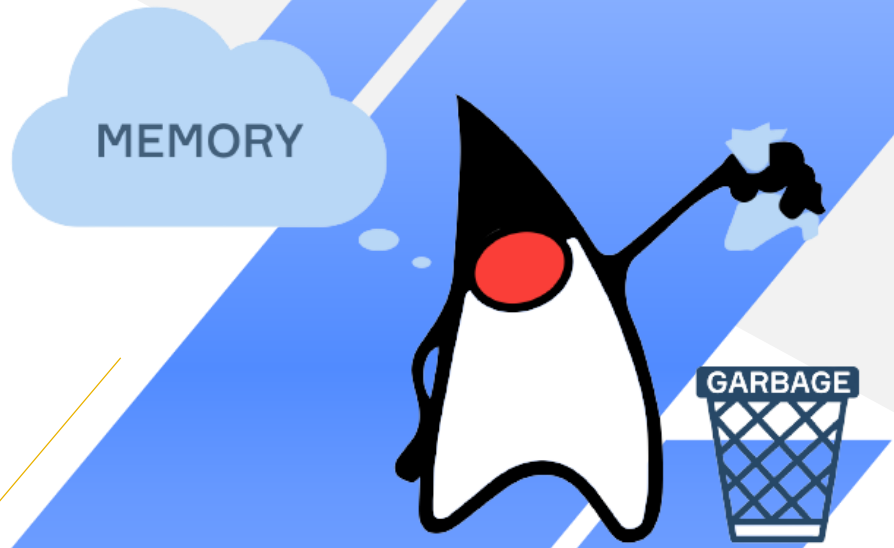
UCA

Programación Orientada a Objetos I

JAVA – CLASES (II)

Destrucción de objetos

- JAVA no utiliza destructores ya que posee una forma **automática** de liberar memoria a través del Garbage Collector y sus procesos automatic garbage collection.
- El Garbage Collector en tiempo de ejecución rastrea cada objeto creado, detecta aquellos que ya no son referenciados y libera el objeto en caso de ser necesario.
- **Ventajas**:
  - detecta objetos no utilizados
  - reutilización de la memoria
  - no necesita explícita liberación (destructores)
- **Desventaja**:
  - incrementa el tiempo de ejecución



Método de finalización

- Los métodos de finalización son miembros especiales de la clase que se invocan **antes** de que el objeto sea recolectado como basura y reclame memoria.
- Java tiene un solo método de finalización, el método ***finalize()***. Si no se declara este método en la clase, JAVA "crea" uno por default sin argumentos, retornando ***void***, pero sin realizar ninguna acción.
- Su invocación **no provoca** que un objeto sea llamado para ser recolectado como basura.
- Se utilizan para:
  - optimizar la remoción de un objeto
  - eliminar las referencias a otro objeto
  - liberar recursos externos
- Ejemplo:

```
void finalize(){
  return;
}
```

Cláusula *final*

- Impide la redefinición de métodos, la extensión completa de clases y el cambio de valores de los atributos.
  - Un atributo final es aquel al cual no puede cambiarse su valor.
  - Un método final es aquel que no puede ser re-escrito.
  - Una clase final es aquella de la cual no se puede heredar.

- En Atributos:

```
class Circulo {
  public final static float PI = 3.14;
}
```

- En Métodos:

```
class Empleado {
  public final void Sueldo(int porc){
  }
}
```

- En Clases:

```
final class Ejecutivo{
  //definición de la clase
}
```

Clases y métodos abstractos

Métodos abstracto:

- Un método abstracto es un método declarado en una clase para el cual esa clase ***no proporciona la implementación*** (no tiene código).
- Están disponibles en las clases derivadas, las cuales deben definir el mismo (polimorfismo).

