FUNDAMENTOS DE PROGRAMACIÓN (IW-SI)

Ejercicios del TEMA 4: Clases y Objetos

Ejercicio 1 (OBJETOS)

Dada la clase *Intervalo* del ejemplo de teoría y las siguientes declaraciones de objetos:

```
Intervalo intervalo0 = null;
Intervalo intervalo1 = null;
Intervalo intervalo2 = new Intervalo(-5, 5);
Intervalo intervalo3 = new Intervalo(0, 7.77);
Intervalo intervalo4 = intervalo3;
Intervalo intervalo5 = new Intervalo(-5, 5);
```

Indicar que se mostraría por pantalla para cada una de las siguientes instrucciones:

```
System.out.println(intervalo0 == intervalo1);

System.out.println(intervalo2 == intervalo5);

System.out.println(intervalo3 == intervalo4);

System.out.println(intervalo0.getMaximo());

System.out.println(intervalo3.getMaximo());
System.out.println(intervalo4.getMaximo());

intervalo3.setMaximo(88);
System.out.println(intervalo3.getMaximo());

System.out.println(intervalo4.getMaximo());
```

Ejercicio 2 (OBJETOS)

Sea la clase *Intervalo* vista en las transparencias de teoría, y cuyo código se puede encontrar en moddle, en el material del tema 4. Se considera la siguiente vista pública de dicha clase:

```
class Intervalo {
   public Intervalo(double min, double max)
   public Intervalo()
   public double getMinimo()
   public double getMaximo()
   public void setMinimo(double min)
   public void setMaximo(double max)
   public double longitud()
   public double puntoMedio()
   public void desplazar(double desplazamiento)
   public String toString()
   public void escalar (double escala)
   public boolean incluye(double punto)
}
```

Definir en una nueva clase *PruebasIntervalo* los siguientes dos métodos estáticos para probar el funcionamiento de la clase Intervalo:

1. Método ***temperaturasDia***, que lea del teclado dos valores double que representan la temperatura mínima y máxima de un día, y devuelva como resultado un intervalo formado por dichos valores. En caso de que la temperatura máxima sea menor que la mínima, el método volverá a pedir los datos hasta que sean correctas.
   ```
   public static Intervalo temperaturasDia()
   ```
2. Método ***main*** que utilice el método *temperaturasDia* para leer las temperaturas máximas y mínimas de un total de 7 días, calcule la temperatura media de los 7 días y determine cuántos días han sido templados. Se considera que un día ha sido templado si en algún momento se han alcanzado los 15 grados.

Ejercicio 3 (OBJETOS)

Sea la clase *Avión*, cuyo código se puede encontrar en moddle, en el material del tema 4. Se considera la siguiente vista pública de dicha clase:

```
public Avión(String marca, String modelo, String matrícula,
       int númeroDeAsientos, double máximoCombustible)
public double getCapacidad()
public double getCombustible()
public void llenarDepóstio()
public void vaciarDepósito()
public void gastarCombustible(double númeroLitros)
public void repostarCombustible(double númeroLitros)
public String toString()
```

Definir una nueva clase *PruebasAvión* que desarrolle los siguientes métodos estáticos que prueben el correcto funcionamiento de la clase *Avión*

1. Una función estática ***leerAvión*** que reciba como argumento un objeto *Scanner*, solicite y lea desde teclado los datos necesarios para crear un avión (marca, modelo, matrícula, número de asientos y combustible máximo) y genere y devuelva un objeto de tipo *Avión* a partir de los datos leídos. Se deberá comprobar que tanto el número de asientos como el combustible máximo sean positivos, volviendo a pedir el dato en caso contrario.
2. Una función estática ***traspasoCombustible*** que reciba como argumentos dos objetos de tipo *Avión* y que, en base a la cantidad de combustible de cada avión, traspase combustible desde el que tiene mayor cantidad de combustible en su depósito hasta el que tiene menor cantidad, hasta que ambos dispongan de la misma cantidad. En caso de que el que tenga menor cantidad de combustible tenga una capacidad menor que la que se alcanzaría si ambos compartieran el mismo combustible, se rellenará éste hasta su máxima capacidad, extrayendo la cantidad justa del otro Avión. La función deberá devolver un *double* indicando la cantidad traspasada desde el primer Avión al segundo (número positivo) o la cantidad recibida por el primer avión desde el segundo (número negativo).
3. Un método main para comprobar las funcionalidades creadas. Este método creará tres aviones usando la función *leerAvión*. A continuación, rellenará los depósitos de los dos primeros aviones, y se pedirá una cantidad de combustible para extraer del segundo avión. Se imprimirá entonces por pantalla los datos de los tres aviones. Luego se realizará un traspaso desde el primero al segundo, mostrando por pantalla la cantidad traspasada entre ambos; y a continuación del primer avión al tercero, también mostrando la cantidad traspasada. Finalmente, se volverá a mostrar por pantalla los datos de los tres aviones tras estos traspasos.

