PROGRAMACIÓN ORIENTADA A OBJETOS

TEMA1: Introducción

Manel Guerrero

Java se parece mucho a C...

IGUAL

- int, double, float, char.
- +, -, *, /, %, ++, --, ==, !=, >=, <, &&, ||, !.
- if else, for, while, do while, switch case.
- int funcion(tipo_arg nombre_arg) { ...

  return n;}

DIFERENTE

- boolean = true o false y byte.
- Un solo fichero (*.c (pri) y *.h (pub) -> *.java).
- Pero las cosas pueden ser «public» o «private».
- En lugar de un exe, un «bytecode» .class y JVM.
- printf, scanf y String.

... pero sin todo lo que nos traía problemas.

- Las variables se inicializan automáticamente a 0/null/false. Esto evita «segmentation faults» porque hemos olvidado una inicialización.
- No más &, *, ni ->.
- Porque no hay punteros...

  Bueno esto no es del todo cierto... Hay «referencias».
- Y si nos salimos de un vector el programa peta y nos dice a que posición estábamos intentando acceder.
- Hay una manera de saber el tamaño de un vector.

De hecho ya casi sabemos programar en Java.

```
/* Factorial.java */
public class Factorial {
    // C con "public static" delante
    public static int factorial(int n) {
        int i, f;

        for (f = 1, i = 2; i <= n; i++) {
            f *= i;
        }
        return f;
    }
    public static void main(String[] args) {
        int x = 6;
        System.out.println(x +
            "! = " + factorial(x));
    }
}

    // "String[] args" es nuevo
    // println() diferente a printf()
```

Ahora sólo falta aprender la parte de objetos, ...

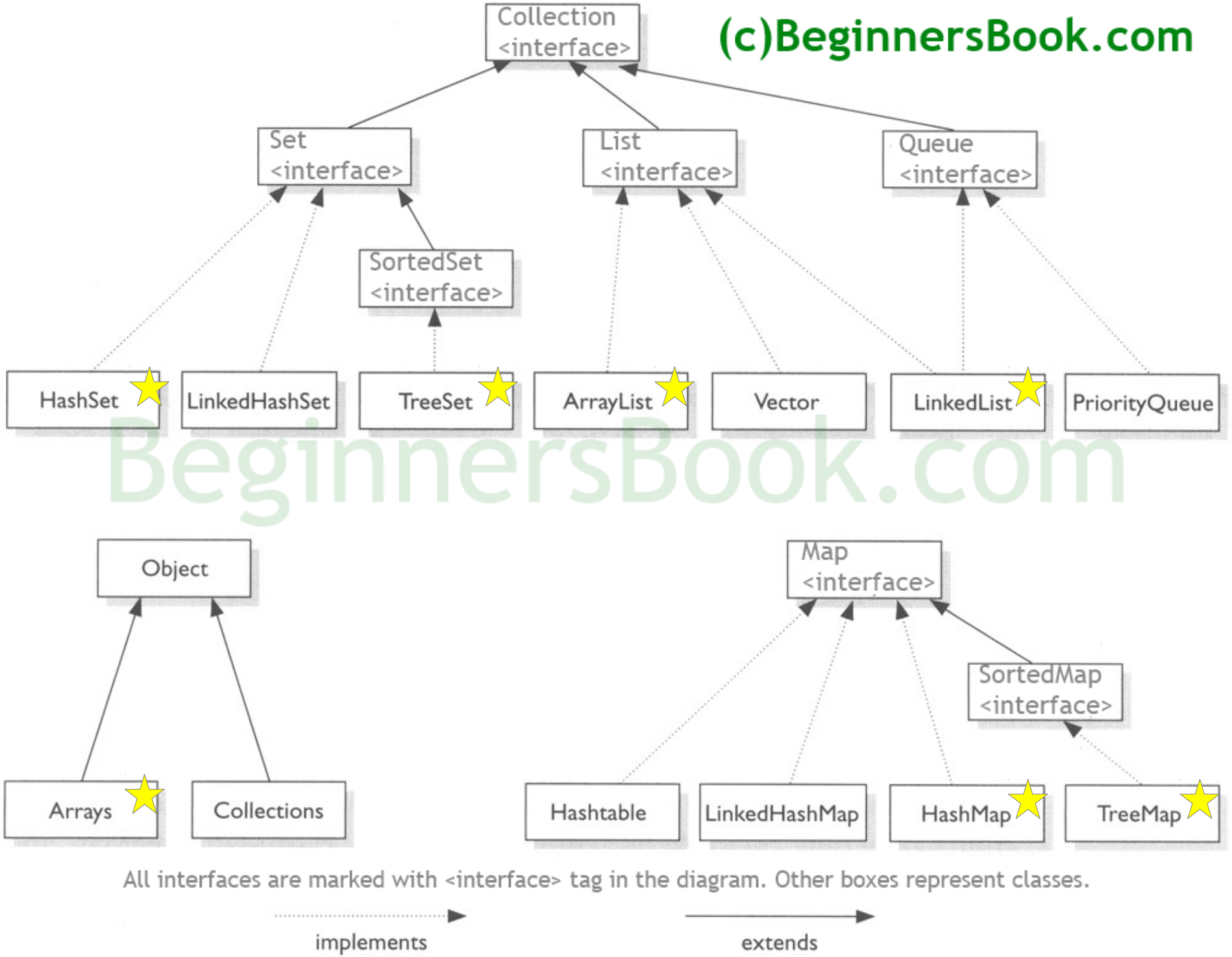
- C es un lenguaje de programación por procedimientos (procedural programming) y Java es Programación Orientada a Objetos (POO/OOP).
- Nos falta aprender que es todo esto de:

