DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

FactorGenerator e MyComplex

Dipartimento di Ingegneria dell'Informazione
Università degli Studi di Padova
A.A. 2023/2024

Esercizio

- Realizzare una classe **FactorGenerator** che ***effettui la scomposizione di un numero intero positivo nei suoi fattori primi*** con un comportamento simile a quello della classe **Scanner** con i suoi metodi **nextInt** e **hasNextInt**

La classe deve avere

- un costruttore che:
  - riceve come unico parametro un numero intero
  - verifica che sia positivo e maggiore di uno
  - lancia **IllegalArgumentException** in caso contrario

La classe deve avere

- un metodo **non statico nextFactor**() che:
  - non riceve parametri espliciti
  - ad ogni successiva invocazione, restituisce uno dei fattori primi in cui viene scomposto il numero fornito nel costruttore
  - invocando un numero sufficiente di volte tale metodo si ottengono tutti e soli i fattori primi del numero (eventualmente ripetuti), in modo che moltiplicandoli si ottenga il numero originario
  - se il metodo viene invocato dopo aver ottenuto tutti i fattori primi, esso lancia **IllegalStateException**

La classe deve avere:

- un metodo **non statico hasMoreFactors**() che:
  - non riceve parametri espliciti
  - restituisce il valore booleano true se e solo se esistono fattori primi non ancora restituiti da nextFactor

Svolgimento

- La classe **FactorGenerator** è una classe che descrive le caratteristiche e il funzionamento di un oggetto

```
public class FactorGenerator
{
  //costruttore secondo specifiche

  //metodo nextFactor secondo specifiche

  //metodo hasMoreFactors secondo specifiche

  //variabili di esemplare
}
```

Scelta delle variabili di esemplare

- Sicuramente una caratteristica peculiare di un oggetto di tipo FactorGenerator è il numero intero positivo di cui si vuole effettuare la scomposizione in fattori primi

```
public class FactorGenerator
{
  //costruttore secondo specifiche

  //metodo nextFactor secondo specifiche

  //metodo hasMoreFactors secondo specifiche

  private int numero;
}
```

Realizzazione del costruttore

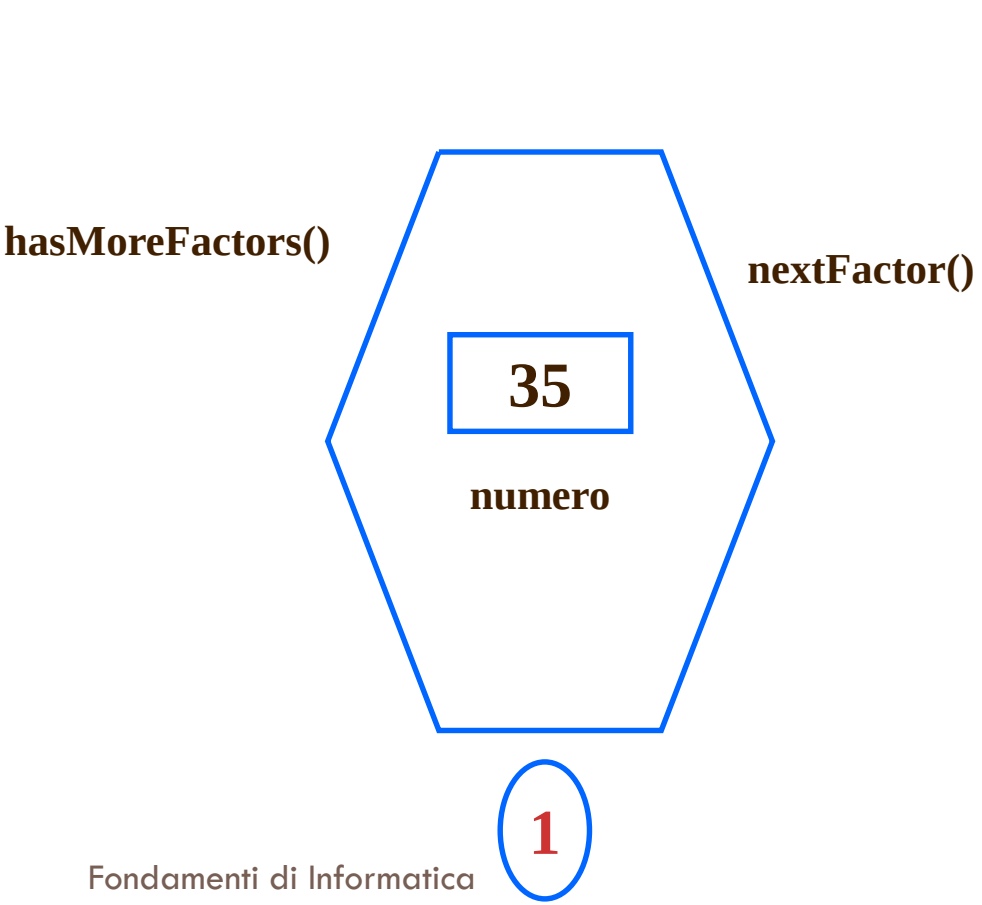
- il costruttore
  - riceve come unico parametro un numero intero
  - verifica che sia positivo e maggiore di uno
  - lanciando **IllegalArgumentException** in caso contrario