   La salida del método main si se introduce como capacidades máximas de combustible 100, 70 y 50 litros respectivamente, y se indica que se extraerán 20 litros del segundo avión, es la siguiente:

```
Antes del traspaso:
Boeing747(343LKD3L): 200 asientos, 100.0 kg de fuel
AirbusA320(274OPT5A): 180 asientos, 50.0 kg de fuel
Boeing777(423LTG9K): 205 asientos, 0.0 kg de fuel
Se traspasan 20.0 litros del avión 1 al avión 2.
Se traspasan 40.0 litros del avión 1 al avión 3.
Después del traspaso:
Boeing747(343LKD3L): 200 asientos, 40.0 kg de fuel
AirbusA320(274OPT5A): 180 asientos, 70.0 kg de fuel
Boeing777(423LTG9K): 205 asientos, 40.0 kg de fuel
```

Ejercicio 4 (CLASES Y OBJETOS)

Codifique una clase ***NumeroComplejo*** para representar números complejos y realizar operaciones con complejos.

- Atributos:
  - **real**: representa la parte real del número complejo, de tipo *double*.
  - **imaginario**: representa la parte imaginaria del número complejo, de tipo *double*.
- Constructores**:**
  - **public NumeroComplejo()**: Inicializa el número a 0 (0 + 0i).
  - **public NumeroComplejo(double real, double imaginario)**: Inicializa el número con los argumentos.
- Operaciones:
  - **public void setReal(double real)**: Cambia el valor de la parte real del número complejo.
  - **public void setImaginario(double imaginario)**: Cambia del valor de la parte imaginaria del número complejo.
  - **public double getReal()**: Devuelve la parte real del número complejo.
  - **public double getImaginario()**: Devuelve la parte imaginaria del número complejo.
  - **public NumeroComplejo suma(NumeroComplejo otro)**: Calcula y devuelve un NumeroComplejo resultado de la suma del número complejo actual y el número complejo pasado como argumento.
  - **public NumeroComplejo resta(NumeroComplejo otro)**: Calcula y devuelve un NumeroComplejo resultado de la resta del número complejo actual menos el número complejo pasado como argumento.
  - **public NumeroComplejo multiplica(NumeroComplejo otro)**: Calcula y devuelve un NumeroComplejo resultado de la multiplicación del número complejo actual y el número complejo pasado como argumento.
  - **public NumeroComplejo divide(NumeroComplejo otro)**: Calcula y devuelve un NumeroComplejo resultado de la división del número complejo actual entre el número complejo pasado como argumento.
  - **public NumeroComplejo conjugado()**: devuelve un nuevo número complejo que es el conjugado del complejo actual.
  - **public double modulo()**: devuelve el módulo o tamaño del número complejo.
  - **public String toString()**: Imprime un número complejo en formato *a+bi*, donde a es la parte real y b la imaginaria.

Escriba también una clase ***PruebaComplejos*** con un método *main* que cree dos números complejos y pruebe las distintas operaciones implementadas con estos números.

Ejercicio 5 (CLASES Y OBJETOS)

Codifique las siguientes clases:

Clase Punto:

- Clase que representa la posición de un punto en un plano.
  - Atributos privados:
    - **x**: Abscisa del punto, de tipo *double.*
    - **y**: Ordenada del punto, de tipo *double*.
  - Constructores:
    - **Punto()**: Inicializa el punto con la posición (0.0, 0.0)
    - **Punto(double x, double y)**: Inicializa el punto con la posición (x, y)
  - Operaciones:
    - **String toString()**: Devuelve un *String* con la posición del punto en el siguiente formato: (*x, y)*
    - **double distancia(Punto punto)**: Calcula la distancia entre punto al que se envía el mensaje y el punto recibido como parámetro. La distancia se calcula como:

$$distancia\ = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Clase Circulo:

