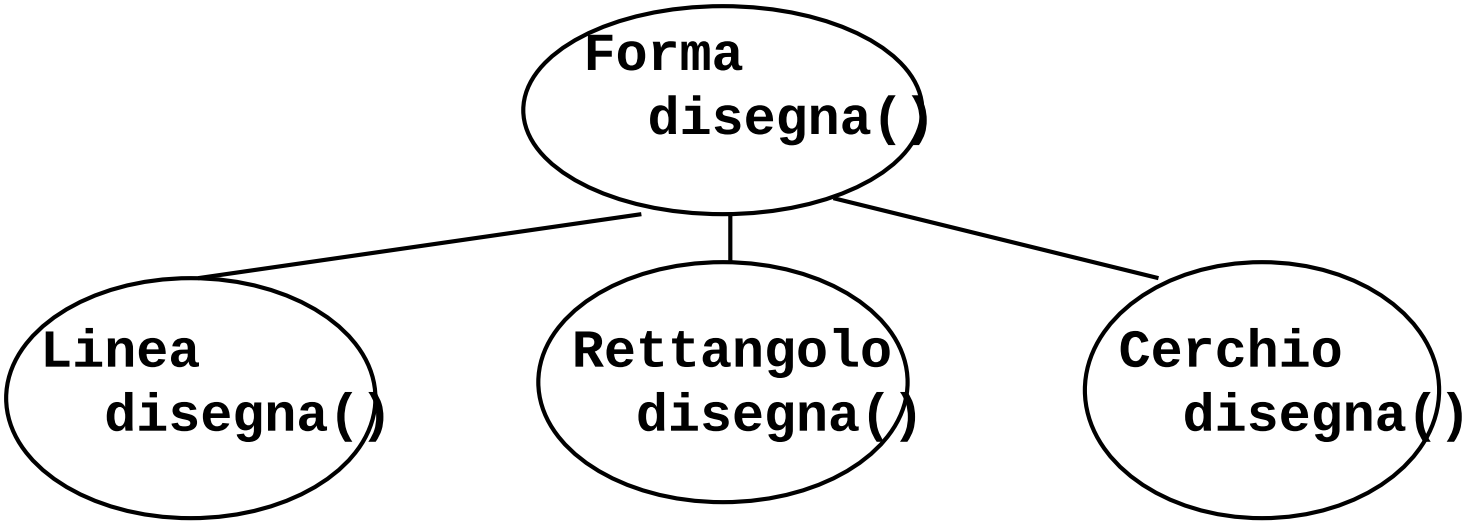
Programmazione III

La classe **Class**:

ovvero come determinare e usare il tipo di un oggetto durante l'esecuzione

RunTime Type Identification (**RTTI**)

