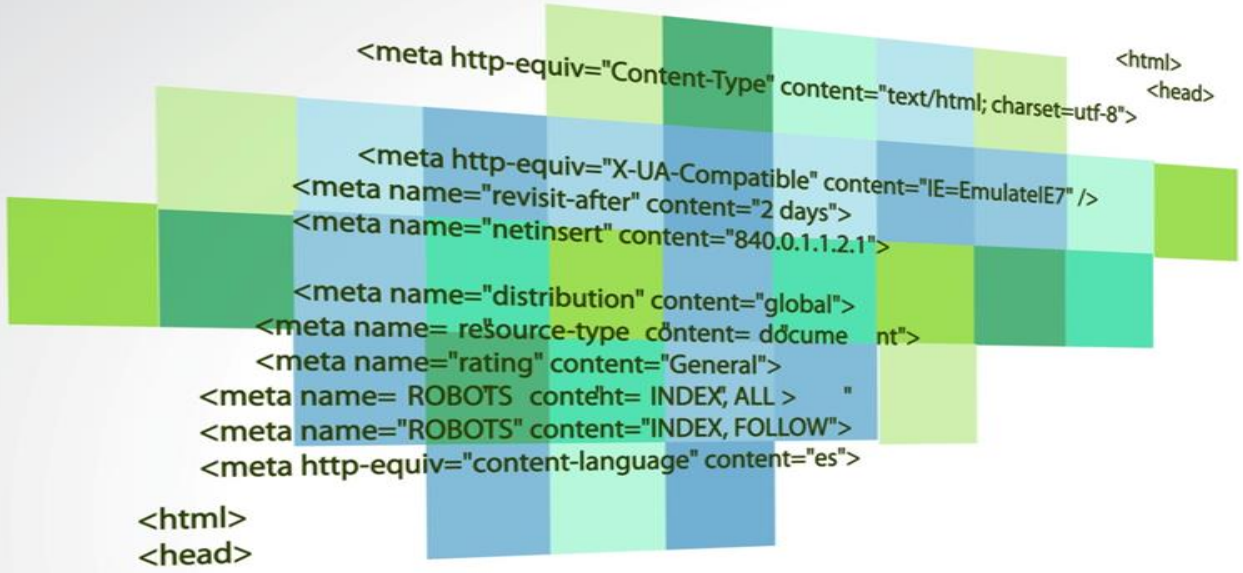


PROGRAMACIÓN ORIENTADA A OBJETOS

Herencia y Polimorfismo

UNIDAD 3

LOGRO DE LA UNIDAD 3

- Al finalizar la unidad el alumno construye programas aplicando los principios de herencia y polimorfismo.

Agenda

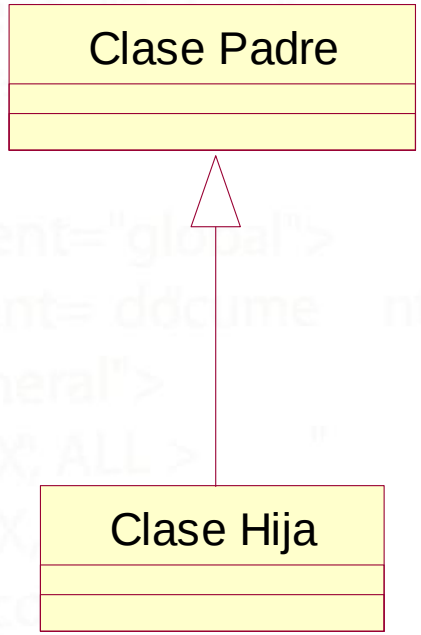
1. Herencia
2. Polimorfismo
3. Interfaces

1. Herencia

- La idea básica es poder crear nuevas clases basadas en clases ya existentes.
- Cuando heredamos de una clase existente, estamos re-usando ó extendiendo código (métodos y propiedades).
- Podemos agregar métodos y variables para adaptar la clase nueva.

