Arreglos

`T[]a=new T[N];`

a: referencia a objeto(arreglo) de N elementos `a[0], ...,a[N-1]`

N: expresión entera>=0 (si <0, excepción)

T: tipo (se inicializa con cero/false) o clase (con null)

inicialización: `T[]a={valor0,...,valorn-1};`

¿tamaño? nombre.length

indexación (acceso a un elemento): `a[índice]`

con 0 ≤ índice (exp int) < nombre.length (sino excepción)

ejemplo

```
String[]d={"lunes",...,"domingo"};
for(int i=0; i<d.length; ++i)
  U.println(d[i]);
```

Arreglos de 2 dimensiones (tablas/matrices):

creación:

```
T[][]nombre=new T[filas][columnas];
```

inicialización:

```
T[][]nombre={{1ªfila},...,{última fila}};
```

indexación:

`nombre[i][j]`

con 0 ≤ i < nombre.length y 0 ≤ j < nombre[i].length

ejemplo:

```
int[][]a={{1,2,3},{1,2},{1}};
for(int i=0; i<a.length; ++i)
  for(int j=0; j<a[i].length; ++j)
    U.print( a[i][j] + " ");
```

escribe: 1 2 3 1 2 1

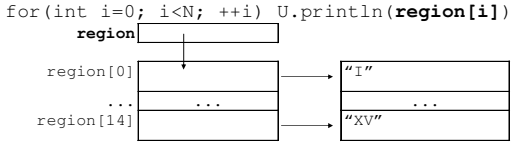
Arreglos como parámetros de Métodos

```
double suma(double[]x){//arreglo completo
  double s=0;
  for(int i=0;i<x.length;++i) s+=x[i];
  return s;
}
double suma(double[]x,int n){//primera parte
  double s=0;
  for(int i=0;i<n;++i) s+=x[i];
  return s;
}
double suma(double[]x,int ip,int iu){//parte
  double s=0;
  for(int i=ip;i<=iu;++i) s+=x[i];
  return s;
}
double[]a={1,2,3,4,5};
U.print(suma(a)+" "+suma(a,3)+" "+suma(a,2,4));
```

escribe 15 6 12

Arreglos de objetos

```
String[]region={"I","II","III",…,"XIV","XV"};
for(int i=0; i<N; ++i) U.println(region[i]);
```




```
setLayout(new GridLayout(N,N));
Button[][]b=new Button[N][N];
for(int i=0;i<N;++i) for(int j=0;j<N;++j){
  add(b[i][j]=new Button(""+(i+1)+","+(j+1)));
  b[i][j].addActionListener(this);
}
```

1,1	...	1,8
...	...	...
8,1	...	8,8

Arreglos en objetos (para representación)

```
class Polinomio{
protected int n;
protected[]double a;
public Polinomio(double[]x){
  a=new double[n=x.length];
  for(int i=0;i<n;++i) a[i]=x[i];
}
public double valor(double x){
  double s=0,p=1;
  for(int i=0;i<n;++i){s+=a[i]*p; p*=x;}
  return s;
}
...
}
double[]coef={5,2,3};
Polinomio p=new Polinomio(coef);//3xx+2x+5
U.print(p.valor(2));//21
```

Toda clase extiende (implícita o explicitamente) a Object

```
class Object{
public String toString(){return ...;}
public boolean equals(Object x){return this==x;}
...//otros métodos
}
class Fraccion extends Object{
protected int a, b;
...
//redefinición de métodos de clase Object
public String toString(){
  return a+"/"+b;
}
public boolean equals(Object x){
  Fraccion f=(Fraccion)x; //casting
  return a*f.b == b*f.a;//compara objetos
}
}
```

- casting para considerar x como Fraccion
- error si x no es de clase Fraccion (o extensión)

Métodos genéricos (aplicables a objetos de distintas clases)

```
//invertir n objetos de arreglo x
static public void invertir(Object[]x,int n){
  for(int i=0; i<n/2; ++i)
    intercambiar(x,i,n-i-1);
}
//intercambiar x[i] con x[j]
static public void intercambiar(Object[]x,int i,int j){
  Object aux=x[i]; x[i]=x[j]; x[j]=aux;
}
//mostrar un arreglo de objetos
static public void mostrar(Object[]x){
  for(int i=0; i<x.length; ++i)
    U.println(x[i].toString());
}
```

- si toString no existe en clase de x[i], se hereda de clase Object
- se puede escribir sólo U.println(x[i]);

Uso de métodos genéricos

```
//para arreglo de strings
String[]s={"C","B","A","B","C"};
mostrar(s);invertir(s,s.length);mostrar(s);

//para arreglo de fracciones
Fraccion[]f={new Fraccion(),
             new Fraccion(1,2),
             new Fraccion("123/4567"),
             new Fraccion(5)};
mostrar(f);invertir(f,f.length);mostrar(f);

//para arreglo heterogeneo
Object[]a={"A",new Fraccion(1/2),"B"};
mostrar(a);invertir(a,a.length);mostrar(a);
```

Comparable: Interface predefinida para clases que admiten comparar objetos

```
interface Comparable{
public int compareTo(Object x);
}
class Fraccion implements Comparable{
...
//redefinición de compareTo de Comparable
public int compareTo(Object x){
  Fraccion f=(Fraccion)x;
  return a*f.b - b*f.a;
}
}
```

