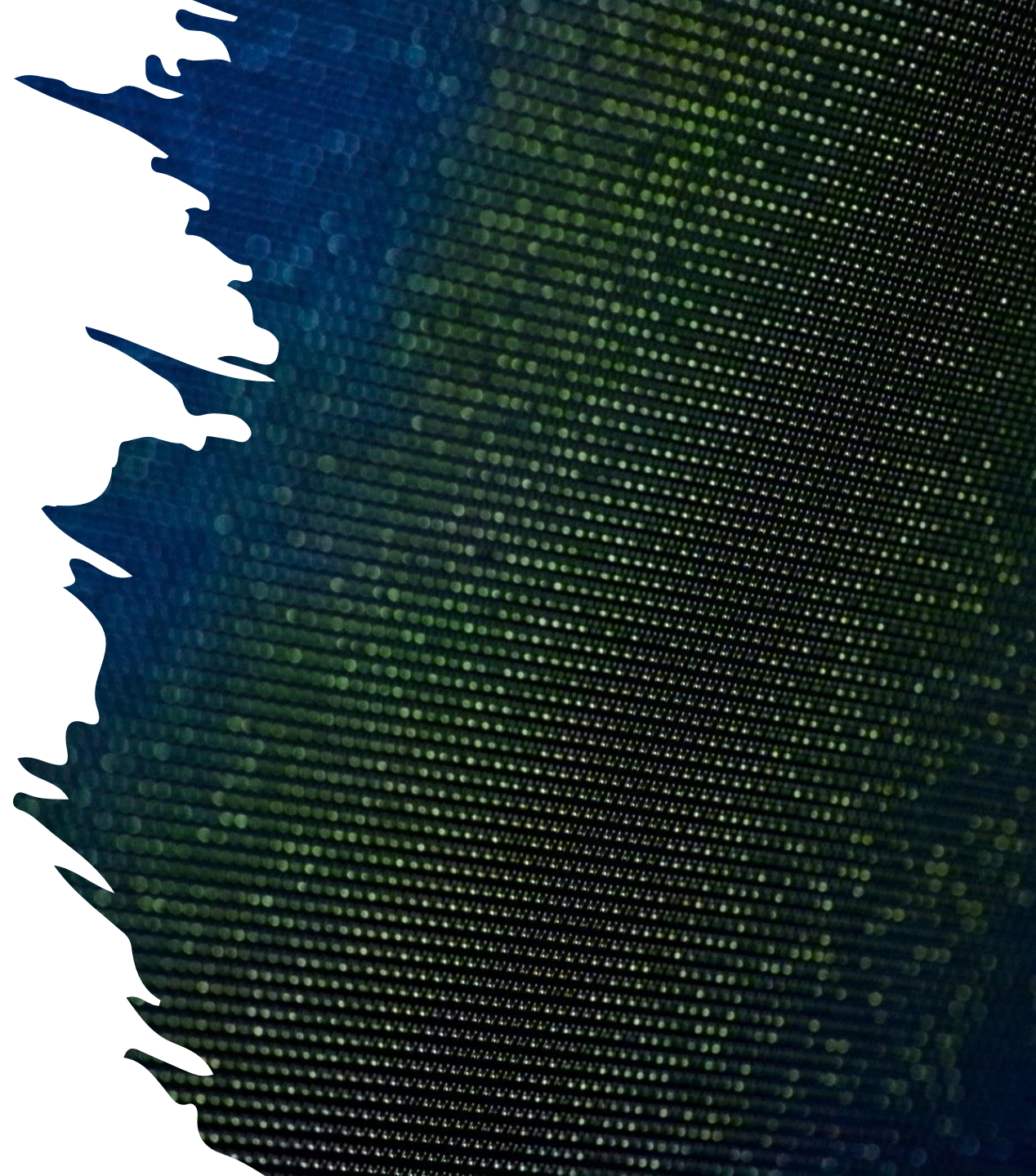


# Estructuras de Datos

M.Sc Steven Santillan Padilla.



