Algoritmos y Estructuras de Datos
Tema 2: Diseño de Algoritmos

Contenidos

Recursividad

- Una **definición** de un **concepto** es ***recursiva***, si es una definición que **emplea el propio concepto** en la **definición**
- Por ejemplo: *Máximo Común Divisor*

$$MCD(m,n) = \begin{cases} n, \text{ si } RESTO(m,n) = 0 \\ MCD(n, RESTO(m,n)) \end{cases}$$

- Un **método** Java es **recursivo** si su cuerpo incluye **llamadas a si mismo**
  - La recursión es **indirecta** cuando un método *m* incluye **llamadas a métodos** que directamente o **indirectamente** incluyen llamadas a *m*

Ejemplos de métodos recursivos

- Una definición matemática de factorial es:

$$\text{factorial(n) es} \begin{cases} 1, \text{ si n <= 1} \\ \text{n * factorial(n-1), en otros casos} \end{cases}$$

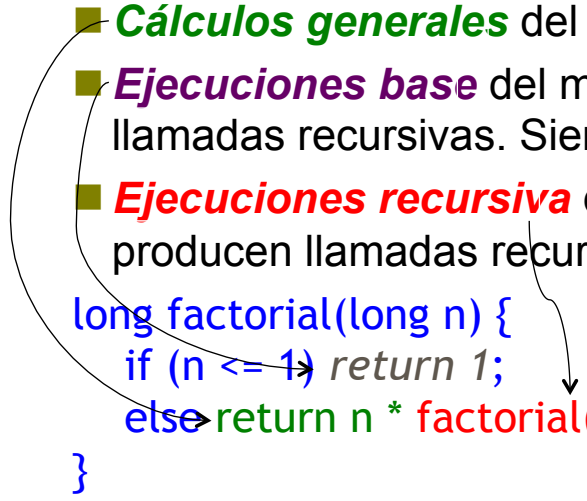
- Una implementación de esa definición en Java es:

```
long factorial(long n) {
    if (n <= 1) return 1;
    else return n * factorial(n - 1);
}
```

- En Java cualquier método puede ser recursivo
  - **No** hay que **confundir** recursividad y llamadas a métodos o constructores **sobrecargados**

Estructura de la recursión simple

- La **llamada recursiva** debe estar **en** el cuerpo de algún tipo de **condición**. Un método recursivo sencillo incluye:
  - ***Cálculos generales*** del método.
  - ***Ejecuciones base*** del método: esas ejecuciones no hay llamadas recursivas. Siempre debe haber al menos una.
  - ***Ejecuciones recursiva*** del método: ejecuciones que producen llamadas recursivas



```
long factorial(long n) {
   if (n <= 1) return 1;
   else return n * factorial(n - 1);
}
```

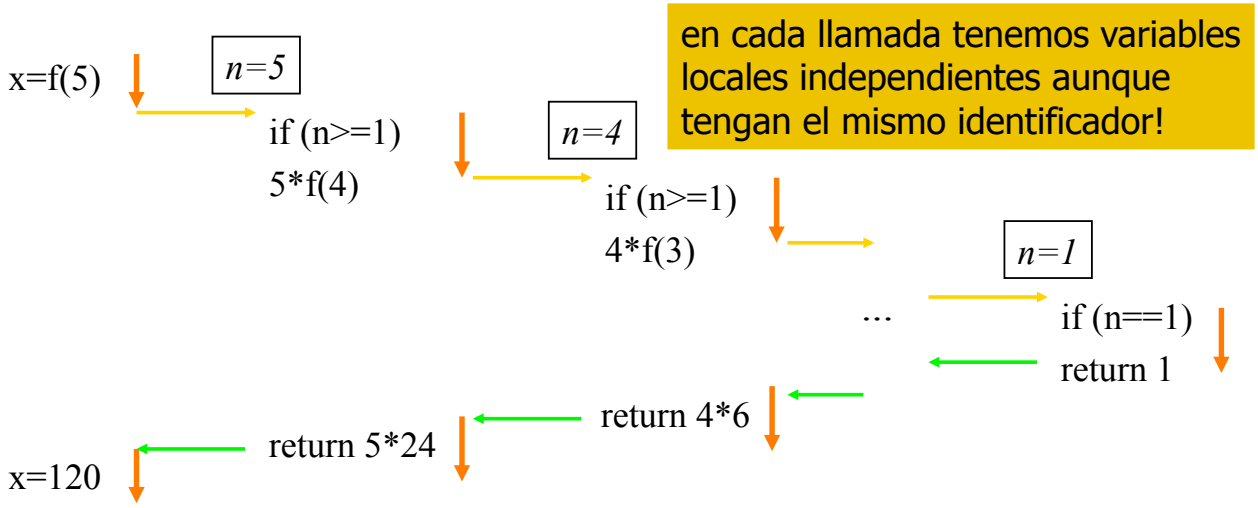

Variables en ejecución

- Variables a las que accede un método recursivo:
  - ***Variables locales y parámetros***: cada llamada recursiva tiene su copia de estas variables, y no puede acceder a los valores de las llamadas anteriores
    - Al final de la ejecución se **destruyen** esas variables
    - Mientras está pendiente de resolverse una llamada recursiva, sus llamadas anteriores también están **pendientes** de terminar
    - Recursión infinita. No hay memoria para almacenar todo:

```
int recursionInfinita() {
   recursionInfinita();}
```

