

TÉCNICAS DE PROGRAMACIÓN ORIENTADA A OBJETOS

30/04/2024 y 02/05/2024

UPN.EDU.PE

UNIDAD 2: RELACIONES DE CLASES DE HERENCIA SIMPLE Y MÚLTIPLE, COMPONENTES SWING Y ACCESO A DATOS



Sesión 12

- Relaciones con Clases abstractas
- Interfaces

UNIDAD 2: RELACIONES DE CLASES DE HERENCIA SIMPLE Y MÚLTIPLE, COMPONENTES SWING Y ACCESO A DATOS



Sesión 13

- Arreglos de objetos y polimorfismo

LOGRO DE LA SESIÓN:



Al término de la sesión, el estudiante implementa ejercicios prácticos con clases abstractas e interfaces en codificación Java utilizando lógica y buenas prácticas.

Temario:

- Clases abstractas
- Interfaces

LOGRO DE LA SESIÓN:



Al término de la sesión, el estudiante implementa un programa, aplicando polimorfismo, utilizando el lenguaje Java y lo evidencia codificando con lógica y buenas prácticas.


Temario:

- Arreglo de Objetos y Polimorfismo.
- Upcasting – Downcasting.
- Resumen.

REFLEXIONA



- ¿Qué entiende por clases abstractas?
- ¿Cómo se representa las interfaces en programación?



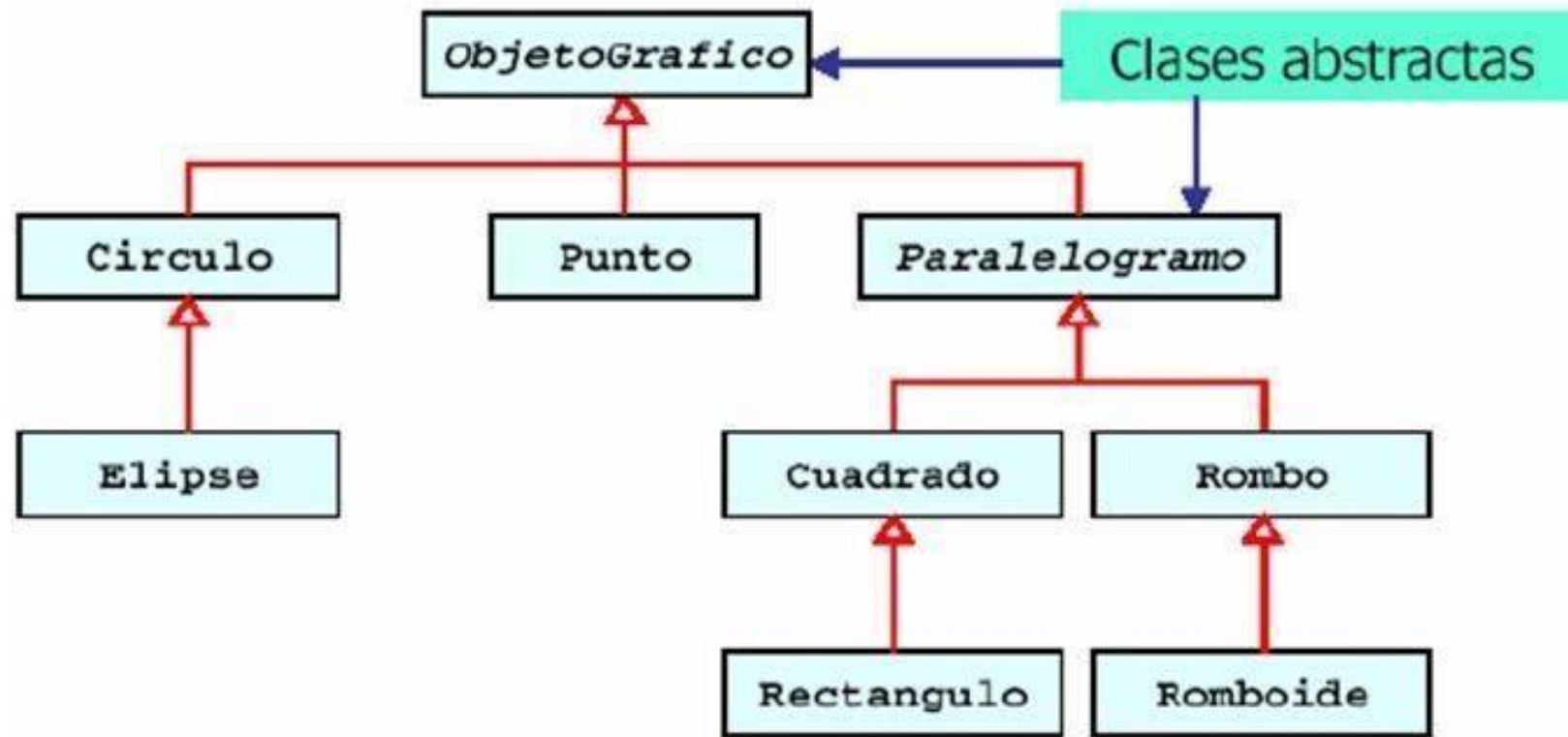
Una clase abstracta...