# Definición

# Recursión

---

- Un proceso recursivo se llama a sí mismo continuamente

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

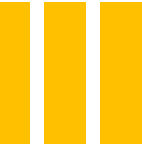
¿Cuál es la salida del código anterior?

```
Hello  
Hello  
Hello  
Hello  
Hello  
Hello  
Hello  
Hello  
Hello  
Hello  
Hello  
Hello  
Hello
```

...

```
java.lang.StackOverflowError
```

# Resultado:



# Solución

---

- Se necesita una condición de salida

```
public class RecursionExample {  
    static int count = 0;  
    static void recursiveMethod() {  
        count++;  
        if(count<=5){  
            System.out.println("Hello " + count);  
            recursiveMethod();  
        }  
    }  
    public static void main(String[] args) {  
        recursiveMethod();  
    }  
}
```

# Resultado

---

- ¿Cuál es la salida del código anterior?

```
Hello 1  
Hello 2  
Hello 3  
Hello 4  
Hello 5
```

# OK, pero ¿por qué?

- Hay problemas que, por naturaleza, son recursivos.
- Por ejemplo, el **factorial** de un número:

$$\text{factorial}(n) = \begin{cases} 1, & \text{si } n = 0 \\ n * \text{factorial}(n-1), & \text{si } n > 0 \end{cases}$$

$$\text{factorial}(3) = 3 * \text{factorial}(2)$$

$$\text{factorial}(3) = 3 * (2 * \text{factorial}(1))$$

$$\text{factorial}(3) = 3 * (2 * (1 * \text{factorial}(0)))$$

$$\text{factorial}(3) = 3 * (2 * (1 * 1))$$

$$\text{factorial}(3) = 6$$

```
public class RecursionExample {  
  
    static int factorial(int n) {  
        if (n == 0) {  
            return 1; // condición de salida o caso base  
        } else {  
            return (n * factorial(n-1)); // llamada recursiva  
        }  
    }  
  
    public static void main(String[] args) {  
        System.out.println ("Factorial of 5 is: " + factorial(5));  
    }  
}
```

Método  
factorial en  
Java



```
factorial(5)
  factorial(4)
    factorial(3)
      factorial(2)
        factorial(1)
          return 1
        return 2*1 = 2
      return 3*2 = 6
    return 4*6 = 24
  return 5*24 = 120
```



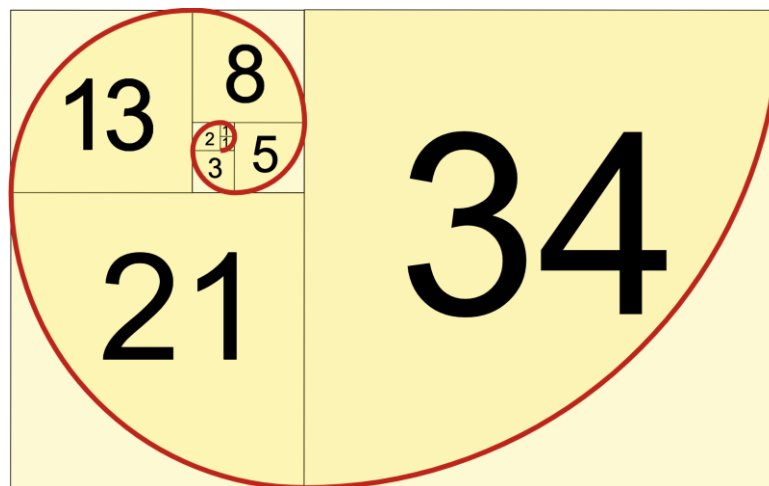
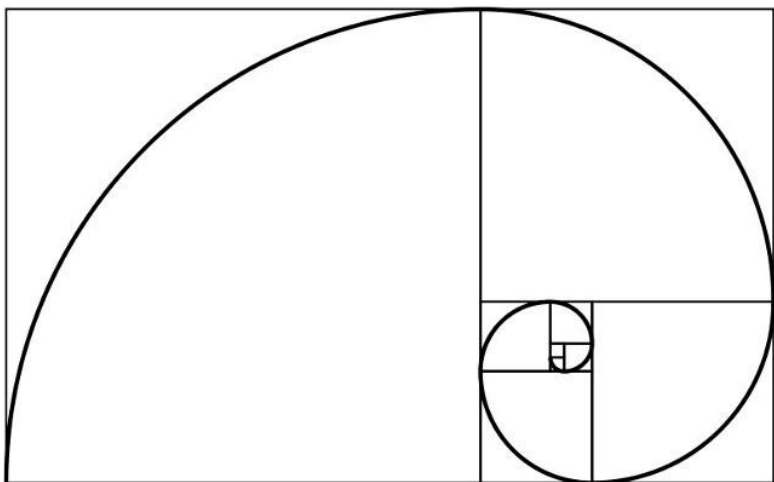
Salida

