Clases Abstractas

Dr. José Lázaro Martínez Rodríguez

Resumen

- Clases abstractas
- Ejemplos
- Ejercicio

Clases abstractas

- Cuando pensamos en un tipo de clase, suponemos que los programas crearán objetos de este tipo.

- Existen casos en lo que es conveniente declarar clases para las cuales el programador **NO** puede instanciar objetos.

- Dichas clases se denominan clases abstractas.

Clases abstractas

- Éstas se utilizan sólo como superclases en las jerarquías de herencias.
- Debido a eso se les suele llamar superclases abstractas
- Estas clases no pueden utilizarse para instanciar objetos porque, como veremos pronto, las clases abstractas están incompletas.
- Las subclases deben declarar las partes faltantes.

Clases abstractas

- El propósito de una clase abstracta es proporcionar una superclase apropiada, a partir de la cual puedan heredar otras clases.
- Las clases que pueden usarse para instanciar objetos se conocen como *clases concretas*.

- Dichas clases proporcionan implementaciones de todos los métodos que declaran.

Ejemplo: Clases abstractas

- Podríamos tener una superclase abstracta llamada **FiguraBidimensional** y derivar a partir de ellas clases concretas como **Cuadrado**, **Círculo** y **Triángulo**.

- También podríamos tener una superclase abstracta llamada **FiguraTridimensional** y derivar de ella clases concretas como **Cubo**, **Esfera** y **Cilindro**.

