# CC10A

## Introducción a la Computación

## Sección 05 - 2004

Profesor: Andrés Muñoz O.

Auxiliares: Agustín Almonte, Marcelo Muñoz, Patricio Salles


# GUÍA DE EJERCICIOS

## CONCEPTOS BÁSICOS DE PROGRAMACIÓN

# Tabla de Contenidos

## Cuenta Intervalos

Simular el siguiente diálogo:

```
Ingrese cota entera inferior? 23
Ingrese cota entera superior? 30
23
24
25
26
27
28
29
30
Ingrese cota entera inferior? …        // repetir por siempre
```

Considere los siguientes casos:

- La cota inferior no puede ser mayor que la superior
- Las cotas pueden ser menores que 0

### *Solución*

Veamos el algoritmo para resolver esto:

1. Escribir en pantalla el texto "Ingrese cota entera inferior?"
2. Pedir el valor de la cota inferior y guardarlo
3. Escribir en pantalla el texto "Ingrese cota entera superior?"
4. Pedir el valor de la cota superior y guardarlo
5. Si la cota inferior es menor o igual que la cota superior:
   5.1. Definir un contador para recorrer paso a paso entre la cota inferior y la cota superior
   5.2. Escribir en pantalla el valor del contador
   5.3. Incrementar el contador en 1
   5.4. Si el contador no es superior a la cota superior, entonces volver a 5.2
6. Si la cota inferior es superior a la cota superior
   6.1. Escribir en pantalla "La cota inferior debe ser menor o igual a la cota superior"
7. Volver al paso 1

Este algoritmo se ve bastante elaborado, pero es entendible. Veamos el código:

```
Console c = new Console();
while (true) {
    c.print("Ingrese cota entera inferior?");
    int cInf = c.readInt();
    c.print("Ingrese cota entera superior?");
    int cSup = c.readInt();
    if (cInf <= cSup) {
        int cont = cInf;
```

```
        while (cont <= cSup) {
            c.println(cont);
            cont = cont + 1;
        }
    }
    else {
        c.println("La cota inferior debe ser menor o igual a la cota superior");
    }
}
```

En el caso de este problema, no posee muchas soluciones distintas, solo algunos "sabores" más que nada, pero siempre se pueden hacer algunas "modificaciones" que lo hacen distinto. Por ejemplo:

```
Console c = new Console();
while (true) {
  c.print("Ingrese cota entera inferior?");
  int cInf = c.readInt();
  c.print("Ingrese cota entera superior?");
  int cSup = c.readInt();
  if (cInf <= cSup) {
    for(int cont=cInf ; cont <= cSup ; cont++)
      c.println(cont);
  }
  else {
    c.println("La cota inferior debe ser menor o igual a la cota superior");
  }
}
```

Un cambio muy leve, pero que en el fondo muestra distintos estilos de programación.

## Detector de Primos

Se desea escribir un programa que despliegue cuales son los números primos entre un intervalo de valores enteros. Para ello, se debe pedir dos valores al usuario: cota inferior y cota superior, para luego mostrar todos los primos que hay entre esas dos cotas (incluyendo ambas).

**Nota**: Recuerde que un número es primo si es divisible por si mismo y por 1, excepto el 1. Considere el 2 como primo. Los negativos no son primos.

### *Solución*

Veamos un algoritmo iterativo para resolver este problema:

1. Escribir en pantalla "Ingrese cota inferior de búsqueda?"
2. Pedir y guardar la cota inferior
3. Escribir en pantalla "Ingrese cota superior de búsqueda?"
4. Pedir y guardar la cota superior
5. Si las cotas son mayores que 0 y la cota inferior es menor o igual a la cota superior
   5.1. Para cada número que se encuentre entre la cota inferior y superior
      5.1.1. Calcular el resto de la división del número y todos los números entre 2 y el número – 1
      5.1.2. Si el resto es distinto de 0
         5.1.2.1. Escribir en pantalla el número concatenado con " es primo!"

Parece un poco complejo el algoritmo. En realidad no lo es tanto si pensamos que estamos comprimiendo en una línea ciclos completos. Veamos la solución y verás que no es tan difícil de escribir:

```
Console c = new Console();
c.print("Ingrese cota inferior de búsqueda?");
int cInf = c.readInt();
c.print("Ingrese cota superior de búsqueda?");
int cSup = c.readInt();
if (cInf > 0 && cSup > 0 && cInf <= cSup) {
    for (int cont=cInf; cont<=cSup; cont++) {
        boolean divisible = false;
        for(int div=2; div<cont; div++) {
            if (cont % div == 0) {
                divisible = true;
            }
        }
        if (!divisible) {
            c.println(cont + " es primo!");
        }
    }
}
```

Apareció una variable que no estaba considerada en el algoritmo. La variable es "divisible". ¿Para qué sirve?, simple. Si analizamos por partes los ciclos internos tenemos:

```
        for(int div=2; div<cont; div++) {
```

Dentro de este ciclo entonces validaremos que el número sea o no divisible por cualquier número entre 2 y el número – 1. Para ello suponemos ANTES del ciclo que el número no es divisible:

```
        boolean divisible = false;
        for(int div=2; div<cont; div++) {
```

Y dentro del ciclo nos preocupamos en buscar cuando encontremos un divisor:

```
            if (cont % div == 0) {
                divisible = true;
            }
```

Una vez que lo hemos encontrado, ya no lo tenemos que imprimir. Entonces, la utilidad es simplemente de saber el "estado" del número cont, que nos indica cuando es o no divisible por otro número distinto de 1 y de si mismo.

Si quisiéramos optimizar un poquito el programa, podríamos escribir lo siguiente:

```
Console c = new Console();
c.print("Ingrese cota inferior de búsqueda?");
int cInf = c.readInt();
c.print("Ingrese cota superior de búsqueda?");
int cSup = c.readInt();
if (cInf > 0 && cSup > 0 && cInf <= cSup) {
    for (int cont=cInf; cont<=cSup; cont++) {
        boolean divisible = false;
        for(int div=2; div<cont; div++) {
            if (cont % div == 0) {
                divisible = true;
                break;
            }
        }
        if (!divisible) {
            c.println(cont + " es primo!");
        }
    }
}
```

Agregando el break, reducimos el tiempo de ejecución del programa sobre todo cuando el intervalo de números es muy grande o cuando las cotas son demasiado altas. Entonces, si encontramos un divisor distinto de 1 y si mismo, no seguimos buscando, ya que no es necesario.

Siendo de aquellos que les gustan los whiles, podemos cambiar la visión del programa y ponerlo en forma de ciclo while:

```
Console c = new Console();
c.print("Ingrese cota inferior de búsqueda?");
int cInf = c.readInt();
c.print("Ingrese cota superior de búsqueda?");
int cSup = c.readInt();
if (cInf > 0 && cSup > 0 && cInf <= cSup) {
    int cont=cInf;
    while (cont<=cSup) {
        boolean divisible = false;
        int div=2;
        while(div<cont) {
            if (cont % div == 0) {
                divisible = true;
                break;
            }
            div++;
        }
        if (!divisible) {
            c.println(cont + " es primo!");
        }
        cont++;
    }
}
```

Aquí está la versión con whiles. Si además contamos las combinaciones que hay entre for y while, podríamos decir que ya tenemos 4 posibles soluciones. Hartas para un problema tan simple.

## Reversador de Textos

Haga un programa que pida una palabra o frase al usuario y que éste devuelva la palabra reversada, es decir, escrita completamente al revés.

### *Solución*

Veamos el algoritmo de la versión iterativa para resolver esto:

1. Escribir en la pantalla el texto "Por favor escriba un texto?"
2. Pedir el valor del texto y guardarlo
3. Darle un valor a un contador igual a la posición del último carácter del texto
4. Escribir en pantalla el carácter indicado por el contador
5. Si el contador no es el primer carácter
   5.1. Quitarle 1 al contador actual
   5.2. Volver al paso 4

Este algoritmo es simple, pero tiene varias soluciones:

```
Console c = new Console();
c.println("Por favor escriba un texto?");
String texto = c.readLine();
int cont = texto.length() - 1;
while(true) {
    c.print(texto.charAt(cont));
    if (cont == 0) {
        break;
    }
    cont--;
}
```

De hecho la solución varía un poquito de lo planteado en la solución algorítmica, ya que en este caso la validación algorítmica era si el contador no era el primer carácter, la validación en Java quedó todo al contrario, como condición de salida del while.

Otra solución más elegante sería utilizar la condición de salida del while:

```
Console c = new Console();
c.println("Por favor escriba un texto?");
String texto = c.readLine();
int cont = texto.length() - 1;
while(cont >= 0) {
    c.print(texto.charAt(cont));
    cont--;
}
```

En este caso la condición varía un poco, ya que si se ejecuta completo el ciclo en el último caso (cont=0), al volver al principio del while el valor será -1. Así nos aseguramos que esté funcionando correctamente.

