



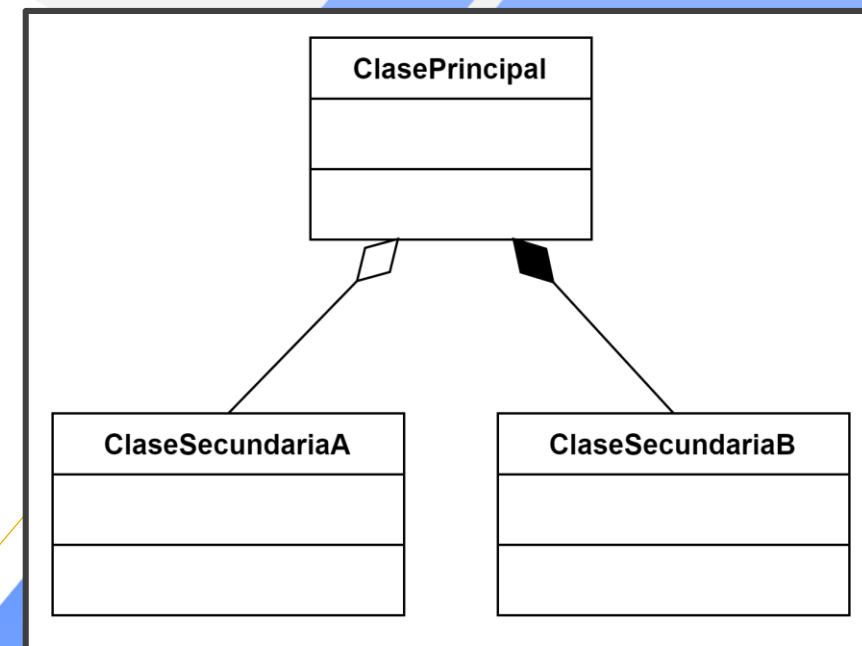
UCA

Programación Orientada a Objetos I

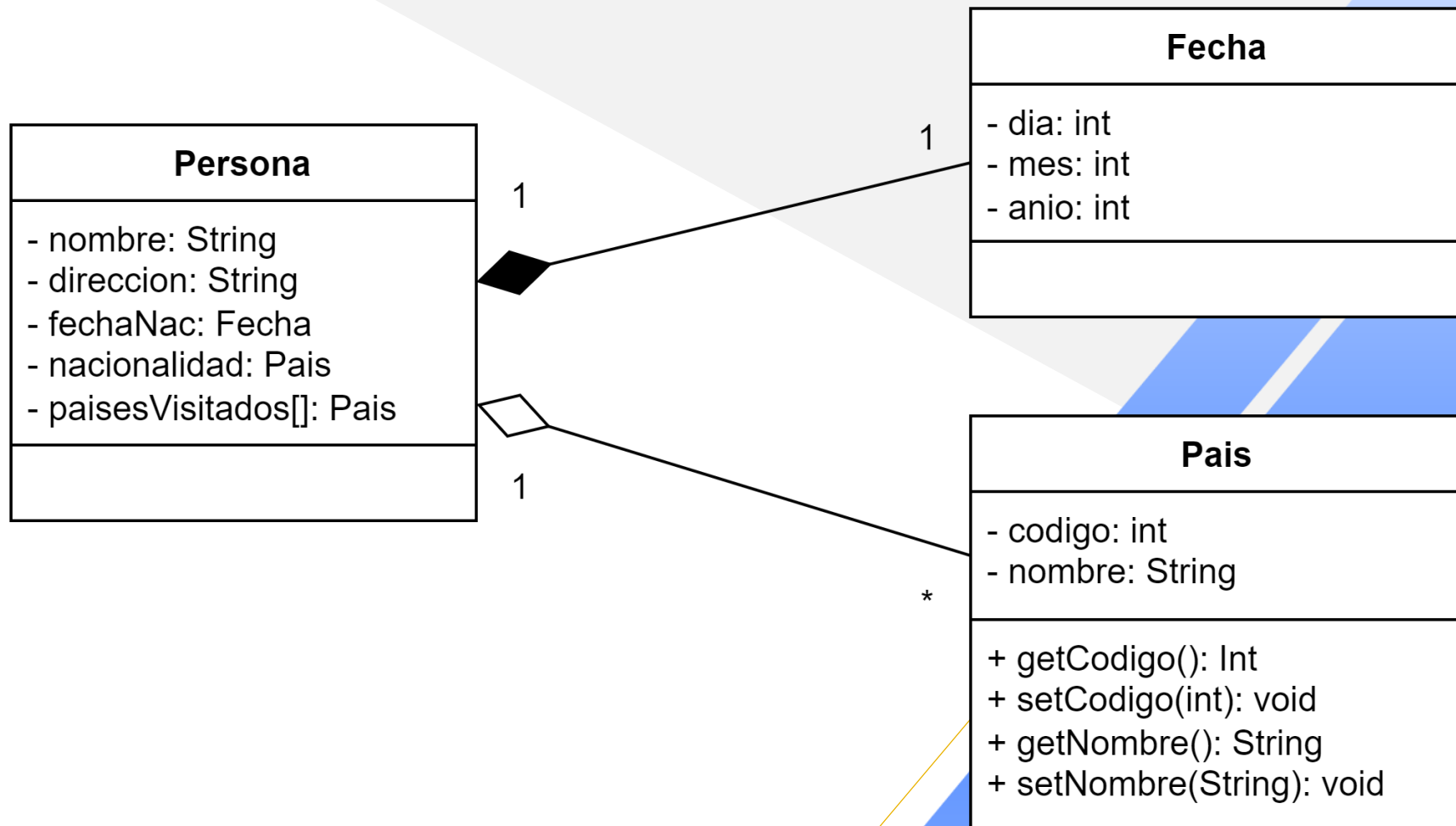
# **JAVA – CLASES (III)**

# Objetos como miembros de una clase

- Un objeto aislado raramente tiene la capacidad de resolver un problema. Los objetos colaboran e intercambian información manteniendo distintos tipos de relaciones.
- La relación de **Asociación** es la más frecuente. Refleja la relación entre dos clases estructuralmente independientes que existe durante la vida de los objetos de esas clases.
- Se implementa en JAVA agregando **referencias a objetos de la clase destino** de la relación (clase secundaria o asociada) **como atributos de la clase origen** (clase principal).
- Esa referencia implementa alguna de las relaciones:
  - Composición
  - Agregación

