Guía de Estudio

Control 2

Profesor: Andrés Muñoz
Auxiliares: Agustín Almonte
Marcelo Muñoz
Patricio Salles

CC10A - 2004

Tabla de Contenidos

1. El Ajedrez (Pregunta 1 Control 2, 2002)

Para mantener una pieza de ajedrez en el tablero de 8 filas y 8 columnas, se dispone de la clase:

```
class Pieza{
  //private puede cambiarse por protected u omitirse
  private int fila, columna;

  public Pieza(int x,int y){fila=x; columna=y;}
  public boolean mover(int x,int y){fila=x; columna=y; return true;}
  public int obtenerFila(){return fila;}
  public int obtenerColumna(){return columna;}
}
```

(a) Escriba las clases **Caballo** y **Torre** que extiendan la clase **Pieza** con los métodos:

Ejemplo	Significado
C=new Caballo(1,1)	constructor que ubica un caballo en fila 1 y columna 1
C.mover(2,3) C.mover(1,2)	Mueve C a fila 2 y col. 3, y entrega true (movimiento válido) Entrega false y no mueve el caballo (movimiento inválido). **Nota.** Un movimiento de un caballo es válido si avanza 2 filas y 1 columna o 2 columnas y 1 fila dentro del tablero.
T = new Torre(5,6)	constructor que ubica una torre en fila 5 y columna 6
T.mover(5,8) T.mover(5,9)	Mueve T a fila 5 y col 8, y entrega true (movimiento válido). Entrega false y no mueve la torre (movimiento inválido). **Nota.** Es válido si avanza en la misma fila o columna dentro del tablero.

(b) Escriba el método **boolean comer(Pieza X,Pieza Y)** que devuelva true si la Pieza **X** "come" en una jugada a la Pieza **Y**, considerando que no hay más piezas en el tablero. Por ejemplo, si **Caballo C=new Caballo(1,2)** y **Torre T=new Torre(3,1)**, entonces **comer(C,T)** devuelve true y **comer(T,C)** devuelve false.

Solución

Versión 1:
Parte a:

```
public class Caballo extends Pieza {
	public Caballo(int x, int y) {
		super(x, y);
	}

	public boolean mover(int nx, int ny) {
		int x = super.obtenerFila();
		int y = super.obtenerColumna();

		// Debe moverse en ambos ejes (x e y <> nx e ny)
		if (nx == x || ny == y) return false;
```

```
                // Debe moverse 3 casillas
                if (Math.abs(nx-x) + Math.abs(ny-y) != 3) return false;

                // Se mueve el caballo
                return super.mover(nx, ny);
        }
}

public class Torre extends Pieza {
        public Torre(int x, int y) {
                super(x, y);
        }

        public boolean mover(int nx, int ny) {
                int x = super.obtenerFila();
                int y = super.obtenerColumna();

                // Debe moverse por solo un eje
                if (nx != x && ny != y) return false;

                // Se mueve el caballo
                return super.mover(nx, ny);
        }
}
```

Parte b:

```
public class MiPrograma {
        public boolean comer(Pieza X, Pieza Y) {
                // Obtenemos la posición de destino de la pieza
                int dx = Y.obtenerFila();
                int dy = Y.obtenerColumna();

                // Verificamos si se puede mover a esa casilla
                boolean come = true;
                if (X.instanceOf() == Caballo)
                        come = ((Caballo) X).mover(dx, dy);
                else
                        come = ((Torre) X).mover(dx, dy);

                // Retornamos lo que ocurrió
                return come;
        }
}
```

Versión 2:

Parte a:

```
class Caballo extends Pieza {
public Caballo(int x,int y){
      super(x,y);
}

public boolean mover(int x,int y){
  if( (1<=x && x<=8) && (1<=y && y <=8)
       &&(Math.abs(x-obtenerFila())==2 &&
       Math.abs(y-obtenerColumna())==1) ||
       (Math.abs(x-obtenerFila())==1 &&
```

```
       Math.abs(y-obtenerColumna())==2)))
    return super.mover(x,y);
  else
    return false;
}

class Torre extends Pieza {
public Torre(int x,int y){
      super(x,y);
}

public boolean mover(int x,int y){
  if( (1<=x && x<=8) && (1<=y && y <=8)
       && (( x == obtenerFila() && y != obtenerColumna() )
       ||( x != obtenerFila() &&
       y == obtenerColumna() ) )
    return super.mover(x,y);
  else
    return false;
}
```

