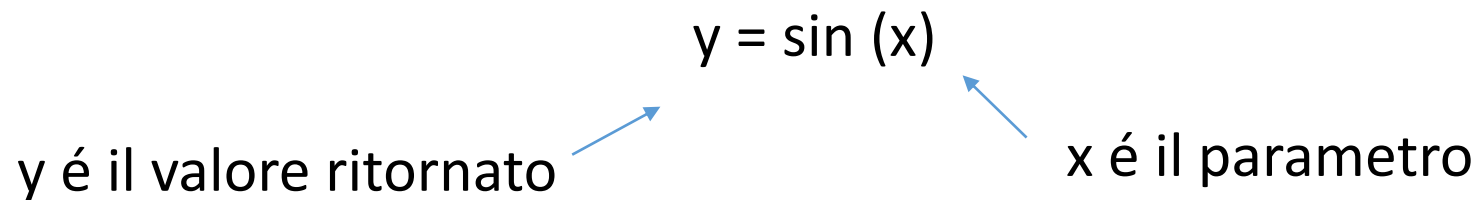
basic examples | [\(Default\)](#) | [Variables](#) | [CmdLineArgs](#) | [StdIn](#) | [ControlFlow](#) | [Sqrt](#) | [ExecLimit](#) | [Strings](#)
method examples | [PassByValue](#) | [Recursion](#) | [StackOverflow](#)
oop examples | [Rolex](#) | [Person](#) | [Complex](#) | [Casting](#)
data structure examples | [LinkedList](#) | [StackQueue](#) | [Postfix](#) | [SymbolTable](#)
java feature examples | [ToString](#) | [Reflect](#) | [Exception](#) | [ExceptionFlow](#) | [TwoClasses](#)

https://cscircles.cemc.uwaterloo.ca/java_visualize/

I metodi in Java

I Metodi

- I metodi rappresentano un blocco di codice invocabile dall'esterno
 - Per noi *metodi* e *funzioni* termini equivalenti
- Possono accettare in input parametri
- Possono restituire un valore
- Analogia: la funzione trigonometrica *sin()*



Il Metodo *main*

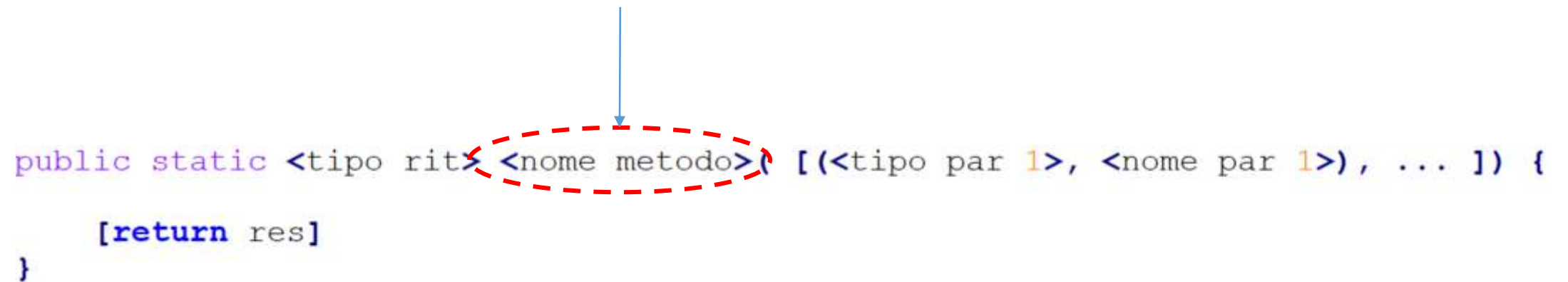
- Ve ne deve essere esattamente uno per ogni programma
- E' il punto di ingresso (e uscita) del nostro programma
- Dichiarato obbligatoriamente come *public static void*
- Rende disponibili i parametri passati via linea da comando al programma Java nell'array di stringhe *args*
- *Può ritornare un codice d'uscita alla console*

Altri metodi – Motivazioni

- Riutilizzabilità codice specializzato
 - Es: funzioni matematiche *IEEE754*
- Compattezza del codice (storico, embedded)
 - Funzioni riutilizzabili
- Mantenibilità del software
 - Aggiornamento librerie di sistema (*.dll, .so*)
- Accesso all'hardware (es: console di input/output)
 - Call a funzioni *di sistema*

