Fundamentos de la programación orientada a objetos

**Apellido y nombre: Copa, Farid Joaquín**

**LU: TUV 000714**

**Año: 2024**

**Ejercicio 1**

**Fase de Análisis**

**Definición del problema:** Obtener el resultado de una expresión sabiendo que para A=2 y B=5.

**Análisis**

**Datos de entrada:**

A, B: Entero

**Datos de salida:**

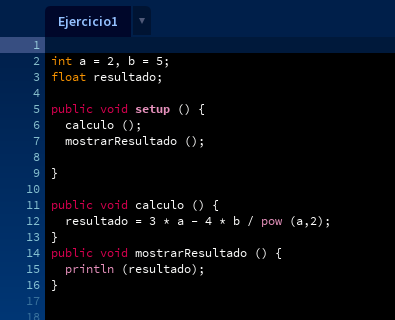
Resultado: Entero

**Proceso**

resultado = 3 \* A – 4 \* B / A ^ 2

**Fase de diseño**

|  |
| --- |
| Entidad que resuelve: Expresiones aritméticas |
| Variables:  A, B : Enteros // Almacenan 2 valores que darán el resultado  Resultado: Entero // Almacena el resultado de la expresión aritmética |
| Nombre del algoritmo: expresión\_aritmética  leer A  Leer B  resultado 🡨 3\*A-4\*B/A^2 // realiza los cálculos y asigna el resultado  mostrar resultado |



**Ejercicio 2**

**Fase de Análisis**

**Definición del problema:** Obtener el resultado de una expresión dada

**Análisis**

**Datos de entrada**

expresion: Real

**Datos de salida**

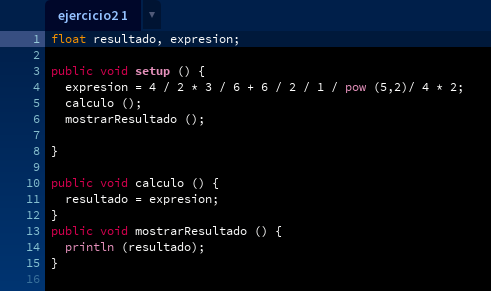
Resultado: Real

**Proceso**

resultado = 4 / 2 \* 3 / 6 + 6 / 2 / 1 / 5 ^ 2 / 4 \* 2

**Diseño**

|  |
| --- |
| Entidad que resuelve: Expresiones aritméticas |
| Variables:  expresion: real // expresion almacena la expresion aritmética a resolver y  resultado: float // almacena el el resultado obtenido luego de realizar los calculos |
| Nombre del algoritmo: expresión\_aritmética2   1. leer expresion 2. resultado 🡨 4 / 2 \* 3 / 6 + 6 / 2 / 1 / 5 ^ 2 / 4 \* 2 // realiza los cálculos y asigna el resultado 3. mostrar resultado |

