INICIACIÓN A LA PROGRAMACIÓN
LENGUAJE JAVA con BlueJ

Tema 3 *Clases y Objetos*

Tema 4 *Comunicación entre objetos. Algoritmos*

Tema 5 *Herencia y abstracción de datos*

Tema 6 *Diseño de clases*

TEMA 5: Herencia y abstracción datos

1. Abstracción de datos
   - Encapsulación de datos
   - Composición
2. Herencia
   - Concepto de Herencia
   - Herencia y constructores
   - Herencia y métodos: redefinición, uso de super
   - Herencia y visibilidad: protected
3. Polimorfismo
4. Clase Object
5. Listas: vectores dinámicos

TEMA 5 : Herencia y abstracción de datos

Abstracción de datos: definición

- En el lenguaje natural continuamente utilizamos palabras que representan la abstracción de un concepto.
- Por ejemplo, la palabra “gato” es la abstracción de:
  - animal con una cabeza, cuatro patas, cola, etc.
  - es capaz de andar, maullar, comer, etc.
- En lenguajes de programación orientados a objetos también abstraemos los conceptos mediante:
  - campos: cabeza, patas, cola, etc.
  - métodos: andar, maullar, comer, etc.
- La **abstracción de datos** es el proceso de identificar los atributos y propiedades de los conceptos.
- El resultado de este proceso es un **tipo de dato abstracto**: Gato, Matriz, Persona.

TEMA 5 : Herencia y abstracción de datos

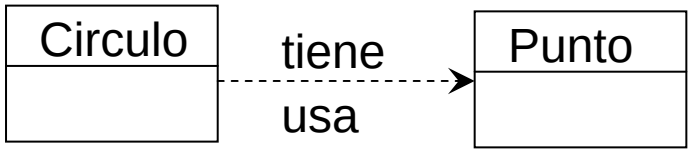
Abstracción de datos: encapsulación

- Hasta el momento, todos los campos que hemos definido han sido `private`
- Razón: buena técnica de programación ocultar la representación interna de los objetos, **encapsulación de datos**.
- Al usuario (programador) no le interesa cómo se representa el nombre de una Persona: con un String, con un array de caracteres, con una clase especial Nombre.
- Solo le interesa cómo obtener el nombre.
- Los detalles de implementación se *ocultan* en la clase.

TEMA 5 : Herencia y abstracción de datos

Abstracción de datos: composición

- En muchas ocasiones al crear una clase hemos utilizado otra como campo.
- Por ejemplo, la clase Círculo tiene como campo un objeto de la clase Punto.
- Se define una relación entre clases: **composición**.
- En el diagrama de clases se representa



- Decimos que la clase Circulo "*tiene un*" Punto.
- La relación se representa mediante una línea punteada

TEMA 5 : Herencia y abstracción de datos

Herencia

- Consideremos las clases

```
public class Alumno {
  private String nombre;
  private int edad;
  private int nip; //identificador
  private Asignatura[] matricula;

  public Alumno(String no, int e, int ni, Asignatura [] mat) {
    . . . . .
  }
  public String getNombre() {
     return nombre;
  }
  public int getEdad() {
     return edad;
  }
  public int getNip() {
     return nip;
  }
  public int hallarCursoMasBajoMatriculado() {
    . . . . .
  }
}
```

TEMA 5 : Herencia y abstracción de datos

Herencia

```
public class Profesor {
  private String nombre;
  private int edad;
  private int nip; //identificador
  private double sueldoBase;
  private int añosTrabajados

  public Profesor(String no, int e, int ni, double sb, int a) {
    . . . .
  }
  public String getNombre() {
     return nombre;
  }
  public int getEdad() {
     return edad;
  }
  public int getNip() {
     return nip;
  }
  public double hallarSueldo() {
    . . . .
  }
}
```

TEMA 5 : Herencia y abstracción de datos

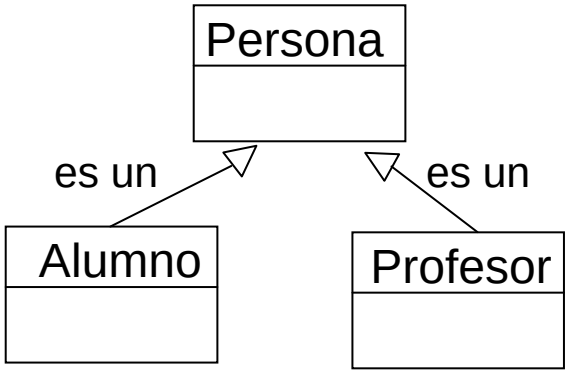
Herencia

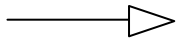
- Las clases Alumno y Profesor comparten:
  - algunos campos: nombre, edad, nip
  - algunos métodos: getNombre(), getEdad(), getNip()
- ¿Es necesario escribir el mismo código dos veces? Para la clase Alumno y Profesor.
- Usando el mecanismo de **herencia** no es necesario

