

Programmazione I-B 2020-21

Laboratorio T2

Attilio Fiandrotti

attilio.fiandrotti@unito.it

12 Novembre 2020

Outline

- Soluzione esercizio combinazioni carte
- Esercizi sulla ricorsione
 - Stampa segmento numerico covariante
 - Stampa segmento numerico controvariante
 - Moltiplicazione e Java Visualizer
 - Esercizi per casa

Soluzione esercizio 5 Novembre

Combinazioni di carte

- Un giocatore riceve k carte da un mazzo di n carte. Si scriva un programma che richieda all'utente di inserire il numero di carte k ricevute e calcoli il numero di differenti combinazioni di k carte che può ricevere. Il programma verifichi che l'input inserito dall'utente sia corretto (es, $k < n$) e, se necessario, lo richieda nuovamente finché questo non è corretto.

Carte - Approccio

- Il numero di possibili combinazioni di k elementi da un set di n é dato dal coefficiente binomiale

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

- L'operatore fattoriale $n!$ é calcolabile come

$$n! = \prod_{k=1}^n k = 1 \cdot 2 \cdot 3 \cdots (n-1) \cdot n$$

Carte - Approccio

- Si implementino nella classe Aritmetica i nuovi metodi fattoriale() e binomiale()
- Si sviluppi l'opportuna classe di test che si occupa di interagire con l'utente tramite i metodi della classe SIn ed esegua il calcolo richiesto
- Si ponga attenzione a che tutti risultati intermedi siano correttamente rappresentabili in formato intero con segno a 32 bit (int)

Carte – fattoriale e binomiale

- Nuovi metodi classe Aritmetica

```
public static int fattoriale(int n) {  
    int k = 1;  
    int ret = 1;  
    while (k <= n) {  
        ret = Aritmetica.perU(ret, k);  
        k = k + 1;  
    }  
    return ret;  
}
```

Versione Iterativa



```
public static int binomiale(int n, int k) {  
    return quo(fattoriale(n), perU(fattoriale(k), fattoriale(n-k)));  
}
```

Carte – test case

```
public static void main(String[] args) {
    int n = 0, k = 0, nComb = 0;
    int nCarteMazzo = 40 // Carte nel mazzo

    boolean inputCorretto = false;
    while (inputCorretto == false) {
        inputCorretto = true;
        System.out.print("Inserire il numero n di carte nel mazzo: ");
        n = SIn.readInt(); // ipotesi: n inserito > 0
        if (n > nCarteMazzo) {
            System.out.println("ERRORE: n deve essere < " + nCarteMazzo);
            inputCorretto = false;
        }
    }

    // Parsing di k analogo (verificare che k < n)
    System.out.print("Inserire il numero k di carte estratte: ");
    k = SIn.readInt(); // ipotesi: k inserito > 0

    nComb = Aritmetica.binomiale(n, k);
    System.out.println("Estraendo " + k + " carte da un mazzo di "
        + n + " ci sono " + nComb + " combinazioni");
}
```


Carte – test case

```
> java CarteTest
Inserire il numero n di carte nel mazzo: 4
Inserire il numero k di carte estratte: 2
Estraendo 2 carte da un mazzo di 4 ci sono 6 combinazioni
```

```
> java CarteTest
Inserire il numero n di carte nel mazzo: 12
Inserire il numero k di carte estratte: 1
Estraendo 1 carte da un mazzo di 12 ci sono 12
combinazioni
```

```
> java CarteTest
Inserire il numero n di carte nel mazzo: 13
Inserire il numero k di carte estratte: 1
Estraendo 1 carte da un mazzo di 13 ci sono 4 combinazioni
```

Carte – test case fattoriale

```
public class FattorialeTest {  
    //Valori attesi dalla funzione fattoriale  
    //https://it.wikipedia.org/wiki/Fattoriale  
    // n          n!  
    // ...  
    //12  479001600  
    //13  6227020800  
  
    public static void main(String[] args) {  
        System.out.print("Inserire argomento n di fattoriale: ");  
        int n = SIn.readInt(); // ipotesi: n inserito > 0  
  
        long fatt = Aritmetica.fattoriale(n);  
        System.out.println("fattoriale(" + n + ") = " + fatt);  
    }  
}
```

Carte – test case fattoriale

```
> java FattorialeTest  
Inserire argomento n di fattoriale: 12  
fattoriale(12) = 479001600  
  
> java FattorialeTest  
Inserire argomento n di fattoriale: 13  
fattoriale(13) = 1932053504
```

Valore atteso 6 227 020 800

6 227 020 800 –

1 932 053 504 =

4 294 967 296

Errore pari a 2^{32}

Carte – test case fattoriale

- Il massimo numero rappresentabile su *int* é 2 147 483 647
 - *Overflow* del risultato di 2^{32} bit
 - Il risultato di $13!$ non é rappresentabile su 32 bit, nemmeno se *unsigned*
- Occorrerà reappresentare $13!$ su *long*

Table 5.1 · Some basic types

Value type	Range
<i>Int</i>	32-bit signed two's complement integer (-2^{31} to $2^{31} - 1$, inclusive)
<i>Long</i>	64-bit signed two's complement integer (-2^{63} to $2^{63} - 1$, inclusive)

Carte – test case fattoriale con *long*

```
public class FattorialeLongTest {  
    public static void main(String[] args) {  
  
        int n = 13;  
        long outputAtteso = 6227020800L;  
        long fatt = fattorialeLong(n);  
        System.out.println("fattorialeLong(" + n + ") = " + fatt +  
                           " => " + (fatt == outputAtteso));  
    }  
  
    public static long fattorialeLong(int n) {  
        int k = 1;  
        long ret = 1;  
        while (k <= n) {  
            ret = k * ret;  
            k = k + 1;  
        }  
        return ret;  
    }  
}
```

Esercizi sulla ricorsione

La ricorsione


- Approccio *divide et impera*
 - Individuare un caso che si sa risolvere (*caso base*)
 - Ricondursi al caso base per gli altri casi
- Caveat
 - Out of memory (*stack overflow*)
 - Complessità chiamata a funzione

La ricorsione - fattoriale

$$n! = \prod_{k=1}^n k = 1 \cdot 2 \cdot 3 \cdots (n-1) \cdot n$$

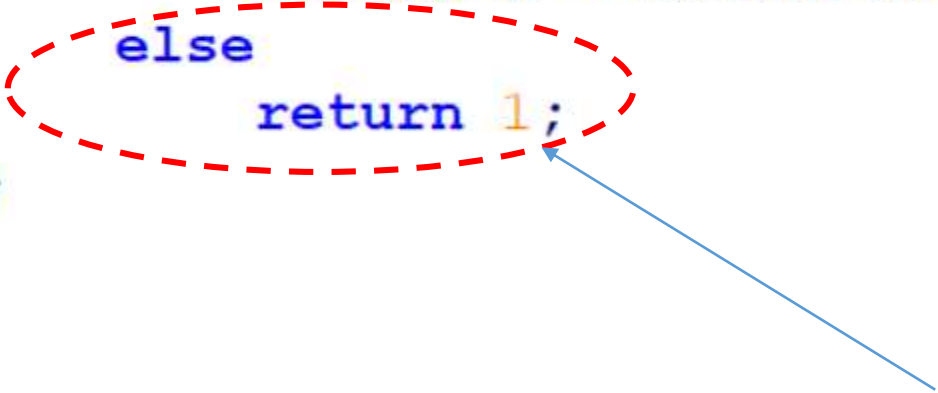
$$n! = n \times (n-1)! .$$

Caso base


$$x! = \begin{cases} 1 & \text{se } x = 0 \\ x \times (x-1)! & \text{se } x > 0 \end{cases} .$$

La ricorsione - fattoriale

```
public static int fattorialeRic(int n) {  
    if (n >= 1)  
        return n * fattorialeRic(n - 1);  
    else  
        return 1;  
}
```



Caso base ($n==0$)

Esercizio 1 – ricorsione covariante $[0, \dots, n)$

- Stampare i numeri naturali nel segmento $[0, \dots, n)$ in ordine crescente e decrescente, con metodi ricorsivi co-varianti.
- Metodo:
 - individuare un caso base e ricondursi ad esso

Esercizio 1 – ricorsione crescente [0, ..., n)

```
/* Metodo covariante che stampa i primi
n numeri naturali in ordine crescente,
cioè da 0 incluso a n escluso. */
public static void stampaU(int n){
    if (n == 0) {
        /* Stampa i numeri da 0 incluso a 0 escluso,
        cioè nessun numero */
    } else {
        stampaU(n-1); /* Stampa i numeri da 0 incluso a n-1 escluso. */
        System.out.print(n-1 + " "); /* Stampa n-1 */
        /* Avrà appena stampato i numeri da 0 incluso
        a n-1 incluso, quindi da 0 a n escluso */
    }
}
```