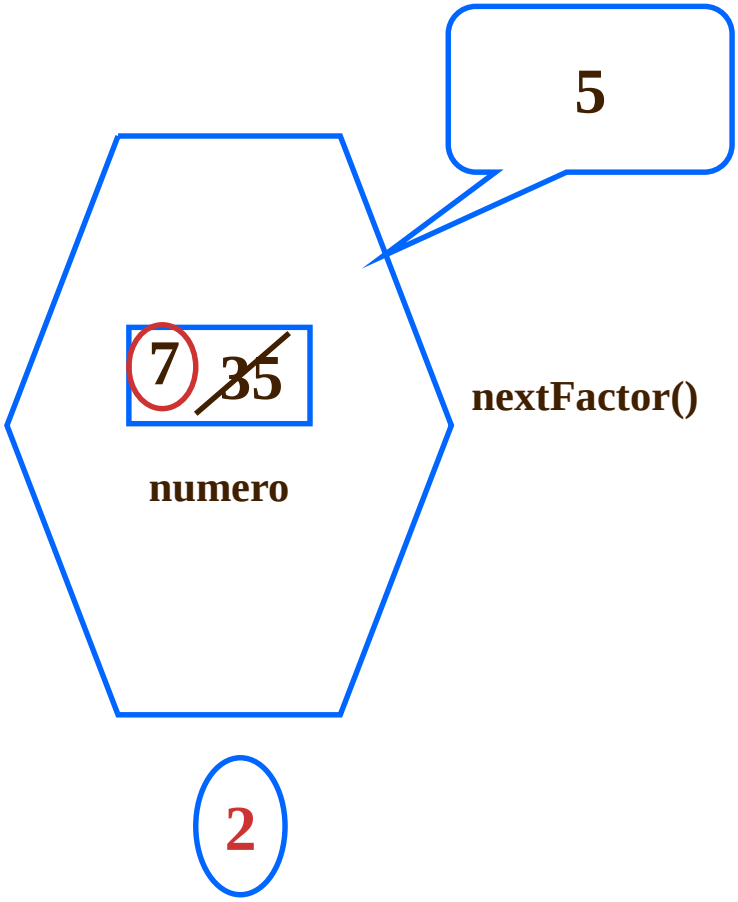
```
public class FactorGenerator
{ public FactorGenerator(int valore)
  { if(valore<=1)
      throw new IllegalArgumentException();
    numero = valore;
  }

  //metodo nextFactor secondo specifiche
  //metodo hasMoreFactors secondo specifiche
  private int numero;
}
```

Svolgimento

- Il metodo **nextFactor()** restituisce uno dei fattori primi del numero (finché ce ne sono)





Svolgimento

- Sarà sufficiente eseguire le divisioni fra il numero e tutti i numeri interi a partire da 2 fino a N, restituendo il divisore quando il resto della divisione risulta 0 e aggiornando la variabile numero con il quoziente

Realizzazione di nextFactor()

