Problema. Escribir un programa que calcule el promedio de una cantidad indeterminada de números:

```
//acumulador(sumatoria) y contador de números
double suma=0; int n=0;

//obtener primer número
double numero=U.readDouble("n° ? ");

//repetir hasta fin de datos
while( numero != 0 )
{
  //procesar numero
  suma=suma+numero; n=n+1;
  U.println("promedio="+suma/n);

  //obtener siguiente número
  numero=U.readDouble("n° ? ");
}
```

Solución 2: usando operador de asignación

```
double suma, numero; int n;

suma = n = 0;

while((numero=U.readDouble("n°?"))!=0)

  U.println("promedio=" +
    (suma=suma+numero)/(n=n+1));
```

Nota. Solución más breve:
- 5 líneas en lugar de 7
- 5 instrucciones en lugar de 8

Operador de asignación

ejemplo
```
suma=suma+numero
```

sintaxis : `variable = expresión (sin ;)`

semántica
1º evaluar expresión
2º asignar resultado a variable
3º entregar resultado (del tipo de la variable)

prioridad: más baja. Ej: `x=y+z` ⇔ `x=(y+z)`

asociatividad : de derecha a izquierda.
`Ej: suma=n=0` ⇔ `suma=(n=0)`

Operadores de asignación aritméticos

ejemplo
```
U.println("promedio="+(suma+=numero)/(n+=1));
```

sintaxis
```
variable operador= expresión
operador:  +, -, *, /, %
```

semántica
`variable = variable operador (expresión)`

`x*=y+z` ⇔ `x=x*(y+z)` ⇎ `x=x*y+z`

Operadores de incremento y decremento

ejemplo
```
U.println("promedio="+(suma+=numero)/++n);
```

sintaxis
```
++variable
--variable
```

semántica
`variable+=1` ⇔ `variable=variable+1`
`variable-=1` ⇔ `variable=variable-1`

Operadores de incremento/decremento de postfijo o sufijo

ejemplo
```
int n=10;
U.print(n++); //escribe 10
U.print(n);   //escribe 11
```

sintaxis
```
variable++
variable--
```

semántica
1º recordar (guardar) valor original de variable
2º sumar/restar 1 a variable
3º devolver valor original como resultado

Operador de expresión condicional
sintaxis
```
condición ? expresión1 : expresión2   ¡¡ 3 ops!!
```

semántica
si condición es true, evaluar y entregar resultado de expresión1
sino (es false) evaluar y entregar resultado de expresión2

ejemplo
```
static public int max(int x,int y){
  return x>y ? x : y ;
}
```
es equivalente a las 4 líneas:
```
  if(x>y)
    return x;
  else
    return y;
```

Más ejemplos:
```
static public int factorial(int x){
  return x==0 ? 1 : x*factorial(x-1);
}
```
es equivalente a:
```
static public int factorial(int x){
  if(x==0) return 1; else return x*factorial(x-1);
}

static public int digitos(int x){
  return x<10 ? 1 : 1+digitos(x/10);
}
```
es equivalente a:
```
static public int digitos(int x){
  if(x<10) return 1; else return 1+digitos(x/10);
}
```

```
static public double potencia(double x,int y)
{
  if(y==0) return 1;
  if(y<0) return 1/potencia(x,-y);
  double p=potencia(x, y/2);
  return y%2==0 ? p*p : x*p*p;
}

//máximo común divisor
static public int mcd(int x,int y)
{
  int max=Math.max(x,y), min=Math.min(x,y);
  return min==0 ? max : mcd(min, max % min);
}
```

Problema. Escribir la sgte tabla:
11 12 13 14 15 16 17 18 19 20
…
91 92 93 94 95 96 97 98 99 100
Solución 1:
```
int i=11;
while(i<=100){
  if(i%10==0)U.println(i);else U.print(i+" ");
  ++i;
}
```
Solución 2:
```
int i=11;
while(i<=100){
  U.print(i + (i%10==0 ? "\n" : " "));
  ++i;
}
```
Nota: "\n" significa newline (salto a la sgte línea)