****

**Ejercicio 4**

**Fase de Análisis**

**Definición del problema:** Obtener las siguientes expresiones aritméticas

**Análisis**

**Datos de entrada**

expresión: Real

**Datos de salida**

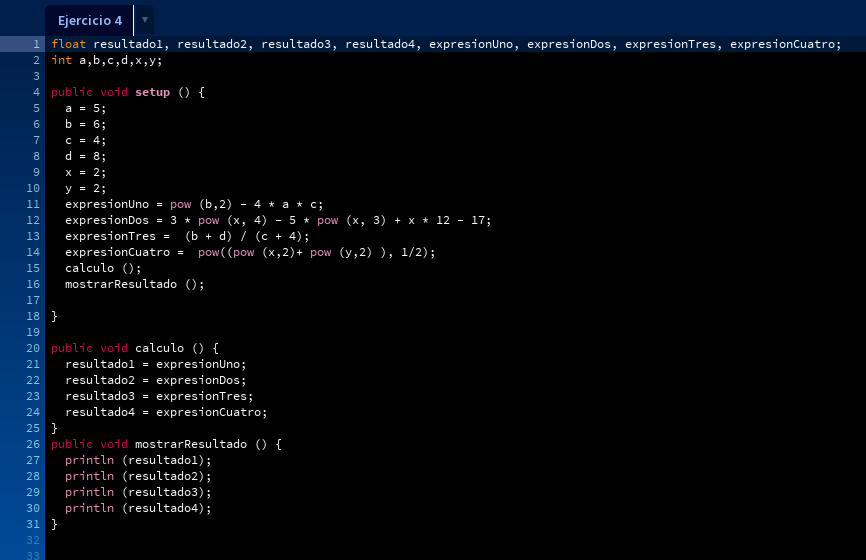
Resultado: Real

**Proceso**

resultado = 4 / 2 \* 3 / 6 + 6 / 2 / 1 / 5 ^ 2 / 4 \* 2

**Fase de diseño**

|  |
| --- |
| Entidad que resuelve: Expresiones aritméticas |
| Variables:  expresión: real // expresión almacena la expresion aritmética a resolver y  resultado: flota // almacena el el resultado obtenido luego de realizar los calculos |
| Nombre del algoritmo: expresión\_aritmética2   1. leer expresión 2. resultado 🡨 4 / 2 \* 3 / 6 + 6 / 2 / 1 / 5 ^ 2 / 4 \* 2 // realiza los cálculos y asigna el resultado 3. mostrar resultado |

****

**Ejercicio 5:**

**Fase de análisis**

**Definición del problema**: Evaluar las expresiones dado A, B y C

**Análisis:**

**Datos de entrada:**

A, B y C: enteros

**Datos de salida:**

resultadoA, resultadoB, resultadoC: real

**proceso:**

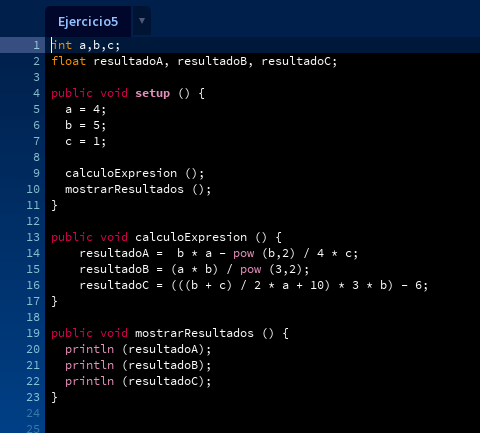
resultadoA: B = A – B ^ 2 / 4 \* C

resultadoB = (A \* B) / 3 ^ 2

resultadoC = (((B + C) / 2 \* A + 10) \* 3 \* B) – 6

**Fase de diseño**

|  |
| --- |
| Entidad que resuelve: Expresiones aritméticas |
| Variables  a, b, c: enteros  resultadoA, resultadoB, resultadoC: float |
| Nombre del algoritmo: calculo\_de\_expresiones  Inicio  Leer a  Leer b  Leer c  resultadoA 🡨 B \* A – B ^ 2 / 4 \* C  resultadoB 🡨(A \* B) / 3 ^ 2  resultadoC🡨(((B + C) / 2 \* A + 10) \* 3 \* B) – 6  mostrar resultadoA  mostrar resultadoB  mostrar resultadoC  fin |

****

**Ejercicio 6**

**Análisis**

Definición del problema: evaluar el resultado dado x = 3, y = 4, z= 1;

**Datos de entrada:**

x, y, z: enteros

**datos de salida:**

r1: entero

r2: boolean

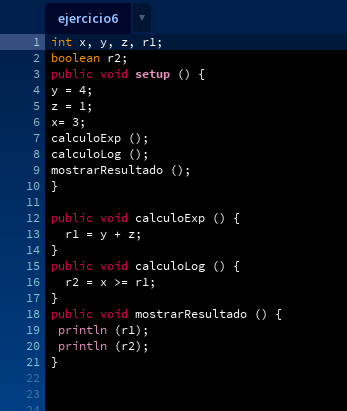
**Proceso:**

R1 = y + z

R2 = x >= R1

**Fase de análisis**

|  |
| --- |
| Entidad que resuelve: expresión algebraica |
| Variables  x, y, z: enteros  R1: enteros  R2: boolean |
| Nombre del algoritmo: evaluar\_resultados  Inicio  Leer x  Leer y  Leer z  R1 🡨 y + z  R2 🡨 x>= R1  Mostrar R1  Mostrar R2  Fin |