1.1 Herencia en Java

- Java permite definir una clase como subclase de una clase padre (superclase).




```
class clase_hija extends clase_padre
{
//cuerpo de la clase
.....
}
```

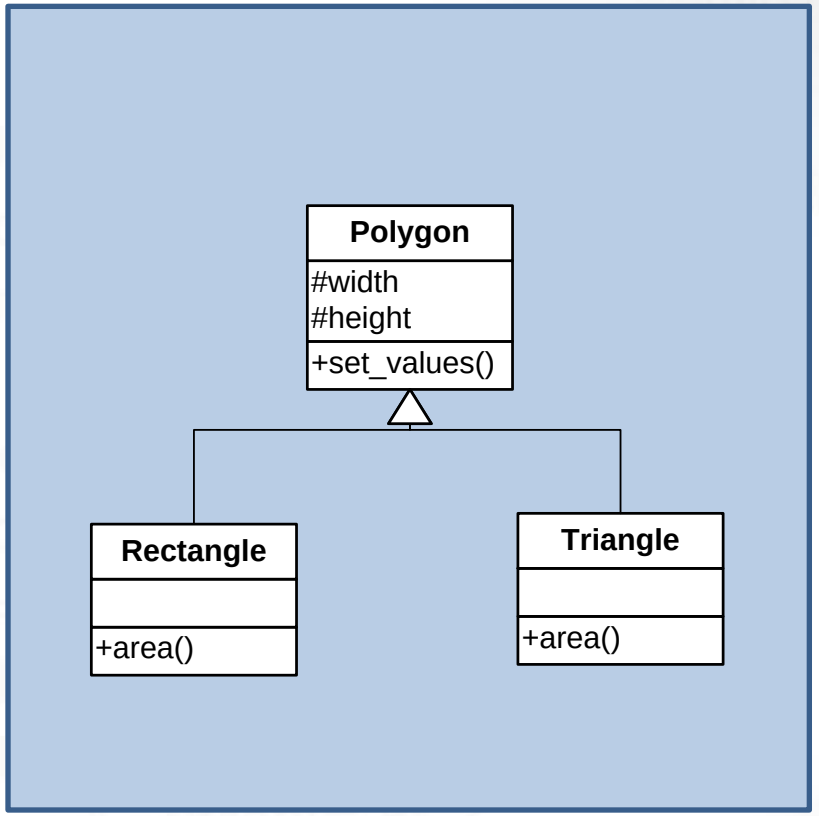
Ejemplo de Herencia

```
class Polygon {
  protected int width, height;
  public void Polygon (int a, int b) {
   width=a; height=b;
  }
}

class Rectangle extends Polygon {
 public int area() {
   return (width * height);
 }
}

class Triangle extends Polygon {
 public int area() {
   return (width * height / 2);
}

class Demo{
 public static void main(String[] args) {
 Rectangle rect;
 Triangle trgl;
 rect = new Rectangle(4,5);
 trgl = new Triangle(5,6);
 System.out.println("area rect." + rect.area()
 System.out.println("area triang:" + trgl.area);
 }
}
```



1.2 Redefinir funciones miembros de la clase padre

```
class Persona {
    private String nombre;
    private int edad;
    ..................
    public String toString() { return nombre + edad; }
    public void setEdad(int e) { edad = e; }
}
class Alumno extends Persona {
    private int curso;
    private String nivelAcademico;
    ..................
    public String toString() {
        return super.toString() + curso + nivelAcademico;
    }
    public void setCurso(int c) { curso = c; }
}
```

Observa el método toString() como se redefine o se sobreescribe.

2. Polimorfismo

- Una misma llamada ejecuta distintos comportamientos dependiendo de la clase a la que pertenezca el objeto al que se aplica el método.
- Supongamos que declaramos: *Persona p;*
- Podría suceder que durante la ej. del programa, p referencie a un profesor o a un alumno en distintos momentos, y
- Entonces:
  - Si p referencia a un alumno, con p.toString(), se ejecuta el toString de la clase Alumno.
  - Si p referencia a un profesor, con p.toString(), se ejecuta el toString de la clase Profesor.
- **Enlace dinámico**: Se decide en **tiempo de ejecución** qué método se ejecuta.

