Módulo 8: Recursión
Capítulo 11

La recursión es definir la solución a un problema mediante la solución de una versión más pequeña del mismo problema.

Ejemplo

Definamos un algoritmo para encontrar el camino para llegar a tu casa. El algoritmo debe poder comenzar en cualquiera de las 5 casas.

Llamaremos nuestro algoritmo `getHomeFrom()`.

house4 **house3** **house2** **house1** **myHouse**

`#1: getHomeFrom(myHouse)`

i. Print “I’m home!”

house4 **house3** **house2** **house1** **myHouse**

#2: getHomeFrom(house1)

i. Move to next house

ii. getHomeFrom(myHouse)

house4 house3 house2 house1 myHouse

#3: getHomeFrom(house2)

i. Move to next house

ii. getHomeFrom(house1)

house4 house3 house2 house1 myHouse

#3: getHomeFrom(house3)

i. Move to next house

ii. getHomeFrom(house2)

house4 house3 house2 house1 myHouse

#4: getHomeFrom(house4)

i. Move to next house

ii. getHomeFrom(house3)

house4 house3 house2 house1 myHouse

Recursión

Un algoritmo recursivo SIEMPRE debe tener por lo menos dos partes:

1. Un caso base.
2. Un caso recursivo.

Modelemos el ejemplo anterior utilizando esta lógica:

`Clase GoHome`

```java
public class GoHome {
  public static final int MY_HOUSE = 0;

  public static void getHomeFrom(int houseNumber) {
    //Base case
    if (houseNumber == MY_HOUSE) {
      System.out.println("I'm home!");
    } else {
      //Recursive case
      System.out.println("I'm at house" + houseNumber);
      //move forward to next house
      int currentHouse = houseNumber - 1;
      GoHome.getHomeFrom(currentHouse);
    }
  }

  public static void main(String[] args) {
    int house3 = 3;
    GoHome.getHomeFrom(house3);
  }
}
```

Ejemplo:

```
public static void main(String[] args) {
    int house3 = 3;
    GoHome.getHomeFrom(house3);
 }
```

```java
public static void getHomeFrom(int houseNumber) {
  if (houseNumber == MY_HOUSE) {
    System.out.println("I'm home!");
  } else {
    System.out.println("I'm at house" + houseNumber);
    int currentHouse = houseNumber - 1;
    GoHome.getHomeFrom(currentHouse);
  }
}
```

```java
public static void getHomeFrom(int houseNumber) {
  if (houseNumber == MY_HOUSE) {
    System.out.println("I'm home!");
  } else {
    System.out.println("I'm at house" + houseNumber);
    int currentHouse = houseNumber - 1;
    GoHome.getHomeFrom(currentHouse);
  }
}
```

```java
public static void getHomeFrom(int houseNumber) {
  if (houseNumber == MY_HOUSE) {
    System.out.println("I'm home!");
  } else {
    System.out.println("I'm at house" + houseNumber);
    int currentHouse = houseNumber - 1;
    GoHome.getHomeFrom(currentHouse);
  }
}
```

```java
public static void getHomeFrom(int houseNumber) {
  if (houseNumber == MY_HOUSE) {
    System.out.println("I'm home!");
  } else {
    System.out.println("I'm at house" + houseNumber);
    int currentHouse = houseNumber - 1;
    GoHome.getHomeFrom(currentHouse);
  }
```

Output

```
I’m at house3
I’m at house2
I’m at house1
I’m home!
```

getHomeFrom(2)

getHomeFrom(1)

getHomeFrom(0)