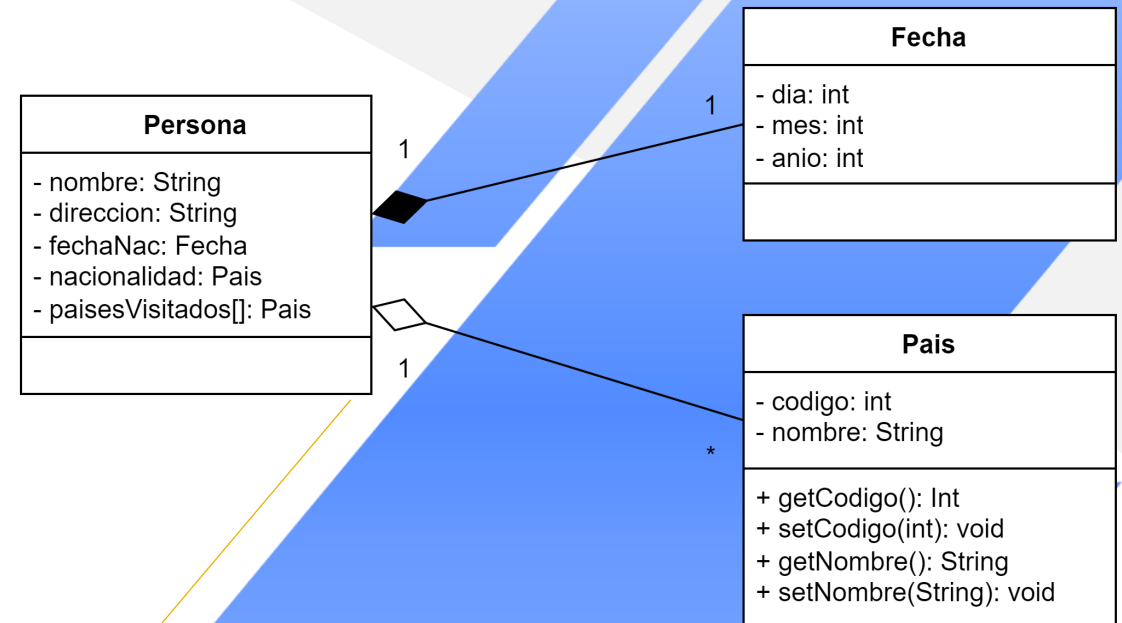


# Objetos como miembros de una clase



# Composición

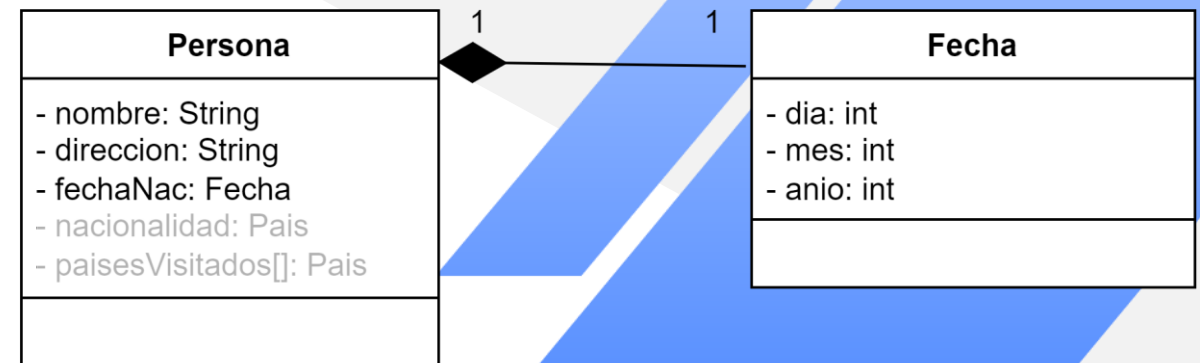
- La clase principal compone un objeto nuevo de la clase relacionada (clase secundaria o asociada).
- En general, el objeto miembro de la clase relacionada no tiene sentido si no existe la clase principal.
- La clase *Persona* almacena el nombre de la persona, su dirección y su fecha de nacimiento. El dato miembro *fechaNac* puede ser un objeto de clase *Fecha*.
- Cuando se crea un objeto *Persona*, primero se invoca al constructor de la clase *Fecha* y después, al constructor de la clase *Persona*.



# Composición

```
public class Persona {  
    private String nombre, direccion;  
    private Fecha fechaNac;  
  
    public Persona(String pNombre, String pDir) {  
        nombre = pNombre;  
        direccion = pDir;  
        fechaNac = new Fecha();  
    }  
    public Persona(){  
        fechaNac = new Fecha();  
    }  
    public String getNombre() {  
        return nombre;  
    }  
    public Fecha getFechaNac(){  
        return fechaNac;  
    }  
    public void setFechaNac(int d, int m, int a) {  
        fechaNac.setDia(d);  
        fechaNac.setMes(m);  
        fechaNac.setAnio(a);  
    }  
}
```

a) `Persona pers1 = new Persona("Christian", "LIR");`  
b) `pers1.setFechaNac(25,4,1999);`  
c) `System.out.println(pers1.getNombre());`  
d) `System.out.println(pers1.getFechaNac());`  
e) `System.out.println(pers1.getFechaNac().getDia());`



¿**QUÉ** hace cada línea? ¿**EN QUÉ ÓRDEN** lo hace?  
¿Cuál es el resultado **EXACTO** de cada línea?