Por último, no falta la forma de "for":

```
Console c = new Console();
c.println("Por favor escriba un texto?");
String texto = c.readLine();
for(int cont=text.length()-1; cont >= 0; cont--) {
    c.print(texto.charAt(cont));
}
```

Se ve más corta aún, pero sin desmedro de funcionar con while. Pero esto no es todo. Transformemos esta última solución a un método:

```
static public String reversar(String texto) {
    String rev = "";
    for(int cont=text.length()-1; cont >= 0; cont--) {
        rev = rev + texto.charAt(cont);
    }
}
```

Y la llamada sería más simple aún:

```
Console c = new Console();
c.println("Por favor escriba un texto?");
String texto = c.readLine();
c.println(reversar(texto));
```

¿Cuál es la idea? es que un proceso de estas características puede ser reutilizable, por lo que podríamos dejar este código en una "biblioteca". De todas formas, si no nos gustase la forma de hacerlo (con for), podemos cambiar el método, sin afectar el programa principal.

Seamos más visionarios. ¿Se puede hacer recursivo?. La respuesta es si, ya que se puede suponer lo siguiente:

- **Caso Base:** El reverso de un texto de una letra, es la misma letra
- **Caso General:** El reverso de un texto es la última letra concatenado con el reverso de las anteriores

Veamos esto puesto en el método en Java, y verás que es muy elegante:

```
static public String reversar(String texto) {
    return texto.charAt(texto.length()-1) +
           reversar(texto.substring(0, texto.length()-1);
}
```

# Búsqueda de un Texto

Todos conocen el funcionamiento de IndexOf. La idea es programar dos métodos que representen las formas de este método, solo usando comparaciones y obtenciones de caracteres. Esto significa que debes programar:

- **public static int encontrar(String texto, String patron)** que retorna el índice de donde se encuentra el primer carácter del patrón dentro del texto del parámetro. Si el patrón no se encuentra íntegramente dentro de texto, debe retornar -1.

- **public static int encontrar(String texto, String patron, int comienzo)** que hace exactamente lo mismo, pero partiendo desde el carácter indicado como comienzo en adelante.

- **public static int cuentaApariciones(String texto, String patron)** que retorna el número de apariciones que tiene el patrón en el texto indicado. Este método debe usar al menos uno de los métodos anteriores (no use IndexOf).

## *Solución*

Primero, y como lo hemos hecho hasta ahora, es escribir el algoritmo. Veamos cada una de las partes por separado, pensando en texto y patrón como valores conocidos:

### Método para encontrar primera aparición

1. Para cada letra de texto
   1.1. Definir un valor -1 para la posición de encontrado
   1.2. Para cada letra de patrón
      1.2.1. Si la letra indicada de texto no es igual a la letra indicada de patrón
         1.2.1.1. Dejar de verificar patrón
      1.2.2. Pasar a la siguiente letra de patrón y de texto
   1.3. Si se han comparado todas las letras de patrón encontrándolo dentro de texto
      1.3.1. Dar valor para la posición de encontrado igual a la posición de la letra indicada de texto y terminar de recorrer texto
2. Retornar el valor de la posición de encontrado

Uff, algo no muy simple de leer. Veamos el siguiente:

### Método para encontrar aparición desde una posición

1. Para cada letra de texto a partir de la posición indicada
   1.1. Definir un valor -1 para la posición de encontrado
   1.2. Para cada letra de patrón
      1.2.1. Si la letra indicada de texto no es igual a la letra indicada de patrón
         1.2.1.1. Dejar de verificar patrón
      1.2.2. Pasar a la siguiente letra de patrón y de texto
   1.3. Si se han comparado todas las letras de patrón encontrándolo dentro de texto

1.3.1. Dar valor para la posición de encontrado igual a la posición de la letra indicada de texto y terminar de recorrer texto
2. Retornar el valor de la posición de encontrado

Pues realmente igual a la anterior, excepto que parte desde una posición específica. Sigamos con el tercer método:

## Método para contar apariciones de un patrón

1. Definir cuenta de apariciones como 0
2. Buscar y guardar el valor del patrón buscando en todo el texto (1ª solución)
3. Si el resultado es mayor que -1
   3.1. Incrementar en 1 la cuenta de apariciones
   3.2. Buscar y guardar el valor del patrón buscando desde la posición anteriormente encontrada (2ª solución).
   3.3. Volver al punto 3
4. Retornar la cuenta de apariciones

Esto es más fácil de leer, considerando que las búsquedas son los métodos definidos anteriormente.

Ahora que ya tenemos los algoritmos medianamente explícitos, veamos las soluciones en JAVA:

```
public static int encontrar(String texto, String patron) {
    int n = -1;
    for (int i=0; i<texto.length(); i++) {
        n = i;
        for (int j=0; j<patron.length(); j++) {
            if (texto.charAt(i + j) != patron.charAt(j)) {
                n = -1;
                break;
            }
        }
        if (n != -1)
            break;
    }
    return n;
}

public static int encontrar(String texto, String patron, int comienzo) {
    int n = -1;
    for (int i=comienzo; i<texto.length(); i++) {
        n = i;
        for (int j=0; j<patron.length(); j++) {
            if (texto.charAt(i + j) != patron.charAt(j)) {
                n = -1;
                break;
            }
        }
        if (n != -1)
            break;
```

```
    }
    return n;
}

public static int cuentaApariciones(String texto, String patron) {
    int n = 0;
    int pos = encontrar(texto, patron);
    while (pos > -1) {
        n++;
        pos = encontrar(texto, patron, pos+1);
    }
    return n;
}
```

Esta sería la respuesta "correcta". Por otro lado, si somos algo más ingeniosos al momento de escribir los 3 métodos, podríamos llegar a la conclusión que estamos duplicando el proceso de búsqueda tanto para el encontrar desde el comienzo, como el encontrar desde un punto en particular (solución 1 y 2). Entonces, pensando cuál es más general, podemos decir que el encontrar desde el comienzo podemos escribirlo en función de la otra como:

```
public static int encontrar(String texto, String patron) {
    return encontrar(texto, patron, 0);
}
```

De esta forma estaríamos llamando en forma genérica a un solo encontrar. ¿Alguna ventaja? Pues es simple ver una, ya que si ponemos un patrón más largo que el texto o realizamos la siguiente búsqueda:

```
encontrar("alabado", "dominó");
```

Va a ocurrir un error. ¿por qué?, porque el for que busca el texto "dominó" dentro de "alabado" va a tratar de buscar más allá del final del texto. Así que si modificamos el método encontrar (con comienzo) modificaríamos ambos métodos al mismo tiempo. La modificación es:

```
public static int encontrar(String texto, String patron, int comienzo) {
    int n = -1;
    for (int i=comienzo; i<texto.length(); i++) {
        n = i;
        for (int j=0; j<patron.length(); j++) {
            if ((i+j) >= texto.length() ||
                    texto.charAt(i + j) != patron.charAt(j)) {
                n = -1;
                break;
            }
        }
        if (n != -1)
            break;
    }
    return n;
}
```

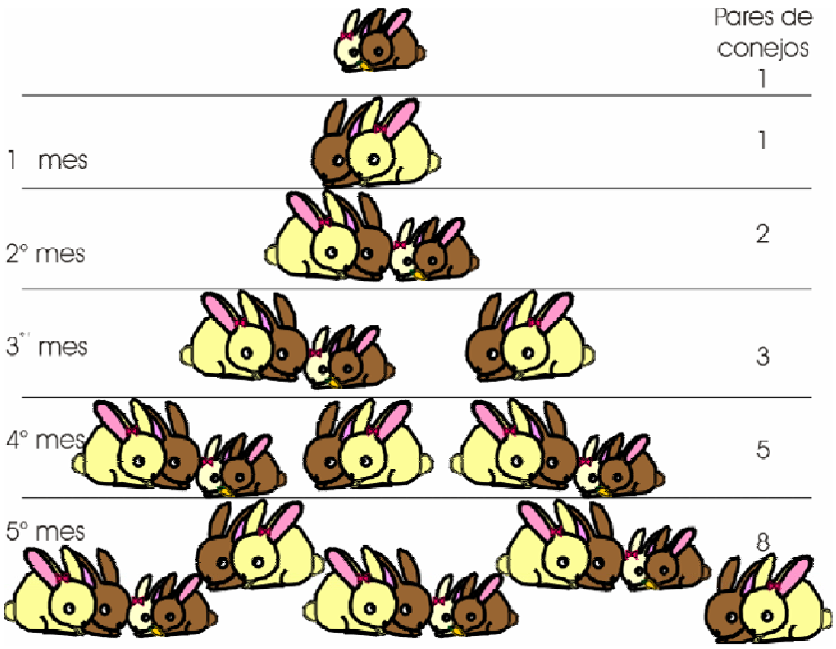
Con esta condición adicional se soluciona el problema de búsqueda más allá del final del texto. Por lo que la solución final completa quedaría:

```
public static int encontrar(String texto, String patron) {
    return encontrar(texto, patron, 0);
}

public static int encontrar(String texto, String patron, int comienzo) {
    int n = -1;
    for (int i=comienzo; i<texto.length(); i++) {
        n = i;
        for (int j=0; j<patron.length(); j++) {
            if ((i+j) >= texto.length() ||
                    texto.charAt(i + j) != patron.charAt(j)) {
                n = -1;
                break;
            }
        }
        if (n != -1)
            break;
    }
    return n;
}

public static int cuentaApariciones(String texto, String patron) {
    int n = 0;
    int pos = encontrar(texto, patron);
    while (pos > -1) {
        n++;
        pos = encontrar(texto, patron, pos+1);
    }
    return n;
}
```