Abuso (encajonamiento de operador ?)

```
int max(int x,int y,int z){
  return x>y ? (x>z?x:z) : (y>z?y:z);
}
int max(int x,int y,int z){
  return x>=y && x>=z ? x : (y>z?y:z);
}
```
Uso
```
int max(int x,int y,int z){
  return  max(max(x,y),z);
}
int max(int x,int y){
  return x>y ? x : y;
}
```

prioridades de operadores (orden de evaluación)

Nº	tipo	operadores
1	sufijo	++ --
2	unario	+ - ! ++ --
3	casting	(tipo)
4	multiplicativo	* / %
5	aditivo	+ -
6	relación	< > <= >=
7	igualdad	== !=
8	and	&&
9	or	||
10	condicional	?:
11	asignación	= op=

Nota. Los operadores condicional y de asignación tienen asociatividad de derecha a izq. Ej: a=b=c es a=(b=c)

Problema. función que entregue true si un entero es primo (o false si no)

Solución 1. "fuerza bruta" (dividir nº por 2,3,…,nº-1)

```
static public boolean primo(int x)
{
  int i=2;
  while(i<x){ //i=2,5,…,x-1
    if(x%i==0) return false;//divisibles
    ++i;
  }
  return true;//divisibles por 1
}
```

Solución 2. probando sólo con impares

```
if(x%2==0) return x==2;//2 es primo
int i=3;
while(i<x){ //i=3,5,…,x-1 (impares)
  if(x%i==0) return false;
  i += 2;
}
return true;
```

Solución 3. probando hasta raíz (si hay divisor > también hay <)

```
if(x%2==0) return x==2;
int i=3;
while(i<=(int)Math.sqrt(x)){//i=3,5,…,√x
  if(x%i==0) return false;
  i += 2;
}
return true;
```

Solución 2. Con instrucción for

```
static public boolean primo(int x)
{
  if(x%2==0) return x==2;

  for(int i=3; i<=(int)Math.sqrt(x); i+=2)
    if(x%i==0) return false;

  return true;
}
```

Instrucción for

sintaxis:

```
for(inicialización; condición; reinicialización)
  instrucción //o { instrucciones }
```

semántica:

```
{
  inicialización;
  while(condición){
    instrucción(es)
    reinicialización;
  }
}
```

Notas

- inicialización y reinicialización puede ser cualquier instrucción
- inicialización o condición o reinicialización puede omitirse

  `Ej:for(;;){…break;…}` ⇔ `while(true){…break;…}`

- Si inicialización es una instrucción de declaración, las variables "desaparecen" al terminar la ejecución de la instrucción for y por lo tanto pueden ser redefinidas posteriormente (puesto que instrucción for está rodeada implícitamente por un bloque)

```
U.println("primos terminados en 1:");
for(int i=1; i<=100; i+=10)
  if(primo(i)) U.println(i);

U.println("primos terminados en 3:");
for(int i=3; i<=100; i+=10)
  if(primo(i)) U.println(i);
```

- instrucciones for pueden encajonarse (anidarse)

```
for(int i=1; i<=9; i+=2)
{
  U.println("primos terminados en "+i);
  for(int j=i; j<=100; j+=10)
    if(primo(j)) U.println(j);
}
```

Problema. Función que entregue el Nº de días de un mes

```
static public int diasMes(int x){ ... }
```