Parte b:

```
boolean comer(Pieza X, Pieza Y){
  int i = Y.obtenerFila();
  int j = Y.obtenerColumna();
  return X.mover(i,j);
}
```

2. Morse (Pregunta 2 Control 2, 2002)

La siguiente figura muestra la interfaz de un Applet o Frame que permite ingresar una palabra en código morse en la forma indicada en el siguiente ejemplo en que se ingresa la palabra OLA.

Button	punto	raya	Button
Button	espacio	ingresar otra palabra	Button
Label	Palabra en morse:	-.-.- .-.. .-	Label
Label	Palabra en castellano:	OLA	Label

Escriba la clase que controle la interfaz anterior de acuerdo a las siguientes reglas:

- un click en el botón "punto" o "raya" debe ingresar un carácter . o un carácter -
- un click en el botón "espacio" indica fin del código de una letra
- un click en el botón "ingresar otra palabra" restablece la situación inicial

Nota: Suponga que existe el método Morse que devuelve la letra correspondiente a un código. Por ejemplo, Morse(".-..") devuelve "L".

Solución

Para solucionar este problema tenemos dos versiones, una en que la clase principal implemente el Listener y otra en que la clase principal contenga una clase listener.

Versión 1:

Como debemos hacer un frame hacemos que la clase principal sea nuestro frame extendiendo de la clase Frame, con esto nuestra clase principal heredara todos los metodos de la clase Frame y podremos agregar componentes directamente sin crear un objeto Frame aparte.

```
class Traductor extends Frame implements ActionListener {
```

Otra forma de comenzar es no heredando la clase principal de frame, si hacemos esto debemos crear un objeto de tipo Frame para contener los componentes (botones, etiquetas y campos de texto) y utilizar el nombre de este objeto para invocar los metodos de Frame como add, pack y show.

```
class Traductor implements ActionListener {
      private Frame f=new Frame();
```

A continuacion definimos los components que utilizaremos en nuestra interfaz gráfica. Como vemos en el enunciado del problema necesitaremos 4 botones y 4 labels, y les daremos los parámetros correspondientes para obtener la interfaz solicitada.

```
private Label
  L1 = new Label("Palabra en morse:"),
  L2 = new Label(""),
  L3 = new Label("Palabra en Castellano:"),
  L4 = new Label("");
private Button
  punto = new Button("punto"),
  raya = new Button("raya"),
  espacio = new Button("espacio"),
  otra = new Button("ingresar otra palabra");
```

Para procesar la información que desplegaremos en la interfaz necesitaremos ademas de las componentes gráficas dos variables de tipo string para guardar el codigo morse ingresado hasta el momento y su traducción.

```
private String
  codigo="",
  palabra="";
```

Una vez que tenemos todas las componentes gráficas y variables de la clase definidas, creamos un constructor que se encargara de posicionar las componentes dentro del frame y asociar los Listeners de acciones. Las acciones escuchadas serán las provenientes de cada uno de los cuatro botones de la interfaz.

```
public Traductor(){
  this.setLayout(new GridLayout(4,2));

  this.add(punto);
  this.add(raya);
  this.add(espacio);
  this.add(otra);
  this.add(L1);
  this.add(L2);
  this.add(L3);
  this.add(L4);

  punto.addActionListener(this);
  raya.addActionListener(this);
  espacio.addActionListener(this);
  otra.addActionListener(this);
}
```

Como nuestra clase implementa el listener ActionListener estamos obligados a definir el metodo actionPerformed en el que definiremos la accion que realizaremos frente a cada evento definido. Utilizando el metodo getSource() de la clase ActionEvent obtenemos la fuente de donde proviene la accion capturada por el listener, con esto podemos diferenciar las modificaciones que haremos en la interfaz dependiendo de cual boton fue oprimido.

```
public void actionPerformed(ActionEvent x){
```

Si la accion proviene del boton punto, debemos agregar el simbolo "." Dentro de la variable destinada a contener el codigo morse.

```
  if(x.getSource() == punto)
      codigo += ".";
```

