Retos de Programación Orientada a Objetos

Reto 1. Árbol de navidad

Objetivo: Pintar un árbol de navidad como el que se muestra a continuación (basado en asteriscos), programado de la forma más automatizada posible mediante la utilización de bucles.

Tipo de trabajo: Colaborativo en grupos de 4 alumnos.

Duración: 20 minutos

Datos: Se deberá automatizar para que sea capaz de pintar árboles de cualquier número de pisos, ancho y pareja de caracteres a representar (**, ##, ^^, etc.).

A modo de ejemplo, para validar el resultado del programa, se deberá programar el siguiente árbol:

Árbol de 3 pisos, con las siguientes características:

1. Altura de 5 líneas y un ancho máximo de 10 asteriscos
2. Altura de 7 líneas y un ancho máximo de 14 asteriscos
3. Altura de 9 líneas y un ancho máximo de 18 asteriscos

El ancho mínimo es de dos asteriscos.

Salida del programa:

```
        **
      ****
    ******
  ********
**********
        **
      ****
    ******
  ********
**********
  **********
**********
        **
      ****
    ******
  ********
**********
  **********
**********
  **********
**********
```

Solución

In [1]:
```
class Arbol
{
    static void crear(int ancho, int pisos, String caracteres)
    {
        ancho++;
        int anchoMax = ancho * pisos;

        for(int i=0;i<pisos;i++)
        {
            for(int size=1;size<ancho;size++)
            {
                for(int rama=0;rama<(anchoMax - size);rama++)
                    System.out.print(" ");

                for(int rama=0;rama<size;rama++)
                    System.out.print(caracteres);

                System.out.println();
            }
            ancho += 2;
        }
    }
}
```

In [2]:
```
Arbol.crear(5, 3, "**")
```

```
          **
         ****
        ******
       ********
      **********
          **
         ****
        ******
       ********
      **********
     ************
    **************
          **
         ****
        ******
       ********
      **********
     ************
    **************
   ****************
  ******************
```

Reto 2. Benchmarking cálculo del número PI

Objetivo: Realizar un becnhmarking sobre el cálculo del número PI mediante el Método de Montecarlo que se suministra con distinto número de iteraciones.

Tipo de trabajo: Colaborativo en grupos de 4 alumnos.

Duración: 15 minutos

Datos: Las iteraciones a probar serán desde 1 hasta 10^8, en incrementos de x10.

Salida del programa: Mostrar una tabla de resultados que muestre los siguientes valores:

El significado de cada columna es:

```
Valor obtenido de PI, número de iteraciones, tiempo de ejecución, error absoluto o
btenido.
```

No será necesario mostrar el encabezado anterior.

```
4.0, 1, 0ms., 858407 error
2.8, 10, 0ms., 341592 error
3.28, 100, 0ms., 138407 error
3.016, 1000, 0ms., 125592 error
3.1628, 10000, 12ms., 21207 error
3.14064, 100000, 113ms., 952 error
3.140548, 1000000, 935ms., 1044 error
3.141434, 10000000, 9971ms., 158 error
```

In [3]:

```
class App
{
    static double calculoPIMontecarlo(long iteraciones)
    {
        double x;
        double y;
        int exito = 0;
        for (int i=0;i<iteraciones;i++)
        {
            x = Math.random();
            y = Math.random();
            if ((Math.pow(x, 2) + Math.pow(y, 2)) <= 1)
                exito++;
        }
        return (double) (4*exito)/iteraciones;
    }
}
```

Solución

In [4]:

```
for (long iteraciones=1;iteraciones<Math.pow(10, 8);iteraciones*=10)
{
    long tiempo0 = System.nanoTime();
    double calculoPI = App.calculoPIMontecarlo(iteraciones);
    long tiempo1 = System.nanoTime();
    int error = (int) ((calculoPI-Math.PI)*1000*1000);
    long tiempoTotal = (tiempo1-tiempo0)/(1000*1000); //En milisegundos
    System.out.println(calculoPI + ", " + iteraciones + ", " + tiempoTotal + "ms., " +
}
```

```
4.0, 1, 0ms., 858407 error
4.0, 10, 0ms., 858407 error
3.08, 100, 0ms., 61592 error
3.036, 1000, 1ms., 105592 error
3.1412, 10000, 3ms., 392 error
3.14284, 100000, 28ms., 1247 error
3.144516, 1000000, 130ms., 2923 error
3.1419116, 10000000, 546ms., 318 error
```

Reto 3. Comportamiento de StringBuilder a Persona

Objetivo: Recientemente hemos visto como se puede trabajar con StringBuilder de la siguiente forma:

```
StringBuilder sb = new StringBuilder();
sb.append("A");
sb.append("B");
sb.append("C");
```

Y también hemos visto que permite otras formas equivalente, muy cómodas y muy utilizadas en la actualidad:

```
StringBuilder sb = new StringBuilder();
sb.append("A")
  .append("B")
  .append("C");
```

Haced los cambios pertinentes a la clase Persona para que permita asignar valores mediante sus setters de la misma forma que puede hacer StringBuilder.

