

UD02

Ejercicios



Este material está bajo una [Licencia Creative Commons Atribución-Compartir-Igual 4.0 Internacional](https://creativecommons.org/licenses/by-nc-sa/4.0/).
Derivado a partir de material de David Martínez Peña (<https://github.com/martinezpenya>).

1. **Actividades**
2. **Ejercicios**
3. **Fuentes de información**

1. Actividades

1. **Actividad 01** - Crear una clase llamada Temperatura con dos métodos:

- `celsiusToFahrenheit`. Convierte grados *Celsius* a *Fahrenheit*.

$$F = (1,8 * C) + 32$$

- `fahrenheitToCelsius`. Convierte grados *Fahrenheit* a *Celsius*.

$$C = \frac{F - 32}{1,8}$$

2. **Actividad 02** - A partir de la siguiente clase:

```
public class Moto {

    private int velocidad;

    Moto() {
        velocidad=0;
    }

}
```

Añade los siguientes métodos:

- `int getVelocidad`. Devuelve la velocidad del objeto moto.
- `void acelera(int mas)`. Permite aumentar la velocidad del objeto moto.
- `void frena(int menos)`. Permite reducir la velocidad del objeto moto.

3. **Actividad 03** - Crea una clase Rebajas con un método `descubrePorcentaje()` que descubra el descuento aplicado en un producto. El método recibe el precio original del producto y el rebajado y devuelve el porcentaje aplicado. Podemos calcular el descuento realizando la operación:

$$porcentajeDescuento = \frac{precioOriginal - precioRebajado}{precioOriginal}$$

4. **Actividad 04** - Realiza una clase finanzas que convierta dólares a euros y viceversa. Codifica los métodos `dolaresToEuros` y `eurosToDolares`. Prueba que dicha clase funciona correctamente haciendo conversiones entre euros y dólares. La clase tiene que tener:

- Un constructor `finanzas()` por defecto el cual establece el cambio Dólar-Euro en 1.02.
- Un constructor `finanzas(double cambio)`, el cual permitirá configurar el cambio Dólar-euro a una cantidad personalizada.

5. **Actividad 05** - Realiza una clase `minumero` que proporcione el doble, triple y cuádruple de un número proporcionado en su constructor (realiza un método para `doble`, otro para `triple` y otro para `cuádruple`). Haz que la clase tenga un método `main` y comprueba los distintos métodos.

6. **Actividad 06** - Realiza una clase número que almacene un número entero y tenga las siguientes características:

- Constructor por defecto que inicializa a 0 el número interno.

- Método `anade` que permite sumarle un número al valor interno.
- Método `resta` que resta un número al valor interno.
- Método `getValor`. Devuelve el valor interno.
- Método `getDoble`. Devuelve el doble del valor interno.
- Método `getTriple`. Devuelve el triple del valor interno.
- Método `setNumero`. Inicializa de nuevo el valor interno.

7. **Actividad 07** - Crea la clase peso, la cual tendrá las siguientes características:

- Deberá tener un atributo donde se almacene el peso de un objeto en kilogramos. En el constructor se le pasará el peso y la medida en la que se ha tomado ("Lb" para libras, "Li" para lingotes, "Oz" para onzas, "P" para peniques, "K" para kilos, "G" para gramos y "Q" para quintales).
- Deberá de tener los siguientes métodos:
 - `getLibras`. Devuelve el peso en libras.
 - `getLingotes`. Devuelve el peso en lingotes.
 - `getPeso`. Devuelve el peso en la medida que se pase como parámetro ("Lb" para libras, "Li" para lingotes, "Oz" para onzas, "P" para peniques, "K" para kilos, "G" para gramos y "Q" para quintales).
- Para la realización del ejercicio toma como referencia los siguientes datos:
 - 1 Libra = 16 onzas = 453 gramos.
 - 1 Lingote = 32,17 libras = 14,59 kg.
 - 1 Onza = 0,0625 libras = 28,35 gramos.
 - 1 Penique = 0,05 onzas = 1,55 gramos.
 - 1 Quintal = 100 libras = 43,3 kg.
- Crea además un método `main` para testear y verificar los métodos de esta clase.

8. **Actividad 08** - Crea una clase con un método `millasAMetros()` que toma como parámetro de entrada un valor en millas marinas y las convierte a metros. Una vez tengas este método escribe otro `millasAKilometros()` que realice la misma conversión, pero esta vez exprese el resultado en kilómetros. *Nota: 1 milla marina equivale a 1852 metros.*

9. **Actividad 09** - Crea la clase `Coche` con dos constructores. Uno no toma parámetros y el otro sí. Los dos constructores inicializarán los atributos `marca` y `modelo` de la clase. Crea dos objetos (cada objeto llama a un constructor distinto) y verifica que todo funciona correctamente.

