

# Programmazione I-B 2020-21

Laboratorio T2

Attilio Fiandrotti

[attilio.fiandrotti@unito.it](mailto:attilio.fiandrotti@unito.it)

21 Ottobre 2020

# Outline

- Le variabili statiche
- Testing di una classe
- Esercizio: libreria di funzioni aritmetiche
- Esercizio: libreria di funzioni booleane
- Esercizio: libreria di funzioni «grafiche»

# Le variabili statiche

# Le variabili statiche

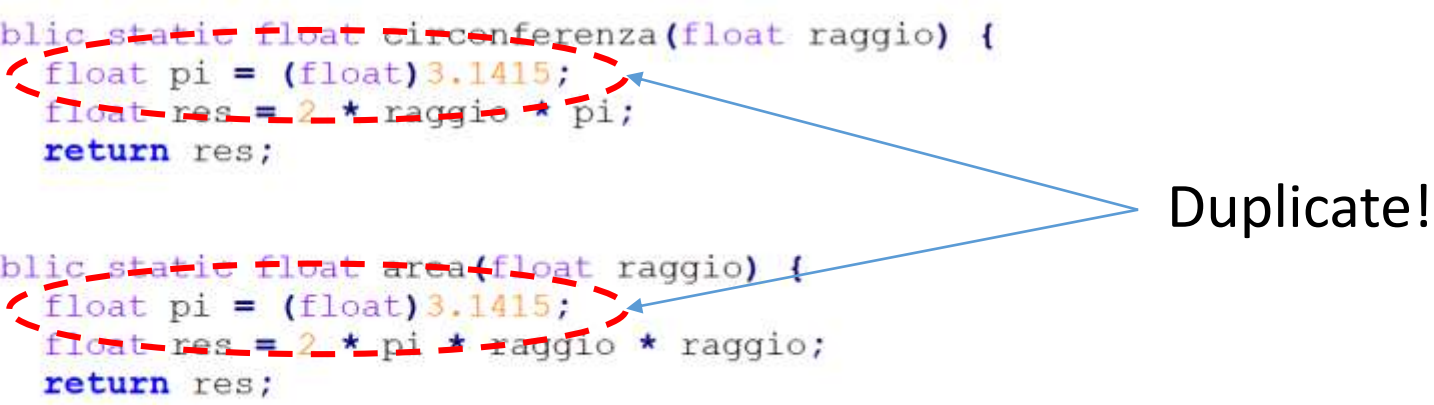
- Scrivere un programma Java che dato il raggio di un cerchio come numero reale ne calcoli la circonferenza

```
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 }
```

# Le variabili statiche

- Scrivere un programma Java che dato il raggio di un cerchio come numero reale ne calcoli circonferenza ed area

```
1 public class CirconferenzaArea {
2     public static void main (String []args) {
3         float raggio = 5;
4         float circ = circonferenza(raggio);
5         float area = area(raggio);
6         System.out.println("Raggio " + raggio + " -> circonferenza " + circ + " area " + area);
7     }
8
9     public static float circonferenza(float raggio) {
10         float pi = (float)3.1415;
11         float res = 2 * raggio * pi;
12         return res;
13     }
14
15     public static float area(float raggio) {
16         float pi = (float)3.1415;
17         float res = 2 * pi * raggio * raggio;
18         return res;
19     }
20 }
```



Duplicate!

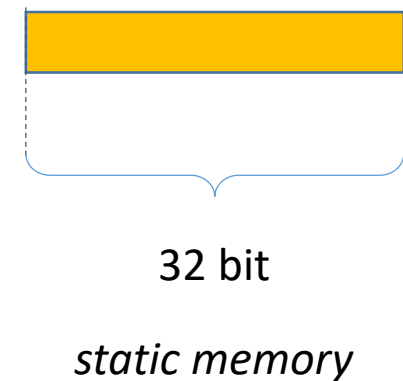
# Le variabili statiche

Definita nella  
classe, non nel  
metodo!

```
1 public class CirconferenzaAreaStatPi {  
2     final static float pi = (float)3.1415;  
3  
4     public static void main (String []args) {  
5         float raggio = 5;  
6         float circ = circonferenza(raggio);  
7         float area = area(raggio);  
8         System.out.println("Raggio " + raggio + " -> circonferenza " + circ + " area " + area);  
9     }  
10  
11     public static float circonferenza(float raggio) {  
12         float res = 2 * raggio * pi;  
13         return res;  
14     }  
15  
16     public static float area(float raggio) {  
17         float res = 2 * pi * raggio * raggio;  
18         return res;  
19     }  
20 }
```