Sería pasar de este formato actual y convencional:

```
Persona persona = new Persona();
persona.setNombre("Luis");
persona.setEdad(22);
persona.setDireccion("Calle del Pez, 17");
```

Al siguiente:

```
Persona persona = new Persona();
persona.setNombre("Luis")
       .setEdad(22)
       .setDireccion("Calle del Pez, 17");
```

Tipo de trabajo: Colaborativo en grupos de 4 alumnos.

Duración: 5 minutos

In [5]:

```java
public class Persona
{
    private String nombre;
    private int edad;
    private String direccion;

    public void setNombre(String nombre)
    {
        this.nombre = nombre;
    }

    public void setEdad(int edad)
    {
        this.edad = edad;
    }

    public void setDireccion(String direccion)
    {
        this.direccion = direccion;
    }

    @Override
    public String toString()
    {
        return nombre + " (" + edad + ") en " + direccion;
    }
}
```

In [6]:

```java
Persona persona = new Persona();
persona.setNombre("Luis");
persona.setEdad(22);
persona.setDireccion("Calle del Pez, 17");

System.out.println(persona);
```

```
Luis (22) en Calle del Pez, 17
```

Solución

In [7]:
```
public class Persona
{
    private String nombre;
    private int edad;
    private String direccion;

    public Persona setNombre(String nombre)
    {
        this.nombre = nombre;
        return this;
    }

    public Persona setEdad(int edad)
    {
        this.edad = edad;
        return this;
    }

    public Persona setDireccion(String direccion)
    {
        this.direccion = direccion;
        return this;
    }

    @Override
    public String toString()
    {
        return nombre + " (" + edad + ") en " + direccion;
    }
}
```

In [8]:
```
Persona persona = new Persona();
persona.setNombre("Luis")
       .setEdad(22)
       .setDireccion("Calle del Pez, 17");

System.out.println(persona);
```

```
Luis (22) en Calle del Pez, 17
```

Reto 4. Modificadores de acceso

Objetivo: Dado el siguiente programa, establecer los modificadores de acceso **más restrictivos posibles** para que siga un diseño correcto y además, funcione el programa.

Tipo de trabajo: Colaborativo en grupos de 4 alumnos.

Duración: 5 minutos

```
package dominio;

class A
{
```

```
    int a;

    A(int a){....}

    int getA(){....}

    void setA(int a){....}

    int m()
    {
        int x = this.m2() + this.m3();
    }

    int m2(){....}

    int m3(){....}

    String toString(){....}
}

package dominio;

class B extends A
{
    int b;

    B(int a, int b){....}

    int getB(){....}

    void setB(int b){....}

    int m()
    {
        int x = this.m2() + this.m4();
    }

    int m4(){....}

    String toString(){....}

}

package ui;

class App
{
    static void main(String args[])
    {
        A a1 = new A(1);
        B b1 = new B(1, 1);
        a1.m();
```

```
        b1.m();
    }
}
```

Solucíón

```
package dominio;

public class A
{

    private int a;

    public A(int a){....}

    public int getA(){....}

    public void setA(int a){....}

    public int m()
    {
        int x = this.m2() + this.m3();
    }

    protected int m2(){....}

    private int m3(){....}

    @Override
    public String toString(){....}
}

package dominio;

public class B extends A
{
    private int b;

    public B(int a, int b){....}

    public int getB(){....}

    public void setB(int b){....}

    @Override
    public int m()
    {
        int x = this.m2() + this.m4();
    }
```

```
    private int m4(){....}

    @Override
    public String toString(){....}

}
```

```
package ui;

import dominio.A;
import dominio.B;

public class App
{
    public static void main(String args[])
    {
        A a1 = new A(1);
        B b1 = new B(1, 1);
        a1.m();
        b1.m();
    }
}
```

In [ ]: