Ingegneria del Software

Esercitazione 4

Contatto

Giovanni Quattrocchi
Professore a contratto, post-doc @ POLIMI

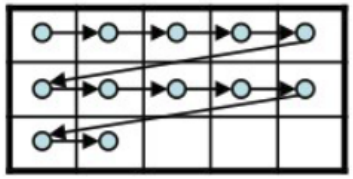
Mail: **giovanni.quattrocchi@polimi.it**

Stack + Iterator

Aggiungere un Iteratore alla classe Stack

Matrice

Scrivere un iteratore per gestire l'accesso sequenziale (Per riga) agli elementi di una matrice



```
public class Matrix{

    public int rows(){ /**/}
    public int columns(){ /**/}

    public float element(int row, int column){/**/}

}
```

Polinomi

- Scrivere un iteratore che restituisca i coefficienti del polinomio, ordinati per grado
- **Esempio**:
- per 3 + 2x deve restituire la sequenza

  3

  2

  NoSuchElementException

Iteratore di Fibonacci

$$F_1 = 1,$$
$$F_2 = 1,$$
$$F_n = F_{n-1} + F_{n-2} \text{ (per ogni n>2)}$$

```
class Father {
  int x;
  public Father(int x) {
    this.x = x;
  }
  public int m(Father f) {
    return (f.x - this.x);
  }
}
class Son extends Father {
  int y;
  public Son(int x, int y) {
     super(x); this.y = y;
  }
  public int m(Father f) {
    return 100;
  }
  public int m(Son s) {
    return super.m(s) + (s.y - this.y);
  }
}
```

```
public class MainClass {
  public static void main(String args[]) {
    Father f1, f2;    Son s1, s2;
    f1 = new Father(3);
    f2 = new Son(3,10);
    System.out.println(f1.m(f2));          /* 1 */
    System.out.println(f2.m(f1));          /* 2 */
    s1 = new Son(4,21);
    System.out.println(s1.m(f1) + s1.m(f2)); /*3*
    System.out.println(f1.m(s1) + f2.m(s1)); /*4*
    s2 = new Son(5,22);
    System.out.println(s1.m(s2));          /* 5 */
  }
}
```

- Risposta: La classe Son definisce un metodo m(Father), che effettua un overriding del metodo m(Father) in Father, e un overloading di m, con signature m(Son).
- Istruzione 1:
  - Parte statica (overloading): f1 ha tipo statico Father -> il metodo viene cercato nella classe father. f2 ha tipo statico Father -> viene cercato un metodo la cui signature è compatibile con m(Father). Il metodo viene trovato, è proprio Father.m(Father), e occorre cercare tra i metodi che ne effettuano un overriding.
  - Parte dinamica (overriding): f1 ha tipo dinamico Father -> viene scelto il metodo Father.m(Father). Stampato f2.x - f1.x, ossia 0.
- Istruzione 2:
  - Parte statica (overloading): ancora, f1 e f2 hanno tipo statico Father. Quindi viene sempre scelto Father.m(Father) (o uno che ne fa overriding).
  - Parte dinamica (overriding): stavolta f2 ha tipo dinamico Son, e quindi viene scelto il metodo Son.m(Father), che effettua overriding. Viene stampato 100.

- System.out.println(s1.m(f1) + s1.m(f2)); /*3*/
- Istruzione 3:
  - Parte statica: le due chiamate hanno come tipo statico Son.m(Father). Quindi viene scelto questo metodo, o un metodo che ne fa override...
  - Parte dinamica: ...ma nessun metodo fa override di Son.m(Father), quindi per entrambe le chiamate viene eseguito questo. Notare che, nonostante f2 abbia tipo dinamico Son, s.m(f2) NON esegue Son.m(Son)!!! Viene stampato 200.
- System.out.println(f1.m(s1) + f2.m(s1)); /*4*/
- Istruzione 4:
  - Parte statica: le due chiamate hanno come tipo statico Father.m(Son). Non esiste un metodo con questa signature, ma Father.m(Father) è compatibile. Viene scelto quindi Father.m(Father), o un metodo che ne fa override.
  - Parte dinamica: dal momento che f1 e f2 hanno diversi tipi dinamici, la prima chiamata usa il metodo Father.m(Father), la seconda usa il metodo overridden Son.m(Father).

    Il risultato è 1 + 100 = 101.

- Istruzione 5:
  - Statico è Son.m(Son), e non ci sono metodi che ne fanno overriding. All'interno, viene effettuata una chiamata di super.m(s), con s parametro il cui tipo statico è Son; super significa "della superclasse statica" - quindi di Father. Staticamente, questo significa cercare Father.m(Son), che non esiste: Però esiste Father.m(Father), che è compatibile. A runtime viene invocato questo. Quindi, super.m(s) restituisce 1, e l'istruzione 5 stampa 2 a schermo.

StringBuffer vs String

Testare la differenza tra la costruzione di stringhe con la classe String e con la classe StringBuffer.

Functional

Functional ArrayList

Estendere ArrayList con i metodi

- map
- filter
- reduce

Takewhile

Estendere ArrayList con il metodo takewhile che prende un predicato e ritorna un nuovo ArrayList contente tutti gli elementi fino al primo elemento che viola il predicato.

```
var a = MyArrayList<String>();
a.add(“abc”);
a.add(“d”);
a.add(“efg”);
a.takewhile((i)->i.length() > 1); // [“abc”]
a.takewhile((i)->i.length() > 0); // [“abc”, “d”, “efg”]
a.takewhile((i)->i.length() > 5); // []
```

makeGreet

Creare un generatore di saluti il cui funzionamento ricalchi l'esempio seguente.

```
var a = Greet.make("Hello");
a.accept("Giovanni"); // prints Hello Giovanni
a.accept("Andrea"); // prints Hello Andrea
var b = Greet.make("Ciao");
b.accept("Silvia"); // prints Ciao Silvia
```