Linguaggi di Programmazione

a.a. 14/15

docente: Gabriele Fici

gabriele.fici@unipa.it

3 - Le Classi (parte II)

3 - Le Classi

- Un concetto molto importante nella programmazione OO è l'*incapsulamento* dei dati
- I metodi e le variabili dell'interfaccia pubblica vengono dichiarati con il modificatore `public`
- Quelli dichiarati con il modificatore `private` non possono essere acceduti dall'esterno della classe

Esempio:

```
private int livello;          // attributo privato
public int getLivello() {     // metodo pubblico
   return livello;
}
```

Abbiamo fornito un metodo pubblico per accedere ad un attributo privato, senza possibilità di modificarlo

3 - Le Classi

- In genere è buona norma dichiarare i metodi e le classi con il modificatore `public` mentre gli attributi con il modificatore `private`

Esempio:

```
public class Serbatoio {

   private int livello;

   public Serbatoio (){ livello = 10; }

   public void rifornimento (int j) {
       livello += j; }

   public int getLivello () {
       return livello; }
}
```

3 - Le Classi

- Quando in un metodo richiamiamo un attributo della classe, implicitamente ci stiamo riferendo alla variabile *dell'oggetto a cui sarà applicato il metodo*
- A volte può essere utile esplicitare questo parametro, usando il riferimento `this`

Esempio:

```
public Serbatoio () {
  this.livello = 10; // equiv. a livello = 10
}

public void rifornimento (int j) {
  this.livello += j;  // equiv. a livello += j
}
```

3 - Le Classi

- Ad esempio, se vogliamo realizzare un metodo `travasa`, che sposta il contenuto da un serbatoio a un altro, possiamo rendere il codice più chiaro con `this`

```
void travasa (int l, Serbatoio s){
  this.rifornimento(l);
  s.consumo(l);
}
...

Serbatoio t1 = new Serbatoio();
Serbatoio t2 = new Serbatoio();

t1.travasa(8, t2);
```

3 - Le Classi

- Un'altra applicazione tipica del `this` è quella di usare lo stesso nome per un attributo e per una variabile locale (anche se non è necessario farlo)

```
public Serbatoio (int livello){

  this.livello = livello;
  // livello = livello va bene (perché?)
  // ma è sconsigliato
}
```

3 - Le Classi

- All'interno di una classe possono esserci più metodi con lo stesso nome, ma devono differire per numero e/o tipo di parametri (cosiddetta *firma* del metodo)
- Questo principio si chiama *overloading* di metodi
- Tipicamente, questo si fa per i costruttori

Esempio:

```
Serbatoio (){     // primo metodo costruttore
  this.livello = 10;
}
Serbatoio (int j){ // secondo metodo costruttore
  this.livello = j;
}
```

3 - Le Classi

- Se cambia solo il tipo di ritorno, il compilatore vede un metodo duplicato e dà errore
- Tuttavia, se i parametri sono diversi, i valori di ritorno possono essere diversi

Esempio:

```
int minimo (int a, int b){…}

double minimo (double a, double b){…} // overloading

double minimo (int a, int b){…} // errore
```

3 - Le Classi

- In caso di overloading con parametri numerici di tipi compatibili, ci sono delle regole automatiche di scelta (i tipi più piccoli di `int` interpretati come `int`, mentre `float` come `double`)

```
public class Num {

   Num(short i){ System.out.println("short " + i); }

   Num(int i){ System.out.println("int " + i); }

   Num(float i){ System.out.println("float " + i); }

   Num(double i){ System.out.println("double " + i); }

   public static void main(String[] args){
     Num ogg1 = new Num(123456789);
     Num ogg2 = new Num((short) 123456789);
     Num ogg3 = new Num(.123456789);
     Num ogg4 = new Num(.123456789f);  }
 }
```

3 - Le Classi

- Il parametro `this` seguito dalle parentesi si usa per invocare un costruttore all'interno di un altro costruttore
- Va messo come prima istruzione del costruttore
- Come fa il compilatore a sapere quale costruttore deve richiamare con `this`? Dipende dai parametri, infatti ci può essere un solo costruttore con una data firma

3 - Le Classi

Esempio:

```
public class Rectangle {
  private int x, y;
  private int width, height;

  public Rectangle(int x, int y, int width, int height)
       { this.x = x;
         this.y = y;
         this.width = width;
         this.height = height; }

  public Rectangle()
       { this(0, 0, 1, 1); }

  public Rectangle(int width, int height)
       { this(0, 0, width, height); }
   ...
```