Si la accion proviene del boton raya, debemos agregar el símbolo "-" a la variable destinada a almacenar el codigo morse actual.

```
  else if(x.getSource() == raya)
      codigo += "-";
```

Si la accion proviene del boton espacio, significa que se ha terminado de ingresar la palabra y debemos traducir el codigo morse almacenado, guardar la traducción en la variable de traducción y vaciar la variable que guarda el codigo morse.

```
  else if(x.getSource()==espacio){
      palabra += Morse(codigo);
      codigo = "";
  }
```

Si la accion no proviene de ninguno de los botones anteriores, entonces significa que la fuente del evento capturado debe provenir del boton otra. En este caso debemos vaciar las variables de almacenamiento para devolver la interfaz a su estado inicial.

```
  else {
      codigo="";
      palabra="";
  }
```

Por cada evento recibido actualizamos los valores desplegados por los labels.

```
  L2.setText(L2.getText()+codigo);
  L4.setText(palabra);
}
```

Por ultimo lo unico que falta es un metodo principal (main) para ejecutar la clase. Dentro de este metodo creamos un objeto de nuestra clase y lo desplegamos.

```
static public void main(String[]args){
  Traductor r=new Traductor();
  r.pack();
  r.show();
}
```

En caso de haber definido la clase Traductor no extendiéndola de frame, debemos llamar a los metodos pack y show dentro del constructor de la clase utilizando la variable f de tipo Frame.

```
}
```

Versión 2:

Para la versión 2 los que cambiaremos de nuestra clase Traductor será la implementacionde la clase ActionListener, esta vez el listener será una clase interna a la clase principal. Los cambios no son muchos por lo que mostraremos solo los cambios.

```
class Traductor extends Frame implements ActionListener {
.
.
.
public Traductor(){
.
.
.
  punto.addActionListener(new escucha());
  raya.addActionListener(new escucha());
  espacio.addActionListener(new escucha());
  otra.addActionListener(new escucha());
}
.
.
.
class escucha implements ActionListener{
       public void actionPerformed(ActionEvent x){
       .
       .
       .
       }
}
.
.
.
}
```

3. Censo (Pregunta 3 Control 2, 2002)

Los resultados del último censo de población están grabados en el archivo "censo.txt". Cada línea contiene el nombre de una comuna (columnas 1 a 20) y su cantidad de habitantes (columnas 21 a 26).

Escribir un programa que muestre la siguiente tabla de resultados:

Habitantes	Cantidad comunas	% de comunas
0 – 9999	Nº	xx.x
10000 – 19999	Nº	xx.x
. . .	. . .	. . .
90000 – 99999	Nº	xx.x
100000 – 119999	Nº	xx.x
120000 – 139999	Nº	xx.x
. . .	. . .	. . .
180000 – 199999	Nº	xx.x
200000 y más	Nº	xx.x
Total	Nº	100.0

Solución

Usaremos un arreglo para hacer la tabla que almacenará rango de habitantes y cantidades de comunas en ese rango.

Versión 1:

```
Console c = new Console();

// Inicializamos el arreglo
int rangos[] = new int[21];
for (int i=0; i<21; i++) {
      rangos[i] = 0;
}

// Procesamos el archivo
BufferedReader b = new BufferedReader(new FileReader("censo.txt"));
int nDatos = 0;
int nHab = 0;
while(true) {
      String linea = b.readLine();
      int hab = Intger.parseInt(linea.substring(20));
      if (hab < 200000) {
            rango[hab / 10000]++;
      }
      else {
            rango[20]++;
      }
      nHab += hab;
      nDatos++;
}
b.close();
```

```
// Desplegamos tabla
c.println("Habitantes\tCantidad de Comunas\t% Comunas");
for (int i=0; i<10; i++) {
      c.print((i*10000) + "-" + ((i+1)*10000-1) + "\t");
      c.print(rango[i] + "\t");
      c.println((100.0*rango[i]/nDatos)+"%");
}
for (int i=10; i<20; i+=2) {
      c.print((i*10000) + "-" + ((i+2)*10000-1) + "\t");
      c.print(rango[i]+rango[i+1] + "\t");
      c.println((100.0*(rango[i]+rango[i+1])/nDatos)+"%");
}
c.print("200000 y más\t");
c.print(rango[20]+ "\t");
c.println((100.0*(rango[20])/nDatos)+"%");

// Imprime el total
c.print("Total\t");
c.print(nDatos + "\t");
c.println("100.0%");
```

