Programmazione 2
Esercitazione 6

Livio Pompianu - Simone Porcu

Contatti e consegna esercizi

- Consegna su Moodle

Oppure, SOLO se Moodle non è disponibile, indirizzate le mail ad **entrambi** indicando, all'inizio dell'oggetto: **[PR2]**

- Livio Pompianu: pompianu.livio@gmail.com
- Simone Porcu: simone.porcu@diee.unica.it

Inviare i testi degli esercizi in un archivio con nome

- `matricola_cognome_nome.zip`

Static

Static

La keyword `static` si può utilizzare sia per metodi che per attributi

Si può accedere al metodo o all'attributo senza dichiarare un oggetto della classe, utilizzando `NomeClasse.metodo()` o `NomeClasse.attributo`

I metodi statici sono concettualmente simili alle funzioni di libreria del C (es `Math.sqrt` o `Arrays.toString`)

Gli attributi statici sono utili per mantenere uno stato condiviso tra tutti gli oggetti della stessa classe. Utile per:

- contare il numero di istanze
- definire costanti (magari globali con public)

Final

Final

La keyword **`final`** può essere utilizzata su

- classi
- metodi
- attributi di una classe
- variabili

Classi Final

Una classe dichiarata final **non può essere estesa da nessuna altra classe**.

```
public final class String {
    ...
}
```

Quando è utile utilizzare classi final? Classi immutabili.

Le classi immutabili, ovvero quelle che non consentono di modificarne lo stato, non dovrebbero poter essere estese, poiché potrebbero perdere la loro natura "immutabile" (nuovi attributi mutabili, nuovi metodi, etc...)

Metodi Final

Un metodo dichiarato final **non può essere sovrascritto da una sottoclasse**.

```
public class MP3Player {

    public final void play {...}
}
```

Quando è utile utilizzare metodi final?

Quando la sovrascrittura del metodo può compromettere la consistenza dell'oggetto.

Ad esempio, i metodi invocati dal costruttore della classe dovrebbero essere dichiarati final, altrimenti una sottoclasse potrebbe ridefinirli e ottenere effetti indesiderati.

Attributi e variabili final

Un attributo o una variabile dichiarati final **possono essere inizializzato una sola volta**.

```
final String s = "hello world";

s = "mad world"; // compile error
```

Quando è utile utilizzarli?

- rappresentare lo stato di una classe immutabile
- definire costanti in una classe
- in generale per prevenire riscritture accidentali

Costanti

```
public class MyClass{

    public static final double PI   = 3.14;

}
```

Perché final?

Perché static?

private o public?

Stato immutabile

```
public class MyClass {

    private final int state;
}
```

Compila?

Perché?

Stato immutabile

```
public class MyClass {

    private final int state;
}
```

Compila?

Perché?

Le variabili final non vengono inizializzate al loro valore di default!

```
error: variable state not initialized in the default
constructor
```

Come risolvo?

Stato immutabile

```
public class MyClass {

    private final int state = 0;
}
```

```
public class MyClass {

    private final int state;

    public MyClass() { this.state = 0; }
}
```

Costanti

```
public class MyClass {

    private static final int CONSTANT;
}
```

Compila?

Perché?

Costanti

```
public class MyClass {

    private static final int CONSTANT;
}
```

Compila?

Perché?

Le variabili final non vengono inizializzate al loro valore di default!

```
error: variable CONSTANT not initialized in the default
constructor
```

Come risolvo?

Costanti

```
public class MyClass {

    private static final int CONSTANT = 0;
}
```

```
public class MyClass {

    private static final int CONSTANT;

    public MyClass() { this.CONSTANT = 0; }
}
```

```
error: cannot assign a value to final variable CONSTANT
```

Interfacce e Classi astratte

Interfacce

Nell'ingegneria del software ci sono situazioni in cui è necessario che diversi gruppi di programmatori *si accordino* su come i loro componenti software devono comunicare. Le **interfacce** definiscono questo accordo.

In Java, le interfacce sono un **tipo riferimento**, come le classi. Possono contenere

- costanti (public, static e final di default)
- dichiarazioni di metodi (senza corpo, public di default)
- metodi default
- metodi statici
- ...

Le interfacce **NON possono essere istanziate**.

Le interfacce **possono essere estese** (da altre interfacce) o **essere implementate** (da altre classi)

Una classe **può implementare una o più interfacce**.

Interfacce

```
public interface GroupedI extends Interface1, Interface2 {

    // constant declarations

    // base of natural logarithms
    double E = 2.718282;                    // public static final

    // method signatures
    void doSomething (int i, double x); // public
    int doSomethingElse(String s);      // public
}

public class MyClass implements GroupedI {
    ...
}
```

Interfacce

```
GroupedI foo = new Myclass();

Interface1 foo1 = new Myclass();

foo.doSomething(5, 0);

~~foo1.doSomething(5, 0);~~


public class AnotherClass {

    public AnotherClass(GroupedI a, Interface1 b) {

        ...
    }

}
```

Classi astratte

Una classe astratta non può essere istanziata.

Una classe astratta può definire dei **metodi astratti** omettendo il corpo del metodo. L'implementazione viene demandata alle sottoclassi:

- se astratta, può continuare a demandare l'implementazione
- se concreta, deve obbligatoriamente implementare tutti i metodi astratti

```
public abstract class Animale {

    protected String nome;

    abstract public String verso();
}
```

Classi astratte vs Interfacce

Similarità

- non possono essere istanziate
- generalmente definiscono metodi senza implementazione

Diversità

- le interfacce possono definire solo costanti pubbliche
- tutti i metodi delle interfacce sono pubblici
- una classe può estendere una sola classe, ma implementare diverse interfacce

Usare le classi astratte se

- volete condividere codice tra classi strettamente correlate
- le classi avranno uno stato comune, metodi privati comuni da condividere

Usare le interfacce se

- le classi che la implementeranno non sono necessariamente correlate
- volete specificare solo il comportamento delle classi, senza preoccuparvi dell'implementazione

Esercizi