Paradigma!

Un grupo de amigos egresados de un conocido colegio secundario han decidido juntarse para crear su propia página web: Paradigma!, un portal en que usuarios suben información de interés general mediante posts.

La página cuenta con un sistema de valoración de los posts que permite categorizar a los usuarios. Los posts pueden ser comentados y puntuados: Al comentar un post se agrega un pequeño comentario textual y al puntuarlo se le incrementa una determinada cantidad de puntos al post.
El valor de un post equivale a la cantidad de comentarios que tenga más la puntuación. Hay algunos post, llamados premium, para los cuales se considera el doble de la puntuacion recibida. La valoración de un usuario es la suma de todos los valores de sus posts.
Un usuario que recién ingresa es de categoría "Novato" y solo puede crear posts comunes. Cuando alcanza una valoracion de 100 puede ser promovido a categoría "Intermedio", en la que sigue creando post comunes.
Cuando un usuario "Intermedio" tiene una valoración de 1000 puntos al menos un post con más un valor de más de 500, puede ser promovido a la última y definitiva categoría, llamada "Experto", lo que le permite empezar a crear posts premium. Al cambiar de categoría los usuarios mantienen los posts anteriores.

La funcionalidad requerida es:

1. Poder crear un usuario y que esté listo para empezar a postear. 2. Hacer que un usuario postee un contenido dado. 3. Comentar un post 4. Puntuar un post 5. Saber el valor de un post. 6. Saber la valoracion de un usuario 7. Saber si un usuario es nuevo, es decir, si tiene algún post sin comentarios.	8. Promover a un usuario a su categoría siguiente, si es posible. 9. Obtener el post de un usuario que más éxito tuvo (el de mayor valor). 10. Saber la cantidad de posts interesantes de un usuario. Un post es interesante si tiene al menos un comentarios largos (más de 100 caracteres). Los posts premium son más exigentes para ser interesantes: deben tener además más de 50 puntos.

Se debe realizar:

- El diagrama de clases
- La codificación de los métodos necesarios
- Las siguientes justificaciones teóricas:
  ¿Dónde aparece el concepto de polimorfismo, cómo es usado y a qué objetos le resulta útil?
  ¿Fue de utilidad la herencia y la redefinición? ¿Para qué?

Solución

La consigna da a entender que va a haber dos tipos de “cosas” que van a tener comportamiento: los usuarios y los posts. Por lo que estas serán nuestras clases. También se podría discutir la idea de si los comentarios podrían ser otra clase, pero esto lo vamos a ver más adelante.

1. Poder crear un usuario y que esté listo para empezar a postear.

Ya decidimos que va a existir la clase *Usuario*. Lo que hay que discutir ahora es los atributos que tendrá un usuario. Un usuario debería conocer todos los posts que realizó, para poder luego calcular su puntaje y demás... por lo que tendremos una variable de instancia *posts* que será una colección de los mismos. Para esto hay que inicializar la colección, lo haremos con el método *initialize* que se envía automáticamente al objeto instanciado, cuando se hace un new.

```
class Usuario {
	var posts = []
	var categoria = categoriaNovato
}
```

Este paso es importante ya que no se podria agregar elementos a la colección sin inicializarla. La variable posts necesita estar inicializada previamente como una lista.

2. Hacer que un usuario postee un contenido dado.

Para este punto lo que hay que hacer es un método que haga lo siguiente:

- Crear una nueva instancia de *Post,* diciendole que su título o texto es el recibido por parámetro.
- Agregar el post creado a la colección de posts del usuario.

.

Esto lo hacemos de la siguiente forma:

```
method postear(unTitulo) {
	posts.add(categoria.crearPost(unTitulo))
	}

class Categoria {
	method crearPost(unTitulo) {
		return new Post(unTitulo)
	}
}

class Categoria {
	method crearPost(unTitulo) {
		return new Post(unTitulo)
	}
}

object categoriaIntermedio inherits Categoria {}
object categoriaNovato inherits Categoria {}
```

```
object categoriaExperto inherits Categoria {
	override method crearPost(unTitulo) {
		return new PostPremim(unTitulo)
	}
}
```

Para que esto funcione necesitamos definir la clase Post con un constructor
Pensemos también en que un post para funcionar va a tener una colección de comentarios, y un puntaje. Es importante inicializar el puntaje en 0 también para poder acumularlo luego mediante otros métodos.

```
class Post {
	var titulo
	var puntos = 0
	var comentarios = []

	constructor(t) {
		titulo = t
	}
}
```

3. Comentar un post.

Este punto es sencillo, la colección de comentarios del usuario ya viene inicializada por lo que lo único que hay que hacer es un *add( )*. Será una colección de strings.