# Composición

a) `Persona pers1 = new Persona("Christian", "LIR");`

- Se "crea" un objeto de tipo Fecha
- Se llama al constructor de Fecha
- Se crea un objeto de tipo Persona
- Se llama al constructor de Persona
- Se asigna la referencia pers1 al objeto Persona creado

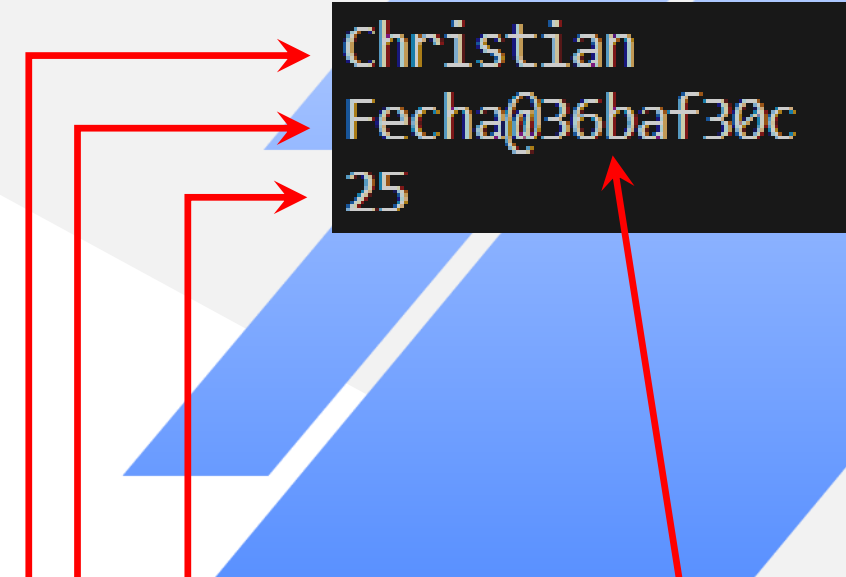
b) `pers1.setFechaNac(25, 4, 1999);`

- Se llama al método `setFechaNac` con los parámetros (25,4,1999)

c) `System.out.println(pers1.getNombre());`

d) `System.out.println(pers1.getFechaNac());`

e) `System.out.println(pers1.getFechaNac().getDia());`



Christian  
Fecha@36baf30c  
25

¿Qué es esto!

# toString

- Como dijimos anteriormente todas las clases que creamos son hijas de la clase **object**, y como tales heredan sus datos y métodos miembro
- Algunos métodos de la clase object para destacar:
  - finalize(): se llama antes de destruir el objeto
  - clone(): hace una copia superficial o shadow copy
  - getClass(): devuelve la clase del objeto
  - equals(): compara si dos objetos son iguales
  - **toString()**: devuelve una representación en string del objeto
    - System.out.println() ejecuta este método
    - Por defecto muestra el nombre de la clase y el hash del objeto separado por el símbolo “@”

