Object: clase raíz de toda la jerarquía de herencia de Java

```
class Object
{

public String toString(){
  return ...;
}
//entrega true sólo si es el mismo objeto
public boolean equals(Object x){
  return this==x;//compara referencias
}
...//otros métodos
}
```

Toda clase extiende (implícita o explicitamente) a Object

```
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){
  if( x instanceof Fraccion){
     Fraccion f=(Fraccion)x; //casting
     return a*f.b == b*f.a;//compara objetos
  }
  return false;
}
}
```

- casting para considerar x como Fraccion

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);
```

Búsqueda (de un objeto en un arreglo de objetos)

```
//buscar objeto x en arreglo y de n objetos
//entregar índice de 1ªaparición (-1 si no está)
static public
int indice(Object x,Object[]y,int n){
  for(int i=0; i<n; ++i)
    if(y[i].equals(x)) return i;
  return -1;
}
```

Uso

`indice("A",s,s.length)` entrega 2

`indice(new Fraccion(2,4),f,f.length)`
entrega 1 puesto que f[1] es new Fraccion(1,2)

`indice("A",a,a.length)` entrega 0

Búsqueda Secuencial (lineal, serial)

- búsqueda O(n)
  - búsqueda de orden n (nº de elementos)
  - significado: realiza del orden (de magnitud) de n comparaciones
  - ¿Nº de comparaciones?
  - mínimo=1 (mejor caso)
  - máximo=n (peor caso)
  - promedio= (1+2+3+...+n)/n =sumatoria(i)/n=(n+1)/2≈n/2
  - ej: 500 comparaciones para n=1000 elementos
- ¿búsqueda en arreglo ordenado?

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){
  if( !(x instanceof Fraccion) )
   U.abort("Objeto a comparar no es fraccion");
  Fraccion f=(Fraccion)x;
  return a*f.b - b*f.a;
}
}
```

Nota
- clase String extiende Object e implementa Comparable

Problema. Ordenar un arreglo de n objetos (comparables)

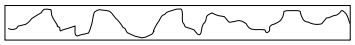
encabezamiento

```
static public void ordenar(Comparable[]x,int n)
```

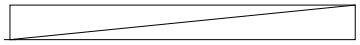
Invocación (ejemplos)

```
String[]s={...};          Fraccion[]f={...};
ordenar(s,s.length);      ordenar(f,f.length);
```

Situación inicial: arreglo desordenado



Situación final: arreglo ordenado



Solución 1. Algoritmo de **selección y reemplazo**/intercambio

Repetir n veces
- **seleccionar** el menor
- **intercambiar** menor con el primero

¿situación intermedia (invariante)?

ip im

ip: índice del primero (de los que aún están desordenados)

im: índice del menor (entre los desordenados)

Si se intercambia x[ip] con x[im] y se avanza ip, entonces la invariante se restablece (mantiene)

```
static public void ordenar(Comparable[]x,int n){
  //repetir n veces
  for(int ip=0; ip<n; ++ip)//para ip=0,...,n-1
  {
    //seleccionar el (indice del) menor
    int im=ip;
    for(int i=ip+1; i<n; ++i)
      if(x[i].compareTo(x[im])<0) im=i;

    //intercambiar menor con el primero
    intercambiar(x,im,ip);
  }
}
```

Algoritmo $O(n^2)$
- las n iteraciones realizan n-1, n-2, ..., 1 comparaciones
- total de comparaciones: Σi, para i=1,...,n-1=n(n-1)/2=$n^2/2$ - n/2
- orden de magnitud: n^2
- ej, si n=100, se realizan del orden de 10.000 comparaciones

Solución 2 ("simétrica")
- **seleccionar** el mayor
- **intercambiar** mayor con el último

Repetir n veces (o recursivamente hasta que n<2)

```
void ordenar(Comparable[]x,int n){
  if(n<2) return; //caso base
  intercambiar(x, n-1, indiceMayor(x,n));
  ordenar(x, n-1);
}
int indiceMayor(Comparable[]x,int n){
  if(n<2) return n-1;//0 si n=1, -1 si n=0
  int im=indiceMayor(x,n-1);
  return x[im].compareTo(x[n-1])>0? im: n-1;
}
```

Notas
- existen otros algoritmos que se estudiarán posteriormente
- algunos $O(n\log_2 n)$, es decir, que si n=100, realizan del orden de $100*\log_2 100 \approx 700$ comparaciones (mucho menos que 10000).

