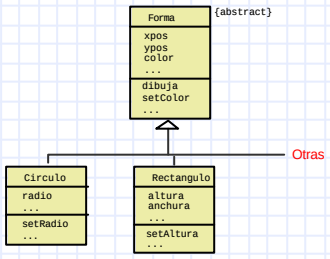
Clases abstractas e interfaces

Clases abstractas

- Clases cuya descripción es incompleta. Una clase abstracta declara métodos, pero no tiene que implementarlos.
  - No proporcionan la implementación de todos sus métodos
    - Los métodos no implementados se declaran como *abstract*
  - Una clase con un método abstracto debe declararse como clase abstracta
    - Pero una clase puede declararse como abstracta aunque no tenga ningún método abstracto

Clases abstractas



Clases abstractas

```
public abstract class Forma {
   private int xpos, ypos;
   private Color color;
   // ...
   public abstract void dibuja();
   public void setColor(Color c){ /*...*/ };
}
public class Circle extends Forma{
   private int radio;
   // ...
   public void dibuja(){ /*...*/ };
   public void setRadio(int){/*...*/};
}
public class Rectangle extends Forma{
   private int altura, anchura;
   // ...
   public void dibuja(){ /*...*/ };
   public void setAltura(int){/*...*/};
}
```

los métodos abstactos no tienen cuerpo

dibuja un círculo

dibuja un rectángulo

Clases abstractas

- Las subclases de una clase abstracta deben:
  - Sobreescribir todos los métodos abstractos de la superclase, o bien
  - Ser declaradas como clases abstractas
- Una clase abstracta no puede instanciarse
  - No se pueden crear objetos de una clase abstracta
- Una clase abstracta puede incluir variables y métodos no abstractos.
- No se pueden definir constructores abstractos o métodos estáticos abstractos.

Ejemplo clase abstracta


```
public abstract class Figura {
  int x, y;
  public void mostrarOrigen() {
      System.out.println("x= "+x+" y= "+y);}
  public abstract double area();  // No tiene implementación
  public abstract double mostrarNombre();
}
public class Triangulo extends Figura {
  protected int base, altura;
  public Triangulo (int ba, int al) { base=ba; altura=al; }
  public double area() { return base*altura/2; }
  public void mostrarNombre() { System.out.println("triangulo"); }
}
public class Cuadrado extends Figura {
  protected int lado;
  public Cuadrado (int lado) { this.lado=lado; }
  public double area() { return lado*lado; }
  public void mostrarNombre() { System.out.println("cuadrado");}}
```

Prueba clase abstracta

```
public class PruebaClaseAbstracta {
        public static void main(String args[]) {
            Figura fig;
            Triangulo tri;
            Cuadrado cua;

            fig = new Figura(); // error no se puede
                      //instanciar una clase abstracta
            tri = new Triangulo(4,3);
            tri.mostrarOrigen();
            tri.mostrarNombre();

            fig = tri;
            fig.mostrarNombre();
            System.out.println("Area triangulo: "+fig.area());

            cua = new Cuadrado(5);
            fig = cua;
            System.out.println("Area cuadrado: "+fig.area());
        }
}
```

Interfaces

- Sólo declaran comportamiento
  - Se utiliza la palabra clave *interface*
  - Por defecto todos sus métodos son públicos y abstractos
    - No implementan el comportamiento
  - Por defecto todos sus atributos son públicos, constantes y de clase
    - Por legibilidad normalmente los declaramos *static* y *final*

Interfaces

- Permite simular algunos aspectos de la herencia múltiple
  - Define un tipo de datos
  - Posibilita el enlace dinámico
- Otras clases pueden implementar un interfaz
  - Cualquier clase que implemente un interfaz debe definir todos los métodos de dicho interfaz
    - Debe proporcionar la implementación de dichos métodos
  - Si la clase no proporciona la implementación para todos los métodos del interfaz debe ser declarada como abstracta

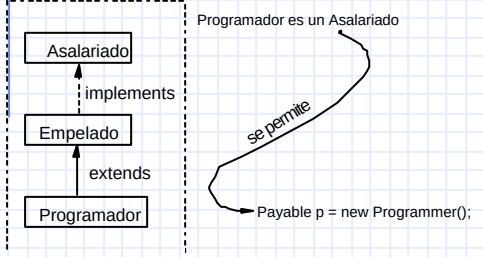
Declaración de interfaces

- Sintaxis

```
interface NombreInterfaz {
      tipo static final NOMBRECONSTANTE1 = valor;
      ……..…….
      public tipoDevuelto nombreMetodo1(listaParámetros);
      ……..…….
}
```

```
class NombreClase implements NombreInterfaz1
                    [, NombreInterfaz2 ..] {
      // declaración atributos y métodos de la clase
      ……..…….
      }
```


interfaces



Ejemplo de interfaz

```
public interface Nombrable {
     static final boolean  CIERTO = true;
     public void mostrarNombre();
}
public class Elemento implements Nombrable {
     String nombre;
     public Elemento(String nom) {
          nombre = nom; }
// obligatorio implementar método mostrarNombre
     public void mostrarNombre(){
          System.out.println("Nombre: "+nombre);
          if (CIERTO)
     System.out.println("Constante CIERTO ");
     }}
```

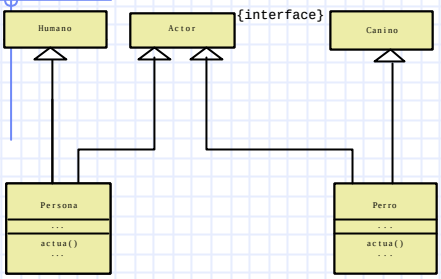
Uso del interfaz con enlace dinámico

```
public class PruebaInterfaz {
     public static void main(String args[]) {
          Elemento elem;
          Nombrable inter;

          elem = new Elemento("Luis");
          elem.mostrarNombre();

          // una referencia a interfaz puede
          //utilizarse con una instancia de
          // una clase que lo implemente
          inter = elem;
          inter.mostrarNombre();   }
}
```

Ejemplo de interfaces



Ejemplo de interfaces

```
interface Actor
{
   void actua();
}
```

sin cuerpo

```
public class Persona extends Humaon implements
Actor {
   public void actua(){ /*...*/};
   //...
}

public class Perro extends Canino implements
Actor {
   public void actua(){ /*...*/};
   //...
}
```


Extensión de interfaces

- Se puede definir un interface que especialice a otro interface mediante *extends*
  - Es similar a la herencia de clases

Extensión de interfaces

- No obstante un interface puede extender a varios interfaces a la vez
  - Aquí la herencia múltiple no plantea problemas porque no se hereda código

```
interface ElementoOrdenado extends
Comparable, Cloneable, java.io.Serializable {
// miembros y métodos propios del interfaz
//ElementoOrdenado
.....
    }
```


Résumen de interfaces

- *Las interfaces sirven para:*
  - *Declarar métodos que serán implementados por una o más clases.*
  - *Determinar la interfacede programación de un objeto, sin mostrar el cuerpo de la clase.*
  - *Capturar similitudes entre clases no relacionadas, sin forzar una relación entre ellas.*
  - *Describir objetos "tipo-función", que podrán ser utilizados como argumentos al invocar métodos sobre objetos.*

Résumen de interfaces

Tipo	Class	Abstract Class	Interface
herencia	extends (simple)	extends (simple)	implements (multiple)
nstanciable	yes	no	no
mplementa	metodos	algún método	nada
datos	Se permiten	Se permiten	no se permiten