Caso base n==0

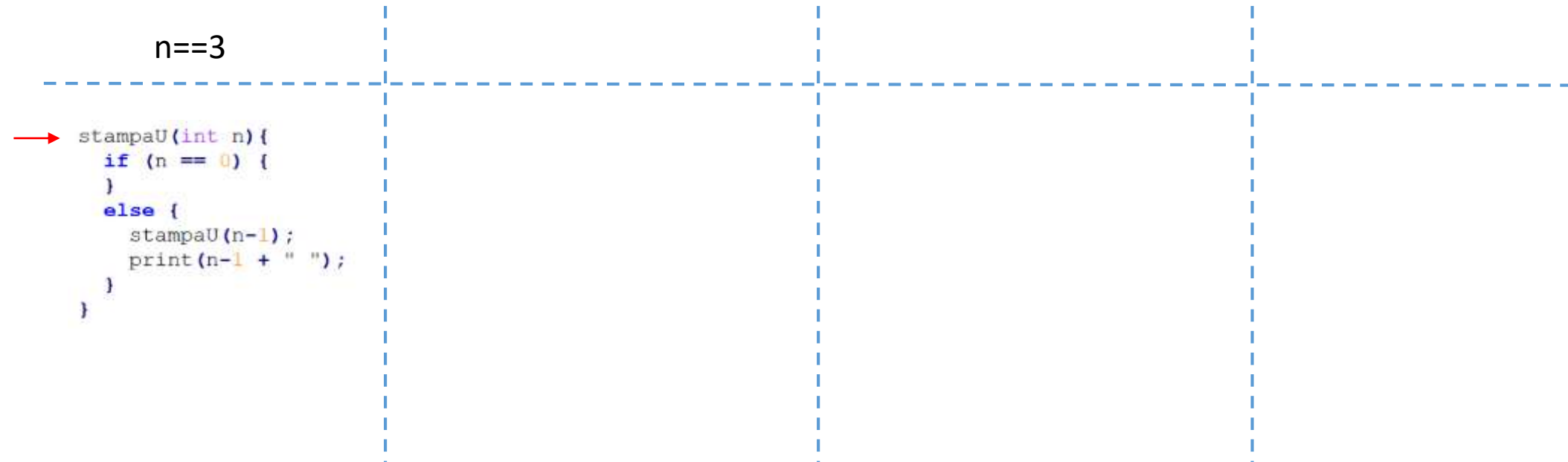
Attenzione
all'ordine!



Esercizio 1 – ricorsione crescente - insight

```
>java StampaSegmCovFinaleTest
```

- Esempio per $n=3$

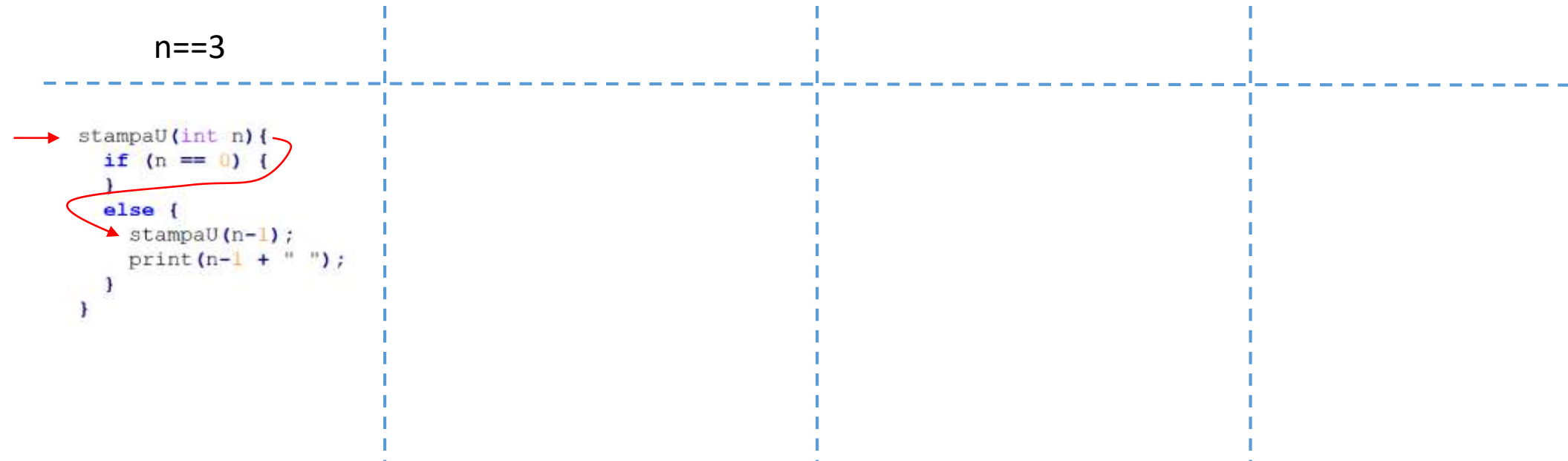


Chiamata a `stampaU(3)`

Esercizio 1 – ricorsione crescente - insight

```
>java StampaSegmCovFinaleTest
```

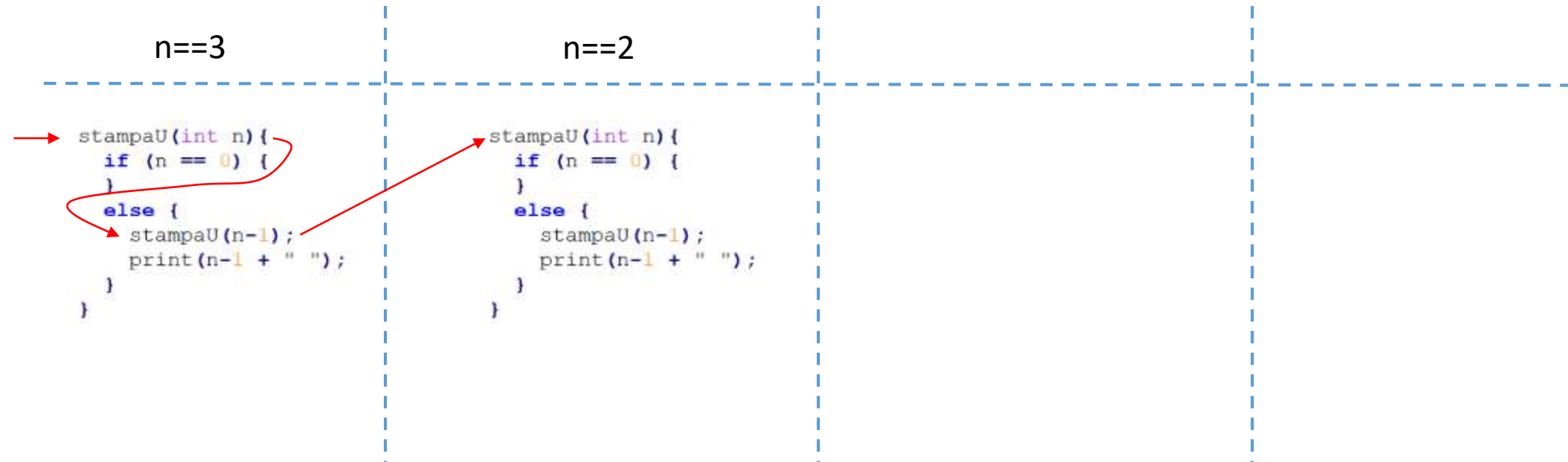
- Esempio per $n=3$



Esercizio 1 – ricorsione crescente - insight

```
>java StampaSegmCovFinaleTest
```