TEMA 5 : Herencia y abstracción de datos

Herencia

- Alumno y Profesor comparten que son Personas.
- Se crea la clase Persona con la parte común a Alumno y Profesor.
- Se crea Alumno y Profesor heredando de Persona.
- La herencia proporciona una relación entre clases *“es un”:*



- Un Alumno *es una* Persona, un profesor *es una* Persona
- En el diagrama de clases, la relación de herencia: 

TEMA 5 : Herencia y abstracción de datos

Herencia

```
public class Persona {
  private String nombre;
  private int edad;
  private int nip; //identificador

  public Persona(String no, int e, int ni) {
    . . . .
  }
  public String getNombre() {
     return nombre;
  }
  public int getEdad() {
     return edad;
  }
  public int getNip() {
     return nip;
  }
 }
```

TEMA 5 : Herencia y abstracción de datos

Herencia: en Java

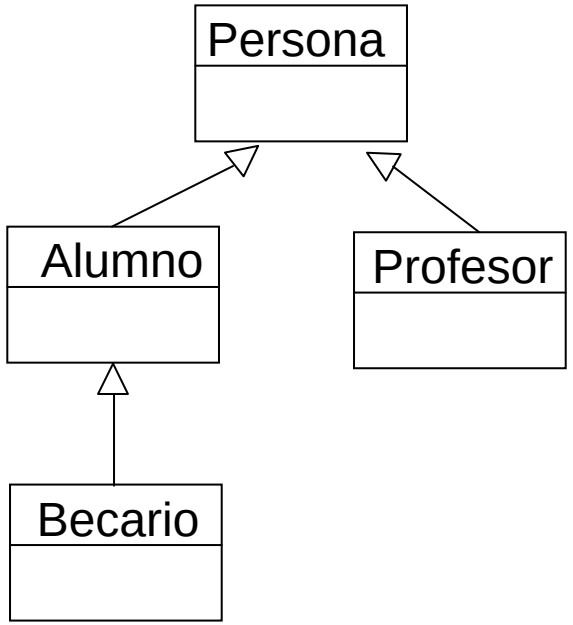
```
public class Profesor extends Persona {
  private double sueldoBase;
  private int añosTrabajados

  public Profesor(String no, int e, int ni, double sb, int a) {
    . . . .
  }
  public double hallarSueldo() {
    . . . .
  }
 }
```

```
public class Alumno extends Persona {
  private Asignatura[] matricula;

  public Alumno(String no, int e, int ni, Asignatura [] mat) {
    . . . .
  }

  public int hallarCursoMasBajoMatriculado() {
    . . . .
  }
 }
```

TEMA 5 : Herencia y abstracción de datos

Herencia

- El mecanismo de herencia se puede aplicar tantas veces como sea necesario
- La clase que hereda se llama: clase **derivada** o **heredada**
- Alumno es una clase derivada de Persona
- La clase de la que se hereda se llama: clase **base** o **padre**
- Persona es la clase base o padre de Alumno. Alumno es la clase base o padre de Becario
- Todas las clases por “encima” de la jerarquía de clases: **superclases**
- Las superclases de Becario son: Alumno y Persona



Jerarquía de clases

TEMA 5 : Herencia y abstracción de datos

Herencia

- Las clases derivadas heredan todos los campos y métodos de su clase base.
- Lo único que **no se hereda** son los constructores.

En otras palabras:

- Los campos de la clase Alumno son
  - Los propios de Alumno: Asignatura[] matricula
  - más los de la clase Persona: nombre, edad, nip
- Los campos de Profesor son
  - los propios de Profesor: sueldoBase, añosTrabajados
  - más los de la clase Persona: nombre, edad, nip
- Los métodos de Alumno son:
  - los propios de Alumno: hallarCursoMasBajoMatriculado()
  - más los de la clase Persona: getNombre(), getEdad(), getNip()
- Los métodos de Profesor son:
  - los propios de Profesor: hallarSueldo()
  - más los de la clase Persona: getNombre(), getEdad(), getNip()