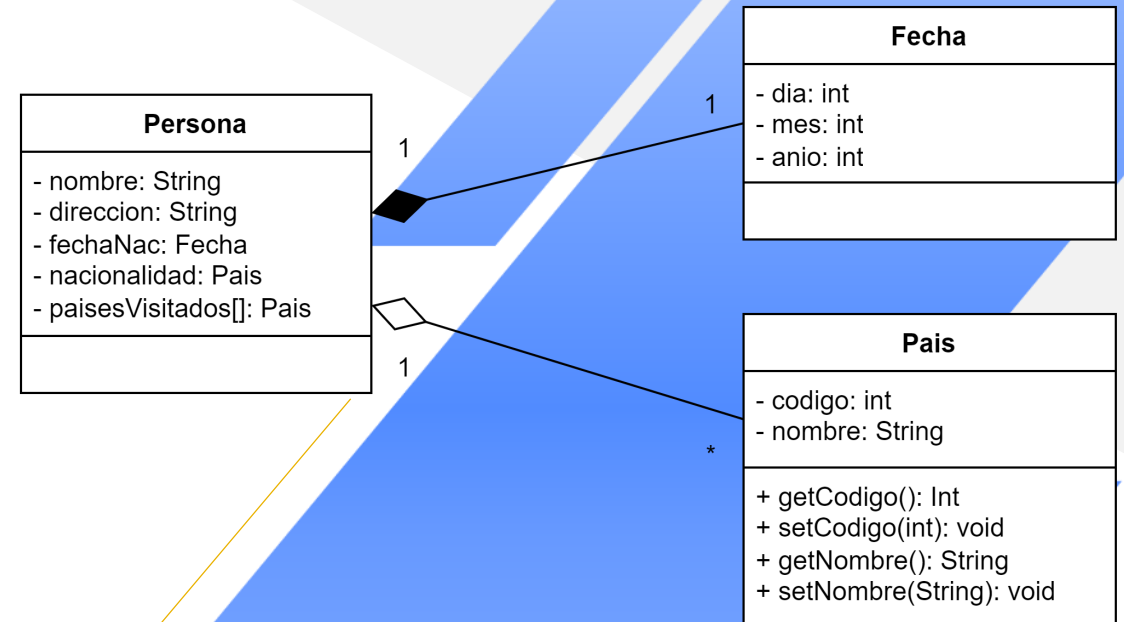
Entonces podemos “mejorar” la visualización del contenido de un objeto sobrescribiendo el método toString() con nuestra propia implementación

```
public String toString(){  
    String str = "";  
    str += "Dia: " + dia;  
    str += " - Mes: " + mes;  
    str += " - Anio: " + anio;  
    return str;  
}
```

```
Christian  
Dia: 25 - Mes: 4 - Anio: 1999  
25
```

# Agregación

- La clase principal agrega un objeto existente.
- La existencia del objeto miembro de una clase asociada es *independiente de la existencia* del objeto de la clase principal. Esto es lo que la diferencia de la composición.
- En la clase *Persona* almacenamos el país que corresponde con la *nacionalidad*.
- Este dato *nacionalidad* debe ser un objeto de clase *Pais*, la cual tiene los datos miembros *codigo* y *nombre*.
- Cuando se crea un objeto *Persona*, el constructor de la misma, recibe como parámetro un objeto ya existente de la clase *Pais*.
- El objeto *Persona* **no necesita invocar al constructor** de la clase *Pais* porque éste ya existe

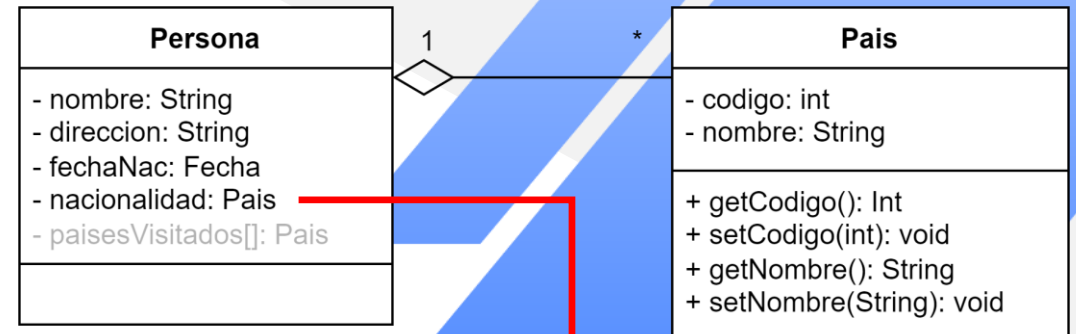




# Agregación

```
public class Persona {  
    private String nombre, direccion;  
    private Fecha fechaNac;  
    private Pais nacionalidad;  
  
    public Persona(String pNombre, String pDir,  
                    Pais pPais) {  
        nombre = pNombre;  
        direccion = pDir;  
        fechaNac = new Fecha();  
        nacionalidad = pPais;  
    }  
    public Persona(){  
        fechaNac = new Fecha();  
    }  
    public Pais getNacionalidad(){  
        return nacionalidad;  
    }  
    public void setNacionalidad(Pais pPais) {  
        nacionalidad = pPais;  
    }  
}
```