- Il metodo nextFactor deve restituire ad ogni invocazione un fattore primo del numero

```
public class FactorGenerator
{ public FactorGenerator(int valore)
  { if(valore<=1)
      throw new IllegalArgumentException();
    numero = valore;
  }
  public int nextFactor()
  { for (int i = 2; i <= numero; i++)
        if (numero % i == 0)
        { numero /= i;    //numero = numero/i;
          return i;
        }
    throw new IllegalStateException();
  }
  //metodo hasMoreFactors secondo specifiche
  private int numero;
}
```

Svolgimento

- Il metodo hasMoreFactors() restituisce true se e solo se esistono fattori primi non ancora restituiti da nextFactor

- Sarà sufficiente controllare il valore di numero:
  - se è diverso da 1=> ci sono ancora fattori primi
  - se è uguale a 1=> non ci sono più fattori primi

Realizzazione di hasMoreFactors()

```
public class FactorGenerator
{ public FactorGenerator(int valore)
  { if(valore<=1)
      throw new IllegalArgumentException();
    numero = valore;
  }
  public int nextFactor()
  { for (int i = 2; i <= numero; i++)
        if (numero % i == 0)
        { numero /= i;
          return i;
        }
    throw new IllegalArgumentException();
  }
  public boolean hasMoreFactors()
  { return (numero != 1);
  }
private int numero;
}
```

Collaudo

- Scrivere un programma **FactorGeneratorTest** che:
  - legge dallo standard input un numero intero maggiore di uno
  - crea un esemplare di FactorGenerator fornendo come parametro il numero ricevuto
  - visualizza sull'uscita standard i fattori primi del numero

FactorGeneratorTester

```
import java.util.*;
public class FactorGeneratorTester
{ public static void main(String[] args)
  {Scanner console = new Scanner(System.in);
   int n;
   do
   { System.out.println("Inserire un numero intero > 1");
     try
     { n = console.nextInt();
     }
     catch (InputMismatchException e)
     { n = 0; // un qualsiasi valore minore di 2
       console.nextLine();
     }
   } while (n < 2);
   FactorGenerator f = new FactorGenerator(n);
   System.out.println("Fattori primi del numero " + n);
   while (f.hasMoreFactors())
       System.out.println(f.nextFactor());
  }
}
```

Esercizio: realizzare la classe dei numeri complessi

I numeri complessi

- In Java non esiste un tipo di dati fondamentali per elaborare i numeri complessi
- Non è fornita neppure una classe nella java platform API per questa funzione
- Scrivere la classe **MyComplex** per rappresentare i numeri complessi

I numeri complessi

- Realizzare le seguenti operazioni
  - Somma, sottrazione, prodotto, divisione
    - Con metodi non statici (un operando e' il parametro implicito, l'altro viene passato come parametro esplicito)
    - Con metodi statici (passando entrambi gli operandi come parametri espliciti)
  - Parti reale e immaginaria, modulo, complesso coniugato, reciproco
  - Descrizione testuale a + ib

La classe deve avere

- un costruttore che:
  - riceve come parametri la parte reale e la parte immaginaria del numero complesso
- un costruttore che:
  - riceve come parametro solo la parte reale del numero complesso (parte immaginaria = 0)
- un costruttore che:
  - Non riceve parametri e setta parte reale e immaginaria a 0

La classe deve avere

- un metodo **non statico add()** che:
  - Riceve come parametro un oggetto myComplex da sommare
  - Restituisce un oggetto myComplex in cui la parte reale e quella immaginaria sono la somma della parte reale (rispettivamente immaginaria) dell'oggetto invocante e del parametro esplicito
- un metodo **statico add()** che:
  - Riceve come parametri due oggetto myComplex da sommare
  - Restituisce un oggetto myComplex somma dei due

La classe deve avere

- un metodo **non statico sub()** che:
  - Riceve come parametro un oggetto myComplex da sottrarre
  - Restituisce un oggetto myComplex in cui la parte reale e quella immaginaria sono la differenza della parte reale (rispettivamente immaginaria) dell'oggetto invocante e del parametro esplicito
- un metodo **statico sub()** che:
  - Riceve come parametri due oggetto myComplex
  - Restituisce un oggetto myComplex differenza dei due

