CUATROVIENTOS

Lenguaje Java

1º Desarrollo de Aplicaciones Multiplataforma

Sistemas Informáticos

1. Introducción

Ejercicio 1

¿Cuáles son las propiedades principales del lenguaje de programación java? ¿A qué lenguajes se parece?

Ejercicio 2

Crea con Eclipse un proyecto llamado *HelloWorld*, que tenga una clase llamada *HelloWorld.java* como esta:

```
public class HelloWorld {
	public static void main(String[] args) {
		System.out.println("Hello World!");
	}
}
```

Compílalo y ejecútalo.

1 Variables

Ejercicio 1

¿Qué tipo de variable usarías para estos valores?

a) Tu edad
b) La temperatura de un día de invierno
c) Un sueldo en euros
d) Tu altura en centímetros
e) La nota de un examen
f) Una letra del alfabeto
g) El nombre de una persona

Ejercicio 2

Crea un proyecto con una clase llamada *Variables* en el que declares todas las variables del ejercicio anterior, usando el tipo más adecuado y un nombre de variable descriptivo.

Ejercicio 3

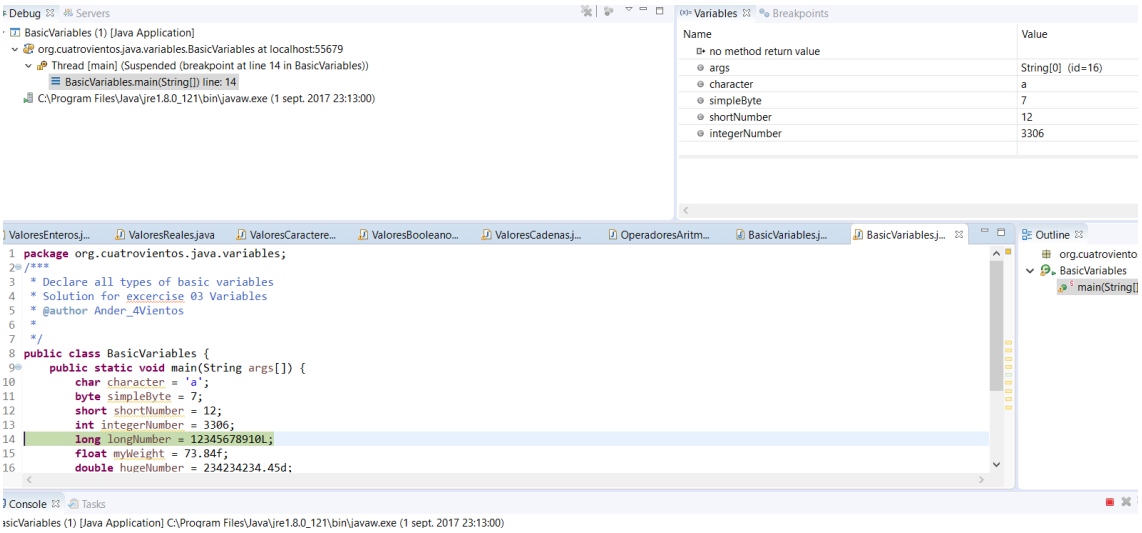
Crea un proyecto con una clase llamada *BasicVariables* en el que definas e inicialices una variable de todos los tipos básicos de java.

Ejercicio 4

Depura los dos proyectos anteriores y comprueba paso a paso los valores que toman las variables. Para ello, abre su código, introduce un *breakpoint* haciendo *click* en el lateral del editor y utiliza *Debug* en lugar de *Run* para ejecutar el programa:

```java
package org.cuatrovientos.java.variables;
/***
 * Declare all types of basic variables
 * Solution for excercise 03 Variables
 * @author Ander_4Vientos
 *
 */
public class BasicVariables {
    public static void main(String args[]) {
        char character = 'a';
        byte simpleByte = 7;
        short shortNumber = 12;
        int integerNumber = 3306;
        long longNumber = 12345678910L;
        float myWeight = 73.84f;
        double hugeNumber = 234234234.45d;
        boolean iLoveYou = true;
        String sayMyName = "kuh-THOO-lhoo";
    }
}
```

En la perspectiva *debug* pulsa F5 o el botón de *Step into* e irás viendo los valores asignados a las variables:



Ejercicio 5

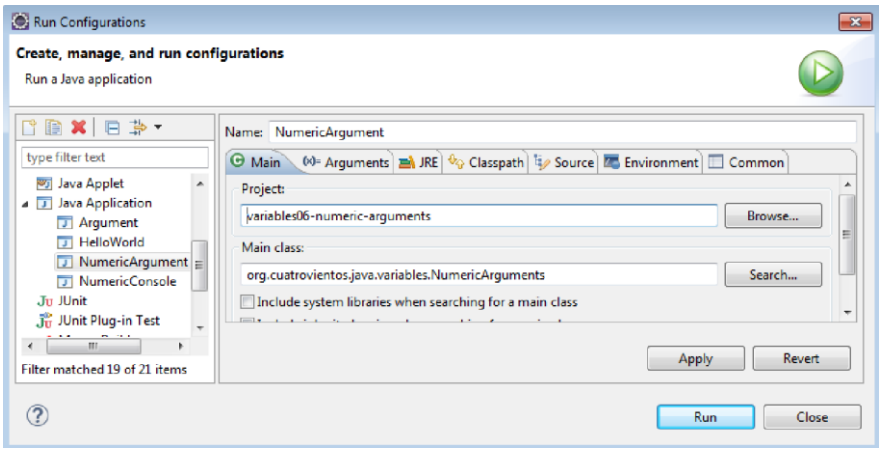
Crea un proyecto con una clase llamada *NumericConsole* en el que definas todos los tipos de datos numéricos y asígnales un valor desde la consola (`consola.readLine()`...). Luego debes mostrar los valores recogidos por consola. Puedes partir de este código, donde se muestra lo necesario para leer un número entero por la consola:

```
Scanner readFromConsole = new Scanner(System.in);
String line = "";
System.out.println("Enter an integer number:");
line = readFromConsole.nextLine();
int number = Integer.parseInt(line); System.out.println("You introduced: " + number);
```