# Números de Fibonacci

Leonardo de Pisa (1170-1250), mejor conocido como Fibonacci, que significa hijo de Bonaccio, fue uno de los más grandes matemáticos en la Europa de la edad media, Fibonacci creció en el norte de África, donde adquirió los conocimientos de las matemáticas avanzadas de los estudiosos árabes. En 1202 escribe Liber Abaci, el libro del ábaco, texto donde defiende el uso de los números arábigos, que usamos hoy en día, y explica como sumar, restar, multiplicar y dividir en este sistema, así como la resolución de otros tipos de problemas sobre álgebra y geometría, uno de los cuales es el siguiente:

El problema de los conejos. "En un patio cerrado, se coloca una pareja de conejos, recién nacidos, para ver cuántos descendientes produce en el curso de un año, y se supone que cada mes a partir del segundo mes de su vida, cada pareja de conejos da origen a una nueva."




Para resolver este problema se supondrá adicionalmente, que en este periodo de tiempo, ningún conejo muere y que la hembra siempre produce una nueva pareja formada por un macho, y una hembra. También y como la primer parea es recién nacida, y no se reproduce hasta el segundo mes de vida, al finalizar el primer mes tenemos sólo una pareja, pero al finalizar el segundo, se produce una nueva pareja (f(0) = f(1) = 1).

Con estas premisas y un análisis matemático se llega a la sucesión de Fibonacci:

$$f(n+2) = f(n+1) + f(n)$$

Escriba un programa que dada una cantidad de meses (por el usuario), pueda calcular cuántos conejos habrán al final del período.

## *Solución*

Este problema no es algo simple de resolver, pues la gracia está en su generalización. Si resumimos la fórmula inductiva de Fibonacci, nos quedaría:

$$f(n+2) = f(n+1) + f(n)$$
$$f(0) = f(1) = 1$$

Se ve rápidamente como es, así que veamos un "algoritmo" iterativo de solución para analizar el tema:

1. Escribir en pantalla "Ingrese la cantidad de meses a calcular?"
2. Pedir y guardar el número de meses
3. Si el número de meses es 0 o 1
   3.1. Escribir en pantalla "La cantidad de conejos es 1 pareja"
4. Si no se cumple
   4.1. Definir f1 y f2 como los f(n) y f(n+1) con valores igual a como dice la condición inicial
   4.2. Repetir mientras un contador sea menor que el número de meses
      4.2.1. Definir f = f1 + f2
      4.2.2. Asignar ahora f1 = f2
      4.2.3. Asignar ahora f2 = f
      4.2.4. Volver a 4.2
   4.3. Escribir en pantalla "La cantidad de conejos es " concatenado con f y " parejas"

¿Simple?. Quizás no mucho en palabras. En JAVA quedaría

```
Console c = new Console();
c.println("Ingrese la cantidad de meses a calcular?");
int meses = c.readInt();
if (meses <= 1 && meses >= 0) {
    c.println("La cantidad de conejos es 1 pareja");
}
else {
    int f1 = 1;
    int f2 = 1;
    for (int i=2; i<=meses; i++) {
        f = f1 + f2;
        f1 = f2;
        f2 = f
    }
    c.println("La cantidad de conejos es " + f + " parejas");
}
```

Parece simple la solución, pero de todas formas tienes que ir cambiando y corriendo los valores para f, f1 y f2, que a veces puede ser complejo desplazarlos. Hagamos el programa más simple dejándolo de la siguiente forma:

```
Console c = new Console();
c.println("Ingrese la cantidad de meses a calcular?");
```

```
int meses = c.readInt();
c.println("La cantidad de conejos es " + fibonacci(meses) + " parejas");
```

Con esta simpleza traspasamos el control a un método, el cual quedaría como sigue:

```
static public int fibonacci(int meses) {
    if (meses <= 1 && meses >= 0) {
        return 1;
    }
    else {
        int f1 = 1;
        int f2 = 1;
        for (int i=2; i<=meses; i++) {
            f = f1 + f2;
            f1 = f2;
            f2 = f
        }
        return f;
    }
}
```

Si lo miramos a simple vista el cambio es mínimo del programa, aunque funcionalmente no hay variaciones. Pero la gracia de esto es que podríamos además cambiar el método fibonacci a gusto, por ejemplo, si lo quisiéramos hacer en forma recursiva. Veamos primero, pensando en el método recursivo, un algoritmo representativo:

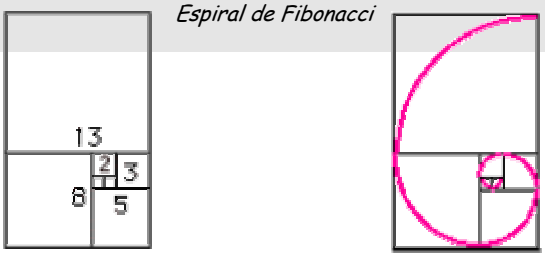
1. Si la cantidad de meses es 0 o 1
   1.1. Retornar 1
2. Retornar la suma del fibonacci del mes anterior y del fibonacci del mes anterior al anterior

Esto sería el algoritmo recursivo, ya que estaríamos llamando cada vez al método cambiando el número de meses a calcular. Escribamos eso en JAVA ahora:

```
static public int fibonacci(int meses) {
    if (meses <= 1 && meses >= 0) {
        return 1;
    }
    return fibonacci(meses-1) + fibonacci(meses-2);
}
```

Mucho más simple y más directo, casi sacado de la fórmula de la serie original.

*Espiral de Fibonacci*



## Calculadora de Notas

Los alumnos de la Universidad de Chile normalmente a fin de semestre o de año están preocupados por el examen, ya que no sabe qué nota pueden obtener en él para aprobar. Para ello se le ha pedido que realice un programa que le ayude al alumnado a tranquilizar sus ánimos y poder saber directamente qué nota puede sacarse con el siguiente diálogo:

```
Código del Curso? CC10A
Cuántos notas de controles tienes? 5
Ingresa la nota del control 1? 6.5
...
Ingresa la nota del control 5? 4.7
Tu promedio de presentación en CC10A es 5.3
Cuánto vale el examen (%)? 30%
En el examen necesitas una nota igual o superior a 2.2 para aprobar
```

Consideraciones:

- Las notas van entre 1.0 y 7.0
- No puede tener una nota inferior a 1.0 en el examen tampoco
- Si la nota que necesita es más de un 7.0, escribir "Estás reprobado"
- Ojo con las aproximaciones, ya que si el alumno necesita un 2.14 en el examen, no es lo mismo que un 2.1, ya que quedaría con nota inferior a 4.

### *Solución*

Veamos el algoritmo a seguir para resolver el problema.

1. Desplegamos en pantalla el mensaje: "Código del Curso? ".
2. Leemos y guardamos el código de curso ingresado.
3. Consultamos la cantidad de notas a ingresar: "Cuantas notas de controles tiene? ".
4. Leemos y guardamos la cantidad n ingresada.
5. Definimos la suma de las notas como cero.
6. Mientras no se reciban las n notas.
   6.1. Imprimir en pantalla: "Ingresa la nota del control" concatenada con el valor del contador.
   6.2. Leer y sumar la nota recibida a la suma de notas guardada.
   6.3. Volver al punto 6.
7. Calcular el promedio de las notas ingresadas.
8. Informar el promedio de presentación al examen : "Tu promedio de presentación en" concatenado con el código del curso y el promedio calculado.
9. Imprimir en pantalla : "Cuanto vale el examen ?".
10. Recibir y guardar la ponderación de la nota de examen.
11. Calcular la nota necesaria para aprovar.
12. Si la nota necesaria es menor a 1.0
    12.1. Nota=1.0
13. Si la nota es mayor a 7.0

13.1. Desplegar el mensaje: "Estas reprobado".
13.2. Terminar.
14. Desplegar el mensaje: "En el examen necesitas una nota igual o superior a " concatenado a la nota necesaria calculada.
15. Terminar.