- Esempio per $n=3$

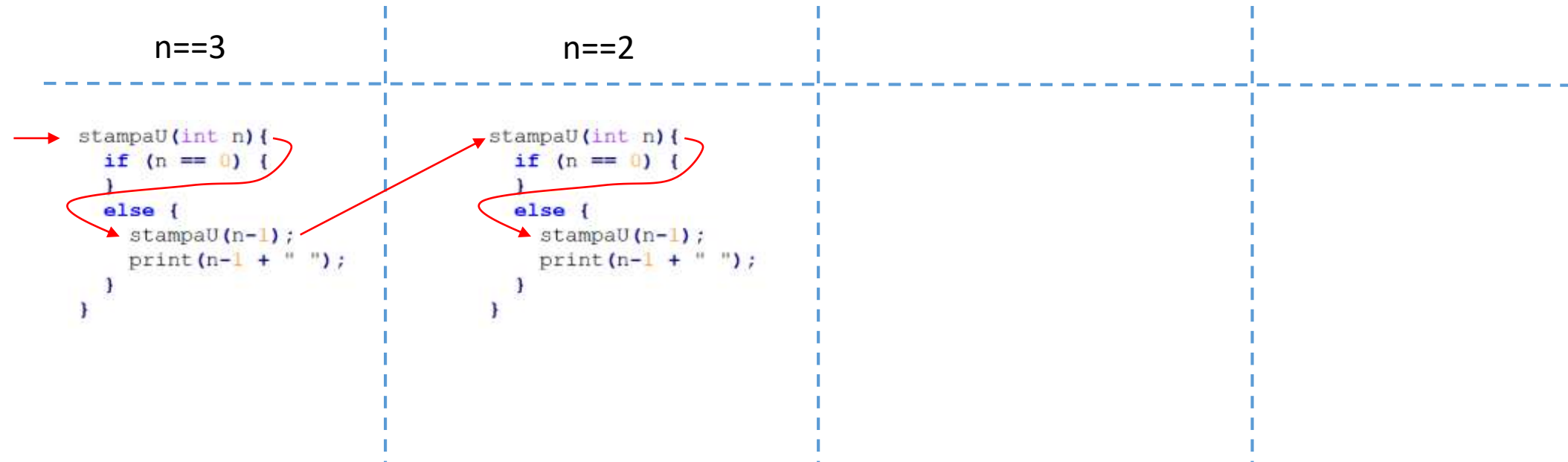


Chiamata a `stampaU(2)`

Esercizio 1 – ricorsione crescente - insight

```
>java StampaSegmCovFinaleTest
```

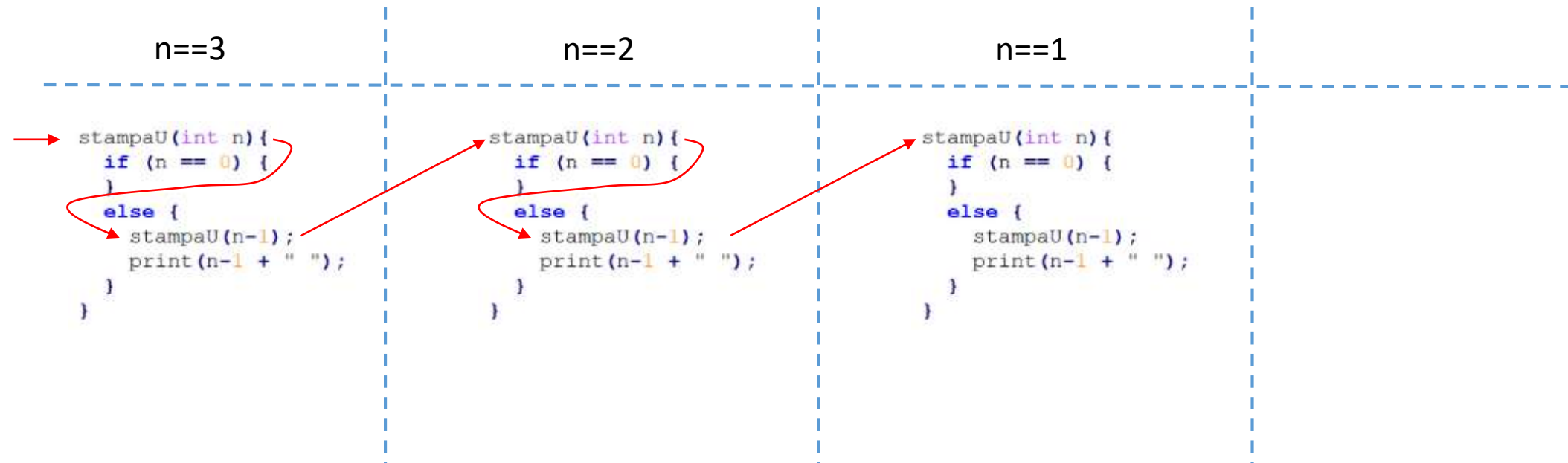
- Esempio per $n=3$



Esercizio 1 – ricorsione crescente - insight

```
>java StampaSegmCovFinaleTest
```

- Esempio per $n==3$

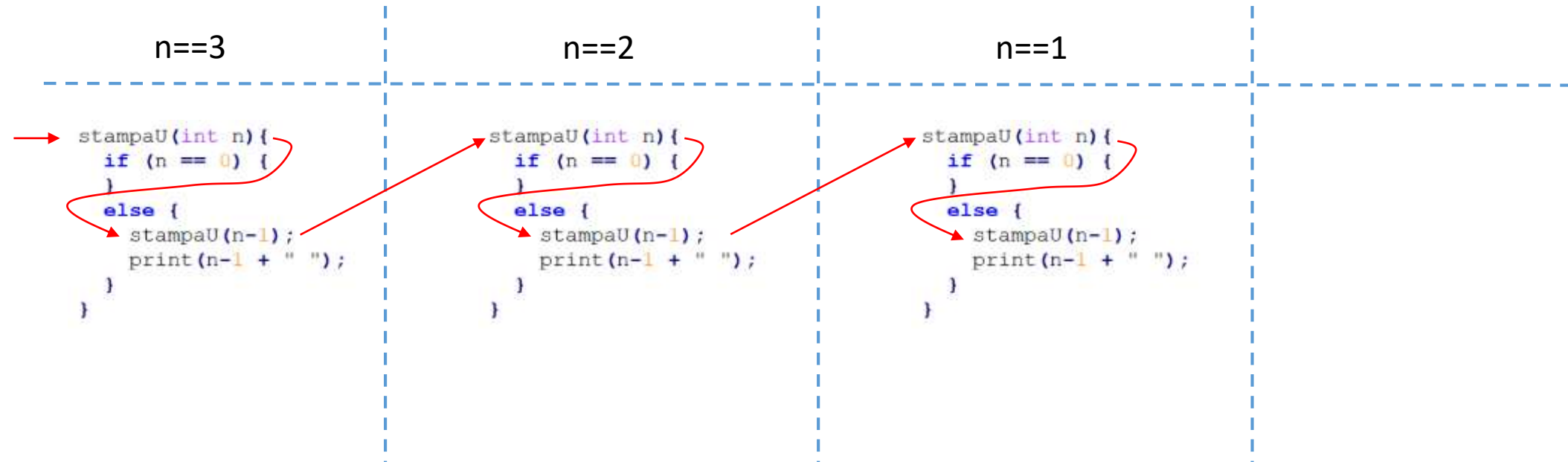


Chiamata a `stampaU(1)`

Esercizio 1 – ricorsione crescente - insight

```
>java StampaSegmCovFinaleTest
```

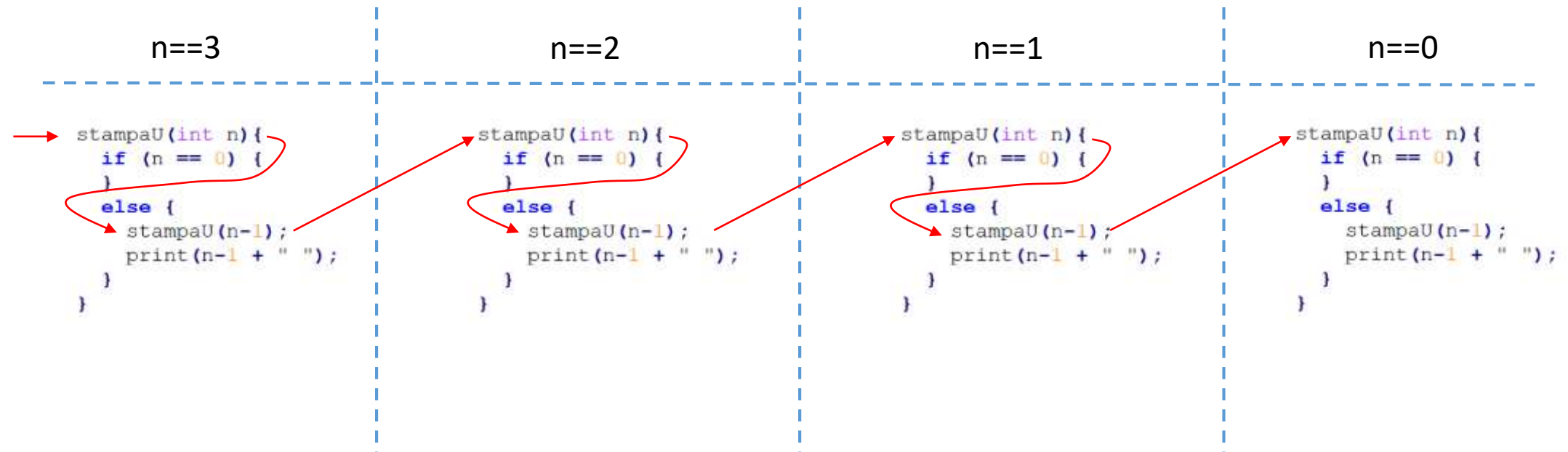
- Esempio per $n=3$



Esercizio 1 – ricorsione crescente - insight

```
>java StampaSegmCovFinaleTest
```