Dichiarazione di metodo Java

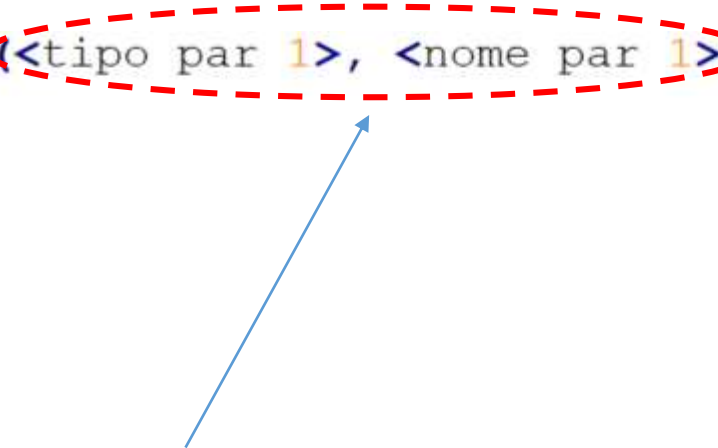
Nome simbolico metodo, unico nella classe



```
public static <tipo rit> <nome metodo> ( [<tipo par 1>, <nome par 1>, ... ] ) {  
    [return res]  
}
```

Dichiarazione di metodo Java

```
public static <tipo rit> <nome metodo>( [(<tipo par 1>, <nome par 1>), ... ] ) {  
    [return res]  
}
```



Coppie (<tipo parametro> <nome simbolico parametro>)
per ogni parametro accettato in input dal metodo

Dichiarazione di metodo Java


Table 5.1 · Some basic types

Value type	Range
Byte	8-bit signed two's complement integer (-2^7 to $2^7 - 1$, inclusive)
Short	16-bit signed two's complement integer (-2^{15} to $2^{15} - 1$, inclusive)
Int	32-bit signed two's complement integer (-2^{31} to $2^{31} - 1$, inclusive)
Long	64-bit signed two's complement integer (-2^{63} to $2^{63} - 1$, inclusive)
Char	16-bit unsigned Unicode character (0 to $2^{16} - 1$, inclusive)
String	a sequence of Chars
Float	32-bit IEEE 754 single-precision float
Double	64-bit IEEE 754 double-precision float
Boolean	true or false

<https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html>

Dichiarazione di metodo Java


Per ora sempre modificatori *public static* come main()



```
public static <tipo rit> <nome metodo>( [(<tipo par 1>, <nome par 1>), ... ] ) {  
    [return res]  
}
```

Dichiarazione di metodo Java

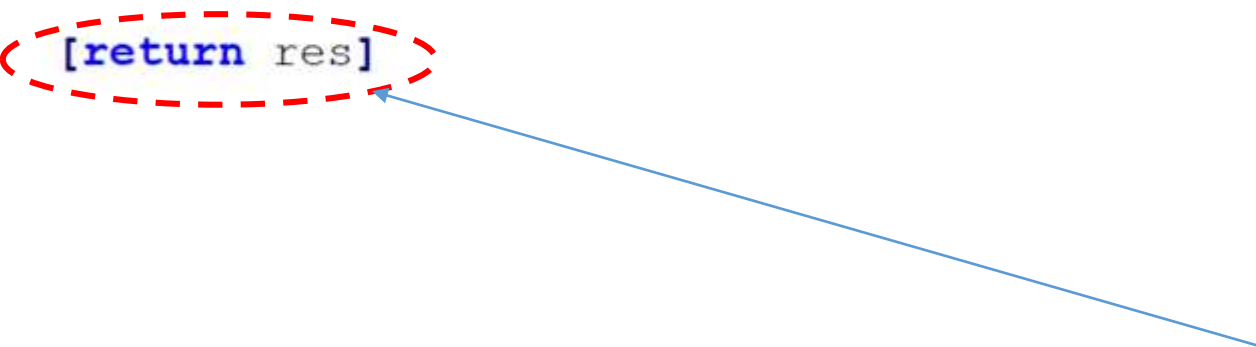
Tipo restituito, oppure tipo speciale *void*



```
public static <tipo rit> <nome metodo>( [(<tipo par 1>, <nome par 1>), ... ] ) {  
    [return res]  
}
```

Dichiarazione di metodo Java

```
public static <tipo rit> <nome metodo>( [(<tipo par 1>, <nome par 1>), ... ] ) {  
    [return res]  
}
```




