1 Lezione del 07-10-25

1.1 Classi final

All'opposto delle classi astratte ci sono le classi `final`: queste sono classi che non possono essere estese. I metodi definiti nelle classi `final` sono necessariamente `final`, cioè non sovrascrivibili.

In Java, le classi `final` sono poche: principalmente si usano metodi `final` per impedire a chi estende la classe che contiene il metodo di modificarne il comportamento.

1.2 Interfacce

Le **interfacce** sono un paradigma simile a quello delle classi astratte, ma solitamente più usate in Java.

Dal punto di vista sintattico si definiscono come una raccolta di metodi astratti: in pratica ci permettono di definire modelli di comportamento standard che più classi possono implementare. Quando una classe implementa un interfaccia, chi sa interagire con quell'interfaccia sa automaticamente interagire con quella classe. In particolare, si possono creare *riferimenti* di tipo interfaccia, ma non *oggetti* di tipo interfaccia. I riferimenti di tipo interfaccia riferiranno a un qualsiasi oggetto che implementa l'interfaccia.

Per definire un'interfaccia si usa la parola chiave `interface` e una sintassi simile a quella delle classi. Automaticamente tutti i metodi definiti sono `abstract` e `public`.

1.2.1 Implementare interfacce

Per una classe, implementare l'interfaccia (attraverso la parola chiave `implements`) significa definire *tutti* i metodi dell'interfaccia, a meno che la classe sia astratta. Una classe può implementare tutte le interfacce che vuole: solo le `extends` sono forzatamente unarie. Si può quindi avere.

```
class A extends B implements C, D, ... { /* ... */ }
```

1.3 Tipi enumerazione

I **tipi enumerazione** sono sostanzialmente interfacce che definiscono variabili costante intere (i campi sono automaticamente `public`, `static` e `final` nelle interfacce), che si possono poi usare come identificatori costanti (e semanticamente coerenti con lo scopo del tipo enumerazione).

Si può usare la parola chiave `enum` per non dover definire esplicitamente gli interi, ad esempio:

```
enum Colori {
  ROSSO,
  VERDE,
  GIALLO
}
```

Abbiamo che gli `enum` in Java sono effettivamente definiti come una classe, la classe `Enum<E>` implementata in `java.lang.Enum`.

Sono per questo dotati di alcune funzioni di utilità, tra cui `toString()` e `name()` per ottenere i nomi dei campi dell'`enum` a tempo di esecuzione (impossibile in linguaggi come il

C++, a meno di non definire macchinose conversioni esplicite in stringhe). In particolare, la differenza fra `toString()` e `name()` è che `name()` è `final`, mentre `toString()` è ridefinibile da ogni classe `enum`.