Funciones

Antes de empezar, aprendan que es buena práctica evitar acentos y la letra ñ en el código lo más posible. (Por molesto que sea leer algo con mala ortografía).

Jerarquía de operaciones

En Java podemos hacer fácilmente las operaciones como sumar, restar, multiplicar y dividir. Después veremos cómo hacer cosas más complicadas como raíces, potencias y logaritmos, pero me interesa que sepan desde ahora que existe una **jerarquía de operaciones**, un orden en el que la computadora realiza las operaciones. En español, la sigla es PEMDSR: paréntesis, exponentes, multiplicar y dividir (de izquierda a derécha) y finalmente sumar y restar (de izquierda a derecha). Por ejemplo $5 \times 2 + 3 = 13$ pero $5 \times (2 + 3) = 25$.

Estructura de una función

Hasta ahora sólo han hecho códigos cortos que siempre escribían dentro de un archivo misterioso como el de abajo

```
public class NombreDeLaClase{

  public static void main(String[] args){
    //Y aquí escribían código
  }
}
```

Lo prometido es deuda, y en el siguiente documento van a terminar de entener public y class; pero ahora vamos analizar el concepto de **función**, de las cuales `main(String[] args)` es la primera que conocen. Mi responsabilidad moral como matemático es aclarales que lo que entendemos como función en programación de ninguna manera se parece a la definición formal de función en matemáticas, pero intuitivamente sí son parecidas: pueden pensar en una función como algo que recibe **parámetros** y devuelve un único resultado calculado de alguna manera con esos parámetros.

Abajo les pongo ejemplo de dos funciones sencillas para empezar a analizar la sintaxis de Java, una de las cuales hicieron ustedes en clase:

```
public static double max(double x, double y){
  return x > y ? x : y;
}

public static int nesimoFibonacci(int n){
  int a[] = new int[n];
```

```
  a[0] = a[1] = 1;

  for(int i = 2; i < n; i++){
    a[i] = a[i-1] + a[i-2];
  }

  return a[n-1];
}
```

Analicemos linea por línea qué está sucediendo. Como pueden ver, cada función tiene un **tipo**; el primero es de tipo `double` y el segundo de tipo `int`. Usualmente una función regresa un valor calculado, y el tipo de la función es el tipo del valor que queremos regresar. Regresar el valor es justamente lo que hace el comando `return`.

Lo siguiente que pueden notar es que la función tiene un **nombre** y que, en los paréntesis que le siguen al nombre, especificamos los **parámetros** que recibe. La lista de parámetros aclara cuál es el tipo de variable y después su nombre. Ojo, este es el nombre que la variable tiene *dentro de la función*, cosa que aclararemos en la siguiente sección.

El algoritmo de Fibonacci ya lo conocen, y lo único que hice fue ponerlo dentro de una función llamada `nesimoFibonacci` que recibe un número entero `n`. En el que hicimos en clase, n estaba fijo en 10, pero ahora pueden cambiarlo y pasarlo a la función como parámetro. De nuevo, en la siguiente sección veremos cómo hacerlo.

La función `max`, con esa sintaxis complicada, hace lo que se imaginan: devuelve el máximo entre dos números. Lo que está después del `return` se llama **operador ternario** y tiene la estructura "condicion ? valorVerdadero : valorFalso". Esto debeen leerlo como "Si se cumple 'condición' el valor es valorVerdadero; si no, el valor es valorFalso". Por ejemplo, el siguiente código imprime si un número entero `n` es par o impar

```
String esPar = x % 2 == 0 ? "par" : "impar";
System.out.println(esPar);
```

Vean cómo usamos el operador ternario para asignarle valor a una variable de tipo `String` llamada `esPar`.

Sólo para seguir practicando, supongamos que ahora queremos escribir el código de arriba en una función que reciba el número `x`. ¿Cuál sería el tipo de la función? Realmente la función no hace más que imprimir, así que no hay una variable que regrese. A estas funciones las llamamos de tipo **void**, que significa "no regresa nada". La función entonces se vería así:

```
public static void muestraParidad(int x){
  String esPar = x % 2 == 0 ? "par" : "impar";
  System.out.println(esPar);
```

```
  return;
}
```

Noten que aunque no regresa nada, lo ponemos para señalar el final de la función. Esto no es necesario, pero es un buen hábito que sobre todo les será útil cuando veamos recursión.

¿Pero cómo usamos las funciones que podamos programar? Como les dije en clase, la función que la computadora ejecuta se llama `main`. Cuando inicializan una archivo Class de Java como hemos hecho (y que les voy a explicar la siguiente vez) tiene un método `public static void main(String[] args)`. En este punto ya deberían saber que `args` es un arreglo de variables `String` que pasan como parámetros a una función llamada `main` que no regresa nada. Después volvemos a public y a static. Si ustedes extienden la clase Fibonacci que hicimos en clase para que se vea así

```
public class Fibonacci{
  public static int nesimoFibonacci(int n){
    int a[] = new int[n];
    a[0] = a[1] = 1;

    for(int i = 2; i < n; i++){
      a[i] = a[i-1] + a[i-2];
    }

    return a[n-1];
  }

  public static void main(String[] args){
    int m = 5;
    int num = nesimoFibonacci(5);
    System.out.println(num);
  }
}
```

están dividiendo el trabajo en dos: la función `nesimoFibonacci` calcula el n-ésimo número de Fibonacci y les permite especificar `n`. Cuando ustedes corren el programa, se guarda el valor `m = 5` y ese pasa como parámetro a la función que calcula. Aquí quiero que noten dos cosas:

1. Cuando pasan argumentos a funciones *no* necesitan especificar el tipo. Esto es, cuando la programamos arriba, escribimos `nesimoFibonacci(int n)`, pero cuando la **llamamos** (o sea, le pedimos a la computadora que la ejecute) no es necesario especificar otra vez el tipo de la variable.
2. La variable no se tiene que llamar igual cuando programamos la función que cuando la llamamos. Por ejemplo, aquí la programamos usando `n` y el número que enviamos se llama `n`.