TEMA 5 : Herencia y abstracción de datos

Herencia: ejemplo

```
public class Punto2D {
  private double x;
  private double y;

  public Punto2D() {
    x = 0; y = 0;
  }
  public Punto2D(double nx,
                 double ny) {
     x = nx; y = ny;
  }
  public double getX() {
    return x;
  }
  public double getY() {
    return y;
  }
}
```

```
public class Punto3D extends Punto2D{
  private double z;

  public Punto3D() {
    . . . .
  }

  public Punto3D(double nx, double ny,
                           double nz) {
     . . . . .
  }

  public double getZ() {
    return z;
  }
}
```

TEMA 5 : Herencia y abstracción de datos

Herencia: constructores

- En el ejemplo anterior, ¿cómo se implementan los constructores de Punto3D?
- Primera aproximación **ERRÓNEA:** como los campos **x** e **y** son de Punto3D, asignamos los valores directamente

```
public class Punto3D extends Punto2D{
  private double z;
  . . . .

  public Punto3D(double nx, double ny, double nz) {
     x = nx; y = ny; z = nz; //error compilación
  }
  . . . .
}
```

- No se puede: los campos **x** e **y** son privados. Solo se tiene acceso directo a ellos dentro de la clase Punto2D
- Solución **chapuza**: declaro los campos públicos. El código sería correcto sintácticamente. **NO SE DEBE HACER NUNCA**.

TEMA 5 : Herencia y abstracción de datos

Herencia: constructores

- Para construir un objeto de una clase derivada es **necesario** llamar a un constructor de la clase base
- La llamada se hace mediante: *super(arg1,arg2,..)* donde los argumentos se corresponden con algún constructor clase base.
- Con la llamada a *super* se construye una parte del objeto, la correspondiente a la clase base.
- Después hay que dar valor a los campos propios de la clase derivada

```
public class Punto3D extends Punto2D{
  private double z;
  . . . .
  public Punto3D(double nx, double ny, double nz) {
     super(nx, ny);//llamada constructor clase base Punto2D
     z = nz; //valor de la parte propia de Punto3D
  }
  . . . .
}
```

Herencia: constructores

- ¿Qué ocurre si la clase tiene un constructor por defecto?
- En este caso, y solo en este, si no se llama explícitamente a un constructor de la clase base, **implícitamente** se llama al constructor por defecto de la clase base.
- Esto es, se hace una llamada a *super()*.

```
public class Punto3D extends Punto2D{
  . . . .
  public Punto3D() {//para crear el punto (0,0,0)
      z = 0; //valor de la parte propia de Punto3D
  }
  . . . .
```

```
public class Punto3D extends Punto2D{
  . . . .
  public Punto3D() {//para crear el punto (0,0,0)
    super(); //llamada a constructor por defecto clase base
             //no es necesario, equivalente anterior
    z = 0;   //valor de la parte propia de Punto3D
  }
```

TEMA 5 : Herencia y abstracción de datos

Herencia: ejemplo constructores

```
public class Persona {
  . . .
  public Persona(String no, int e, int ni) {
    nombre = no; edad = e; nip = ni;
  }
  . . .
}
```

```
public class Alumno extends Persona {
  . . .
  public Alumno(String no, int e, int ni, Asignaturas[] mat) {
    super(no, e, ni);
    matricula = mat;
  }
  . . .
}
```

```
public class Profesor extends Persona {
  . . .
  public Profesor(String no, int e, int ni, double sb, int a) {
    super(no, e, ni);
    sueldoBase = sb; añosTrabajados = a;
  }
  . . .
}
```

Herencia: ejemplo uso métodos

```
Punto2D p2d = new Punto2D(1, 2);
Punto3D p3d = new Punto3D(4, 5, 6);

System.out.println(p2d.getX()); //correcto
System.out.println(p3d.getY()); //correcto
System.out.println(p3d.getZ()); //correcto

System.out.println(p2d.getZ()); //incorrecto.
                                // no hay método getZ en Punto2D
```

TEMA 5 : Herencia y abstracción de datos

Herencia: redefinición de métodos

- Añadimos a la clase Punto2D un método para hallar la distancia al origen

```
public class Punto2D{
   private double x;
   private double y;
   . . .
   public double hallarDistanciaOrigen() {
     return Math.sqrt(x*x + y*y);
   }
   . . . .
}
```

- La clase derivada Punto3D hereda el método hallarDistanciaOrigen()
- Por tanto,

```
Punto3D p3d = new Punto3D(1, 2, 3);
System.out.println(p3d.hallarDistanciaOrigen());
```

es correcto sintácticamente, pero no da el resultado buscado.