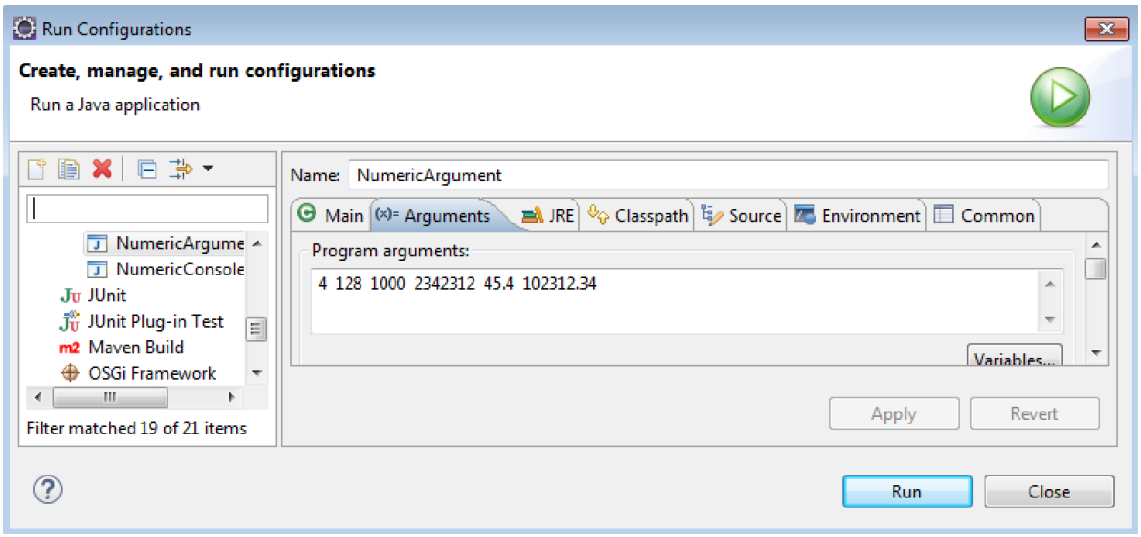
Ejercicio 6

Crea un proyecto con una clase llamada *NumericArguments* en el que definas todos los tipos de datos numéricos (como en el anterior) y asígnales un valor desde los argumentos de *main (`args[0]`, `args[1]`,...).*

Recuerda que para pasar los argumentos deberás ir al menú *Run As* > *Run configurations*...



Y en *Arguments* añadir los valores necesarios para el programa:



Ejercicio 7

¿Son correctos estos nombres de variables? Crea un proyecto con una clase llamada *CheckVariables* que las declare para comprobarlo:

a) 666la_bestia
b) labestia666
c) la.bestia.666
d) LABESTIA
e) Labestia
f) la-bestia
g) brujería

h) 1. Entrada/Salida básica

Ejercicio 1

Declara 4 variables (entero, decimal, carácter y cadena) e inicializalas.

Ejercicio 2

Imprime por pantalla el valor de estas 4 variables acompañadas de un texto identificativo (Ejm. Esto es un entero: 4).

Ejercicio 3

Crea un proyecto que se llame EntradaPorConsola que lea un entero, un decimal, una cadena y un booleano y los imprima por pantalla.

Ejercicio 4

Crea un proyecto que se llame GestiónAlumnos que pida al usuario Nombre, Apellido, Ciclo y Curso y lo imprima por pantalla de manera ordenada.

2 Operadores

Ejercicio 1

¿Qué resultados se obtienen en las siguientes operaciones? Crea un proyecto con una clase llamada *CheckOperators* que lleve a cabo las siguientes operaciones y muestre el resultado. Si **alguna es incorrecta, indícalo con un comentario y no la hagas**.

a) 10 - 7;
b) 40 + 2;
c) 10 * 4 + 2
d) 56 % 4
e) (100 / 4) + 25 – (16 / 2)
f) 39.56F + 3 * 100
g) (2++) + 5 / 3

Puedes empezar así, definiendo una variable que se ajuste a la operación y luego mostrándolo con `System.out.println`:

```
int result;
result = 10 - 7;
System.out.println(result);
```

Ejercicio 2

¿Qué resultados se obtienen de las siguientes operaciones booleanas? Crea un proyecto con una clase llamada *CheckOperatorResult* que lleve a cabo las siguientes operaciones. Luego ejecútalo en el depurador.

int x = 4;
int y = 6;

1. (x > 0)
2. (x > 0) || (y>7)
3. (x <= 4) && (y != 4)
4. !((x < 4) && (y >42))
5. !(x < 4) || !(y >42)
6. x == 4 && y == 7

Puedes empezar así:

```
int x = 4;
int y = 6;
boolean result = false;
result = (x > 0);
System.out.println(result);
```

Ejercicio 3

Crea un proyecto con una clase llamada *Average* que solicite por consola 5 números al usuario y calcule la media de esos valores. Toma números enteros y luego comprueba si obtienes una media precisa.

Puedes empezar así:

```
Scanner readFromConsole = new Scanner(System.in);
String line = "";
int number1 = 0;
int number2 = 0;
int number3 = 0;
int number4 = 0;
int number5 = 0;
float result = 0;
System.out.println("Please enter a number:");
line = readFromConsole.nextLine();
number1 = Integer.parseInt(line);
```

Ejercicio 4

Crea un proyecto con una clase llamada *Converter* que solicite por consola al usuario un valor en dólares y lo convierta a euros.

Ejercicio 5

Crea un proyecto con una clase llamada *Booleans* que solicite al usuario dos valores booleanos, lleve a cabo la operación and y or y muestre el resultado. La ejecución podría ser así:

```
Please enter true/false
true
Please enter true/false
false
AND operation
true and false is: false

OR operation
true or false is: true
```

Ejercicio 6

Crea un proyecto con una clase llamada *MassBodyIndex* que solicite por consola dos valores. Los valores deben ser el peso en kilos y la altura en centímetros de una persona y calcule el Índice de Masa Corporal. Este valor se obtiene dividiendo el peso por el cuadrado de la altura. Luego multiplícalo por 10000 para sacar un número más legible por pantalla. **Utiliza variables *float*** para una mayor precisión.

3 Estructuras de control

Ejercicio 1

Crea un proyecto con una clase llamada *OddOrEven* que solicite al usuario un número entero y muestre por pantalla si ese número es par o impar. Usa el operador % para comprobar si ese número es par o impar.

