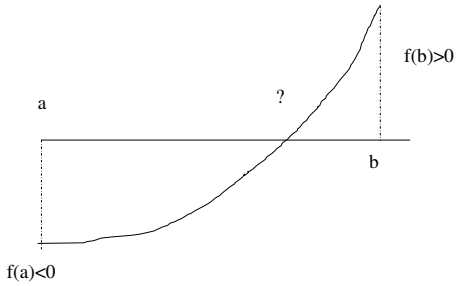
Problema. Calcular raíz de función continua en [a,b] con precisión ε

static public double **raiz**(Funcion **f**,double **a**,double **b**,double **epsilon**)
interface Funcion{public double valor(double x);}

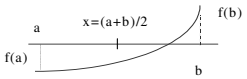


Solución 2. Recursiva

```
static public double raiz(Funcion f,
double a, double b, double epsilon)
{
   double x = (a+b)/2;

   if(b-a <= epsilon) return x;

   if(signo(f.valor(x))==signo(f.valor(a)))
     return raiz(f,x,b,epsilon);
   else
     return raiz(f,a,x,epsilon);
}
```

Solución 1. Método de búsqueda binaria
Algoritmo:
- Determinar x como punto medio del intervalo
- Si f(x) tiene el mismo signo de f(a) entonces repetir el proceso en intervalo [x,b]
- Si f(x) tiene el mismo signo de f(b) entonces repetir el proceso en intervalo [a,x]

Nota. Las iteraciones se detienen cuando el tamaño del intervalo de búsqueda se reduzca hasta alcanzar un epsilon.



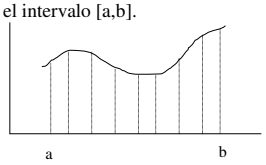
Ejemplo de uso:

```
class F implements Funcion{
public double valor(double x){
   return ...;//expresión con x
}
}
class Coseno implements Funcion{
public double valor(double x){
   return Math.cos(x);
}
}
class Polinomio implements Funcion{
...
public double valor(double x){...}
}
```

```
static public double raiz(Funcion f,
double a, double b, double epsilon){
   double x;
   while(true){
      x = (a + b) / 2;
      if(b-a <= epsilon) break;;
      if(signo(f.valor(x))==signo(f.valor(a)))
         a=x;
      else
         b=x;
   }
   return x;
}
int signo(double x){
   if(x==0) return 0;
   return x>0 ? 1 : -1;
}
```

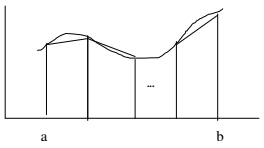
```
class Raices{
static public void main(String[] args)
{
   U.println(
      raiz(new F(),0,3,0.0001));

   U.println(
      raiz(new Coseno(),0,Math.PI,1.0E-5));

   double[]a={1,-3,1}
   U.println(
      raiz(new Polinomio(a),2,3,1.0E-2));

}
}
```

Problema. Calcular el área bajo la curva de una función continua en el intervalo [a,b].

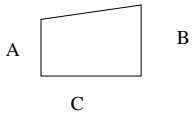



```
static public double area(
Funcion f,double a,double b,int n)
```

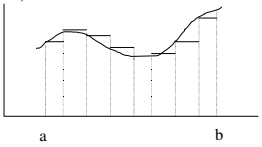
Solución 2. Aproximación por suma de trapecios (del mismo ancho)



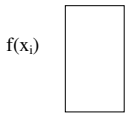
Area de un trapecio de lados A y B y base C = $\frac{A+B}{2} * C$



Solución 1. Aproximación por suma de n rectángulos (del mismo ancho)



Area de cada rectángulo: $f(x_i)$* ancho




```
static public double area(
Funcion f,double a,double b,int n)
{
   double s=0, x=a, ancho=(b-a)/n;

   for(int i=1; i<=n; ++i){
      s += f.valor(x) + f.valor(x+ancho);
      x += ancho;
   }
   return ancho/2 * s;
}
```

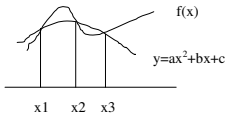
Notas
- Función se evalúa 2 veces en cada punto
- Ancho se suma 2 veces

```
static public double area(
Funcion f,double a,double b,int n)
{
   double
     s=0,
     x=a,
     ancho=(b-a)/n; //de cada rectángulo

   for(int i=1; i<=n; ++i)
   {
      s += f.valor(x);
      x += ancho;
   }
   return s*ancho;
}
```

desarrollando: ancho/2*y_1 + ancho*(y_2+y_3+...+y_{n-1}) + ancho/2*y_n

```
static public double area(
double a,double b,int n,Funcion f)
{
   double s=0, x=a, ancho=(b-a)/n;

   for(int i=2; i<=n-1; ++i)
      s += f.valor( x+=ancho );

   return ancho/2
        * (f.valor(a)+2*s+f.valor(b));
}
```

Solución 3. Aproximación por método de Simpson (por parábolas que unen 3 puntos)



para una función cuadrática $y=ax^2+bx+c$ en $[x_1,x_3]$
área bajo la curva = $ancho/3*(y_1+4y_2+y_3)$
con $y_i=f(x_i)$

desarrollando: $ancho/3*(y_1+4y_2+2y_3+4y_4+2y_5+...+y_n)$

más eficiente (y con menos error):

```
double ancho=(b-a)/n, ancho2=ancho*2;

//sumar pares
double sp=0, x=a-ancho;
for(int i=2; i<=n-1; i+=2)
   sp += f.valor(x+=ancho2);

//sumar impares
double si=0; x=a;
for(int i=3; i<=n-1; i+=2)
   si += f.valor(x+=ancho2);

//resultado final
return ancho/3*
  (f.valor(a) + 4*sp + 2*si + f.valor(b));
```

```
static public double area(
Funcion f,double a,double b,int n)
{
   double s=0, x=a, ancho=(b-a)/n;

   for(int i=2; i<=n-1; ++i){
     double y=f.valor(x+=ancho);
     if(i%2==0) //par?
       s+=4*y;
     else
       s+=2*y;
   }
   return ancho/3*(f.valor(a)+s+f.valor(b));
}
```

Nota.
- `s+=f.valor(x+=ancho)*(i%2==0? 2 : 4);`
- realiza n-2 multiplicaciones por 2 o 4

```
static public double area(
Funcion f,double a,double b,int n)
{
  //sumar pares e impares
   double sp=0, si=0, x=a, ancho=(b-a)/n;
   for(int i=2; i<n; i+=2){
      sp += f.valor(x+=ancho);
      si += f.valor(x+=ancho);
   }
   //resultado final
   return ancho/3*
     (f.valor(a) + 4*sp + 2*si + f.valor(b));
}
```

```
static public double area(
Funcion f,double a,double b,int n)
{
   double sp=0, si=0, x=a, ancho=(b-a)/n;
   for(int i=2; i<=n-1; ++i){
     double y=f.valor(x+=ancho);
     if(i%2==0) //par?
       sp+=y;
     else
       si+=y;
   }
   return
ancho/3*(f.valor(a)+2*sp+4*si+f.valor(b));
}
```

Nota.
- `(i%2==0 ? sp+=y : si+=y);`
- realiza n-2 if con operación resto

Generalización. Aproximación por método de Simpson de orden n (por curvas que unen n puntos)

$Y=\sum a_i x^i$

$Y_1 = \sum a_i x_1^i$

...

$Y_n = \sum a_i x_n^i$

$a_1,...,a_n$ se obtienen resolviendo un sistema de n ecuaciones con n incógnitas (propuesto)