UNIVERSITÀ DELLA CALABRIA

DIPARTIMENTO DI **MATEMATICA E INFORMATICA**

Interfacce Grafiche e Programmazione ad Eventi

Carmine Dodaro

Anno Accademico 2019/2020

Schema di definizione di una classe

- La definizione di classe determina la struttura di ogni **oggetto** (istanza) della classe.
- I **campi** definiscono lo **stato** degli oggetti, ossia il loro contenuto informativo.
- I **metodi** definiscono il **comportamento** degli oggetti, ossia le funzioni che ne manipolano lo stato.
- Gli specificatori di accesso (public, private, protected) vanno inserite prima della dichiarazione del campo o del metodo. Se un campo o un metodo non ha uno specificatore di accesso, sarà visibile nella sua classe e nelle classi appartenenti allo stesso package.
- I package sono il meccanismo attraverso il quale si possono creare librerie di classi correlate. Ad esempio java.lang o java.util.

Classi e package

- I package sono insiemi di classi logicamente coordinati (come le librerie C++).
- Le classi di un package si trovano raggruppate in una sottodirectory che ha lo stesso nome del package.
- Tutte le classi facenti parte di un package iniziano con la direttiva package < nome >, dove < nome > è l'identificativo del package.
- Per utilizzare le classi di un package all'interno di altre classi, è necessario specificare direttive import nell'intestazione delle classi utilizzatrici.
- È possibile che i package contengano sotto-package.
- Ad esempio:
  - import java.util.Scanner importa la classe Scanner, del package java.util (sottopackage di java).
  - import java.io.* importa tutte le classi e gli eventuali sotto-package di java.io.
  - Se usiamo una classe che è all'interno dello stesso package non è necessario importarla.

Esempio di classe

File EsempioClasse.java

```
public class EsempioClasse {
  //Costruttore
  public EsempioClasse() {
  }

  public void metodo1() {
  }

  public int getCampo() {
      return campo;
  }

  public void setCampo(int c) {
      this.campo = c;
  }

  int campo;
  private int campoPrivato;
  protected int campoProtected;
}
```

La classe Complex

File Complex.java

```
public class Complex {
  private double re;
  private double im;

  public Complex(double r, double i) {
    re = r;
    im = i;
  }

  public void add(Complex c) {
    this.re += c.re;
    this.im += c.im;
  }
}
```

- I campi re e im contengono una **rappresentazione** di un numero complesso.
- Il metodo add somma il numero complesso dato con quello su cui il metodo è **invocato**.

Classi e oggetti

File Complex.java

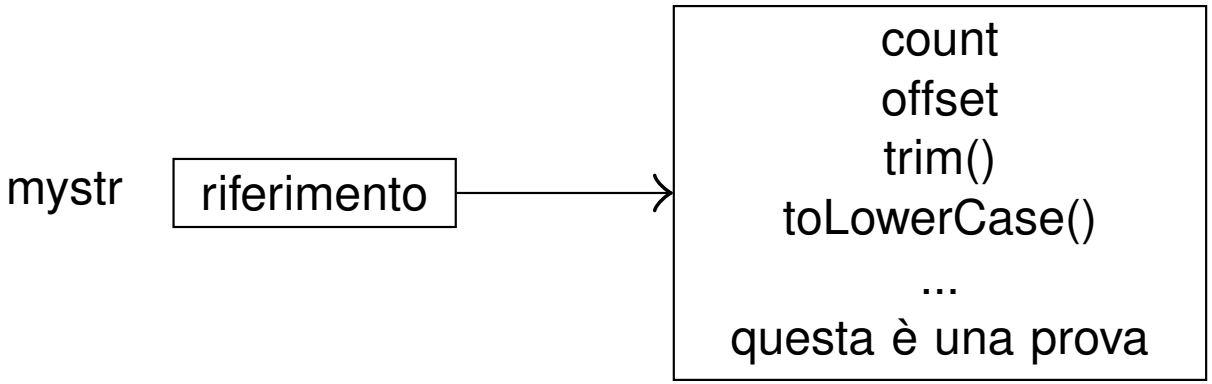
```
public class TestComplex {
  public static void main(String[] args) {
    Complex c1 = new Complex(1,1);    // c1.re = 1  c1.im = 1
    Complex c2 = new Complex(-1,-1); // c2.re = -1 c2.im = -1
    c1.add(c2);  // c1.re = 0 c1.im = 0  c2 invariato
    System.out.println(c1);
  }
}
```

- Un oggetto viene creato con il metodo **new**.
- Il metodo **add** può essere applicato ad un oggetto di tipo Complex: in questo caso c1.
- Cosa succederebbe se invocassimo c2.add(c1)?

