Fondamenti di informatica T-1 (A – K)
Esercitazione 7: metodi, classi, oggetti

AA 2018/2019

Tutor

Lorenzo Rosa

lorenzo.rosa@unibo.it

Esercitazione 7

- Introduzione al calcolatore e Java
- Linguaggio Java, basi e controllo del flusso
- Eclipse ed esercizi di consolidamento
- Stringhe ed array
- **Metodi, classi, oggetti**
- Ereditarietà e polimorfismo
- Collezioni Java e interfacce
- Esercizi d'esame

I metodi

- Alcune sequenze di istruzioni vanno ripetute più volte all'interno di un programma: pertanto, è utile poter scrivere tali sequenze una volta sola e farvi riferimento ogni volta che serve. I **metodi** assolvono a questa funzione.

- Un **metodo** raggruppa una sequenza di istruzioni che realizzano una funzionalità del programma e assegna loro un **nome**.

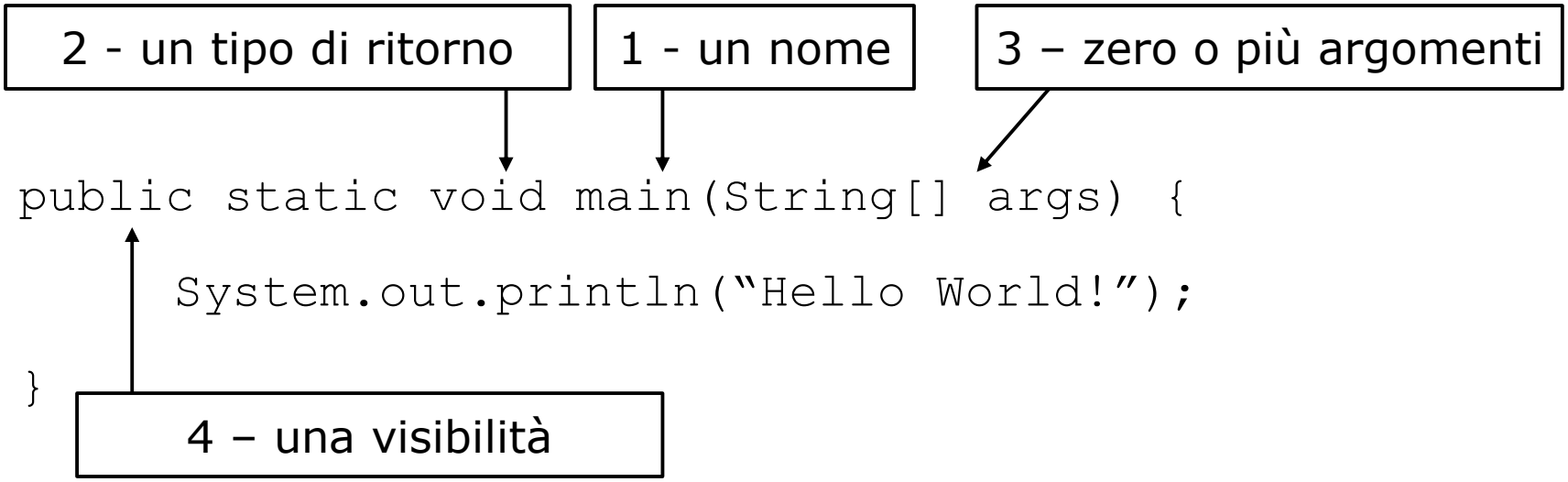
- Quando si usa un metodo, si dice che si **invoca** (o si **chiama**) il metodo stesso.

Il metodo `main`

- Anche il *main* è un metodo. In particolare, è un **metodo statico**, ovvero appartiene alla classe in cui è definito, ma non alle sue istanze.
- Da un metodo statico, si possono chiamare altri metodi soltanto se anch'essi sono statici.
- Una classe può essere usata per raggruppare metodi statici.

I metodi

- Ogni metodo ha:



- Un metodo appartiene sempre a una classe. Può appartenervi come metodo **statico** o come metodo **di istanza**.