Ejercicio 2

Crea un proyecto con una clase llamada *PositiveNegativeOrZero* que solicite al usuario un valor entero y muestre por pantalla si ese número es positivo, negativo o 0.

Ejercicio 3

Crea un proyecto con una clase llamada *PositiveEven* que solicite al usuario un número entero y muestre por pantalla si ese número es par y positivo. En caso contrario debe indicar si es negativo, impar o ambos.

```
Introduce un número:
10
Positivo y par
```

```
Introduce un número:
-3
Negativo y es impar
```

Ejercicio 4

Crea un proyecto con una clase llamada *CompareNumbers* que solicite al usuario dos valores enteros, los compare y muestre por pantalla si uno es mayor que el otro o si son iguales.

Ejercicio 5

Crea un proyecto con una clase llamada *Multiple* que solicite al usuario dos valores enteros y muestre por pantalla si el primero es múltiplo del segundo.

Ejercicio 6

Crea un proyecto con una clase llamada *MassBodyIndexDiagnostic* que solicite al usuario su peso en kilos y su altura en centímetros y calcule el IMC (peso / altura$_2$); debe mostrar el resultado y luego hacer el diagnóstico:

Si el IMC es menor que 16 se muestra el mensaje: "You need to eat more".

Si el IMC está entre (>=)16 y 25(<) se muestra el mensaje: "You are fine"

Si el IMC está entre 25 y 30(<) se muestra el mensaje: "You are eating too much".

Si el IMC es superior a 30 se muestra el mensaje: "Go to hospital"

Ejercicio 7

Crea un proyecto con una clase llamada *LanguageMessage* que solicite al usuario un nombre de lenguaje y saque un mensaje distinto según el nombre de ese lenguaje:

- Si el lenguaje es "C++" el mensaje a sacar será "The best language ever"
- Si el lenguaje es "Java" el mensaje a sacar será "The second best language ever"
- Si el lenguaje es "JavaScript" el mensaje a sacar será "The language of the present"
- Si es cualquier otro debe decir "Nothing to say about that"

Ejercicio 8

Crea un proyecto con una clase llamada *Position* que solicite al usuario un dorsal de jugador y haga lo siguiente: comprobar que ese número está entre 0 y 99. Y a continuación el programa debe mostrar un texto con la posición que corresponde a cada dorsal:

- Si el usuario ha tecleado 1 el texto será "Keeper"
- Si el usuario ha tecleado 3,4,5 el texto será "Defender"
- Si el usuario ha tecleado 6, 8, 11 el texto será "Midfield"
- Si el usuario ha tecleado 9 el texto será "Striker".

- Para cualquier otra opción el texto será “Any”.

Ejercicio 9

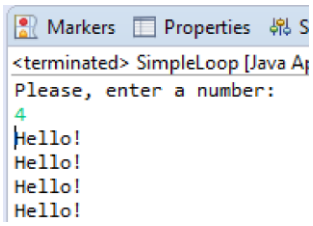
Crea un proyecto con una clase llamada *CurrencyConverter* que solicite al usuario una cantidad monetaria y un carácter d, p, y (dollar, pound, yen). Según el carácter introducido por el usuario el programa debe convertir la cantidad monetaria (que serán euros) a la moneda correspondiente.

No es preciso que el tipo de cambio sea real.

4 Estructuras repetitivas

Ejercicio 1

Crea un proyecto con una clase llamada *SimpleLoop* que solicite al usuario un valor entero y comprueba si es mayor que 0. Si no lo es debes mostrar un mensaje de advertencia al usuario y volver a pedirlo tantas veces como sea necesario. Tras esto mostrará por pantalla un saludo tantas veces como el valor del número.



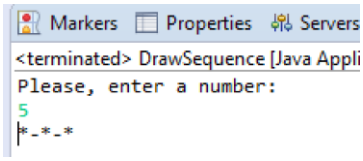
Ejercicio 2

Crea un proyecto con una clase llamada *DrawStars* que solicite al usuario un valor entero positivo y además par. Si el valor introducido no cumple los requisitos debes mostrar un mensaje de advertencia al usuario y volver a pedirlo las veces que sea necesario. Cuando sea válido mostrará por pantalla una línea con el carácter "*" (asterisco) tantas veces como el valor del número. Debes usar `System.out.print("*");` Por ejemplo, si introduce un 8 mostrará: ********

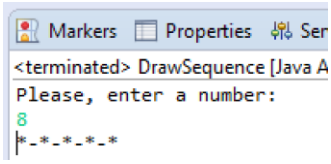
Ejercicio 3

Crea un proyecto con una clase llamada *DrawSequence* parecido al anterior pero la línea que debes mostrar debe tener este aspecto: *-*-*-*-*-*-*-*-*-*-* Y siempre debe terminar en "*"

Por ejemplo, si introducen el 5:



Si introducen el 8:



Ejercicio 4

Crea un proyecto con una clase llamada *DoWhile* que solicite al usuario una palabra y hasta que esa palabra no sea "out!" el programa no debe terminar y debe seguir solicitando una palabra.

Ejercicio 5

Crea un proyecto con una clase llamada *DrawSquare* que solicite al usuario un número entero y usando ese valor debe "dibujar" en la consola un cuadrado formado por "*". Por ejemplo, si introduce 4 se mostrará:

****
****
****
****

Ejercicio 6

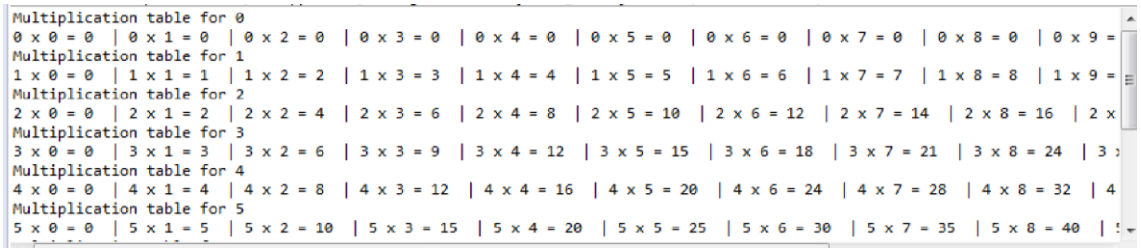
Crea un proyecto con una clase llamada *Factorial* que solicite al usuario un número entero y calcule su factorial. Por ejemplo, el factorial de 5 sería 5 x 4 x 3 x 2 x 1 = 120

Ejercicio 7

Crea un proyecto con una clase llamada *PrimeNumber* que solicite al usuario un número entero y comprueba si ese número es primo o no, es decir si solamente es divisible por sí mismo o por 1.