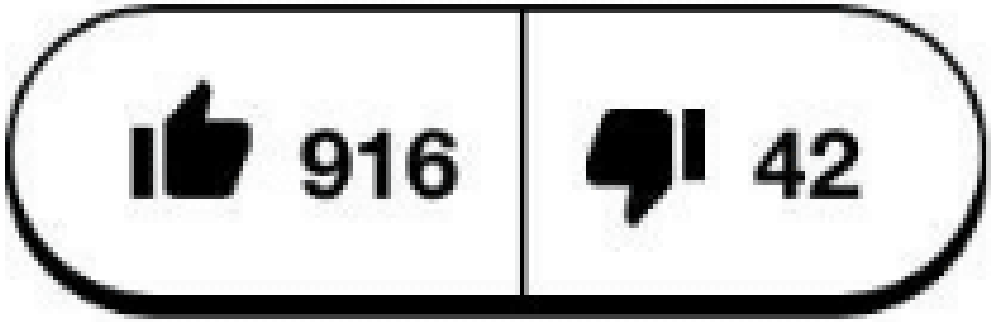
`GetHomeFrom(3)`

- `getHomeFrom(3)` → Imprimir en la consola "I'm at house 3" e invocar `getHomeFrom(2)`.
- `getHomeFrom(2)` → Imprimir en la consola "I'm at house 2" e invocar `getHomeFrom(1)`.
- `getHomeFrom(1)` → Imprimir en la consola "I'm at house 1" e invocar `getHomeFrom(0)`.
- `getHomeFrom(0)` → Imprimir en la consola "I'm home!"

recursion

See recursion.

by Anonymous December 05, 2002



Diseñar recursión eficiente

Un algoritmo recursivo SIEMPRE debe tener por lo menos dos partes:

1. **Un caso base.** Debe contener los escenarios que ya no pueden ser reducidos a un caso menor.
2. **Un caso recursivo.** El argumento del caso recursivo debe ser un caso reducido del argumento original.

Modelemos el ejemplo anterior utilizando esta lógica:

`Clase GoHome`

Recursión infinita

Supongamos que dejemos fuera la condición de salida:

```
public static void getHomeFrom(int houseNumber) {
  System.out.println("I'm at house" + houseNumber);
  //move forward to next house
  int currentHouse = houseNumber - 1;
  GoHome.getHomeFrom(currentHouse);
}
```

```java
public static void getHomeFrom(int houseNumber) {
  System.out.println("I'm at house" + houseNumber);
  //move forward to next house
  int currentHouse = houseNumber - 1;
  GoHome.getHomeFrom(currentHouse);
}
```

ERROR! Ciclo Infinito

I’m at house3
I’m at house2
I’m at house1
I’m at house0
I’m at house-1
I’m at house-2
…
I'm at house-8611
I'm at house-8612
I'm at house-8613

ERROR! Ciclo Infinito

```
I'm at house-8614Exception in thread "main" java.lang.StackOverflowError
        at java.io.FileOutputStream.write(Unknown Source)
        at java.io.BufferedOutputStream.flushBuffer(Unknown Source)
        at java.io.BufferedOutputStream.flush(Unknown Source)
        at java.io.PrintStream.write(Unknown Source)
        at sun.nio.cs.StreamEncoder.writeBytes(Unknown Source)
        at sun.nio.cs.StreamEncoder.implFlushBuffer(Unknown Source)
        at sun.nio.cs.StreamEncoder.flushBuffer(Unknown Source)
        at java.io.OutputStreamWriter.flushBuffer(Unknown Source)
        at java.io.PrintStream.newLine(Unknown Source)
        at java.io.PrintStream.println(Unknown Source)
        at pckg1.GoHome.goHome(GoHome.java:7)
        at pckg1.GoHome.goHome(GoHome.java:9)
```

Caso de Estudio

Diseña un método que reciba un número entero positivo y retorne la cantidad de ceros que contenga.

Ejemplos: `getNumberOfZeros(10320)` → `2`

```
public static int getNumberOfZeros(int n) {

}
```

1. ¿Cuáles son los casos base?
2. ¿En qué casos podemos reducir aún más el problema?

Caso de Estudio

Caso base:

1. Si el número recibido es igual a 0, entonces no podemos reducir el problema. Retornamos 1.
2. Si el número recibido está entre 1 y 9, no podemos reducir el problema. Retornamos 0.

Ejemplo:

- `getNumberOfZeros(0)` → `1`
- `getNumberOfZeros(1)` → `0`
- `getNumberOfZeros(7)` → `0`