**Factorial of 5 is: 120**

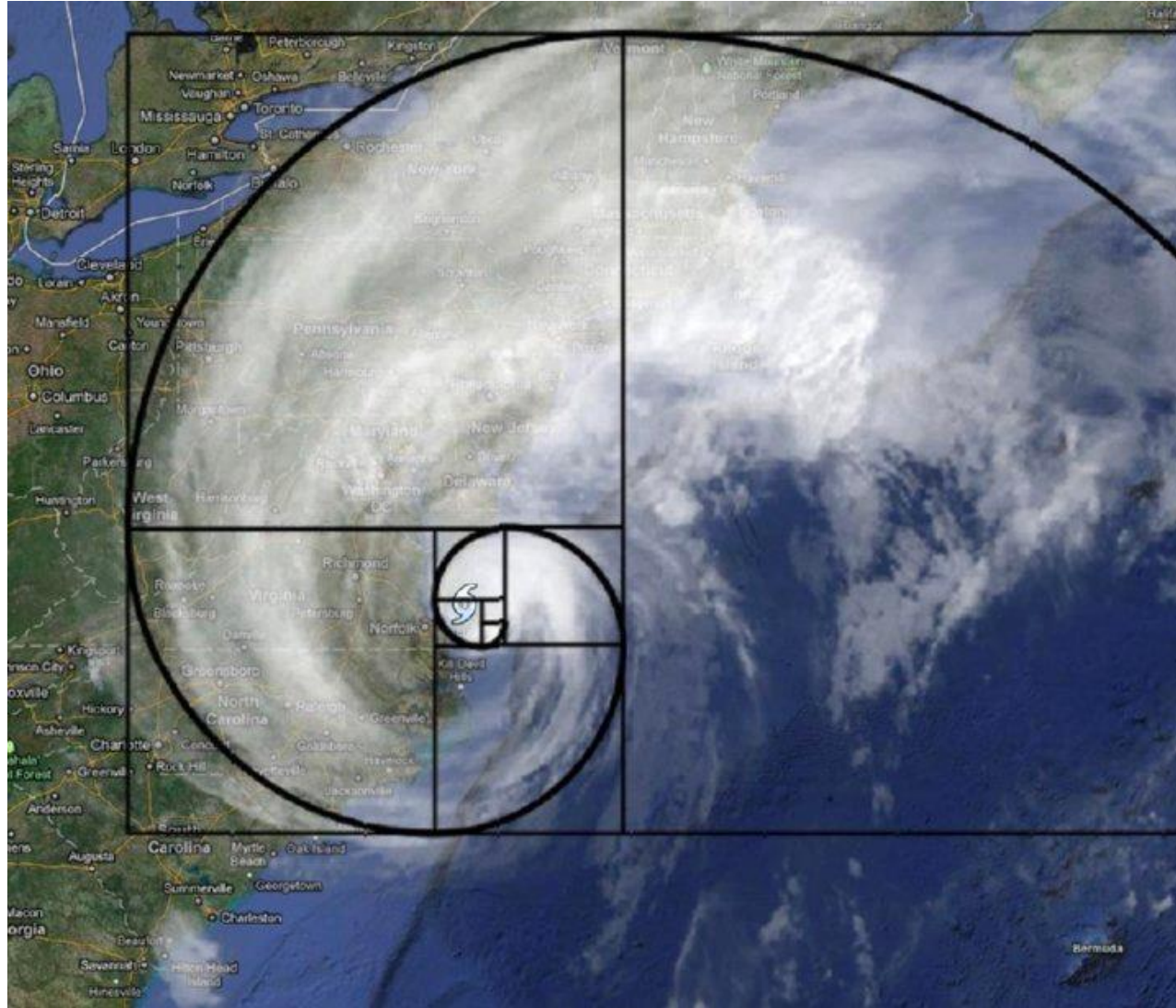


# Ejercicio: La sucesión de Fibonacci

- La serie comienza con los números 0 y 1
  - Cada nuevo término de la sucesión es la suma de los dos anteriores
  - **0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987...**
  - **¿Por qué es esta serie importante?**
-