Ejercicio 8

Crea un proyecto con una clase llamada MultiplicationTable que muestre todas las tablas de multiplicar desde el número 0 al 10.



5 Arrays

Ejercicio 1

Crea un proyecto con una clase llamada *ShowArray* que defina un array de 10 números enteros y luego muestre cada elemento en un bucle.

Ejercicio 2

Crea un proyecto con una clase llamada *NameArray* que defina un array de 10 Strings vacíos y luego en un bucle solicite al usuario que introduzca cada elemento. Después muestra todos los elementos con otro bucle.

Ejercicio 3

Crea un proyecto con una clase llamada *NumberArray* que defina un array de 10 números enteros y luego en un bucle solicite al usuario que introduzca cada elemento. Luego crea otro bucle que determine si en el array hay algún elemento repetido. Con que encuentre uno repetido es suficiente. Si no hay ninguno deberá indicarlo con otro mensaje.

Ejercicio 4

Crea un proyecto con una clase llamada *IncrementArray* que defina un array de 10 números enteros y que en un bucle solicite al usuario que introduzca cada elemento. Luego crea otro bucle que incremente en uno cada uno de los elementos y los muestre.

Ejercicio 5

Crea un proyecto con una clase llamada *AverageArray* que defina un array de 10 números con decimales (float) y luego en un bucle solicite al usuario que introduzca cada elemento. Luego crea otro bucle que calcule la media de todos los números.

Ejercicio 6

Crea un proyecto con una clase llamada *CountArray* que defina un array de 10 números enteros y luego en un bucle solicite al usuario que introduzca cada elemento. Luego crea otro bucle que contabilice el total de números positivos, negativos y los que sean 0.

Ejercicio 7

Crea un proyecto con una clase llamada *CheckPrimes* que defina un array de 10 números enteros y luego en un bucle solicite al usuario que introduzca cada elemento. Luego crea otro bucle que descubra que números son primos.

Ejercicio 8

Crea un proyecto con una clase llamada *RandomArray* que defina un array de 10 elementos. Crea un bucle que inicialice los valores del array usando números aleatorios:

```
// Variable para generar números aleatorios, incluye import java.util.*; Random rnd = new Random();
int miAleatorio rnd.nextInt(30);
// número aleatorio entre 0 y 30
```

Después de eso crea otro bucle que si encuentra el número 15 en algún elemento interrumpa el bucle y muestre la posición en la que está.

```
Random value inserted in 0: 28
Random value inserted in 1: 19
Random value inserted in 2: 23
Random value inserted in 3: 5
Random value inserted in 4: 12
Random value inserted in 5: 27
Random value inserted in 6: 1
Random value inserted in 7: 20
Random value inserted in 8: 15
Random value inserted in 9: 19
Number 15 was found at: 8
```

Vamos a variar la versión anterior para hacer que el *array* sea de dos dimensiones y almacene 5x5 elementos. De la misma manera inicializarás todas las posiciones con números aleatorios e indicarás si el número 15 ha sido encontrado en alguna posición, en cuyo caso se interrumpirá el bucle. El resultado sería:

```
Random value inserted in [0][0] 18
Random value inserted in [0][1] 18
Random value inserted in [0][2] 17
Random value inserted in [0][3] 3
Random value inserted in [0][4] 19
Random value inserted in [1][0] 2
Random value inserted in [1][1] 14
Random value inserted in [1][2] 0
Random value inserted in [1][3] 2
Random value inserted in [1][4] 12
Random value inserted in [2][0] 11
Random value inserted in [2][1] 9
Random value inserted in [2][2] 8
Random value inserted in [2][3] 2
Random value inserted in [2][4] 3
Random value inserted in [3][0] 28
Random value inserted in [3][1] 6
Random value inserted in [3][2] 14
Random value inserted in [3][3] 18
Random value inserted in [3][4] 15
Random value inserted in [4][0] 27
Random value inserted in [4][1] 27
Random value inserted in [4][2] 0
Random value inserted in [4][3] 14
Random value inserted in [4][4] 28
Number 15 was found at posición [3][4]
```

Ejercicio 9

Crea un proyecto con una clase llamada *ShuffleArray* que defina un array de 10 números enteros y luego en un bucle solicite al usuario que introduzca cada elemento. En un bucle muestra por pantalla todos los elementos. Luego crea otro bucle que baraje los elementos usando el método *nextInt* del ejercicio 8 en los índices (es decir, aleatoriamente saca un índice del array, y luego otro. Intercambia el contenido del *array* para ese índice con el del otro). Finalmente muestra el resultado.

```
Generated values:
0: 18
1: 25
2: 12
3: 15
4: 29
5: 19
6: 8
7: 2
8: 24
9: 28
After shuffle:
0: 28
1: 2
2: 15
3: 24
4: 18
5: 19
6: 29
7: 12
8: 25
9: 8
```

Ejercicio 10

Crea un proyecto con una clase llamada *DimensionalArray* que defina un array bidimensional de 5x5 números enteros. Posteriormente deberás solicitar al usuario que introduzca cada elemento:

```
Pos [0][0] - Enter a number: 2
Pos [0][1] - Enter a number: 3
Pos [0][2] - Enter a number: 4
Pos [0][3] - Enter a number: 5
Pos [0][4] - Enter a number: 6
Pos [1][0] - Enter a number: 7
Pos [1][1] - Enter a number: 8
Pos [1][2] - Enter a number: 9
Pos [1][3] - Enter a number: 0
Pos [1][4] - Enter a number: 1
. . .
```

El resultado mostrará la media de cada una de las filas:

```
La media de la fila 0 es: 4
La media de la fila 1 es: 5
La media de la fila 2 es: 14
La media de la fila 3 es: 10
La media de la fila 4 es: 6
```

Ejercicio 11

Crea un proyecto con una clase llamada *Menu* que tenga las siguientes opciones:

```
Introduzca la opción:
1. Generar array
2. Buscar número
3. Borrar número
4. Salir
```

1. **Generar array**: se generará un array de longitud 5 con números aleatorios de entre 0 y 10.
2. **Buscar número**: se pide un número por consola y se indica si se encuentra o no en el array anterior.
3. **Borrar número**: se pide un número por consola y se sustituye en todas las posiciones en las que se encuentre por el número 0. En caso de que no se encuentre, debe mostrar un mensaje indicándolo.