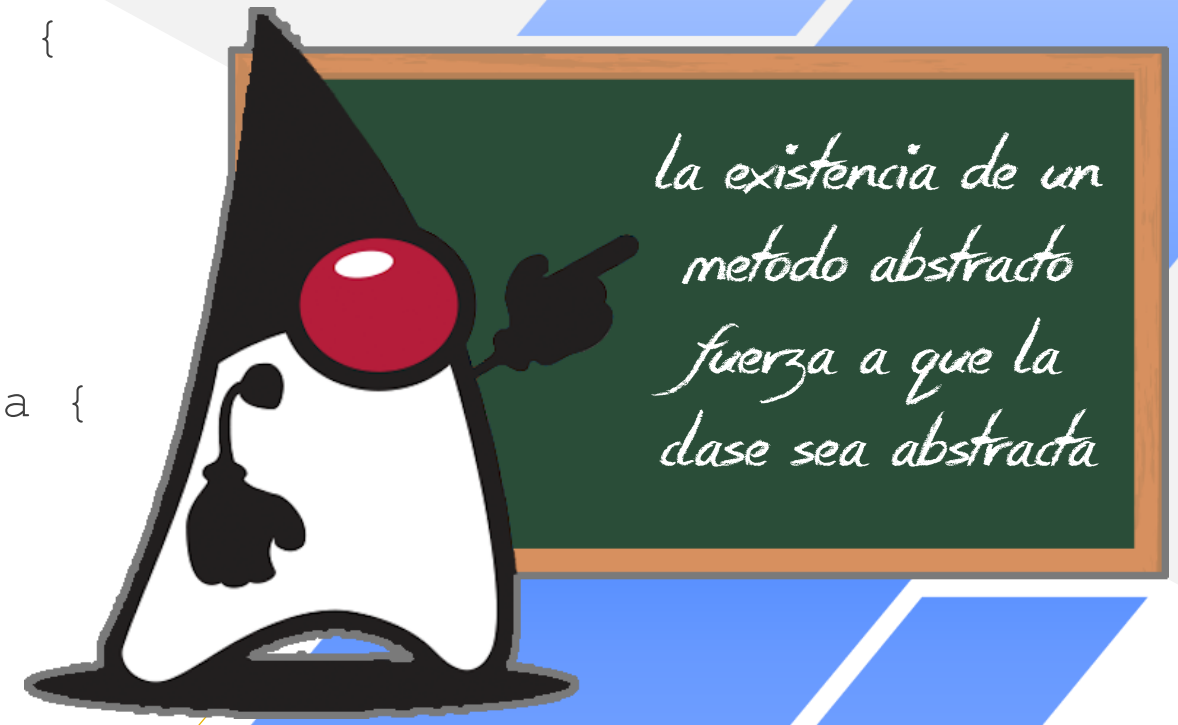
Clase abstracta:

- Una clase abstracta es aquella que tiene ***al menos un método abstracto***.
- Una clase que extiende a una clase abstracta ***debe implementar los métodos abstractos***, o bien, volverlos a declarar como abstractos, con lo que ella misma se convierte también en clase abstracta.

Clases y métodos abstractos

```
abstract class FiguraGeometrica {
    public abstract double perimetro();
}

class Circulo extends FiguraGeometrica {
    private double radio;
    public double perimetro(){
        return Math.PI * radio;
    }
}
class Cuadrado extends FiguraGeometrica {
    private double lado;
    public double perimetro() {
        return lado*4;
    }
}
```



Clases y métodos abstractos

- Se pueden crear referencias a clases abstractas.

```
FiguraGeometrica figura;
```

- Pero una clase abstracta **no se puede instanciar** (no se pueden crear objetos)

```
FiguraGeometrica figura = new FiguraGeometrica();
// La línea de arriba va a dar error
```

- Esto es coherente dado que una clase abstracta **no tiene completa su implementación** y algo abstracto **no puede materializarse**.
- Para que sirve entonces? Para implementar el **Polimorfismo de herencia**

Miembros *static*

- Un dato miembro de una clase declarado *static* implica que **sólo existirá una copia de ese miembro**, la cual es compartida por todos los objetos de la clase.
- También conocida como Variable de Clase, **existe aunque no existan objetos de esa clase**.
- La inicialización de un miembro estático es igual que la de cualquier dato miembro de la clase, anteponiendo la palabra reservada ***static***.
- **Debe referenciarse sobre el nombre de la clase** directamente, aunque JAVA permita ~~hacerlo desde un objeto~~ (no respeta la orientación a objetos).
- Para transformarlo en constante, se debe agregar la palabra clave ***final*** después de la palabra clave *static*.
- Los métodos de acceso al mismo también son estáticos, y se declaran agregando la palabra ***static*** en el encabezado del método.