TEMA 5 : Herencia y abstracción de datos

Herencia: redefinición de métodos

- Solución: redefinir el método hallarDistanciaOrigen() en la clase derivada Punto3D

```
public class Punto3D extends Punto2D{
   private double z;
   . . .
   public double hallarDistanciaOrigen() {
     return Math.sqrt(x*x + y*y + z*z);
   }
   . . . .
}
```

- Redefinir un método en una clase derivada es cambiar la implementación del método de una superclase para que se adapte a las necesidades de la clase derivada.

- No todos lo métodos necesitan ser redefinidos. Por ejemplo, los métodos getX(), getY() de Punto2D no están redefinidos en Punto3D

TEMA 5 : Herencia y abstracción de datos

Herencia: uso de super en métodos

- Añadimos a la clase Persona método toString()

```
public class Persona{
   . . . .
   public String toString() {
     Strin res = “Nombre: “ + nombre + “. Edad: “ + edad +
                                                “. NIP: “ + nip;
     return res;
   }
   . . . .
}
```

- En la clase Profesor y Alumno deberíamos redefinir el método toString() para que apareciesen los datos propios de las clases.

- ¿Cómo?

Herencia: uso de super en métodos

¿Es correcto este código?

```
public class Profesor extends Persona{
   . . . .
   public String toString() {
     //ERROR
     String res = “Nombre: “ + nombre + “. Edad: “ + edad +
                                                “. NIP: “ + nip;
     res += “. Años trabajados: ” + añosTrabajados;
     res += “. SueldoBase: “ + sueldoBase;
     return res;
   }
   . . . .
}
```

NO. Los campos nombre, edad, nip son privados de Persona. No se tiene acceso a ellos directamente en la clase Profesor

TEMA 5 : Herencia y abstracción de datos

Herencia: uso de super en métodos

Solución 1: **bastante mejorable**. Usar los métodos públicos de Persona que nos devuelven los campos nombre, edad, nip.

```
public class Profesor extends Persona{
   . . . .
   public String toString() {
     Strin res = “Nombre: “ + getNombre();
     res += “. Edad: “+ getEdad();
     res += “. NIP: “ + getNip;
     res += “. Años trabajados: ” + añosTrabajados;
     res += “. SueldoBase: “ + sueldoBase;
     return res;
   }
   . . . .
}
```

Herencia: uso de super en métodos

Solución 2: **preferida**. Llamar al método toString() usando la cláusula

```
super.llamada_metodo_clase_base
```

```
public class Profesor extends Persona{
   . . . .
   public String toString() {
     Strin res = super.toString(); //toString() de Persona
     res += “. Años trabajados: ” + añosTrabajados;
     res += “. SueldoBase: “ + sueldoBase;
     return res;
   }
   . . . .
}
```

TEMA 5 : Herencia y abstracción de datos

Herencia

- La relación de herencia nos permite referenciar objetos de las clases derivadas mediante variables de las clases bases.
- Un Alumno es una Persona, pero no todas las Personas son Alumnos

```
Alumno al = new Alumno(. . . . );
Profesor pr = new Profesor(. . . .);
Persona p = new Persona(. . . . );

Persona per1 = al; //Correcto, un Alumno es Persona
Persona per2 = pr; //Correcto, un Profesor es Persona

al = p  //incorrecto, al es una variable de tipo Alumno
        //no puede hacer referencia a una persona
```

TEMA 5 : Herencia y abstracción de datos

Visibilidad: protected

- La palabra clave `protected` se usa para indicar la visibilidad de campos y métodos.

- Se tendrá acceso a un campo o método declarado como `protected` desde cualquier clase que pertenezca al mismo paquete o **desde cualquier clase derivada**.

- En este curso no vamos a usar casi nunca este tipo de visibilidad.

TEMA 5 : Herencia y abstracción de datos

Polimorfismo

- Encapsulación de datos, herencia y polimorfismo son tres conceptos clave de la Programación Orientada a Objetos.
- El polimorfismo es la capacidad que tienen los métodos de las clases derivadas de comportarse de distinta forma con referencias a superclases.

```
Alumno al = new Alumno(. . . . );
Profesor pr = new Profesor(. . . .);
Persona persona;

persona = pr;
System.out.println(persona.toString());
//se ha usado el método toString() de Profesor

persona = al;
System.out.println(persona.toString());
//se ha usado el método toString() de Alumno
```