Esercizio 1

- Creare un programma Java (classe `Prova`) in cui dal metodo main si chieda all'utente di inserire un intero N.

- Si definisca un metodo *`stampa()`* che stampi a video la stringa "Hello world!" e lo si invochi N volte.

Esercizio 1 - Soluzione

```
import java.util.Scanner;

public class Prova {

  public static void stampa() {
       System.out.println("Hello world!");
  }

  public static void main(String[] args) {
       Scanner tastiera = new Scanner(System.in);
       System.out.print("Inserire un numero intero: ");
       int num = tastiera.nextInt();

       //Controllo sia positivo
       if(num < 0) {
              System.out.print("Il valore deve essere positivo!");
              System.exit(0);
       }
       for(int i=0; i< num; i++)
              stampa();
  }
}
```

Tipo di ritorno - void

- L'intestazione dei metodi che non restituiscono alcun valore, come visto nell'esercizio precedente, vanno preceduti dalla parola **void**.

```
public static void stampa() {
   System.out.println("Hello world!");
}
```

Metodi che restituiscono un valore

1. L'intestazione di un metodo che restituisce un valore è simile, ma al posto della parola chiave **`void`** bisogna indicare il **tipo di ritorno**.

2. Il corpo della definizione di un metodo che restituisce un valore è simile a quello di un metodo **`void`**, ma al suo interno deve contenere almeno **un'istruzione `return` .**

Metodi che restituiscono un valore

```
public static int zero() {
      int v = 0;
      return v;
}

public static void main {
   ....
   int n = zero();
   System.out.println(n); // Stampa 0
  ...
}
```

Metodi con parametri

- Ad un metodo è possibile anche **passare dei parametri**, utilizzandolo come una **funzione** matematica.

- Nella definizione di un metodo, è necessario specificare quali sono i parametri che gli dovranno essere passati nell'intestazione, fra parentesi e dopo il nome del metodo.

```
public static void stampa(int i) {
  System.out.println("Valore: " + i);
}
```

Metodi con parametri

```
public static void stampa(int i) {
   System.out.println("Valore: " + i);
}
```

- `i` viene detto **parametro formale**. Il valore che assumerà è quello passato dal chiamante e viene detto **parametro attuale**.

- La visibilità della variabile (*scope*) è il **corpo del metodo**.

Metodi con parametri

- Quando viene chiamato un metodo, ogni suo parametro viene inizializzato con il valore dell'argomento corrispondente nell'invocazione del metodo.

```
public static void main(String[] args) {
        ...
        int num = 3;
        for(int i=0; i< num; i++)
                stampa(i);
 }
```

- Utilizzando il metodo della slide precedente, questo programma stampa:

```
Valore: 0
Valore: 1
```

Metodi con parametri

- Nel caso in cui il metodo accetti più parametri, questi devono essere passati nell'ordine in cui il metodo li aspetta.