4. **Salir**: finaliza el programa.

Nota: las opciones 2 y 3 no deberían poder ser realizadas si la 1 no se realiza antes. Así que en caso de que escojan la 2 ó 3, sin haber sido generado el array, se indicará un mensaje de error y se volverá a mostrar el menú.

6 Clases

Ejercicio 1

Crea un proyecto con una clase llamada *Hello* con un atributo `String greet (iniciado con un valor concreto)` y un método `sayHello()` que muestre ese atributo por consola. Incluye el método *main* y crea una instancia de la clase para probarla.

Ejercicio 2

Crea un proyecto con una clase llamada *Coin*. La clase debe tener un único método llamado `flip` cuyo resultado debe ser aleatoriamente un número entero, que, en función de su valor, mostrará: "CROSS" o "PILE", es decir, cara o cruz. Incluye el método *main* y tira la moneda 5 veces para ver como cae cada vez.

Ejercicio 3

Crea un proyecto con una clase llamada *Conversor* que tenga varios métodos para convertir monedas. Incluye el método `main` y crea una instancia de la clase para probarla.

Para definir cada uno de los cambios podrías utilizar una constante. Esto lo veremos más adelante, pero para definir constantes en Java:

```
private static final double CHANGE_PESETAS_EUROS = 166.386d;
private static final double CHANGE_DOLLARS_EUROS = 0.9d;
private static final double CHANGE_POUNDS_EUROS = 0.8d;
```

Estos son los métodos que se piden:

```
1) public double pesetas2Euros(double amount)
2) public double euros2Pesetas(double amount)
3) public double euros2Dollars(double amount)
4) public double dollars2Euros(double amount)
5) public double euros2Pounds(double amount)
6) public double pounds2Euros(double amount)
```

Ejercicio 4

Crea un proyecto con una clase llamada *Array* que sirva para crear un array de enteros. La clase debe contener un atributo para contener el array de 10 elementos y los siguientes métodos:

- `public Array()`: constructor. Se encarga de invocar (llamar) al siguiente método: *init*.
- `private void init ():` inicia el array con números aleatorios. Se le llama desde el constructor
- `public void increment():` incrementa cada elemento del array en 1.
- `public void decrement():` decrementa cada elemento en 1
- `public int countEven():` devuelve el número de elementos pares en el array

Añade una clase *Main* con un método *main* que cree una instancia de Array, muestre el total de números pares, incremente, muestre el total de números pares, decremente y muestre el total de números pares. Debería volver al valor inicial:

Ejercicio 5

Crea un proyecto con una clase llamada *Dice* para simular el comportamiento de un dado de N caras. Estos serán los métodos de la clase:

- `private int sides`: atributo que guarda el número de caras
- `private boolean allowZero = false`: atributo que nos dice si el dado puede devolver el valor 0. Por defecto vale false.
- `public Dice ()` : constructor sin parámetros, asigna a sides = 6
- `public Dice (int sides)` : constructor con parámetro, establece el atributo sides
- `public Dice (int sides, boolean allowZero):` constructor con parámetros, establece los dos atributos.
- `public int roll ():` método que simula el lanzamiento del dado y retorna un entero con el resultado. Debe tener en cuenta al atributo `allowZero`.

Para probarlo, crea una clase principal con método *main* que genere un dado de 6 caras, un dado de 10 caras y un dado de 20 que permita ceros. Haz 100 lanzamientos de cada uno:

Ejercicio 6

Crea un proyecto con una clase llamada *Square* que nos servirá para dibujar cuadrados de caracteres por la consola. Estos son los métodos de la clase:

- `private char character; // El "carácter" que usaremos para dibujar`
- `public Square (): // constructor sin parámetros, asigna a la variable carácter = '#'`
- `public Square (char character):` constructor parametrizado para asignar al atributo carácter lo que queramos
- `public void setCharacter (char character):` método *setter* para cambiar el carácter una vez creada la instancia.
- `private String generate (int size):` método que genera el cuadrado según el tamaño pasado y lo retorna como *String*. Por ejemplo: `generate(4)` retorna:

```
####
####
####
####
```

- `public` void show (`int` size): método público que lo único que hace es invocar al método `generate ()` y mostrar por consola el resultado.

Crea una clase principal con un método *main* que genere algunos cuadrados y los muestre por pantalla.

Ejercicio 7

Crea un proyecto con una clase llamada *Names*, que servirá para generar nombres:

- `private int length`: atributos para la longitud de los nombres.
- `public Names ()`: constructor sin parámetros. Establece la longitud a un valor que elijas.
- `public Names (int length)`: constructor que establece el atributo longitud al valor recibido.
- `public String generate()`: método que genera el nombre y lo retorna como String. El nombre podrá empezar con vocal o consonante. Para ello crea un *array* de consonantes y otro de vocales, vete obteniendo de forma aleatoria y alterna vocales y consonantes (vocal-consonante-vocal etc o

consonante-vocal-consonante etc) hasta que completes la longitud del nombre. Finalmente retórnalo.

Incluye el método `main` y crea varias instancias de la clase para probarla, en ocasiones indicando la longitud del mismo y en otras dejando que sea la inicial por defecto (es decir, trabaja con los dos constructores) en diferentes instancias.

Ejercicio 8

Crea un proyecto con una clase llamada *Nicknames*, que servirá para generar apodos y mostrarlos por pantalla.

