

Introducción a la Programación

Clase 06



Agenda



- Revisión de clase anterior
- Encapsuladores
- Constructores
- Elementos estáticos
- Enumeraciones
- Construcción de un ejemplo que involucre todos los elementos.

- Consultas de la clase anterior



• Encapsuladores

Cuando creamos un atributo o un método, estos pueden tener al menos uno de los modificadores `public` o `private`, los cuales nos permiten definir la visibilidad de estos desde fuera de la clase.

Si nuestros componentes de la clase son públicos, permitiremos que desde fuera de la clase se pueda acceder a estos para modificar su contenido o utilizarlos.



Encapsuladores

```
public class Curso {  
    public int id = 0;  
    public String nombre = "";  
    private String correo = "";
```

Atributos
públicos

```
public static void main(String[] args) {
```

```
    Curso curso = new Curso();
```

```
    curso.
```

Atributo
privado

```
}
```

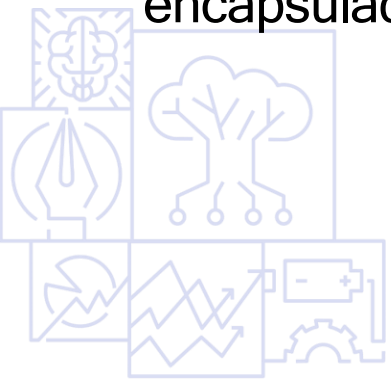
■ id	int
■ nombre	String
● equals(Object obj)	boolean
● getClass()	Class<?>

Se pueden acceder
desde una instancia

• Encapsuladores

Cada clase debería de tener la posibilidad de autocontrolar los datos que muestra o que recibe, pero los atributos al ser variables, solamente pueden controlar que se cumpla con el tipo de dato que reciben, por ejemplo, un atributo público para el correo puede recibir “123” sin ser esta cadena un correo correcto.

Es aquí donde toma importancia contar con atributos privados que no reciban datos desde el exterior, pero para que sean funcionales deben contar con métodos relacionados llamados encapsuladores.



Encapsuladores

```
public class Curso {  
    public int id = 0;  
    public String nombre = "";  
    private String correo = "";  
  
    public String getCorreo() {  
        return correo;  
    }  
}
```

```
public void setCorreo(String correo) {  
    this.correo = correo;  
}
```

Los encapsuladores pueden tener cualquier nombre, pero se utiliza la preposición set y get para identificarlos.

Encapsuladores

```
public String getCorreo() {  
    return correo.substring(0,3) + "....." + correo.substring(correo.indexOf("@"));  
}
```

En esta versión del encapsulador `getCorreo()` no se devuelve el correo completo, sino una versión para verificar por parte del usuario. Fuera de la clase será imposible conocer la cuenta de correo completa (por temas de seguridad). Este método es muy utilizado para verificar cuentas de correo, teléfonos, tarjetas de crédito, etc. ¿Qué problema podría presentar?

Encapsuladores

```
public void setCorreo(String correo) {  
    if (!correo.contains("@")) {  
        System.out.println("No es una cuenta de correo válida");  
    } else {  
        this.correo = correo;  
    }  
}
```

El encapsulador setCorreo verifica que la cadena recibida al menos tenga el carácter @, de lo contrario no lo asignará en el atributo y lo indicará por la consola. La validación de una cuenta de correo implica muchos elementos adicionales.

- Encapsuladores (ejercicio)

Construya una clase Usuario que cuente al menos con 3 atributos, uno de ellos debe ser la contraseña, este atributo debe ser privado y contar con los encapsuladores correspondientes.

El encapsulador get devolverá solamente asteriscos, tantos como caracteres tenga la contraseña.

El encapsulador set validará:

1. Que la contraseña cuente al menos con 8 caracteres.
2. Que no existan dos caracteres iguales consecutivos.

- Constructores

El constructor es un método implícito en todas las clases que es ejecutado en el momento en que se crea una instancia de dicha clase.

Si se desea programar acciones en el constructor, este deja de ser implícito y realiza el código que se especifique.

```
Curso curso = new Curso();
```

Constructor de la
clase Curso

- Constructores

```
public Curso() {  
    // Aquí se realizan las opciones iniciales  
    id = (int) (Math.random() * 1000) + 1;  
}
```

El constructor inicializa el atributo id con un valor al azar entre 1 y 1000

Constructores

```
public Curso() {  
    // Aquí se realizan las opciones iniciales  
    id = (int) (Math.random() * 1000) + 1;  
}  
  
public Curso(int nuevoId) {  
    // Aquí se realizan las opciones iniciales  
    id = nuevoId;  
}
```

Es posible escribir más de un constructor, siempre y cuando su definición (parámetros) los distingan

Esta condición de los constructores aplica para todos los métodos.