- Esempio per $n==3$

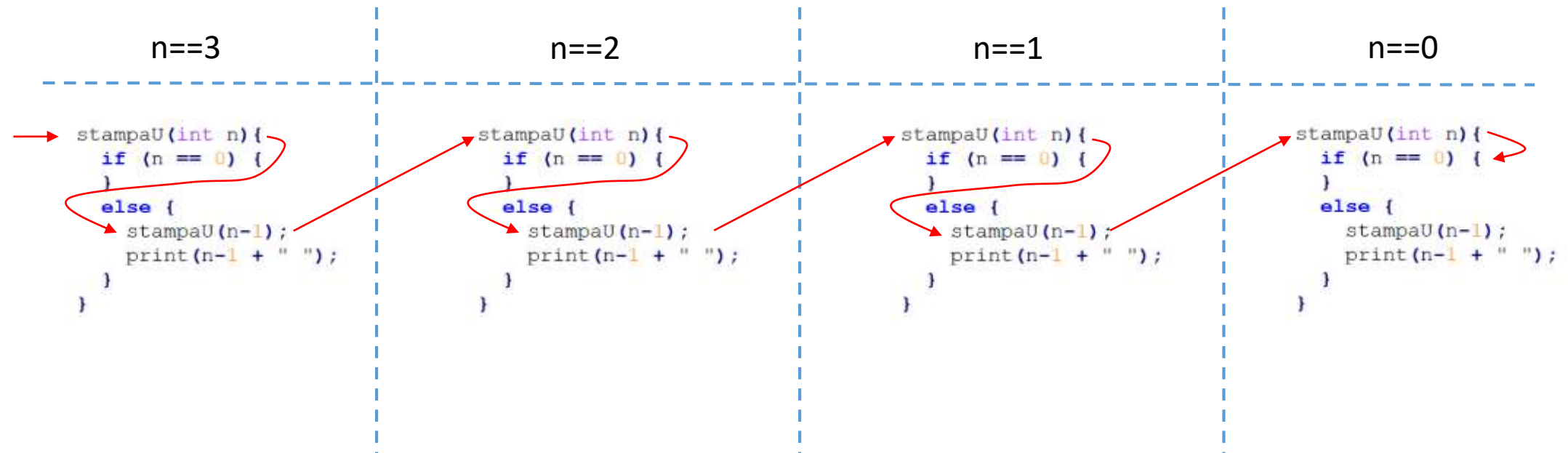


Chiamata a `stampaU(0)`

Esercizio 1 – ricorsione crescente - insight

```
>java StampaSegmCovFinaleTest
```

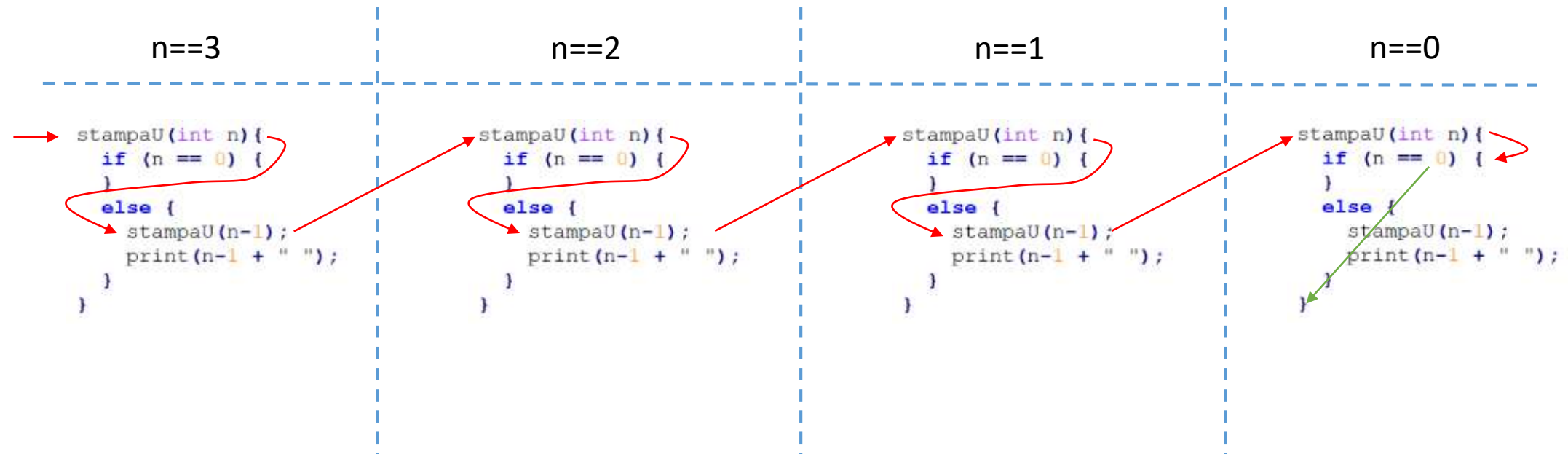
- Esempio per $n==3$



Esercizio 1 – ricorsione crescente - insight

```
>java StampaSegmCovFinaleTest
```

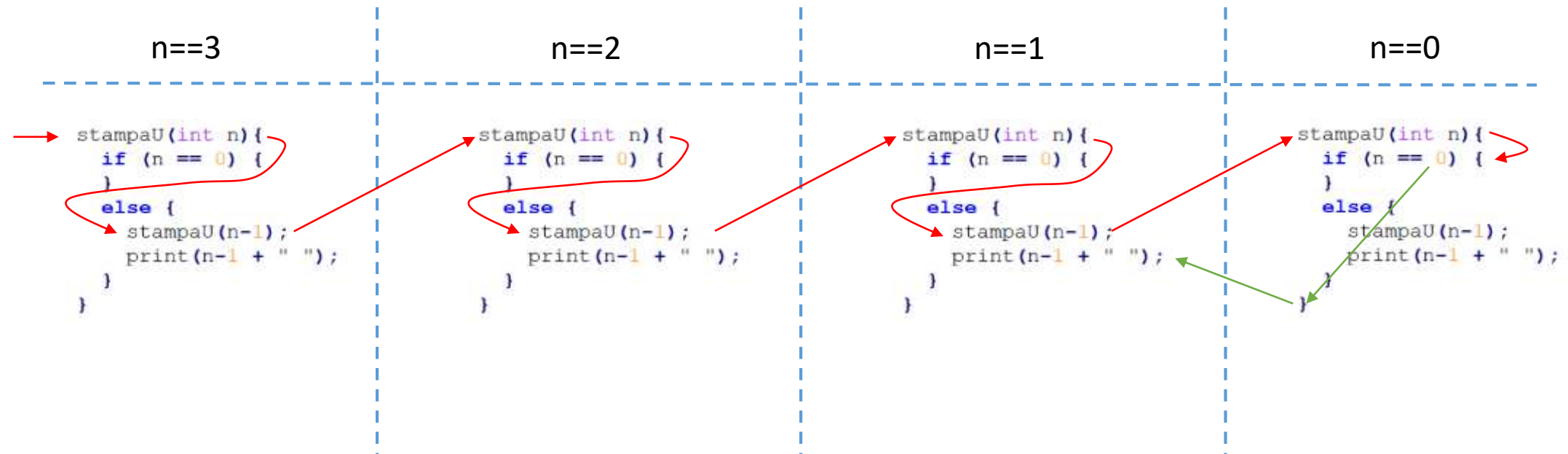
- Esempio per $n==3$



Esercizio 1 – ricorsione crescente - insight

```
>java StampaSegmCovFinaleTest
```