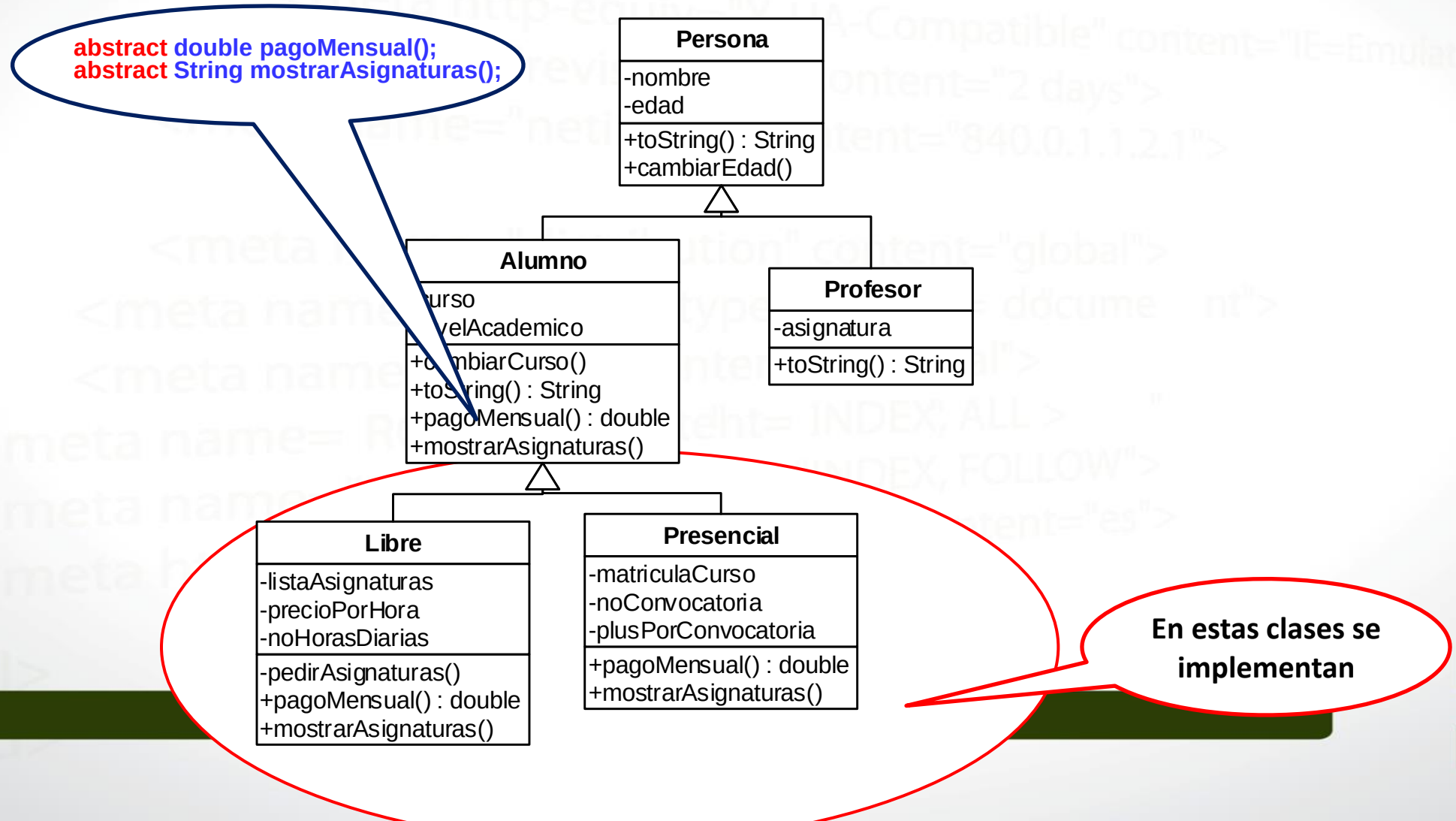
2.1 Clases Abstractas

- Si una clase contiene al menos un **método abstracto**, entonces es una clase abstracta.
- Una **clase abstracta** es una clase de la que no se pueden crear objetos, pero puede ser utilizada como clase padre para otras clases.
- Declaración:

```
abstract class NombreClase {
    ..............
}
```

Para aplicar Polimorfismo debe haber herencia con métodos abstractos, por eso es importante saber en que consiste y como se usan.

