GG20210615

fidélitas
Universidad

SC-202 Introducción a la Programación

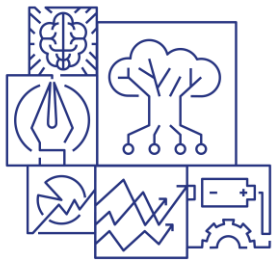
Semana 6

Agenda

- Revisión de clase anterior
- Encapsuladores
- Constructores
- Clases como atributos
- Clases estáticas
- Enumeraciones

fidélitas Universidad

Consultas de la clase anterior

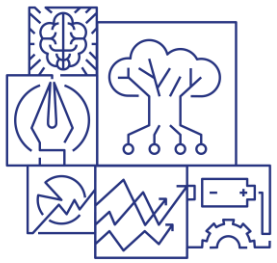


fidélitas Universidad

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.



fidélitas Universidad

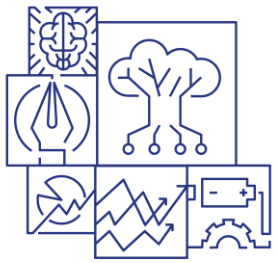
Encapsuladores

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


public static void main(String[] args) {

    Curso curso = new Curso();
    curso.
                id                          int
                nombre                      String
                equals(Object obj)          boolean
                getClass()                  Class<?>
}
```

Atributos públicos

Atributo privado

Se pueden acceder desde una instancia

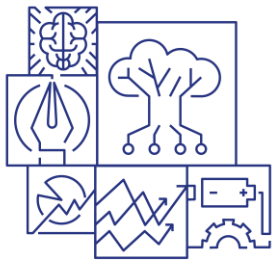


fidélitas Universidad

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**.



fidélitas Universidad

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;
    }
```

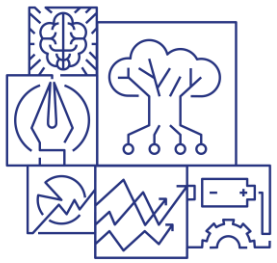
Los encapsuladores pueden tener cualquier nombre, pero se utiliza la preposición set y get para identificarlos.

fidélitas Universidad

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?

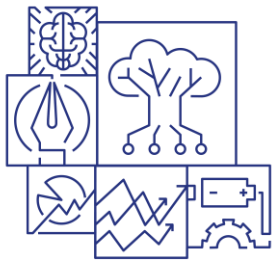


fidélitas Universidad

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.



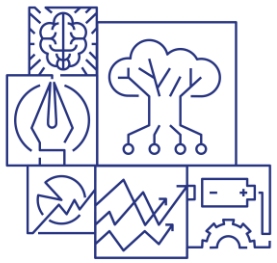
fidélitas Universidad

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.



fidélitas Universidad

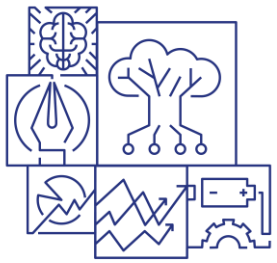
Constructores

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 debe programarse en la clase con el código que se especifique.

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

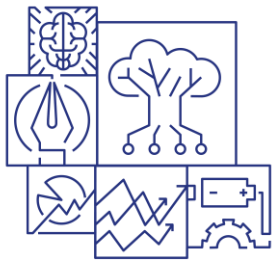
Constructor de la clase Curso



Constructores (ejemplo)

```
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



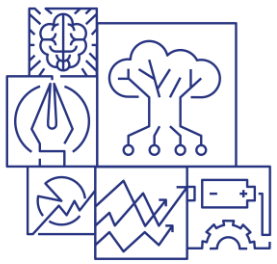
fidélitas Universidad

Constructores

```java
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.



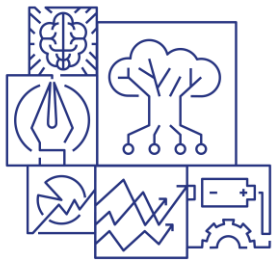
fidélitas Universidad

Constructores

```
Curso curso = new Curso();

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

Se crea el objeto con un ID al azar

Se crea el objeto con el ID 20

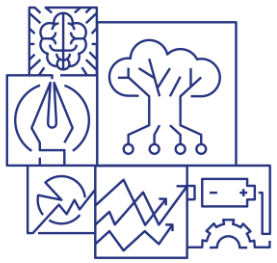


fidélitas Universidad

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).



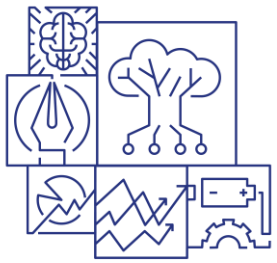
fidélitas Universidad

Clases como atributos

Hasta el momento hemos utilizado atributos con tipos simples (enteros, String, double, etc.), pero en muchas ocasiones se requiere un atributo que almacene datos más complejos. Un atributo de tipo de una clase.

