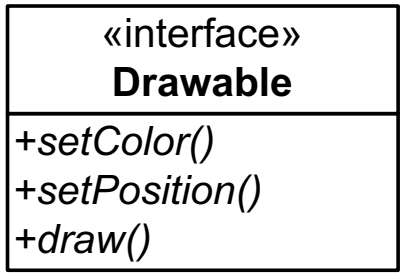
Interfacce

Interfacce

- Fortunatamente Java ci mette a disposizione uno strumento per risolvere questo problema: le interfacce
- Possiamo definire l'interfaccia Drawable in questo modo:

```
public interface Drawable
{
  public void setColor(int c);
  public void setPosition(double x, double y);
  public void draw();
}
```



- La definizione di un'interfaccia è molto simile a quella di una classe astratta, è un'elenco di metodi senza implementazione
- A differenza di una classe astratta: tutti i metodi sono privi di implementazione
- A lato la rappresentazione UML

Uso delle interfacce

- Possiamo scrivere DrawableRectangle così:

```
public class DrawableRectangle
  extends Rectangle implements Drawable
{
  private int c;
  private double x, y;
  public DrawableRectangle(double w, double h)
  { super(w,h); }
  public void setColor(int c) { this.c = c; }
  public void setPosition(double x, double y)
  { this.x = x; this.y = y; }
  public void draw()
   { System.out.println("Rettangolo, posizione"+x +
     " " +y+" colore "+c);}
}
```

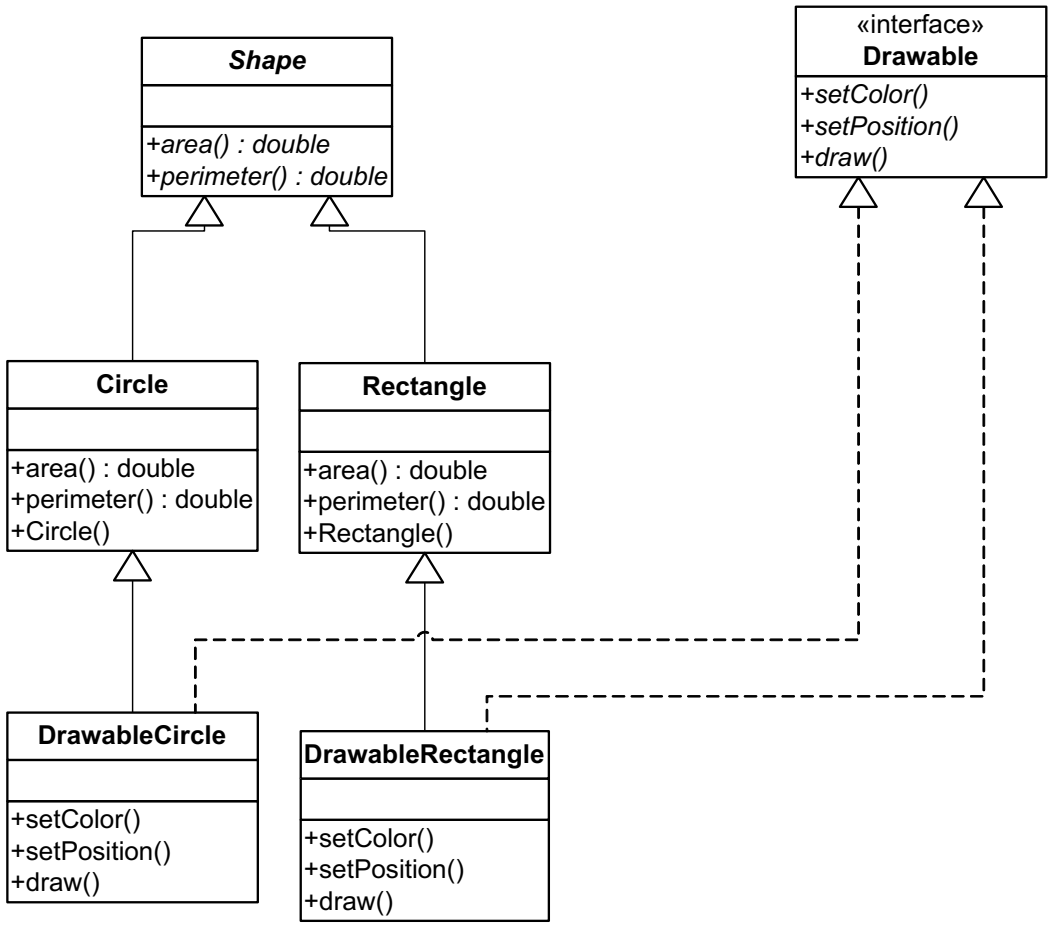
Uso delle interfacce

- In maniera del tutto simile possiamo definire DrawableCircle che sarà dichiarata come:

```
public class DrawableCircle
  extends Circle implements Drawable
{
  private int c;
  private double x, y;
  public DrawableCircle(double r)
  { super(r); }
  public void setColor(int c) { this.c = c; }
  public void setPosition(double x, double y)
  { this.x = x; this.y = y; }
  public void draw()
   { System.out.println("Cerchio, posizione"+x +
     " " +y+" colore "+c);}
}
```