Y ahora traspasemos nuestro algoritmo a JAVA:

```
class calculadora{
    public static void main(String arg[]){
            Console c=new Console();
            c.print("Codigo del Curso? ");
            String cod=c.readLine();
            c.print("Cuantas notas de controles tienes? ");
            int cant=Integer.parseInt(c.readLine());
```

Hasta acá hemos leído y guardado en variables los valores para el código del curso y para la cantidad de controles, en este último valor utilizamos la función parseInt para trasformar el string leído en un valor numérico entero.

```
            double suma=0;
            for(int i=0;i<cant;i++){
                    c.print("Ingresa la nota del control "+(i+1)+": ");
                    suma+=Double.parseDouble(c.readLine());
            }
            double prom=suma/cant;
            c.println("Tu promedio de presentacion en "+cod+" es "+prom);
```

Ahora hemos creado la variable suma de tipo double para sumar las notas ingresadas utilizando un ciclo for. Para pasar el string leído a tipo numérico decimal hemos utilizado la función parseDouble de la clase Double (esta es con mayúscula, no es lo mismo que el tipo de datos double) utilizándola de la misma forma que hacemos con parseInt.

Una vez terminada la recolección y suma de las notas ingresadas calculamos el promedio y desplegamos el resultado.

```
            c.print("Cuanto vale el examen (%)? ");
            String linea=c.readLine();
            int porcentaje=Integer.parseInt(linea.substring(0,linea.length()-1));
```

Obtenemos el valor de la ponderación de las notas y parseamos su valor a entero teniendo cuidado con quitar antes el símbolo % ingresado por el usuario.

```
double nota=(3.95-prom*(100-por)/100)*100/porcentaje;
```

Calculamos la nota necesaria para la aprobación del ramo resolviendo la ecuación:

$$39.5 = N_{examen} \cdot \left( \frac{Porcentaje}{100} \right) + N_{controles} \cdot \left( \frac{1 - Porcentaje}{100} \right)$$

Donde nuestra incógnita es la nota de examen.

```
            nota=Math.ceil(nota*10)/10;
            if(nota<1.0)
                  nota=1.0;
            if(nota>7.0)
                  c.println("Estas reprobado");
            else
                  c.println("En el examen necesitas una nota igual o superior a
"+nota+" para aprobar.");
    }
}
```

Por último, debemos hacer las aproximaciones necesarias para obtener la nota . Para esto debemos aproximar el valor obtenido hacia arriba dejando solo un decimal. Este tipo de aproximación lo logramos multiplicando la nota por 10, para no perder el primer decimal de la nota, luego aproximando hacia arriba con la función ceil de la librería Math de JAVA y por último dividimos por 10 para dar el decimal a la nota.

Solo resta dar la respuesta al usuario. En esta etapa aplicamos las reglas como que no puedes sacar menos de un 1.0 en el examen, por lo que si la nota es menor a 1.0 fijamos su valor en 1.0. Por otro lado si la nota necesaria es mayor a 7.0 desplegamos el mensaje "Estas reprobado" y en otro caso se imprime el mensaje indicando la nota necesaria para aprobar el curso.

## Reemplazador de Cadenas (Propuesto)

En un String, existe un método que se llama replace y que permite cambiar todas las apariciones de una subcadena por otra.

Por ejemplo:

```
Console c = new Console();
String texto = "mi mama me mima";
texto = text.replace("mi", "mu");
c.println(texto);
```

Este texto produce en pantalla la aparición de "mu mama me muma". No podemos hacer en forma independiente que el texto quede "mu mama me mima".

Escriba un método que permita reemplazar solo la cantidad de subcadenas que indique un parámetro, con el siguiente encabezado:

**public static String reemplaza(String texto, String patron, String reemplazo, int veces)**

Considerando que:

- El número de cadenas a reemplazar parte desde izquierda a derecha
- Si el número supera la cantidad de veces que aparece, es lo mismo que eliminarlas todas

## Reemplazador de Cadenas II

**a)** Escribir la función (método) de encabezado:

**static public String eliminar(String X,String Y,String Z)**

que elimine del string **X** todos los substrings comprendidos entre los delimitadores **Y** y **Z**. Por ejemplo:
**eliminar("a]b[cd]e[f]g[h","[","]")** entrega **"a]beg[h"**

En caso que el segundo delimitador sea un string vacío, se deben eliminar todos los caracteres a partir del primer delimitador.
Por ejemplo, **eliminar("ab--cd","--","")** entrega **"ab"**.

**b)** Utilice la función **eliminar** en un programa que lea líneas y las escriba eliminando los substrings de la forma **<caracteres>** y **%caracteres**.
Por ejemplo, **"ana <maria>rios%gonzalez"** debe escribirse **"ana rios"**. El fin de los datos se indica con una línea que contiene sólo </html>.

## *Solución*

**a)**

Algoritmo de solución:

1. Buscar ocurrencias del primer delimitador en el string a modificar.
2. Si el delimitador esta dentro del string
   - 2.1. Verificar la existencias del segundo delimitador.
   - 2.2. Si existe el segundo delimitador
     - 2.2.1. Buscar ocurrencias del segundo delimitador a partir de la posición en que se encontró el primero dentro del string a modificar.
     - 2.2.2. Si encontramos el delimitador
       - 2.2.2.1. Eliminar el substring contenido entre las posiciones en que se encontró a los delimitadores
       - 2.2.2.2. Volver a 1.
     - 2.2.3. Retornar el string
   - 2.3. Retornar el string comprendido entre la posición 0 y el índice en que se encontró el primer delimitador.
3. Retornar el string.

Como suele ocurrir al momento de programar existen diversas formas de interpretar un algoritmo. Para ejemplificar esto mostraremos dos versiones de este programa una recursiva y otra iterativa.

**Solución recursiva:**

En el caso de este problema el caso base de la recursión se da cuando existiendo el delimitador este no se encuentra dentro del string, entonces retornamos el string.

```
static public String eliminar(String X,String Y,String Z){
  int i = X.indexOf(Y);
  if( i < 0 )
      return X;
```

Como acordamos en nuestro algoritmo, buscamos ocurrencias del primer delimitador en el String, si no encontramos el delimitador caemos en un caso base de la recursión retornando el String.

```
if( Z.equals("") )
return X.substring(0,i);
```

Verificamos la existencia del segundo delimitador. En caso de no existir retornamos el String eliminando el substring desde la ultima ocurrencia del primer delimitador hasta el final.

```
int j = X.indexOf(Z,i+1);
if( j < 0 )
        return X;
```

En el caso de llegar a este punto del programa sabemos que si existe un segundo delimitador. Realizamos la búsqueda dentro del string y en caso de no encontrarlo caemos en otro caso base en que no hay nada que cambiar en el string, luego lo retornamos.

```
  return X.substring(0,i) + eliminar(X.substring(j+Z.length()),Y,Z);
}
```

En este punto estamos seguros de tener ubicados ambos delimitadores dentro del string por lo que solo nos resta eliminar el substring comprendido en el rango de los indices i y j. Utilizando la recursividad retornamos el substring valido (el que esta antes de la ocurrencia del primer delimitador) y llamamos a la funcion eliminar sobre el substring que aun no ha sido revisado (el que esta despues del indice del segundo delimitador).

**Solucion no recursiva:**

En esta solución el trabajo realizado por la recursión es asignado a un ciclo infinito que busca los delimitadores hasta llegar al caso base de la recursión retornando.

```
while(true){
int i = X.indexOf(Y);
if( i < 0 )
return X;
if( Z.equals("") )
return X.substring(0,i);
int j = X.indexOf(Z,i+1);
if( j < 0 )
return X;
X = X.substring(0,i) + X.substring(j+Z.length());
}
```

**b)**

Es claro que utilizando la funcion creada en la parte a) del problema la resolución de esta parte es bastante sencilla. Veamos nuestro algoritmo:

1. Leemos lineas ingresadas por el usuario.
2. Si la linea no es la indicadora de termino "</html>".
   2.1. Aplicamos la función eliminar sobre la linea leída utilizando como separadores los caracteres "<" y ">".
   2.2. Aplicamos la función eliminar usando como delimitador único el carácter "%".
   2.3. Imprimimos la linea modificada en pantalla
   2.4. Volvemos a 1.

Ahora veamos como se ve nuestro algoritmo en java.

```
while(true){
String L = C.readLine();
if(L.equals("</html>")) break;
L = eliminar(L,"<",">");
L = eliminar(L,"%","");
C.println(L);
}
```

Hemos usado un ciclo infinito en donde hacemos los cambios necesarios a la linea de entrada y teniendo como condición de quiebre que la linea sea igual al string "</html>".