TEMA 5 : Herencia y abstracción de datos

Polimorfismo

- En este ejemplo, se simula el lanzamiento una moneda.
- Si sale cara, se introduce un alumno en el array
- Si sale cruz, se introduce un profesor
- ¿Qué aparece por pantalla?
- No se usa el método toString() de Persona.
- Se usa el toString() de Alumno o Profesor

```
Alumno al = new Alumno(. . . . );
Profesor pr = new Profesor(. . . .);
Persona [] gente = new Persona[2];
Random gen = new Random();
for (int i = 0; i < gente.length; i++) {
   int moneda = gen.next(2);
   if (moneda == 0) {
      gente[i] = al;
   } else {
      gente[i] = pr;
   }
}

for (int i = 0; i < gente.length; i++) {
   System.out.prilntln(gente[i].toString());
}
```

TEMA 5 : Herencia y abstracción de datos

Clase Object

- Suponer que no tenemos declarado en la clase Punto2D el método toString().
- Si se ejecuta el siguiente código

```
Punto2D p = new Punto(1, 2);
System.out.println(p.toString());
```

- Aparece por pantalla algo como

```
123f3567@Punto
```

- ¿Por qué se puede invocar el método toString() si no lo hemos definido?
- Razón: todas las clases en Java, incluso las creadas por nosotros, derivan automáticamente de una clase llamada ***Object***.

TEMA 5 : Herencia y abstracción de datos

Clase Object

- La clase Object tiene implementados varios métodos. Alguno de ellos son:
  - int hashCode(): devuelve un entero que identifica al objeto
  - String toString(): devuelve un String compuesto por

    nombre de la clase + '@' + el número devuelto por hashCode()

- Notar que cada vez que hemos implementado el método toString() en una clase, hemos redefinido el método ya implementado de la clase Object.

```
public class Punto2D {
  . . . //redefinición del método toString() de Object
  public String toString() {
     String res = "(" + x + "," + y + ")";
     return res;
  }
  . . . . .
}
```

TEMA 5 : Herencia y abstracción de datos

Listas: vectores dinámicos

- Los arrays definidos hasta el momento tienen capacidad para un número fijo de elementos.
- Problema: guardar una serie de números enteros.
- Solución:
  - estática: en un array tipo `int[]`.
    - Incoveniente: cuando creamos el array hay que decidir la capacidad del array. ¿Si la serie tiene más números que la capacidad?

```
int [] guardar = new int[50];
```

  - dinámica: usar un objeto de la clase ***ArrayList***
    - Ventaja: en este tipo de vectores, que llamaremos listas, no hay capacidad fijada. Se pueden introducir y quitar los elementos que se quieran.

TEMA 5 : Herencia y abstracción de datos

Listas: vectores dinámicos

- ¿Cómo se usa un *ArrayList?*
- La clase *ArrayList* pertence al paquete *java.util*
- Cuando se declara el objeto de tipo *ArrayList* hay que indicar qué tipo de dato se guardará entre <tipoDato>

```
ArrayList<Integer> guardar = new ArrayList<Integer>();
ArrayList<Persona> poblacion = new ArrayLis<Persona>();
```

- Cuidado: si queremos guardar algún tipo de dato primitivo

ERROR	CORRECTO
ArrayList<int>	ArrayList<**Integer**>
ArrayList<long>	ArrayList<**L**ong>
ArrayList<char>	ArrayList<**Character**>
ArrayList<boolean>	ArrayList<**B**oolean>
ArrayList<double>	ArrayList<**D**ouble>

TEMA 5 : Herencia y abstracción de datos

Listas: vectores dinámicos

- Una vez declarado y creado el ArrayList se utilizan los métodos de la clase

```
ArrayList<Integer> lista = new ArrayList<Integer>();
lista.add(5); //añade un elemento al final de la lista
lista.add(6);
int tamaño = lista.size(); //total de elementos guardados en lista
int n = lista.get(0);//devuelve el elemento guardado en posición 0
//mostrar todos los elementos por pantalla
for (int i = 0; i < lista.size(); i++) {
    System.out.println(lista.get(i));
}
lista.remove(0); //elimina el elemento guardado en la posición 0
```

TEMA 5 : Herencia y abstracción de datos

Listas: bucle *para cada*

- Las colecciones de elementos tipo ArrayList o arrays se pueden recorrer también mediante el tipo de bucle *para cada*

```
ArrayList<Integer> lista = new . . . ;
//mostrar todos los elementos por pantalla
for (int n : lista) { //para cada n en lista
    System.out.println(n);
}
```

Sintaxis:

• Si *colec* es una colección de datos de tipo *Tipo* (*ArrayList*<*Tipo*>, *Tipo[]*)

```
for (Tipo o : colec) { // para cada o en colec
  . . . . .
}
```