- Para generar apodos puedes crear dos arrays de Strings, uno con la primera parte del apodo y otro con la segunda por ejemplo: `private String[] nicknameStart = {“Ojos”,”Puño”,”Espada”,”Viento”} private String[] nicknameEnd = {“Negro”,”de Fuego”, “del Infierno”,”Helada”}` así un apodo podría salir “*Ojos de Fuego*” o “*Espada Helada*”
- `public Nicknames ():` constructor sin parámetros
- `public String generate()` : método que genera el apodo y lo retorna como String

Crea una clase principal con un método ***`main`*** que cree una instancia de la clase *Names* del ejercicio anterior y otra de Apodo para generar 10 nombres completos con apodo. Reaprovecha el código de la clase Name del ejercicio anterior.

Este podría ser el código de la clase principal:

```
public static void main(String[] args) {
Nicknames nicknames = new Nicknames();
	Names names = new Names();
	String name = "";
	for (int i=0;i<5;i++) {
		name = names.generate() + " " + nicknames.generate();
		System.out.println(name);
	}
}
```

Y esta una posible salida:

```
syho Espada de Fuego
bylle Viento Helada
fomi Ojos Helada
gafi Viento Helada
xiwi Ojos del Infierno
```

Ejercicio 9

Crea un proyecto con una clase llamada *Passwords*, que servirá para generar *passwords* aleatorias y mostrarlas por pantalla.

- `private int length`: atributos para la longitud de las *passwords*.
- `public Password ()`: constructor sin parámetros. Establece el valor longitud con la que elijas.
- `public Password (int length)`: constructor que establece el atributo longitud con la longitud recibida.

- `public String generate()` : método que genera el *password* y lo retorna como String. Para generar *Passwords* puedes simplemente alternar letras, números y caracteres especiales. Para seleccionarlos aleatoriamente puedes incluirlos en un *array* o en varios, y sacarlos usando un índice aleatorio. Vete concatenándolos hasta llegar a la longitud.
- `public String generate(int qty)`: método que genera una cantidad (qty) de passwords y los retorna con saltos de línea entre cada uno.

Como siempre incluye una clase con el método *main* y crea varias instancias de la clase para probarla.

7 POO. Herencia

Ejercicio 1

Crea un proyecto llamado *Devices* que incluya las siguientes clases.

1. Clase **Device**: tiene los atributos protegidos String name, String brand y String price. Un constructor usando los atributos, los set y get y un método público toString mostrando los atributos.
2. Clase **Mobile**: es una subclase de Device, hay que añadir el atributo privado String number. Crea el constructor y el método toString aprovechando los de la superclase. Añade el método público call (int number), que saque por pantalla una cadena diciendo "calling *number*"
3. Clase **Computer**: es una subclase de Device, hay que añadir el atributo privado String processor. Crea el constructor y el método toString aprovechando los de la superclase
4. Clase main en la que creas un objeto de cada clase y muestras lo que escriba el método toString. Y para el caso del objeto Mobile hace una llamada a un número que te inventes.

Ejercicio 2

Crea un proyecto llamado School que incluya una serie de clases. Las clases tendrán las siguientes características.

1. Clase **Person**: tiene los atributos protegidos String *name*, int *age*. Un constructor vacío, otro constructor usando los dos atributos, los métodos set/get y un método toString()
2. Clase **Teacher**: es una subclase de Person. Debe tener el atributo privado String degree, y otro que sea String [] *subjects* (un array de asignaturas). Debe tener un constructor con todos los campos. Además, debe tener lo métodos set/get para los atributos y también un método toString generado automáticamente.
3. Clase **Student**: es una subclase de Person. Debe tener los atributos privado String course y String degree. Debe tener un constructor con todos los campos, métodos set y get y un método toString.
4. Clase **Main**: debe crear una instancia de Teacher y otra de Student y sacarlas por pantalla llamando a sus respectivos métodos toString().

Ejercicio 3

Crea un proyecto llamado *War* que incluya una serie de clases. Las clases tendrán las siguientes características.

1. Clase **Unit**: tiene los atributos protegidos String división y String name. Un constructor vacío, otro constructor usando los dos atributos, los métodos set/get y un método toString(). También debe tener un método público llamado fire() y otro llamado move() que devuelven un entero aleatorio menor que 10 o 100 respectivamente.
2. Clase **Tank** (hereda de Unit): tiene los atributos privados int armor, int ammo, String model. Un constructor con todos los parámetros, métodos set/get y un método toString(). También debe tener un método público llamado turn que devuelve un número aleatorio no superior a 360.

3. Clase **Soldier** (hereda de Unit) y tiene los atributos privados String rank, int age.Un constructor con todos los parámetros, métodos set/get y un método toString(). También debe tener un método llamado prone() inferior a 2.
4. Clase **Main**: debe tener el método main, crear una instancia de Tank y de Soldier y sacarlas por pantalla llamando a sus respectivos métodos toString().

Ejercicio 4

Crea un proyecto java llamado HelloKitty en el que la protagonista gestiona una Cesta de comida. La comida será de varios tipos. Estas son las clases que se deben hacer,

1. Clase **Food**: es una clase abstracta que tiene los atributos protegidos String nombre y float peso. Un constructor usando los atributos, los set y get y un método público toString mostrando los atributos.
2. Clase **Fruit**: es una subclase de Food, y hay que añadir el atributo privado String vitamina. Crea el constructor y el método toString aprovechando los de la superclase.
3. Clase **Candy**: es una subclase de Food y hay que añadir el atributo privado int calorías. Crea el constructor y el método toString aprovechando los de la superclase.
4. Clase **Basket**, tiene un atributo ArrayList de elementos tipo Food. Se inicializa en el constructor. Tiene los siguientes métodos públicos:
   - Uno que reciba un objeto de la clase Food y permita añadir esa comida a la cesta: void meterComida(Food food),
   - Otro que devuelva el peso total de la comida.
   - El método toString para mostrar toda la comida de la cesta.
5. Clase Principal, contiene el método main. En él se debe crear una instancia de la clase Basket, cargarla con comida y luego mostrar el contenido.

Ejercicio 4-Avanzado

Ahora deberás añadir los siguientes métodos en Food:

- equals indicando que 2 objetos de este tipo se consideran iguales siempre que les coincida el nombre (no importando como esté escrito respecto a mayúsculas o minúsculas) y de la clase a la que pertenezcan. Es decir, no puede haber una fruta con el mismo nombre que un caramelo.