****

**Ejercicio 7**

**Fase de análisis**

**Definición del problema**: Obtener los resultados sabiendo que contador1 = 3 y contador2 = 4

**Datos de entrada:**

Contador1, contador2: enteros

**Datos de salida:**

R1 entero

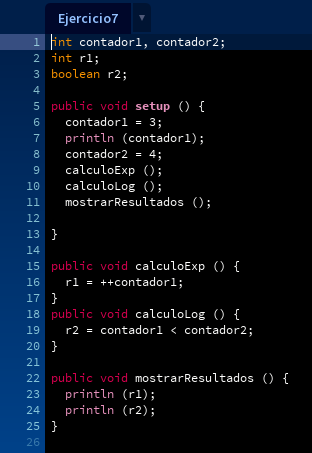
R2 boolean

**Procesos:**

R1 = ++ contador

R2= contador1 < contador2

|  |
| --- |
| Entidad que lo resuelve: expresiones Aritméticas |
| Variables  Contador1, contador2: enteros  R1: entero  R2: boolean |
| Nombre del algoritmo: obtener\_ resultados  Inicio  Leer contador1  Leer contador2  R1 🡨 ++contador1  R2🡨 contador1<contador2  Mostrar contador1  Mostrar contador2  Fin |



Ejercicio 8

**Fase de Análisis**

**Definición del problema**: Evaluar resultados sabiendo que a = 31, b= -1, x =3, y=2

**Datos de entrada:**

a, b, x, y: enteros

**Datos de salida:**

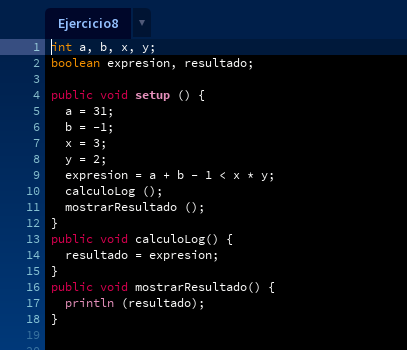
resultado: real

**Proceso:**

resultado = a+b-1 < x \*y

**Fase de diseño**

|  |
| --- |
| Entidad que lo resuelve expresiones algebraicas / lógicas |
| Variables  a, b, x, y: entero  resultado: boolean |
| Nombre del algoritmo: obtener\_resultado  Inicio  Leer a  Leer b  Leer x  Leer y  Resultado 🡨 **a+b-1 < x \*y**  *mostrar* resultado  Fin |

****

**Ejercicio 9**

**Fase de análisis**

**Definición del problema**: obtener el resultado sabiendo que x =6, y =8

**Datos de entrada:**

x, y: entero

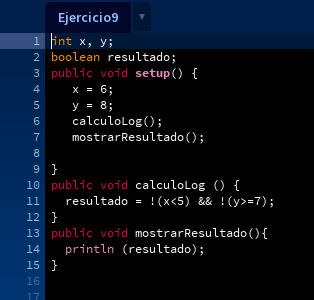
**Datos de salida**

resultado: boolean

**Proceso**

Resultado = !(x<5) && !(y>=7)

|  |
| --- |
| Entidades que lo resuelven: expresiones lógicas |
| Variables  x, y: entero  Resultado boolean |
| Nombre del Algoritmo: obtener\_resultaod  Inicio  Leer x  Leer y  Resultado 🡨 !(x<5) && !(y>=7)  *Mostrar* resultado  Fin |

****

**Ejercicio 10**

**Fase de análisis**

**Descripción del problema**: obtener los resultados, sabiendo que i = 22 y j=3

**Datos de entrada**

i, j = enteros

**Datos de salida**

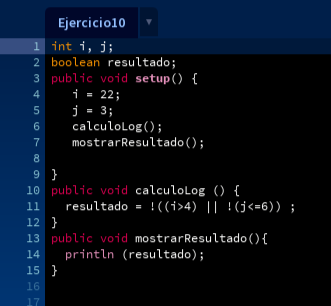
resultado: boolean

**Proceso:**

!((i>4) || !(j<=6))

**Fase de diseño:**

|  |
| --- |
| Entidad que lo resuelve: expresiones lógicas |
| Variables  i, j: enteros  resultado: boolean |
| Nombre del algoritmo: obtener\_resultados  Inicio  Leer I  Leer j  Resultado 🡨 !((i>4) || !(j<=6))  *Mostrar* resultado  Fin |

****

**Ejercicio 11**

**Fase de análisis**

**Descripción del problema:** obtener los resultados sabiendo que a = 34b =12 y c= 8

**Datos de entrada:**

a, b, c: enteros

**Datos de salida:**

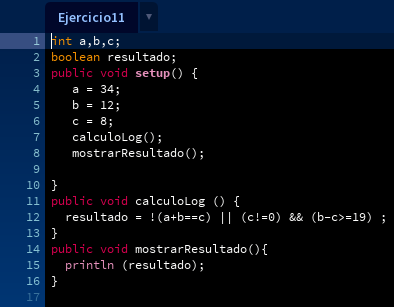
resultado: boolean

**Proceso**

Resultado = !(a+b==c) || (c!=0)CC(b-c>=19)