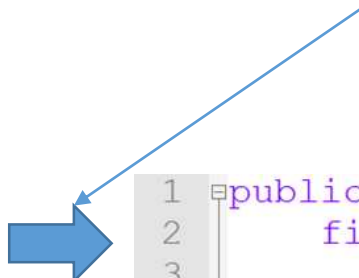
# Le variabili statiche

- Definite nello *scope* della classe anziché metodo
  - Accessibili da tutti i metodi della classe
- Read-only con modificatore *final*
  - Es: costanti condivise da più metodi della classe
- Sono allocate nella *static memory*
  - Non sono allocate nella stack memory
- *Condivise da tutte le istanze della classe*



# Le variabili statiche

Prossima linea da eseguire  
(non ancora eseguita)



```
1 public class CirconferenzaAreaStatPi {  
2     final static float pi = (float)3.1415;  
3  
4     public static void main (String []args) {  
5         float raggio = 5;  
6         float circ = circonferenza(raggio);  
7         float area = area(raggio);  
8         System.out.println();  
9     }
```



32 bit

*static memory*



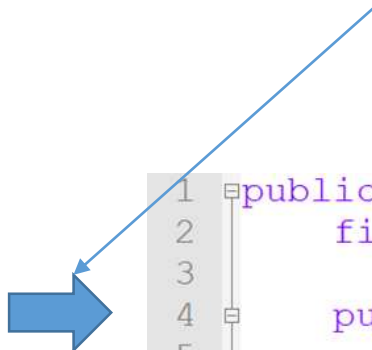
32 bit

*stack memory*

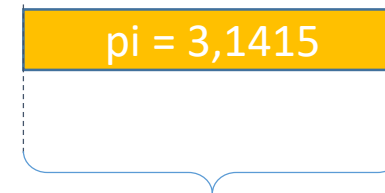


# Le variabili statiche

Prossima linea da eseguire  
(non ancora eseguita)



```
1 public class CirconferenzaAreaStatPi {  
2     final static float pi = (float)3.1415;  
3  
4     public static void main (String []args) {  
5         float raggio = 5;  
6         float circ = circonferenza(raggio);  
7         float area = area(raggio);  
8         System.out.println();  
9     }
```



32 bit

*static memory*

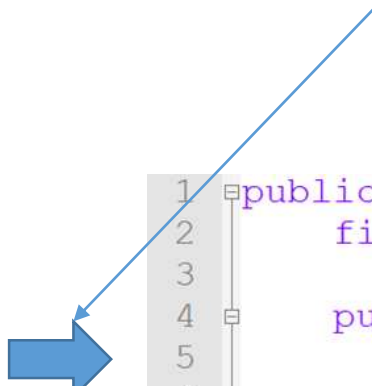


32 bit

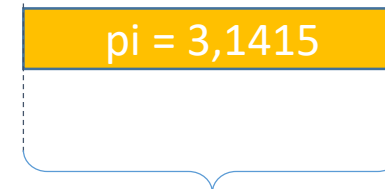
*stack memory*

# Le variabili statiche

Prossima linea da eseguire  
(non ancora eseguita)

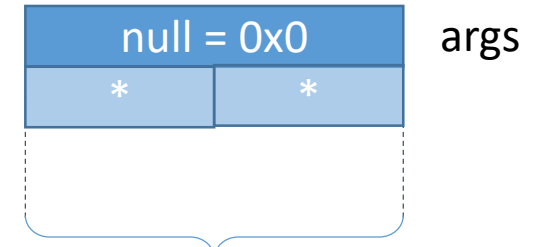


```
1 public class CirconferenzaAreaStatPi {  
2     final static float pi = (float)3.1415;  
3  
4     public static void main (String []args) {  
5         float raggio = 5;  
6         float circ = circonferenza(raggio);  
7         float area = area(raggio);  
8         System.out.println();  
9     }
```



32 bit

*static memory*



32 bit

*stack memory*

# Le classi di test

# Testing del codice

- Finora tutto il nostro codice in una sola classe
  - Metodi specifici implementano algoritmi specifici
  - Il metodo main richiama il metodo da *testare* e stampa *l'output*

```
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 }
```

# Testing del codice

- Due problemi fondamentali in questo approccio
  - Testing non automatizzato
  - *Business logic* e *test logic* mischiate nello stesso file

```
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 }
```

# Testing del codice

```
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 }
```

```
>java CirconferenzaAll  
Raggio 5.0 -> circonferenza 31.415
```