En Basket

- Modifica el método meterComida para que devuelva un mensaje indicando si la ha añadido o no, porque ya estaba.
- Métodos que devuelvan la cantidad de piezas que hay de fruta, caramelos o comida en general.
- Crea un nuevo método que, recibiendo el nombre de una comida, permita borrar la que le corresponde. (devolviendo un mensaje que indique si lo ha podido borrar o no).

Modifica la clase principal, para que permita realizar las operaciones:

1. Inicialmente añadirá diferentes comidas a la cesta, mostrando el mensaje devuelto por la función indicando si la ha podido añadir o no. Pon objetos que cumplan todas las posibilidades.
2. Mostrar la cesta en ese momento y la cantidad de peso de toda la comida, así como la cantidad de cada uno de los tipos
3. Borrar comida de la lista, mostrando también los posibles mensajes.
4. Volver a escribir los datos anteriores

Ejercicio 5-Avanzado

Crea un proyecto java llamado JavaKart contenga las siguientes clases

2. Clase ***abstracta*** **Vehiculo**, que contiene los atributos protegidos nombre, velocidad, aceleración y agarre. Tiene un constructor con el parámetro nombre, los set y get para los atributos y un método toString generado automáticamente. Tiene estos métodos:
   - inicializar: asigna a los atributos velocidad, aceleración y agarre valores entre 0 y 10, de forma aleatoria.
   - mover: devuelve el resultado de la suma entre velocidad, aceleración y un valor aleatorio entre 0 y 5
   - maniobra: devuelve la suma del agarre y un valor aleatorio entre 0 y 5.
3. Clase **Coche**, que es una subclase o extiende la clase Vehículo, además de un atributo matrícula. Crea el constructor y el método toString aprovechando los de la superclase. Dos coches se consideran iguales si les coincide la matrícula (independientemente de cómo esté escrita)
4. Clase **Circuito**, que contiene los atributos privados nombre, y longitud. Crea el constructor con los campos y el método toString
5. Clase **Carrera**, con los atributos privados nombre, circuito (contiene un objeto de la clase Circuito) y un ArrayList de vehículos llamado participantes. Además de un constructor que incluya el nombre y una instancia de Circuito, debes incluir estos métodos:
   - meterVehículo: intenta añadir un objeto de la clase Vehículo recibido por parámetro en el vector de vehículos, añadiéndolo solo en el caso de que no existiese. Devuelve un mensaje indicando si lo ha añadido o no.
   - correr: Mediante los bucles que consideres oportunos, se hace "correr" a los vehículos, finalizando la carrera cuando uno haya conseguido llegar a la meta, es decir, recorrer todos los kilómetros. El método devuelve el vehículo ganador.
6. Clase **Principal**, contiene el método main. En él se debe crear una instancia de la clase Carrera con su nombre y Circuito, meter varios coches (alguno existente para comprobar que no lo ha añadido). Tras hacerlos correr mostrará los datos de la carrera, así como los del vehículo ganador.

Ejercicio 6-Avanzado

Crea un proyecto java llamado Net4flix contenga las siguientes clases.

1. Clase **Persona**, que contiene los atributos protegidos nombre y apellido. De esta clase no se crearán objetos
2. Clase **Actor**, que no tiene atributos propios, los hereda de Persona
3. Clase **Director**, hereda de persona y tiene un atributo nPeliculas que guarda el número de películas que ha dirigido. Tiene un método sumar película que suma uno a su nPeliculas
4. Clase **Audiovisual,** contiene un atributo título y una lista de actores, De esta clase no se crearán objetos. Su constructor no recibirá todos sus parámetros, los actores se introducirán a través de un método anyadirActor.
5. Clase **Serie**, hereda de Audiovisual y tiene como atributos la cantidad de temporadas y de capítulos (capítulos por temporada).
6. Clase **Película,** hereda de Audiovisual y tiene como atributo duración y un director.
7. Clase **Net4vflix**, tiene un ArrayList de Audiovisual. Tendrá los siguientes métodos:
   a. anyadir: añade un objeto Audiovisual al ArrayList
   b. mostrarPeliculas: Muestra todas las películas de la lista.
   c. mostrarSeries: Muestra todas las series de las películas de la lista.

   d. directorPelicula: Recibe el nombre de una película y devuelve el director de la película que tenga el nombre pasado como parámetro, si no existe la película se devuelve null
   e. capitulosSerie(String serie): Recibe el nombre de una serie y devuelve el total de capítulos de la serie que le corresponda, teniendo en cuenta todas las temporadas. Si no existe ninguna con ese nombre devuelve -1
8. Clase **AppNet4vflix**, clase principal que deberá probar todos los métodos de la clase Net4vflix

Todas las clases tendrán sus métodos get, set y toString y constructor con todos sus parámetros a excepción de Audiovisual como ya se ha indicado.

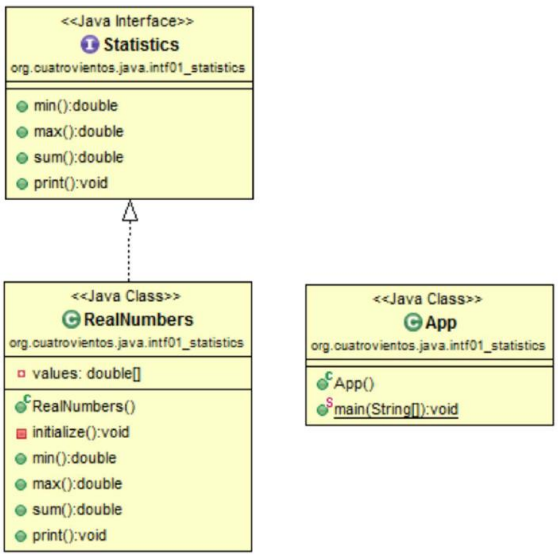
i) 8. Interfaces

Ejercicio 1

Construye una interfaz Statistics con los siguientes métodos. Será implementada por una clase RealNumbers, que contendrá un array de double.