Ereditarietà




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

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

```
...
Forma f = new Linea();
f.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
E' possibile verificare il tipo di un oggetto a run-time:

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

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

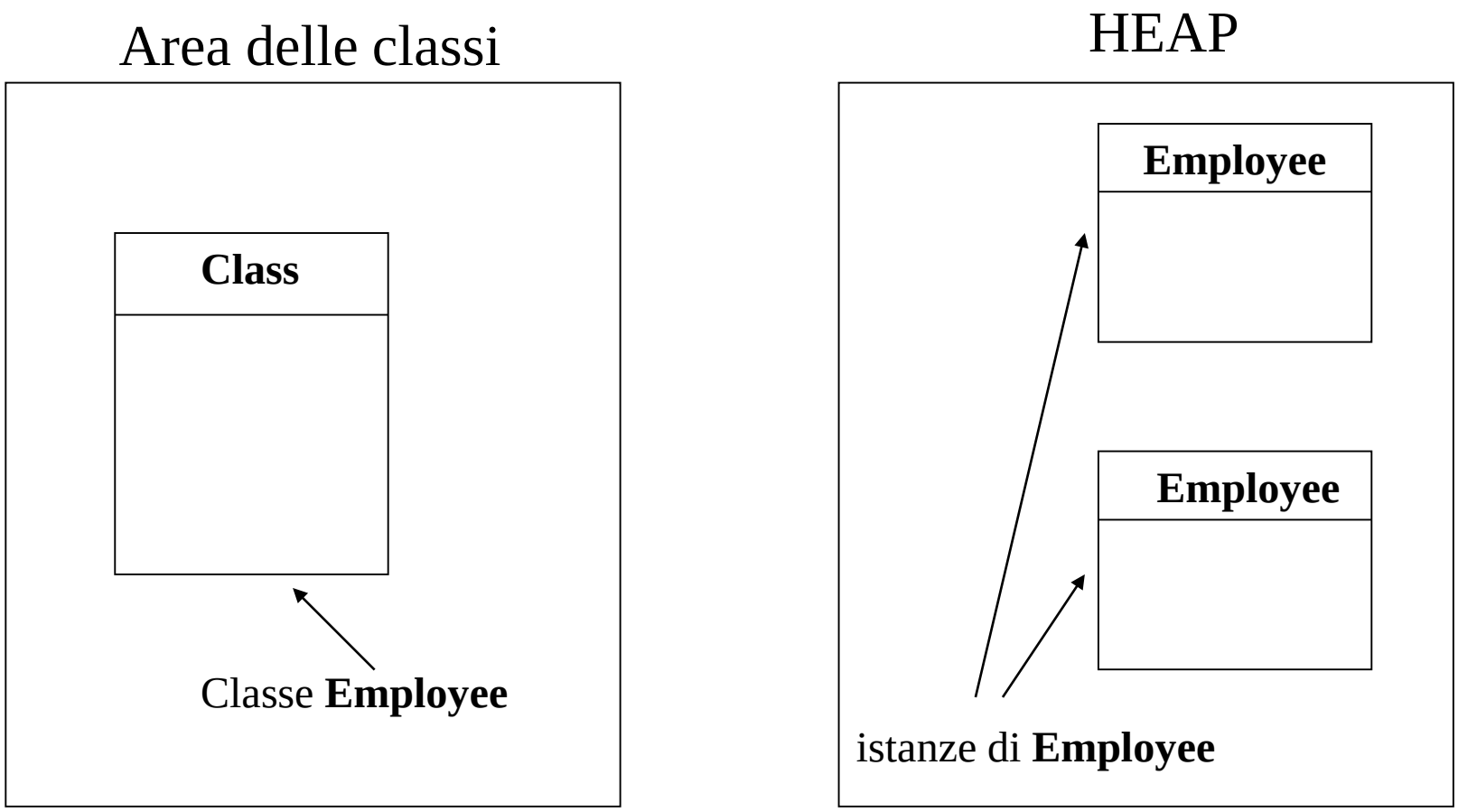
La classe **Class** (metaclassi)

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.
Esiste un oggetto Class per ogni tipo: classi, tipi enumerativi, interfacce, annotazioni, array e tipi primitivi. Serve per analizzare la classe (nome, membri, etc.).

Quando un programma è in esecuzione, il sistema runtime di Java conserva sempre la RunTime Type Identification (RTTI) di ogni oggetto.
Per ogni oggetto *o* si mantiene il riferimento all'oggetto **Class** che rappresenta la classe di *o*.

Classi e istanze



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;
o anche attraverso gli oggetti della classe:
Cerchio c = new Cerchio(); c.getClass();

`c.getSuperclass();` restituisce la sopraclasse

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

Come un oggetto **Employee** descrive le proprietà di un determinato impiegato, un oggetto **Class** descrive le proprietà di una determinata classe.

L'oggetto di tipo **Class** che rappresenta la classe **C** viene creato (**caricato**) 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.

```
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");
		try{Class c=Class.forName("Dato2");}
		  catch(ClassNotFoundException e) {}
	}
}
```

Eseguendo il main, in output si ottiene

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

Notare che, nell'esempio precedente, se la classe **Dato1** non c'è, si ottiene un errore di **compilazione** alla linea

```
Dato1 d1 = new Dato1();
```

Viceversa, se **Dato2** non c'è, si verifica un errore a runtime e viene lanciata **l'eccezione** `ClassNotFoundException` quando si esegue

```
Class c=Class.forName("Dato2");
```

RIFLESSIONE: un meccanismo molto potente fornito da Java per analizzare le funzionalità delle classi, ad esempio 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**

Ad esempio è possibile leggere il nome di una classe e **dinamicamente** estrarre le informazioni su campi e metodi della classe.

La riflessione è usata in *JavaBeans*, l'architettura a componenti di Java, per analizzare dinamicamente le proprietà di nuovi componenti.