es una clase que ~~no~~ se puede instanciar

se usa únicamente para definir subclases

¿Cuándo es una clase abstracta?

En cuanto uno de sus métodos no tiene implementación (en Java, el método abstracto se etiqueta con la palabra reservada `abstract`).



- Clases Abstractas: ObjetoGrafico y Paralelogramo
- En el programa de dibujo sólo se van a crear **objetos gráficos concretos** de : puntos, elipses, círculos, cuadrados, rectángulos, rombos o romboides.

¿Cuándo se utilizan clases abstractas?

Cuando deseamos definir una abstracción que englobe objetos de distintos tipos y queremos hacer uso del polimorfismo.

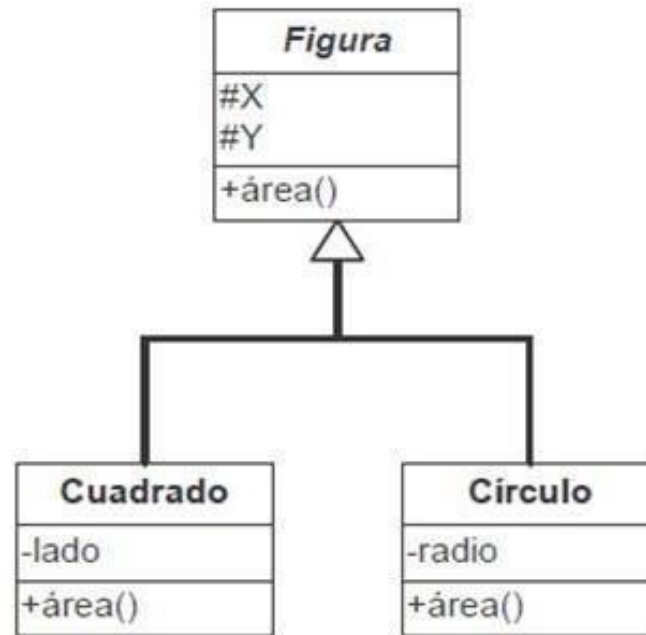



Figura es una clase abstracta (nombre en cursiva en UML) porque no tiene sentido calcular su área, pero sí la de un cuadrado o un círculo. Si una subclase de Figura no redefine `area()`, deberá declararse también como clase abstracta.

- 
- Para definir una clase como abstracta se coloca la palabra reservada **abstract** antes de **class**:

- `public abstract class ObjetoGrafico {...`

- Si en la clase abstracta se quiere obligar a que las subclases implementen un determinado método, basta declararlo como método abstracto. No tendrá cuerpo y terminará en punto y coma:

```
public abstract class ObjetoGrafico { // Abstracta
```

Métodos
Abstractos

```
    public abstract String toString();  
    public abstract void desplaza();  
    public abstract boolean esCerrada();  
    public abstract double area();  
}
```

- Para definir un **método como abstracto** se coloca la palabra reservada **abstract** antes de este
- Las subclases (no abstractas) no podrán compilarse si no implementan métodos con esos prototipos.