## División Controlada

Al dividir dos números enteros positivos se puede obtener un resultado con una cantidad determinada de decimales. Al respecto, escriba un programa que siga el diálogo siguiente:

Ingrese el dividendo : 22
Ingrese el divisor : 7
Ingrese la cantidad de decimales del resultado : 20
Resultado = 3,14285714285714285714

**Nota**. Los dígitos de la parte decimal se obtienen sucesivamente del resultado de la división entera entre el resto o residuo anterior (multiplicado por 10) y el divisor. Para el ejemplo:

| Dividendo | 22 | 10 | 30 | 20 | 60 | 40 | 50 | 10 | 30 | 20 | 60 | 40 | 50 | 10 | 30 | 20 | 60 | 40 | 50 | 10 | 30 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Cuociente | **3** | **1** | **4** | **2** | **8** | **5** | **7** | **1** | **4** | **2** | **8** | **5** | **7** | **1** | **4** | **2** | **8** | **5** | **7** | **1** | **4** |
| Resto | 1 | 3 | 2 | 6 | 4 | 5 | 1 | 3 | 2 | 6 | 4 | 5 | 1 | 3 | 2 | 6 | 4 | 5 | 1 | 3 | 2 |

## *Solución:*

Formalicemos el algoritmo dado en el enunciado.

1. Desplegamos el mensaje: "Ingrese el dividendo: ".
2. Capturamos y guardamos el valor ingresado
3. Desplegamos el mensaje: "Ingrese el divisor: ".

4. Capturamos y guardamos su valor.
5. Desplegamos el mensaje: "Ingrese la cantidad de decimales del resultado: "
6. Capturamos y guardamos su valor.
7. Desplegamos el valor entero de la división
8. Mientras el contador sea menor a la cantidad de decimales deseada.
   8.1. Desplegamos el resultado entero de la división entre el resto de la división anterior, multiplicado por 10, y el divisor.
   8.2. Incrementamos el contador.

Ahora Traduzcamos nuestro algoritmo a java.

```
C.print("Ingrese el dividendo : ");
int a = C.readInt();
C.print("Ingrese el divisor : ");
int a = C.readInt();
C.print("Ingrese la cantidad de decimales del resultado : ");
int n = C.readInt();
C.print( "Resultado = " + a/b + ",");
```

Lo principal del programa es el manejo que se le da al resto. Durante la ejecución del ciclo se realizan divisiones en que solo cambia el divisor, mientras el dividendo pasa a ser en cada división el valor del resto anterior multiplicado por 10.

```
for(int i=1; i<=n; ++i)
{
        a = a % b * 10;
        C.print( a/b );
}
```

Nota: Al usar la asignación "a=a%b*10;" el lado derecho siempre se evalúa primero, luego en a estará el valor anterior del dividendo con el que se calcula el resto anterior.

## Números Pitagóricos

Tres números enteros positivos a, b y c se consideran pitagóricos si cumplen con la relación $a^2+b^2=c^2$. Por ejemplo 3, 4 y 5 son pitagóricos puesto que $3^2+4^2=5^2$. Nótese que si los números representan las longitudes de los lados de un triángulo, entonces forman un triángulo rectángulo de catetos a y b e hipotenusa c.
Al respecto, escriba un programa que lea el valor de la hipotenusa y busque los valores de los catetos del triángulo rectángulo correspondiente, siguiendo el diálogo indicado en el siguiente ejemplo:

```
Determinar catetos de triangulo rectángulo
Ingrese valor de hipotenusa: 10
Catetos = 6 8
```

En caso que no se puedan obtener valores enteros para los catetos debe responderse "Catetos no son enteros".

### *Solución:*

Lo primero es solicitar el valor de la hipotenusa al usuario.

```
class Pitagoricos{
static public void main(String args[]){
Console C=new Console();
C.println("Determinar catetos de triangulo rectangulo")
C.println("Ïngrese valor de hipotenusa");
int c= Integer.parseInt(C.readLine());
```

Ahora declaramos una variable que utilizaremos para saber si se encontraron los lados enteros que cumplen con la ecuación de pitagoras.

```
boolean sonEnteros = false;
```

Para encontrar los lados usaremos ciclos anidados que generaran todas las combinaciones de lados enteros entre 1 y el valor de la hipotenusa.

```
for(int a=1; a < c; ++a)
        for (int b=1;b < c; ++b)
```

Ahora que generamos pares (a,b), posibles candidatos, probamos si cumplen con la ecuación de pitagoras. En caso que cumplan imprimimops en pantalla los valores encontrados, usamos la variable sonEnteros como memoria y quebramos los ciclos.

```
if(a*a + b*b == c*c){
C.println("catetos="+ a + " y " + b);
sonEnteros = true;
break;
}
```

Por ultimo, debemos chequear la variable sonEnteros para saber si los catetos fueron encontrados y en caso negativo informamos al usuario.

```
if( sonEnteros == false)
C.println("catetos no son enteros");
}
}
```

## El Número Secreto

Para operar con un número secreto generado por el computador se dispone de la clase Secreto con los métodos indicados en la siguiente tabla:

| encabezamiento | significado | ejemplo |
|---|---|---|
| Secreto(int x,int y) | Constructor que calcula y guarda un N° secreto (entero entre x e y generado al azar) | Secreto s = new Secreto(10,20) |
| int comparar(int x) | -1 si N° secreto < x, 0 si N° secreto = x, 1 si N° secreto > x | s.comparar(15) |
| int menorMasCerca() | Entrega el N° menor más cercano que ha sido comparado con el N° secreto | s.menorMasCerca() |
| int mayorMasCerca() | Entrega el N° mayor más cercano que ha sido comparado con el N° secreto | s.mayorMasCerca() |

**a)** escribir un programa que utilice la clase anterior para implementar un juego en que el usuario tenga que adivinar un N° secreto generado por el computador, siguiendo las reglas y el diálogo indicado en el siguiente ejemplo:

```
adivine un N° X entre 1 y 100
X ? 27        (pregunta)
X < 27        (respuesta)
X ? 15
X  > 15
X ? 0         (solicitar ayuda)
15 < x < 27   (responder con menor y mayor más cercanos)
...
X ? 25
```

```
X = 25 Felicitaciones
```

**b)** Escriba las instrucciones del método comparar, considerando la siguiente declaración de la clase Secreto:

```
class Secreto {
  private int secreto, menor, mayor;
  public Secreto(int x,int y) { //constructor
    secreto = x + (int)(Math.random()*(y-x+1));
    menor = x - 1;
    mayor = y + 1;
  }
  public int menorMasCerca(){ return menor; }
  public int mayorMasCerca(){ return mayor; }
  public int comparar(int x){ . . .}
}
```

## *Solución:*

**a)**

El Algoritmo seria:

1. Generar un numero secreto.
2. Dar las instrucciones al usuario: "Adivine un No entreo 1 y 100"
3. Capturar los intentos del usuario.
   - 3.1. Si el usuario ingresa un "0"
     - 3.1.1. Deslegar la ayuda y vover a 3
   - 3.2. Comparar el número ingresado con el secreto e informar al usuario el resultado
   - 3.3. En caso de acierto terminar
   - 3.4. En otro caso volver a 3

Creemos ahora el codigo para nuestro algoritmo:

Damos las instrucciones al usario y creamos un objeto de la clase Secreto. Como vimos en la definición de la clase al crear este objeto el programa genera el número secreto al azar entre los rangos dados de parámetro.

```
C.println("Adivine un No entre 1 y 100")
Secreto s =new Secreto(1,100);
```

El siguiente paso es crear el ciclo del juego, este ciclo solo debe detenerse en el momento en que el usuario adivine el número secreto por lo que usamos un ciclo infinito que quebraremos en caso de acierto.

```
while(true){
C.print("No ?");
```

```
int n =Integer.parseInt(C.readLine());
```

Una vez capturado el número ingresado por el usuario tenemos dos posibilidades, puede tratarse de una solicitud de ayuda o efectivamente ser un intento de acierto. La solicitud de ayuda es notificada por el ingreso de un 0, en este caso debemos retornar los intentos mas cercanos al numero secreto utilizando las funciones provistas por la clase Secreto para ese fin.

```
if( n==0)
C.println(s.menorMasCerca()+ " < No < "+ s.mayorMasCerca());
```

Para el caso de un intento de acierto usamos la funcion comparar de la clase Secreto para obtener la relación entre el número ingresado y el secreto. El usuario es informado del resultado de su intento y en caso de acertar quebramos el ciclo.

```
if(s.comparar(n)==0) {
C.println("No = " + n + " Felicitaciones");
break;
}
if(s.comparar(n)<0)
C.println("No < " + n);
if(s.comparar(n)>0)
C.println("No > "+n);
}
```

**b)**

Para responder esta parte del problema debemos prestar especial atencion en como esta definida la clase, en sus variables de instancia y en la relación que hay entre los valores de estas y la invocación de una función.

