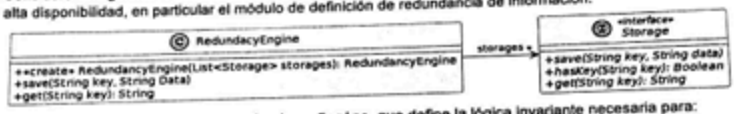
Framework

Ejercicio 3 - Frameworks

Considere el siguiente extracto de código y diagrama de clases UML de un framework para sistemas de alta disponibilidad, en particular el módulo de definición de redundancia de información.



Este framework provee una clase, `RedundancyEngine`, que define la lógica invariante necesaria para:
(i) almacenar la información en más de un espacio de almacenamiento, esto es, el almacenamiento de cada clave/valor en todos los *Storage* provistos, y,
(ii) recuperar el valor correspondiente a una clave desde el primer storage que posea dicha información.
Por otro lado, la interface *Storage*, también provista por el framework, define el protocolo necesario para que `RedundancyEngine` pueda almacenar y recuperar la información solicitada.

Es responsabilidad de las personas que utilizan el framework definir el comportamiento específico para almacenar la información en, por ejemplo, un archivo, en memoria, o cualquier otro soporte. Siempre mediante la implementación de la interface *Storage*.
Considerando que, como usuarios/as del framework, ya hemos definido dos clases que implementan la interface *Storage* llamadas `FileStorage` e `InMemoryStorage`, deberíamos usar `RedundancyEngine` para acceder y almacenar información de la siguiente manera:

```
List<Storage> storages = new ArrayList<Storage>();
storages.add(new InMemoryStorage(...));
storages.add(new FileStorage(...));

RedundancyEngine redundancyEngine = new RedundancyEngine(storages);

redundancyEngine.save("name", "Diego Maradona");

// La linea siguiente deberia retornar el String "Diego Maradona"
redundancyEngine.get("name");

public class RedundancyEngine{
      List<Storage> storages;
      public RedundancyEngine(List<Storage> storages){
            super();
            this.storages = storages;
      }
      public void save(String key, String data){
          for (InformationStorage storage : this.getStorages()){
              storage.save(key, data)
          }
      }
      public String get(String key){
          for (InformationStorage storage : this.getStorages()){
              if (storage.hasKey(key)){
                  return storage.get(key);
              }
          }
          return null;
      }
}
public interface Storage{
      public void save(String key, String data);
      public Boolean hasKey(String key);
      public String get(String key);
}
```

Responda las siguientes preguntas, basándose en el subconjunto de clases y métodos del framework presentado anteriormente:

1. ¿El comportamiento variable del framework (hotspots), está implementado mediante herencia o composición? Justifique su respuesta.
2. ¿Cuáles son los hook methods?
3. ¿Cuál es el Frozen Spot?

Respuesta inciso 1:

El comportamiento variable del framework, se da a partir de **COMPOSICION** porque el mismo framework utiliza una clase y comportamientos externos a el propio framework, como es el caso de la lista de storages. Utiliza el comportamiento de las clases que implementen la interfaz 'storage'.

Para que este framework utilice el comportamiento variable a partir de herencia, la clase RedundancyEngine deberia ser una clase abstracta, y deberia tener metodos abstractos o protected para que las subclases puedan extender su comportamiento. Pero como el

framework usa clases externas a el, los comportamientos variables (hotspots) se dan a partir de composicion.

Respuesta inciso 2:

El concepto de hooks methods es aquel comportamiento variable o puntos de extension a partir de metodos que utiliza el framework. En este ejemplo, los hooks methods serian los metodos de:

- Dentro de la interface Storage:
  - save (String key String data)
  - hasKey (String key) : Boolean
  - get(String key) : String
  Esos metodos serian los hooks methods, que utiliza el framework RedundancyEngine.

Respuesta inciso 3:

El concepto de frozen spots es el comportamiento invariante del framework, no estan pensados para que se puedan redefinir y no pueden redefinirse.
En el framework dado, los frozen spots serian los metodos de:

- Dentro de la clase RedundancyEngine
  - public void save(String key,, String data)
  - public String get (String key)
  Ambos metodos mencionados, son frozen spots, ya que no estan pensados para que se pueda extender su comportamiento y no pueden modificarse.

Nota: No pregunta sobre la inversion de control, pero podemos decir que:

En los frozen spots mencionados, es donde se produce la inversion de control, que sucede cuando el framework llama / utiliza nuestro codigo. En este caso, el codigo nuestro seria crear una clase que implemente la interfaz y poder redefinir el comportamiento de los metodos de la interfaz