3 - Le Classi

- E' possibile dichiarare attributi costanti mediante la parola chiave `final`
- Questi attributi non sono modificabili
- E' consigliabile inizializzarli al momento della dichiarazione (anche se non è obbligatorio)
- Utile per impedire modifiche accidentali
- Analogo alle macro in C

Esempio:

```
final int MAX_VALUE = 1000;
```

3 - Le Classi

- E' possibile definire attributi "di classe", cioè propri della classe e non dei suoi oggetti, usando la parola chiave `static`
- Vengono allocati in memoria indipendentemente dagli oggetti della classe
- Quindi è possibile accedere ad essi anche senza istanziare oggetti della classe
- L'allocazione in memoria di un attributo `static` è indipendente dalle allocazioni di memoria degli oggetti della classe che contiene l'attributo

3 - Le Classi

Esempio:

```
public class Cerchio{

  static final double PI_GRECO = 3.1415;
  static int num = 22;

   public static void main(String[] args) {
     System.out.println(Cerchio.PI_GRECO);
          // nessun oggetto e' stato istanziato!
     System.out.println(Cerchio.num);
         // oppure System.out.println(num);
     Cerchio c = new Cerchio();
     System.out.println(c.num);
  }
} // Stamperà 3.1415, 22, 22
```

- In questo esempio, `PI_GRECO` si vuole immutabile, quindi è dichiarato anche `final`

3 - Le Classi

- E' possibile definire anche *metodi* statici, sempre usando la parola chiave `static`
- Anch'essi possono essere usati senza istanziare oggetti della classe
- Sono i metodi che non si invocano su oggetti (analoghi alle funzioni in C)

3 - Le Classi

Esempio:

```
public class Serbatoio {

    private int livello;

    public void rifornimento (int j) {
         livello += j; }

    public static int stupidStaticMethod (int i) {
         return i; }
   ...

    public static void main(String[] args) {
       System.out.println(stupidStaticMethod(5)); }
}
```

- Omettendo `static` nella dichiarazione del metodo si ha un errore di compilazione: “error: non-static method <nome> cannot be referenced from a static context”

3 - Le Classi

- Se dichiariamo un metodo `static` potremo evitare di istanziare la classe che lo contiene e richiamare il metodo statico da una qualunque classe con la sintassi `nomeClasse.nomeMetodo()`

- Ad esempio, nella classe `Math` del package `java.lang` (che è caricato di default in ogni programma) c'è il metodo statico `max`, quindi possiamo scrivere:

```
int i = Math.max(3,5);
```

anche se non abbiamo istanziato nessun oggetto della classe `Math`.

3 - Le Classi

- Un altro esempio di metodo `static` è il metodo `main`:

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

- Il metodo `main` è necessariamente `static`, in quanto viene invocato prima di tutti, e quindi quando nessun oggetto è stato ancora creato

3 - Le Classi

- E' possibile raggruppare file relativi a diverse classi in un *package*
- Un package è una cartella contenente sottocartelle che contengono classi.
- Ad es. la cartella `Geometria` può contenere la sottocartella `FigurePiane` che contiene il file `Cerchio.java`
- Il file `Cerchio.java` *deve iniziare* con la dichiarazione del percorso del package:

```
package Geometria.FigurePiane;

public class Cerchio{
   ...
}
```

3 - Le Classi

- Per compilare una classe del package ci metteremo nella directory contenente il package (in questo caso la directory contenente la cartella `Geometria`) e digiteremo il percorso della classe da compilare:

  ```
  javac Geometria/FigurePiane/Cerchio.java
  ```

- Per eseguire il main della classe dovremo specificare il suo percorso nel package separato da punti:

  ```
  java Geometria.FigurePiane.Cerchio
  ```

3 - Le Classi

- Il modificatore di accesso di default (cioè quando non si mette né `public` né `private` né altro) rende oggetti e metodi accessibili all'interno di tutto il package (ma non dall'esterno)

Esempio:

```
public class Cerchio{
   final static double PI = 3.14159;
   ...
}
```

La costante statica `PI` non è stata dichiarata né `public` né `private`, e quindi di default sarà accessibile da tutti gli elementi del package `Geometria` in cui è contenuta la classe `Cerchio`, ma non dall'esterno di esso

3 - Le Classi

Modificatore	Classe	Package	Sottoclasse	Ovunque
public	Si	Si	Si	Si
protected	Si	Si	Si	No
nessuno	Si	Si	No	No
private	Si	No	No	No

- Vedremo il modificatore `protected` quando parleremo di sottoclassi ed ereditarietà