10. **Actividad 10** - Implementa una clase `Consumo`, la cual forma parte del "ordenador de a bordo" de un coche y tiene las siguientes características:

- Atributos:
 - kilometros.
 - litros. Litros de combustible consumido.
 - vmed. Velocidad media.
 - pgas. Precio de la gasolina.
- Métodos:
 - `getTiempo`. Indicará el tiempo empleado en realizar el viaje.
 - `consumoMedio`. Consumo medio del vehículo (en litros cada 100 kilómetros).
 - `consumoEuros`. Consumo medio del vehículo (en euros cada 100 kilómetros).

No olvides crear un constructor para la clase que establezca el valor de los atributos. Elige el

11. **Actividad 11** - Para la clase anterior implementa los siguientes métodos, los cuales podrán modificar los valores de los atributos de la clase:

- `setKms`
- `setLitros`
- `setVmed`
- `setPgas`

12. **Actividad 12** - Un restaurante cuya especialidad son las patatas con carne nos pide diseñar un método con el que se pueda saber cuántos clientes pueden atender con la materia prima que tienen en el almacén. El método recibe la cantidad de patatas y carne en kilos y devuelve el número de clientes que puede atender el restaurante teniendo en cuenta que por cada tres personas, utilizan un dos kilos de patatas y un kilo de carne.

13. **Actividad 13** - Modifica el programa anterior creando una clase que permita almacenar los kilos de patatas y carne del restaurante. Implementa los siguientes métodos:

- . `public void addCarne(int x)` . Añade x kilos de carne a los ya existentes.
- . `public void addPatatas(int x)` . Añade x kilos de patatas a los ya existentes.
- . `public int getComensales()` . Devuelve el número de clientes que puede atender el restaurante (este es el método del ejercicio anterior).
- . `public double getCarne()` . Devuelve los kilos de carne que hay en el almacén.
- . `public double getPatatas()` . Devuelve los kilos de patatas que hay en el almacén.

14. **Actividad 14** - Crear un clase llamada `Proveedor` con las siguientes propiedades:

- `CIF`
- `nombreEmpresa`
- `descripcion`
- `sector`
- `direccion`
- `telefono`
- `poblacion`
- `codPostal`
- `correo`

Crear para la clase `Proveedor` los métodos:

- Constructor que permite crear una instancia con los datos de un proveedor.
- Métodos `get (getters)`.
- Métodos `set (setters)`.
- Método `verificaCorreo` que devuelve true si la dirección de correo contiene @.
- Método que muestre todos los datos del proveedor.

Crear una clase principal `main` ejecutable que:

- Cree una instancia del objeto `Proveedor` llamado `proveedor`.
- Cambie el sector del `proveedor`.
- Muestre el sector del `proveedor`.
- Verifique si el correo es válido.
- Muestre todos los datos del `proveedor`.

15. **Actividad 15** - Crear una clase llamada `Producto` con las siguientes propiedades:

- `codProducto`
- `nombreProducto`

- `categoria`
- `peso`
- `precio`
- `stock`

Crear para la clase `Producto` los siguiente métodos:

- `Producto`: Permite crear una instancia con los datos de un producto.
- `aumentaStock`: Permite aumentar el stock de unidades del producto. Se le pasa el dato de unidades que aumentamos.
- `disminuyeStock`: Permite disminuir el stock de unidades del producto. Se le pasa el dato de unidades que disminuimos.
- `ivaProducto`: Permite calcular el IVA aplicado al precio del producto. Se le pasa el dato del porcentaje de IVA.
- `mostrarDatos`: Muestra los datos del producto.

Crear una clase principal `main` ejecutable que:

- Crear dos instancias de la clase `Producto` llamadas `productoHardware` y `productoSoftware`.
- Mostrar los datos de los dos objetos `Producto` que hemos creado.
- Aumenta el stock de unidades del `productoHardware` en 12 unidades.
- Disminuir el stock de unidades del `productoSoftware` en 5 unidades.
- Calcula el IVA de los dos objetos `Producto` que hemos creado.
- Mostrar los datos de los dos objetos `Producto`, así como sus importes de IVA y los precios finales de cada una de las instancias.