Versión 2:

```
int[] limite = {10000,20000,30000,40000,50000,60000,70000,
      80000,90000,100000,120000,140000,160000,180000,200000,
      1000000};
Console c=new Console();
final int N=16;
int[]frecuencia = new int[N];
for(int i=0; i<N; ++i)
      frecuencia[i] = 0;
int total = 0;

BufferedReader A = new BufferedReader(new FileReader("censo.txt"));
while(true){
      String linea = A.readLine();
      if( linea == null ) break;

      int h = Integer.parseInt(linea.substring(20,25));

      for(int i=0, i<N, ++i)
            if(h < limite[i] ){
                  ++frecuencia[i];
                  break;
            }

      ++total;
}

c.println("Habitantes\tcomunas\t%");
for(int i=0; i<N; ++i){
      c.println(
            ((i==0)?0:limite[i-1] )  + "-" +
            (i==N-1)?"y más":(limite[i]-1)    +  "\t" +
            frecuencia[i]    + "\t" +
            100.0*frecuencia[i]/total);
}
c.println("total\t" + total +"\t100.0");
```

4. Ternas de Valores (Pregunta 1 Control 2, 2001)

La siguiente clase permite asignar y recuperar los valores de una terna de números enteros:

```
class Terna {
  protected int a,b,c;
  public Terna( ){ a=0; b=0; c=0; }
  public void set1(int x){a=x;}
  public void set2(int x){b=x;}
  public void set3(int x){c=x;}
  public int get1(){return a;}
  public int get2(){return b;}
  public int get3(){return c;}
}
```

a) Escriba la clase Triangulo que extienda la clase Terna agregando los siguientes métodos:

Ejemplo	Resultado	Significado
Triangulo()	-	constructor que inicializa con ceros los valores de los tres lados
T.esTriangulo()	boolean	true si T corresponde a un triangulo o false si no (3 números positivos forman un triángulo si todas las sumas de 2 de ellos son mayores que el 3º)
T.ladosIguales()	int	cantidad de lados iguales (0, 2 o 3) del triángulo T
T.graficar(x)	void	grafica el triángulo T de color x (String "rojo","azul"o"verde")

Nota. NO debe escribir el método graficar

b) Escriba un programa que dibuje 100 triángulos (de lados enteros entre 1 y 100 generados al azar) de modo que los equiláteros (tres lados iguales) se dibujen de color rojo, los isósceles (dos lados iguales) de color azul, y los escalenos (todos los lados distintos) de color verde.

Nota. Recuerde que Math.random() devuelve un Nº real de tipo double en el rango [0,1[

Solución

Parte a:

Comenzamos con la definicion de la clase Triangulo que debe extender de Terna.

```
class Triangulo extends Terna{
```

Luego para construir un objeto de tipo Triangulo creamos un constructor para la clase que utiliza el constructor de la clase padre Terna utilizando el metodo super().

```
        public Triangulo(){super();}
```

Ahora continuamos con los metodos solicitados. Para el metodo esTriangulo verificamos que se cumplan las reglas definidas en el enunciado utilizando una clausula if y en el caso contrario retornamos false.

```
        public boolean esTriangulo(){
                if(a+b>c && a+c>b && c+b>a)
                        return true;
                return false;
        }
```

En el metodo ladosIguales debemos comenzar preguntando por el caso extremo en que todos los lados son iguales e ir disminuyendo desde ahí, asi preguntamos primero por tres lados iguales luego por dos y terminamos con el caso por defecto en que no hay lados iguales.

```
        public int ladosIguales(){
                if(a==b && b==c)
                        return 3;
                if(a==b || b==c || c==a)
                        return 2;
                return 0;
        }
}
```

Parte b:

Para crear 100 triangulos debe ser claro la utilización de un ciclo, en nuestro caso utilizaremos un ciclo de tipo for.

```
        for(int i=0;i<100;i++){
```