Cosa succede quando creiamo un oggetto?

Creazione di una stringa

String mystr = “questa è una prova”;



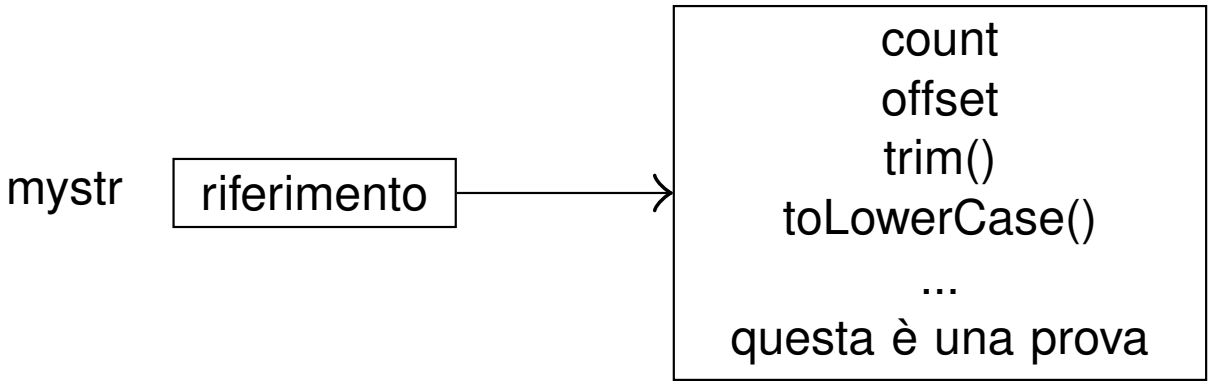
Cosa sono i riferimenti?

Concettualmente sono simili ai puntatori del C++: un riferimento rappresenta un indirizzo di memoria dove è stato allocato un oggetto ma a differenza del puntatore non è un indirizzo di memoria.

Cosa succede quando creiamo un oggetto?

Creazione di una stringa

String mystr = “questa è una prova”;



Cosa sono i riferimenti?

Concettualmente sono simili ai puntatori del C++: un riferimento rappresenta un indirizzo di memoria dove è stato allocato un oggetto ma a differenza del puntatore non è un indirizzo di memoria.
Come abbiamo già visto: non vanno deallocati!

Un esempio di riferimento

Creazione di un oggetto Complex

```
Complex c1 = new Complex();
```

- Qual è il riferimento di c1?

Un esempio di riferimento

Creazione di un oggetto Complex

```
Complex c1 = new Complex();
```

- Qual è il riferimento di c1? Complex@6d06d69c

Un esempio di riferimento

Creazione di un oggetto Complex

```
Complex c1 = new Complex();
```

- Qual è il riferimento di c1? Complex@6d06d69c

Spiegazione

A sinistra del simbolo @ c'è il tipo dell'oggetto, nel nostro caso Complex.
A destra del simbolo @ c'è un valore esadecimale che rappresenta l'indirizzo effettivo dell'oggetto.

Un esempio di riferimento

Creazione di un oggetto Complex

```
Complex c1 = new Complex();
```

- Qual è il riferimento di c1? Complex@6d06d69c

Spiegazione

A sinistra del simbolo @ c'è il tipo dell'oggetto, nel nostro caso Complex.
A destra del simbolo @ c'è un valore esadecimale che rappresenta l'indirizzo effettivo dell'oggetto.

Se provate a stampare c1, otterrete in output il riferimento di c1.