16. **Actividad 16** - Crear una clase llamada `Password` con las siguientes características:

- Propiedades: `clave`.
- Los métodos que implementa serán:
 - Un constructor sin parámetros que generará una clave aleatoria con longitud 8.
 - Un constructor que recibirá por parámetro un `int` que le indicará la longitud de la clave a generar.
 - `generarClave()`: genera la clave del objeto con la longitud que tenga.
 - Método `get` para clave.
 - Método `set` para clave.
- Crear una clase principal `main` que compruebe todos los métodos creados.

17. **Actividad 17** - Crea una clase llamada `Cuenta` que tendrá los siguientes atributos: `titular` y `cantidad` (puede tener decimales).

Al crear una instancia del objeto `Cuenta`, el titular será obligatorio y la cantidad es opcional. Crea dos constructores que cumplan lo anterior, es decir debemos crear dos métodos constructores con el mismo nombre que será el nombre del objeto.

Crea sus métodos `get`, `set` y el método `mostrarDatos` que muestre los datos de la cuenta. Tendrá dos métodos especiales:

- `ingresar(double cantidad)`: se ingresa una cantidad a la cuenta, si la cantidad introducida es negativa, no se hará nada.
- `retirar(double cantidad)`: se retira una cantidad a la cuenta, si restando la cantidad actual a la que nos pasan es negativa, la cantidad de la cuenta pasa a ser 0 retirando el importe máximo en función de la cantidad disponible en el objeto.

- Crear una instancia del objeto Cuenta llamada `cuentaParticular1` con el nombre del titular.
- Crear una instancia del objeto Cuenta llamada `cuentaEmpresa1` con el nombre del titular y una cantidad inicial de dinero.
- Mostrar el titular de la instancia `cuentaParticular1`.
- Mostrar el saldo de la instancia `cuentaEmpresa1`.
- Ingresar 1000 € en la instancia `cuentaParticular1`.
- Retirar 500 € en la instancia `cuentaEmpresa1`.
- Mostrar los datos de las dos instancias del objeto `Cuenta`.

18. **Actividad 18** - Crea una clase llamada `Libro` que guarde la información de cada uno de los libros de una biblioteca. La clase debe guardar las siguientes propiedades:

- `título`
- `autor`
- `editorial`
- `número de ejemplares totales`
- `número de prestados`

La clase contendrá los siguientes métodos:

- Constructor por defecto.
- Constructor con parámetros.
- Métodos Setters/getters.
- Método `prestamo` que incremente el atributo correspondiente cada vez que se realice un préstamo del libro. No se podrán prestar libros de los que no queden ejemplares disponibles para prestar. Devuelve `true` si se ha podido realizar la operación y `false` en caso contrario.
- Método `devolucion` que decremente el atributo correspondiente cuando se produzca la devolución de un libro. No se podrán devolver libros que no se hayan prestado. Devuelve `true` si se ha podido realizar la operación y `false` en caso contrario.
- Método `perdido` que decremente el atributo número de ejemplares por pérdida de ejemplar. No se podrán devolver libros que no tengan ejemplares. Devuelve `true` si se ha podido realizar la operación y `false` en caso contrario.
- Método `mostrarDatos` para mostrar los datos de los libros.

Crear una clase principal `main` ejecutable:

- Crear una instancia del objeto libro `libroInformatica1` con los datos de un libro.
- Consultar el título de la instancia `libroInformatica1`.
- Cambiar la editorial de la instancia `libroInformatica1` por Anaya.
- Realiza el préstamo de la instancia `libroInformatica1`.
- Realiza otro préstamo de la instancia `libroInformatica1`.
- Muestra los prestamos de la instancia `libroInformatica1`.
- Realiza la devolución de la instancia `libroInformatica1`.
- Muestra los prestamos de la instancia `libroInformatica1`.
- Gestiona la pérdida de un ejemplar de la instancia `libroInformatica1`.
- Muestra los ejemplares de la instancia `libroInformatica1`.
- Muestra todos los datos de la instancia `libroInformatica1`.

19. **Actividad 19** - Crear una clase llamada `Hospital` con las siguientes propiedades y métodos:

- Propiedades:

- `nombreHospital`
- `direccion`
- `telefono`
- `poblacion`
- `codPostal`
- `habitacionesTotales`
- `habitacionesOcupadas`

○ Métodos:

- Hospital: Permite crear una instancia con los datos de un hospital.
- Métodos get.
- Métodos set.
- Método `ingreso` que incrementa las habitaciones ocupadas. No puede realizarse el ingreso si las habitaciones ocupadas son iguales a las habitaciones totales del hospital. Devuelve `true` si se ha podido realizar el ingreso.
- Método `alta` que decrementa las habitaciones ocupadas. No puede realizarse el alta si las habitaciones ocupadas son 0. Devuelve `true` si se ha podido realizar el alta.
- Método que muestre todos los datos del hospital.