En el caso de esta clase vemos que posee tres variables de instancia que almacenan el numero secreto, el menor mas cercano y el mayor mas cercano. El valor del número secreto vemos y es normal que se setee dentro del constructor, este valor no varia durante la existencia del objeto. Por otro lado las variables menor y mayor deben cambiar a medida que el usuario intenta adivinar el número secreto. Con respecto a esto, tenemos dos formas de notar que este cambio debe realizarce dentro de la función comparar, uno, para realizar los cambios debemos usar las comparaciones, y dos (la mas evidente de ambas), o queda otra funion no declarada que pueda hacerlo.

Tomadas todas estas directrices veamos como sería una versión de la funcion comparar.

```
public int comparar(int x){
        if(secreto == x)
          return 0;
        if(secreto < x){
          if(x<mayor) mayor =x;
        return -1;
        }
```

Comenzamos comparando el numero a comparar con nuestra variable que almacena el número secreto. Tenemos tres posibles resultados, igualdad en cuyo caso debemos retornar 0 sin otro cambio, número secreto mayor y número secreto menor. En estos últimos dos casos es cuando debemos recordar las variables de clase mayor y menor y cambiarlas en caso que el nuevo número supere en su cercanía a alguna de las variables.

```
if(secreto > x){
  if(x>menor) menor=x;
       return 1;
     }
}
```

## El Periodista

Un periodista que cuenta con el texto de un diario almacenado en un archivo (llamado diario.txt) quiere saber en cuantas líneas y en qué contexto esta citado algún personaje que le interesa (como por ejemplo, algún Salas o Ríos). Para eso se le pide a usted que escriba en programa que primero lea de la consola un string y luego imprima para cada línea donde aparece ese string lo siguiente: la línea anterior, la línea donde aparece y la línea posterior, cada una precedida por un número que indica el número de línea dentro del archivo. Ej. en la columna de la izquierda se ve el contenido del archivo y en la derecha el diálogo en la consola.

| | |
|---|---|
| En el último partido de la selección<br>Chilena en la cual la selección uruguaya<br>no clasificada para el mundial, empató a 2<br>goles, Zamorano, sin estar en uno de sus<br>mejores días hizo el primer gol de nuestra<br>selección, momento en el cual todo parecía<br>indicar que el resultado del partido favorecería<br>ampliamente a Chile. Recordemos que Zamorano<br>había perdido un penal en el partido anterior contra<br>Argentina, así que seguramente Zamorano se<br>Sentía obligado a hacer por lo menos un gol. | Ingrese palabra a buscar: Zamorano<br>3no clasificada para el mundial, empató a 2<br>4goles, Zamorano, sin estar en uno de sus<br>5mejores días hizo el primer gol de nuestra<br>7indicar que el resultado del partido favorecería<br>8ampliamente a Chile recordemos que Zamorano<br>9había perdido un penal en el partido anterior contra<br>9había perdido un penal en el partido anterior contra<br>10Argentina, así que seguramente Zamorano se<br>11 sentía obligado a hacer por lo menos un gol. |

Nota: Tenga presente que si el string se encuentra en la 1° o última línea entonces sólo deben mostrarse 2 líneas.

Indicación: Tome el chequeo de la primera y última línea como casos particulares. Para el resto use un ciclo.

## *Solución:*

Comenzamos definiendo las variables que necesitaremos, la consola para imprimir, el lector de archivo, un contador para guardar el numero de linea, un string para guardar la palabra buscada y tres strings para guardar las tres lineas de interes en la busqueda todas inicializadas con valor null.

```
import java.io.*;
public class Busqueda{
    public static void main(String args[]) throws IOException {
            Console c = new Console();
            BufferedReader A = new BufferedReader(new FileReader("diario.txt"));
            int contador=2;
            String palabra,lineant=null,lineact=null,linesig=null;
```

Ahora, solicitamos al usuario que ingrese la palabra a buscar e inicializamos la variable palabra.

```
c.print("Ingrese la palabra a buscar ");
palabra = c.readLine();
```

Inicializamos las variables de lineas. En caso que el archivo tenga menos de tres lineas, las variables sobrantes quedarán con valor null.

```
lineant=A.readLine();
lineact=A.readLine();
linesig=A.readLine();
```

Los casos particularres de este problema son las lineas de los extremos. Aca nos preocupamos de buscar en la primera linea del archivo, siempre teniendo la precaución que el archivo podría tener cero o solo una linea.

```
if(lineant!=null && lineant.indexOf(palabra)>=0){
        c.println("1 "+lineant);
        if(lineact!=null)
                c.println("2 "+lineact);
}
```

Ahora procesamos el resto del archivo. En cada iteración buscamos la palabra en la variable lineact, reasignamos los valores de las lineas y aumentamos el contador.

Como la condición del ciclo es sobre la linea actual, no tenemos control sobre la posibilidad de linesig==null por lo que debemos preocuparnos de consultar antes de imprimirla.

El caso de una ocurrencia de palabra en la ultima linea queda cubierto con laasignación hecha la final del ciclo, sea como sea en algún momento lineact será igual a la linea final del archivo.

```
while(lineact!=null){
        if(lineact.indexOf(palabra)>=0){
                c.println((contador-1)+" "+lineant);
                c.println(contador+" "+lineact);
                if(linesig!=null)
                        c.println((contador+1)+" "+linesig);
        }
        lineant=lineact;
        lineact=linesig;
        linesig=A.readLine();
        contador++;
}
```

Para terminar cerramos el lector de archivo, el metodo main y la clase.

```
            A.close();
    }
}
```

Para tener una mejor apreciación del programa se recomienda probar a mano con un archivo corto.

## Ayudando al Ayudante

El archivo notas.txt contendrá los nombres y las notas de los alumnos en este control, en la forma indicada en los siguientes ejemplos:

Patricia Fernandez, P1=40, P2=52
Juan Perez,P2=48
Gloria Hernandez, P2=51,P1=50
Jose Urrutia
Hector Gonzalez, P1=70

Observe que la información viene separada por comas y que el nombre vienesiempre en primer lugar.

La nota de cada pregunta del alumno está expresada en la forma "Px=nn", en que x es 1 o 2 y nn es un número entero entre 10 y 70 (la nota multiplicada por 10). Por ejemplo "P1=55" o "P2=30". Estas notas pueden aparecer en cualquier orden. Si la nota de una pregunta no aparece, entonces se considera con el valor 10.

Escriba un programa que lea el archivo notas.txt y escriba el archivo notas2.txt donde se normalicen los datos de los alumnos rellenando las notas faltantes y agregando el promedio de ambas. Cada línea debe contener nombre:xx.yy:zz; en que xx e yy representan las notas de las preguntas 1 y 2, y zz debe calcularse como el promedio redondeado de ambas. Para los ejemplos anteriores, el archivo producido debe ser:

Patricia Fernandez:40:52:46
Juan Perez:10:48:29
Gloria Hernandez:50:51:51
Jose Urrutia:10:10:10
Hector Gonzalez:70:10:40

## *Solución:*

Escribamos un algoritmo para guiarla programación:

1. Para cada linea del archivo notas.txt
   - 1.1. Recuperamos el nombre del alumno
   - 1.2. Buscamos las notas ingresadas
   - 1.3. Identificamos la nota P1 y P2
   - 1.4. Definimos las notas no ingresadas con 10
   - 1.5. Calculamos el promedio de notas
   - 1.6. Escribimos la estadistica en notas1.txt

Comenzamos el programa creando las variables que necesitaremos para el almacenamiento de los datos a obtener del archivo. Declaramos variables para la linea leida, los substrings extraidos, las notas parseadas y el lector y escritor de archivo.

```
import java.io.*;
public class Notas {
public static void main(String args[]) throws IOException{
        String linea, nombre, aux1 = null, aux2 = null;
        int n1 = 0, n2 = 0;
        BufferedReader R = new BufferedReader(new FileReader("notas.txt"));
        PrintWriter W = new PrintWriter(new FileWriter("notas2.txt"));
```

Comenzamos con el un ciclo para recorrer el archivo completo. Esta forma de ciclo es equivalente a poner la asignación, la comparación y la condición de quiebre separadas como lo hemos hecho en los ejercicios anteriores.