Diagramma UML

- Il diagramma UML complessivo è riportato qui sotto
- Le relazioni implements sono rappresentate in modo simile a extends, ma con una riga tratteggiata



Precisazioni sulle interfacce

- Un'interfaccia è una collezione dichiarazioni di metodi, simile ad una classe astratta
- Una classe oltre a discendere da una superclasse, specificata con la parola chiave extends, può implementare una o più interfacce usando la parola chiave implements
- Attenzione: se una classe dichiara che implementa una interfaccia deve obbligatoriamente fornire un'implementazione di tutti i metodi dell'interfaccia
- Non può implementare solo alcuni metodi!
- In Java è possibile definire variabili (riferimenti) che hanno come tipo un'interfaccia:
  `Drawable d;`
- A cosa servono?

Interfacce e subtyping

- Java prevede una forma estesa di subtyping
- Nella definizione classica il subtyping ci permette di utilizzare una classe derivata al posto della classe base
- Quindi ci permette di scrivere

```
Shape s;
s = new Circle(5.7);
```

- Il subtyping in Java ci permette anche di utilizzare al posto di un'interfaccia qualunque classe la implementi
- Possiamo quindi scrivere, usando una variabile che ha come tipo l'interfaccia Drawable

```
Drawable d;
d = new DrawableCircle(6.5);
```

Esempio

- Vediamo la classe EsempioDrawable, simile a EsempioShape:

```
public class EsempioDrawable
{
  public static void main(String args[])
  {
    Drawable[] drawables = new Drawable[3];
    drawables[0] = new DrawableCircle(2.5);
    drawables[1] = new DrawableRectangle(1.2, 3.0);
    drawables[2] = new DrawableRectangle(5.5, 3.8);
    for (int i=0; i<drawables.length; i++)
    {
      drawables[i].setColor(i);
      drawables[i].setPosition(i*10.0,i*20.0);
      drawables[i].draw();
  }
}
```

- Grazie all'uso dell'interfaccia abbiamo potuto costruire un array che contiene indifferentemente istanze di classi diverse che implementano l'interfaccia Drawable e disegnarle tutte insieme con un solo ciclo for

Ancora sulle interfacce

- Un altro aspetto interessante è la possibilità di definire una classe che implementa Drawable, ma non discende da Shape
- Esempio: testo disegnato ad una data posizione:

```
public class DrawableText implements Drawable
{
  private int c;
  private double x, y;
  private String s;
  public DrawableText(String s) { this.s = s }
  public void setColor(int c) { this.c = c; }
  public void setPosition(double x, double y)
  { this.x = x; this.y = y; }
  public void draw()
   { System.out.println("Testo, posizione"+x +
     " " +y+" colore "+c);}
}
```

- Si ha quindi una forma di compatibilità e di sostituibilità tra classi indipendente dalla catena di ereditarietà

Esempio

- Potremmo riscrivere EsempioDrawable così:

```
public class EsempioDrawable
{
  public static void main(String args[])
  {
    Drawable[] drawables = new Drawable[3];
    drawables[0] = new DrawableCircle(2.5);
    drawables[1] = new DrawableRectangle(1.2, 3.0);
    drawables[2] = new DrawableText("Ciao");
    for (int i=0; i<drawables.length; i++)
    {
      drawables[i].setColor(i);
      drawables[i].setPosition(i*10.0,i*20.0);
      drawables[i].draw();
    }
  }
}
```

- Abbiamo trattato l'istanza di DrawableText in modo del tutto uniforme a DrawableRectangle e DrawableCircle

Precisazioni

- A differenza di extends dopo la clausola implements possiamo aggiungere un numero qualsiasi di nomi di interfacce

```
public class DrawableRectangle
  extends Rectangle
  implements Drawable, Sizeable, Draggable
```

- Una classe può implementare più interfacce
- Un'interfaccia è un contratto tra chi la implementa e chi la usa
- La classe che implementa un'interfaccia, essendo obbligata ad implementarne tutti i metodi, garantisce la fornitura di un servizio
- Se una classe implementa un'interfaccia si ha la garanzia che il contratto di servizio è effettivamente realizzato: ogni metodo specificato nell'interfaccia è implementato nella classe e invocabile sulle sue istanze.

Precisazioni

- A partire dalla versione 8, Java permette di definire interfacce che contengono implementazioni di default di metodi

```
interface Logger {
    default void log(String s) { System.out.println(s); }
 }
```

- Attenzione che questa feature ripropone il problema del diamond pattern inheritance!!!
- Caveat emptor…

Riprenderemo Classi astratte e interfacce con ulteriori esempi in una lezione successiva