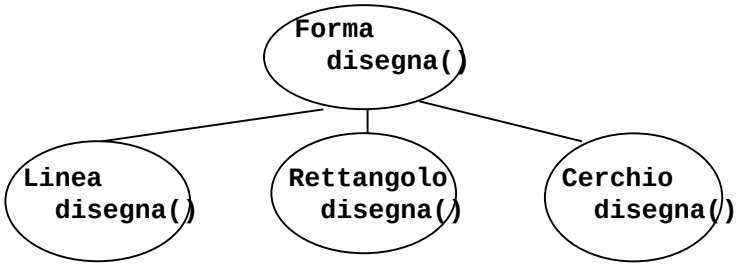
RTTI
Run-time type identification

Come determinare il tipo di un oggetto durante l'esecuzione

Ereditarietà




```
interface Forma {
   void disegna();
}

...
Forma f = new Linea();
f.disegna();
```

```
class Linea implements Forma {
    void disegna() { ......}
}
```

Si vuole eseguire una operazione **op** particolare sui cerchi

```
Forma f;
...
(Cerchio)f.op()
```

Se f non è un cerchio, viene sollevata una eccezione a run-time

```
Forma f;
...
if (f instanceof Cerchio) (Cerchio)f.op()
```

Non abusare. Altrimenti si ricade nello stile tradizionale di programmazione.

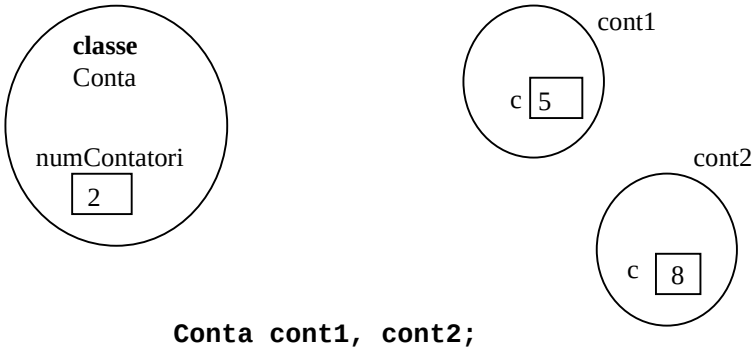
La classe **Class** (metaclassi)

La notazione **instanceof** è statica. Deve essere specificato il nome del tipo (Cerchio, Triangolo, ecc.)

In Java esiste la classe **Class**.
Per ogni classe **C** usata in un programma, c'è un unico oggetto a run-time di tipo **Class** che rappresenta quella classe.

L'oggetto di tipo **Class** che rappresenta la classe **C** viene creato dall'interprete, a partire dal file **C.class**, nel momento in cui la classe **C** è usata.
Se il file **C.class** non c'è, l'interprete lancia un'eccezione.

Classi e istanze




```
Conta cont1, cont2;
cont1 = new Conta();
cont2 = new Conta();
```



```
class Dato1 {
      static {System.out.println("carica Dato1");}
}
class Dato2 {
      static {System.out.println("carica Dato2");}
}

public class Carica {
      public static void main(String[] args) {
            System.out.println("inizia main");
            Dato1 d1 = new Dato1();
            System.out.println("continua");
            Dato2 d2 = new Dato2();
      }
}
```

Eseguendo il main, in output si ottiene

```
inizia main
carica Dato1
continua
carica Dato2
```

Object ha il metodo **getClass** che restituisce la classe dell'oggetto

```
Forma f;
...
Class c = f.getClass();
System.out.println(c.getName());
```

getName restituisce il nome della classe come stringa

Versione dinamica di **instanceof:**

```
Class c;
...
c.isInstance(f)
```


Altri metodi di **Class**

```
Class c = Class.forName("Cerchio");
```

```
Class c = Cerchio.class;
```

due modi per ottenere l'oggetto associato alla classe Cerchio

`c.getSuperclass();` restituisce la sopraclasse

`c.newInstance();` crea un nuovo oggetto della classe **c** (di tipo **Object**)

```
public class Leggi {
       public static void main(String[] args) {
        try{
              ObjectInputStream in = new ObjectInputStream(
                     new FileInputStream("serial.txt"));
              Object obj = in.readObject();
              Class c = obj.getClass();
              System.out.println(c.getName());
       } catch (IOException e) {
              System.out.println("eccezione IO");
       } catch(ClassNotFoundException e) {
              System.out.println("classe non trovata");}
       }
}
```

L'interprete, dopo aver letto l'oggetto, cerca il file **.class** della classe dell'oggetto e lo carica.

Se non lo trova lancia una **ClassNotFoundException**.

RIFLESSIONE: per ottenere a run-time informazioni su campi, metodi, costruttori, …

Il package **java.lang.reflect** contiene le classi **Field**, **Method**, **Constructor**.

La classe **Class** contiene metodi come: **getFields**, **getMethods**, **getConstructors**

La classe **Method** contiene i metodi: **getParameterTypes**, **invoke**

```
static void showMethods(Object o) {
   Class c = o.getClass();
   Method[] theMethods = c.getMethods();
   for (int i = 0; i < theMethods.length; i++) {
      String methodString = theMethods[i].getName();
      System.out.println("Name: " + methodString);
      String returnString =
        theMethods[i].getReturnType().getName();
      System.out.println("   Return Type: " + returnString);
      Class[] parameterTypes =
         theMethods[i].getParameterTypes();
      System.out.print("   Parameter Types:");
      for (int k = 0; k < parameterTypes.length; k ++) {
         String parameterString = parameterTypes[k].getName();
         System.out.print(" " + parameterString);
      }
      System.out.println();
   }
}
```



```
public String append(String firstWord, String secondWord) {
   String result = null;
   Class c = String.class;
   Class[] parameterTypes = new Class[] {String.class};
   Method concatMethod;
   Object[] arguments = new Object[] {secondWord};
   try {
       concatMethod = c.getMethod("concat", parameterTypes);
       result = (String) concatMethod.invoke
          (firstWord, arguments);
    } catch (NoSuchMethodException e) {
       System.out.println(e);
    } catch (IllegalAccessException e) {
       System.out.println(e);
    } catch (InvocationTargetException e) {
       System.out.println(e);
    }
    return result;
}
```