- Esempio per $n==3$

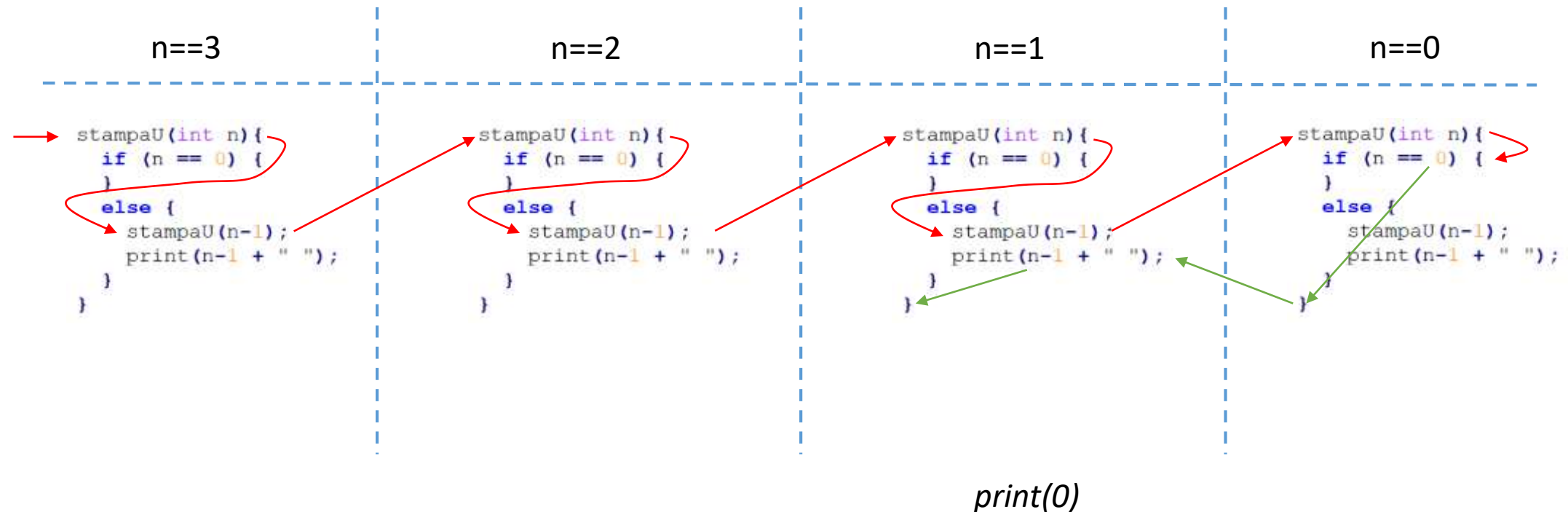


Ritorno a *stampaU(1)*

Esercizio 1 – ricorsione crescente - insight

- Esempio per $n==3$

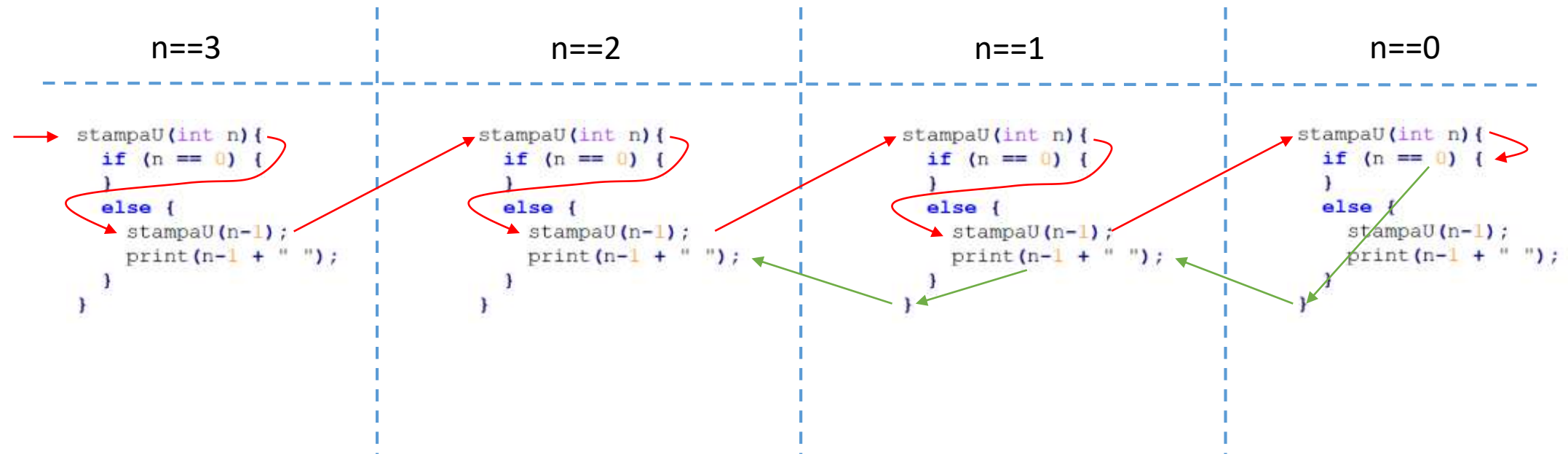
```
>java StampaSegmCovFinaleTest  
0
```



Esercizio 1 – ricorsione crescente - insight

- Esempio per $n==3$

```
>java StampaSegmCovFinaleTest  
0
```

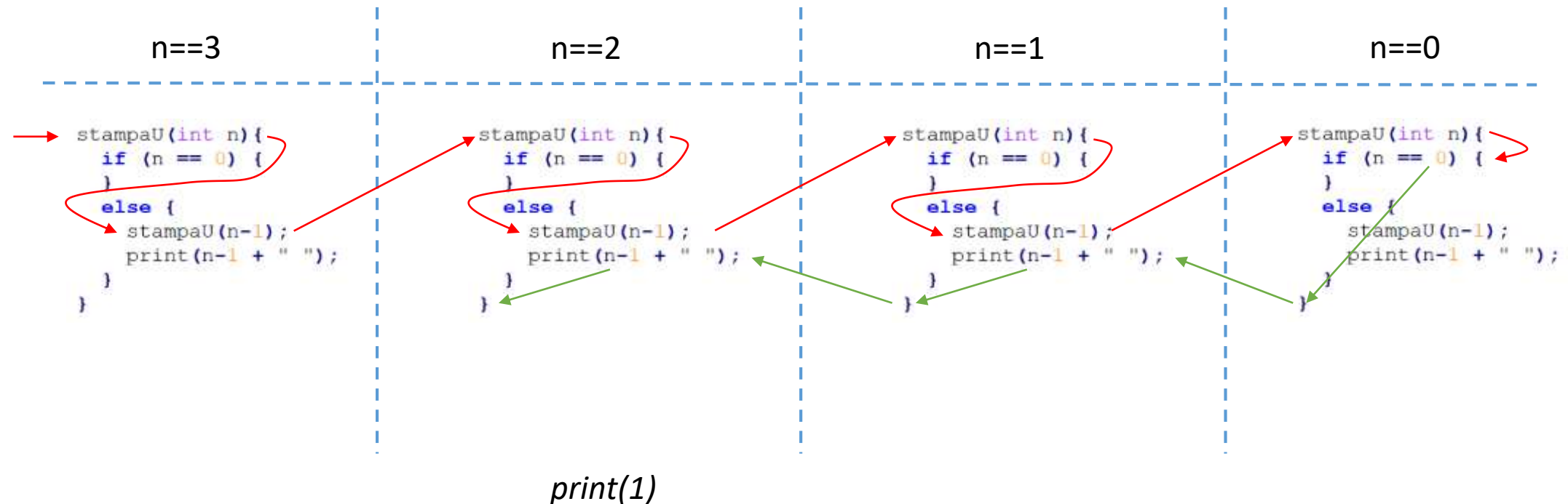


Ritorno a `stampaU(2)`

Esercizio 1 – ricorsione crescente - insight

- Esempio per $n==3$

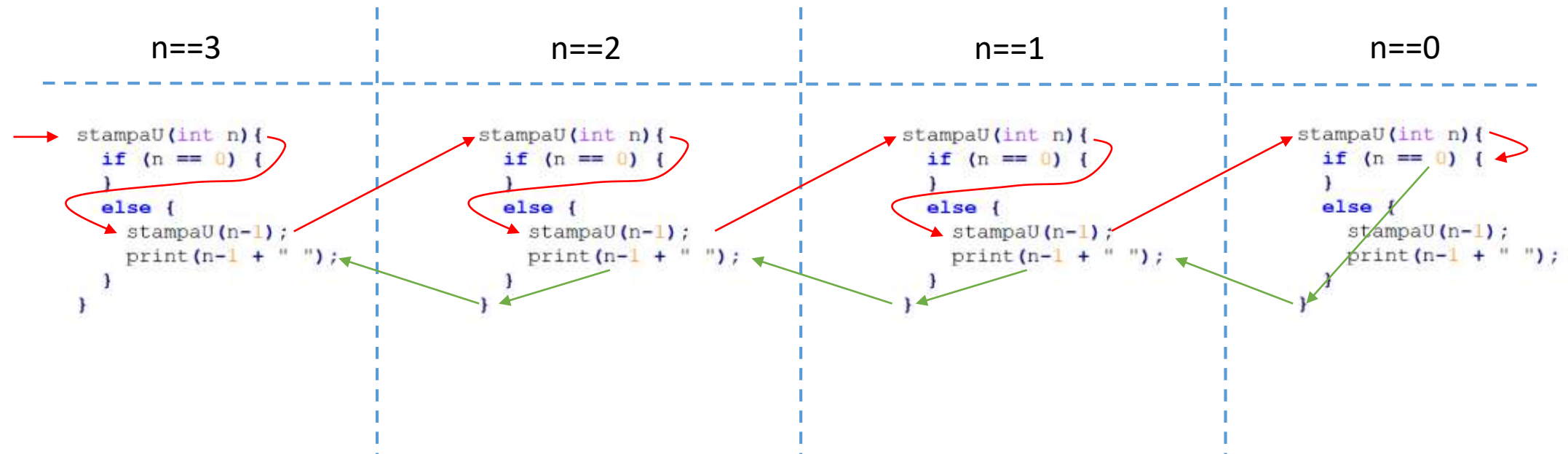
```
>java StampaSegmCovFinaleTest  
0 1
```