Se il tipo restituito NON é void, specifica nome simbolico della variabile da restituire di tipo *tipo rit*

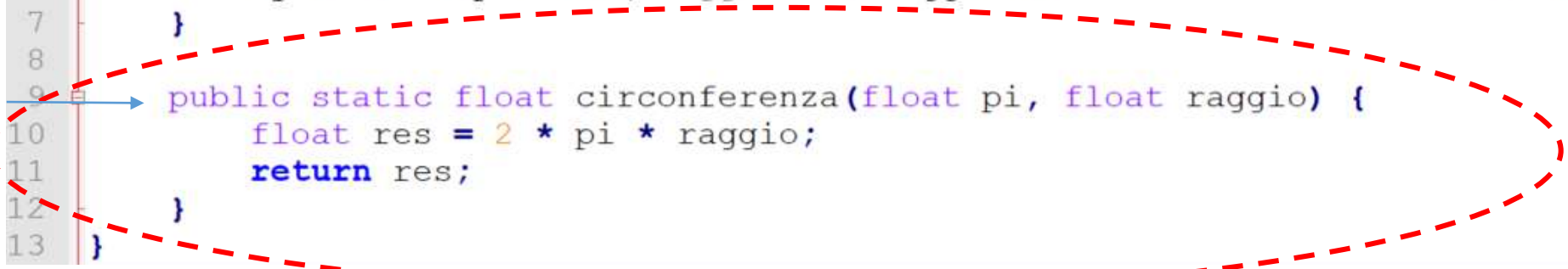
Esempio: calcolo della circonferenza

```
1 public class CirconferenzaMain {  
2     public static void main (String[] args) {  
3         float pi = (float)3.1415;  
4         float raggio = (float)5.0;  
5         float circ = 2 * raggio * pi;  
6         System.out.println("Raggio " + raggio + " -> circonferenza " + circ);  
7     }  
8 }
```

main() metodo
chiamante



```
1 public class Circonferenza {  
2     public static void main (String []args) {  
3         float pi = (float)3.1415;  
4         float raggio = 5;  
5         float circ = circonferenza(pi, raggio);  
6         System.out.println("Raggio " + raggio + " -> circonferenza " + circ);  
7     }  
8  
9     public static float circonferenza(float pi, float raggio) {  
10        float res = 2 * pi * raggio;  
11        return res;  
12    }  
13 }
```



circonferenza()
metodo chiamato

Esempio: calcolo della circonferenza

```
float circ = circonferenza(pi, raggio);
```

Restituisce un *float* su 32bit

Richiede in input due float a 32 bit
con nomi simbolici *pi* e *raggio*

```
public static float circonferenza(float pi, float raggio) {  
    float res = 2 * pi * raggio;  
    return res;  
}
```

Omonimia con variabili in *main()* casuale

Esempio: calcolo della circonferenza

In Java i parametri sono posizionali

```
float circ = circonferenza(pi, raggio);
```


```
public static float circonferenza(float pi, float raggio) {  
    float res = 2 * pi * raggio;  
    return res;  
}
```



```
public static float circonferenza(float topolino, float pippo) {  
    float res = 2 * topolino * pippo;  
    return res;  
}
```

Il modello di memoria JVM (semplificato)

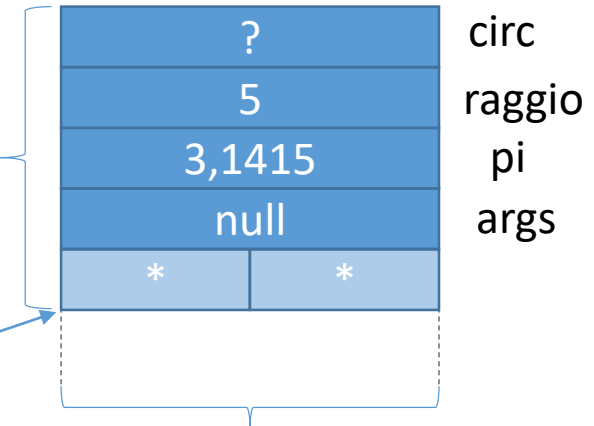
Prossima linea da eseguire (non ancora eseguita)