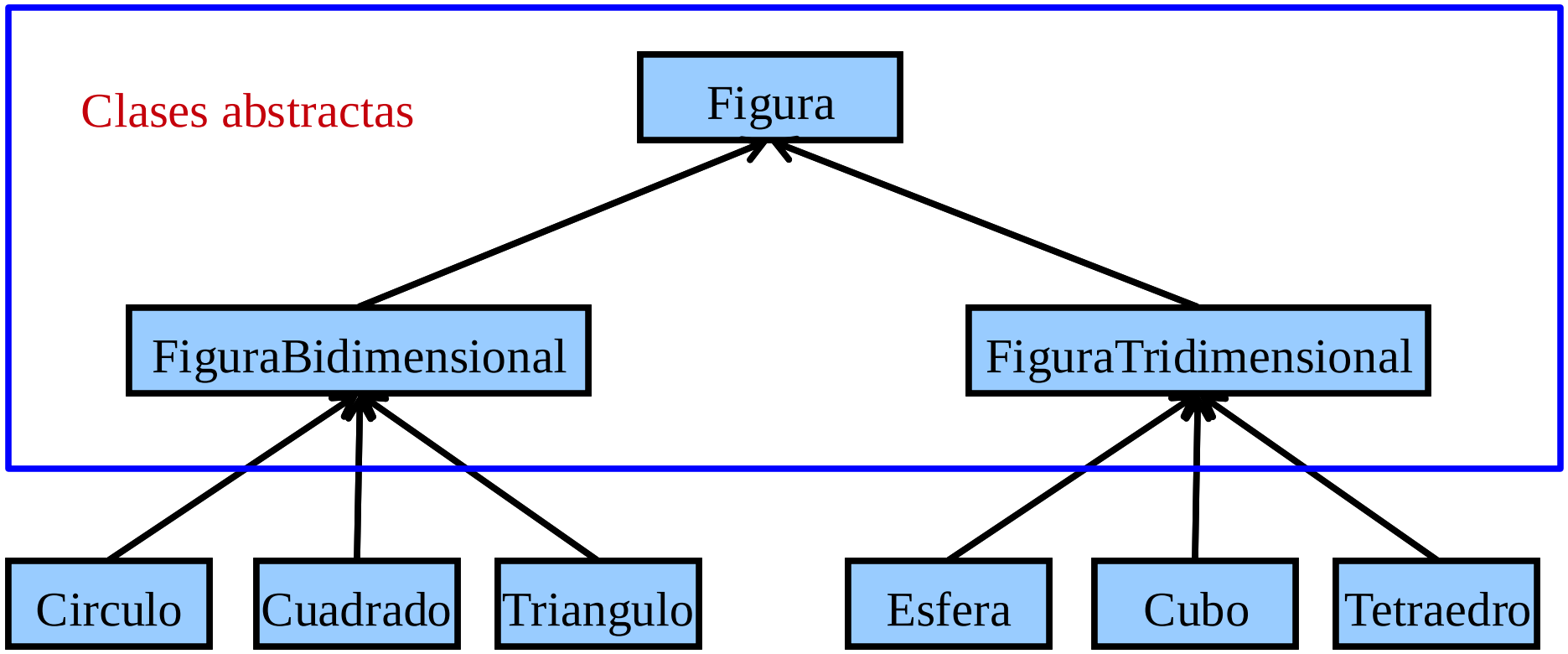
Clases abstractas

- Las clases abstractas son demasiado genéricas como para crear objetos reales; *sólo especifican lo que las subclases tienen en común*.
- Necesitamos ser más específicos antes de poder crear objetos.
- Por ejemplo, si alguien les dice "dibuje una figura", ¿qué figura dibujaría?
- Las clases concretas proporcionan los aspectos específicos que hacen que sea razonable el crear instancias de objetos

Clases abstractas

- Una jerarquía de herencia no necesita contener clases abstractas.
- Sin embargo, comúnmente se utilizan jerarquías de clases encabezadas por superclases abstractas para reducir las dependencias de código cliente en tipos de subclases específicas.
- En ocasiones las clases abstractas constituyen varios niveles de la jerarquía.

Clases abstractas



Clases Abstractas

- Clase que AL MENOS TIENE UN método abstracto
- Define una "guía" de comportamiento en al menos uno de sus métodos
- "materialización" de comportamiento según las clases derivadas
- cada clase derivada tiene la **OBLIGACIÓN** de implementar el método **O** volver a dejarlo abstracto
- Una clase que extiende una clase abstracta puede también ser abstracta
- Etiqueta **abstract**

```
abstract class FiguraGeometrica {
    . . .
    abstract void dibujar();
    . . .
}
```

```
class Circulo extends FiguraGeometrica {
    . . .
    void dibujar() {
        // codigo para dibujar Circulo
    }
}
```

Referencias a Abstractas

- NO se pueden crear objetos de clases abstractas
- Se emplea upcasting

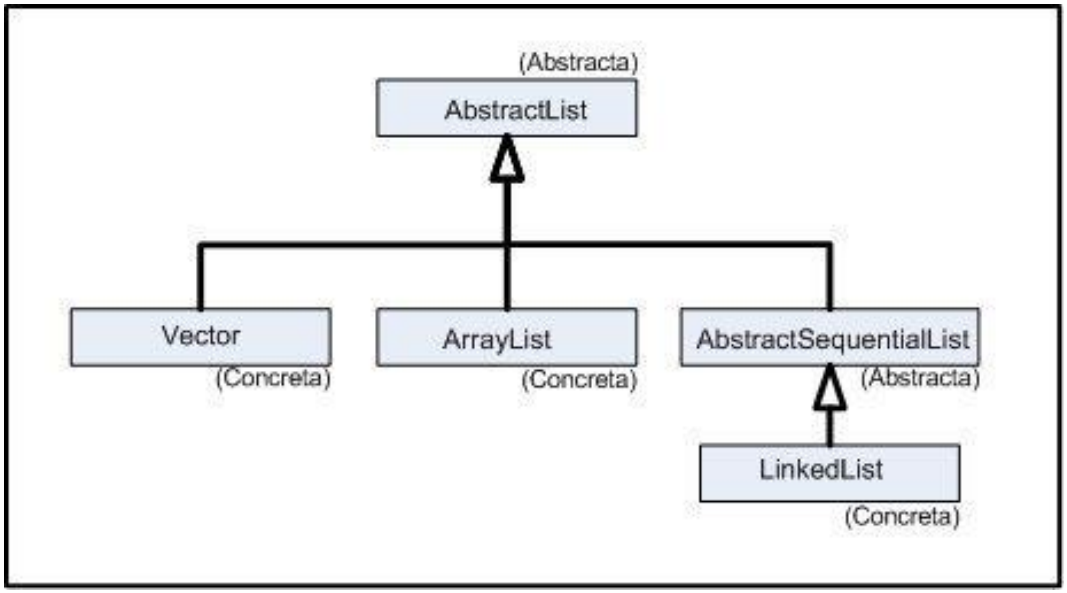
```
FiguraGeometrica figura; //OK

FiguraGeometrica figura = new FiguraGeometrica(); //ERROR

FiguraGeometrica figura = new Circulo(. . .); //OK
figura.dibujar();      //OK
```