Esercizio 1 – ricorsione crescente - insight

- Esempio per $n==3$

```
>java StampaSegmCovFinaleTest  
0 1
```

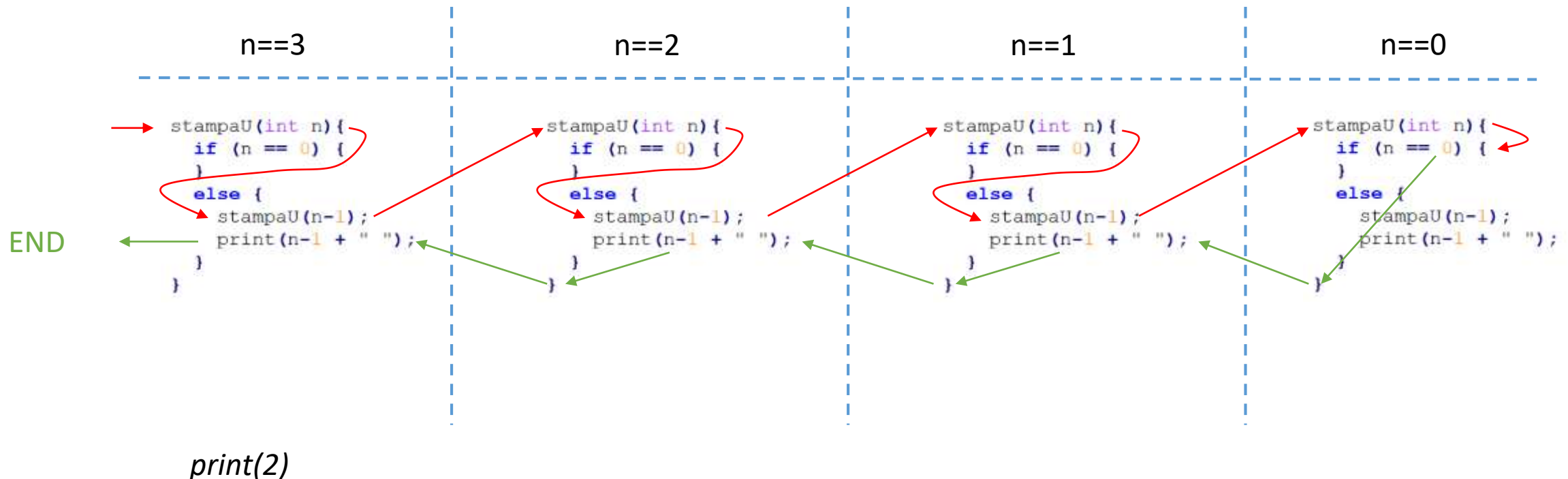


Ritorno a `stampaU(3)`

Esercizio 1 – ricorsione crescente - insight

- Esempio per $n==3$

```
>java StampaSegmCovFinaleTest  
0 1 2
```



Esercizio 1 – ricorsione decrescente

```
/* Metodo covariante che stampa i primi
n numeri naturali in ordine decrescente,
cioè da n escluso a 0 incluso. */
public static void stampaD(int n){
    if (n == 0) {
        /* Stampa i numeri da 0 escluso a 0 incluso,
        cioè nessun numero */
    } else {
        System.out.print(n-1 + " "); /* Stampa n-1 */
        stampaD(n-1); /* Stampa i numeri da n-1 escluso a 0 incluso. */
        /* Avrà appena stampato i numeri da n-1 incluso,
        a 0 incluso, quindi da n escluso a 0 incluso */
    }
}
```

Caso base n==0

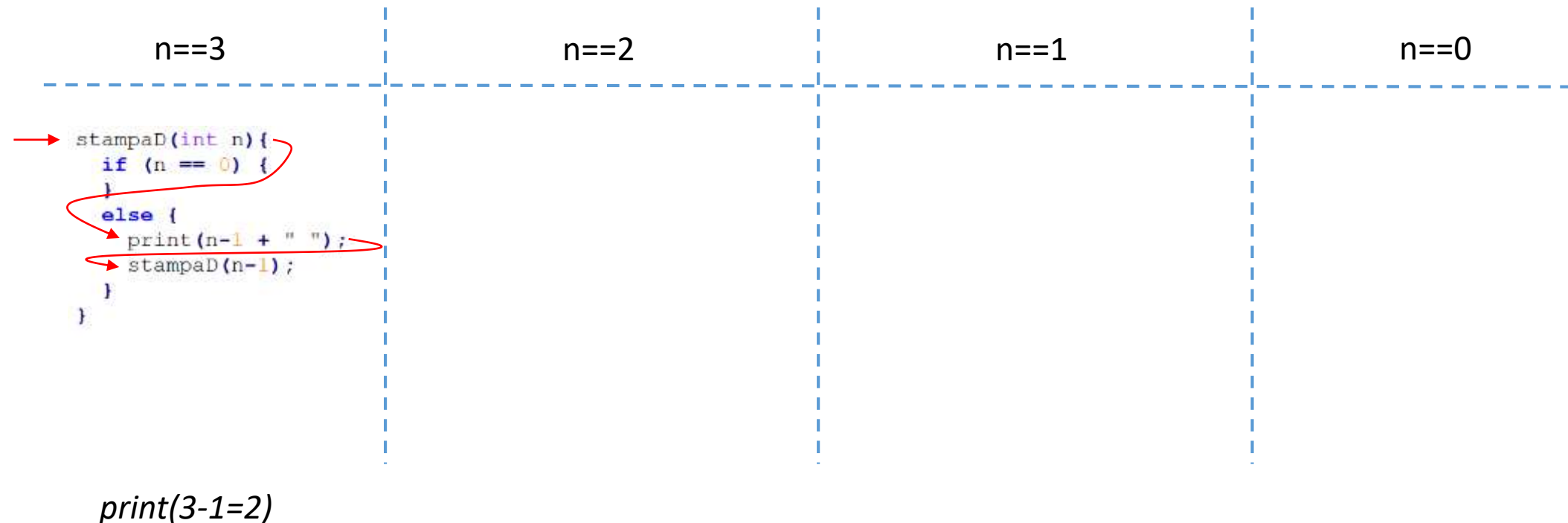
Attenzione
all'ordine!



Esercizio 1 – ricorsione decrescente - insight

- Esempio per $n=3$

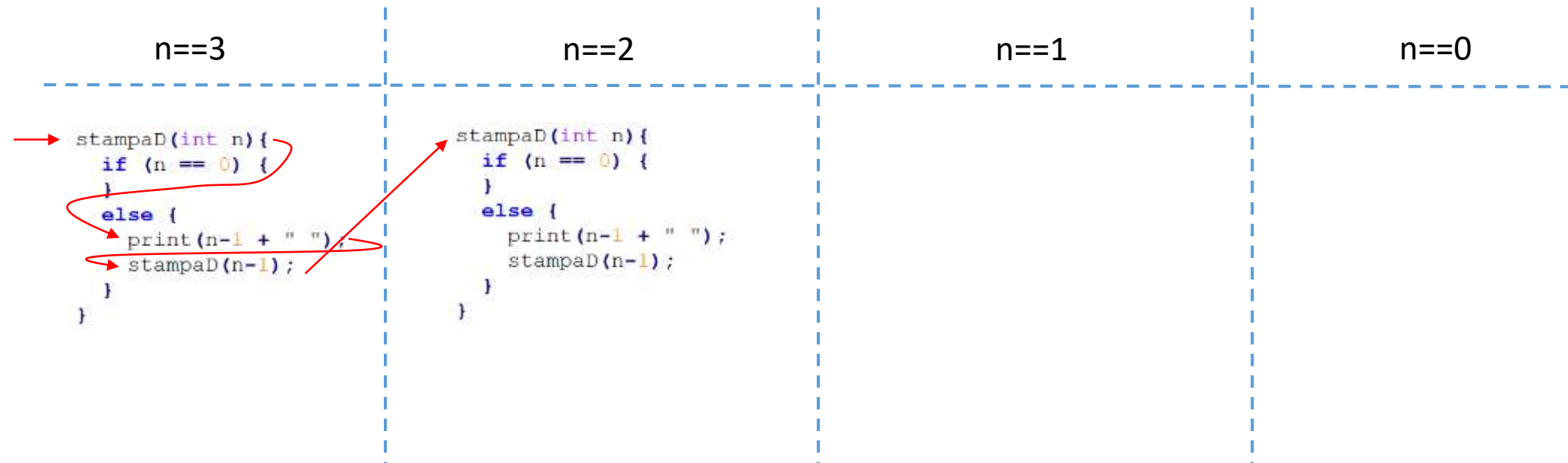
```
>java StampaSegmCovFinaleTest  
2
```