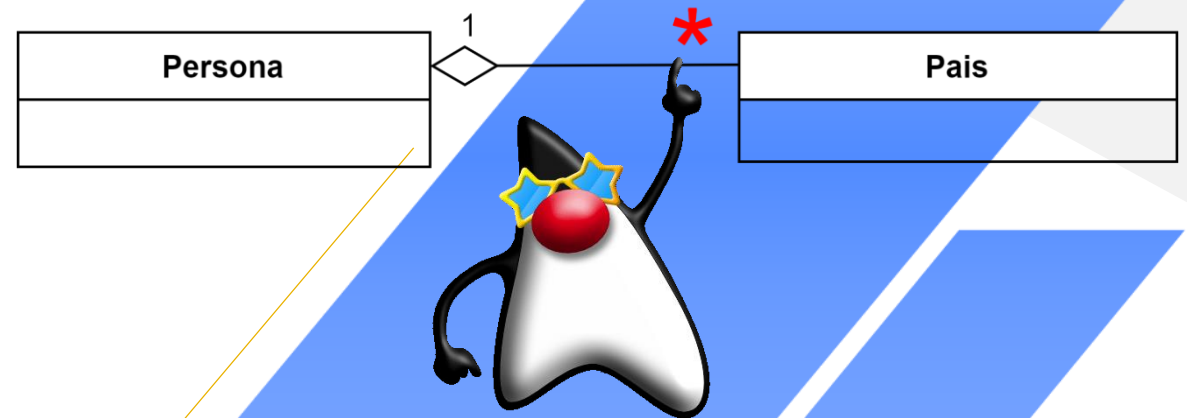
```
Pais paises[] = {new Pais(1, "Argentina"),  
                 Pais(2, "Uruguay"),  
                 Pais(3, "Brasil")}  
  
Persona pers2 = new Persona("Chris", "LIR", paises[3]);  
System.out.println(pers2.getNacionalidad());  
System.out.println(pers2.getNacionalidad().getNombre());
```



¡Solo estamos  
ejemplificando esto!

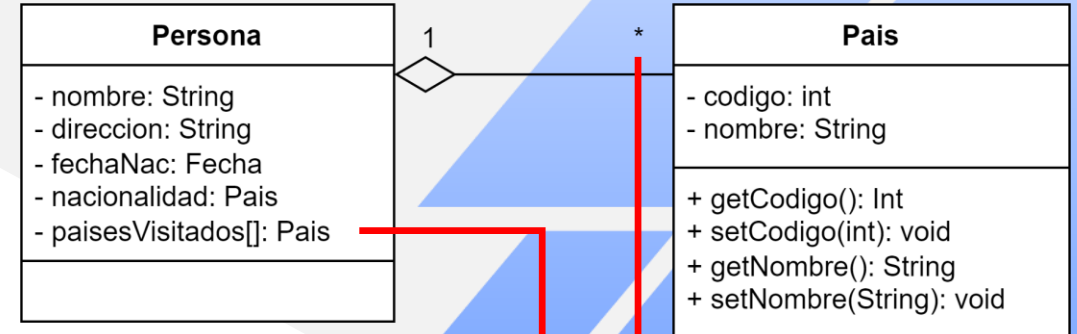
# Array como miembros de una clase

- No solamente podemos incluir datos miembros individuales, sino también la definición de datos miembros múltiples implementados como arrays.
- Recordemos que un array es una estructura de elementos del mismo tipo bajo un mismo nombre o identificador.
- Estas estructuras sirven tanto para datos primitivos como para objetos.
- Representan cardinalidad múltiple (cuando **no es** "uno a uno") entre los objetos de las clases relacionadas.
- Ejemplos:
  - Alumno ---- Notas
  - Persona ---- PaisesVisitados
  - CuentaBancaria ---- Movimientos