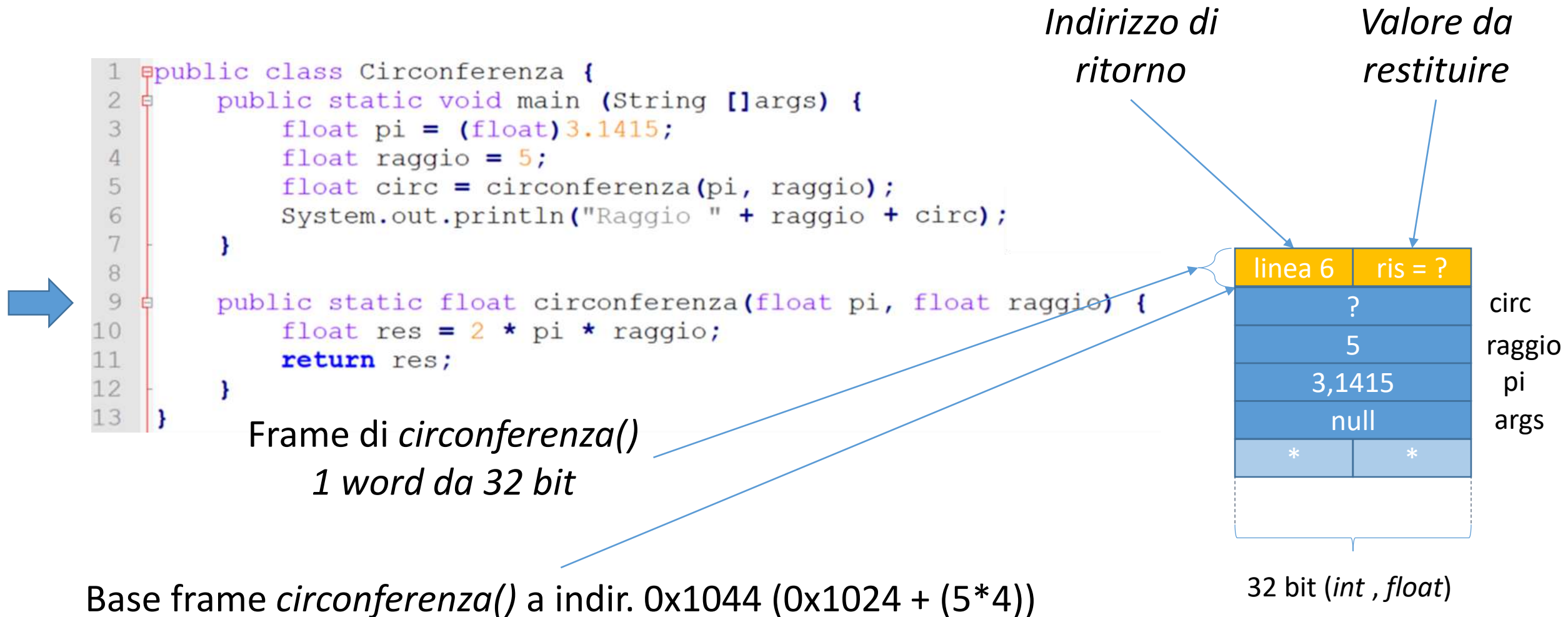
```
1 public class Circonferenza {
2     public static void main (String []args) {
3         float pi = (float)3.1415;
4         float raggio = 5;
5         float circ = circonferenza(pi, raggio);
6         System.out.println("Raggio " + raggio + circ);
7     }
8
9     public static float circonferenza(float pi, float raggio) {
10        float res = 2 * pi * raggio;
11        return res;
12    }
13 }
```