Lo primero que hacemos dentro del ciclo es verificar la existencia de alguna coma, esto nos indicaría que por lo menos existe un anota ingresada. El primer caso que analizamos es aquel en que la linea extraída del archivo no contiene comas por lo que se trata de solo el nombre del alumno y sus notas serán 10's.

```
        while((linea = R.readLine()) != null){
            if (linea.indexOf(",") < 0){
                nombre = linea;
                n1 = 10;
                n2 = 10;
            }
```

El siguente caso es en el que encontramos una coma, entonces debemos extraer el nombre del alumno utilizando el índice de la coma y luego eliminamos el nombre y la coma ya encontrada de la variable que contiene la linea en proceso. Con esto quedamos solamente trabajando con las notas que tiene ingresadas el alumno.

```
            else {
                nombre = linea.substring(0, linea.indexOf(","));
                linea = linea.substring(linea.indexOf(",") + 1);
```

Luego buscamos una segunda coma, con el fin de averiguar si el alumno tiene 1 o 2 notas ingresadas. Si no encontramos el delimitador (",") solo existe una nota del alumno que asignamos a la primera variable auxiliar marcando la segunda como nula, en caso contrario parseamos la linea y almacenamos el texto correspondiente a ambas notas en las correspondientes variables.

```
                if (linea.indexOf(",") < 0){
                  aux1 = linea;
                  aux2 = null;
                }else{
```

```
                    aux1 = linea.substring(0, linea.indexOf(","));
                    aux2 = linea.substring(aux1.length()+1);
                }
```

Una vez que ya tenemos los datos extraídos separados necesitamos identificar cada uno y calcular el promedio entre las notas.

El siguiente paso es recuperar las notas del alumno desde las variables auxiliares. Para esto debemos identificar el orden en que están ingresadas las notas, comenzamos suponiendo que la nota de la pregunta 1 se encuentra en aux1 extraemos su valor y luego extraemos la nota 2 de aux2 en caso que no sea nula, siempre cuidando que si una nota no esta se debe setear en 10.

```
                if (aux1.indexOf("P1") > 0){
                    n1 = Integer.parseInt(aux1.substring(aux1.indexOf("=")+1));
                    if (aux2 != null)
                        n2 = Integer.parseInt(aux2.substring(aux2.indexOf("=")+1));
                    else
                        n2 = 10;
                }
```

En caso que la nota de la pregunta 1 no estuviera en aux1, tenemos la posibilidad que sea la nota de la pregunta 2 la almacenada en aux1 y seguimos el mismo procedimiento que en el bloque anterior.

```
                else if (aux1.indexOf("P2") > 0){
                    n2 = Integer.parseInt(aux1.substring(aux1.indexOf("=")+1));
                        if (aux2 i= null)
                            n1 = Integer.parseInt(aux2.substring(aux2.indexOf("=")+1));
                        else
                            n1 = 10;
                    }
                }
```

Por ultimo solo nos resta escribir los datos obtenidos en el archivo de destino con el formato indicado y cerrar el lector y escritor de archivos.

```
                    W.println(nombre+":"+n1+":"+n2+":"+((n1+n2)/2));
                }
                R.close();
                W.close();
        }
}
```

## Dígito Verificador

Escriba un programa que utilice una función para calcular el dígito verificador del RUT ingresado por el usuario. El programa debe seguir un diálogo como el siguiente:

```
Ingrese su Rut sin dígito verificador:15780984
El dígito verificador es:9
Ingrese su Rut sin dígito verificador:9845215
El dígito verificador es:K
Ingrese su Rut sin dígito verificador:0
```

## *Solución:*

Este es un problema muy común en sistemas de identificación de usuarios, e incluso para autentificación de códigos de utilizan formulas como la usada para verificar la autenticidad del rut.

Veamos un ejemplo para aprender a calcular el digito verificador :

Rut= 16147325-x

Debemos empezar por calcular

5*2=10
2*3=6
9*4=36
7*5=35
4*6=24
1 *7=7
6*2=12
1 *3=3
---
total 133

la idea es multiplicar cada dígito del rut (invertido) por un factor entre 2 y 7 de manera ascendente como en el ejemplo y luego sumar.

Luego calculamos el resto de 133/11 que sería 1. Y obtenemos el dígito verificador como:

Dv=11-1=10

Y lo interpretamos como:

$$Dv = \begin{cases} n & si \quad n < 10 \\ 0 & si \quad n = 11 \\ k & si \quad n = 10 \end{cases}$$

Antes de ir al código veamos algunos trucos que nos serán útilies en el cálculo.

Obtención del ultimo dígito de un número:

```
int n=1419847;
int d=n%10;
c.println(d);//imprime "7"
```

Para recorrer todos los digitos del rut podemos usar el truco anterior y en cada iteración eliminar el dígito ya utilizado con la asignación:
n=n/10;

Dividiremos el programa entre una funcion que retorne el dígito verificador y un metodo para interactuar con el usuario.

Comenzamos la funcion dandoel encabezado y definiendo las variables que necesitaremos para el calculo.

```
public static int calcula_dv (int rut){
  int aux=rut;
  int digito;
  int factor=2;
  int suma=0;
  int dv;
```

Luego necesitamos el ciclo que recorra los dígitos del rut. Como usaremos la división entera para recorrer los dígitos del rut, nuestra condición de quiebre será cuando se acaben los dígitos (aux==0).

```
    while (aux!=0){
      digito = aux % 10;
      suma = suma + factor * digito;
      aux /= 10;
      factor++;
      if (factor==8)
        factor = 2;
    }
```

Utilizamos los trucos antes descritos para extraer el ultimo dígito y eliminarlo de la variable. Ademas tenemos un controlador para que el valor de factor vaya entre 2 y 7.
Lo siguiente es terminar la fórmula del dígito verificador y retornar.

```
    dv=11-suma%11;
    return dv;
  }
```

Ahora creemos la interacción con el usuario. Lo primero es crear las variables y consultar el rut.

```
public static void main (String args[]){
  Console c=new Console();
  int rut;

  c.print("Ingrese su rut sin digito verificador:");
  rut=c.readInt();
```

Ahora creamos el ciclo para que consulte mientras no ingrese "0". Dentro de este ciclo llamamosa la función de calculo e interpretamos su resultado informando al usuario.

```
  while (rut != 0){
    int dv=calcula_dv(rut);
    if (dv == 11)
      c.println("El digito verificador es 0");
    else
      if (dv==10)
        c.println("El digito verificador es K");
      else
        c.println("El digito verificador es "+dv);

    c.print("Ingrese su rut sin digito verificador:");
    rut = c.readInt();
  }
}
```

## El Método de Newton

El método de Newton para calcular una aproximación de la raíz cuadradode un número X se basa en el siguiente cálculo iterativo:

$$R_{i+1} = \frac{1}{2}\left( R_i + \frac{X}{R_i} \right)$$

En que $R_{i+1}$ es una mejor aproximación del valor de la raíz cuadrada de X, que se calcula usando la aproximación anterior $R_i$. Para iniciar las iteraciones se uas como primera aproximación el valor $R_0 = \frac{X}{2}$.

a) Escriba una función con el siguiente encabezado: **double raiz(double x,int i)** que retorne la aproximación de la raíz cuadrada del número X realizando i iteraciones.

b) Use la función anterior para escribir un programa que establezca el siguiente dialogo:

```
Ingrese el numero: 6
Valores aproximados de raiz cuadrada de 6
1   aprox1          %error
2   aprox2          %error
.
.
.
20  aprox20         %error
```

donde, aproxi= valor aproximadocon i iteraciones
%error= diferencia entre aprox y Math.sqrt, dividido por Math.sqrt.

## *Solución:*

a) Algoritmo:

1. Definir $R_0$ con el parametro recibino n, $R_0 = n/2$
2. Crear un ciclo de iiteraciones
   2.1. Recalcular la aproximación en base a la formula de newton
3. Retornar la aproximación.

El algoritmo no solo se ve simple , sino que es sencillo. Veamos el código:

```
public static double raiz(double n, int i) {
      double x = n/2;
      for (int j = 1; j <= i; j++)
        x = (x + n/x)/2;
      return x;
  }
```

Recordemos que en toda asignación siempre se evalua primero el lado derecho, es por esto que funciona nuestra función.

b) Algoritmo

1. Preguntamos al usario : "ingrese el numero".
2. Capturamos el numero ingresado n.
3. Creamos un ciclo de 20 iteraciones, con contador i
   3.1. Obtenemos la aproximación de n con i iteraciones.
   3.2. Calculamos el error
   3.3. Informamos el resultado.

Veamos ahora el código:

```
  public static void main(String[] args) {
     Console c = new Console();
     c.println("ingrese el numero ");
     int n = Double.parseDouble(c.readLine());
     for (int i = 1; i <= 20; i++) {
        double aprox = raiz(n,i);
        double error = Math.abs((aprox-Math.sqrt(n))/Math.sqrt(n)*100.0);
        System.out.println(i+"  "+aprox+"  "+error+"%");
     }
  }
```

La división por 100 en la formula del error es para obtener un error porcentual.