# Agregación

```
public class Persona {  
    private String nombre, direccion;  
    private Fecha fechaNac;  
    private Pais nacionalidad  
    private Pais paisesVisitados[] = new Pais[5];  
  
    public Persona(String pNombre, String pDir,  
                    Pais pPais,  
                    paisesVisitados[] pVisitados) {  
        nombre = pNombre;  
        direccion = pDir;  
        fechaNac = new Fecha();  
        nacionalidad = pPais;  
        paisesVisitados = pVisitados;  
    }  
}
```



Ahora estamos  
ejemplificando  
esta parte  
(cardinalidad múltiple)

# Clase Math

- La clase Math representa la librería matemática de Java. Contiene las mismas funciones que otros lenguajes pero encapsuladas en la clase Math.
- Forma parte de la librería java.lang.\*
- Al ser private el constructor de la clase, no se pueden crear instancias de la clase.
- Al ser public puede ser llamada desde cualquier sitio y al ser static no necesita ser inicializada.
- Incluye dos constantes: PI (3.1416) y E (representa el número  $e=2.718$ ).
- Incluye varios métodos static para su manejo, que retornan:
  - `abs(a)`: valor absoluto de a.
  - `ceil(a)`: número entero mayor más cercano.
  - `floor(a)`: número entero menor más cercano.
  - `max(a,b)`: número más grande entre a y b.
  - `min(a,b)`: número menor entre a y b.
  - `pow(a,b)`: número a elevado al exponente b.
  - `sqrt(a)`: raíz cuadrada de a.
  - `round(a)`: valor redondeado al entero más cercano.

## Ejemplos:

```
Math.pow(2, 3) = 8  
Math.sqrt(9) = 3  
Math.ceil(5.6) = 6.0  
Math.floor(5.6) = 5.0  
Math.round(5.6) = 6  
Math.round(5.4) = 5
```

# Clase String

- Forma parte de la librería `java.lang.*`
  - `length`: largo de la cadena.
  - `charAt(int)`: carácter existente en la posición dada.
  - `equals(String)`: verdadero si dos cadenas contienen el mismo valor. De lo contrario, devuelve falso.
  - `indexOf(char, posIni)`: posición donde encuentra el primer carácter coincidente. (-1=not found).
  - `lastIndexOf(char, posIni)`: posición de la última ocurrencia del carácter coincidente. (-1=not found).
  - `substring(int posIni, int posFin)`: devuelve una subcadena que comienza en el primer argumento y finaliza en el segundo menos 1 (`length=ult.carácter`).
  - `replace(oldchar, newchar)`: cambia un carácter por otro.
  - `toUpperCase()/toLowerCase()`: devuelve la cadena en letras mayúsculas/minúsculas.
  - `startsWith(String, pos)/endsWith(String)`: devuelve true si el String comienza o finaliza con el primer argumento. El segundo, indica la posición de inicio.
  - `contains(String)`: devuelve true cuando el parámetro forma parte de la cadena original.
  - `concat(String)`: devuelve otro String con ambas cadenas concatenadas.
  - `split (String)`: a partir del separador indicado como argumento, divide al String en tantas ocurrencias como existan, retornando un array con las mismas.

# Clase String

```
String texto = "Hola, mundo!";
```

```
int longitud = texto.length();  
System.out.println("Longitud de la cadena: " + longitud); // Longitud de la cadena: 12
```

```
char caracter = texto.charAt(4);  
System.out.println("Carácter en la posición 4: " + caracter); // Carácter en la posición 4: ,
```

```
String otraCadena = "Hola, mundo!";  
boolean sonIguales = texto.equals(otraCadena);  
System.out.println("Las cadenas son iguales: " + sonIguales); // Las cadenas son iguales: true
```

```
int indice = texto.indexOf("mundo");  
System.out.println("Índice de 'mundo': " + indice); // Índice de 'mundo': 6
```

```
int ultimoIndice = texto.lastIndexOf("o");  
System.out.println("Último índice de 'o': " + ultimoIndice); // Último índice de 'o': 10
```

```
String subcadena = texto.substring(0, 4);  
System.out.println("Subcadena: " + subcadena); // Subcadena: Hola
```

```
String reemplazado = texto.replace("mundo", "amigo");  
System.out.println("Cadena reemplazada: " + reemplazado); // Cadena reemplazada: Hola, amigo!
```