**Fase de diseño**

|  |
| --- |
| Entidad que lo resuelve: expresiones lógicas |
| Variables:  a, b, c: enteros  resultado: boolean |
| Descripción del algoritmo: obtener\_resultado  Inicio  Leer a  Leer b  Leer c  Resultado 🡨 !(a+b==c) || (c!=0)CC(b-c>=19)  Mostrar resultado  Fin |

****

**Ejercicio 12**

**Fase de Análisis**

**Descripción del problema**: saludar con el nombre brindado

**Datos de entrada:**

Nombre: string

**Datos de salida**

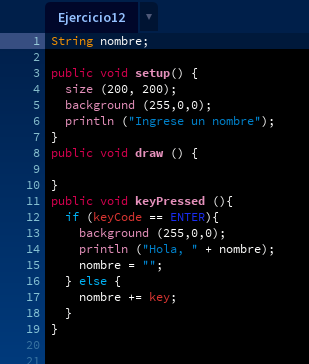
Saludo: string

**Procesos**

Saludar con el nombre ingresado

**Fase de diseño**

|  |
| --- |
| Entidad que lo resuelve: consola |
| Variables  Nombre string  Saludo string |
| Nombre del algoritmo: Nombre\_saludo  Inicio  Nombre 🡨ingresar  Leer nombre  *Mostrar* saludo + nombre  Fin |

****

**Ejercicio 13**

**Fase de analisis**

**Descripción del problema**: Calcular el perímetro y área de un rectángulo sabiendo su base y altura

**Datos de entrada:**

base, altura: entero

**Datos de salida**

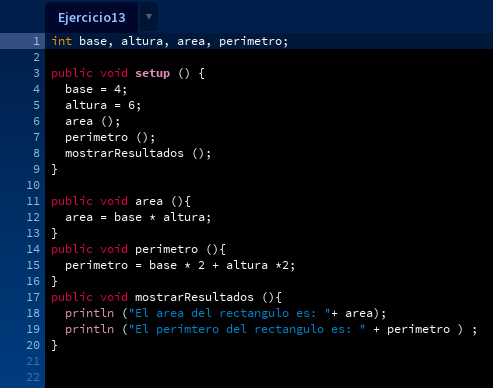
área, perímetro: entero

**procesos:**

área = base \* altura

perímetro = base \*2 + altura \* 2

|  |
| --- |
| Entidad que lo resuelve: Expresiones aritméticas |
| Variables:  base, altura: entero  area, perímetro: entero |
| Nombre del algoritmo: calculos\_rectangulo  Inicio  leer altura  leer base  area 🡨base \* altura  perímetro 🡨base \*2 + altura \* 2  *mostrar* area  *mostrar* perímetro |

****

**Ejercicio 14**

**Fase de análisis**

**Definición del problema**: Obtener la hipotenusa de un triangulo rectángulo conociendo sus catetos

**Datos de entrada:**

catetoA, catetoB: entero

**datos de salida**:

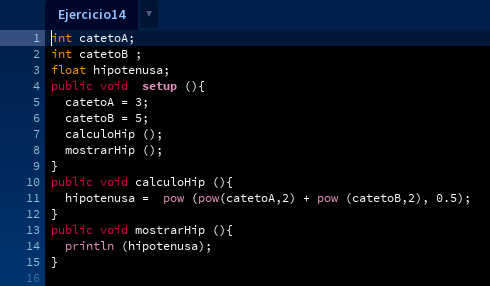
hipotenusa: real

**procesos**

hipotenusa = √(catetoA) ^2 + (catetoB) ^2

**Fase de diseño**

|  |
| --- |
| Entidad que lo resuelve: Teorema de Pitágoras |
| Variables  catetoA, catetoB: entero  hipotenusa: real |
| Nombre del algoritmo: calculo\_hipotenusa  Inicio  Leer catetoA  Leer catetoB  hipotenusa 🡨pow (pow(catetoA,2) + pow (catetoB,2), 0.5)  mostrar hipotenusa  fin |