La classe deve avere

- un metodo **non statico mult()** che:
  - Riceve come parametro un oggetto myComplex
  - Restituisce un oggetto myComplex prodotto del parametro implicito e del parametro esplicito
- un metodo non **statico div()** che:
  - Riceve come parametro un oggetto myComplex
  - Restituisce un oggetto myComplex quoziente del parametro implicito e del parametro esplicito

La classe deve avere

- un **metodo non statico rev()** che:
  - Non riceve parametri espliciti
  - Restituisce un oggetto myComplex inverso del parametro implicito
- un metodo **non statico conj()** che:
  - Non riceve parametri espliciti
  - Restituisce un oggetto myComplex coniugato del parametro implicito

La classe deve avere

- un metodo **non statico mod()** che:
  - Non riceve parametri espliciti
  - Restituisce un double pari al modulo del parametro implicito
- un metodo **non statico re()** che:
  - Non riceve parametri espliciti
  - Restituisce la parte reale del parametro implicito
- un metodo **non statico img()** che:
  - Non riceve parametri espliciti
  - Restituisce la parte immaginaria del parametro implicito

La classe deve avere

- un metodo **non statico toString()** che:
  - Non riceve parametri espliciti
  - Restituisce un oggetto stringa rappresentazione matematica del numero immaginario
    - Es: 3+i6 oppure 5-i9

La classe deve avere

- un metodo **non statico approxEquals()** che:
  - Riceve come parametro esplicito un oggetto myComplex
  - Confronta, con tolleranza, il parametro implicito e quello esplicito
    - La tolleranza deve essere verificata sia dalle parti reali che dalle parti immaginarie
    - Suggerimento: implementare ed utilizzare un metodo ausiliario per il confronto tra due double

Tester

- Scrivere una semplice classe di prova che acquisisca due numeri complessi da standard input e invii a standard output:
  - La descrizione testuale dei due numeri
  - La loro somma, sottrazione, prodotto e divisione
    - Calcolata invocando i metodi non statici
    - Calcolata invocando i metodi statici
  - Gli inversi, i moduli, le parti reali e immaginarie
- Individuare casi di prova

Svolgimento: la struttura di base

```
public class myComplex
{
  //costruttori secondo specifiche

  //metodi add() secondo specifiche
  //metodi sub() secondo specifiche
  //metodo mult() secondo specifiche
  //metodo div() secondo specifiche
  //metodo inv() secondo specifiche
  //metodo conj() secondo specifiche
  //metodo mod() secondo specifiche
  //metodo re() secondo specifiche
  //metodo img() secondo specifiche
  //metodo toString() secondo specifiche
  //metodo approxEquals() secondo specifiche

  //variabili di esemplare
}
```

Variabili di esemplare

```
/**
 * MyComplex
 * Classe per la gestione dei numeri complessi
 */

public class myComplex
{
  //costruttori secondo specifiche

  //metodi vari

  private double re;     // parte reale
  private double im;     // parte immaginaria
}
```

Costruttori

```
/** inizializza il numero complesso al valore re + i*im
    @param re parte reale
    @param im parte immaginaria
*/
public MyComplex(double real, double immag)
{   re = real;
    im = immag;
}

/** inizializza il numero complesso al valore real + i0
    @param re parte reale
*/
public MyComplex(double real)
{    re = real;
     im = 0;
```

Costruttori

```
/** inizializza il numero complesso al valore 0 + i0
*/
public MyComplex()
{    re = 0;
     im = 0;
}
```

Metodi: somma

```
/** Esegue la somma di due numeri complessi
   @param z addendo (il primo addendo e' il parametro
this)
   @return la somma this + z
*/

public MyComplex add(MyComplex z)
{ return new MyComplex(re + z.re, im + z.im);
}

/** Esegue la somma di due numeri complessi - metodo
statico
   @param z1 primo addendo
   @param z2 secondo addendo
   @return numero complesso pari alla somma this + z
*/
public static MyComplex add(MyComplex z1, MyComplex z2) {
return new MyComplex(z1.re + z2.re, z1.im + z2.im);
}
```

Metodi: sottrazione

```
/** Esegue la sottrazione di due numeri complessi
    @param z sottraendo (il minuendo e' il parametro this)
    @return numero complesso pari a this - z
*/
public MyComplex sub(MyComplex z)
{ return new MyComplex(re - z.re, im - z.im);
}

/** Esegue la sottrazione di due numeri complessi - metodo
statico
    @param z1 minuendo
    @param z2 sottraendo
    @return numero complesso pari a this - z
*/
public static MyComplex sub(MyComplex z1, MyComplex z2) {
return new MyComplex(z1.re - z2.re, z1.im - z2.im);
}
```

Metodi: moltiplicazione e divisione

```
/** Esegue la moltiplicazione di due numeri complessi
   @param z secondo fattore del prodotto (il primo fattore
      e' this)
   @return il prodotto this * z
*/
public MyComplex mult(MyComplex z)
{ return new MyComplex
         (re * z.re - im * z.im, re * z.im + im * z.re);
}
/** Esegue la divisione fra due numeri complessi @param z
divisore (il dividendo e' this) @return il quoziente this
/ z
*/
public MyComplex div(MyComplex z)
{  return mult(z.inv());
}
```

Metodi: inverso e coniugato

```
/** Calcola l'inverso rispetto al prodotto di un numero
complesso
   @return 1/z
*/
public MyComplex inv()
{ return new MyComplex
         (re / (mod() * mod()), -im / (mod() * mod()));
}


/** Calcola il coniugato di un numero complesso
   @return z^
*/
public MyComplex conj()
{  return new MyComplex(re, -im);
}
```

Metodi : modulo e metodi di accesso

```
/** Restituisce il modulo di un numero complesso
   @return |z|
*/
public double mod()
{  return Math.sqrt(re * re + im * im);
}
/** Restituisce la parte reale di un numero complesso
   @return re
*/
public double re()
{  return re;
}
/** Restituisce la parte complessa di un numero complesso
   @return im
*/
public double im()
{  return im;
}
```

Metodi: toString

```
/** restituisce una stringa che rappresenta il numero
complesso in formato matematico: x +iy.
    (Esempio: "1 +i9" o "2 -i7")
*/

public String toString()
{  final String UNITA_IMMAGINARIA = "i";
   if (equalsApprox(im , 0))
      return String.valueOf(re);
   String sign = "-";
   if (im > 0)
      sign = "+";
   if (equalsApprox(re , 0))
      return sign + UNITA_IMMAGINARIA + Math.abs(im);
   return re + " " + sign + UNITA_IMMAGINARIA+Math.abs(im);
}
```

Metodi: confronto con tolleranza

```
/** Confronta con tolleranza due numeri complessi. Il due
numeri sono considerati uguali con tolleranza se entrambe
le parti reali e immaginarie sono uguali con tolleranza.
   @return true se i numeri sono eguali con tolleranza,
   false altrimenti
*/

public boolean equalsApprox(MyComplex z)
{  return equalsApprox(re,z.re) && equalsApprox(im, z.im);
}
```

Metodo privato ausiliario

```
/* metodo privato (utilizzato sono all'interno della
classe) per confrontare con tolleranza due numeri di
tipo double. Nota: per i metodi privati non si fanno
commenti nello
stile 'javadoc'
*/
private static boolean equalsApprox(double x, double y)
{  final double EPSILON = 1E-14; // tolleranza
   return Math.abs(x-y) <= EPSILON *
Math.max(Math.abs(x), Math.abs(y));
}
```

Esercizio

- Scrivere una semplice classe di prova che acquisisca due numeri complessi da standard input e invii a standard output:
  - La descrizione testuale dei due numeri
  - La loro somma, sottrazione, prodotto e divisione
    - Calcolata invocando i metodi non statici
    - Calcolata invocando i metodi statici
  - Gli inversi, i moduli, le parti reali e immaginarie
- Individuare casi di prova
- Creare la documentazione con javadoc

Leggere da input e inizializzare i due numeri complessi

```
public class MyComplexTester{

    public static void main(String[] args){

        Scanner console = new Scanner(System.in);
        System.out.println("Inserire la parte reale e la parte
                    immaginaria del primo numero");
        double re1 = console.nextDouble();
        double im1 = console.nextDouble();
        System.out.println("Inserire la parte reale e la parte
                    immaginaria del secondo numero");
        double re2 = console.nextDouble();
        double im2 = console.nextDouble();

        MyComplex c1 = new MyComplex(re1,im1);
        MyComplex c2 = new MyComplex(re2,im2);

        … // continua nella slide successiva

    }
}
```