Assegnamento degli oggetti

File Esempio.java

```
public class Esempio {
  private int campo;
  public Esempio(int c) { campo = c; }
  public int getCampo() { return campo; }
  public void setCampo(int c) { campo = c; }

  public static void main(String[] args) {
    Esempio e1 = new Esempio(1);
    Esempio e2 = new Esempio(2);
    System.out.println(e1.getCampo()); //Risultato: 1
    System.out.println(e2.getCampo()); //Risultato: 2
    e2 = e1;
    System.out.println(e2.getCampo()); //Risultato: 1

    e2.setCampo(3);
    System.out.println(e2.getCampo()); //Risultato: 3
    System.out.println(e1.getCampo()); //Risultato: 3

    e1.setCampo(5);
    System.out.println(e1.getCampo()); //Risultato: 5
    System.out.println(e2.getCampo()); //Risultato: 5
  }
}
```

Passaggio degli argomenti ai metodi

File Esempio.java

```
public class Esempio {
  private int campo;
  public Esempio(int c) { campo = c; }
  public int getCampo() { return campo; }

  public void modifica(Esempio e1, Esempio e2) {
    campo = e1.campo + e2.campo;
    e1.campo=0;
    e2.campo=0;
  }

  public static void main(String[] args) {
    Esempio e1 = new Esempio(1);
    Esempio e2 = new Esempio(2);
    Esempio e3 = new Esempio(5);
    System.out.println(e3.getCampo()); //Risultato: 5

    e3.modifica(e1,e2);
    System.out.println(e1.getCampo()); //Risultato: 0
    System.out.println(e2.getCampo()); //Risultato: 0
    System.out.println(e3.getCampo()); //Risultato: 3
  }
}
```

Passaggio degli argomenti ai metodi

- Tutti gli argomenti sono passati come una copia del valore.
- Se l'argomento è un tipo base (int, float, ecc.), allora viene fatta una copia dell'elemento.
- Se l'argomento è una variabile riferimento, allora viene fatta una copia del suo riferimento.

Metodi statici

File MyMath.java e TestMyMath.java

```
public class MyMath {
  public static int gcd(int m, int n) {
    if (n == 0)
      return m;
    else
      return gcd(n, m % n);
  }
}

public class TestMyMath {
  public static void main(String[] args) {
    int result = MyMath.gcd(60,24);
    System.out.println(result);
  }
}
```

- I metodi static possono essere invocati utilizzando il nome della classe.
- Non possono utilizzare campi o invocare metodi che non siano static.
- I campi static sono condivisi da tutte le istanze degli oggetti.

Oggetti speciali: stringhe

- Gli oggetti String rappresentano sequenze di caratteri alfanumerici e sono **immutabili**.
- È possibile **accedere** ad un qualsiasi carattere con il metodo charAt, ma non è possibile **modificarlo**.
- Data una stringa a, gli indici dei caratteri vanno da 0 a a.length() - 1.
- Esistono metodi per l'**uguaglianza** e per la **concatenazione**.
- L'istruzione

  String c = a + b

  con a e b di classe String crea una **nuova** stringa con i contenuti di a concatenati a quelli di b.

Esempio di stringhe

File TestString.java

```
import java.util.Scanner;
public class TestString {
  public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    String text1 = in.nextLine();
    String text2 = in.nextLine();
    //Confronto del riferimento
    if(text1==text2)
      System.out.println("Confronto ==");

    //Confronto del valore
    if(text1.equals(text2))
      System.out.println("Confronto equals");

    //Confronta ignorando maiuscole/minuscole
    if(text1.equalsIgnoreCase(text2))
      System.out.println("Confronto equalsIgnoreCase");

    in.close();
  }
}
```

Classe StringBuffer

```
String x = "Testo"; // Un oggetto 'String'
String y = x + "!";                 // Un nuovo oggetto 'String'

StringBuffer z = new StringBuffer(x);
z.append("!");
z.setCharAt(0,'t');
```

- La classe String costruisce oggetti **immutabili**.
- Ogni operazione su oggetti String che ha come risultato String, **genera** nuovi oggetti.
- La classe StringBuffer mette a disposizione **oggetti mutabili** per la gestione di testi.

Classe StringBuilder

```
x = "Un testo di prova";

StringBuffer z = new StringBuffer(x);
z.append("!");
z.setCharAt(0,'u');

StringBuilder w = new StringBuilder(x);
w.append("!");
w.setCharAt(0,'u');
```

- La classe StringBuilder mette a disposizione **oggetti mutabili** per la gestione di testi.
- È tipicamente più efficiente di StringBuffer ma **non garantisce la sincronizzazione** nella gestione multi-thread.