Miembros *static*

En la clase *Fecha*:

```
private static int diasAnio;

public static int getDiasAnio(){
   return diasAnio;
}
public static void setDiasAnio(int d){
   diasAnio=d;
}
```

Otro dato miembro estático y constante:

```
   private static final int DIAS_MES=30;
```

Miembros *static*

```
public class MiClase {
    private static int contador = 0;

    public MiClase() {
        contador++;  // Incrementa el contador cada vez que se
                     // crea una instancia de la clase
    }
    public static int getContador() {
        return contador;
    }
}
```

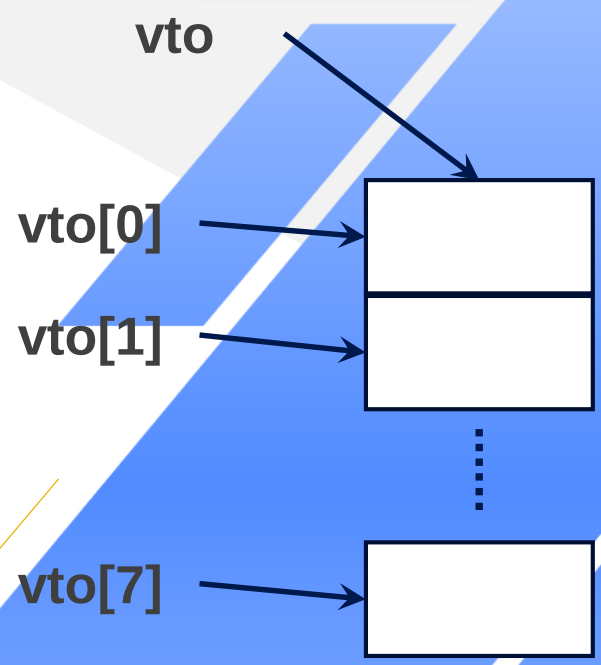
```
System.out.println("Contador: " + MiClase.contador);
     // Imprime la cantidad de objetos creados
```

Arrays de objetos

- Se puede crear un array de objetos de la misma forma que se crea un array de elementos de otro tipo.
- La única condición es que la clase debe tener un método constructor que pueda ser llamado sin argumentos para evitar errores.

Ejemplo: Fechas de vencimiento

```
// declaración y dimensión
// de un array de  objetos
Fecha vto[]=new Fecha[8];
```



Arrays de objetos

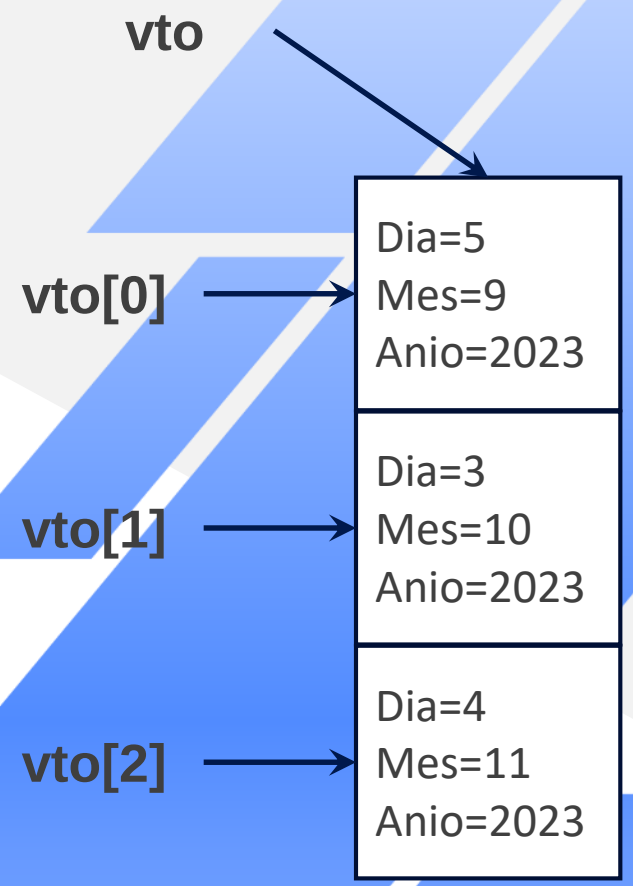
Ejemplo de inserción de objetos en el array:

```
for (int i=0; i<8; i++){
    vto[i] = new Fechas();
}
```

Se puede combinar la creación y la incorporación de objetos en un array en la misma operación.