Solución 1. Con if sin else

```
int d=0;
if( x==1 ) d=31;
if( x==2 ) d=28;
if( x==3 ) d=31;
if( x==4 ) d=30;
if( x==5 ) d=31;
if( x==6 ) d=30;
if( x==7 ) d=31;
if( x==8 ) d=31;
if( x==9 ) d=30;
if( x==10 ) d=31;
if( x==11 ) d=30;
if( x==12 ) d=31;
return d;
```

evalúa 12 condiciones siempre

Solución 2. Con selección múltiple if-else if-...-else

```
int d=0;
if( x==1 ) d=31;
else if( x==2 ) d=28
else if( x==3 ) d=31;
else if( x==4 ) d=30;
else if( x==5 ) d=31;
else if( x==6 ) d=30;
else if( x==7 ) d=31;
else if( x==8 ) d=31;
else if( x==9 ) d=30;
else if( x==10 ) d=31;
else if( x==11 ) d=30;
else if( x==12 ) d=31;
return d;
```

evalúa x condiciones (12 en el peor caso)

Solución 3. Con Instrucción switch (una evaluación)

```
int d;
switch(x){
case 1:  d=31; break;
case 2:  d=28; break;
case 3:  d=31; break;
case 4:  d=30; break;
case 5:  d=31; break;
case 6:  d=30; break;
case 7:  d=31; break;
case 8:  d=31; break;
case 9:  d=30; break;
case 10: d=31; break;
case 11: d=30; break;
case 12: d=31; break;
default: d=0;
}
return d;
```

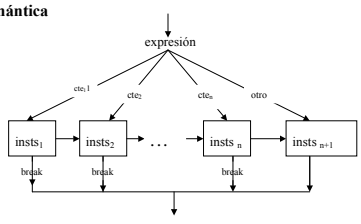
Instrucción switch

Sintaxis

```
switch(expresión){
case constante1 : instrucciones1; break;
. . .
case constanten : instruccionesn; break;
default: instruccionesn+1; break;
}
```

- expresión y constantes: tipo entero
- instrucciones, break y default se pueden omitir

semántica



Notas

- La expresión se evalúa una sóla vez y se bifurca directamente a las instrucciones correspondientes
- Si las instrucciones no terminan con break se pasa automaticamente a las instrucciones del caso siguiente

Solución 4

```
int d=0;
switch(x){
case 1:case 3:case 5:case 7:case 8:case 10:case 12:
  d=31; break;
case 4:case 6:case 9:case 11:
  d=30; break;
case 2:
  d=28; break;
}
return d;
```

alternativamente:

```
switch(x){
case 1:case 3:case 5:case 7:case 8:case 10:case 12:
  return 31;
case 4:case 6:case 9:case 11:
  return 30;
case 2:
  return 28;
}
return 0;
```

Ejercicio

```
//escribir dígito en palabras (sin saltar línea)
//ej: printDigito(7); escribe siete
static public void printDigito(int x){//0<=x<=9
…
}
//escribir en palabras cuenta regresiva desde 10
static public void main(String[]args){
…
}
```

Resultado:

```
diez
nueve
…
uno
cero
```

```
//escribir dígito en palabras
static public void printDigito(int x)
{
  switch(x){
  case 0: U.print("cero"); break; //o return;
  case 1: U.print("uno"); break;
  case 2: U.print("dos"); break;
  case 3: U.print("tres"); break;
  case 4: U.print("cuatro"); break;
  case 5: U.print("cinco"); break;
  case 6: U.print("seis"); break;
  case 7: U.print("siete"); break;
  case 8: U.print("ocho"); break;
  case 9: U.print("nueve"); break;
  }
}
```

Nota. return; regresa un método void a la instrucción sgte a la llamada

```
//escribir en palabras cuenta regresiva desde 10
static public void main(String[]args)
{
  U.println("diez");
  for(int i=9; i>=0; --i){
    U.printDigito(i);
    U.println("");//U.print("\n");
  }
}
```

Propuestos

1. void printDD(int x): escribe número de 2 dígitos
2. Cuenta regresiva desde 100
3. void printDDD(int x): escribe número x de 3 dígitos
4. void printEnRomano(int x)//x de 2 dígitos