Construcción  
de Espirales

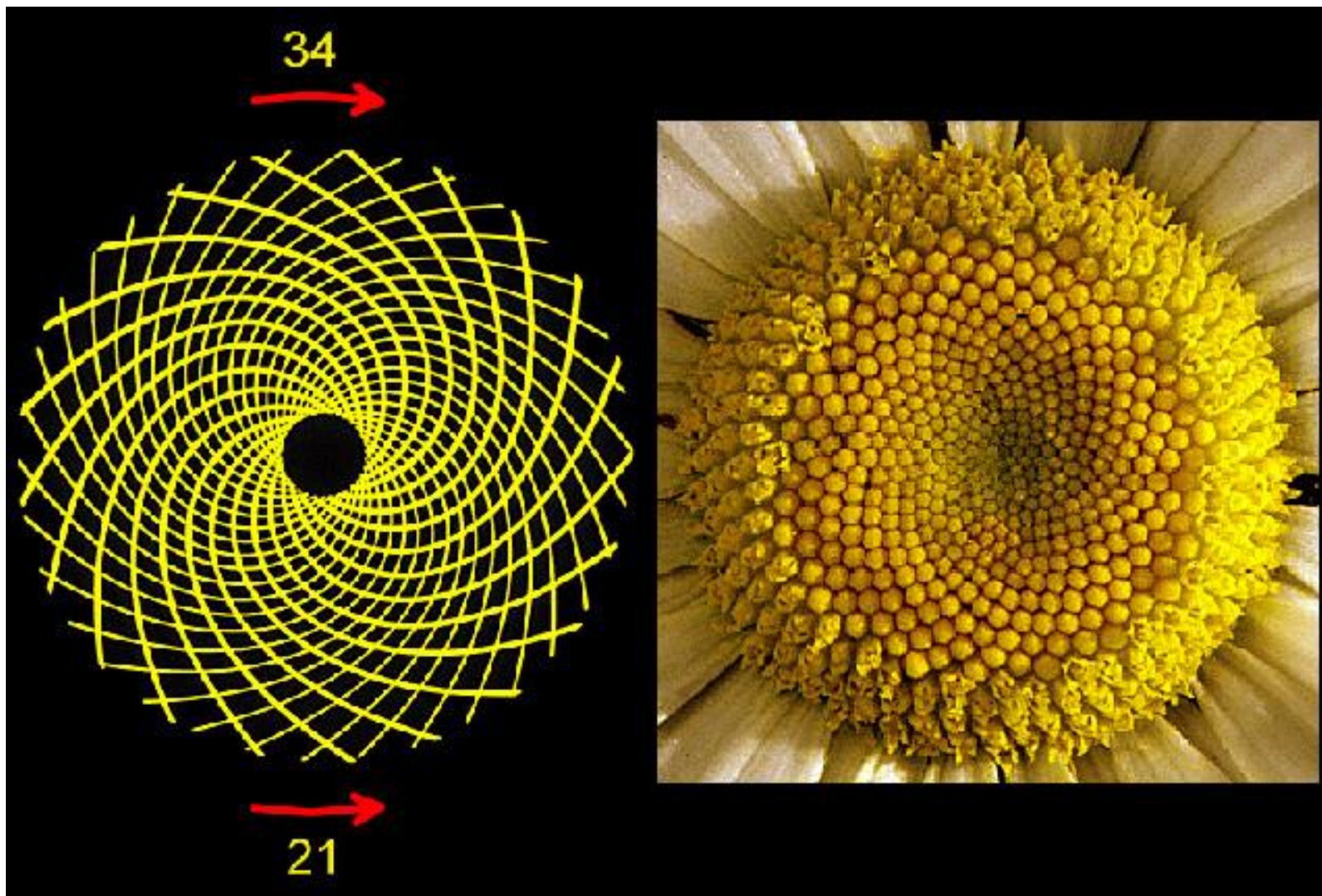


# Construcción de Espirales

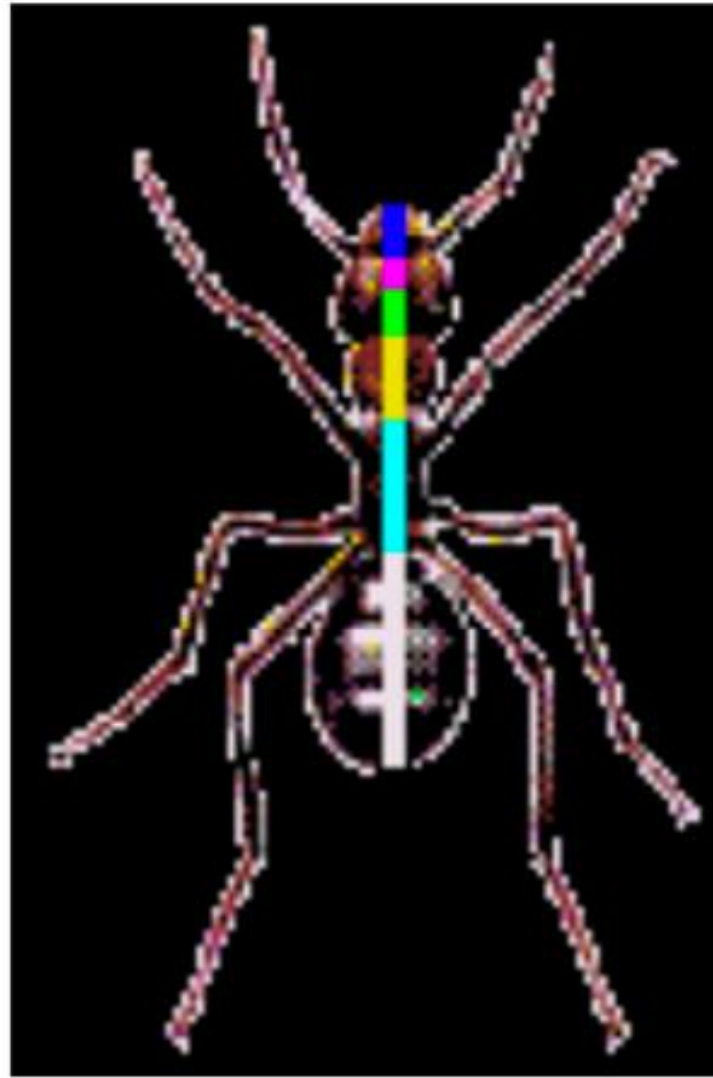


Ramas de  
los Árboles





Distribución  
de semillas



Proporcion  
es en  
cuerpos

Número de Mes	Explicación de la genealogía	Parejas de conejos
Comienzo del mes 1	Nace una pareja de conejos (pareja A).	1 pareja en total.
Fin del mes 1	La pareja A tiene un mes de edad. Se cruza la pareja A.	$1+0=1$ pareja en total.
Fin del mes 2	La pareja A da a luz a la pareja B. Se vuelve a cruzar la pareja A.	$1+1=2$ parejas en total.
Fin del mes 3	La pareja A da a luz a la pareja C. La pareja B cumple 1 mes. Se cruzan las parejas A y B.	$2+1=3$ parejas en total.
Fin del mes 4	Las parejas A y B dan a luz a D y E. La pareja C cumple 1 mes. Se cruzan las parejas A, B y C.	$3+2=5$ parejas en total.