○ Crear una clase principal `main` ejecutable que:

- Cree una instancia de la clase `Hospital` llamada `hospitalRibera`.
- Cambie el número de habitaciones de la instancia `hospitalRibera`.
- Muestre el número de habitaciones de la instancia `hospitalRibera`.
- Realiza un ingreso de la instancia `hospitalRibera`.
- Muestra las habitaciones ocupadas de la instancia `hospitalRibera`.
- Realiza un alta de la instancia `hospitalRibera`.
- Muestra las habitaciones ocupadas de la instancia `hospitalRibera`.
- Muestre todos los datos de la instancia `hospitalRibera`.

20. **Actividad 20** - Crear un clase llamada `Medico` con las siguientes propiedades y métodos:

○ Propiedades:

- `codMedico`
- `nombre`
- `apellidos`
- `dni`
- `direccion`
- `telefono`
- `poblacion`
- `codPostal`
- `fechaNacimiento`
- `especialidad`
- `sueldo`

○ Métodos:

- Medico: Permite crear una instancia con los datos de un médico.
- Métodos get. Recuperan datos de la instancia del objeto.
- Métodos set. Asignan datos a la instancia del objeto.
- `retencionMedico`: Permite calcular la retención aplicada al sueldo del médico. Se

- Crear una clase principal `main` ejecutable que:
 - Crear dos instancias de la clase `Medico` llamados `medicoDigestivo` y `medicoTraumatologo`.
 - Cambia el sueldo del `medicoTraumatologo`.
 - Muestra el sueldo del `medicoTraumatologo`.
 - Cambia el dni del `medicoDigestivo`.
 - Muestra el dni del `medicoDigestivo`.
 - Calcula la retención de las dos instancias de la clase `Medico` que hemos creado.
 - Mostrar los datos de las dos instancias de la clase `Medico` que hemos creado, así como las retenciones y los sueldos finales de cada una.

2. Ejercicios

Estos ejercicios utilizan la interfaz gráfica a la que dedicaremos más tiempo hacia finales de curso. De momento con entender algunos conceptos muy básicos de como dibujar elementos gráficos en una ventana podemos intentar resolverlos usando los conceptos de objetos, clases, herencia, métodos, etcétera que hemos visto en teoría.

El primero está resuelto y comentado para que te ayude a resolver el resto por tu cuenta o con la ayuda del docente.

1. **Ejercicio 1 - (LlenarConCirculo)** Crear una pizarra cuadrada y dibujar en ella un círculo que la ocupe por completo.

```
//importaciones necesarias para los ejercicios, no necesitas más.
import javax.swing.JFrame;
import javax.swing.JPanel;
import java.awt.Color;
import java.awt.Graphics;

/*
    Necesitamos que nuestra clase LlenarConCirculo herede de JPanel para poder
    pintar en su interior.
*/
public class LlenarConCirculo extends JPanel {

    @Override
    public void paint(Graphics g) {
        //Fijamos el color que tendrá la figura
        g.setColor(Color.RED);
        /*
            Dibujamos un ovalo relleno fijando las 4 esquinas que lo delimitan:
            - x1, y1, x2, y2
            En nuestro caso además hacemos uso de la función reflexiva
            this.getWidth() y this.getHeight() para conocer la anchura y altura
            (respectivamente) de nuestra ventana.
        */
        g.fillOval(0, 0, this.getWidth(), this.getHeight());
        /*
            Otras funciones disponibles para dibujar son:
            - fill3DRect(int x, int y, int width, int height, boolean raised)
              Paints a 3-D highlighted rectangle filled with the current color.
            - fillArc(int x, int y, int width, int height, int startAngle, int
arcAngle)
              Fills a circular or elliptical arc covering the specified
rectangle.
            - fillOval(int x, int y, int width, int height)
              Fills an oval bounded by the specified rectangle with the current
color.
            - fillPolygon(int[] xPoints, int[] yPoints, int nPoints)
              Fills a closed polygon defined by arrays of x and y coordinates.
            - fillPolygon(Polygon p)
              Fills a closed polygon defined by the specified Polygon object.
        */
    }
}
```

```

        context's current color.
    - fillRect(int x, int y, int width, int height)
      Fills the specified rectangle.
    - fillRoundRect(int x, int y, int width, int height, int arcWidth,
int arcHeight)
      Fills the specified rounded corner rectangle with the current
color.
    - fill3DRect(int x, int y, int width, int height, boolean raised)
      Paints a 3-D highlighted rectangle filled with the current color.
    - fillArc(int x, int y, int width, int height, int startAngle, int
arcAngle)
      Fills a circular or elliptical arc covering the specified
rectangle.
    - fillOval(int x, int y, int width, int height)
      Fills an oval bounded by the specified rectangle with the current
color.
    - fillPolygon(int[] xPoints, int[] yPoints, int nPoints)
      Fills a closed polygon defined by arrays of x and y coordinates.
    - fillPolygon(Polygon p)
      Fills the polygon defined by the specified Polygon object with
the graphics
      context's current color.
    - fillRect(int x, int y, int width, int height)
      Fills the specified rectangle.
    - fillRoundRect(int x, int y, int width, int height, int arcWidth,
int arcHeight)
      Fills the specified rounded corner rectangle with the current
color.
    -
  */
}

public static void main(String[] args) {
    //Creamos una nueva ventana
    JFrame MainFrame = new JFrame();
    //Fijamos su tamaño en 300px de ancho por 300px de alto
    MainFrame.setSize(300, 300);
    //Creamos el objeto que vamos a dibujar con el método paint()
    LlenarConCirculo circlePanel = new LlenarConCirculo();
    //Añadimos el objeto recién creado a la ventana
    MainFrame.add(circlePanel);
    //Hacemos visible la ventana (con el dibujo)
    MainFrame.setVisible(true);
}
}