INTERFACES EN JAVA:

- Es una clase completamente abstracta sin implementación.
- Se declara con la palabra reservada `interface`.
- En la declaración de interfaces lo único que puede aparecer son las declaraciones de métodos (sin implementación) y/o definiciones constantes.
- Java indica que una clase implementa una interfaz usando la palabra reservada `implements`.
- La clase debe implementar todos los métodos definidos en la interfaz.



INTERFACES EN JAVA:

- Una interfaz define un tipo de Datos
- Contenido de una Interfaz.
- Nombre de atributos y visibilidad.
- Eventuales otras interfaces extendidas
- Declaración de métodos
- Constantes (Declaradas como static final)
- Una interfaz no provee:
- Variables de instancia o de clase
- Implementación de métodos
- Son útiles cuando una clase debe usar objetos de distintas clases, peroque operan de la misma forma.

DECLARACIÓN:

```
interface nombre_de_interfaz {  
    Tipo_de_retorno Nombre_del_metodo1 (lista de parametros);  
    Tipo_de_retorno Nombre_del_metodo2 (lista de parametros); ...  
    tipo variable_final1 = valor_constante; tipo variable_final2 =  
    valor_constante;  
    ...  
}
```


interface Animal{
 public void comer();
 public int respirar();
}



class Perro **implements** Animal{

 public void comer(){
 //definimos cómo come el perro
 }

 public int respirar(){
 //definimos cómo respira el perro
 }

 public String ladrar(){
 //definimos un método exclusivo del
 perro
 }

}

EJEMPLO DE INTERFAZ:

```
public interface Despertable {  
    public static final int DORMIDO = 1;  
    public static final int DESPIERTO = 2;  
    public void despierta();  
}
```

La interfaz
Despertable



```
public class Persona implements Despertable{  
    int estado = DESPIERTO;  
    public void dormir() {  
        estado = Despertable.DORMIDO;  
    }  
    public void despierta(){  
        estado = DESPIERTO;  
    }  
    ...  
}
```

Una clase que
implementa la
interfaz
Despertable

LA HERENCIA MÚLTIPLE ES CONOCIDA COMO INTERFACES

```
interface Agua{  
    public void lavar();  
}  
  
interface Fuego{  
    public void encender();  
}  
  
public class Elementos implements Agua, Fuego {  
    .....  
}
```




USO DE INTERFACES:

- ❖ Una clase puede implementar múltiples interfaces:

```
public class Anfibio implements Terrestre, Acuático { ... }
```

- ❖ Una interfaz puede extender otras interfaces.

```
public interface Anfibio extends Terrestre, Acuático { ... }
```

- ❖ Algunos enuncian que el uso de interfaces representa una forma de enfrentar el problema de la "herencia múltiple" en Java.

EJEMPLO:



❖ El Aerosub ("Viaje al fondo del mar")

```
public interface Aéreo {  
    public void despegar();  
    public void acuatizar();  
}
```

```
public interface Acuático {  
    public void emerger();  
    public void sumergirse();  
}
```

```
public class Aerosub implements Aéreo, Acuático{  
    public void despegar(){...}  
    public void acuatizar(){...}  
    public void emerger(){...}  
    public void sumergirse(){...}  
}
```



HERENCIA MÚLTIPLE DE INTERFACES:

Una clase puede implementar varios interfaces simultáneamente, pese a que, en Java, una clase sólo puede heredar de otra clase (herencia simple de implementación, múltiple de interfaces).