Dentro de cada ciclo crearemos un triangulo que dado el constructor de la clase comenzara teniendo sus lados con el valor 0.

```
                Triangulo t=new Triangulo();
```

Luego setearemos el valor de sus lados utilizando la funcion Math.random() para generar un nuemro entre 1 y 100.

```
                t.set1((int)Math.floor(Math.random()*100+1));
                t.set2((int)Math.floor(Math.random()*100+1));
                t.set2((int)Math.floor(Math.random()*100+1));
```

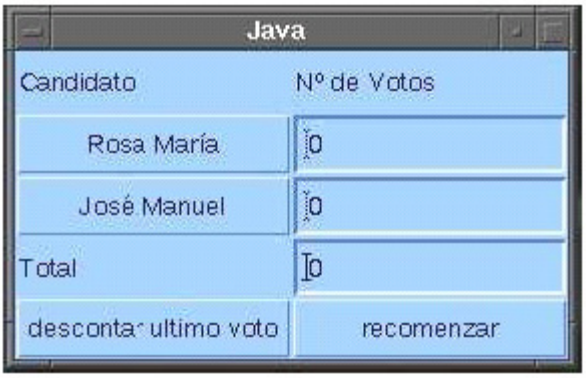
Una vez definidos los valores de los lados del triangulo nada nos garantiza que los lados generados al azar puedan formar un triangulo por lo que debemos consultar utilizando la funcion esTriangulo(). Si los lados pueden formar un triangulo, entonces consultamos la cantidad de lados iguales con la funcion ladosIguales() y dependiendo del valor obtenido graficamos el triangulo con el color correspondiente. Si los lados no pueden ser un triangulo

entonces no debemos contar este triangulo generado dentro de los 100 solicitados por lo que disminuimos el contador del ciclo for para volver a crearlo.

```
            if(t.esTriangulo){
                  if(t.ladosIguales()==3)
                        t.graficar("rojo");
                  else if(t.ladosIguales()==2)
                        t.graficar("azul");
                  else if(t.ladosIguales()==0)
                        t.graficar("verde");
            }
            else i--;
      }
```

5. Votaciones (Pregunta 2 Control 2, 2001)

La siguiente figura muestra la interfaz de un programa (Frame) que permite apoyar el recuento de votos de una elección con dos candidatos ("Rosa María" y "José Manuel"):



Escriba la clase que controle la interfaz anterior de acuerdo a las siguientes reglas:

- Cada vez que se da un click en el botón de un candidato se debe incrementar en uno el Nº de votos del candidato (y el total de votos)
- El botón "recomenzar" debe volver a poner ceros en los tres contadores de votos
- El botón "descontar último voto" sirve para corregir un error al registrar la última preferencia, es decir, debe descontar un voto al candidato que recibió el voto más reciente (y al total de votos)

Solución

Comenzamos definiendo la clase y sus variables. En este caso necesitaremos un frame y componentes para agregar la funcionalidad del enunciado.

```
class Recuento{

      public Frame f;
      private LabelL1,L2,L3;
      private TextField votos1,votos2,votos3;
      private Button      cdto1, cdto2, descontar, recomenzar;
```

Ademas de las variables gráficas utilizaremos tres variables de tipo int, dos para guardar la suma de los votos de cada candidato y una tercera para tener en memoria que candidato recibio el ultimo voto.

```
      private int v1,v2,ultimoVoto;
```

Luego en el constructor de la clase nos encargamos de inicializar cada una de las variables graficás con los parámetros adecuados para obtener la interfaz mostrada en el enunciado.

```
public Recuento(){

        f=new Frame();

        L1= new Label("Candidato");
        L2= new Label("No de Votos");
        L3= new Label("Total");

        votos1 = new TextField("0");
        votos2 = new TextField("0");
        votos3 = new TextField("0");

        cdto1= new Button("Rosa Maria");
        cdto2= new Button("Jose Manuel");
        descontar= new Button("descontar ultimo voto");
        recomenzar= new Button("recomenzar");
```

Seteamos el layout del frame y agregamos las componentes.

```
        f.setLayout(new GridLayout(5,2));

        add(L1); add(L2);
        add(cdto1); add(votos1);
        add(cdto2); add(votos2);
        add(L3); add(votos3);
        add(descontar); add(recomenzar);
```

