Programación orientada a objetos

Dada una cadena de caracteres, encontrar todos los dígitos y devolver la suma de éstos

```
int sumaDigitos(String cadena)
```

```java
public static int sumInts(String cadena){
    int res = 0;
    for(char letra : cadena.toCharArray())
        if(Character.isDigit(letra))
            res += Integer.parseInt(
                    Character.toString(letra));
    return res;
}
```

Abstracción: clases e instancias

¿Qué son `int`, `String`, `char`, `Character` e `Integer`?

Son **conceptos generales**

O *clases* (aunque `int` y `char` son primitivos)

¿Qué son `cadena`, `res` y `letra`?

Son **objetos específicos**: *instancias de clases/tipos/conceptos*

Abstracción: paso de mensajes

`s.toCharArray()` => "s, sos un String, ustedes saben convertirse en arreglos de chars, ¡hacelo!"

`Character.isDigit(c)`: los primitivos son *mudos* y necesitan *wrappers* (digámosles Chaperones).

¿Cómo funcionan `toCharArray` o `isDigit`?

¡A nadie le importa!

Abstracción: estado y comportamiento

En el mundo de los objetos hay dos cosas

Lo que son por dentro: *estado (propiedades)*

¿Qué hay dentro de un String?

Y lo que pueden hacer: *comportamiento (métodos)*

El resultado: programar diciendo *qué* hacer, no *cómo*

¿Para qué sirve?

Para preocuparte del *qué* y no el *cómo*

Si te olvidás del *cómo* de algo, podés hacer cosas más complejas

¿Qué hace la multiplicación? vs ¿cómo funciona?

object

Código infernal

Hacé un programa que maneje una lista de N personas y luego calcule el promedio de sus edades y determine cuántas de ellas se llaman "Karl" y son mayores de edad

```
//tienen que estar "sincronizados"
int[] edades = ;
String[] nombres;
int acum, karls;
for(int edad: edades)
    acum += edad;
System.out.println(acum/edades.length);
for(int i=0;i<edades.length;i++)
    if (edades[i] > 21
        && nombres[i].equals("Karl"))
        karls++;
System.out.println(karls);
```

¿Y si un arreglo se hace más pequeño/grande?

Ah, ahora tenés que guardar apellidos

¡No! La mayoría de edad es a los 18

Hacé lo mismo dentro de unos meses

¿No sería mejor algo así?

```
Persona[] personas;
int acum, karls;
for(Persona p: personas){
    acum += p.edad;
    if(p.esMayorDeEdad() && p.seLlama("Karl"))
        karls++;
}
System.out.println("...");
```

Ok, me convenciste, ¿cómo se hace?

```
class NOMBRE_DE_CLASE{
    //PROPIEDADES
    //MÉTODOS
}
```

```java
public class Persona{
    String nombre;
    int edad;

    public boolean esMayorDeEdad(){
        return this.edad >= 21;
    }

    public boolean seLlama(String n){
        return this.nombre == n;
    }
}
```

¿Y cómo creás personas nuevas?

```
String s = "hola"
String s = new String(new char[]{'h','o','l','a'});
String p = new Persona("Karl", 60);
```

Un método especial: el constructor

```
public class Persona{
    //Propiedades...
    public Persona(String n, int e){
        this.nombre = n;
        this.edad = e;
    }
    //Métodos...
}
```

¿Qué acabamos de hacer?

Una definición de clase dice qué podrá *ser y hacer* cada instancia

El *constructor* es un método especial: dice cómo hacer una instancia

Las *propiedades de instancia* definen el estado

Los *métodos de instancia* definen el comportamiento

`this` se refiere a la *instancia actual*

(Cada instancia tiene su propio estado)

Definiendo clases

Si no ponés constructor, se crea uno por defecto

Todos los métodos pueden estar sobrecargados

A veces es necesario imprimirlos: `toString`

A veces es necesario ver si son iguales: `equals`

Miembros de instancia vs. miembros de clase

Cuando ponés `static`, decís: todas las instancias van a compartir esto, y no tendrán su propia copia

```java
class Persona{
    int personasCreadas;
    int edad;
    public Persona(){
        System.out.println("Personas: "+personasCreadas);
    }
}
//...
class Main{
    public static void main(String[] args){
        Persona a,b,c;
        a = new Persona(); //Personas: 1
        b = new Persona(); //Personas: 1
        c = new Persona(); //Personas: 1
    }
}
```

```java
class Persona{
    static int personasCreadas;
    int edad;
    public Persona(){
        System.out.println("Personas: "+personasCreadas);
    }
}
//...
class Main{
    public static void main(String[] args){
        Persona a,b,c;
        a = new Persona(); //Personas: 1
        b = new Persona(); //Personas: 2
        c = new Persona(); //Personas: 3
    }
}
```

Referencias vs Valores

Una variable es una *referencia* a un objeto

I.e.: dónde puedo encontrarlo

`a==b` se pregunta si están el mismo lugar

Referencias

MIT: "java preparation for 6.170"

MIT: "introduction to programming in java"

Referencia oficial de Java: "Learning the java language"

Princeton: "Introduction to programming in java"