Ejemplo de clase abstracta alumno



Ejemplo de clase abstracta

```
abstract class Alumno extends Persona {
    protected int curso;
    private String nivelAcademico;
    public Alumno (String n, int e, int c, String nivel) {
        super(n, e);
        curso = c; nivelAcademico = nivel;
    }
    public String toString() {
        return super.toString() + curso + nivelAcademico;
    }
    abstracta double pagoMensual();
    abstract String getAsignaturas();
}
```

```
class Libre extends Alumno {
    private String []listaDeAsignaturas;
    private static float precioPorHora=10;
    private int noHorasDiarias;
    private void pedirAsignaturas()
    public double pagoMensual() {
        return precioPorHora*noHorasDiarias*30; }
    public String getAsignaturas() {
        String asignaturas="";
        for (int i=0; i<listaDeAsignaturas.length; i++)
            asignaturas += listaDeAsignaturas[i] + ' ';
        return asignaturas;
    }
    public Libre(String n, int e, int c, String nivel, int horas)
      {super(n,e,c,nivel); noHorasDiarias = horas; pedirAsignaturas(); }
}
```

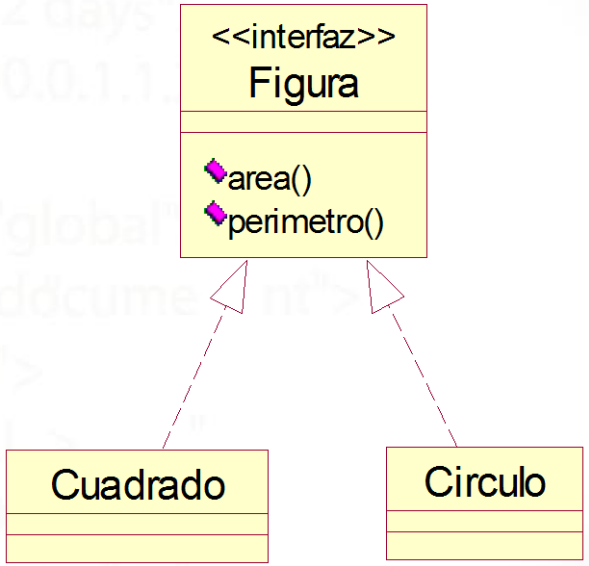
```
class Presencial extends Alumno {
    private double matriculaCurso;
    private double plusPorConvocatoria;
    private int noConvocatoria;
    public double pagoMensual()
    { return (matriculaCurso+plusPorConvocatoria*noConvocatoria)/12; }
    public String getAsignaturas(){
        return "todas las del curso " + curso;
    }
    public Presencial(String n, int e, int c, String nivel,
        double mc, double pc, int nc) {
        super(n,e,c,nivel);
        matriculaCurso=mc;
        plusPorConvocatoria=pc;
        noConvocatoria=nc;
    }
}
```

3. Interfaces

- Cuando solo hay métodos que comparten las clases.
- Ejemplo:
  - Clases: Circulo, Elipse, Triangulo, ...
  - Todas esas clases incluyen los métodos: área, perimetro, cambiarEscala, etc.
- Podríamos definir una interfaz común que agrupe todos los métodos comunes (como métodos abstractos).
- Y luego definir varias clases de modo que implementen una misma interfaz.
- Las interfaces tienen otro significado que lo veremos mejor en patrones de diseño, sirven para desacoplar componentes.

Ejemplo de Interfaz

```
publico interface Figura {
    abstract double area();
    abstract double perimetro();
}
public class Circulo implements Figura {
    private double radio;
    private static double PI=3.1416;
    ..............
    public double area() { return PI*radio*radio; }
    public double perimetro() { return 2*PI*radio; }
}
public class Cuadrado implements Figura {
    private double lado;
    ..............
    public double area() { return lado*lado; }
    public double perimetro() { return 4*lado; }
}
```



3.1 Implementando Interfaces

```
public interface InterfaceName
{
    public void metodo();
}
```

A las interfaces también podríamos usarlo para implementar polimorfismo.

```
Public class ClassName implements InterfaceName [, InterfaceName2, ...]
{
    public void metodo()
    {
    }
}
```