Agregamos los listener a los botones, empacamos y mostramos la interfaz, e inicializamos las variables de acumulación.

```
        cdto1.addActionListener(new MiEscuchador());
        cdto2.addActionListener(new MiEscuchador());
        descontar.addActionListener(new MiEscuchador());
        recomenzar.addActionListener(new MiEscuchador());

        f.pack(); f.show();

        v1=0; v2=0; ultimoVoto=0;
}
```

Para poder capturar los eventos debemos definir la clase que implementa el ActionListener utilizado en el constructor.

```
class MiEscuchador implements ActionListener{
        public void actionPerformed (ActionEvent x){
```

Con el metodo getSource() diferenciamos las Fuentes de eventos.
Si el evento proviene del boton cdto1 debemos dar un voto al candidato 1 incrementando la variable que almacena sus votos y actualizando la variable de memoria para el ultimo voto.

```
                if(x.getSource()==cdto1){
                        ++v1;
                        ultimoVoto=1;
                }
```

Si el evento fue recibido desde el boton cdto2 debemos hacer lo equivalente al evento asociado a cdto1 pero sobre el candidato 2.

```
                    else if(x.getSource()==cdto2){
                          ++v2;
                          ultimoVoto=2;
                    }
```

Si el boton presionado fue recomenzar, entonces volvemos todas nuestras variables de conteo a 0.

```
                    else if(x.getSource()==recomenzar)
                          v1=v2=0;
```

Y el ultimo caso que nos queda en la captura de un evento es el boton descontar, entonces debemos decrementar la variable de acumulación de votos que indica la variable ultimoVoto.

```
                    else {
                          if(ultimoVoto==1) --v1;
                          else --v2;
                    }
```

Actualizamos los valores desplegados por la interfaz por cada vez que recibimos un evento.

```
                    votos1.setText(""+v1);
                    votos1.setText(""+v2);
                    votos1.setText(""+(v1+v2));
              } //acitonPerformed
       }//class MiEscuchador
```

Y para terminar creamos el metodo principal que ejecutara la clase de la interfaz.

```
       static public void main(String args[]){
             Recuento r=new Recuendot();
       }
}
```

6. Divisibles por Y (Pregunta 3 Control 2, 2001)

a) Escriba un método de encabezamiento

int reordenar(int n, int[]x, int y)

que reordene los primeros n elementos del arreglo x en dos grupos: a la izquierda los divisibles por y, y a la derecha los que no son divisibles por y. El método debe entregar adicionalmente como resultado la cantidad de elementos que son divisibles por y. Por ejemplo:

```
int[] a = { 5, 8, 6, 9, 3, 2, 4 };      //arreglo de 7 elementos
int i = reordenar(7, a, 2);             //reordenar los primeros 6
                                        //elementos del arreglo
```

deja el arreglo a con los valores {8,6,2,5,9,3,4} y la variable i con el valor 3

b) Escribir un programa que use el método anterior para escribir todos los números pares entre 1 y 1000 que son divisibles por 3 y por 5.

Solución

Parte a:

```
int reordenar(int n, int []x, int y){
       int i=0;
       int j=n-1;
       while(i<j){
              if(x[i]%y!=0 && x[j]%y==0){
                     int temp=x[j];
                     x[j]=x[i];
                     x[i]=temp;
              }
              if(x[i]%y==0) i++;
              if(x[j]%y!=0) j--;
       }
       return i;
}
```

Parte b:

```
Console c=new Console();
for(int i=1;i<=1000;i++)
       a[i-1]=i;

int largo=a.length;

largo=reordenar(largo;a;5);
largo=reordener(largo;a;3);

for(int i=0;i<largo;i++)
       c.println(a[i]);
```

7. Calendario (Pregunta 1 Examen, 1999)

Programe el siguiente método:

void cal(int[] diasXMes, String[] nombreMes, int diaSemana)

Este método debe escribir en el archivo calen.txt un calendario como este:

```
Enero
Mo Tu We Th Fr Sa Su
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31

Febrero
Mo Tu We Th Fr Sa Su
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28
```

cal recibe en diasXmes[i] el numero de días que posee el i-ésimo mes (28, 30 o 31) y en nombreMes[i] el nombre de ese mes (Enero, Febrero, etc.). Ambos arreglos poseen 12 elementos. El parámetro diaSemana indica en que día de la semana comienza el año (1 corresponde 1 a un Lunes, 2 a un Martes, ... y 7 a un Domingo).