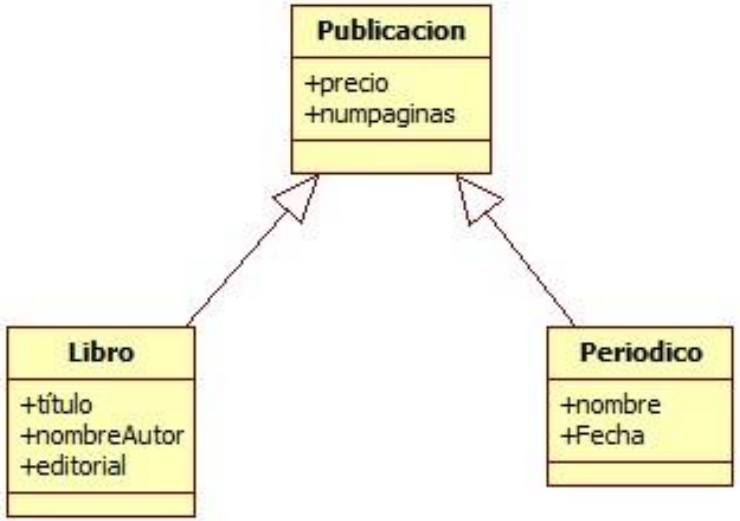
Clases abstractas en el API de Java

- Java utiliza clases abstractas en el API de la misma forma que podemos nosotros usarlas en nuestros programas. Por ejemplo, la clase AbstractList del paquete java.util es una clase abstracta con tres subclases:



Ejercicio:

- De la aplicación anterior.
- Convertir la clase Publicación a clase abstracta, con su método abstracto mostrarAtributos



Implementación

- Note que el método mostrarAtributos **NO** se implementa en la clase abstracta

```
public abstract class Publicacion {

    private double precio;
    private int numPaginas;

    public double getPrecio() {
        return precio;
    }
    public void setPrecio(double precio) {
        this.precio = precio;
    }
    public int getNumPaginas() {
        return numPaginas;
    }
    public void setNumPaginas(int numPaginas) {
        this.numPaginas = numPaginas;
    }
    public abstract void mostrarAtributos();
}
```

- Vea que Libro hereda de Publicacion por lo que debe implementar el método mostrarAtributos()

```
public class Libro extends Publicacion {

    private String titulo;
    private String nombreAutor;
    private String editorial;

    public Libro() {
    }

    public Libro(String titulo, String nombreAutor, String
editorial, double precio, int paginas) {
        this.titulo = titulo;
        this.nombreAutor = nombreAutor;
        this.editorial = editorial;
        super.setNumPaginas(paginas);
        super.setPrecio(precio);
    }

    public void mostrarAtributos() {
        System.out.println("Titulo: " + this.titulo);
        System.out.println("Autor:" + this.nombreAutor);
        System.out.println("Editorial" + this.editorial);
        System.out.println("Páginas" + super.getNumPaginas());
        System.out.println("Precio" + super.getPrecio());
    }
```

Setters y getters

```
    public String getTitulo() {
        return titulo;
    }

    public void setTitulo(String titulo) {
        this.titulo = titulo;
    }

    public String getNombreAutor() {
        return nombreAutor;
    }

    public void setNombreAutor(String nombreAutor) {
        this.nombreAutor = nombreAutor;
    }

    public String getEditorial() {
        return editorial;
    }

    public void setEditorial(String editorial) {
        this.editorial = editorial;
    }
}
```

- De forma similar se realiza la clase Periódico que implementa Publicacion, debe implementar el método mostrarAtributos() con la funcionalidad que le corresponde porque tiene atributos distintos que la clase Libro

```
public class Periodico extends Publicacion {

    private String nombre;
    private String fecha;

    public Periodico() {
    }

    public Periodico(String nombre, String fecha, double precio,
int numPaginas) {
        this.nombre = nombre;
        this.fecha = fecha;
        super.setPrecio(precio);
        super.setNumPaginas(numPaginas);
    }

    public void mostrarAtributos() {
        System.out.println("---");
        System.out.println("Nombre: " + this.nombre);
        System.out.println("Fecha:" + this.fecha);
        System.out.println("Páginas" + super.getNumPaginas());
        System.out.println("Precio" + super.getPrecio());
        System.out.println("---");
    }
    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getFecha() {
        return fecha;
    }
    public void setFecha(String fecha) {
        this.fecha = fecha;
    }
}
```