- Clase que representa un círculo en el plano.
  - Atributos privados:
    - **centro**: Centro del círculo, de tipo *Punto*.
    - **radio**: Radio del círculo, de tipo *double*.
  - Constructores:
    - **Circulo(Punto centro, double radio)**: Inicializa el círculo con el centro y el radio recibidos como parámetros.
  - Operaciones:
    - **String toString()**: Devuelve un *String* con los datos del círculo en el siguiente formato: *[centro: (x, y), radio: r]*
    - **boolean contiene(Punto punto)**: Devuelve *true* si el punto recibido como parámetro está dentro del círculo, y *false* si está fuera. Se considera que el punto está dentro del círculo si la distancia entre el punto y el centro del círculo es menor o igual que el radio.

Clase PruebasCírculo:

- con un método ***main*** para comprobar que las dos clases anteriores funcionan correctamente:
  - Se crearán dos puntos, uno con cada constructor.
  - Se mostrarán por pantalla los dos puntos utilizando el método *toString*.
  - Se calculará la distancia entre los dos puntos y se mostrará por pantalla.
  - Se creará un círculo.
  - Se mostrará por pantalla el círculo utilizando el método *toString*.
  - Se comprobará si alguno de los puntos definidos está dentro del círculo, y en caso afirmativo, se mostrarán por pantalla el círculo y el punto con el formato:
    El punto (x, y) esta dentro del circulo [Centro: (x, y), Radio: r].

Ejercicio 6 (CLASES Y OBJETOS)

Codifique la clase **Fecha**, que encapsule las variables enteras usadas para definir la fechade un día concreto.

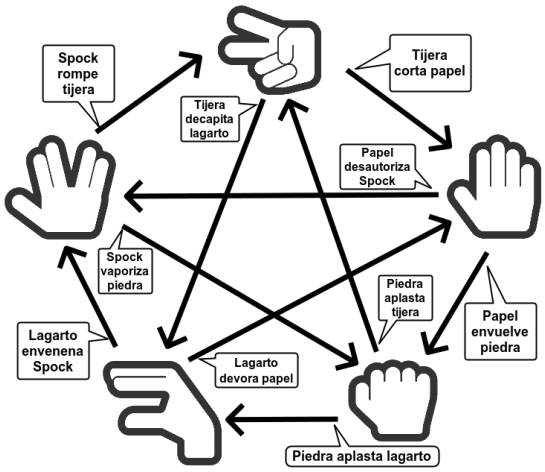
- Atributos:
  - **dia**: dia de la fecha, de tipo *int*, con valor entre 1 y 31.
  - **mes**: mes de la fecha, de tipo *int*, con valor entre 1 y 12.
  - **año**: año de la fecha, de tipo *int*, superior a 1900
- Constructores:
  - **Fecha(int dia, int mes, int año)**. En caso de que la fecha sea incorrecta, inicializará los atributos con 1/1/1900 y dará un mensaje de error. Para comprobar si la fecha es correcta, utilizará el método estático comprobarFecha, que se propone seguidamente.
- Métodos de clase (estáticos)
  - **boolean bisiesto(int anyo)**: Devuelve *true* si el año recibido como parámetro es un año bisiesto. Se considera un año bisiesto si:
    - Es múltiplo de 4 no siendo múltiplo de 100 (2020 es bisiesto, pero 2100 no es bisiesto).
    - O bien, es múltiplo de 400 (2000 y 2400 sí son bisiestos)
  - **boolean comprobarFecha(int dia, int mes, int agno):** Función que comprueba si una fecha es correcta. Una fecha es correcta cuando los valores de dia, mes y año se encuentra dentro de sus correspondientes rangos, y además se tiene en cuenta los rangos específicos para cada día de cada mes, incluyendo la consideración de años bisiestos para dar el 29 de febrero como válido.
- Métodos de instancia:
  - **Getters** de todos los atributos.
  - **Fecha copia()**: que devuelve una copia de la fecha actual (un nuevo objeto).
  - **void siguienteDia()**: que cambiará el estado de la fecha para evolucionar al siguiente día. Hay que tener en cuenta que se puede acabar el mes actual y que se puede acabar el año actual (se recomienda utilizar el método estático fechaCorrecta).
  - **boolean posteriorA(Fecha otraFecha)**: que devuelve true si la fecha actual es posterior o no a la fecha recibida como parámetro.
  - **boolean enBisiesto()**: que determina si la fecha actual se encuentra en un año bisiesto o no.
  - **String imprimir()**: que visualiza los datos de la fecha actual con el formato *dd/mm/yyyy*.