Nota

- clase String extiende Object e implementa Comparable

Búsqueda secuencial: O(n) comparaciones

```
int indice(Object x,Object[]y,int n)
{
  for(int i=0; i<n; ++i)
   if(y[i].equals(x)) return i;
  return -1;
}
```

Búsqueda secuencial para arreglo ordenado: O(n) comparaciones

```
int indice(
Comparable x,Comparable[]y,int n)
{
  for(int i=0; i<n; ++i){
    int c=y[i].compareTo(x);
    if(c==0) return i;
    if(c>0) break;
  }
  return -1;
}
```

Búsqueda binaria: O($\log_2 n$) comparaciones

```
int indice(Comparable x,Comparable[]y,int n){
  int ip=0, iu=n-1, im, c;
  while(ip<=iu){
    int im=(ip+iu)/2, c=x.compareTo(y[im]);
    if(c==0) return im;
    if(c<0) iu=im-1; else ip=im+1;
  }
  return -1;
}
int indice(
Comparable x,Comparable[]y,int ip,int iu){
  if(ip>iu) return -1;
  int im=(ip+iu)/2, c=x.compareTo(y[im]);
  if(c==0) return im;
  if(c<0) iu=im-1; else ip=im+1;
  return indice(x,y,ip,iu);
}
```

Ordenamiento: algoritmo O(n^2) de selección y reemplazo

```
void ordenar(Comparable[]x,int n){
  if(n<2) return;
  intercambiar(x,n-1,indiceMayor(x,n));
  ordenar(x,n-1);
}
void intercambiar(Object[]x,int i,int j){
  Object aux=x[i]; x[i]=x[j]; x[j]=aux;
}
int indiceMayor(Comparable[]x,int n){
  int im=0;
  for(int i=1; i<n; ++i)
    if(x[i].compareTo(x[im])>0) im=i;
  return im;
}
```

Introducción a la computación numérica
Propósito. Resolver problemas numéricos (con mucho cálculo de reales) con **eficiencia** (en poco tiempo) y **precisión** (con poco error)

Problemas numéricos:
1.Evaluar polinomios
2.Evaluar series
3.Calcular raíces de funciones continuas
4.Calcular área bajo la curva (integral definida)

"Errores" en números reales

Si A y B son reales, A=a±a' y B=b±b' (a´,b´: errores)

A + B = (a+b) ± (a'+b')

A * B = (a*b) ± (a*b'+a´*b+a'*b')

Nota. Tiempo de * y / ≈ 10 a 100 veces tiempo de +,-,y comparar

Evaluar polinomio

```
public double valor(double x){
   double suma=0, potencia=1;
   for(int i=0; i<=n; ++i) {
      suma += a[i] * potencia; // 1+ y 1*
      potencia *= x;           // 1*
   }
   return suma;
}
```

Evaluar serie

```
double exp(double x,double eps){
   double suma=1, termino=1;
   for(int i=1; termino>eps; ++i)
     suma += termino *= x/i;
   return suma;
}
```

Calcular raíz y área de función f, continua en [a,b]

```
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);
}
static public double area(
Funcion f,double a,double b,int n){
   double sp=0, si=0, x=a, d=(b-a)/(n-1);
   for(int i=2; i<n; i+=2){
      sp += f.valor(x+=d); //pares
      si += f.valor(x+=d); //impares
  }
   return d/3*(f.valor(a)+4*sp+2*si+f.valor(b));
}
```

Matlab: vectores

```
>>a=ones(1,5); %1 1 1 1 1
>>a=zeros(1,5); %0 0 0 0 0
>>a=1:5; %1 2 3 4 5
>>a=1:2:9; %1 3 5 7 9
>>a=linspace(0,1,5) ; %0 0.2500 0.5000 0.7500 1.0000
>>a=rand(1,5); %0.xxxx 0.xxxx 0.xxxx 0.xxxx 0.xxxx
>>a=[1 2 ; 3 4];
>> a(2,2); %4 arreglo(nºfila,nº columna)
>>a(1,: ); %fila 1
>>a(:,2);  %columna 2
>>a(2,2:end); %2ª a última columna de fila 2
>>a*a; %[7 9; 15 22] multiplicación de matrices
>>a.*a;%[1 4;9 16] punto a punto
```

Matlab: instrucciones if/else, while, for

```
a=rand(1);
if a<1/3
  disp('piedra');
elseif a<2/3
  disp('papel');
else
  disp('tijeras');
end
i=1
while i<=10
  disp(i); i=i+1;
end
for i=1..10; %variable=vector
  disp(i);   %instrucciones
end
```

Matlab: funciones

```
%raizArea(a,b,eps,n): raiz y area de f en [a,b]
function[raiz area]=raizArea(a,b,eps,n)
while b-a > eps
  x=(a+b)/2;
  if signo(f(x))==signo(f(a))a=x; else b=x; end
end
raiz=(a+b)/2;
d=(b-a)/(n-1);
par = a+d : 2*d : b-d;
imp = a+2*d : 2*d : b-d;
area=d/3*(f(a)+4*sum(f(par))+2*sum(f(imp))+f(b));
function y=signo(x)%función interna del archivo
if x<0 y=-1; elseif y>0 r=1; else y=0; end
```