La segunda observación nos permite entrar al siguiente tema de manera natural.

Alcance y paso de las variables

En el último pedazo de código hicimos algo curioso: declaramos una variable m en el main, y luego la pasamos a una función donde se llama n. ¿Cómo funciona esto?

Cuando llamamos a una función y le pasamos argumentos, Java **pasa por valor**. Como este nombre sugiere, cuando llamamos a `nesimoFibonacci(m)`, lo que estamos pasando no es a m como *objeto en la memoria* sino únicamente su *valor*. Es complicado de entender, pero tal vez la analogía les ayude. Si Fátima le pide a Ángel el código de algún ejercicio, Ángel puede copiarlo y mandárselo por Whatsapp. Cuando Fátima lo recibe, lo copia en Eclipse y le hace los cambios que le parezcan. Ángel sigue teniendo su copia del código, y los cambios que Fátima haga en su computadora no le van a afectar a menos que Fátima se los mande de regreso. Esta es la idea de pasar por valor.

Ahora bien, ¿qué pasaría si Ángel le contestaba con un link a su Google Drive dándole privilegios de editar? Cuando Fátima haga sus cambios, estría modificando la *única* copia que existe de ese código, y por lo tanto afecta tanto a ella como a Ángel para cuando lo quieran leer. Esta es la idea de **pasar por referencia**.

Cuando llamas a un método, Java le pasa las variables por valor. En nuestro capítulo de programación orientada a objetos les propongo un ejercicio que explora cómo pueden fallar ideas sencillas si no se es cuidadoso con cuándo se pasa por valor y cuándo por referencia.

Vamos a retomar el ejemplo desde otro ángulo. Cuando la función `main` le pasa (aunque sea por valor) un argumento a la función `nesimoFibonacci`, parece que estamos bajando un nivel de algún tipo, como entrando a una función pequeña adentro de otra función grande (después de POO tendremos toda una sección sobre estos procesos recursivos). Las variables que se inicializan dentro de la función pequeña reciben el nombre de **locales**, pues sólo existen dentro del código de esa función, y por lo tanto sólo mientras se ejecuta. Ustedes ya aprovecharon este fenómeno sin darse cuenta. Recordemos que la sintaxis de un ciclo `for`es

```
for(int i = a; i < n; i++){
  //hacer algo
}
```

El poner `int` en la condición de inicialización del ciclo, estamos declarando una variable local que desaparece cuando finaliza el ciclo. Para entender mejor este fenómeno, copia y pega el siguiente código dentro del `main` de alguna clase (o mejor, hazlo una función void y llámala desde `main`).

```java
int i = 0;
while(i<5){
  System.out.println(i);
  i++;
}
System.out.print("Ya no entró al ciclo y al final i vale ");
System.out.print(i);

for(int j=0; j<5; j++){
  System.out.println(j);
}
System.out.print("Ya no entró al ciclo y al final j vale");
System.out.print(j);
```

¿Ven por qué da un error? Como declaramos `i` antes de iniciar el `while`, sigue existiendo cuando el `while` termina. `j`, en cambio, la declaramos dentro del `for` y por lo tanto existe solamente mientras el `for` dura, y después es eliminada de la memoria.

Ejercicios

1. Crea un archivo Java Class llamado EjerciciosArreglos y añade dos funciones:

- `swap(int[] a, int n, int i, int j)` recibe un arreglo `a` de tamaño `n` y cambia en él la posición del elemento `i` con el `j`. Ojo, sé cuidadoso con los casos que puedan ser problemáticos, como si `i` o `j` no son válidos.
- `reverse(int[] a, int n)` recibe un arreglo `a` de tamaño `n` y lo voltea. Es decir, el primer elemento es ahora el último, el segundo el penúltimo y así sucesivamente. Pista: fíjate en si n es par o impar y usa la función `swap`.

2. Pasa las funciones que programaste en la parte dos de los ejercicos de la introducción a esta clase nueva, todas recibiendo parámetros. Adáptalas para usar `swap`.
3. Crea un `main` en la misma clase EjerciciosArreglos para probar tus funciones. Recuerda que el nombre completo de `main` es public static void main(String[] args).
4. Supón que se ejecutan los siguientes bloques de código. ¿Qué imprime la consola en cada caso? Intenta hacerlo en tu mente.

```java
public class Ejercicio4{

  public static double f(double x, double y){
    return 5*x + y;
```

```
  }

  public static double g(double x, double y){
    x = 5*x;
    return x+y;
  }

  public static void main(String[] args){
    double x=1, y=0, area;

    //Caso 1
    area = f(x, y);
    System.out.println(area);

    //Caso 2
    area = f(y, x);
    System.out.println(area);

    //Caso 3
    area = g(y, x);
    System.out.println("x = " + x);
    System.out.println("y = " + y);
    System.out.println("area = " + area);

  }
}
```