# Testing del codice

*Automatizzabile  
da file batch*

```
public static void main (String []args) {  
    float pi = (float)3.1415;  
    float raggio = 5;  
    float valAtteso = (float)31.415;  
    float circ = circonferenza(pi, raggio);  
    System.out.println("circonferenza(" + raggio + ") returns " + circ + " -> " + (circ == valAtteso));  
}
```

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



```
>java Circonferenza  
circonferenza(5.0) returns 31.45 -> true
```

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



```
>java Circonferenza  
circonferenza(5.0) returns 15.7075 -> false
```

# Testing del codice

- Due problemi fondamentali in questo approccio
  - ~~Testing non automatizzato~~
  - *Business logic* e *test logic* nello stesso file

```
public class Circonferenza {  
    public static void main (String []args) {  
        float pi = (float)3.1415;  
        float raggio = 5;  
        float valAtteso = (float)31.415;  
        float circ = circonferenza(pi, raggio);  
        System.out.println("circonferenza(" + raggio + ") returns " + circ + " -> " + (circ == valAtteso));  
    }  
  
    public static float circonferenza(float pi, float raggio) {  
        float res = 2 * pi * raggio;  
        return res;  
    }  
}
```



# Testing del codice – il *test case*

- Creare la (le) classe(i) con gli algoritmi organizzati in metodi statici
  - es: classe Cerchio.java, metodi *circonferenza()* ed *area()*
- Creare una classe di Test (detto *TestCase*)
  - Es: classe *CerchioTest* per verificare la classe *Cerchio*
- La classe di test contiene solo il metodo main ()
  - (Crea un'istanza della classe da testare)
  - Invoca il(i) metodo(i) opportuno(i) (con relativi parametri)
  - Verifica che l'output sia quello atteso (vedi sopra)
- Per ora lavoreremo senza supporto ad oggetti
  - Esclusivamente metodi *statici* con relative conseguenze

# Testing del codice – il *test case*

*Cerchio.java*

```
1 public class Cerchio {  
2     public static final float pi = (float)3.1415;  
3  
4     public static float circonferenza(float raggio) {  
5         float res = 2 * Cerchio.pi * raggio;  
6         return res;  
7     }  
8 }
```

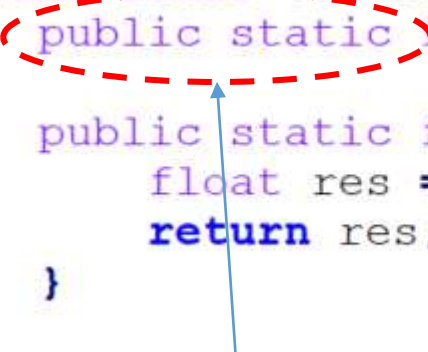
*CerchioTest.java*

```
1 public class CerchioTest {  
2     public static void main (String []args) {  
3         float raggio = 5;  
4  
5         float valAtteso = (float)31.415;  
6         float circ = Cerchio.circonferenza(raggio);  
7         System.out.println("Cerchio.circonferenza(" + raggio + ") returns "  
8             + circ + " -> " + (circ == valAtteso));  
9     }  
10 }
```

# Testing del codice – la classe da verificare

*Cerchio.java*

```
1 public class Cerchio {  
2     public static final float pi = (float)3.1415;  
3  
4     public static float circonferenza(float raggio) {  
5         float res = 2 * Cerchio.pi * raggio;  
6         return res;  
7     }  
8 }
```



Variabile *pi* di tipo *float*

*public*: accessibile da classi diverse da *Cerchio*

*static*: accessibile anche da metodi *static*

*final*: non modificabile a runtime

# Testing del codice – la classe da verificare

*Cerchio.java*

```
1 public class Cerchio {  
2     public static final float pi = (float)3.1415;  
3  
4     public static float circonferenza(float raggio) {  
5         float res = 2 * Cerchio.pi * raggio;  
6         return res;  
7     }  
8 }
```

Metodo *circonferenza()* é *static*  
Può accedere a variabili *static*  
Accessibile senza *istanziatura* di classe

Variabile *pi* di tipo *float*

*public*: accessibile da classi diverse da *Cerchio*

*static*: accessibile anche da metodi *static*

*final*: non modificabile a runtime

# Testing del codice – la classe da verificare

*Cerchio.java*

```
1 public class Cerchio {  
2     public static final float pi = (float)3.1415;  
3  
4     public static float circonferenza(float raggio) {  
5         float res = 2 * Cerchio.pi * raggio;  
6         return res;  
7     }  
8 }
```