Frame di *main()*
5 words da 32 bit

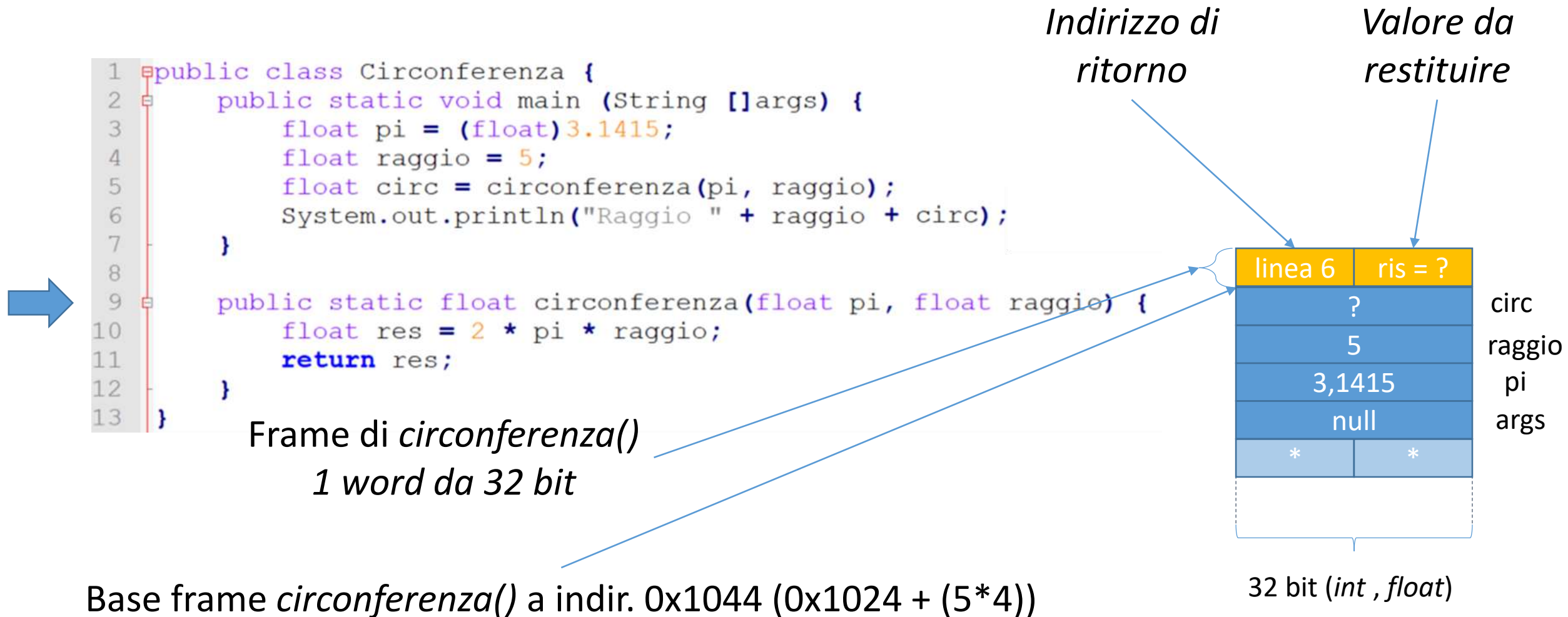
Base del frame , es: 0x1024



Il modello di memoria JVM (semplificato)



Il modello di memoria JVM (semplificato)

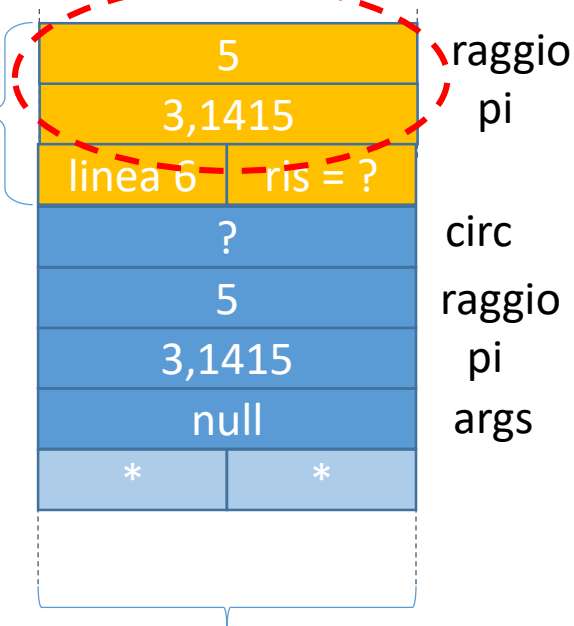


Il modello di memoria JVM (semplificato)

Parametri di *circonferenza()*
impilati nello stack

```
1 public class Circonferenza {  
2     public static void main (String []args) {  
3         float pi = (float)3.1415;  
4         float raggio = 5;  
5         float circ = circonferenza(pi, raggio);  
6         System.out.println("Raggio " + raggio + circ);  
7     }  
8  
9     public static float circonferenza(float pi, float raggio) {  
10        float res = 2 * pi * raggio;  
11        return res;  
12    }  
13 }
```

