Examen Programación II – extraordinaria – junio 2024

Partiendo del examen anterior en el que trabajábamos con una agencia espacial DeustoSpace, se ha elaborado un código que tiene un elemento adicional. Se ha añadido **otra clase hija de Personal**, **Especialista**, que añade una especialidad (texto) y una dificultad de la misma, valorada de 0 a 100 (con control de excepción fuera de ese rango).

Realiza las siguientes tareas - puntuación sobre 10 entre paréntesis, clases a modificar entre corchetes.

TAREA 1 – Competencia 1 (Programación orientada a objetos)

TAREA 1A. Interfaces (1,5) [PaisComprobable, Nave y otras que necesites]

Define un interfaz **PaisComprobable** que permite ciertas comprobaciones/operaciones sobre el país, a través de dos métodos:

```
/** Devuelve el país del objeto
 * @return      nombre de país, null si no lo tiene definido
 */
public String getPais();
/** Informa si el país es ajeno a la agencia DeustoSpace
 * @return      true si el país es ajeno, false en caso contrario
 */
public boolean esPaisAjeno();
```

Haz que ese comportamiento **lo implementen las clases Nave, Astronauta, Especialista y Tierra**. Hazlo **evitando toda la redundancia de código posible**. Puedes usar para ello el método ya definido comprobarPaisAjeno en la clase DeustoSpace (**observa bien su cabecera para usarlo**), teniendo en cuenta la equivalencia conocida de países-proveedores de nave: Arianespace → France, SpaceX → USA, Roscosmos → Russia.

TAREA 1B. Interfaces y polimorfismo (2) [DeustoSpace, Principal]

Implementar el método **generarPaisesAjenos()** de DeustoSpace, que utiliza el interfaz definido para visualizar en consola y devolver una lista polimórfica con todas las ocurrencias de naves o personas de países ajenos (aunque se repitan), dentro de las misiones de DeustoSpace.

Si prefieres puedes visualizar y devolver la lista de países ajenos (strings) en su lugar. En este caso el ejercicio se valora sobre 1,5 puntos en lugar de 2.

Descomenta la llamada al método desde Principal para comprobar que salen los países ajenos en consola (*tienes en la cabecera del método la información de qué países deberían aparecer, para poder comprobarlo*).

TAREA 2 – Competencia 2 (Estructuras de datos y entrada-salida)

TAREA 2A. Carga de fichero de texto (1,5) [DeustoSpace, Principal]

Programa el método **cargarMisionesTabs** de DeustoSpace, cuyo objetivo es reiniciar la información de misiones y cargarla desde un fichero de texto integrado (*misiones.txt*) en el que cada misión, nave o personal viene dado en una línea del fichero, con datos separados por tabuladores. Puedes observar las particularidades del formato en ese fichero misiones.txt.

Observa que ya tienes disponible un método crearDesdeLineaTabulada(String línea) en cada una de las clases Mision, Nave, Astronauta, Tierra, Especialista. Puedes usar ese método para ir cargando cada línea, aprovechando que el método devuelve null si no se puede crear correctamente el objeto partiendo de esa línea.

Cada vez que aparece una misión se crea y añade una nueva misión, y cada nave o personal que aparece debe asignarse a esa última misión creada.

Descomenta la llamada de Principal para comprobar que la carga funciona. Puedes quitar el método inicDatosDePrueba(), que produce la misma información que la existente en el fichero.

TAREA 2B. Salida de fichero de texto (1,5) [DeustoSpace, Mision]

Programa el método **guardarMisionesTab** de la clase DeustoSpace, teniendo en cuenta que para ello debes programar el método **aLineasTabuladas()** de la clase Mision, en el que puedes utilizar los métodos aLineaTabulada() de las clases Nave y la jerarquía de Personal, **ya codificados y funcionales**. La misión tiene una línea separada con tabuladores

MISION <*tab*> nombre <*tab*> lugar <*tab*> destino <*tab*> fecha

La fecha se guarda como una cuenta de días (busca el método que lo hace en la clase LocalDate).

Después de esa línea se deben sacar todas las líneas, separadas por salto de línea, de la nave (solo si existe) y del personal de la misión. *(recuerda: métodos de salida ya programados)*

Si consigues que funcione, la llamada al método en Principal (ya existente) debería generar el fichero *salida.txt*. Si es correcto, *debería tener exactamente el mismo contenido que misiones.txt*.

TAREA 2C. Mapas (1,5) [DeustoSpace]

Programa el método **comprobarDificultadEspecialidades()** de la clase DeustoSpace.

En los especialistas, se han cometido errores al codificar las dificultades asociadas a cada uno. En principio, los valores de dificultad de una misma especialidad no deberían tener más de 30 puntos de diferencia entre sí. Este método debe visualizar en consola aquellas especialidades de especialistas donde ocurra este error, indicando en cada caso qué valores de dificultad han aparecido (ordenados y sin repetir). Para los datos de prueba, debería aparecer en consola:

```
Biología --> [61, 95]
Ingeniería Mecánica --> [25, 61, 72]
Matemáticas --> [43, 58, 85]
Meteorología --> [20, 61]
```

Observa que en esos casos el valor mínimo con respecto al máximo difiere en más de 30.

Para programar este método, recorre todos los especialistas dentro de las misiones, y **define un mapa donde la clave sea cada especialidad y el valor una colección de los valores de dificultad** enteros que los distintos especialistas con esa especialidad tienen. Hazlo lo más eficazmente posible para que esos valores de dificultad **estén ordenados y no se repitan**.

Tras calcular el mapa, recórrelo para comprobar cuándo la distancia entre el menor y el mayor valor de cada especialidad es mayor que 30.

TAREA 3 – Competencia 3 (Pruebas unitarias)

TAREA 3A. Prueba de constructor - datos (1) [EspecialistaTest]

Prueba el funcionamiento correcto del constructor de la clase Especialista usando JUnit en la clase nueva EspecialistaTest. Comprueba que el constructor inicializa correctamente nombre, país, especialidad y dificultad. Prueba dificultades correctas, pero incluye varios casos con distintos valores de especialidad que podrían recibirse (observa la documentación incluida en el propio constructor).

TAREA 3B. Prueba de constructor - excepción (0,5) [EspecialistaTest]

Prueba el funcionamiento correcto de la excepción generada por el constructor de la clase Especialista, al menos usando dos valores de rango incorrecto.

TAREA 3c. Corrección de acuerdo a prueba (0,5) [Especialista]

Es probable que en la tarea 3A hayas detectado un error en la clase `Especialista`. Corrígelo en esa clase para que la prueba funcione correctamente, intentando que no se generen errores tampoco en otros métodos.