Codifique también una clase **PruebasFecha** en la que pruebe el correcto funcionamiento de la clase anterior:

- Crear una fecha comienzoCurso con el día 11/9/2023 y visualizarla
- Realizar una copia de la fecha anterior en cursoAvanzado.
- Comprobar si cursoAvanzado es posterior a comienzoCurso. (se obtendrá false)
- Avanzar 150 días la fecha cursoAvanzado y visualizarla (se obtendrá 7/2/2024
- Comprobar nuevamente si cursoAvanzado es posterior ComienzoCurso (se obtendrá true).
- Comprobar si la fecha de cursoAvanzado está en un año bisiesto (resultado true)
- Comprobar si el año 2100 es un año bisiesto (resultado false)

Ejercicio 7 (ENUM)

Desarrolle un programa que permita jugar al juego extendido del Jankenpon, en el que hay cinco jugadas posibles: Tijeras, Papel, Piedra, Lagarto o Spock.



En un archivo JuegoJankepon.java cree una clase pública *JuegoJankepon* y un enumerado no público Jankenpon con las opciones en este orden indicado anteriormente. En la clase se creará un método *main* que le pedirá al usuario (el Jugador) que introduzca su jugada, y esta se almacenará en una variable de tipo *Jankenpon*. A continuación, usando la función Math.random(), el programa generará una jugada aleatorio, y se comparará con la jugada generada por el jugador para comprobar quien es el ganador. Se mostrará por pantalla "Empate", "Gana el jugador" o "Gana el ordenador" según el resultado de esta comparación. A continuación se muestra un ejemplo de ejecución:

```
Su jugada (tijeras, papel, piedra, lagarto o Spock): Spock
Has jugado SPOCK
El ordenador ha jugado PAPEL
Gana el ordenador
```

PISTA: Con el orden definido del tipo enumerado *Jankenpon*, si la diferencia entre la jugada del jugador y la del ordenador es positiva (la jugada del jugador es posterior a la del ordenador) y par, o negativa e impar, gana el jugador. En caso contrario, ganará el ordenador.

Ejercicio 8 (STRING)

Desarrolle un programa que pida al usuario introducir por teclado un texto y seguidamente le pida una palabra. El programa mostrará por pantalla cuantas veces aparece esa palabra en el texto introducido. A continuación, se pedirá al usuario que introduzca otra palabra para reemplazar a la primera, y se mostrará el texto con las palabras cambiadas.

Ejercicio 9 (CLASES, ENUM Y STRING)

En un archivo *Fecha* cree una clase pública *Fecha* y un enumerado *Mes* que contendrá los nombres de los meses del año *ENERO*, *FEBRERO*,…, *DICIEMBRE*. En la clase *Fecha* almacene el día y el año como enteros, pero el mes como un valor del enumerado *Mes*. Además, defina los siguientes métodos

- **public Fecha(int día, String mes, int año)**: constructor de la clase, que deberá convertir el mes recibido como String en un tipo enumerado *Mes*.
- **public int compareTo (Fecha otra)**: Comparará la fecha del objeto con otra fecha, devolviendo un entero positivo si nuestra fecha es posterior que la otra, negativo en caso contrario y 0 si son iguales.
- **public String toString()**: Devuelve un *String* que contiene la fecha en formato texto "<día> de <Mes> de <año>", por ejemplo "15 de Abril de 1990".

Asimismo, se deberá crear una clase *PruebasFecha* con un método *main* que lea de teclado dos fechas en formato dd Nombre_Mes aaaa (por ejemplo, 29 Abril 1990), y cree los objetos *Fecha* correspondientes. A continuación mostrará ambas fechas por pantalla usando el método *toString* y las comparará indicando cual es la más reciente usando el método *compareTo*.

Ejercicios de examen de cursos pasados sobre este tema

El enunciado y la solución de estos ejercicios se encuentra en Moodle.

- Ejercicio 1. Segundo parcial 2022/23
- Ejercicio 1. Examen global 2022/23
- Ejercicio 2. Examen global 2022/23
- Ejercicio 3. Examen extraordinario 2022/23
- Ejercicio 2. Examen segundo parcial 2021/22

Ejercicio 4. Examen extraordinario 2021/22