```java
package com.mycompany.ejemplo;

public class Vehiculo {
    public String placa;
    public Motor motor;
}
```

```java
package com.mycompany.ejemplo;

public class Motor {
    public String numero;
    public int cilindrada;
    public String marca;
    public int cilindros;
}
```



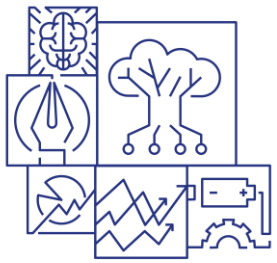
fidélitas Universidad

Clases como atributos

De igual manera en el constructor podríamos crear una nueva instancia para este atributo.

```
package com.mycompany.ejemplo;

public class Vehiculo {
    public String placa;
    public Motor motor;

    public Vehiculo() {
        this.motor = new Motor();
    }
}
```

Instancia de la clase Motor

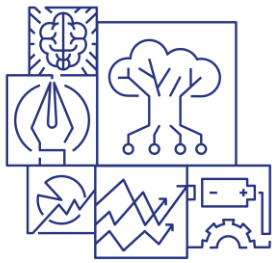


fidélitas Universidad

Clases como atributos

De igual manera en el constructor podríamos crear una nueva instancia para este atributo.

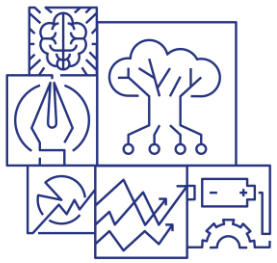
```
public static void main(String[] args) {
    Vehiculo micarro = new Vehiculo();

    micarro.placa = "123123";
    micarro.motor.marca = "Suzuki";
    micarro.motor.cilindros = 4;
    micarro.motor.cilindrada = 1500;
    micarro.motor.numero = "SUZ3276351";
}
```



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.

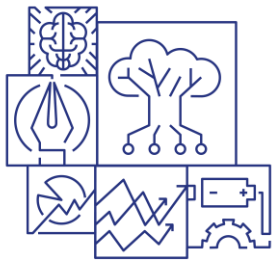
En algunas casos podemos tener clases en las cuales tenemos la certeza que una única instancia es lo que requerimos o que alguno de sus elementos debe compartirse entre todas las instancias.



Elementos estáticos

```
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.

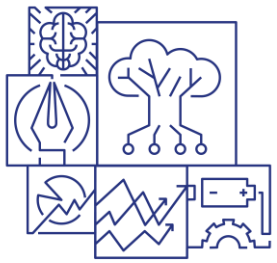


fidélitas Universidad

Elementos estáticos

```
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.

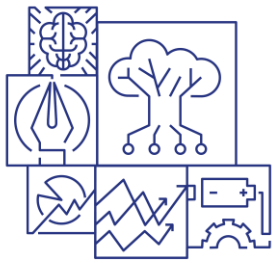


fidélitas Universidad

Enumeraciones

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
}
```



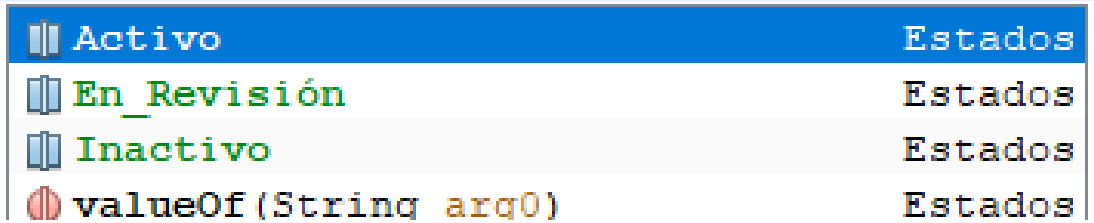
fidélitas Universidad

Enumeraciones

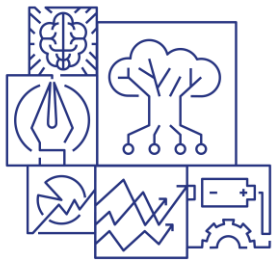
```
public int id = 0;
public String nombre = "";
private String correo = "";
public Estados estado;
```

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

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



Se puede seleccionar uno de los 3 posibles

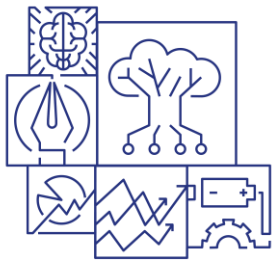


fidélitas Universidad

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.



Cada uno de estos componentes relacionados con la **Programación Orientada a Objetos**, nos permite implementa mejores y más simples estructuras de programación. La selección de las mismas dependerá de las condiciones y de nuestras habilidades para su identificación, es por esto que la práctica constante es importante.

fidélitas
Universidad