E.S. Ingeniería y Tecnología

Ingeniería Informática

Lenguajes y Sistemas Informáticos

Lenguajes y Paradigmas de Programación

Práctica de laboratorio #8

Esta práctica de laboratorio se ha de realizar utilizando el sistema de *control de versiones git*, el lenguaje de programación Ruby, la metodología de *Desarrollo Dirigido por Pruebas con RSpec*, la herramienta de gestión de dependencias Bundler, la herramienta de comprobación continua Guard, las herramientas de generación de documentación rdoc o YARD.

Se ha de trabajar con el repositorio que se aceptó en `GitHub Classroom` denominado "Gema".
(`https://classroom.github.com/a/ePZukRWX`)

Todo el código a desarrollar se ha de integrar en la gema Ruby para representar aparcamientos.

1. Cree una jerarquía de clases Ruby para representar *vehículos* y *vehículos de motor*.

   Se define un **vehículo** como el aparato apto para circular por las vías afectadas por la Ley de Tráfico. Un **Vehículo de motor** es un vehículo provisto de motor para su propulsión.

   Ejemplos de expectativas son las siguientes:

```
Aparcamiento
  Representación de un Vehículo - Aparcamiento::Vehículo
      Atributos de la clase Vehículo
        Tiene una clase para representar vehículos
        Tiene un atributo para identificar al vehículo
        Tiene un atributo con la altura en metros
        Tiene un atributo con el ancho en metros
        Tiene un atributo con el largo en metros
        Tiene un atributo con el peso en toneladas
        Se obtiene una cadena con la información del vehículo correctamente formateada
      Herencia de la clase Vehículo
        Se espera que una instancia de la clase Vehículo sea un Vehículo
        Se espera que una instancia de la clase Vehículo sea un objeto (Object)
        Se espera que una instancia de la clase Vehículo sea un objeto básico (BasicObject)
        No se espera que una instancia de la clase Vehículo sea una cadena (String)
        No se espera que una instancia de la clase Vehículo sea un número (Numeric)
  Representación de un Vehículo de motor - Aparcamiento::Motor
      Atributos de la clase Motor
        Tiene una clase para representar vehículos de motor
        Tiene un atributo para el número de ruedas
        Tiene un atributo para el número de plazas
        Tiene un atributo para la potencia del motor en centímetros cúbicos
        Tiene un atributo para la velocidad máxima en kilómetros por hora
        Se obtiene una cadena con la información del vehículo a motor correctamente formateada
      Herencia de la clase Motor
        Se espera que una instancia de la clase Motor sea un vehículo de motor
        Se espera que una instancia de la clase Motor sea un Vehículo
        Se espera que una instancia de la clase Motor sea un objeto (Object)
        Se espera que una instancia de la clase Motor sea un objeto básico (BasicObject)
        No se espera que una instancia de la clase Motor sea una cadena (String)
        No se espera que una instancia de la clase Motor sea un número (Numeric)
```

2. Se ha de contar el número de objetos que se instancia de la clase Vehículo.

3. Los vehículos han de ser comparables según su *volumen* y los vehículos de motor según su *número de plazas*.

4. Revise la jerarquía de clases Ruby de la clase para representar los datos de un aparcamiento. Ejemplos de expectativas son las siguientes:

```
Aparcamiento
  Representación de los Datos de un aparcamiento - Aparcamiento::Datos
  Atributos de la clase Aparcamiento::Datos
    Todo aparcamiento tiene el atributo de accesibilidad (1..5)
    Todo aparcamiento tiene el atributo de seguridad (1..10)
    Un aparcamiento tiene un atributo para su identificación
    Un aparcamiento tiene un atributo para su nombre comercial
    Un aparcamiento tiene un atributo para su descripción (Cubierto - Aire libre - Mixto)
    Tiene un atributo para el tipo de aparcamiento (autobuses, bicicletas, coches, motos)
    Tiene un atributo para representar el conjunto de plazas del aparcamiento (altura, longitud, anchura)
    Tiene un atributo para representar el conjunto de plazas libres y ocupadas
    Tiene un método para obtener una cadena con la información del aparcamiento correctamente formateada
    Tiene un método para devolver el número de plazas del aparcamiento
    Tiene un método para devolver el número de plazas libres del aparcamiento
  Herencia de la clase Aparcamiento::Datos
    Se espera que una instancia de la clase Datos sean los Datos de un aparcamiento
    Se espera que una instancia de la clase Datos sea un objeto (Object)
    Se espera que una instancia de la clase Datos sea un objeto básico (BasicObject)
    No se espera que una instancia de la clase Datos sea un Vehículo
    No se espera que una instancia de la clase Datos sea un Motor
```

5. Revise la jerarquía de clases Ruby de las funcionalidades de un aparcamiento. Ejemplos de expectativas son las siguientes:

```
Aparcamiento
   Interfaz de las funcionalidades - Aparcamiento::Funciones
     Existe una constante para representar si el aparcamiento está completo
     Existe una constante para representar si el aparcamiento tiene plazas libres
     Existe una funcion para mostrar el estado de un aparcamiento (completo, plazas libres)
     Existe una clase para representar los datos de un aparcamiento
   Herencia del módulo Aparcamiento
     Se espera que un aparcamiento sea un objeto de la clase Module
     Se espera que un aparcamiento sea un objeto (Object)
     Se espera que un aparacamiento sea un objeto básico (BasicObject)
     No se espera que un aparcamiento sea una clase (Class)
     No se espera que un aparcamiento sea un vehículo
     No se espera que un aparcamiento sea un vehículo a motor
```

6. Escribir la dirección HTTP del repositorio de la organización 'ULL-ESIT-LPP-2223/gema-XXX' en la tarea habilitada en el campus virtual.

Referencia

Visic, N., Fill, H. G., Buchmann, R. A., Karagiannis, D. (2015, May). A domain-specific language for modeling method definition: From requirements to grammar. In 2015 IEEE 9th International Conference on Research Challenges in Information Science (RCIS) (pp. 286-297). IEEE Computer Society.