Attenzione alla  
sintassi  
<classe>.variabile

Metodo *circonferenza()* é *static*  
Può accedere a variabili *static*  
Accessibile senza *istanziatura* di classe

Variabile *pi* di tipo *float*

*public*: accessibile da classi diverse da *Cerchio*

*static*: accessibile anche da metodi *static*

*final*: non modificabile a runtime

# Testing del codice – il *test case*

Attenzione alla sintassi per la chiamata a metodo *static*  
`<classe>.metodo()`

*CerchioTest.java*

```
1 public class CerchioTest {  
2     public static void main (String []args) {  
3         float raggio = 5;  
4  
5         float valAtteso = (float)31.415;  
6         float circ = Cerchio.circonferenza(raggio);  
7         System.out.println("Cerchio.circonferenza(" + raggio + ") returns "  
8                             + circ + " -> " + (circ == valAtteso));  
9     }  
10 }
```

# Testing del codice – procedura

- Ipotesi: tutte le classi nella directory corrente
  - altrimenti impostare CLASSPATH
- Compilare la (le) classe(i) di test
  - `javac Cerchio.java` -> produce `Cerchio.class`
- Compilare la classe di test
  - `javac CerchioTest.java` -> produce `CerchioTest.class`
- Eseguire la classe di test
  - `java CerchioTest`
- Se modifico il file `xyz.java` ricompilerò solo quello

# Esercizio: metodo calcolo area

- Scaricare i sorgenti *Cerchio.java*, *CerchioTest.java*, compilarli e verificare che il test del metodo *circonferenza()* sia superato.
- Implementare un nuovo metodo *area()* nella classe *Cerchio* e verificare il suo funzionamento tramite aggiunta di un opportuno test nella classe di test
  - Come gestire eventuali arrotondamenti nelle operazioni fra numeri in virgola mobile, ovvero verificare che il risultato sia corretto con margine *epsilon* a piacere ?



# Soluzione: metodo calcolo area

*Cerchio.java* (mutualmente esclusivi)

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

*CerchioTest.java*

```
public static void main (String []args) {  
    float raggio = 5;  
  
    float valAttesoArea = (float)157.075;  
    float area = Cerchio.area(raggio);  
    System.out.println("Cerchio.area(" + raggio + ") returns "  
        + area + " -> " + (area == valAttesoArea));  
}
```

*Risultato*

```
>java Circonferenza  
Cerchio.area(5.0) returns 157.07501 -> false
```

# Soluzione: metodo calcolo area

*CerchioTest.java*

```
public static boolean compareEpsilon (float input, float target, float epsilon) {  
    float diff = input - target;  
    if (diff < 0)  
        diff = -diff;  
  
    if (diff < epsilon)  
        return true;  
  
    return false;  
}
```

```
System.out.println("Cerchio.area(" + raggio + ") returns " + area + " -> " +  
    CerchioTest.compareEpsilon(area, valAttesoArea, (float)0.0001));
```

Quale sarà l'epsilon  
massimo tollerabile ?

*Risultato*

```
>java Circonferenza  
Cerchio.area(5.0) returns 157.07501 -> true
```

# Esercizio: libreria di funzioni aritmetiche

# Esercizio: libreria di funzioni aritmetiche

Si sviluppi una classe che implementi le funzioni aritmetiche somma, sottrazione, moltiplicazione, divisione, esponenziale, resto.

Per scelta, i metodi siano definiti senza rifarsi ad operatori Java built-in (es: +, -, \*, /, ...) ma utilizzando le funzioni di assegnamento, incremento e decremento 1, salto condizionato effettivamente disponibili in alcuni microcontrollori.

Si sviluppi quindi l'opportuna classe di test e si collaudi la classe sviluppata.

# Libreria di funzioni aritmetiche – somma(a,b)

- Approccio: *sommo ad a il valore 1 un numero di volte pari a b*
  - Memorizzo i risultati parziali nell'*accumulatore* res

```
public static int piu(int a, int b) {  
    int res = a;  
    int i = b;  
    while (i > 0) {  
        res = res + 1;  
        i = i - 1;  
    }  
    return res;  
}
```

