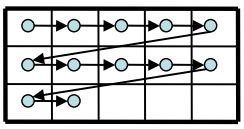
Esercitazioni di Ingegneria del Software

Iteratori

1.IterableMatrix

- Estendiamo la classe Matrix (06_astrazioni-dati) in modo che restituisca un iteratore per accedere agli elementi in maniera sequenziale, come in figura.



- La classe IterableMatrix aggiunge il metodo *iterator()* senza modificare specifica e implementazione degli altri metodi.
- La soluzione presentata richiede di modificare la visibilità dell'array *mat* nella classe Matrix. Rendendolo protected, si consente di raggiungerlo direttamente anche nelle sottoclassi

1.IterableMatrix

```
public class Matrix implements Iterable<Double> {
  /* stessa implementazione della Matrice  */
 public Iterator<Double> iterator() {
  return new RawMatrixIterator();
 }

  private class RawMatrixIterator
                 implements Iterator<Double> {
 /*inner class non statica: può accedere a metodi e campi
   (anche privati) dell'oggetto che la contiene */

  private int c_row, c_col; // riga e colonna correnti
  //@ private invariant 0<=c_row<=mat.length &&
  //@           0<=c_col< mat[0].length;

  RawMatrixIterator() { c_row = 0; c_col = 0; }
…
```


1.IterableMatrix

```
 …
  public boolean hasNext() { return c_row < mat.length; }

  public Double next() throws NoSuchElementException {
   if (!hasNext()) throw new NoSuchElementException();

   double res = mat[c_row][c_col++];

  if (c_col == mat[0].length) { c_col = 0; c_row++; }

   return res;
  }

  public void remove() throws UnsupportedOperationException {
   throw new UnsupportedOperationException();
 }
} }
```

2. Iteratore di Zenone

- Simile all'iteratore di Fibonacci, non restituisce oggetti di una collezione ma passi di lunghezza decrescente, secondo l'idea del paradosso di "Achille e la Tartaruga"

2. Iteratore di Zenone

```
import java.util.*;
public class Zenone {
 public static Iterator<Double> zenoIter(double d)
     throws IllegalArgumentException {
  return new IZ(d);
 }
 private static class IZ implements Iterator<Double> {
  private double inc; //prossimo incremento
  public IZ(double d) throws IllegalArgumentException {
   if (d <= 0) throw new IllegalArgumentException();
   inc = d/2;
  }
  public boolean hasNext() { return inc > 0; }
```

2. Iteratore di Zenone

```
…
  public Double next() throws NoSuchElementException {
   if (inc < 0) throw new NoSuchElementException();
   double ris = inc;
   inc /= 2;
   return ris; //autoboxing (sarebbe "return new Double(ris)")
  }
  public void remove() throws UnsupportedOperationException{
   throw new UnsupportedOperationException();
  }
 } //end of inner class "IZ"
}
```


2. Iteratore di Zenone - test

```
import java.util.*;
public class ZenoTest {
 public static void main(String[] args) {
  if (args.length != 1)
   System.out.println("Missing argument: distance.");
  else {
   try {
    double d = Double.parseDouble(args[0]);
    Iterator<Double> iter = Zenone.zenoIter(d);
    double percorsa = 0;
    while (iter.hasNext()) {
     percorsa += it.next();
     System.out.print(percorsa + " ");
    } catch (Exception e) { … }
 } }
//raggiungerà il Pelide Achille la tanto agognata tartaruga?
```