Resum d'herència bàsica en Java

Fa que els atributs siguen visibles només des de `Figura` i les seues derivades (`Cercle` i `Cilindre`)

```
public class Figura {
    protected String tipus;
    protected String color;

    public Figura(String t, String c) {
        tipus = t; color = c;
    }

    public String getTipus() { return tipus; }

    public String getColor() { return color; }

    public void setColor(String c) { color = c; }

    public String toString() {
        return tipus + "de color " + color;
    }
}
```

`Figura` és la **classe base** o **superclasse**.
`Cercle` és una **classe derivada** o **subclasse** de `Figura`.
`Cilindre` és una classe derivada de `Cercle` i també de `Figura`.
La classe base de totes les classes Java és **`Object`**.

sobreescriptura del mètode `toString()` d'`Object`

Qualsevol mètode no privat de la classe base que es definisca de nou en la classe derivada es **sobreescriu**: el mètode té el mateix perfil que en la classe base però té un codi nou a propòsit per a la classe derivada.

indica que `Cercle` hereta de `Figura`

Fa que l'atribut siga visible només des de `Cercle` i la seua derivada (`Cilindre`)

```
public class Cercle extends Figura {
    protected double radi;

    public Cercle(String c, double r) {
        super("Cercle", c);
        radi = r;
    }

    public double getRadi() { return radi; }

    public void setRadi(double r) { radi = r; }

    public double getArea() {
        return Math.PI * radi * radi;
    }

    public double getPerimetre() {
        return 2 * Math.PI * radi;
    }

    public String toString() {
        return super.toString() + " i radi " + radi;
    }
}
```

`Cercle` hereta de `Figura` els atributs `tipus` i `color` i els mètodes:
`getTipus()`, `getColor()`, `setColor(String)` i `toString()` (que sobreescriu).

Crida al constructor de la classe base. Ha de ser la primera instrucció del constructor de la classe derivada. Si no es crida explícitament, el compilador insereix una crida al constructor per defecte (`super();`) que, si en la classe base no existeix, provoca un error de compilació.

sobreescriptura del mètode `toString()` de `Figura`

crida a `toString()` de `Figura`

indica que `Cilindre` hereta de `Cercle`

```
public class Cilindre extends Cercle {
    private double altura;

    public Cilindre(String c, double r, double a) {
        super(c, r);
        super.tipus = "Cilindre";
        altura = a;
    }

    public double getAltura() { return a; }

    public void setAltura(double a) { altura = a; }

    public double getArea() {
        return 2 * super.getArea()
            + super.getPerimetre() * altura;
    }

    public double getVolum() {
        return super.getArea() * altura;
    }

    public String toString() {
        return super.toString()
            + " i altura " + altura;
    }
}
```

`Cilindre` hereta de `Figura` els atributs `tipus` i `color` i els mètodes:
`getTipus()`, `getColor()`, `setColor(String)` i `toString()` (sobreescrit en `Cercle`).

`Cilindre` hereta de `Cercle` l'atribut `radi` i els mètodes:
`getRadi()`, `setRadi()`, `getArea()` (que sobreescriu), `getPerimetre()` i `toString()` (que sobreescriu).

sobreescriptura del mètode `getArea()` de `Cercle`

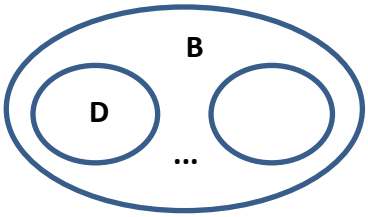
crides a `getArea()` i `getPerimetre()` de `Cercle`

sobreescriptura del mètode `toString()` de `Cercle`

crida a `toString()` de `Cercle`

Resum d'herència bàsica en Java

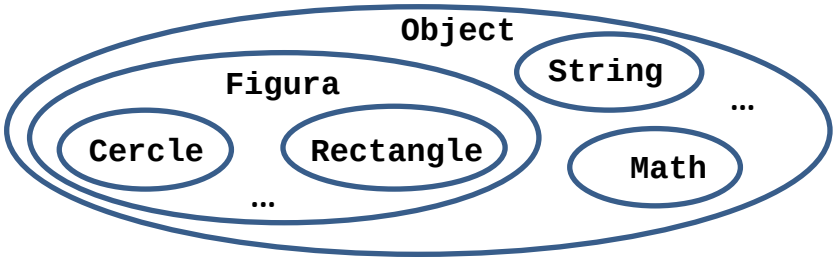
Compatibilitat de tipus en presència d'herència: dos tipus són compatibles si pertanyen a la mateixa línia de la jerarquia en sentit descendent. Siga **B** la classe base i **D** una classe derivada, açò és, **D** $\subseteq$ **B**, en general:



- Si **d** és una variable declarada de tipus **D** (o un objecte creat de tipus **D**), **d** és també de tipus **B**.
- A una variable de tipus **B**, se li pot assignar **d**.
- A **d** se li poden aplicar els mètodes de **B** (heretats per **D**).

El contrari no es compleix sempre: no es pot assegurar, en general, que una variable de tipus **B** estiga referenciant a un objecte de tipus **D**.

EXEMPLE:




```
Cercle c = new Cercle("cercle", "roig", 5.0);   ✓
double r = c.getRadi();                         ✓
Figura f = c;                                   ✓
r = f.getRadi();                                × ERROR de COMPILACIÓ
c = f;                                          × ERROR de COMPILACIÓ
```

Es pot forçar al compilador mitjançant un *càsting* si el programador preveu que l'objecte referenciat va a ser del tipus inferior. Però si s'equivoca, es produeix l'error d'execució `ClassCastException`.

EXEMPLE:

```
Cercle c = new Cercle("cercle", "roig", 5.0);   ✓
Figura f = c;                                   ✓
c = (Cercle) f;                                 ✓ Sense el càsting ERROR de COMPILACIÓ
Object o1 = f, o2 = "adeu";                     ✓
Cercle c2 = (Cercle) o1;                        ✓
Cercle c3 = (Cercle) o2;                        × ERROR d'EXECUCIÓ: ClassCastException
```

L'operador `instanceof` permet verificar si un objecte és d'una classe determinada.

EXEMPLE:

```
Cercle c = new Cercle("cercle", "roig", 5.0);   ✓
Figura f = c;                                   ✓
c = (Cercle) f;                                 ✓ Sense el càsting ERROR de COMPILACIÓ
Object o1 = f, o2 = "adeu";                     ✓
Cercle c2 = null, c3 = null;                    ✓
if (o1 instanceof Cercle) { c2 = (Cercle) o1; }  ✓  c2 != null
if (o2 instanceof Cercle) { c3 = (Cercle) o2; }  ✓  c3 == null
```