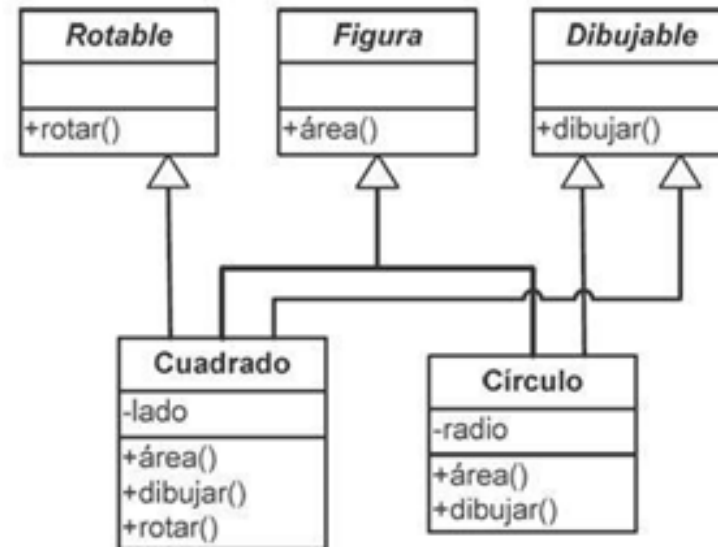
```
public abstract class Figura
{
    public abstract double area ();
}

public interface Dibujable
{
    public void dibujar ();
}

public interface Rotable
{
    public void rotar (double grados);
}
```

```
public class Circulo extends Figura
                        implements Dibujable
...

public class Cuadrado extends Figura
                        implements Dibujable, Rotable
...
```





```
public abstract class Figura
{
    public abstract double area ();
}

public interface Dibujable
{
    public void dibujar ();
}

public interface Rotable
{
    public void rotar (double grados);
}
```

```
public class Circulo extends Figura
    implements Dibujable
...

public class Cuadrado extends Figura
    implements Dibujable, Rotable
...
```

LOGRO DE LA SESIÓN:



Al término de la sesión, el estudiante implementa un programa, aplicando polimorfismo, utilizando el lenguaje Java y lo evidencia codificando con lógica y buenas prácticas.

Temario:

- Arreglo de Objetos y Polimorfismo.
- Upcasting – Downcasting.
- Resumen.



POLIMORFISMO (EN GRIEGO SIGNIFICA MUCHAS FORMAS):