  - *Los campos* están asociados a los objetos y clases y todas las llamadas actúan sobre esos **mismos** campos y sus valores

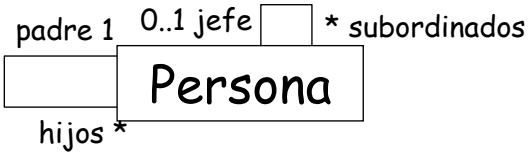
Ámbito dinámico (recursivo)



en cada llamada tenemos variables locales independientes aunque tengan el mismo identificador!

Estructuras recursivas

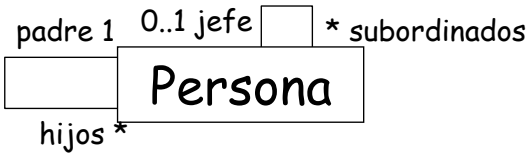
- Muchas estructuras de datos incluyen **referencias** a objetos de su **misma clase**



- Algunos conjuntos de los objetos crean **estructuras recursivas**. Ejemplos de estructuras generales: estructuras jerárquicas y de listas
- Algunas operaciones que tratan esas referencias se pueden hacer con recursividad
  - Por ejemplo: búsquedas en ramas de árboles

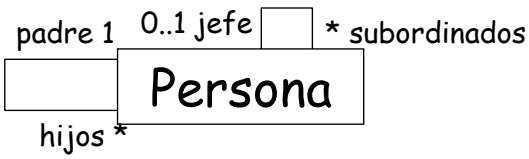
Estructuras recursivas

```
class Persona {
  String nombre;
  Persona padre;
  Persona[] hijos;
  Persona jefe;
  Persona[] subordinados;
  Persona jefeEmpresa() {


  }
  String descendientes() {




  }
}
```



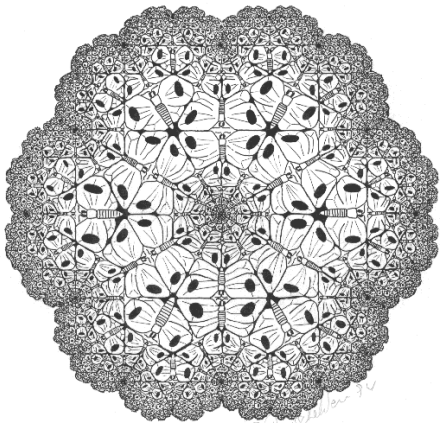
Estructuras recursivas

```
class Persona {
  String nombre;
  Persona padre;
  Persona[] hijos;
  Persona jefe;
  Persona[] subordinados;
  Persona jefeEmpresa() {
   if (jefe == null) return this;
    else return jefe.jefeEmpresa();
  }
  String descendientes() {
   String nombres="";
   if (hijos.length == 0) return nombres;
   for (int i=0; i < hijos.length; i++)
     nombres=hijo[i].nombre+ " "+hijos[i].descendientes();
   return nombres;
  }
}
```