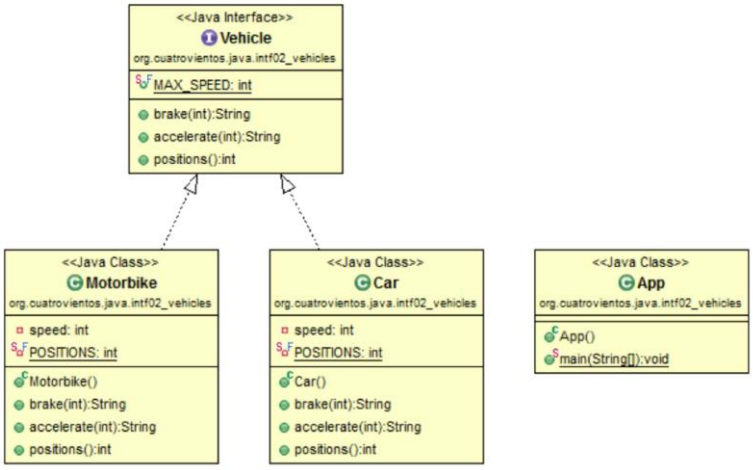
Dentro del constructor de la clase RealNumbers, crearás el array con una dimensión inicial de 5, y llamarás al método privado initilize(), que se encargará de inicializar las posiciones del array con números double aleatorios entre 0 y 99.

También implementa el resto de métodos de la interfaz:



Ejercicio 2

Vamos a representar el siguiente UML:



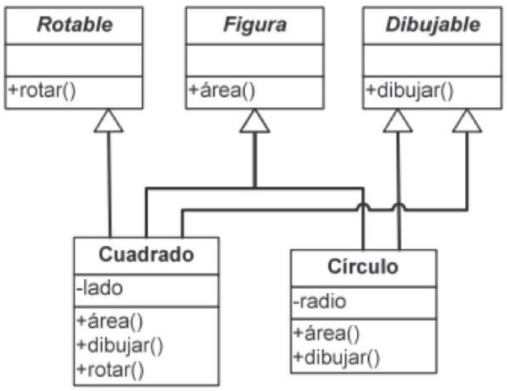
Crearemos una interfaz Vehicle. Fíjate que una interfaz solo debe contener métodos (públicos) sin implementar o bien CONSTANTES (o atributos con un valor inicial). MAX_SPEED vale 120.

Tanto Moto como Coche deben implementar la interfaz con sus características. El método frenar recibe la cantidad a frenar, y retorna un mensaje indicando la velocidad actual de la moto o el coche, una vez frenada. Lo mismo sucederá con acelerar, teniendo en cuenta que nunca podemos exceder la velocidad máxima. El método que devuelve las posiciones, en realidad retorna la constante de la clase (POSITIONS) que será inicializada en función del número de plazas del vehículo: Motorbike, 2, y Car, 4.

Finalmente, crea un objeto de tipo Motorbike y otro de tipo Car, dentro del main, y muestra el resultado de la llamada a sus métodos frenar y acelerar.

Ejercicio 3

Vamos a representar el siguiente UML:



La clase Figura es abstracta, luego su método area() será también abstracto. A su vez Rotable y Dibujable son interfaces. A continuación, representa la clase Cuadrado y Circulo dando una implementación a cada uno de sus métodos. La implementación de dibujar o rotar es libre, puedes simplemente mostrar un mensaje por pantalla indicando quien rota o es dibujado, ejemplo *rota el cuadrado de lado* ___y en cuanto al área, escribirá el área del objeto en concreto.

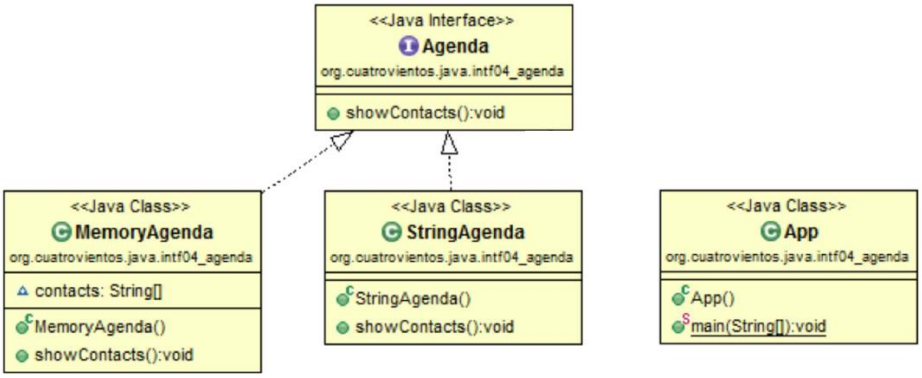
Para obtener el lado o radio, define su getter y setter o bien que la clase lo reciba en el constructor

Ejercicio 4

Crea un proyecto con una interfaz llamada Agenda que contenga el siguiente método:

```
public void showContacts()
```

En concreto este será el UML a representar:



Como ves, existe:

- Una clase **StringAgenda**, que implementará el método showContacts: Debe solicitar un contacto por pantalla y, simplemente, volver a mostrarlo.
- Una clase **MemoryAgenda**, que implementará el método showContacts: Esta clase tendrá un ArrayList que habrás inicializado con varios contactos (String con nombre del contacto). Este método se encargará mostrar los contactos del ArrayList.

Finalmente, prueba el método showContacts desde las distintas clases, en el método main.

j) 9. Excepciones

Ejercicio 1

Crea un proyecto con una clase Main, desde la cual vamos a hacer el tratamiento adecuado de las distintas excepciones. Te copio, a continuación, distintos fragmentos de código, que incluirás dentro de tu main y que son susceptibles de lanzar excepciones. Se pide que las captures, y muestres un mensaje acorde a dicha excepción.

a)

```
int a = 4;
int b = 0;
System.out.println(a / b);
```

b)

```
int[] array = new int[5];
array[5] = 1;
```

c) Prueba a introducir un decimal:

```
Scanner scn = new Scanner(System.in);
System.out.println("Introduce un número entero:");
int n = scn.nextInt();
System.out.println("Número introducido " + n);
```

Ejercicio 2

Define una referencia a un objeto e inicialízala a null. Trata de invocar un método a través de esta referencia. Ahora rodea el código con una cláusula try-catch para probar la excepción.

Ejercicio 3

Crea tu propia clase de excepción utilizando la palabra clave extends. Escribe un constructor para dicha clase que tome un argumento String y lo establezca. Escribe un método que muestre la cadena de caracteres almacenada. Desde la clase Principal, en el método main crea una cláusula try-catch para probar la nueva excepción.