Esercizio 1 – ricorsione decrescente - insight

- Esempio per $n=3$

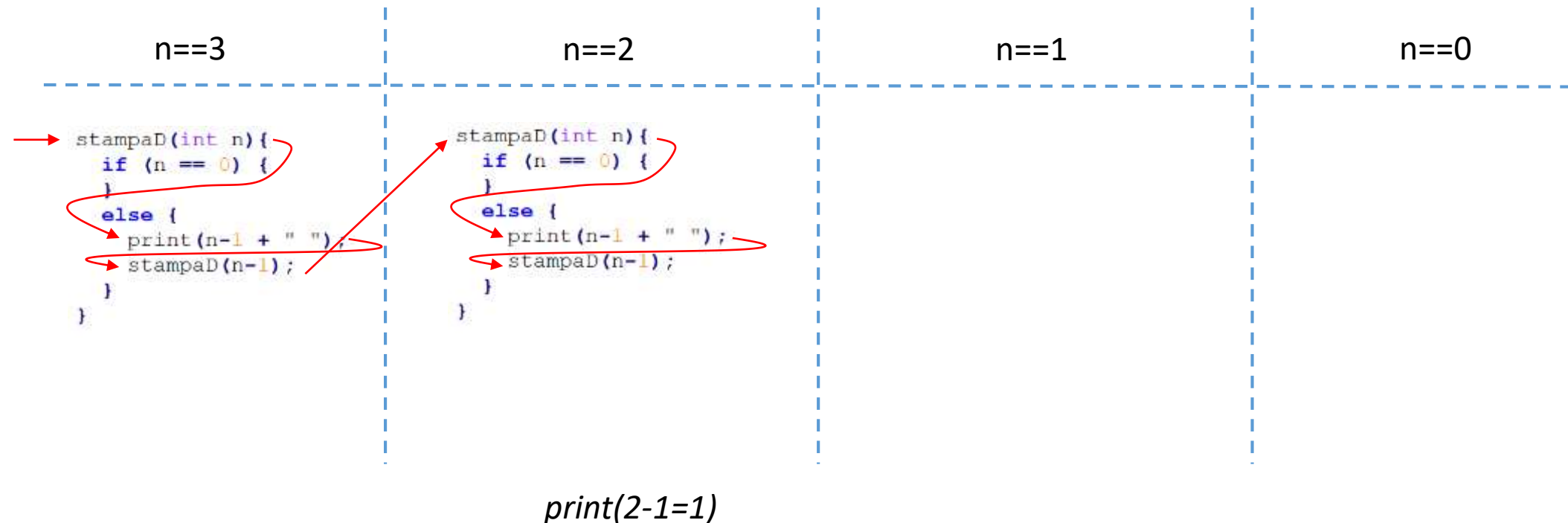
```
>java StampaSegmCovFinaleTest  
2
```



Esercizio 1 – ricorsione decrescente - insight

- Esempio per $n=3$

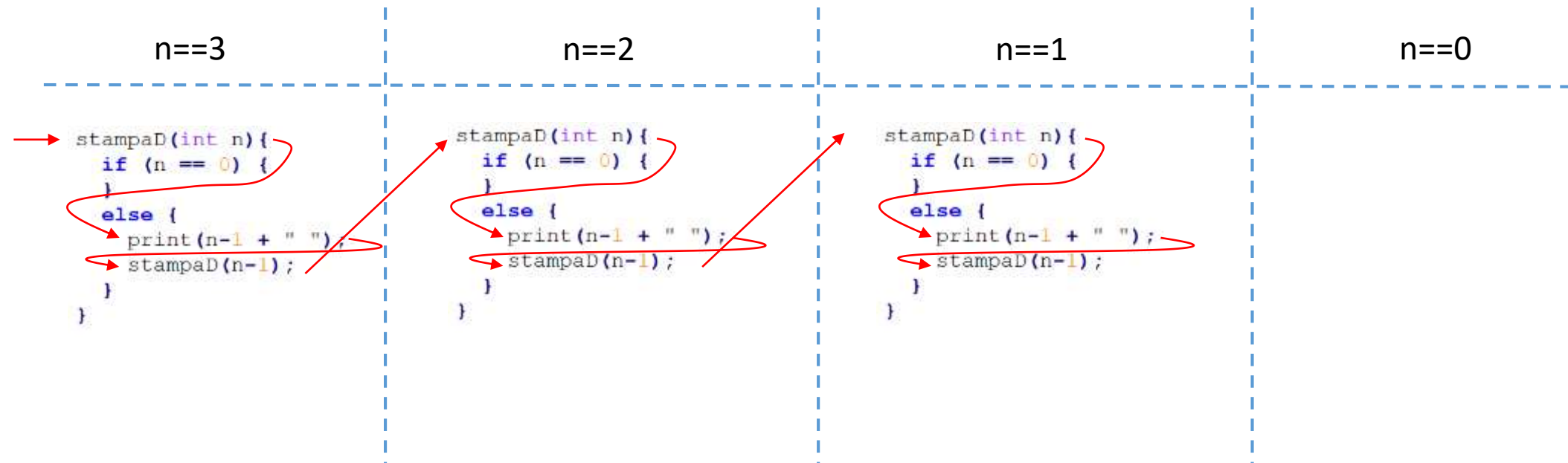
```
>java StampaSegmCovFinaleTest  
2 1
```



Esercizio 1 – ricorsione decrescente - insight

- Esempio per $n=3$

```
>java StampaSegmCovFinaleTest  
2 1 0
```

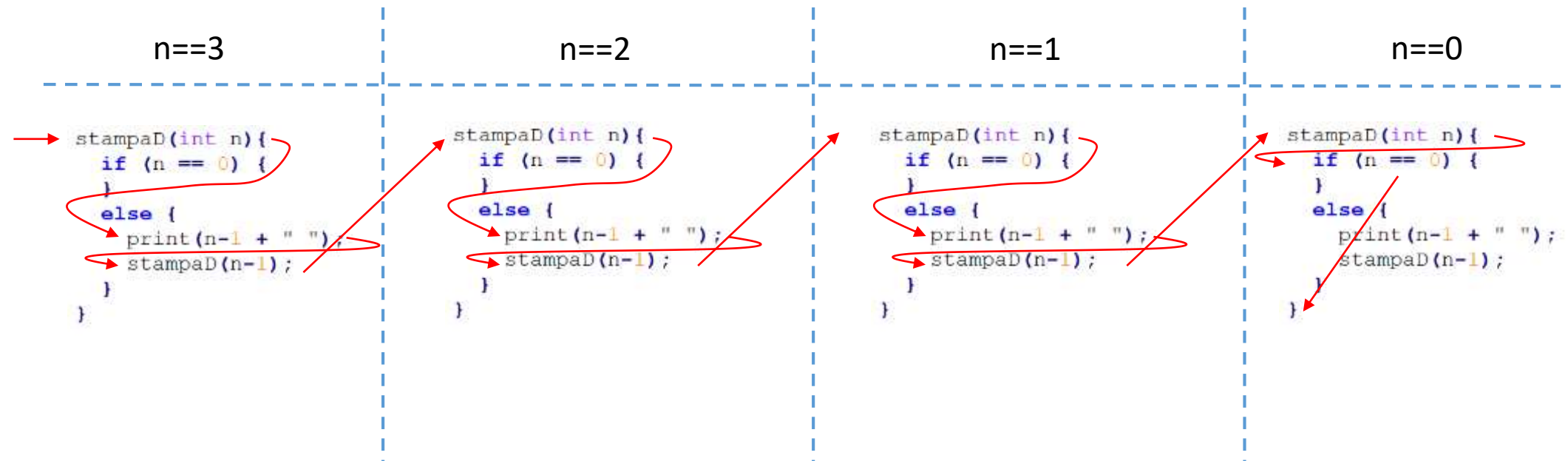


print(1-1=0)

