UNIVERSIDAD NACIONAL DE COLOMBIA
FACULTAD DE INGENIERÍA
ESTRUCTURAS DE DATOS
TALLER 1

Juan David Gaitán Forero (jdgaitanf@unal.edu.co)
Karen Atsumi Osako (kaosako@unal.edu.co)
Nelson Felipe Moreno Gómez (nfmorenog@unal.edu.co)

Abstracción

Acción y efecto de abstraer. Abstraer, es el proceso por el cual se pretende representar cualquier objeto del mundo real. Esto se logra al caracterizar dicho objeto de alguna manera alejándolo del contexto en el que pueda encontrarse.

En programación el término significa exactamente lo mismo, es transferir características de un objeto de la vida real a código. En la programación Orientada a Objetos esto se logra mediante el uso de clases.

Ejemplo: Una silla ¿Qué características podemos mencionar, tiene cualquier silla, independiente del contexto en el que esté?
Color, altura, peso y material, entre otras, pueden ser algunas de las características que podemos mencionar. No dependen de si la silla está en una escuela o una cárcel (por ejemplo); para todas las sillas en general podríamos mencionar dichas características.

Transferir dichas características a código en una clase se hace de la siguiente manera:

```
public class Silla {
    private String color;
    private String material;
    private int peso;
    private int altura;
}
```

Clase

Es una abstracción de un objeto del mundo real especificando sus atributos o características y métodos.

```
public class Person
{
     private String id;
     private String name;
```

```
	//Constructors
	public Person(String id, String name){
		this.id = id;
		this.name = name;

	}
	public Person(){

	}
	//Getters
	public String getId(){
		return id;
	}
	public String getName(){
		return name;
	}

}
```

Objeto

Es un elemento que representa una entidad abstracta o concreta, un objeto contiene sus atributos y métodos.

Ejemplo: Un objeto puede ser una bicicleta, que contiene de atributos llantas, frenos, material entre otros atributos propios, y el método puede ser informar el tipo de freno, el tipo de llantas que también puede ser un objeto o el material:

```
Class Bicicleta{
	private string freno;
	private string tipoLlantas;
	private int nLlantas;
	pubic Bicicleta() {
		freno="xxx";
		tipoLlantas="sss";
		nLLantas="2";
	}
}
Bicicleta Bici1 = new Bicicleta();//este sería el objeto de tipo
bicicleta
```

Atributo

Cuando hablamos de características nos referimos a detalles que, en conjunto, diferencian a un objeto de otro y dan cuenta de su aparencia, estado y otras particularidades de dicho objeto.

En programación estos atributos se guardan en variables que se denominan variables de instancia y, aunque se declaran en la clase, cambian y son particulares para cada objeto.

Ejemplo: Los atributos de una silla son las características que mencionáblamos cuando nos referíamos al ejemplo de “Abstracción”. En el ejemplo, en cursiva, se detallan los que son atributos de la clase Silla.

```
public class Silla {
    private String color;
    private String material;
    private int peso;
    private int altura;
}
```

Método

Es un conjunto de instrucciones que están definidas en una clase y realizan una determinada tarea, esta se puede usar escribiendo su nombre y puede retornar algún resultado o simplemente no retornar datos.

```
class Ejemplo{

      public static int fibonacci(int n)
      {
            if(n<2) return 1;
            else
                  return fibonacci(n-1) + fibonacci(n-2);
      }
}
```

Y para usarla escribimos

```
Ejemplo: fibonacci(5);//esto retorna la sucesión de fibonnaci hasta el 5.
```

Encapsulamiento

El encapsulamiento la subdivisión de una entidad en varias partes menores que contienen sus propios atributos y métodos que no se pueden acceder directamente. De modo a proteger sus características para que no se pueda acceder directamente, así solamente a través de métodos se puede realizar alteraciones.

Ejemplo: el modo como se protege una variable de la entidad, es utilizando el prívate, de modo que solo se puede realizar cambios sobre esta variable dentro de la clase a través de métodos propios, un ejemplo es cuando se tiene un sistema de control de finanzas, que no son todos que pueden tener acceso a los valores:

```
private int totalMoney;
public int get valueTotalMoney(){
      return totalMoney;
}
public int set valueTotalMoney(int  value){
      this.totalMoney = value;
```

```
}
```

Herencia

Podríamos decir que en general, la palabra “herencia” hace referencia a algo que se transmite. En programación, el término hace referencia a lo mismo: son un conjunto de atributos y métodos que se transfieren de una clase (Primaria o padre) a otra (denominada clase hija). Es necesario entender que la relación entre estas clases responde a la pregunta ¿Es [la clase hijo] un/una [clase padre]? donde [clase padre] y [clase hijo] se refieren a los objetos que buscamos relacionar; de otra manera, la relación que buscamos no es de herencia.

Ejemplo:

Para el siguiente ejemplo crearemos la clase padre “persona” con los atributos nombre, apellidos y edad. Luego crearemos la clase hijo “profesor” que *heredará* todas los atributos de persona, pero añadirá uno más: IdProfesor.