Frame di *circonferenza()*
3 words da 32 bit



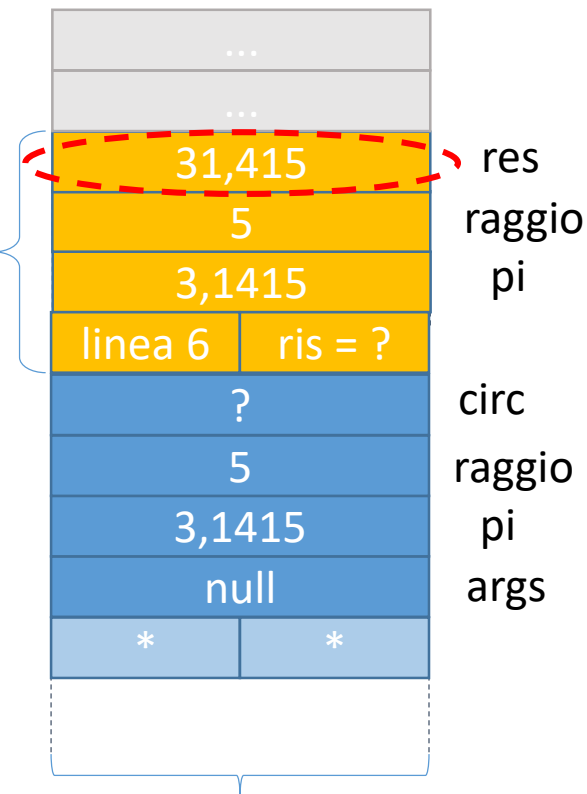
32 bit (int , float)

Il modello di memoria JVM (semplificato)

Evoluzione *operand*
stack omessa

```
1 public class Circonferenza {  
2     public static void main (String []args) {  
3         float pi = (float)3.1415;  
4         float raggio = 5;  
5         float circ = circonferenza(pi, raggio);  
6         System.out.println("Raggio " + raggio + circ);  
7     }  
8  
9     public static float circonferenza(float pi, float raggio) {  
10        float res = 2 * pi * raggio;  
11        return res;  
12    }  
13 }
```

Frame di *circonferenza()*
4 words da 32 bit

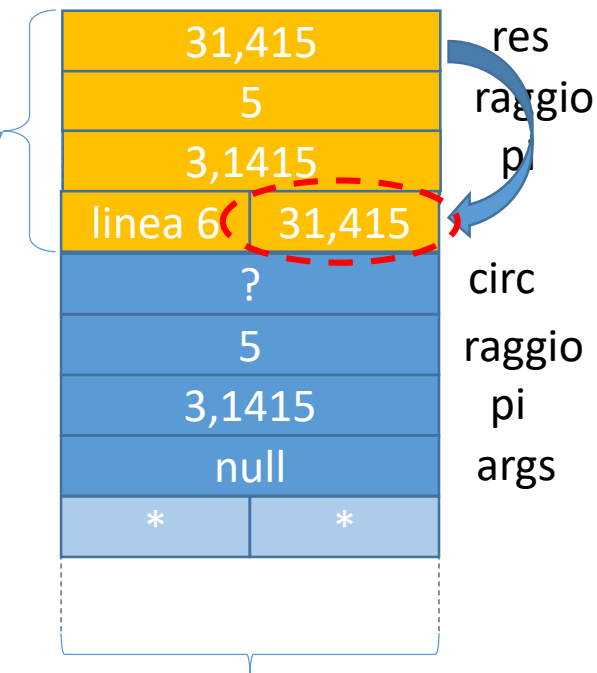


32 bit (*int* , *float*)

Il modello di memoria JVM (semplificato)

```
1 public class Circonferenza {  
2     public static void main (String []args) {  
3         float pi = (float)3.1415;  
4         float raggio = 5;  
5         float circ = circonferenza(pi, raggio);  
6         System.out.println("Raggio " + raggio + circ);  
7     }  
8  
9     public static float circonferenza(float pi, float raggio) {  
10        float res = 2 * pi * raggio;  
11        return res;  
12    }  
13 }
```