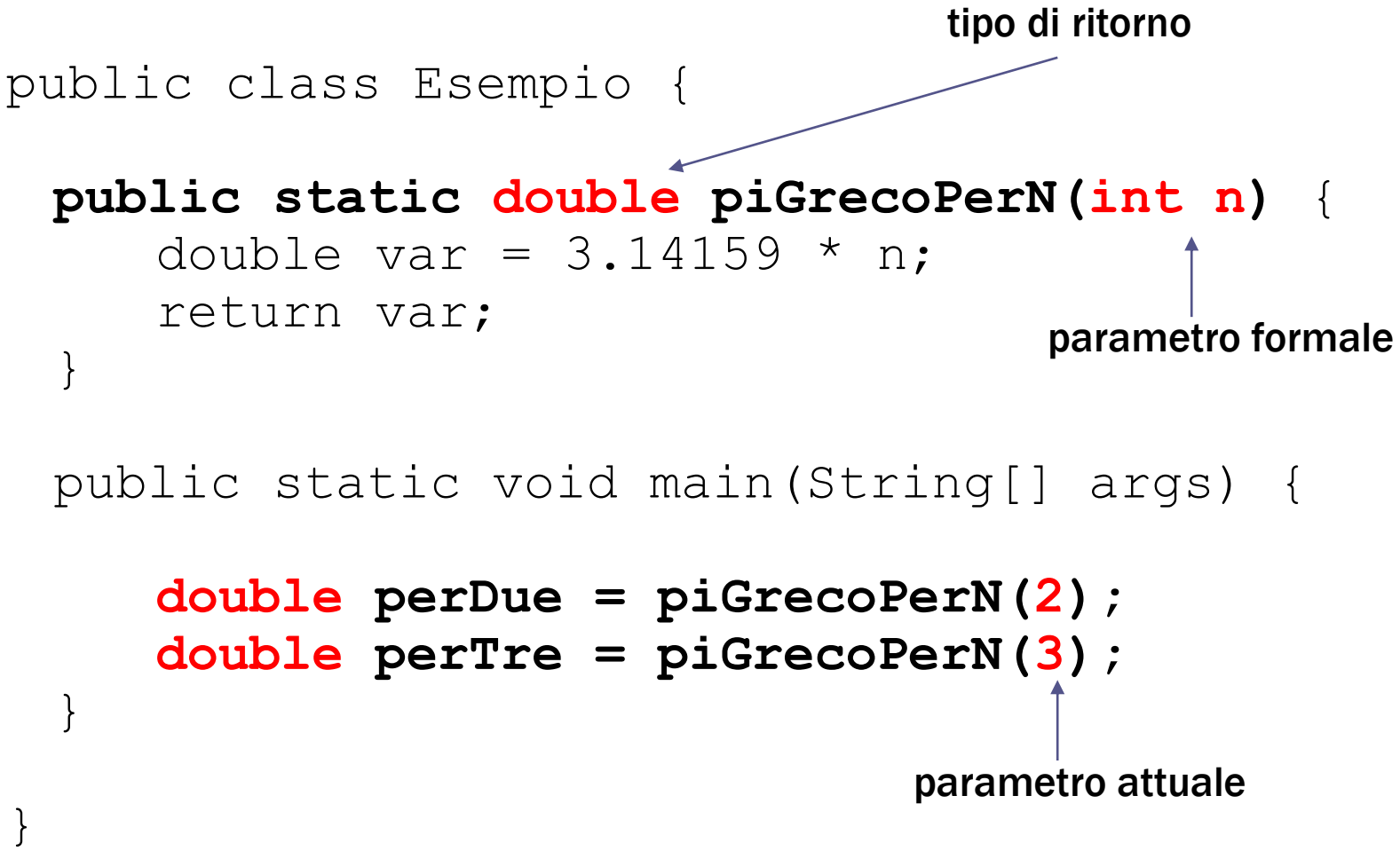
```
public static void stampa(int i, String s) {
 System.out.println(i + " " + s);
}

public static void main {
   ....
   int n = 0;
   String g = "ciao";
   stampa(n, g);
  ...
}
```

1. nome
2. tipo di ritorno
3. tipo e l'ordine dei parametri

formano
la **firma (*signature*)** del metodo.

Esempio


tipo di ritorno
public class Esempio {
public static double piGrecoPerN(int n) {
double var = 3.14159 * n;
return var;
}
public static void main(String[] args) {
double perDue = piGrecoPerN(2);
double perTre = piGrecoPerN(3);
}
}
parametro formale
parametro attuale

Variabili locali ai metodi

Ricordare sempre:

- Una variabile definita all'interno di un metodo o più in generale in un blocco (parentesi graffe) è detta "locale" di tale metodo o blocco.

- Le variabili locali possono essere usate esclusivamente all'interno del metodo o blocco in cui sono state definite.

- Se due metodi o blocchi hanno variabili locali con lo stesso nome, si tratta comunque di due variabili distinte.

Esercizio 2

- Creare un programma Java (classe Aree) per il calcolo dell'area di tre forme geometriche: triangolo, rettangolo, cerchio.