Caso de Estudio

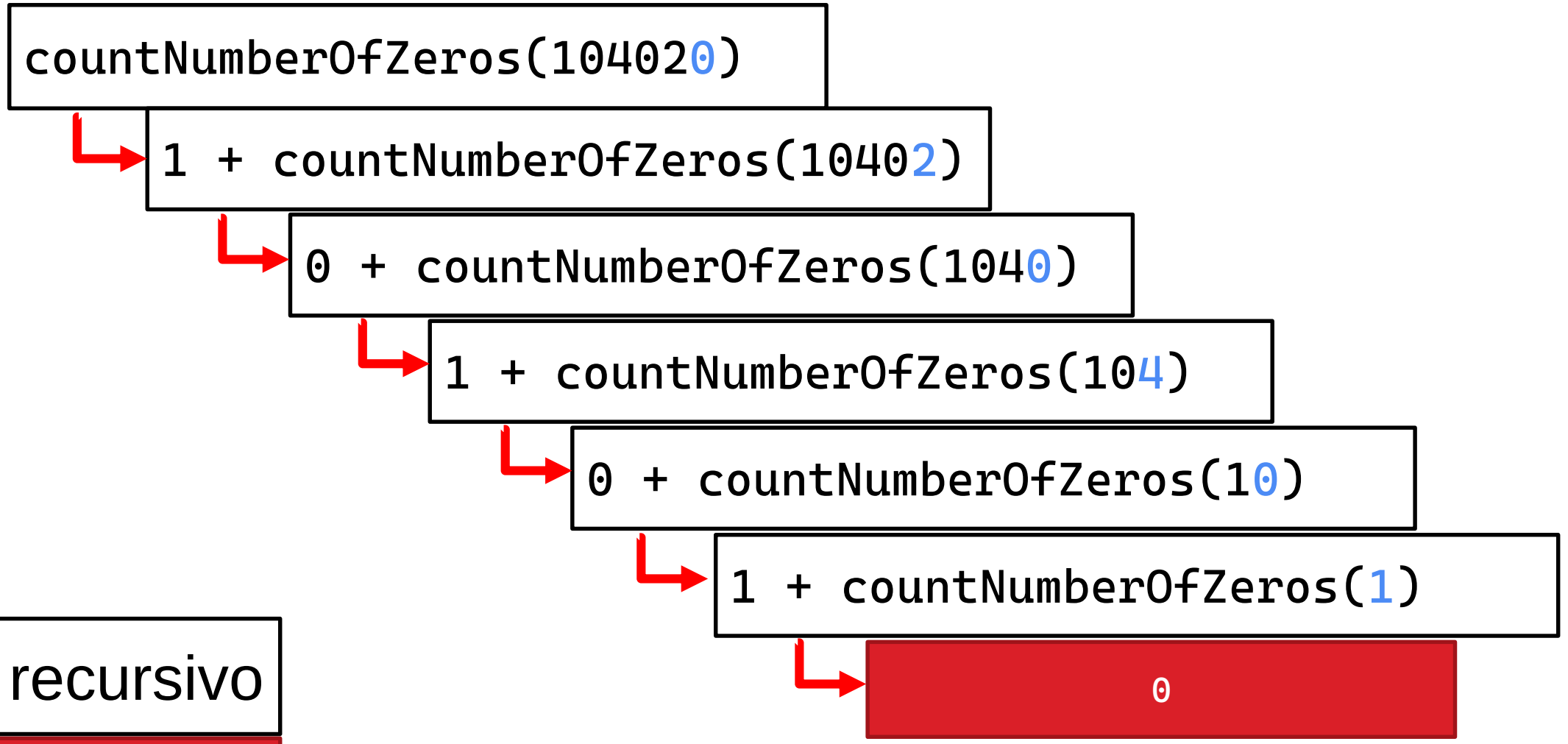
Caso recursivo:

1. Si el número recibido es mayor o igual a 10, revisaremos el dígito menos significativo y revisamos si este es igual a 0.
    1. Si es igual a 0, retornamos 1 + getNumberOfZeros(number/10).
    2. Si es diferente a 0, retornamos 0 + getNumberOfZeros(number/10).

Ejemplo:

- `getNumberOfZeros(123) → 0 + getNumberOfZeros(12)`
- `getNumberOfZeros(100) → 1 + getNumberOfZeros(10)`
- `getNumberOfZeros(607) → 0 + getNumberOfZeros(60)`

Ejemplo: Ejecución de `countNumberOfZeros(104020)`



Caso recursivo

Caso base

```java
public static int getNumberOfZeros(int number) {
  // Base case #1: 0 is exactly 1 zero
  if (number == 0)
    return 1;
  // Base case #2: Numbers between 1 and 9 contain exactly 0 zeros
  if (number <= 9)
    return 0;
  //Calculate newNumber by dividing number by 10, to remove least significative digit
  int newNumber = number / 10;
  //Recursive case #1: Least significative digit is 1 zero
  if (number % 10 == 0) {
    return 1 + getNumberOfZeros(newNumber);
  //Recursive case #2: Least significative digit is not a zero
  } else {
    return 0 + getNumberOfZeros(newNumber);
  }
}
```