```

Este es el esquema básico que necesitas para resolver todos los ejercicios planteados:

```

//importaciones necesarias para los ejercicios, no necesitas más.
import javax.swing.JFrame;
import javax.swing.JPanel;
import java.awt.Color;
import java.awt.Graphics;

```

```

        pintar en su interior.
    */
    public class TuClaseEjercicio extends JPanel {

        @Override
        public void paint(Graphics g) {
            // INSERTA TU CÓDIGO AQUÍ!!! <--
            //Fijamos el color que tendrá la figura
            //Dibuja la/s figura/s que te pide el ejercicio
        }

        public static void main(String[] args) {
            //Creamos una nueva ventana
            JFrame MainFrame = new JFrame();
            //Fijamos su tamaño en 300px de ancho por 300px de alto
            MainFrame.setSize(300, 300);
            //Creamos el objeto que vamos a dibujar con el método paint()
            LlenarConCirculo tuDibujo = new LlenarConCirculo();
            //Añadimos el objeto recién creado a la ventana
            MainFrame.add(tuDibujo);
            //Hacemos visible la ventana (con el dibujo)
            MainFrame.setVisible(true);
        }
    }
}

```

2. **Ejercicio 2** - (*LlenarConRectangulo*) Crear una pizarra de tamaño aleatorio y dibujar en ella un rectángulo que la ocupe por completo.
3. **Ejercicio 3** - (*MitadYMitad*) Crear una pizarra de tamaño aleatorio y dibujar un rectángulo ROJO que ocupe la mitad izquierda y uno VERDE que ocupe la mitad derecha.
4. **Ejercicio 4** - (*Dos partes*) Crear una pizarra de tamaño aleatorio y dibujar un rectángulo ROJO que ocupe la parte superior (25% de la altura) y uno VERDE que ocupe la parte inferior (75% restante).
5. **Ejercicio 5** - (*CentrarFiguras*) Crear una pizarra de tamaño aleatorio. Dibujar en el centro un cuadrado de lado 100 y un círculo de radio 25.
6. **Ejercicio 6** - (*RadioAleatorioCentrado*) Crear una pizarra de tamaño aleatorio. Dibujar en centro de la pizarra un círculo de radio aleatorio (entre 50 y 200 pixels de radio).
7. **Ejercicio 7** - (*RadioAleatorio*) Crear una pizarra de tamaño aleatorio. Dibujar en la esquina superior izquierda un círculo de radio aleatorio (entre 50 y 200).

3. Fuentes de información

- [Wikipedia](#)
- [Programación \(Grado Superior\) - Juan Carlos Moreno Pérez \(Ed. Ra-ma\)](#)
- Apuntes IES Henri Matisse (Javi García Jimenez?)
- Apuntes AulaCampus
- [Apuntes José Luis Comesaña](#)
- [Apuntes IOC Programació bàsica \(Joan Arnedo Moreno\)](#)
- [Apuntes IOC Programació Orientada a Objectes \(Joan Arnedo Moreno\)](#)