# Programación orientada a objetos en Java

Si alguien se quiere retirar, adelante, pero...
¿de verdad crees que ya lo sabes todo de clases, objetos e instancias?

# POO

- Modelo de programación que basa su esquema de pensamiento, análisis y diseño en interacciones entre objetos. Cada objeto tiene un comportamiento definido y que puede ser relacionado con otros objetos.
  - Colaborativos y cooperativos
  - Reusables
  - Distribuíbles
  - Localizables

# Objeto

- Un **objeto** es un ente que tiene estado, comportamiento e identidad englobados en una sola unidad.
- La estructura de un objeto deriva del concepto de tipo de dato abstracto.

# Clase

- Una **clase** es la definición formal de un objeto en los términos de estructura y comportamiento común.
- Así podemos usar la definición de una clase para crear objetos de ese tipo de clase, esto es, crear objetos que contengan todos los componentes especificados en la clase (instancias).

# Encapsulamiento

- Se llama **encapsulamiento** a la conjugación de propiedades y comportamiento de un objeto.
- Esto logra también que se oculte la implementación y variables de ese objeto.

# Componentes de una Clase

- Una definición formal de una clase se compone de:
  - Campos. Estos son variables que almacenan datos referentes al objeto.
  - Funciones. Estos son las operaciones que se pueden realizar sobre objetos de esa clase. También son conocidos como **métodos**.
- Los campos pueden ser tipos de datos primitivos u objetos. Los métodos se asemejan a la estructura formal de las funciones. Tanto los campos como los métodos se les considera **miembros**.

# Clase en JAVA

- Una clase en Java se define mediante la palabra reservada **class** y enseguida, el identificador de la clase. Las propiedades y el comportamiento se definen dentro del cuerpo de la clase.

```
class NombreClase {
    //Propiedades
    ...
    //Comportamiento
}
```

# ... Clase en JAVA

```
class MiClase {
    tipo1 miVariable1;
    tipo2 miVariable2;
    ...
    tipoK miMetodo1(tipoK1 arg1, ...) {
    }
    tipoL miMetodo1(tipoL1 arg1, ...) {
    }
}
```

# Instancias

- Un objeto es también conocido como una instancia de la clase a la que pertenece. Entonces al crearse la instancia, el objeto contendrá los campos definidos en la clase.
- Los miembros pueden clasificarse como:
  - Miembros de instancia, y
  - Miembros de clase

# Miembros de instancia y clase

- Miembros de instancia
  - Cada objeto tendrá su propia copia local de cada variable definida en clase
  - Estas variables existen cuando se genera la instancia
- Miembros de clase
  - Son variables que existen en la clase y solo existe una sola copia para todas las instancias.
  - El valor es compartido y el mismo para todas las instancias.
  - Estas variables existen AÚN que no exista ni una instancia de esa clase.

# ... Miembros de instancia y clase

```
public class Circulo {
    //variable de clase
    static double PI = 3.14;
    //variables de instancia
    double x;
    double y;
    double radio;
}
```

```
public class Circulo {
    //variable de clase
    static double PI = 3.14;
    //variables de instancia
    double x;
    double y;
    double radio;

    double area() {
        return PI*radio*radio;
    }

    static double pi() {
       return PI;
    }
}
```

# ... Miembros de instancia y clase

- Suponga la clase A con una variable de instancia y un método de clase:

```
public class A {
    int x;
    static int cuadradoX() {
    return x*x;
    }
}
```

- Esto marca error, ¿Por qué?

```
public class A {
    public int inc(int j) {
        ++j;
        return j;
    }
}
...
A a;
int j = 10, i;
...
i = a.inc(j);
```

- ¿Cuánto vale i y j?

# Acceso a campos y métodos

- Campos y métodos de instancia
  - `A a = new A();`
  - `...`
  - `a.i = 5;`
  - `a.imprime();`
- Campos y métodos de clase
  - `A.dato = 6;`
  - `A.ejecuta();`

# Constructores

- Un constructor es un método especial que no devuelve ningún tipo de dato, que posee el mismo nombre de la clase y que tiene la finalidad de:
  - Crear espacio en memoria para el objeto
  - Inicializar las variables de instancia

# ... Constructores

```
class Clock {
   int hour;
   Clock() {
      hour = 12;
   }
   void setHour(int hour) {
      this.hour = hour;
   }
}
```

# ... Constructores

- Cuando un objeto es declarado para su uso posterior, es imperativo **construir** el objeto mediante una llamada al constructor.

```
class A {
   A() {
   ...
   }
}
...
A a = new A();
```

```
class B{
   int b;
   B() {
      b = 12;
   }
}
...
B b = new B();
B c = b;
b.b = 15;
System.out.println(c.b); //¿?
```

# La variable this

- Esta variable implícita siempre se refiere a la instancia actual.
- Por medio de esta referencia, se pueden acceder a los campos y métodos del objeto en turno.

```
class Clock {
   int hour = 12;
   void setHour(int hour) {
      hour = hour;
   }
}
...
Clock c;
...
c.setHour(24);
```

- ¿Cuánto vale c.hour?
-

# ... La variable this

```
class Clock {
    int hour = 12;
    void setHour(int hour) {
        this.hour = hour;
    }
}
```

# Sobrecarga de métodos

- Hay ocasiones que resulta útil tener un mismo identificador de método para diferentes método con diferente funcionalidad. Ejemplo:

```
class Calculadora {
    double suma(double d1, double d2) {
      return d1 + d2;
    }
    int suma(int i1, int i2) {
      return i1 + i2;
    }
}
```

# ... Sobrecarga de métodos

- De la misma forma, es posible definir sobrecarga para los constructores creando un repertorio amplio de maneras de construir un objeto.

# Tarea

- Investigar las diferencias de trabajar con clases estaticas o con instancias de clases.