Fin del mes 5	A, B y C dan a luz a F, G y H. D y E cumplen un mes. Se cruzan A, B, C, D y E.	$5+3=8$ parejas en total.
Fin del mes 6	A, B, C, D y E dan a luz a I, J, K, L y M. F, G y H cumplen un mes. Se cruzan A, B, C, D, E, F, G y H.	$8+5=13$ parejas en total.
...	...	...
...	...	...

# Ciclos reproductivos





# De regreso a la programación...

La serie comienza con los números 0 y 1

Cada nuevo término de la sucesión es la suma de los dos anteriores

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14
fibonacci (n)	0	1	1	2	3	5	8	13	21	34	55	89	144	233

Escriba un método recursivo que calcule el n-ésimo término de la sucesión de  
Fibonacci

Ejemplo: fibonacci (9) = 21

```
public class Main {  
  
    public static int fibonacci (int n) {  
        if (n == 1) {  
            return 0;  
        } else if (n == 2) {  
            return 1;  
        } else {  
            return fibonacci(n-1) + fibonacci (n-2);  
        }  
    }  
  
    public static void main(String[] args) {  
        for (int i = 1; i <= 14; i++) {  
            System.out.println(i + ": " + fibonacci(i));  
        }  
    }  
}
```

Una posible  
solución

# Una posible solución

```
public class Main {  
    public static int fibonacci (int n) {  
        switch (n) {  
            case 1:  
                return 0;  
            case 2:  
                return 1;  
            default:  
                return fibonacci(n-1) + fibonacci (n-2);  
        }  
    }  
  
    public static void main(String[] args) {  
        for (int i = 1; i <= 14; i++) {  
            System.out.println(i + ": " + fibonacci(i));  
        }  
    }  
}
```

```
1: 0  
2: 1  
3: 1  
4: 2  
5: 3  
6: 5  
7: 8  
8: 13  
9: 21  
10: 34  
11: 55  
12: 89  
13: 144  
14: 233
```

