| | | | | | |
|---|---|---|---|---|---|
| COLEXIO VIVAS S.L. | RAMA: | Informática | CICLO: | DAM | |
| | MÓDULO | Programación | | CURSO: | 1º |
| | PROTOCOLO: | Boletín de ejercicios | AVAL: | DATA: | |
| | AUTOR | Francisco Bellas Aláez (Curro) | | | |

# Boletín de introducción a la POO

## Ejercicio 1

Crea una clase Geometría con las siguientes **propiedades**:

- figura: Será un boolean. Pública. Si esta propiedad tiene el valor true, el objeto estará representando un rectángulo. Por el contrario si tiene el valor false, representa un triángulo.
- altura, base: ambas reales. Privadas con set y get. ¿Se te ocurre alguna comprobación intresante qu ehacer en el set?

Tendrá los siguientes **constructores**:

- Uno vacío (sin parámetros) que inicializa los campos a triángulo (figura a false) y tanto la base como la altura a 2.
- Uno con parámetros base y altura a partir de los que inicializa un triángulo con dicha base y altura que se les pasa como parámetro.
- Un tercero con tres parámetros para inicializar las tres propiedades.

Y los siguientes **métodos**:

- area: Función que devuelve el área de la figura (**base*altura** en el rectángulo y **base*altura/2** en el triángulo). Recuerda que debes usar la base y la altura del propio objeto, no tiene ningún parámetro.
- perimetro: Función que devuelve el perímetro de la figura. Tampoco tiene parámetros. En el caso del triángulo supón que es triángulo rectángulo, por lo que los lados serán la base, la altura y la diagonal (hipotenusa) que calculas en el siguiente método.
- diagonal: Función que devuelve el tamaño de la diagonal del rectángulo o la hipotenusa del triángulo rectángulo (se calcula igual en ambos casos).
- tipo: función que devuelve el string triángulo o rectángulo dependiendo de figura.

En otra clase sitúa el programa principal (main) y crea dos objetos: uno que represente un rectángulo a partir del constructor y otro un triángulo a partir de base y altura que decida el usuario usando setters y el constructor vacío para inicializar. Luego muestra claramente los datos de cada uno.

| | | | | | |
|---|---|---|---|---|---|
| COLEXIO VIVAS S.L. | RAMA: | Informática | CICLO: | DAM | |
| | MÓDULO | Programación | | CURSO: | 1º |
| | PROTOCOLO: | Boletín de ejercicios | AVAL: | DATA: | |
| | AUTOR | Francisco Bellas Aláez (Curro) | | | |

## Ejercicio 2

Se desea crear una clase que permita gestionar fechas de forma cómoda. Se establece la siguiente definición de la clase Fecha:

**Atributos**: Día, mes y año. Son enteros y privados.

- set/get de cada atributo. Si se introduce un día fuera del rango 1-31 guardará 1. Si se introduce un mes fuera del rango 1-12 se guardará 1. No es necesario comprobar el mes para el número de días ni si el año es bisiesto.

**Constructores**:

- Un constructor vacío (sin parámetros que inicialice todo a valores coherentes con una fecha)
- Sobrecargado otro que inicialice las tres propiedades a los parámetros.

**Métodos**:

- *fechaFormateada*: Método que devuelve la fecha en formato cadena. Dispone de un parámetro booleano que si está a true devuelve la fecha en formato numérico "día/mes/año" y si está a false lo hace en modo texto para el mes (por ejemplo "12 de marzo de 1997").
- *diferenciaFechas*: Función **estática** a la cual se le pasan **dos objetos tipo fecha** como parámetros y devuelve **solo** la diferencia de años entre ambas fechas (**int**).

El programa principal (que **no** estará en la clase fecha) pedirá dos fechas al usuario que guardará en sendos objetos. Mostrará las fechas en formato corto y largo y calculará la diferencia de años llamando a *diferenciaFechas*.

***(Opcional)*** Modifica diferenciaFechas para que calcule los días que van de una a otra. Se debe tener en cuenta meses de 30, 31, 28 y 29 días.

| | | | | | |
|---|---|---|---|---|---|
| COLEXIO VIVAS S.L. | RAMA: | Informática | CICLO: | DAM | |
| | MÓDULO | Programación | | CURSO: | 1º |
| | PROTOCOLO: | Boletín de ejercicios | AVAL: | DATA: | |
| | AUTOR | Francisco Bellas Aláez (Curro) | | | |

## Ejercicio 3: Proyecto Empresa I

Crear una clase denominada **Empleado** con los campos privados: nombre (String), apellidos (String), edad (int), dni (String), salario anual (double), irpf en % (double. El dato de % se usa para cálculos y para presentación al usuario, lógicamente en un double se guarda solo el número.)

Debe incluir set/get para cada propiedad salvo para IRPF que **sólo** tendrá get pues cambiará solo cuando cambie el salario.

Por tanto salario debe ser una propiedad de forma que si es modificada, debe de cambiar el IRPF. Concretamente si el salario es menos de 6000 euros, el IRPF será del 7.5, si está entre 6000 y 30000 será del 15 y si es mayor que 30000 euros, el IRPF será del 20.