- Para la ejecución se hicieron dos instancias y se mandó llamar el método mostrarAtributos()

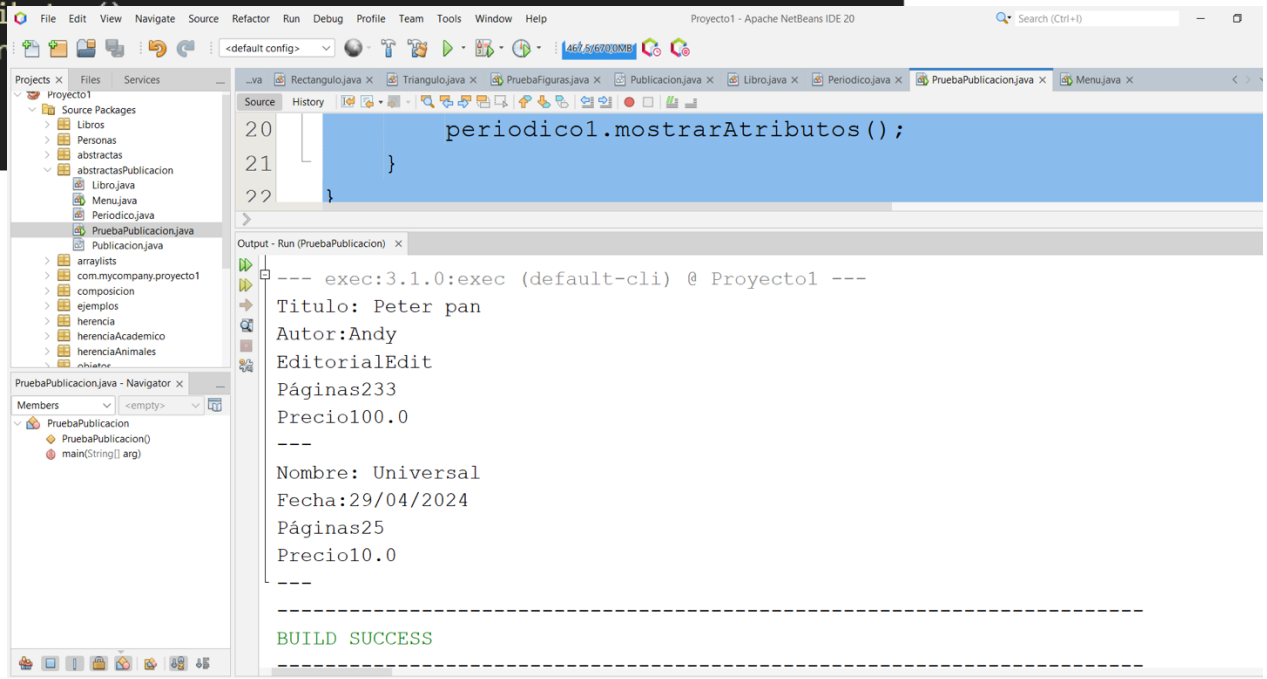
```
public class PruebaPublicacion {

    public static void main(String arg[]) {

      Libro libro1 = new Libro("Peter pan", "Andy", "Edit", 100.0, 233);

      Periodico periodico1 = new Periodico("Universal", "29/04/2024", 10.0, 25);

        libro1.mostrarAtri
        periodico1.mostrar
    }
}
```



Ejercicio:

- Declarar clase Persona. Agregue al menos 5 atributos. Aplique encapsulamiento de datos y los constructores adecuados.
- Declara una clase abstracta Legislador que herede de la clase Persona, con un atributo **distritoQueRepresenta** (tipo String) y otros atributos. Declara un método abstracto **getCamaraEnQueTrabaja**.
- Crea dos clases concretas que hereden de Legislador: la clase **Diputado** y la clase **Senador** que sobreescriban los métodos abstractos necesarios.
- Crear clase PruebaLegislador, donde se deben crear dos objetos de tipo Diputado y dos de Senador. Agregue los atributos necesarios e invoque el método sobreescrito en cada clase (desde los objetos)