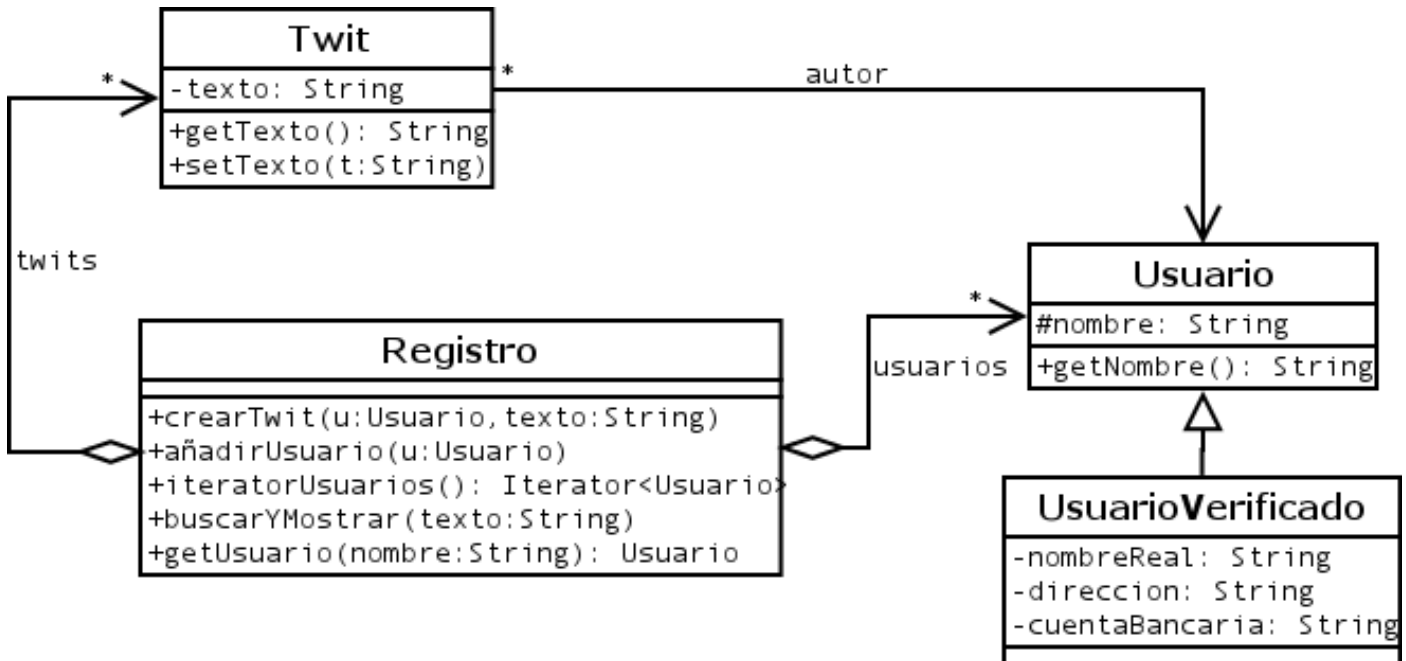


Metodología i Programació Orientada a Objectes

Examen parcial G50. Viernes 24/4/2015.

Estamos diseñando una app clónica del popular servicio Twitter, y tenemos el siguiente diagrama de clases UML:



El **Registro** de la aplicación guarda un registro de **Twits** y uno de **Usuarios**. Cada **twit** encapsula un texto, y el **Usuario** encapsula un nombre corto (**sin** la @). Cada **Twit** pertenece a un **Usuario**.

Es importante que los **Twits** estén almacenados en el registro según un orden dado. También es importante que el registro permita la obtención de los usuarios registrados a partir de su nombre (a través del método `getUsuario`, tal y como está especificado en el diagrama de clases).

Hay un tipo especial de **Usuario**, el **Usuario Verificado**, que es un usuario de pago cuya identidad real está verificada por la empresa. Para este tipo de usuario hay que guardar algunos datos extra, como el nombre real, la dirección, y la cuenta bancaria.