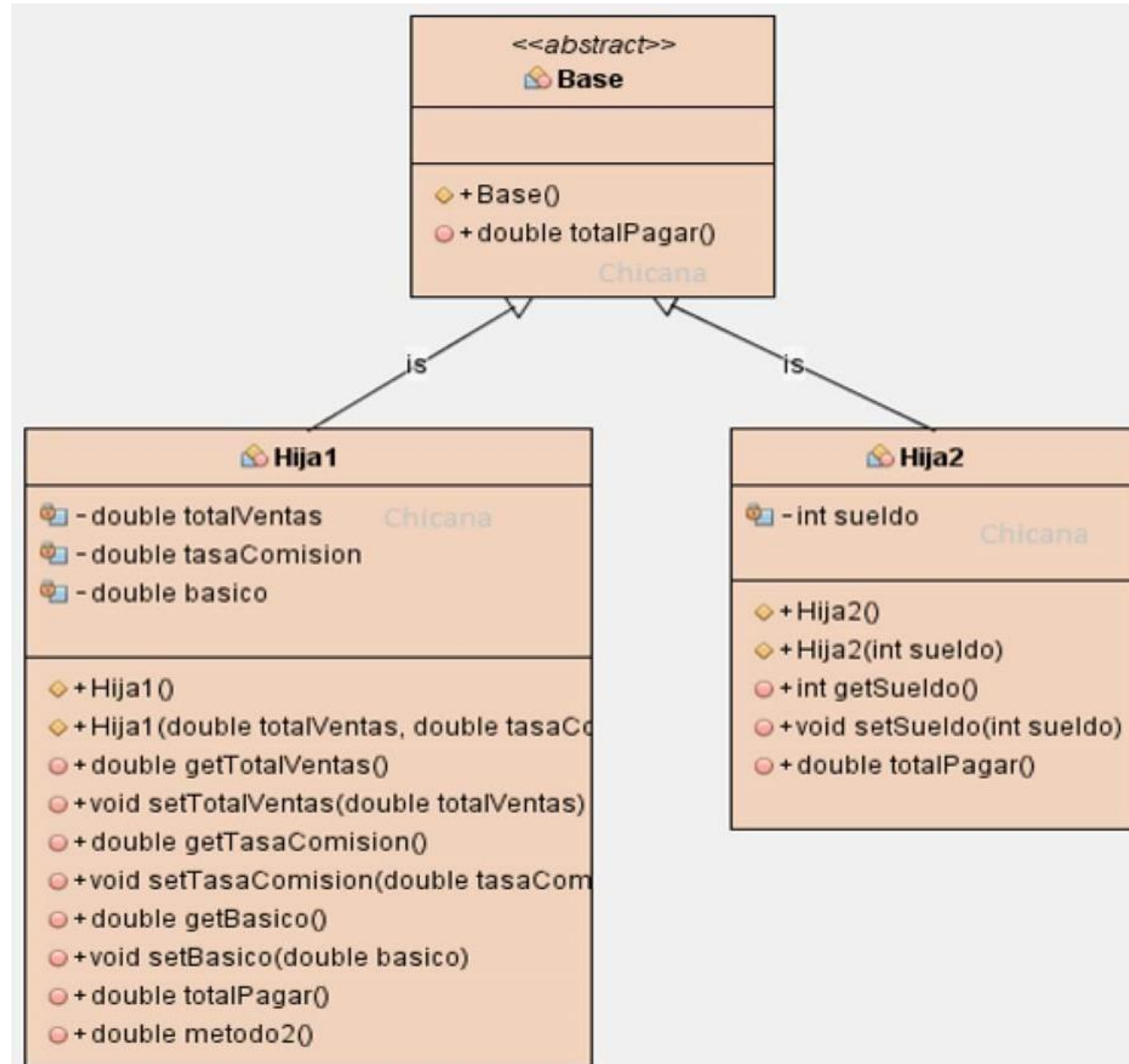
- Programar en forma general en lugar de específica.
- Enviar un mismo mensaje a una variedad de objetos y éstos responden de diferentes formas (tienen diferentes formas de resultados).
- Para aplicar polimorfismo, los objetos deben de pertenecer a una misma jerarquía de herencia.



ARREGLO DE OBJETOS:

- Colección de elementos de datos relacionados.
- Se pueden incorporar nuevos tipos de objetos a una colección de elementos, que pueden responder a la llamada de métodos existentes, sin necesidad de modificar el sistema base; el código del cliente deberá editarse para agregar los nuevos tipos de objetos.

CLASES Y SUBCLASES:



UPCASTING – DOWNCASTING:

```
/*
```

```
* @author Jorge Chicana Aspajo
```

```
*/
```

Palabra Clave:
arreglo de objetos[]

```
Base[] arr = {new Hija1(1000, 0.10, 800), new Hija2(2000)};
```

```
Hija1 h1 = (Hija1) arr[0];
```

```
Hija2 h2 = (Hija2) arr[1];
```

```
System.out.println(""+h1.totalPagar());
```

```
System.out.println(""+h2.totalPagar());
```

Downcasting

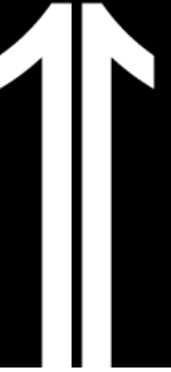
```
Base b = h1;
```

```
System.out.println(""+b.totalPagar());
```

¿Polimorfismo?