1. Il programma dovrà chiedere all'utente di inserire:
   - `"t" se vuole calcolare l'area di un triangolo;`
   - `"r" se vuole calcolare l'area di un rettangolo;`
   - `"c" se vuole calcolare l'area di un cerchio;`
   - `"f" per terminare l'esecuzione.`
2. In base alla scelta fatta, il programma dovrà chiedere all'utente i dati di input necessari, calcolare il risultato, stamparlo a video e tornare al punto 1 (se non ha scelto di terminare).

- Per il calcolo delle aree, definire un metodo per ciascuna forma e invocarlo opportunamente.

Esercizio 2 – Soluzione (1/4)

```
import java.util.Scanner;

public class Aree {

  public static double calcolaAreaTriangolo (double base,
                                             double altezza) {
      return (base * altezza)/2;
  }

  public static double calcolaAreaRettangolo(double base,
                                             double altezza) {
      return base * altezza;
  }

  public static double calcolaAreaCerchio(double raggio) {
      return 3.14 * raggio * raggio;
  }
```

Esercizio 2 – Soluzione (2/4)

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

  Scanner tastiera = new Scanner(System.in);
  String scelta;

  do {
   System.out.println("Inserire: ");
   System.out.println("t per l'area di un triangolo");
   System.out.println("r per l'area di un rettangolo");
   System.out.println("c per l'area di un cerchio");
   System.out.println("f per uscire");

  scelta = tastiera.nextLine();
  double result;
```

Esercizio 2 – Soluzione (3/4)

```
if(scelta.equalsIgnoreCase("t")) {
  double base, altezza;
  System.out.print("Inserire base: ");
  base = tastiera.nextDouble();
  System.out.print("Inserire altezza: ");
  altezza = tastiera.nextDouble();

  result = calcolaAreaTriangolo(base, altezza);
  System.out.println("Area Triangolo: " + result);
}
else if(scelta.equalsIgnoreCase("r")) {
  double base, altezza;
  System.out.print("Inserire base: ");
  base = tastiera.nextDouble();
  System.out.print("Inserire altezza: ");
  altezza = tastiera.nextDouble();

  result = calcolaAreaRettangolo(base, altezza);
  System.out.println("Area Rettangolo: " + result);
}
```

Esercizio 2 – Soluzione (4/4)

```
    else if(scelta.equalsIgnoreCase("c")) {
       double raggio;
       System.out.print("Inserire raggio: ");
       raggio = tastiera.nextDouble();

       result = calcolaAreaCerchio(raggio);
       System.out.println("Area Cerchio: " + result);
    }
  } while(!scelta.equalsIgnoreCase("f"));
 }
}
```

Classi e oggetti

- Una **classe** rappresenta l'astrazione di un'entità del mondo reale. Le istanze di una classe sono dette **oggetti**.
- Una **classe** è la definizione di un **tipo** di oggetto. È come uno stampo per la costruzione di oggetti di un certo tipo.
- A prescindere dall'oggetto che si vuole modellare una classe sarà sempre costituita da:
  - dati (chiamati **attributi**)
  - metodi che operano su di essi
    - metodi statici: appartengono alla classe, non alle sue istanze.
    - metodi di istanza: appartengono a ciascun oggetto.

Esempio di classe

- Classe **`Automobile`**: modella il concetto di automobile attraverso una serie di attributi e operazioni distintivi.

- Attributi:
  - Targa
  - Livello carburante
  - Velocità
  - …

- Metodi:
  - Accelera
  - Accendi
  - Spegni
  - ….

In Java
1 classe = 1 file

Implementazione in Java

definita nel file Automobile.java

```
public class Automobile {

    private double carburante;
    private double velocita;
    private String targa;

    public void setTarga(String nuova_targa){
            targa = nuova_targa;
    }

    public void cambiaVelocita(double delta){
            velocita += delta;
    }

    public void cambiaCarburante(double delta){
            carburante += delta;
    }
}
```

Implementazione in Java

```
public class Automobile {

    private double carburante;
    private double velocita;
    private String targa;

    public void setTarga(String nuova_targa){
            targa = nuova_targa;
    }

    public void cambiaVelocita(double delta){
            velocita += delta;
    }

    public void cambiaCarburante(double delta){
            carburante += delta;
    }
}
```

Attributi

Implementazione in Java