- Constructores

Se crea el objeto
con un ID al azar

```
Curso curso = new Curso();
```

```
Curso alternativo = new Curso(20);
```

Se crea el objeto
con el ID 20

• Constructores (ejercicio)

En la clase **Usuario** que hizo previamente, programe 2 constructores.

1. Debe de recibir una contraseña, validar que sea correcta y almacenarla en el atributo.
2. No debe de recibir parámetros y generar una contraseña con caracteres al azar (al menos 10).



• Elementos estáticos

Muchas de nuestras clases (quizás la mayoría), están conceptualizadas para servir de “plantilla” a múltiples objetos, la clase **Usuario** construida previamente funcionará para múltiples usuarios, de aquí el proceso de escribir una clase y poder crear múltiples instancias.

En algunos casos podemos tener clases en las cuales tenemos la certeza que una única instancia es lo que requerimos o que alguno de sus elementos debe compartirse entre todas las instancias.




```
public class Configuracion {  
    public static double tipoCambio = 580;  
    public static String nombreEmpresa = "Universidad Fidélitas";  
    public static String rutaSistema = "C:\\\\Sistema";  
  
    public double mostrarTipoCambio() {  
        // En este caso la palabra reservada this no aplica  
        return tipoCambio;  
    }  
}
```

Los atributos definidos como estáticos pueden ser accedidos haciendo referencia a la clase directamente o por métodos no estáticos dentro de las instancias.

```
Configuracion config = new Configuracion();  
Configuracion.tipoCambio = 600;  
System.out.println(config.mostrarTipoCambio());
```

El tipo de cambio se modifica en la clase,
no en la instancia, pero se puede leer
desde cualquier instancia por medio del
método no estático.

Las **enumeraciones** son un tipo de clase especial que nos permite contar con un número finito de elementos para definir una categoría, un estado o una condición.

```
public enum Estados {  
    Activo,  
    Inactivo,  
    En_Revisión  
}
```



Enumeraciones

```
public int id = 0;  
public String nombre = "";  
private String correo = "";  
public Estados estado;
```

Posteriormente se puede utilizar como un tipo de dato con valores finitos.

```
curso.estado = Estados.
```

Activo	Estados
En_Revisión	Estados
Inactivo	Estados
● valueOf(String arg0)	Estados

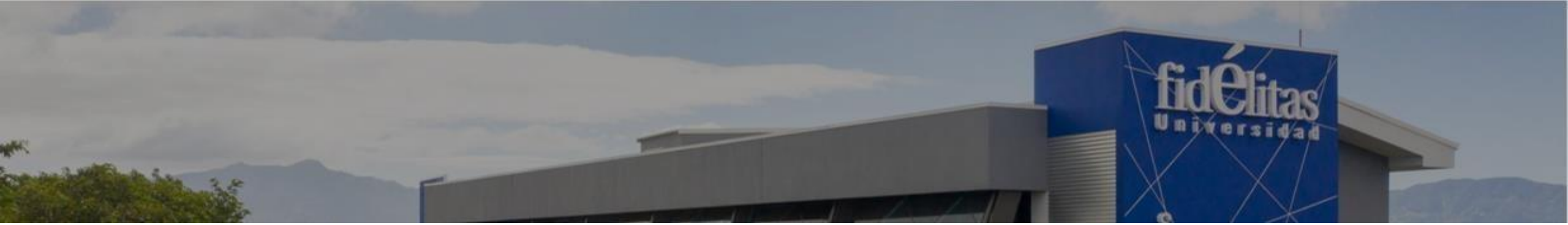
Se puede seleccionar uno de los 3 posibles

- Enumeraciones (ejercicio)

En la clase **Usuario** creada anteriormente, incluya un atributo público llamado “rol”, este rol debe ser de tipo de una enumeración con los valores (Administrador, Asistente, Cajero, Supervisor).

Posteriormente haga este atributo privado e implemente lo necesario para que por medio de encapsuladores set y get se pueda gestionar su contenido.





En los cursos de programación, es importante que el estudiante refuerce lo aprendido con mucha práctica y considerando que las dificultades y errores que se le presenten le van a ayudar a desarrollar sus habilidades de programación.

Desarrolle siempre los ejercicios de la clase buscando alternativas y probando diferentes soluciones.



“Si de todas formas vas a pensar,
mejor piensa en grande.” (D. Trump)

¡Nos vemos la próxima semana!