  clase, objeto o instancia, atributo, método, etc.
- Veremos que se parecen mucho a:

  struct, variable de tipo struct, campo de struct, función.
- (Un objeto es una instancia de una clase, por lo que se pueden intercambiar los términos objeto o instancia.)

y ver parte de su gran librería de funciones que nos facilita la vida.

- A la librería le llamamos la API de Java (Application Programming Interface).
  - Contiene todas classes que se nos dan hechas (es como los includes de C).
  - https://docs.oracle.com/javase/8/docs/api/
- E incluye herramientas como ahora: arrays (vectores), conjuntos, listas, mapas (diccionarios), etc.
  - A estas herramientas que agrupan objetos se les llamada collections.
  - http://beginnersbook.com/java-collections-tutorials/

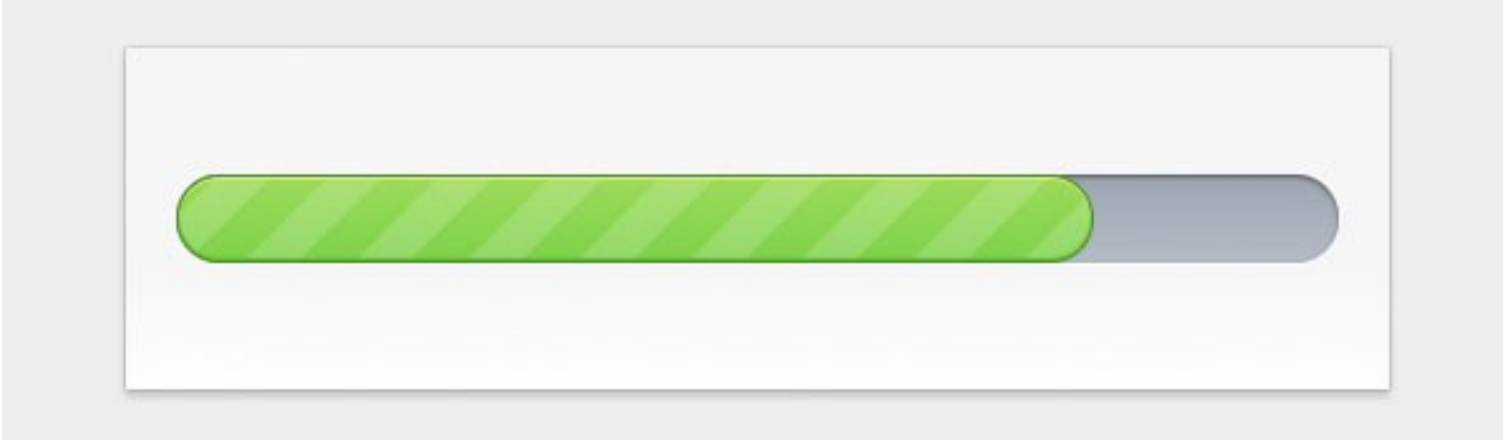

(c)BeginnersBook.com
Collection <interface>
Set <interface>
List <interface>
Queue <interface>
SortedSet <interface>
HashSet
LinkedHashSet
TreeSet
ArrayList
Vector
LinkedList
PriorityQueue
BeginnersBook.com
Object
Map <interface>
SortedMap <interface>
Arrays
Collections
Hashtable
LinkedHashMap
HashMap
TreeMap
All interfaces are marked with <interface> tag in the diagram. Other boxes represent classes.
implements
extends

Pero, en menos de una hora...

... ya sabéis el 70%

de cómo programar en Java.

Barra de progreso:



Pero, empecemos por el principio

- C es un lenguaje de bajo nivel (se parece a como “piensa” un ordenador).
- Java y los lenguajes de POO son de alto nivel (se parecen más a como piensa una persona).
- En Java:
  - Todo es un objeto (o un tipo de objeto). Y, cada objeto es de un tipo.
  - Un objeto puede utilizar otros objetos.
  - Un objeto tiene estado (su memoria con sus variables y otros objetos).
  - Todos los objetos del mismo tipo pueden ser utilizados de la misma forma.