```
public class Automobile {

    private double carburante;
    private double velocita;
    private String targa;

    public void setTarga(String nuova_targa){
            targa = nuova_targa;
    }

    public void cambiaVelocita(double delta){
            velocita += delta;
    }

    public void cambiaCarburante(double delta){
            carburante += delta;
    }
}
```

Metodi di istanza

Implementazione in Java

```
public class Automobile {

    private double carburante;
    private double velocita;
    private String targa;

    public void setTarga(String nuova_targa){
            targa = nuova_targa;
    }

    public void cambiaVelocita(double delta){
            velocita += delta;
    }

    public void cambiaCarburante(double delta){
            carburante += delta;
    }
}
```

Quale valore assumono all'inizio?

I costruttori, un caso speciale di metodo

Si chiama **costruttore** uno speciale metodo che non ha tipo di ritorno ed il cui nome coincide con quello della classe.

- Il costruttore permette di creare un oggetto, **inizializzando i campi di quella istanza** della classe.
- Un costruttore viene sempre automaticamente invocato quando una classe viene istanziata
  - Se non si definisce un costruttore esplicitamente, ne viene creato uno di default.
  - Es: public Automobile(){}
- Un costruttore può avere dei parametri e utilizzarli per inizializzare i campi dell'oggetto.

Costruttore per Automobile

- Un possibile costruttore per Automobile potrebbe essere:

```
public Automobile(String targa){
      this.targa = targa;
      carburante = 0;
      velocita = 0;
}
```

- La keyword **this** si utilizza quando il parametro formale del costruttore ha lo stesso nome del campo privato della classe.

```
public Automobile(String targa){
      this.targa = targa;
```

parametro formale del costruttore

campo privato: "this" ci fa capire che è il campo "targa" di Automobile

Implementazione in Java

```
public class Automobile {

    private double carburante;
    private double velocita;
    private String targa;

    public Automobile(String targa){
            this.targa = targa;
            carburante = 0;
            velocita = 0;
    }

    public void setTarga(String nuova_targa){
            targa = nuova_targa;
    }
    ... altri metodi ...
}
```

Costruttore

Istanziare un oggetto

- Per poter usare un oggetto "Automobile", è necessario creare un'istanza della classe Automobile, invocando il costruttore attraverso la keyword *new*.

Il codice Java deve sempre stare in un metodo, e un metodo in una classe. Quindi abbiamo bisogno di una classe anche per il main.

```
public class Esercizio {
    public static void main(String[] args){
      Automobile auto = new Automobile("AF345BD");
      //poi usare l'oggetto
      auto.cambiaVelocita(30.3);
      ....
    }
}
```

Istanziare un oggetto

```
public class Esercizio {
    public static void main(String[] args){
       Automobile a1 = new Automobile("AF345BD");
       Automobile a2 = new Automobile("AF345BC");
       ....
    }
}
```

a1 e *a2* sono due **istanze della stessa classe**: condividono la stessa struttura (hanno entrambe una velocita, una targa, ecc.) ma sono **oggetti distinti**, perché i valori dei campi interni (stato dell'oggetto) possono essere diversi.

Istanziare un oggetto

- A differenza di quanto abbiamo sempre fatto durante le esercitazioni precedenti, ora (e all'esame) è quindi necessario creare:
  - una classe che contiene solo il main, in un file .java;
  - zero o più classi, ciascuna in un diverso file .java.

- In questo caso:
  - un file EsercizioAutomobile.java → contiene il main;
  - un file Automobile.java → contiene la classe Automobile.

Utilizzare i metodi

- Per accedere a metodi ed attributi pubblici di un oggetto basta utilizzare la notazione puntata:

```
public class Esercizio {

    public static void main(String[] args){

       Automobile auto = new Automobile("AF345BD");

       //poi usare l'oggetto

       auto.cambiaVelocita(30.3);

       ....