****

**Ejercicio 15**

**Fase de análisis**

**Definicion del problema**: Conociendo 2 números obtener suma, resta, multiplicación y división de estos.

**Datos de entrada:**

num1, num2: enteros

**Datos de salida:**

suma, resta, multiplicación, división: entero

**procesos**

suma = num1 + num2

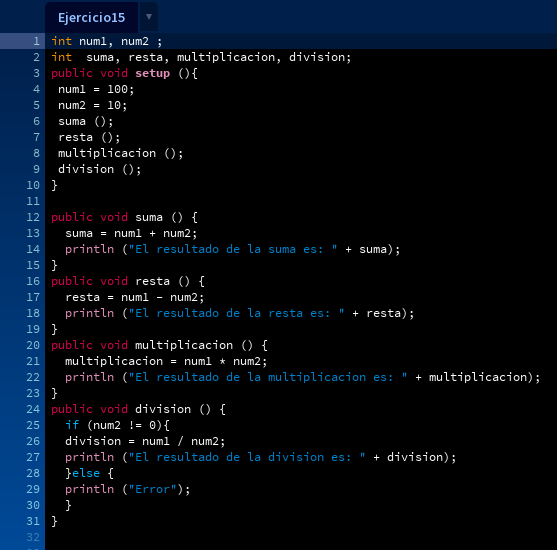
resta = num1 – num2

multiplicación= num1\*num2

división = num1/num2

**Fase de diseño**

|  |
| --- |
| Entidad que lo resuelve: expresiones aritméticas |
| Variables  num1, num2: enteros  suma, resta, multiplicación, división: enteros |
| Definición del algoritmo: operaciones\_mate  Inicio  Leer num1  Leer num2  Suma 🡨num1 + num2  Resta 🡨 num1 – num2  Multiplicación 🡨 num1\* num2  Si num2 != 0 hacer  division = num1 / num2  sino  *mostrar* “error”  fin si  *mostrar* suma  *mostrar* resta  *mostrar* multiplicacion  *mostrar* division  fin |



Ejercicio 16

Definición del problema: Convertir de temperatura Fahrenheit a ˚˚c

Datos de entrada

temperaturaFahrenheit: real

datos de salida

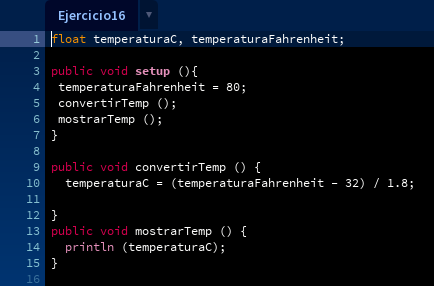
temperaturaC: real

proceso:

temperaturaC = (temperaturaFahrenheit - 32) /1.8

**Fase de diseño**

|  |
| --- |
| Entidad que lo resuelve: entidades aritméticas |
| Variables  temperaturaC, temperaturaFarenheit: real |
| Nombre del algoritmo: convertir\_temp  Leer temperaturaFarenheit  temperaturaC 🡨 (temperaturaFarenheit - 32) /1.8  *mostrar* TemperaturaC |

****

**Ejercicio 17**

**Fase de analisis**

**Definicion del problema:** Calcular la distancia entre el personaje y el power up

**Análisis**

**Datos de entrada:**

x1, y1, x2, y2: real

linkSize, tesoroSize: real

distanciaTesoro: real

**Datos de salida:**

rectangulo y circulo dibujados en el lienzo, el circulo con la posición del mouse

**Procesos:**

Calcular la distancia entre el rectángulo y circulo

**Fase de diseño**

|  |
| --- |
| Entidad que lo resuelve: Lienzo |
| **Variables:**  x1, y1, x2, y2: real  linkSize: real  tesoroSize: real  distanciaTesoro: real |
| **Nombre del algoritmo: distancia\_link**  float x1=100, y1=100, x2=200, y2=400;  float linkSize = 40;  float tesoroSize = 40;  float distanciaTesoro=50;  float coordenadaX, coordenadaY;  float distancia;  String textoDistancia, coordenadas;  Inicio  coordenadaX 🡨 x2 - x1;  coordenadaY 🡨 y2 - y1;    distancia 🡨 sqrt(pow(coordenadaX, 2) + pow(coordenadaY, 2));  textoDistancia 🡨 "la distancia es de: " + distancia;  *mostrar* textoDistancia      si (distancia <= distanciaTesoro) {  *mostrar* ¡Power-Up activado!  fin si  fin |



