Decisiones de diseño
Entrega-1

Alumna: cossio ana

Estructura del proyecto general del proyecto:

/lib/rn/commands/ ⟶ Provista por la cátedra
/lib/rn/exceptions/ ⟶ Contendrá las excepciones de los modelos
/lib/rn/helpers/ ⟶ Contendrá código en común
/lib/rn/models/ ⟶ Tendrá los Modelos Book y Note

Gemas instaladas:

gem 'dry-cli', '~> 0.6'
gem "tty-editor", "~> 0.6.0"

Acerca de Models

Una instancia de Book tendrá el atributo name, así como la información sobre su ruta, que notas posee
La clase Book tendrá la información sobre qué books están persistidos en el disco duro.
Una instancia Note tendrá los atributos: title, book, conten. Pero además la información sobre su ruta se encuentra, su ruta completa (incluyendo su extensión), si no posee contenido.
Estos modelos cuentan

Acerca de las Excepciones

Para el manejo de las excepciones además de usar las excepciones **Errno::ENOTEMPTY** y **Errno::ENOENT** que están relacionadas con los ficheros, decidí tener un módulo Exceptions donde contendrán excepciones personalizadas los modelos Book y Note, dichas excepciones extienden de **StandardError**.

Ejemplo de algunas implementaciones:

```
    def save
      !Dir.exists?(self.path) || raise(Exceptions::Books::NameExists.new(name))
      FileUtils.mkdir_p(self.path)
    end
```

El método **save** pertenece al **Models::Book,** cuando se quiere crear un libro con nombre “book_1”, se creará una instancia de del **Models::Book,** llamará a su método de instancia **save** para corroborar que que no exista en el disco duro un libro con su mismo nombre.
En caso de que hubiera un Book con su mismo nombre lanzaría **Exceptions::Books::NameExists.**

Excepciones implementadas:

Exceptions::Books:NotFound :

Se levanta si no encuentra un libro en el disco.

Exceptions::Books::NameExists:

Se levanta si ya existe un libro con el mismo nombre en el disco.

Exceptions::Notes::TitleExists:

Se levanta si ya existe una nota con el mismo título en el disco.

Acerca de la implementación en Commands::Books

- **Books**
  - **Create**

```
Commands:
        ruby bin/rn books create NAME       " # Creates a new book named "NAME"',
        ruby bin/rn books create "my book"  “ # Creates a new book named my_book'
```

Cuando se quiere crear un libro, este creará una instancia del modelo book, antes de que se cree dicha instancia, zanitizara el nombre para reemplazar en caso de que fuera necesario con un “_” si el nombre contiene caracteres especiales o espacios en blanco.
Una vez creado el objeto book llamará a su método **save**, donde el objeto verificará que no exista otro libro con su mismo nombre en el disco, en caso de que ya exista un libro persistido con su mismo nombre lanzará una excepción **Exceptions::Books::NameExists** donde dicha excepción se propagara y se manejara en **Commands::Books create**

  - **Delete book**

```
Commands:
        ruby bin/rn books delete NAME       " # Creates a new book named "NAME"',
        ruby bin/rn books delete "my book"  “ # Creates a new book named my_book'
        ruby bin/rn books --global              # Deletes all notes from the global book
```

Si se usa **el comando delete “name”**, se creará una instancia del Modelo Book, dicha instancia usará su método delete, si existe en el sistema se eliminará, pero si no se encuentra levantará una excepción, y en el caso que el libro indicado sea el libro global también se levantará una excepción indicando que dicho libro no se puede eliminar.

```
def delete
 Dir.exists?(self.path) || raise(Exceptions::Books::Generico.new("El #{self} no se encontró"))
      !(name ==GLOBAL) || raise(Exceptions::Books::Generico.new('', "No se puede eliminar la carpeta global, ya que es una carpeta por defecto del sistema"))
      FileUtils.rm_rf(self.path)
end
```

En caso que se quiera eliminar los archivos de GLOBAL, ya no es necesario preguntarse si este existe en el disco, pudiendo eliminar de forma directa sus archivos.