Código: (Tomado de aprenderaprogramar.com)

```
//Código de la clase Persona ejemplo aprenderaprogramar.com
public class Persona {
    private String nombre;
    private String apellidos;
    private int edad;
    //Constructor
    public Persona (String nombre, String apellidos, int edad) {
        this.nombre = nombre;
        this.apellidos = apellidos;
        this.edad = edad;                   }
    //Métodos
    public String getNombre () { return nombre;  }
    public String getApellidos () { return apellidos;  }
    public int getEdad () { return edad;   }
} //Cierre de la clase

//Código de la clase profesor, subclase de la clase Persona ejemplo
aprenderaprogramar.com
public class Profesor extends Persona {
    //Campos específicos de la subclase.
    private String IdProfesor;
    //Constructor de la subclase: incluimos como parámetros al menos los
del constructor de la superclase
    public Profesor (String nombre, String apellidos, int edad) {
        super(nombre, apellidos, edad);
        IdProfesor = "Unknown";   } //Cierre del constructor
    //Métodos específicos de la subclase
    public void setIdProfesor (String IdProfesor) { this.IdProfesor =
IdProfesor;   }
    public String getIdProfesor () { return IdProfesor;   }
```

```
    public void mostrarNombreApellidosYCarnet() {
        // nombre = "Paco"; Si tratáramos de acceder directamente a un
campo privado de la superclase, salta un error
        // Sí podemos acceder a variables de instancia a través de los
métodos de acceso públicos de la superclase
        System.out.println ("Profesor de nombre: " + getNombre() + " " +
getApellidos() +
         " con Id de profesor: " + getIdProfesor() ); }
} //Cierre de la clase
```

Polimorfismo

Es la habilidad de 2 o más objetos de diferente tipo y de responder exactamente al mismo método pero de forma diferente.

```
public class Animal {

      public void Hablar(){
            System.out.println("nada que decir");
      }
}

public class Dog extends Animal {

      @Override//sobreescribe el metodo
      public void Hablar(){
            System.out.println("Guau!");
      }
}                          /*los 2 animales responden al mismo metodo
                           hablar() de forma diferente */

public class Cat extends Animal {

      @Override
      public void Hablar(){
            System.out.println("Meoww!");
      }
}
```

Diferencia entre tipo primitivo y referencia

Una variable tipo primitivo, puede guardar solamente un valor de su tipo especificado, cuando se asigna otro valor a esta misma variable, el valor en su interior será sustituido, generalmente son inicializadas con “0”, las variables de tipo referencia son utilizadas haciendo referencia a la localización en la memória de un objeto, estos objetos pueden contener varias variables primitivas y también métodos, generalmente son inicializadas con “null”.

Ejemplo:

```
int cantidadManzanas = 0;
String tipoManzana = null;
cantidadManzana = 50; //Su valor es sustituido por 50
tipoManzana = “manzana orgánica”; //se asigna un conjunto de caracteres
para describir su tipo
```

Diferencia entre usar private, protected, public, default

private – Privado. Solo la clase actual tendrá acceso al atributo o método.

protected – Protegido. Solo la clase a la que pertence el método o atributo, tendrá acceso a él, además tendrán acceso subclases.

public – Público. Cualquier clase puede tener acceso al método o atributo.

default – Por defecto. Permite que solo la misma clase y otras del mismo paquete tengan acceso a los atributos y métodos.

Ejemplo de private y public.

```
public class Vehiculo {
    private String modelo;
    private int velocidad;
    private boolean encendido;
        public Vehiculo(String modelo, int velocidad, boolean encendido)
{
        this.modelo = modelo;
        this.velocidad = velocidad;
        this.arrancado = encendido;
    }   …
```

Ejemplo de protected:

```
public class Parent {
    protected int x = 9; // Solo las subclases o clase hijo tendrán
acceso a esta variable.
}
```

Ejemplo de default:

```
package paquete;

public class Parent {
    default int x = 9; //Solo las clases del paquete “paquete” tendrán
acceso a esta variable.
}
```

Clase abstracta:

La clase abstracta se diferencia de una clase normal solo en que no pueden ser instanciadas y nos sirven para crear métodos generales que recrean un funcionamiento en común pero sin identificar como los hacen. Su único fin es ser extendido a otras clases es decir que se usan para ser las clases padre y luego en la herencia se usa el polimorfismo para modificar el comportamiento de los métodos.

Ejemplo:

```
public abstract class Figura {//abstract define el tipo de la clase y no
puede ser instanciado

      // Atributos:
      public int numeroLados;
      public int area;
      public int volumen;

      // Métodos:
      abstract public void calcularArea();
      abstract public void calcularVolumen();
}
public class Esfera extends Figura{

    public float radio;
    public static float pi = (float)(3.1415);

    public Esfera( int radio ){
        this.radio = radio;
        this.numeroLados = 0;
    }

    // 4*pi*r²
    @Override
    public void calcularArea(){

        this.area =  (int) ((4)*pi*radio*radio);
        System.out.println(area);
    }

    // (4/3)*pi*radio³
    @Override
    public void calcularVolumen(){

        this.volumen =  (int) ((4/3)*pi*radio*radio*radio);
        System.out.println(volumen);

    }

}
```

Interfaz

La interfaz es un conjunto de acciones que deben ser ejecutadas, cada clase puede ejecutarlas de manera distinta, la interfaz contiene valores constantes o valores asignados a través de los métodos, que deben ser implementados dentro de la clase, generalmente son clases complementares abstractas, define lo que la clase deberá hacer.

```
interface Interfaz {
      public void metodoAbstracto();
      public String otroMetodoAbstracto();
}public class Principal implements InterfacePrincipal{
      public void metodoAbstracto() {
}public String otroMetodoAbstracto() {
      return “retorno";
}
```