Debe haber dos constructores. Uno vacío (sin parámetros) que inicialice los datos String a cadena vacía "" y los numéricos a 0. El otro constructor inicializa las propiedades a partir de parámetros. Usa los set para las inicializaciones. Piensa si hay alguna propiedad que no deba ir como parámetro.

Un último método devolverá la cantidad de dinero que se lleva hacienda (usando el IRPF y el salario).

En una clase aparte denominada **IUEmpleado** se hará el interface de usuario (IU) para esta clase. También se denomina vista. Esta clase dispondrá como propiedad pública única un objeto del tipo Empleado la cual se inicializa a través de un paráemtro tipo Empleado en el constructur y dispone de los siguientes métodos:

- Se realizará un método denominado **mostrar** que muestre los campos al usuario.
- Haz otro denominado **pedir** que permita la introducción de los mismos por parte del usuario mediante teclado. Ninguno de estos dos métodos tiene parámetros ni devuelve nada.
- Se sobrecarga el método de mostrar de forma que se le pasa un int que representa a cada campo: 1-Nombre y Apellidos (los dos juntos), 2-edad, 3-DNI, 4-Salario e IRPF (los dos juntos) y 5-Lo que se lleva hacienda. Mostrará solo el dato indicado en el parámetro.

Para probarlo en el programa principal (en clase aparte) crea un objeto de esta clase, rellena con datos que se le pidan al usuario y luego muéstralos. Muestra también lo que lleva hacienda.

## Ejercicio 4: Proyecto Empresa II

Nota: continuación del 3 pero **NO** se validan juntos.

Realiza en el mismo proyecto donde hiciste Empleado y IUEmpleado otra clase denominada **Directivo**. Dispone también de los campos privados nombre (String), apellidos (String), edad (int), dni (String) y añade un campo que indica el nombre del departamento que dirige (String) y otro de porcentaje de beneficios (real).

Todos son privados y tienen set/get, en concreto en el caso del departamento, cuando se guarde un nombre debe guardarlo en mayúsculas y cuando se lea dicho nombre se devolverá entre comillas dobles.

En el caso del porcentaje de beneficios el valor debe estar entre 0 y 100. Si se mete uno fuera de ese rango se establecerá la propiedad a 0.

Realiza un constructor que inicialice todos los campos y otro que no tenga parámetros.

Otro método al que se le pase el dinero global de beneficio que ha tenido la empresa y devuelva la ganancia del directivo que obtiene a partir de la propiedad porcentaje de beneficio y dicho beneficio empresarial (el parámetro). Si los beneficios son negativos devuelve 0.

Debe añadirse una clase denominada **IUDirectivo** que sea equivalente a la de Empeado pero con los datos de Directivo.

Para probarlo en el programa principal crea un objeto de esta clase, rellena con datos que se le pidan al usuario y luego muéstralos. Muestra también el dinero que se lleva de beneficios.

## Ejercicio 5: Proyecto Empresa III

Nota: continuación del 4, pero **NO** se validan juntos.

Finalmente y en el mismo proyecto que Empleado y Directivo crea una tercera clase denominada **Empresa** con un campo de ganancias, otro directivo y dos empleados.

Tendrá set/get para ganancias (puede ser negativo, pues indicaría pérdidas).

Hay dos constructores sobrecargados. El primero tiene 3 parámetros: uno tipo directivo y los otros dos tipo empleado. El segundo además de esos tres parámetros tiene uno más para ganancias.

En los constructores crea también los interfaces de usuario para los empleados

y para el directivo.

En el programa principal se crea un objeto tipo Directivo, dos tipo empleado (inicializando todos con parámetros en la llamada al constructor). Finalmente se crea un objeto de la clase Empresa con los objetos anteriores como parámetros y con ciertas ganancias.

*Nota: Una vez creado el objeto empresa no se permite usar los objetos directivo ni empleados que no estén dentro del propio objeto empresa.*

Luego se plantea un menú con las siguientes opciones:

- Ver datos empleados: Submenú que pregunta si se desean ver todos los datos de ambos empleados o solo un dato de ambos empleados (se usarán los métodos correspondientes). Este submenú tendrá un apartado salir. Este apartado debe estar en una función aparte.
- Ver datos directivo: Mostrará los datos del directivo incluido el beneficio final en euros.
- Modificar datos: Submenú que pregunta por el cambio en directivo o uno de los empleados. Este apartado debe estar en una función aparte.
- Pagar: Se disminuyen las ganancias de la empresa según lo que cobren los empleados. Se muestra el valor antes y después de pagar.
- Cobrar: Simplemente se pide una valor al usuario que incrementa en el campo ganancias. Se muestra el valor antes y después de pagar.
- Salir

## Ejercicio 6 (Opcional)

Realiza un juego conversacional sencillo en consola. El planteamiento es como el del tema anterior pero ahora puedes aprovechar el diseño orientado a objetos. Puedes tomar como referencia Zork (dejo abajo varios enlaces). Haz, eso sí, un mapa más pequeño y una historia más breve. Realiza por lo menos una clase habitación con, entre otros miembros, 4 propiedades que sean a su vez del tipo Habitación así puede desde una habitación apuntar hasta a 4 más (Norte, Sur, Este, Oeste). Si alguna no tiene salida la pones a null.