Esercizio 1 – ricorsione decrescente - insight

- Esempio per $n=3$

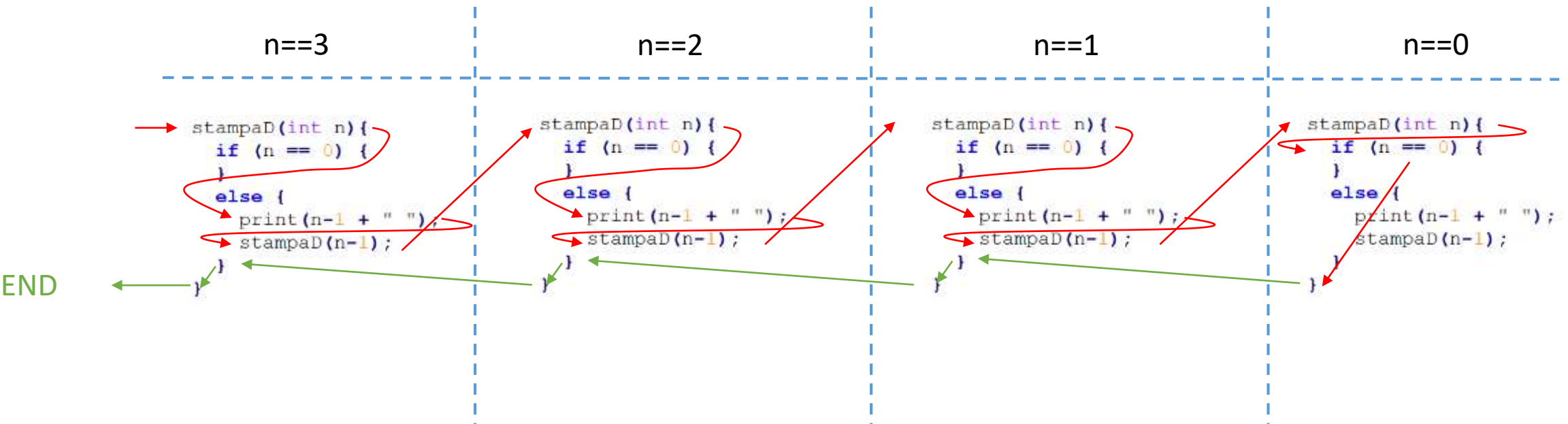
```
>java StampaSegmCovFinaleTest  
2 1 0
```



Esercizio 1 – ricorsione decrescente - insight

- Esempio per $n=3$

```
>java StampaSegmCovFinaleTest  
2 1 0
```



Esercizio 1 – test case

```
public static void main(String[] args){
    System.out.println("--- Test stampaU");
    StampaSegmCovFinale.stampaU(0);System.out.println();
    StampaSegmCovFinale.stampaU(1);System.out.println();
    StampaSegmCovFinale.stampaU(2);System.out.println();
    StampaSegmCovFinale.stampaU(3);System.out.println();
    StampaSegmCovFinale.stampaU(4);System.out.println();

    System.out.println("--- Test stampaD");
    StampaSegmCovFinale.stampaD(0);System.out.println();
    StampaSegmCovFinale.stampaD(1);System.out.println();
    StampaSegmCovFinale.stampaD(2);System.out.println();
    StampaSegmCovFinale.stampaD(3);System.out.println();
    StampaSegmCovFinale.stampaD(4);System.out.println();
}
```

Esercizio 2 – ricorsione controvariante

- Stampa i numeri naturali nel segmento $[0, \dots, n)$ in ordine crescente e decrescente, con metodi ricorsivi contro-varianti.

Esercizio 2 – ricorsione crescente $[0, \dots, n)$

```
/* Metodo controvariante che stampa i primi
n numeri naturali in ordine crescente. */
public static void stampaU(int n, int i){
    if (n == i) {
        /* Stampa i numeri nell'intervallo [0,n-n),
        cioè in [0,0), che è vuoto. Quindi non
        stampa nulla. */
    } else {
        stampaU(n, i+1); /* Per ipotesi induttiva,
                           stampa [0,n-(i+1)), cioè [0,n-i-1) */
        System.out.print(n-(i+1) + " "); /* Stampa n-(i+1)==n-i-1*/
        /* Ha stampato [0,n-i-1) seguito dalla stampa di n-i-1.
        Quindi stampa l'intervallo [0,n-i) dei numeri naturali*/
    }
}

/* Metodo wrapper che stampa i primi
n numeri naturali in ordine crescente
sfruttando quello controvariante. */
public static void stampaU(int n){
    stampaU(n, 0); /* Stampa l'intervallo [0,n) dei numeri naturali*/
}
```

Caso base $n==i$ (cfr $n==0$ covariante)

Inizializzazione $n==0$

Esercizio 2 – ricorsione decrescente [0, ..., n)

```
/* Metodo controvariante che stampa i primi
n numeri naturali in ordine decrescente. */
public static void stampaD(int n, int i){
    if (n == i) {
        /* Stampa i numeri nell'intervallo (n-n,0],
        cioè in (0,0], che è vuoto. Quindi non
        stampa nulla. */
    } else {
        System.out.print(n-(i+1) + " "); /* Stampa n-(i+1)==n-i-1*/
        stampaD(n, i+1); /* Per ipotesi induttiva,
                           stampa (n-(i+1),0], cioè (n-i-1,0] */
        /* Ha stampato (n-i-1,0] preceduto dalla stampa di n-i-1.
        Quindi stampa l'intervallo (n-i,0] dei numeri naturali */
    }
}

/* Metodo wrapper che stampa i primi
n numeri naturali in ordine crescente
sfruttando quello controvariante. */
public static void stampaD(int n){
    stampaD(n, 0); /* Stampa l'intervallo (n,0] dei numeri naturali*/
}
```

Attenzione
all'ordine!



Caso base $n==i$ (cfr $n==0$ covariante)

Inizializzazione $n==0$

Esercizio 3

- Siano m ed n due numeri naturali. Definire (e simulare almeno con Java Visualizer) due metodi ricorsivi, uno co-variante ed uno contro-variante, che stampino, in ordine inverso, i primi n multipli non nulli di m . Ad esempio, se $m = 4$ e $n = 3$, allora verranno stampati i valori [4, 8, 12].

Ovvero, per $n > 0$

Esercizio 3 – covariante crescente

Escludo $n == 0$

```
public static void multipliU(int m, int n) {  
    if (n == 0) {  
        /* Non stampa multipli nulli di m */  
    } else {  
        multipliU(m, n - 1); /* Stampa m 2m .. (n-1)m */  
        System.out.println(n*m + " "); /* Stampa nm */  
        /* Stampa m 2m .. (n-1)m nm */  
    }  
}
```

Esercizio 3 – covariante decrescente

Escludo n == 0

```
public static void multipliD(int m, int n) {  
    if (n == 0) {  
        /* Non stampa multipli nulli di m */  
    } else {  
        System.out.println(n*m + " "); /* Stampa nm */  
        multipliU(m, n - 1); /* Stampa (n-1)m .. 2m m */  
        /* Stampa nm (n-1)m .. 2m m */  
    }  
}
```

```
public static void main(String[] args) {  
    multipliU(4, 3);  
    System.out.println();  
    multipliD(4, 3);  
}
```


Esercizio 3 – controvariante crescente

```
public static void multipliU(int m, int n, int i) {
    if (n == i) {
        /* Non stampa multipli nulli di m */
    } else {
        multipliU(m, n, i + 1); /* Stampa m 2m .. (n-(i+1))m
                                Cioè stampa m 2m .. (n-i-1)m */
        System.out.println((n-i)*m + " "); /* Stampa (n-i)m */
        /* Stampa m 2m .. (n-i-1)m (n-i)m */
    }
}

public static void multipliU(int m, int n) {
    multipliU(m, n, 0);
}
```

Esercizio 3 – controvariante decrescente

```
public static void multipliD(int m, int n, int i) {  
    if (n == i) {  
        /* Non stampa multipli nulli di m */  
    } else {  
        System.out.println((n-i)*m + " "); /* Stampa (n-i)m */  
        multipliD(m, n, i + 1); /* Stampa (n-(i+1))m .. 2m m */  
        /* Stampa (n-i)m (n-(i+1))m .. 2m m */  
    }  
}  
  
public static void multipliD(int m, int n) {  
    multipliD(m, n, 0);  
}
```

Esercizio 4 – per casa

- Realizzare una classe *AritmeticaRic.java*, e relativa *AritmeticaRicTest.java*, con metodi ricorsivi co-varianti e contro-varianti che calcolano le operazioni:
 - somma
 - differenza
 - moltiplicazione
 - potenza
 - quoziente
 - resto