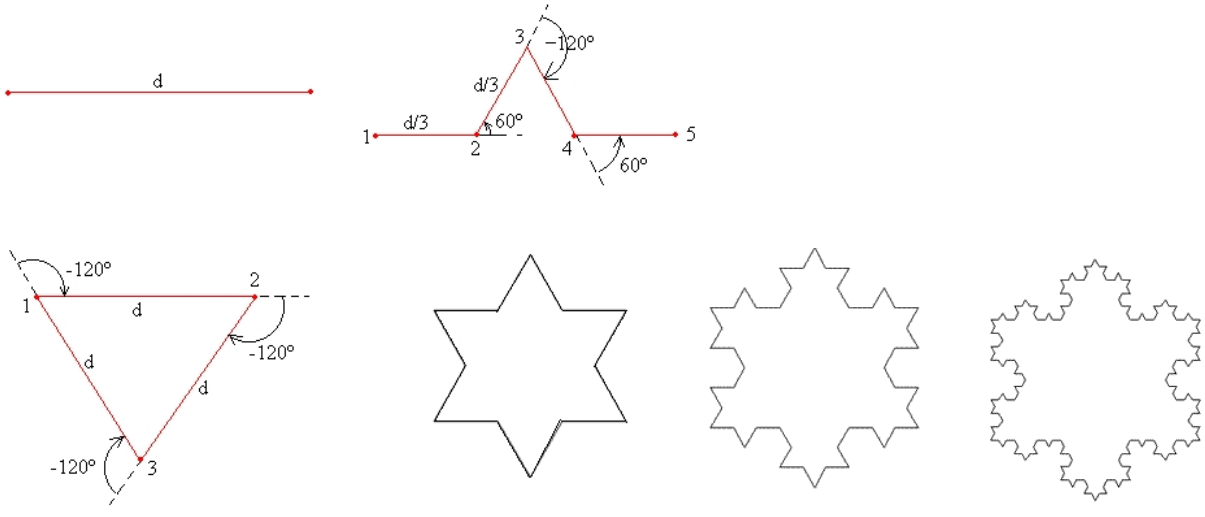
Curvas recursivas: Fractales

- Un **fractal** es un objeto geométrico cuya **estructura básica**, fragmentada o irregular, se **repite** a diferentes escalas
- El método traza una forma geométrica simple
  - En la **siguiente escala** partes de esa figura pueden representarse con la forma geométrica
    - En la **siguiente escala** partes de esa figura pueden representarse con la forma geométrica
      - ...



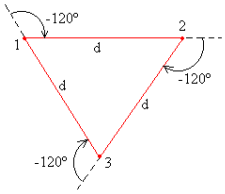
Fractales: curva de Koch

- En el copo de nieve de Koch cada segmento es sustituido por cuatro segmentos, cada uno de ellos de un tercio de la longitud del anterior



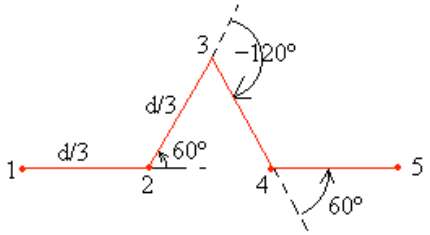
Fractales: curva de Koch

```
public class KochSnowflake extends GraphicsProgram {
   private static final double FRACCION = 0.75;
   private static final int ORDEN = 5;
   private GPen lapiz;

   public void run() {
      double ancho = 500; // getWidth();
      double alto = 500; // getHeight();
      lapiz = new GPen();
      add(lapiz, ancho / 2, alto / 2);
      dibujaFractalKoch(FRACCION * Math.min(ancho, alto), ORDEN);
   }
   private void dibujaFractalKoch(double extremo, int orden) {
      lapiz.move(-extremo / 2, -extremo / (2 * Math.sqrt(3)));
      dibujaLineaFractal(extremo, 0, orden);
      dibujaLineaFractal(extremo, -120, orden);
      dibujaLineaFractal(extremo, +120, orden);
   }
```



Fractales: curva de Koch

```
 private void dibujaLineaFractal(double r, int theta, int orden) {
      if (orden == 0) {
         lapiz.drawPolarLine(r, theta);
      } else {
         dibujaLineaFractal(r / 3, theta, orden - 1);
         dibujaLineaFractal(r / 3, theta + 60, orden - 1);
         dibujaLineaFractal(r / 3, theta - 60, orden - 1);
         dibujaLineaFractal(r / 3, theta, orden - 1);
      }
   }
}
```



Llamadas recursivas

- Cada llamada tiene asociado un **retorno**
  - Una **excepción** aborta solo la **llamada**
    - Las excepciones pueden seguir el anidamiento
- La **primera llamada en terminar** será la **última** llamada hecha, y la **última** en terminar será la **primera**
- La recursividad debe terminar en algún momento y el **número de llamadas anidadas** debe ser **limitado**
  - Si no hay memoria suficiente -> excepción
- La(s) llamada(s) recursiva(s) debe(n) estar en una sentencia alternativa o en el cuerpo de un bucle que no siempre se ejecuta

Recursividad vs bucle

- **Recursividad** es una forma de **iteración**
- Dos formas alternativas de resolver un problema

```
int fact(int n) throws Exception {
  if (n < 0) throw new Exception("argumento no válido");
  int f=1;
  while (n > 1) {
    f=n*f;
    n=n-1;
  }
  return f;
}
```

- En **general**, debemos elegir la que es mas **fácil entender**

Recursividad vs bucle

- Un **bucle** podemos implementarlo con sentencias condicionales (if) y **recursividad**:

```
void rellenar(int[] a, int desde) {
    for (int i=desde; i >= 0; i--)
        a[i]=i;
}

void rellenar(int[ ] a, int n) {
    if (n < 0) return;
    else {
        a[n] = n;
        rellenar(a, n - 1);
    }
}
```

Ejecución Base

Cálculos Generales

Ejecución Recursiva

Método recursivo y fachada

- Las implementaciones de los métodos recursivos suelen emplear **parámetros auxiliares** para controlar los niveles de recursividad
  - Esos parámetros crean confusión (no sabemos que poner) para quién usa el método, porque desconocemos la implementación
- Un **método fachada** es la entrada pública, un método privado recursivo implementa la operación

```
public boolean isIncluido(int x, int[ ] a) {
    return isIncluido (x, a, a.length - 1);
}
private boolean isIncluido(int x, int[ ] a, int n) {
    if (a[n] == x) return true;
    if (n < 0) return false;
    return isIncluido(x, a, n - 1);
}
```

Método fachada para saber si *x* está en *a*

Ejecuciones Base

Ejecución Recursiva