Frame di *circonferenza()*
4 words da 32 bit

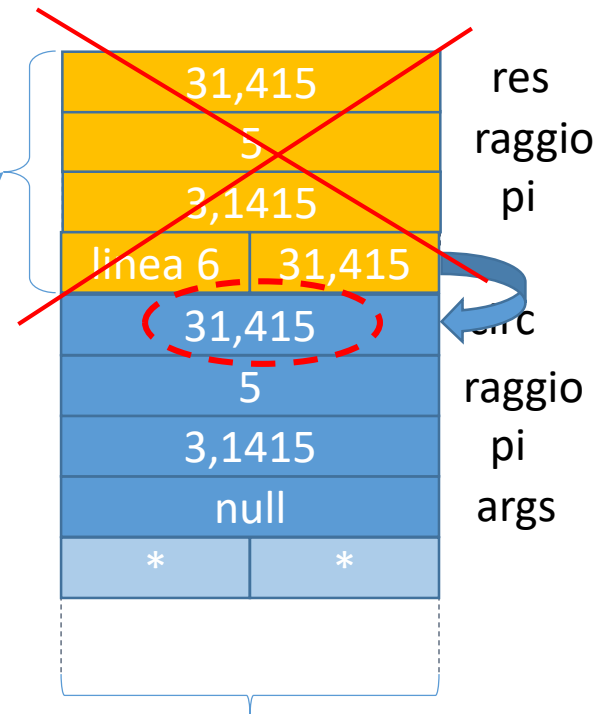


32 bit (int, float)

Il modello di memoria JVM (semplificato)

```
1 public class Circonferenza {  
2     public static void main (String []args) {  
3         float pi = (float)3.1415;  
4         float raggio = 5;  
5         float circ = circonferenza(pi, raggio);  
6         System.out.println("Raggio " + raggio + circ);  
7     }  
8  
9     public static float circonferenza(float pi, float raggio) {  
10        float res = 2 * pi * raggio;  
11        return res;  
12    }  
13 }
```

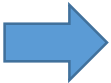
Frame di *circonferenza()*
eliminato



32 bit (*int* , *float*)

Il modello di memoria JVM (semplificato)

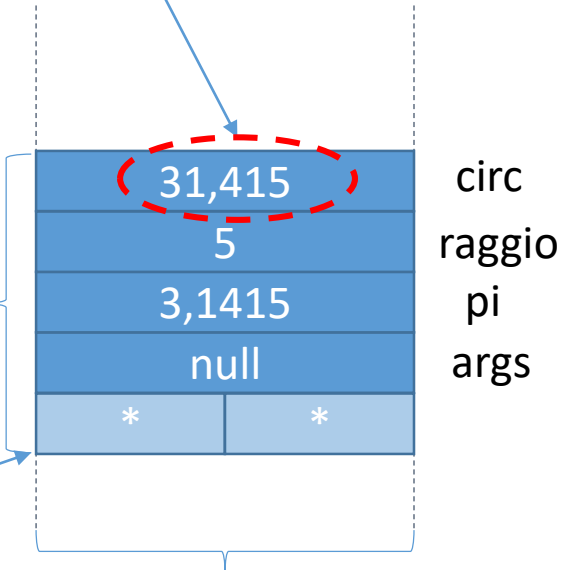
Risultato di *circonferenza()*
in cima al frame di *main()*



```
1 public class Circonferenza {
2     public static void main (String []args) {
3         float pi = (float)3.1415;
4         float raggio = 5;
5         float circ = circonferenza(pi, raggio);
6         System.out.println("Raggio " + raggio + circ);
7     }
8
9     public static float circonferenza(float pi, float raggio) {
10         float res = 2 * pi * raggio;
11         return res;
12     }
13 }
```

Frame di *main ()*

Base del frame 0x00001024



32 bit (int , float)

ESERCIZIO – Espressioni Booleane

- *Valutare su carta le espressioni booleane*
 - *Utilizzare EspressioniBooleane.java come verifica*

a) $3 > 5 \ || \ 10 == 7 + 3$

b) $3 != 5 \ \&\& \ (6 < 2 \ || \ 5 + 2 == 10 - 3)$

c) $3 < 5 < 7$

d) $3 < 5 \ \&\& \ 5 < 7$

e) $3 < 5 \ \&\& \ 7 < 5$

f) $false \ || \ 5 < 10$

```
1 public class EsprBooleane {  
2     public static void main(String[] args) {  
3         System.out.println("<espressione> is " + <espressione>);  
4     }  
5 }  
6
```


ESERCIZIO – Ordinamento con Scambio

- Utilizzare il tool JVisualizer per studiare l'evoluzione del frame stack nel caso dell'algoritmo di ordinamento con scambio visto nella scorsa esercitazione
 - Cosa succede al momento dell'invocazione di un metodo ?
 - Cosa succede al momento del ritorno da un metodo ?