    }

}
```

Possiamo accedere a questo metodo dall'esterno della classe Automobile perché il metodo è public. Non possiamo invece accedere agli attributi, che abbiamo dichiarato come privati.

Esercizio 4

- Creare la classe ***`Variabile`*** dotata di
  - un campo privato intero chiamato “valore”;
  - un costruttore senza argomenti che inizializzi a zero “valore”;
  - Un metodo `getValore()` che restituisca il valore il “valore”;
  - Un metodo `setValore(int)` che imposti il valore di “valore”;
  - Un metodo `resetValore()` che azzeri il valore di “valore”;

- Creare la classe ***`Esercizio4`*** che
  - Definisca il metodo *`main`*
  - Crei un'istanza della classe *`Variabile`*
  - Chiami il metodo *`setValore(8)`* sull'istanza
  - Stampi a video il valore attuale di "valore" utilizzando il metodo *getValore();*
  - Chiami il metodo *resetValore()* sull'istanza;
  - Stampi a video il valore attuale di "valore" utilizzando il metodo *getValore();*

Esercizio 4 – soluzione (1/2)

La classe *Variabile:*

FILE: Variabile.java

```
public class Variabile {
        private int valore;

        public Variabile() {
                this. valore = 0;
        }

        public int getValore() {
                return valore;
        }

        public void setValore(int valore) {
                 this.valore = value;
        }

        public void resetValore() {
                valore = 0;
        }
}
```

***getter/setter* per l'attributo `valore`**

Esercizio 4 – soluzione (2/2)

La classe *Esercizio4*:

FILE: Esercizio4.java

```
public class Esercizio4 {
  public static void main(String args[]) {
      //Creo l'istanza
      Variabile var = new Variabile();

      //Imposto il valore 8
      var.setValore(8);
      System.out.println("Valore variabile: " +
                                          var.getValore());

      //Reset del valore
      var.resetValore();
      System.out.println("Valore variabile: " +
                                           var.getValore());
  }
}
```

Visibilità

- I metodi che abbiamo inserito nella classe *Variabile* sono davvero necessari?
- Sì, perché l'attributo "valore" contenuto nella classe *Variabile* è dichiarato come **private**, quindi non è visibile esternamente alla classe. I metodi invece, dichiarati come pubblici, lo sono.
- Provate ad accedere al campo "valore" dal *main*, tramite la notazione puntata. Il compilatore ve lo consente? Perché?
- Provate a modificare la visibilità della variabile "valore" da *private* in *public*. Ora i metodi inseriti sono ancora necessari?

Esercizio 5

- Vogliamo creare un oggetto che mantenga lo stato di un contatore.

- Modelliamo il concetto di contatore con una classe, che chiameremo *Contatore:*

  - Un campo intero che rappresenta il valore del contatore;
  - Un costruttore che prende come argomento un valore intero, a cui inizializzare il valore interno;
  - Un metodo *incrementa()* che incrementa di una unità il valore;
  - Un metodo *reset()* che azzera il contatore;
  - Un metodo *getValore()* che ritorna il valore del contatore;

Soluzione «Contatore»

```
public class Contatore {

     private int valore;

     public Contatore(int valIniziale) {
              valore = valIniziale;
     }

     public void incrementa() {
              valore = valore+ 1;
     }

     public void reset() {
              valore = 0;
     }

     public int getValore() {
              return valore;
     }

}
```

Esercizio 6/1 (tipo esame)

- L'agenzia di viaggi "Sedano" ha deciso di informatizzare la gestione degli itinerari proposti ai turisti che si recano a visitare lo stato caraibico di St. Marquez.

- Innanzitutto, vanno memorizzati i dati relativi ad ogni località turistica, ovvero il nome della località e la durata (in giorni) ed il costo (in euro) di una visita.

- Si scriva una classe Localita per l'agenzia "Sedano" che:
  1. Possieda un opportuno costruttore con parametri.
  2. Presenti opportuni metodi che permettano di accedere alle variabili di istanza dell'oggetto.
  3. Presenti il metodo toString che fornisca la descrizione della località.

Esercizio 6/2 (tipo esame)

Si scriva un'applicazione (classe `Applicazione`) per l'agenzia "Sedano" che:

1. crei un oggetto Localita, chiedendo all'utente da tastiera le informazioni necessarie.
2. Stampi a video la descrizione della località appena creata.