# Clase String

```
String texto = "Hola, mundo!";
```

```
String mayusculas = texto.toUpperCase();  
String minusculas = texto.toLowerCase();  
System.out.println("Mayúsculas: " + mayusculas); // Mayúsculas: HOLA, MUNDO!  
System.out.println("Minúsculas: " + minusculas); // Minúsculas: hola, mundo!
```

```
boolean comienzaCon = texto.startsWith("Hola");  
boolean terminaCon = texto.endsWith("!");  
System.out.println("Comienza con 'Hola': " + comienzaCon); // Comienza con 'Hola': true  
System.out.println("Termina con '!': " + terminaCon); // Termina con '!': true
```

```
boolean contiene = texto.contains("mundo");  
System.out.println("Contiene 'mundo': " + contiene); // Contiene 'mundo': true
```

```
String concatenado = texto.concat(" ¡Es un buen día!");  
System.out.println("Cadena concatenada: " + concatenado); // Cadena: Hola, mundo! ¡Es un buen día!
```

```
String[] partes = texto.split(",");  
System.out.println("Partes divididas:");  
for (String parte : partes) {  
    System.out.println(parte); // 1ra salida: Hola  
                                // 2da salida: mundo!  
                                //  
                                // con el espacio adelante  
}
```

# Clase Random

- La clase *Random* permite la generación de números aleatorios.
- Forma parte de la librería `java.util.*`
- Generación de objetos:  
`Random r = new Random();`
- Métodos principales:
  - `r.nextInt()`: devuelve números aleatorios enteros
  - `r.nextInt(n)`: devuelve números aleatorios enteros positivos hasta `n-1`
  - `r.nextFloat()`: devuelve números aleatorios de tipo `float`
  - `r.nextDouble()`: devuelve números aleatorios de tipo `double`
  - `r.nextLong()`: devuelve números aleatorios de tipo `long`



# Clase Calendar

- Forma parte de la librería java.util.\*
- Antes solamente existía la clase Date (Date d = new Date();)
- En la actualidad, existe la clase abstracta Calendar que redefine la clase Date, utilizando constantes fijas (DATE, MONTH, YEAR, etc)

```
Calendar c = Calendar.getInstance() // fecha del día
                                     // no usar new - solo instanciar

c.getTime();
```

## Métodos de la Clase Calendar:

```
c.set           // cambia la fecha
c.get           // devuelve la fecha
c.add           // calcula una nueva fecha
c.before/c.after // true cuando la fecha es anterior/posterior
```

# Clase Calendar

```
import java.util.*;

// Obtener una instancia de Calendar
Calendar calendar = Calendar.getInstance();

// Obtener la fecha y hora actual como un objeto Date
Date fechaActual = calendar.getTime();
System.out.println("Fecha y hora actual: " + fechaActual);

// Obtener el año actual
int añoActual = calendar.get(Calendar.YEAR);
System.out.println("Año actual: " + añoActual);

// Establecer el mes en marzo (los meses se cuentan desde 0)
calendar.set(Calendar.MONTH, Calendar.MARCH);
System.out.println("Mes actual después de establecerlo en marzo: " + calendar.get(Calendar.MONTH));

// Añadir 30 días al día actual
calendar.add(Calendar.DAY_OF_MONTH, 30);
System.out.println("Fecha después de agregar 30 días: " + calendar.getTime());

// Comprobar si la fecha actual está antes del 1 de enero de 2023
Calendar otraFecha = Calendar.getInstance();
otraFecha.set(2023, Calendar.JANUARY, 1);
boolean antesDe2023 = calendar.before(otraFecha);
System.out.println("¿Fecha actual antes de 2023? " + antesDe2023);
```