**Ejercicio 18**

**Fase de análisis**

**Descripción del problema**: Obtener las raíces de una ecuación de segundo grado.

**Datos de entrada:**

a, b, c: entero

**datos de salida**

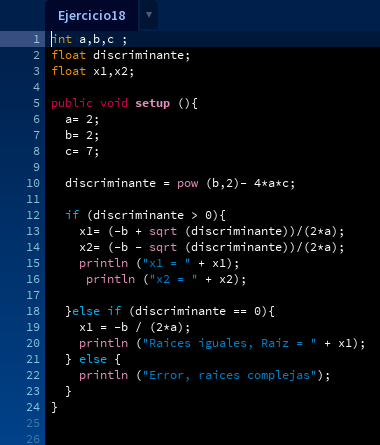
x1, x2 o solamente x1: reales

**proceso:**

discriminante = b ^ 2- 4\*a\*c. Si discriminante > 0 hacer obtener x1 y x2sino si discriminante es = 0x1. Sino mostrar error

Fase de diseño

|  |
| --- |
| Entidad que lo resuelve: discrimante |
| Variables  a, b, c: enteros  discriminante. enteros  x1, x2: reales |
| Nombre del programa: obtener\_raices  Inicio  Leer a  Leer b  Leer c  si  x1🡨 (-b + sqrt (discriminante))/(2\*a);  x2🡨 (-b - sqrt (discriminante))/(2\*a);  *mostrar* x1  *mostrar* x2  sino si discriminante = 0  x1 = -b / (2\*a);  mostrar x1  sino  *mostrar* Error, raíces complejas  fin si  fin |



**Ejercicio 19**

**Fase de análisis**

**Descripción del problema**: Dibujar un circulo y linea en movimiento.

**Datos de entrada**

línea, mov: enteros

**Datos de salida:**

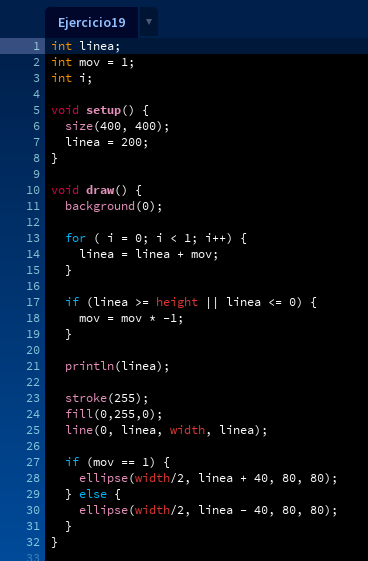
circulo y línea

**Procesos**:

Mostrar en pantalla un circulo y lina en movimiento.

**Fase de diseño**

|  |
| --- |
| Entidad que resuelve: Lienzo |
| Variables:  Línea: entero  mov: entero  i: enteros |
| Nombre del algoritmo: circulo\_linea\_mov  Inicio  Leer línea  Leer mov  Para i 🡨 hasta alto e incrementar en 1 hacre  linea 🡨linea + mov;  fin para  si (linea >= height O linea <= 0)  mov 🡨 mov \* -1;  fin si  mostrar linea  mostrar línea (0, linea, width, linea);  si mov 🡨 1 hacer  dibujo  sinosino  Dibujo  Fin si  fin |