Ejercicios

- (2.75 puntos) Define las clases y sus atributos en Java, de tal manera que se refleje las información de las clases, así como las relaciones entre éstas. Considera las restricciones subrayadas como importantes en el enunciado. No hay que definir cabeceras de métodos ni de constructores, solo la de las clases y sus atributos.
- (0.25 puntos) Pregunta: para que el Registro guarde usuarios verificados, ¿debemos trazar alguna relación directa entre la clase `Registro` y la clase `UsuarioVerificado`? Justifica tu respuesta.
- (1 punto) Programa el método `toString` de la clase `Twit`, de tal manera que, dado un `twit` que pertenece a un usuario, retorne un `String` con la forma: `@usuario: texto del mensaje`. Por Ejemplo:
@Rajoy: el paro está bajando, y el precio de las chuches
- (0.75 puntos) Programa el método `añadirUsuario(Usuario u)` de la clase `Registro`, que guarde en dicho registro el usuario que se le pasa por parámetro para que luego pueda ser obtenido directamente a partir de su nombre.

5. (1.25 puntos) Asumiendo que la clase `Usuario` define únicamente el siguiente constructor:

```
public Usuario(String nombre)
```

Implementa la clase `UsuarioVerificado` completa, de tal manera que se puedan inicializar y leer sus atributos desde fuera de la clase.

6. (2 puntos) Crea el método de la clase `Registro`: `buscarYMostrar(String texto)` que muestra por pantalla todos los `Twits` que contengan el texto que se pasa por parámetro. La búsqueda no debe hacer distinción entre mayúsculas y minúsculas. Por ejemplo, si se llama al método con el valor "torres" como parámetro, para un conjunto dado de `twits` mostraría algo del estilo:

```
@Pablo_Iglesias_: Mañana meeting bajo las torres Mapfre
@Norcoreano: El último doblete de Torres que Recuerdo lo hizo Bin Laden
@MaciasUPC: Tomándome un carajillo de Torres 5
```

Ayuda: recuerda que la clase `String` proporciona los siguientes métodos:

- `boolean contains(String str)`: retorna `true` si el `String` sobre el cual se invoca contiene el `String` pasado por el parámetro `str`. Retorna `false` en caso contrario.
- `String toLowerCase()`: retorna una copia del `String` sobre el cual se invoca, con todas las letras pasadas a minúscula.

7. (2 puntos) Consideremos la siguiente clase incompleta:

```
public class UnaClaseCualquiera {
    public static void muestraTodosLosUsuarios(Registro r) {
        /* Tu código aquí */
    }
}
```

Rellena el cuerpo del método `muestraTodosLosUsuarios`, que mostraría por pantalla todos los `Usuarios` almacenados en el objeto `Registro` que se le pasa por parámetro.

A considerar: el contenedor de `Usuarios` es privado en la clase `Registro`, y no hay *getter*. Puedes usar el método `Iterator<Usuario> iteradorUsuarios()` que está definido en la clase `Registro`, que te devuelve un nuevo iterador a los `Usuarios` registrados.

Solución

Ejercicio 1

```
public class Registro {
    private List<Twit> twits = new ArrayList<Twit>();
    private Map<String, Usuario> usuarios = new HashMap<String,
Usuario>();
}
public class Twit {
    private String texto;
    private Usuario autor;
}
public class Usuario {
    protected String nombre;
}
public class UsuarioVerificado extends Usuario {
    private String nombreReal;
    private String direccion;
    private String cuentaBancaria;
}
```

Ejercicio 2

No se debe trazar ninguna relación directa entre Registro y UsuarioVerificado, ya que esta última hereda las relaciones de su superclase Usuario con Registro.

Ejercicio 3

```
public String toString() {
    return "@"+autor.getNombre() +": " + texto;
}
```

Ejercicio 4

```
public void añadirUsuario(Usuario u) {
    usuarios.put(u.getNombre(), u);
}
```

Ejercicio 5

```
public class UsuarioVerificado extends Usuario {
    private String nombreReal;
    private String direccion;
    private String cuentaBancaria;
    public UsuarioVerificado(String nombre, String nombreReal,
String direccion, String cuentaBancaria) {
        super(nombre);
        this.nombreReal = nombreReal;
        this.direccion = direccion;
        this.cuentaBancaria = cuentaBancaria;
    }
    public String getNombreReal() {
        return nombreReal;
    }
    public String getDireccion() {
        return direccion;
    }
    public String getCuentaBancaria() {
        return cuentaBancaria;
    }
}
```

```
}
```

Ejercicio 6

```
public void buscarYMostrar(String texto) {  
    for(Twit t : twits) {  
        if(t.getTexto().toLowerCase().contains(texto.toLowerCase())) {  
            System.out.println(t.toString());  
        }  
    }  
}
```

Ejercicio 7

```
public static void muestraTodosLosUsuarios(Registro reg) {  
    Iterator<Usuario> usit = reg.iteratorUsuarios();  
    while(usit.hasNext()) {  
        System.out.println(usit.next().toString());  
    }  
}
```