Esercizio: libreria di funzioni  
booleane

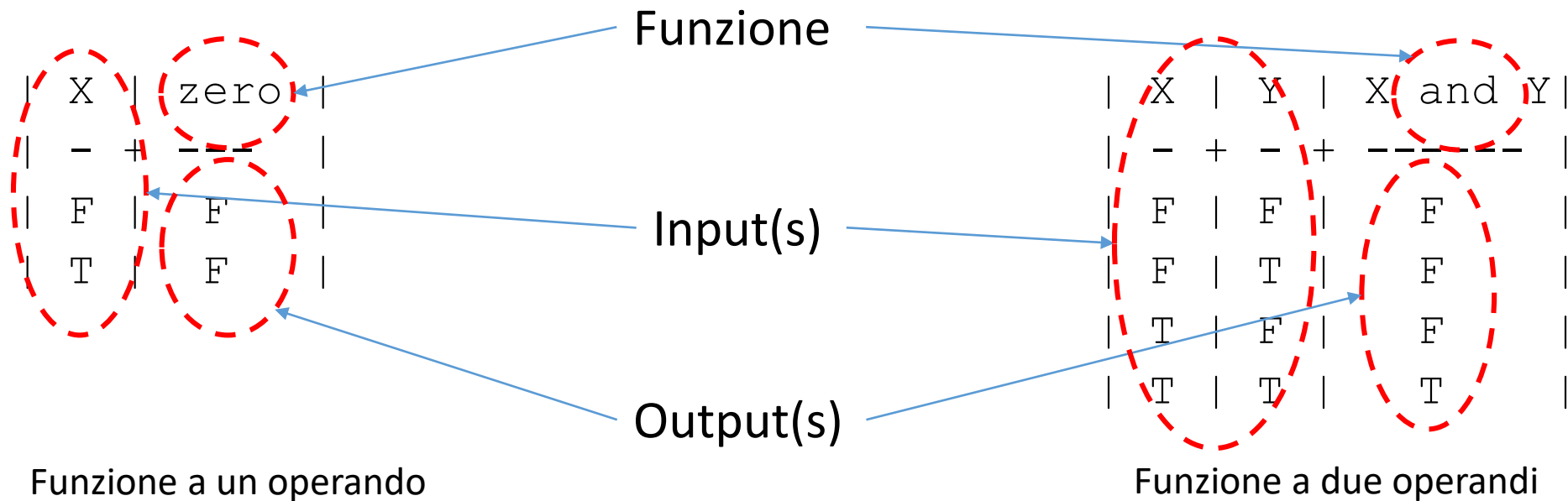
# Esercizio: libreria di funzioni booleane

Si sviluppi una classe che implementi i metodi che realizzano le 16 funzioni booleane a due argomenti specificate nel file *201022-050-Tabelle-di-verita.java*.

Per scelta, i metodi siano definiti usando in maniera essenziale solo i test annidati necessari a produrre il risultato senza rifarsi ad operatori Java built-in (es: `&&`, `//`, `!`).

Si sviluppi quindi l'opportuna classe di test e si collaudi la classe sviluppata.

# Libreria di funzioni booleane - Esempio



```
public static boolean zero(boolean x, boolean y) {  
    return false;  
}
```

```
public static boolean and(boolean x, boolean y)  
{  
    boolean res = false;  
    if(x) {  
        if(y) {  
            res = true;  
        }  
    }  
    return res;  
}
```



# Esercizio: libreria di funzioni grafiche -1

Scrivere una classe Asterischi.java e la corrispondente AsterischiTest.java. Asterischi.java contiene metodi con le seguenti caratteristiche

- `aCapo()` con l'ovvio significato d'andare a capo
- `riga(int n, char c)` -> stampa una riga con n copie del carattere c

ESEMPIO. `riga(10, '*')` stamperà:

```
>java Asterischi
*****
```

- `rettangolo(int n, int m, char c)` -> stampa un rettangolo con n\*m copie del carattere c.

ESEMPIO. `rettangolo(5,7,'*')` stamperà:

```
>java Asterischi
*****
*****
*****
*****
*****
```

# Esercizio: libreria di funzioni grafiche -2

- `triangoloEqSx(int x, char c)` -> stampa un triangolo equilatero sinistro di altezza e base  $x$ .
- `triangoloRovEqSx(int x, char c)` -> stampa un triangolo equilatero rovesciato sinistro di altezza e base  $x$ .
- `triangoloRovEqDx(int x, char c)` -> stampa un triangolo equilatero rovesciato destro di altezza e base  $x$ .
- `triangoloEqDx(int x, char c)` -> stampa un triangolo equilatero destro di altezza e base  $x$ .
- `tetto(int x, char c)` -> stampa un 'tetto' di altezza  $x$  e base  $2x-1$ , assumendo  $x \geq 1$ .