**Ejercicio 20**

**Fase de análisis:**

**Descripción del problema**: Dibujar rectángulo usando estructuras iterativas

**Datos de entrada:**

coordenadaRect: coordenadas cartesianas

ancho, alto, distanciaEntreRect: entero

**Datos de salida:**

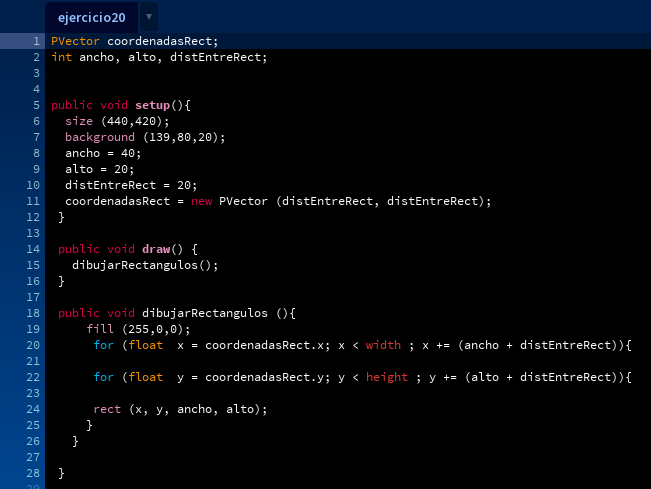
Los rectángulos dibujados en el lienzo

**Proceso:**

Dibujar los rectángulos

**Fase de diseño**

|  |
| --- |
| Entidad: Lienzo |
| Variables:  coordenadasRect: coordenadas  ancho, alto, distanciaEntreRect: enteros |
| Nombre del algoritmo: dibujar\_rectangulos  Algoritmo:  Inicio}  anchoLienzo🡨 440  altoLienzo🡨420  distanciaEntreRect 🡨20  ancho 🡨40  alto🡨20  para x 🡨 coordenadasRect.x hasta anchoLienzo con paso (ancho+ distanciaEntreRect)  para y 🡨coordeandasRect.y hasta altoLienzo con paso (alto+ distanciaEntreRect)  dibujar un rectángulo en (x, y, ancho, alto)  fin para  fin para  fin |



**Ejercicio 21**

**Fase de analisis:**

Descripción del problema: Dibujar escalones con un punto rojo en cada escalon.

**Datos de entrada:**

puntoA, puntoB, puntoB, puntoC, puntoD: ejes cardinales

distancia: entro

**datos de salida:**

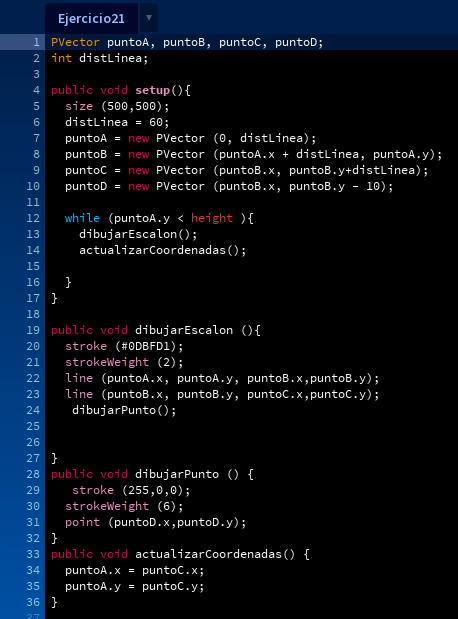
dibujo de los escalones y punto rojo

**procesos**:

Mostrar en el lienzo los escalones junto con el punto rojo

**Fase de diseño**

|  |
| --- |
| Entidad que lo resuelve: lienzo |
| Variables  puntoA,puntoB, puntoC, puntoD: enteros  edistancia: entero |
| Nombre del algoritmo: escalon\_puntos  Inicio  mientras (puntoA.y < anchoDelLienzo hacer  dibujar linea (puntoA.x, puntoA.y, puntoB.x,puntoB.y);  dibujar linea (puntoB.x, puntoB.y, puntoC.x,puntoC.y);  dibujar punto (puntoD.x,puntoD.y);  puntoA.x 🡨 puntoC.x;  puntoA.y 🡨 puntoC.y;  fin mientras |



**Ejercicio 22**

**Fase de analisis**

**Descripción del problema**: Dibujar círculos sobre cada línea de por medio es decir en la línea 1 se dibujan círculos con distanciamiento, en la línea 2 no se dibuja y así sucesivamente

**Datos de entrada:**

numero de líneas y círculos

**Datos de salida:**

Círculos con colores sobre líneas con distanciamiento de por medio.

Proceso: Mostrar en el lienzo círculos con colores aleatorios sobre líneas

**Fase de diseño:**

|  |
| --- |
| Entidad que lo resuelve: Lienzo |
| Variables:  distanciaCiruclo: entero  lineaX, lineaY, circuloX, circuloX, distanciaCirculo : entero |
| Nombre del algoritmo:  hacer  circuloX = distanciaCirculo;  hacer  dibujar línea (lineaX,lineaY,width,lineaY);  dibujar circulos(circuloX,circuloY,50,50);  circuloX 🡨 circuloX + distanciaCirculo\*2;  fin hacer    mientras circuloX sea menor que el ancho hacer  lineaY 🡨 línea + 100;  circuloY 🡨 línea +200;  fin hacer  mientras lineaY sea menor que alto)  fin |