Visualizzare i due numeri complessi, la loro somma e la loro differenza

```
public class MyComplexTester{

 public static void main(String[] args){

  … // continua dalla slide precedente

  System.out.println("Il primo numero complesso e': "+c1.toString());
  System.out.println("Il secondo numero complesso e': "+c2.toString());

  MyComplex r1 = c1.add(c2);
  MyComplex r2 = MyComplex.add(c1,c2);
  System.out.println("Somma con metodo d'esemplare: "+r1.toString());
  System.out.println("Somma con metodo statico: "+r2.toString());

  r1 = c1.sub(c2);
  r2 = MyComplex.sub(c1,c2);
  System.out.println("Sottrazione con metodo d'esemplare: "+r1.toString());
  System.out.println("Sottrazione con metodo statico: "+r2.toString());

    … // continua nella slide successiva
 }
}
```

Visualizzare il prodotto e il quoziente

```
public class MyComplexTester{

 public static void main(String[] args){

  … // continua dalla slide precedente

  r1 = c1.mult(c2);
  r2 = MyComplex.mult(c1,c2);
  System.out.println("Prodotto con metodo d'esemplare: "+r1.toString());
  System.out.println("Ptodotto con metodo statico: "+r2.toString());

  r1 = c1.div(c2);
  r2 = MyComplex.div(c1,c2);
  System.out.println("Quoziente con metodo d'esemplare: "+r1.toString());
  System.out.println("Quoziente con metodo statico: "+r2.toString());

    … // continua nella slide successiva
 }
}
```

Visualizzare, gli inversi, i moduli, parti reali e immaginarie

```
public class MyComplexTester{

 public static void main(String[] args){

  … // continua dalla slide precedente

  //c1.inv() e' un oggetto MyComplex su cui posso invocare toString()
  System.out.println("L'inverso di "+c1.toString()+" e' "
                        +(c1.inv()).toString());

  System.out.println("Il modulo di "+c2.toString()+" e' "
                        +(c2.inv()).toString);

  System.out.println("Il modulo di "+c1.toString()+" e' "+c1.mod());
  System.out.println("Il modulo di "+c2.toString()+" e' "+c2.mod());

  System.out.println(c1.toString()+": re="+c1.re()+" img="+c1.im());
  System.out.println(c2.toString()+": re="+c2.re()+" img="+c2.im());

 }
}
```

Realizzare la documentazione

- Utilizzare il comando javadoc per produrre la documentazione e visualizzarla (aprire i file html con un browser)

Consigli utili

Consigli utili

Assegnare i nomi ai parametri

```
private double re; // variabile di esemplare
private double im; // variabile di esemplare

public MyComplex(double re, double im)
{
      re = re;
      Im = im;

}
```

???

ERRATO!

re e im sono interpretate dal compilatore come variabili di esemplare o come parametri del metodo???

Assegnare i nomi ai parametri

```
private double re; // variabile di esemplare
private double im; // variabile di esemplare

public MyComplex(double re, double im)
{
      this.re = re;
      this.im = im;

}
```

Il parametro implicito this risolve questa ambiguita'

CORRETTO!

Assegnare i nomi ai parametri

```
private double re; // variabile di esemplare
private double im; // variabile di esemplare

public MyComplex(double real, double immag)
{
      re = real;
      im = immag;

}
```

CORRETTO!

Scelta migliore; chiamiamo I parametri con nomi che, pur rimanendo significativi, siano diversi da quelli delle variabili di esemplare

Invocare un costruttore da un altro costruttore

La classe **MyComplex** ha più costruttori

```
public MyComplex(double real, double immag)
{   re = real;
    im = immag;
}

public MyComplex(double real)
{    re = real;
     im = 0;
}

public MyComplex()
{    re = 0;
     im = 0;
}
```

Invocare un costruttore da un altro costruttore

- Il primo crea un nuovo numero complesso con parte reale e immaginaria specificati dai parametri forniti
- il secondo crea un nuovo numero complesso con parte reale specificata dal parametro fornito e parte immaginaria uguale a 0 , cioè è un caso particolare del primo!
- il terzo crea un nuovo numero complesso con parte reale uguale a 0 e parte immaginaria uguale a 0, ovvero un caso particolare del secondo!

