Ejercicios adicionales UT3 (II) - *Estructura condicional*

Ejercicio 7.

- Crea en C:\Programacion\UT3 un nuevo proyecto Ordenador y añade una clase con el mismo nombre. Los objetos de esta clase serán ordenadores y almacenarán la cantidad máxima de memoria que pueden tener (en Mb) en un atributo, *maxMemoria* de tipo entero, y la cantidad de memoria instalada, *memoriaInstalada*, también entero.
- La clase incluye un constructor sin parámetros que inicializa el ordenador a un tamaño máximo de memoria de 1024 Mb y con memoria instalada de 0 bytes
- Define un accesor, getMaxMemoria() que devuelve la memoria máxima permitida y el método obtenerMemoriaDisponible() que devuelve la memoria que queda disponible (por instalar)
- Hay además tres mutadores, vaciarMemoria(), método sin parámetros que "libera" toda la memoria instalada, *instalar256Mb*() que instala 256 Mb de memoria e instalarMemoria() que instala la cantidad de memoria que se le pasa al mutador como parámetro.
- Mejoraremos la clase Ordenador de tal forma que ahora el constructor tome un parámetro y establezca el atributo *maxMemoria* con el valor de dicho parámetro. El constructor verificará que el valor del parámetro es estrictamente mayor que 0. Si es negativo la memoria máxima se inicializará a 1024.
- Añade un nuevo accesor, printInformacion(), sin parámetros y sin valor de retorno que visualiza:

  "Este ordenador puede tener hasta XXXX Mb de memoria.
  Todavía es posible instalar XXXX Mb"

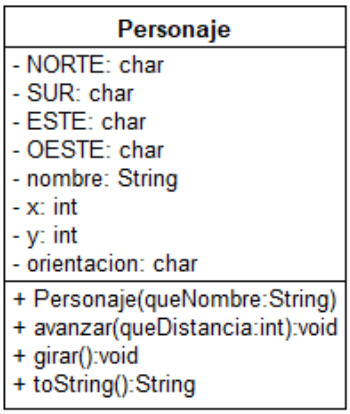
- Modifica el método instalar256Mb() para que devuelva un valor de tipo *boolean*. Si la cantidad de memoria disponible es menor que 256 el método no hace nada y devuelve *false*, en otro caso instala los 256 Mb de memoria y devuelve *true*.
- Modifica el método instalarMemoria() de la misma forma que has modificado el método anterior. Sin embargo ahora se devuelve *true* y se instala la memoria solo si la cantidad especificada en el parámetro es menor o igual que lo que queda disponible y además el parámetro no es negativo. En caso contrario no se instala nada y se devuelve *false*.

Ejercicio 8.

- Crea en C:\Programacion\UT3 un nuevo proyecto Calentador y añade una clase con el mismo nombre.
- Define dentro de la clase Calentador un atributo entero *temperatura*.
- Define un constructor sin parámetros que inicialice el atributo con el valor 15.
- Define los mutadores calentar() y enfriar() cuyo efecto es incrementar y decrementar la temperatura en 5º.
- Define un accesor para la *temperatura*.
- Añade a la clase tres nuevos atributos: *mínimo*, *máximo* e *incremento*. Los valores de *mínimo* y *máximo* se establecen a través de los parámetros que se pasan al constructor. El valor del *incremento* se establece a 5 dentro del constructor.
- Modifica las definiciones de calentar() y enfriar() para que se utilice el valor de *incremento*.
- Verifica que todo funciona bien.
- Modifica el método calentar() de tal forma que no permita que la *temperatura* se establezca a un valor mayor que el *máximo*. Haz lo mismo con enfriar() para que la *temperatura* no baje del valor *mínimo*. Verifica que la clase trabaja apropiadamente.
- Añade el método setIncremento() que toma un parámetro entero y lo utiliza para establecer el valor del *incremento*. Testea la clase creando varios objetos Calentador desde BlueJ.
- ¿Qué ocurre si se pasa un valor negativo como parámetro a setIncremento()? Modifica este método para que esto no ocurra.

Ejercicio 9.

- Abre el proyecto Personaje Videojuego. El proyecto incluye ya dos clases, AplicacionVideojuego e InterfazUsuario.
- Añade una nueva clase Personaje al nuevo proyecto. Esta clase representará personajes de un videojuego.



- Un personaje se caracteriza por tener un nombre, ocupar una posición (en un posible escenario) determinada por un par de coordenadas y poseer una orientación.
- Define las cuatro constantes mostradas en el diagrama para cada personaje
- El constructor de la clase crea el personaje con un determinado nombre que recibe como parámetro. La posición inicial del personaje es (0,0) y su orientación de salida es 'N'
- El método avanzar() modifica las coordenadas del personaje que avanzará la distancia (en unidades) especificada como parámetro. Si la orientación es norte o sur cambiará la coordenada *y*, si la orientación es este u oeste cambiará la coordenada *x*
- El método girar() cambia la orientación del personaje que siempre gira en sentido horario: N → E, S → O, E → S, O → N. Usa en este método una sentencia *switch*.

- Para poder tener una representación visual de la situación del personaje escribe el método toString() que devuelve un String en la forma:

Personaje = PEPITO
Orientacion = S
Posicion = [12 , 6]

- Prueba la clase Personaje creando varios objetos y testeando sus métodos
- Ejecuta la aplicación completa. Para ello:
  a) Crea un objeto de la clase InterfazUsuario y llama al método ejecutar() . Interactúa con la pantalla de texto tecleando las opciones que se presentan
  b) Limpia el Object Bench y llama ahora al método main() de la clase AplicacionVideojuego (no crees ninguna instancia de esta clase)
- Analiza el código de las clases AplicacionVideojuego e InterfazUsuario

Ejercicio 10. Crea un proyecto que incluya una clase Cafetera con atributos *capacidadMaxima* de tipo entero (la cantidad máxima de café que puede contener la cafetera) y *cantidadActual* de tipo entero también (la cantidad actual de café que hay en la cafetera). La clase incluirá los siguientes métodos:

- el constructor que establece la capacidad máxima en 1000 c.c y la cantidad actual a 0 (cafetera vacía)
- accesores y mutadores para los dos atributos
- llenarCafetera() – hace que la cantidad actual sea igual a la capacidad
- sevirTaza(int) – simula la acción de servir una taza con la capacidad indicada como parámetro. Si la cantidad actual de café no alcanza para llenar la taza se sirve lo que quede
- vaciarCafetera() – pone la cantidad actual de café a 0
- agregarCafe(int) – añade a la cafetera la cantidad de café indicada como parámetro