## Mundial de Futbol

Un canal de televisión que va a transmitir los partidos del mundial necesita disponer de la siguiente tabla que muestra los promedios de edad de los jugadores de los países participantes.

```
N° del país País(en orden alfabético) Promedio de edad(en años)
1 Alemania xx
... ... ...
32 Zaire xx
```

Escriba un programa que muestre la tabla anterior en la pantalla, obteniendo la información desde los siguientes archivos:

· "paises.txt": 32 líneas ordenadas alfabéticamente, cada una con el nombre de un país.

· "jugadores.txt": cantidad indeterminada de líneas, ordenadas alfabéticamente, cada una con la siguiente información:

- Columnas 1 a 20: nombre del jugador
- Columnas 21 a 28: fecha de naciemiento (en la forma DD/MM/AA)
- Columnas 29 a 30: números del país(entero entre 1 y 32, por ejemplo 1= Alemania, ..., 32= Zaire).

## *Solución:*

Para resolver este problema utilizando solo estructuras sencillas tendremos que leer el archivo de jugadores tantas veces como paises hay en paises.txt. Veamos el algoritmo:

1. Leemos cada linea de paises.txt
   1.1. Recuperamos el nombre del pais. Con un contador llevamos el numero del pais.
   1.2. Leemos cada linea de jugadores.txt
      1.2.1. Rescatamos su fecha de nacimiento
      1.2.2. Parseamos su año de nacimiento
      1.2.3. Capturamos el numero del pais a que pertenece
      1.2.4. Si el numero del pais coincide con el del pais en piases.txt
         1.2.4.1. Calculamos su edad y la sumamos a nuestra variable acumuladora
         1.2.4.2. Aumentamos el numero de jugadores del pais encontrados
   1.3. Imprimimos la informacion estadistica del pais
   1.4. Reiniciamos el lector de jugadores (Para que se situe en la primera linea)
2. Cerramos los lectores.

Vamos al codigo. Comenzamos creandolas variables del programa: lectores,strings para las lineas de cada archivo,string para el nombre del pais, string para la fecha de nacimiento, int contador para el numero de pais, acumuladores de edades y demases.

```
import java.io.*;
public class Edades{
public static void main(String args[]) throws IOException{
    Console C = new Console();
    BufferedReader P = new BufferedReader(new FileReader("paises.txt"));
    BufferedReader J = new BufferedReader(new FileReader("jugadores.txt"));
    String linea1,linea2,nompais,aux;
    int pais=0, ano=0, edad=0, suma=0, numero=0;
```

Imprimimos el titulo de la tabla.

```
    C.println("No. del pais Pais (en orden alfabetico) Promedio de edad (en años)");
```

Para cada linea del archivo de paises, capturamos el nombre del pais y aumentamos el contador de paises para usarlo como su numero.

```
    while((linea1=P.readLine())!=null){
            nompais=linea1;
            pais++;
```

Para cada linea de jugadores.txt, capturamos la fecha y parseamos su ano de nacimiento y el numero del pais al que pertenece.

```
            while((linea2 = J.readLine()) != null){
                    aux = linea2.substring(20, 28);
                    ano = Integer.parseInt(aux.substring(6));
                    aux = Integer.parseInt(linea2.substring(28));
```

Comparamos el numero de pais del jugador con el del pais que estamos leyendo de paises.txt. Si coinciden calculamos su edad la sumamos a suma y aumentamos el numero de jugadores encontrados del pais, esto para calcular el promedio de edad del equipo.

```
                    if(aux==pais){
                            edad = (104-ano);
                            suma+= edad;
                            numero++;
                    }
```

Cuando hemos recorrido todo el archivo de jugadores estamos listos para imprimir las estadisticas, seteamos la acumulacion de edades en 0 y reiniciamos el lector de jugadores.txt para volver  a leer.

```
            }//segundo while
            C.println((pais) + " " + nompais+ " " +(suma/ Math.max(numero, 1)));
            suma=0;
            J = new BufferedReader(new FileReader("jugadores.txt"));
    }//primer while
```

Una vez terminado el archivo de paises solo nos queda cerrar los lectores.

```
    P.close();
    J.close();
}
}
```

## El Juego de las Piedras (Propuesto)

El juego de las piedras consiste en que dos jugadores contrincantes extraen de un montón de piedras hasta acabar con él . Pierde el jugador que extrae la última piedra. La regla es que un jugador sólo puede extraer 1, 2 o 3 piedras cada vez. El montón tiene inicialmente entre 10 y 40 piedras.

Escriba un programa que arbitre el juego de las piedras. Inicialmente debe generar aleatoriamente el número de piedras con la función random y mostrar el tamaño del montón colocando un * por cuantas piedras saca, mostrando a continuación en una línea de la pantalla el tamaño del nuevo montón. El programa debe señalar quién es el vencedor cuando se extrae la última piedra.

Un ejemplo del diálogo que debe mantener el programa con los jugadores es el siguiente:

```
**********
Cuantas piedras saca el jugador 1? 3
********
Cuantas piedras saca el jugador 2? 1
*******
Cuantas piedras saca el jugador 1? 6
No puede sacar más de 3 piedras.
Cuantas piedras saca el jugador 1? 3
****
Cuantas piedras saca el jugador 2? 3
*
Cuantas piedras saca el jugador 1? 0
Tiene que sacar al menos 1 piedra.
Cuantas piedras saca el jugador 1? 1

El ganador es el jugador 2. Felicitaciones!
```

Durante el juego, su programa debe verificar:

- Que un jugador no saque más de 3 piedras.
- Que un jugador saque al menos una piedra.
- Que un jugador no saque más piedras de las que hay.

## Conjuntos Algebraicos (Propuesto)

La siguiente tabla define los métodos ofrecidos por la clase Conjunto que permite realizar operaciones con conjuntos de caracteres:

| **Ejemplo** | **Significado** | **encabezamiento** |
|---|---|---|
| a.union(b) | a U b | Conjunto union(Conjunto x) |
| a.inter(b) | a intersección b | Conjunto inter(Conjunto x) |
| a.igual(b) | ¿ a = = b ? | boolean igual(Conjunto x) |
| a.contiene(b) | ¿ a contiene a b? | boolean contiene(Conjunto x) |
| a.cardinal() | N° de elementos | int cardinal() |
| a.toString() | String con caracteres de a | String toString() |
| new Conjunto() | {} (conjunto vacío) | Conjunto() |
| New Conjunto("casaca") | {'c','a','s'} (conjunto con caracteres del String) | Conjunto(String x) |

**a)** Implemente la clase Conjunto, considerando la siguiente declaración de la clase:

```
Class Conjunto{
  String elementos; // string con caracteres que corresponden a elementos del conjunto
  .
  .
  .
}
```

b) Usando la clase Conjunto escriba un programa que genere estadísticas sobre caracteres en un archivo. El programa debe seguir el siguiente dialogo:

```
Nombre del archivo ? _____________________
Estadísticas de caracteres utilizados
Tipo          Cantidad      Caracteres
vocales          3          Aei
consonantes     17          CFHLRSTWZbdfghjkz
números          5          03589
```

## El Examen (Propuesto)

El profesor del ramo de Computación se dispone a tomar el examen oral a los alumnos. Los alumnos serán llamados uno por uno para que ingresen a la sala a dar su interrogación. El profesor posee la lista del curso en un archivo llamado "lista.txt". En cada línea de este archivo aparece el nombre completo de un alumno en el formato:

APELLIDO1,APELLIDO2,NOMBRE1,NOMBRE2

El archivo está ordenado alfabéticamente por APELLIDO1.
El profesor no quiere llamar a los alumnos en ningún orden específico para no perjudicar a nadie por su posición en la lista del curso.
Es por eso que le pide a usted que escriba un programa que tome este archivo y genere otro con la misma lista del curso pero en un orden aleatorio. Además, le dice que para llamar a los alumnos a que entren a la sala sólo necesita el primer nombre y el primer apellido de cada alumno y que es indispensable que al principio de cada línea aparezca el número del alumno (el que tenía en el archivo ordenado) para poder así buscar más fácilmente en la lista al momento de registrar la nota.
Es decir, cada línea debe tener el formato:

Número. NOMRE1 APELLIDO1

Luego de tomar el examen a todos los alumnos, el profesor le dice que le gustó tener la lista de los alumnos con sólo el primer apellido, el primer nombre y el número de cada uno de ellos, pero que le gustaría aún más que ahora estuviera ordenada.
Su misión es escribir un programa que genere los dos archivos: la lista desordenada (lista1.txt) y, a partir de ella, la lista nuevamente ordenada (lista2.txt), cuyas líneas deben tener el mismo formato que las de la lista desordenada.

Nota: Puede suponer que todos los alumnos tienen dos apellidos y dos nombres.