Búsqueda secuencial para arreglo ordenado:

O(n), pero peor caso realiza en promedio n/2 (y no n) comparaciones

```
int indice(Comparable x,Comparable[]y,int ip,int iu){
  //recorrer todos los índices (de 1º a último)
  for(int i=ip; i<=iu; ++i){
    //si y[i]=x, devolver índice i
    int c=y[i].compareTo(x);
    if(c==0) return i;
    //si y[i]>x, no está (pq y está ordenado)
    if(c>0) break;
  }
  return -1;
}
```

recursivamente

```
  if(ip>iu) return -1;
  int c=y[ip].compareTo(x);
  if(c==0) return ip;
  return c>0? -1: indice(x,y,ip+1,iu);
```

Problema. Programar el algoritmo de **búsqueda binaria** (para buscar eficientemente en arreglo ordenado)

encabezamiento

```
static public int indice(
Comparable x,Comparable[]y,int ip,int iu)
```

¿algoritmo?

Comparar **x** con elemento que está en la mitad del arreglo **y**

- Si son iguales, devolver índice de mitad
- Si x es menor, buscar en primera mitad del arreglo (descartando segunda mitad)
- Si x es mayor, buscar en segunda mitad del arreglo (descartando primera mitad)

Repetir mientras queden elementos.
Devolver –1 si x no está en arreglo y.

```
static public int indice
(Comparable x,Comparable[]y,int ip,int iu){
  //repetir mientras queden elementos
  while(ip<=iu){
    //comparar x con elemento que está en la mitad
    int im=(ip+iu)/2; //indice del medio
    int c=x.compareTo(y[im]);
    //si son iguales, devolver índice de mitad
    if(c==0) return im;
    //si x es menor, buscar en 1ª mitad del arreglo
    if(c<0)
      iu=im-1;//"subir" índice del último
    //si x es mayor, buscar en 2ª mitad del arreglo
    else
      ip=im+1;//"bajar" índice del primero
  }
  //devolver -1 si x no está en arreglo y.
  return -1;
}
```

Búsqueda binaria recursiva

```
static public int indice
(Comparable x,Comparable[]y,int ip,int iu){
  //devolver -1 si x no está en y(o y vacío)
  if(ip>iu) return -1; //no esta
  //comparar x con elemento que está en la mitad
  int im=(ip+iu)/2;   //indice del medio
  int c=x.compareTo(y[im]);
  //si son iguales,devolver índice de mitad
  if(c==0) return im;
  //si x es menor, buscar en 1ª mitad de arreglo
  if(c<0)
    return indice(x,y,ip,im-1);
  //si x es mayor,buscar en 2ª mitad del arreglo
  else
    return indice(x,y,im+1,iu);
}
```

Ejemplo de uso:

```
static public void main(String[]args){
  String[]mes={"abril",...,"septiembre"};
  String[]month={"april",..., "september"};

  int i=indice(args[0],mes,0,mes.length-1);

  U.println(
    i<0? "mes incorrecto": month[i]);
}
```

invocación

```
java TraducirMes mayo
may
java TraducirMes martes
mes incorrecto
```

Búsqueda Binaria

- búsqueda $O(\log_2 n)$
  - realiza del orden de $\log_2 n$ comparaciones
- ¿Nº de comparaciones?
  - mínimo=1 (mejor caso)
  - máximo=$\log_2 n$ (peor caso)
  - promedio= $\log_2 n/2$
  - ej: 10/2=5 comparaciones para n=1024=2^{10} elementos

Búsqueda binaria modificada (eligiendo índice al azar)

```
static public int indice
(Comparable x,Comparable[]y,int ip,int iu)
{
  if(ip>iu) return -1;

  int i=U.azar(ip,iu);

  int c=x.compareTo(y[i]);
  if(c==0) return i;
  return c<0 ?
    indice(x,y,ip,i-1): //buscar en 1ª parte
    indice(x,y,i+1,iu); //buscar en 2ª parte
  }
}
```