Ejemplo para crear un array de 3 posiciones:

```
Fecha vto[] = { new Fecha(05,09,2023),
                new Fecha(03,10,2023),
                new Fecha(04,11,2023)
};
```



Arrays de objetos

- Cada posición del vector funciona como un objeto independiente.
- Para modificar el valor de un atributo aplicar el método set correspondiente sobre la posición del array donde se ubica el objeto a modificar.
  Ejemplo: cambiar el valor del mes del objeto de la 2da.posición del array:

```
vto[1].setMes(08);
```

- Para consultar el contenido del array, recorrerlo con ciclo for o con ciclo for avanzado.
  Con ciclo for:

```
for(int i=0; i<vto.length;i++){
    System.out.println("Dia: " + vto[i].getDia());
    System.out.println("Mes: " + vto[i].getMes()); }
```

  Con ciclo for avanzado:

```
for(Fecha v:vto){
    System.out.println("Dia: " + v.getDia());
    System.out.println("Mes: " + v.getMes()); }
```

Paquetes

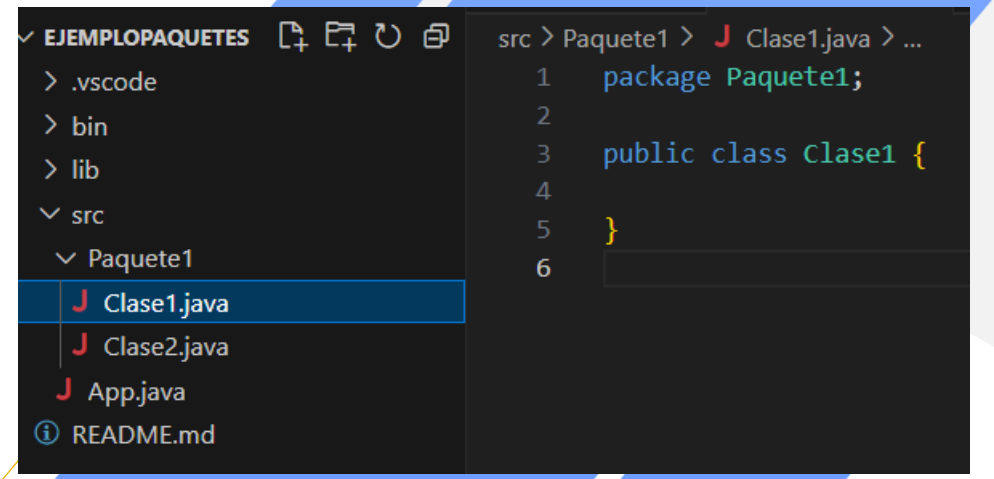
- Un paquete permite agrupar clases e interfaces.
- Para que una clase forme parte de un paquete debe indicarse la palabra reservada ***package*** y el nombre del paquete en la primera línea del programa:

```
package nombrePaquete;
```

- El nombre de un *package* puede constar de varios nombres unidos por puntos:

```
java.awt.event
```

- Todas las clases que forman parte de un paquete deben estar **en el mismo directorio**.
- Si un archivo fuente Java no contiene ningún *package*, se coloca en el paquete por defecto sin nombre.
  Es decir, **en el mismo directorio que el archivo fuente**.
- Todas las clases acceden por defecto al mismo paquete.



Paquetes

- Para utilizar clases de otros paquetes, deben ser cargadas con la palabra clave ***import***, indicando el nombre del paquete y nombre de clase. Se pueden cargar varias clases utilizando un asterisco

```
import java.Date;
import java.awt.*;
import MiPaquete;
```

- Solo se pueden acceder a las clases definidas como públicas.
- Entonces, hay tres tipos de clases:
  - *public*: son accesibles desde otras clases, directamente o por herencia, dentro del mismo paquete en el que se han declarado. Para acceder desde otros paquetes, primero tienen que ser importadas.
  - *abstract*: tiene al menos un método abstracto. No se instancia, sino que se utiliza como clase base para la herencia.
  - *final*: se declara como la clase que termina una cadena de herencia. No se puede heredar de una clase final.

Control de acceso o visibilidad

Visibilidad	Public	Protected	Private	Package
Propia Clase	*Si*	*Si*	*Si*	*Si*
Otra Clase del Paquete	*Si*	*Si*	*No*	*Si*
Otra Clase Fuera del Paquete	*Si*	*No*	*No*	*No*
Clase Derivada dentro del Paquete	*Si*	*Si*	*No*	*Si*
Clase Derivada fuera del Paquete	*Si*	*Si*	*No*	*No*