Upcasting

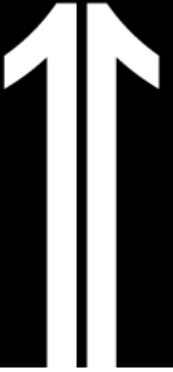
RESUMEN: ACTIVIDAD



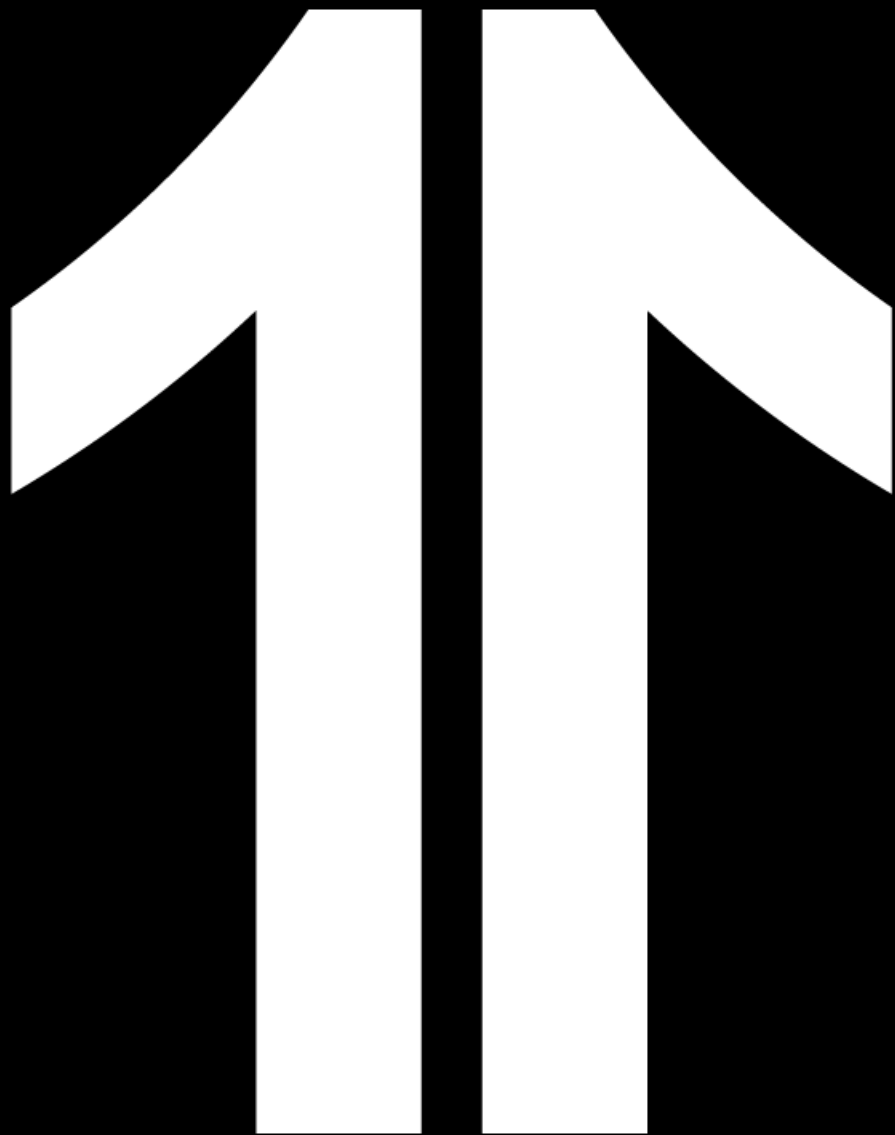
Escriba algunos aspectos importantes sobre el tema:

- _____
- _____
- _____
- _____

REFERENCIAS DIGITALES



- <https://javadesdecero.es/poo/herencia-java-tipos-ejemplos/>
- https://www.mundojava.net/la-herencia-en-java.html?Pg=java_inicial_4_4_6.html
- https://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=653:ejemplo-de-herencia-en-java-uso-de-palabras-clave-extends-y-super-constructores-con-herencia-cu00686b&catid=68&Itemid=188
- http://profesores.fi-b.unam.mx/carlos/java/java_basico3_4.html



GRACIAS