En la habitación ten propiedades para objetos y similares.

Juego en español:
http://iplayif.com/?story=http%3A//media.textadventures.co.uk/games/GQLrNEIS8k26ddJ2nco6ag/Zork.gblorb

Juego original:
http://textadventures.co.uk/games/play/5zyoqrsugeopel3ffhz_vq

Mapa: http://www.caad.es/sites/default/files/descargas/Juegos/Glulx/Map%20(Eng).jpg

Video demostrativo: https://www.youtube.com/watch?v=xzUagi41Wo0

## Ejercicio 7 (Opcional)

Se desea realizar un juego de tablero por turnos para luchar un jugador (que será un guerrero) contra un Orco (que será la CPU). Para ello se plantean las siguientes clases:

Clase Posición:

- Con propiedades x e y teniendo en cuenta que el origen es (1,1) en laparte superior izquierda.
- Constructor que inicializa una posición a partir de dos parámetros.
- Método moverA que mueve de forma absoluta a una posición.
- Método desplazar: que suma cierto desplazamiento a las coordenadas actuales.

Clase Guerrero:

- Tendrá las propiedades:
  - energia: un valor de 0 a 1000.
  - posición: objeto de la clase Posición
  - escudo: booleano que si está a true disminuirá el daño cuando sea atacado.
  - arma: char con los valores:
    - A: arco (ataca a 4 ó 5 casillas con poco daño)
    - E: espada (ataca a 1 ó 2 casillas con más daño)
- Y los métodos:
  - atacar: Como parámetro se le pasa un orco. Se le resta energía al Orco según el siguiente algoritmo:
    - Si tiene una Espada y está a distancia 1 ó 2 casillas se le resta 100±20 ó 50±10 respectivamente.
    - Si tiene un Arco y está a distancia 1, 2, 3, 4, 5 casillas se le

resta 50±5, 40±5, 30±5, 20±5, 10±5.

- El valor ±N se refiere a que al valor se le suma una cantidad aleatoria entre -N y +N.

- recuperarse: Ni se mueve ni ataca pero recupera cierta energía en ese turno (será un parámetro).

Se deja el constructor a decisión del programador.

Clase Orco:

- Tendrá las propiedades:
  - energia: un valor de 0 a 1000.
  - posición: objeto de la clase Posición
  - nivel: dificultad de derrotarle (nivel del juego)
- Y los métodos:
  - atacar: Como parámetro se le pasa un guerrero. Se le resta energía al guerrero según el siguiente algoritmo:
    - Si está a distancia 1 ó 2 casillas en nivel 1 se le resta 100±20 ó 50±10 respectivamente.
    - Cada nivel aumenta en 10 lo que el Orco resta.
    - Cada 5 niveles el orco puede atacar desde una casilla más lejana restando 50 menos que en la distancia anterior.
  - recuperarse: Ni se mueve ni ataca pero recupera (parámetro+10*Nivel) unidades de energía en ese turno.

a) Versión simple: En el programa principal crea objetos Orco y guerrero, sitúalos cerca y haz un juego de un turno.

b) Versión más completa: Para que el juego se pueda llevar a cabo se plantea una clase añadida:

Clase Escenario:

-Propiedades:

- ancho y alto (enteros)
- terreno: char que indica si es Fango (F) lo que beneficia al Orco, campo (C) que beneficia al guerrero o montaña (M) que no beneficia a nadie.
- Un guerrero que será el jugador

- un orco que será el ordenador. El orco y el guerrero se inicializan a través de sendos parámetros que se le pasan al constructor.

– Los métodos:

- turnoJugador: Se le pide la acción al jugador (Mover, Atacar o Recuperarse). Devuelve true si gana. Dependiendo de la acción:
  - Mover: Dependiendo del terreno se puede mover una (Fango), dos (Montaña) o tres (campo) casillas en cualquier dirección.
  - Atacar: Según lo visto en la clase guerrero.
  - Recuperarse: recupera 50, 40 o 30 unidades de energía según el terreno.
- turnoCPU: Devuelve true si gana. La Cpu realiza una acción según, al menos, los puntos siguientes (añade más si quieres darle más "inteligencia al juego":
  - Si se puede atacar se ataca. Si no se puede atacar y hay bastante diferencia de energía a favor del Orco, se mueve hacia el jugador.
  - Si queda poca energía se alterna entre alejarse del jugador y recuperarse (La recuperación será similar a la del jugador y también depende del terreno).

Otras posibilidades:

- Si se tiene menos energía el ataque es más débil.
- Cada 4 turnos el terreno cambia aleatoriamente
- Se puede dar cierta aleatoriedad en la toma de decisión de la CPU

El programa principal puede estar en la clase Juego y ahí se crea el orco, el guerrero (se le pregunta el arma al jugador) y objeto escenario pasándole los dos objetos. Se realiza aquí la gestión por turnos del juego y el fin en caso de que uno gane.