# Comentarios finales

## Para recordar:

- Un proceso recursivo se llama a sí mismo continuamente
- La implementación debe contener al menos un caso base o condición de salida
- De otro modo, el proceso se ejecuta un número infinito de veces
- A lo largo del curso abordaremos más problemas recursivos
- Recursión será un tema **fundamental** en el segundo parcial

```
class GFG{
static void printFun(int test)
{
    if (test < 1)
        return;
    else
    {
        System.out.printf("%d ",test);
        printFun(test-1); // statement 2
        System.out.printf("%d ",test);
        return;
    }
}

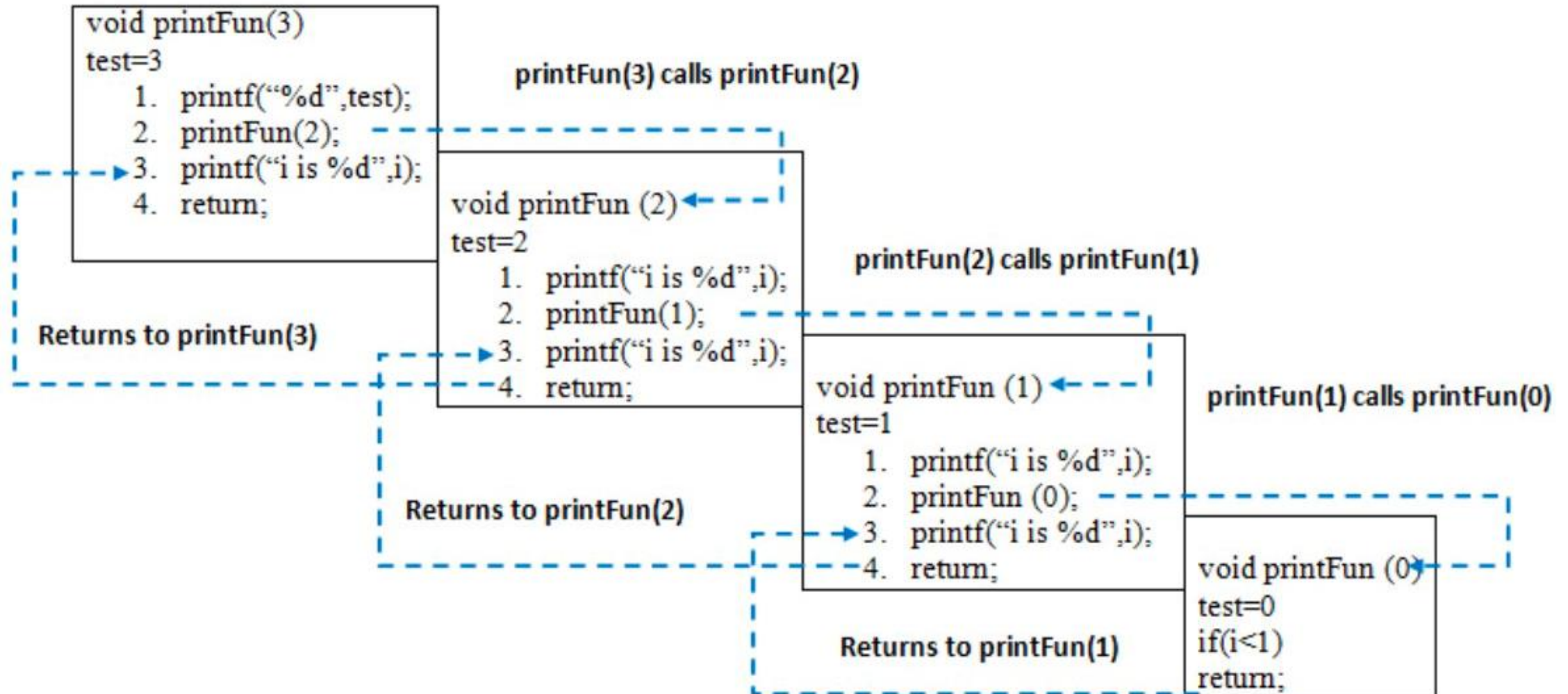
public static void main(String[] args)
{
    int test = 3;
    printFun(test);
}
}
```

## Ejercicio

Cuál es la salida del código mostrado?

# Solución

3 2 1 1 2 3



# Ejercicio

Escribir los siguientes métodos estáticos aplicando una estrategia recursiva:

**esPalindromo** Recibe un objeto de tipo String y retorna si ésta o no es un palíndromo (que se lee igual de izquierda a derecha y viceversa)

**revertir** Recibe un arreglo y lo modifica invirtiendo el orden de los elementos que éste contiene

Nota: Considere primero los casos bases y piense en qué parámetros sus métodos necesitan recibir