/* en Usuario */
```
method comentar(unComentario) {
	comentarios.add(unComentario)
}
```

4. Puntuar un post.

Este es un simple método de acumulación, notemos que si el puntaje del usuario no estuviera inicializado, la primera vez que se llame a este método haciendo por ejemplo, puntuar(15), se estaría queriendo sumar 15 a null.

/* en Usuario */
```
method puntuar (unPuntaje) {
	puntaje += unPuntaje
}
```

5. Saber el valor de un post..

Para hacer este punto hacemos uso del método de colección *size*, que nos devuelve el tamaño de una colección.
/* en Post */
```
method valor() {
```

```
	return self.puntaje() + self.cantidadDeComentarios()
}
method cantidadDeComentarios() {
	return comentarios.size()
}
method puntaje()  =  puntaje
```

/* en Post Premium */

```
override method puntaje() {
	return super()*2
}
```

6. Saber la valuacion de un usuario. El puntaje de un usuario es la suma de todos los valores de sus posts.

Usamos el método *sum( )*, para sumar según una expresión.

```
method valuacion() {
	return posts.sum({unPost => unPost.valor()})
}
```

7. Saber si un usuario es nuevo, es decir, si tiene algún post sin comentarios.

Vamos a usar el método *any( )*, para saber si para algún elemento de una colección se cumple cierta condición.

```
method esNuevo() {
	return posts.any({unPost => unPost.cantidadDeComentarios() == 0})
}
```

Otra forma de hacerlo podría ser:

```
method esNuevo() {
	return posts.any({unPost => unPost.sinComentarios()})
}
method sinComentarios(){
	return comentarios.isEmpty()
}
```

Como verán, lo importante en este punto es no hacer lo siguiente:

```
method esNuevo() {
	return posts.any({unPost => unPost.comentarios().size() == 0})
}
```

Ya que de esta forma el usuario se estaría metiendo en la estructura interna del post, estaría conociendo sus atributos… sabiendo que tiene una colección de comentarios, etc. Esto no es correcto.

8. Promover a un usuario, si es posible.

/* en Usuario*/

```
method promover() {
	if  (categoria.puedoPromover(self)) {
```

```
                categoria = categoría.categoriaSiguiente()
            }
        }
```

/* en categoriaIntermedio */

```
method categoriaSiguiente() {
       return categoriaExperto
}
method puedoPromover(usuario) {
     return usuario.tenesMasDe(1000) & usuario.tenesPostConMasDe(500)
}
```

/* en categoriaNovato */

```
method categoriaSiguiente() {
        return categoriaIntermedio
}
method puedoPromover(usuario) {
        return  usuario.tenesMasDe(100)
}
```

/* en CategoriaExperto */

```
method puedoPromover(usuario) {
       return false
}
```

/* en Usuario */

```
method tenesMasDe(puntos) {
       return self.valuacion() > puntos
}
method tenesPostConMasDe(puntos) {
        return posts.any({unPost => unPost.valor() > puntos})
}
```

9. Obtener el post de un usuario que más éxito tuvo (el de mayor valor).

Vamos a usar el método *max( )*, para obtener el elemento de la colección que haga máxima cierta expresión.

```
method postMasExitoso() {
       return posts.max({unPost => unPost.valor()})
}
```

10. Saber la cantidad de posts interesantes de un usuario. Un post es interesante si tiene al menos 20 comentarios largos (más de 100 caracteres). Los posts premium son más exigentes para ser interesantes: deben tener además más de 50 puntos.

Usamos el método *count( ):*, para contar los elementos de una colección según un criterio.

```
method cantidadPostsInteresantes(){
       ^ self.count({unPost => unPost.esInteresante()})
```

```
}
```

/* en post */

```
method esInteresante() {
	return self.cantidadComentariosLargos() > 20
}
method cantidadComentariosLargos() {
	return comentarios.count({unComentario => unComentario.size() > 100})
}
```

/* en post Premium*/

```
override method esInteresante() {
	return super() && self.puntaje() > 50
}
```