Versiones de Java

Release	Year
JDK Beta	1994
JDK 1.0	1996
JDK 1.1	1997
J2SE 1.2	1998
J2SE 1.3	2000
J2SE 1.4	2002
J2SE 5.0	2005
Java SE 6	2006
Java SE 7	2011

Release	Year
Java SE 8	2014
Java SE 9	2017
Java SE 10 (18.3)	3/2018
Java SE 11 (18.9 LTS)	9/2018

Java 9 is the main publicly supported version (with Java 7 no longer publicly supported and Java 8 end of support announced).

https://en.wikipedia.org/wiki/Java_version_history

OK… OK… A la práctica:

- Recordad el proyecto de programación de FO.
- Ahora cada cosa.c & cosa.h se convierte en cosa.java.
- Lo que estaba en cosa.h escribimos «public».
- Lo que estaba en cosa.c escribimos «private».
- Los campos del typedef struct ahora son atributos del objeto.
- Las funciones ahora son métodos de la clase.

Piedra de Rosetta C/Java

point.h

```
typedef struct {
    double x;
    double y;
} t_point;
void invert_coordinates(t_point *p);
void print_point(t_point p);
```

point.c //falta includes stdio y point

```
void invert_coordinates(t_point *p) {
    double aux = p->x;
    p->x = p1->y; p->y = aux;
}
void print_point(t_point p) {
    printf("(%lf,%lf)", p.x, p.y);
}
```

Point.java

```
public class Point {
    public double x; // atributo
    public double y; // atributo
    public Point(double x, double y) {
        this.x = x; this.y = y;
    } // constructor
    public void invertCoordinates() {
        double aux = x;
        x = y; y = aux;
    } // metodo
    @Override
    public String toString() {
        return "(" + x + "," + y + ")";
    } // metodo usado por print
} // Sin *, &, -> :-)
```

Piedra de Rosetta C/Java (2)

calcula.c

```
#include <stdio.h>
#include "point.h"
void main() {
    t_point p1 = {3,2};
    printf("Point 1: ");
    print_point(p1); printf("\n");
    printf("I invert
coordinates\n");
    invert_coordinates(&p1);
    printf("Point 1: ");
    printf("(%lf,%lf)\n", p1.x,
p1.y);
}
```

PointTester.java

```
public class PointTester {
    public static void main(String[]
args) {
        Point p1 = new Point(3, 2);
        System.out.print("Point 1: ");
        System.out.println(p1);
        System.out.println("I invert
coordinates");
        p1.invertCoordinates();
        System.out.print("Point 1: ");
        System.out.println("(" + p1.x
                + "," + p1.y + ")");
    }
}
```

Ahora solo Java

Point.java

```
public class Point {
    public double x;
    public double y;
    public Point(double x, double y) {
        this.x = x; this.y = y;
    }
    public void invertCoordinates() {
        double aux = x;
        x = y; y = aux;
    }
    @Override
    public String toString() {
        return "(" + x + "," + y + ")";
    }
} // Sin *, &, -> :-)
```

PointTester.java

```
public class PointTester {
    public static void main(String[]
args) {
        Point p1 = new Point(3, 2);
        System.out.print("Point 1: ");
        System.out.println(p1);
        System.out.println("I invert
coordinates");
        p1.invertCoordinates();
        System.out.print("Point 1: ");
        System.out.println("(" + p1.x
                + "," + p1.y + ")");
    }
}
// UpperCamelCase & lowerCamelCase
```

Y el scanf()?

```
import java.util.Scanner; // Include


public class AreaTriangleScanner {
    public static void main(String[] args) {
        double base, altura, area;
        String nombre;
        Scanner in = new Scanner(System.in);

        System.out.print("¿Como te llamas? ");
        nombre = in.nextLine();
```

Y el scanf()? (2)

```
        System.out.println("Hola " + nombre + "! Vamos a
calcular el area de un triangulo. :-)");
        System.out.print("- Base: ");
        base = in.nextDouble();
        System.out.print("- Altura: ");
        altura = in.nextDouble();
        area = base * altura / 2;
        System.out.println("El area de un triangulo de base
" + base + " y altura " + altura + " es " + area + ".");
    } // nextLine(), nextInt(), nextFloat(), nextDouble(),
}     // next() para leer una palabra
```

PROGRAMACIÓN ORIENTADA A OBJETOS

Preguntas

PROGRAMACIÓN ORIENTADA A OBJETOS

Veamos ahora los códigos W1.

t4_sin_cos_b.c / MyBadMath.java

Series de McLaurin notables

https://es.wikipedia.org/wiki/Serie_de_Taylor

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} \quad , \forall x; n \in \mathbb{N}_0$$

$$\sin x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1} \quad , \forall x$$

$$\cos x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} x^{2n} \quad , \forall x$$