Cheatsheet - Introducción a la Programación

ACM UPM

Compilar y ejecutar

```
javac Mi_fichero.java
java  Mi_fichero
```

Tipos y variables

tipo	*bits*
int	32
float	32
double	64
char	16
Boolean	`true | false`

Declarar una variable (*[ ] significa que puede estar o no*):

```
tipo nombre_variable [= valor];
double distancia; // Declara sin valor
double distancia = 3.3; // Declara con valor
```

Operadores

		Aritméticos
+	+=	variable++
-	-=	- -variable
*	*=	
/	/=	
%	%=	
		Lógicos
&&	||	
		Relacionales
==	!= <= >=	

Control de flujo - Switch

```
switch (variable_a_comparar) {
  case valor: instrucción; [break;]
  default: ;
}
```

Control de flujo - Condicionales

También conocido como "control de flujo". Permite cambiar *qué* se ejecuta según ciertas *condiciones*.

```
if (condición) {
  haz esto;
}

if (condición) {
  haz esto;
}else{ // Si no se cumple la condición
  haz lo otro;
}

if (condición){
  haz esto;
}else if(condición2){ //Si no, si...
  haz esto otro;
}
```

Los condicionales se pueden anidar *ad infinitum*. Equivalencia en condicionales:

```
if (condición) {
  if (condición2){ }
}

if (condición && condición2) { }
```

Bucles - for y while

```
for ([declarar var];[cond];[op tras bloque]){}
while([condición]){}
```

Serían válidos por tanto:

```
for(;;){}
while(true){}
```

Correspondencia entre bucles:

```
for (int i=0;i<10;i++){
  System.out.println(i);
}

int i=0;
while(i<10){ System.out.println(i); i++;}
```

Funciones

```
(tipo|void) nombre ([argumentos]){}
int duplicar (int x) { return x*2; }
void no_devuelve () { System.out.println("<3");


class Clase {
  public static void main (String[] args) (
    //Código a ejecutar al llamarse con java
  }
}
```

La función puede tener *modificadores* como: `public`, `private`, `static`. Esto queda fuera del temario de Prog I.

Anexo

13	CR	71	G	100	d
32		72	H	101	e
40	(	73	I	102	f
41	)	74	J	103	g
42	*	75	K	104	h
43	+	76	L	105	i
44	,	77	M	106	j
48	**0**	78	N	107	k
49	1	79	O	108	l
50	2	80	P	109	m
51	3	81	Q	110	n
52	4	82	R	111	o
53	5	83	S	112	p
54	6	84	T	113	q
55	7	85	U	114	r
56	8	86	V	115	s
57	9	87	W	116	t
65	**A**	88	X	117	u
66	B	89	Y	118	v
67	C	90	Z	119	w
68	D	**97**	**a**	120	x
69	E	98	b	121	y
70	F	99	c	122	z

Unete a ACM

Arrays

```
Creación de arrays:
new tipo[n] //Crea un array vacío de tamaño n
<nombre_array> = {e1, e2, ...., en}

new tipo[] {e1, e2, ..., en}
//Crea un array con los elementos designado.


Tamaño de un array:
    Los arrays empiezan en 0 y acaban en -1.
    Se puede comprobar su longitud con:
        <nombre_del_array>.length;
```

Manejo de Strings

Funciones Útiles	Descripción
int length();	Devuelve la longitud
char charAt(int i)	Devuelve el carácter en i

```
Si queremos concatenar dos Strings
 debemos usar el operador '+':

String x = "Cha" + "chi"; // x = Chachi

Se pueden comparar dos strings
usando .equals(String).

s1 = "Chachi";
s2 = "Cha" + "chi";
s1 == s2; // Devuelve false
s1.equals(s2) // Devuelve true
```

Clases e instancias

Los modificadores indican a que clases, atributos o métodos se puede acceder

	Clase	Paquete
Private	OK	NO
sin modificador	OK	OK
protected	OK	OK
public	OK	OK
	Herencia	**Otros**
Private	NO	NO
sin modificador	NO	NO
protected	OK	NO
public	OK	OK

Herencia

```
 public class Padre {
   public int X() {
   }
}
public class Hijo extends Padre {
   public int X() {
     // Este método pertenece al padre,
     // pero el hijo es capaz de heredarlo,
     // aunque también es capaz de
     // sobreescribir.
   }
}

- Upcasting de un tipo a otro cuando se tiene
la certeza de que sea interpretable.
- Downcasting de un tipo a otro cuando no se
está seguros de que sea interpretable.

Vehiculo obj1 = new Coche();
Coche c1 = (Coche) obj1;      // Downcasting

*La clase String extiende de CharSequence*
String str = "";
CharSequence chs = str;       // Upcasting

- Existe el operador instanceof que permite
comprobar si el objeto de la izquierda es
hijo del de la derecha.

Vehiculo v1 = new Coche();
Vehiculo v2 = new Camion();

v1 instanceof Coche; // true
v2 instanceof Camion; // true
v2 instanceof Coche; // false

- Las clases abstractas están implementadas
parcialmente, algunos métodos los tienen que
implementar las clases hijas.

public abstract class Multiplicador {
   public double multiplicar(double numero) {
       return base * altura;
    }

    public abstract double factor();
}
```

Interfaces

```
Las interfaces solo definen los métodos,
no el cuerpo

public class Cubo implements Figura3D {
    private double lado;

    public Cubo(double lado) {
        this.lado = lado;
    }

    public double volumen() {
        return lado * lado * lado;
    }
}
```

Identidad vs. Estado

```
La identidad es la región de memoria que
ocupa nuestra instancia y, por tanto, es
algo único a cada una. Se puede comparar
con el operador ==.

Coche c1 = new Coche("1234ABC", "rojo");
Coche c2 = c1;

c1 == c2; //true


El estado es el conjunto de los valores de
los atributosde una instancia. Solo se puede
comparar con .equals si antes se ha
modificado el método.

*Una vez cambiado el método .equals*
Coche c1 = new Coche("1234ABC", "rojo");
Coche c2 = new Coche("1234ABC", "rojo");

c1.equals(c2); // true
```