Invocare un costruttore da un altro costruttore

- Un costruttore viene utilizzato per costruire oggetti il cui stato iniziale può essere considerato come un caso particolare di quello inizializzato da un altro costruttore
- Questa situazione si verifica spesso se ci sono più costruttori

- E' possibile invocare un costruttore dal corpo di un altro costruttore!

Invocare un costruttore da un altro costruttore

- Per farlo si usa la parola chiave **this seguita da una coppia di parentesi tonde che racchiudono i parametri** da fornire all'altro costruttore che si vuole invocare (come se si invocasse un metodo)

- E' possibile **solo all'interno di un costruttore e deve essere il suo primo enunciato**

Invocare un costruttore da un altro costruttore

```
public MyComplex(double real, double immag)
{
      re = real;
      im = immag;
}

public MyComplex(double real)//inizializza il num. re+i0
{
      this(real, 0);
}

public MyComplex() // inizializza il numero 0+i0
 {
      this(0);
}
```

Invocazione del costruttore Complex(double real, double immag)

Invocazione del costruttore Complex(double real)

La chiamata al costruttore **this(...)** deve essere il primo enunciato del costruttore
Vantaggio: riutilizzo del codice!

Invocare un costruttore da un altro costruttore

- Vantaggi
  - non si devono ripetere gli enunciati di inizializzazione, che possono essere molti
  - il codice è più comprensibile!

- **Attenzione**: non confondere this(…) con this usato come riferimento al parametro implicito

Metodi di accesso e modificatori

- I metodi della classe MyComplex realizzata accedono agli stati interni degli oggetti (variabili di esemplare) senza mai modificarli: **metodi di accesso**

```
public double mod()
{   return Math.sqrt(re * re + im * im);
}
```

- Altre classi hanno metodi che modificano gli stati degli oggetti: **metodi modificatori**

```
// Classe BankAccount
public void deposit(double amount)
{    balance += amount;
}
```

Metodi di accesso e modificatori

- Raccomandazione
  - Ai metodi modificatori assegnare, generalmente, un valore di ritorno **void**
- Classi IMMUTABILI: si definiscono *immutabili* le classi che hanno solo metodi accessori
  - la classe esempio MyComplex è immutabile
  - la classe java.lang.String è immutabile
  - I riferimenti agli oggetti delle classi immutabili possono essere distribuiti e usati senza timore che venga alterata l'informazione contenuta negli oggetti stessi

Metodi predicativi

- I metodi **predicativi** restituiscono un valore booleano, ad esempio il metodo per confrontare due numeri complessi
- Stile consigliato

```
public boolean equalsApprox(MyComplex z)
{    return equalsApprox(re,z.re) && equalsApprox(im,z.im);
}
```

- Alternativa 1

```
public boolean equalsApprox(MyComplex z)
{    if (equalsApprox(re,z.re) && equalsApprox(im,z.im))
          return true;
     else
return false;
}
```

Metodi predicativi

- Alternativa 2

```
public boolean equalsApprox(MyComplex z)
 {    if (equalsApprox(re,z.re) && equalsApprox(im,z.im))
            return true;
      return false;
 }
```

Metodi di esemplare e metodi statici

- In fase di collaudo:
  - Se sto testando un metodo non statico (ovvero di esemplare) lo devo invocare utilizzando un parametro implicito
  - Se sto testando un metodo statico devo farlo precedere dal nome della classe seguito dal punto

Metodi di esemplare e metodi statici

```
public class OtherClass
{ ...
      MyComplex c1 = new MyComplex(1, 2);
      MyComplex c2 = new MyComplex(2, 3);
       ...
      // Invocazione del metodo di esemplare add()
      MyComplex s1 = c1.add(c2);
       ...

      // invocazione del metodo statico add()
       MyComplex s2 = MyComplex.add(c1, c2);


}
```