El calendario se obtiene invocando **cal** de manera que diasXmes[1] sea 28 (año bisiesto) y diaSemana sea 3 (el primero de Enero es Miércoles). Preocúpese de cada cifra quede exactamente debajo del día que le corresponde (en la forma indicada en el ejemplo):

Solución

Definimos el metodo segun el encabezado entregado en el enunciado. Como utilizaremos archivos debemos preocuparnos de las excepciones, en esta ocasion utilizaremos throws y no try y catch.

```
void cal(int[] diasXMes, String[] nombreMes, int diaSemana)
            throws IOException {
```

Para imprimir el calendario, ademas de los parametros recibidos, necesitaremos un escritor de archivos y una variable para mantener la continuidad de los dias de un mes a otro.

```
      PrintWriter pw = new PrintWriter (
            new FileWriter("calen.txt"));

      // Para comenzar los meses
      int ultimoMes = diaSemana;
```

Como debemos crear un calendario de un año en particular, debemos hacer un ciclo sobre los meses que imprimiremos.

```
// Ciclo por mes
for (int mes=0; mes<12; mes++) {
```

Para cada ciclo :

Imprimimos el nombre del mes.

```
// Nombre del mes
pw.println(nombreMes[mes]);
```

Imprimimos los dias de la semana (este string es igual para todos los meses).

```
// Días de la semana
pw.println("Mo Tu We Th Fr Sa Su");
```

Rellenamos los dias que ocupo el mes pasado en la presente semana. Esto lo hacemos saltándonos tres espacios tantas veces como nos indique la variable ultimoMes que utilizamos para almacenar el dia en que termino el mes anterior.

```
// Se salta los primeros días
for (int i=1; i<ultimoMes; i++)
      pw.print("   "); // 3 espacios
```

Desplegamos las fechas correspondientes al mes. Para esto utilizamos un ciclo hasta la cantidad indicada en diasXMes[mes] que nos indicara la cantidad de dias del presente mes y utilizamos una nueva variable semana para avanzar en los dias y notar cuando termina una semana.

```
// Se ponen los días dependiendo de la semana
int semana = ultimoMes;
for (int dia=1; dia<=diasXMes[mes]; dia++) {
```

Si el numero a imprimir es mayor a 9 imprimimos un espacio para mantener la alineación y si es menor a 10 debemos imprimir dos espacios.

```
if (dia > 9)
      pw.print(" "); // 1 espacio
else pw.print("  "); // 2 espacios
```

Imprimimos el fecha y aumentamos la variable semana.

```
pw.print(dia);
semana++;
```

Si semana tiene un valor superior a 7 significara que debemos pasar a la proxima semana. En este caso imprimimos el fin de linea y regresamos semana a 1. Con esto terminamos el ciclo de escritura dentro de un mes.

```
                    if (semana > 7) {
                        pw.println();
                        semana = semana - 7;
                    }
                }
```

Al cambiar de mes debemos actualizar la variable que indican en que dia comenzar ultimoMes e imprimir las lineas de separación entre meses. Con esto terminamos el ciclo que recorre los meses.

```
                // Para el siguiente mes
                ultimoMes = semana;
                pw.println();
                pw.println();
        }
```

Y para terminar cerramos el escritor de archivo para guardar lo escrito.

```
        // Fin de la escritura
        pw.close();
}
```

8. Votaciones II (Pregunta 2 Examen, 1999, Propuesto)

El servicio electoral le pide a Ud. que escriba un programa que agilice el recuento de votos para una posible segunda vuelta en las próximas elecciones. El programa debe desplegar una ventana (Frame) con dos botones (uno para cada candidato de al segunda vuelta) y 4 campos de texto que muestren la cantidad de votos que ha percibido cada candidato y el porcentaje relativo de votos. La siguiente figura muestra la apariencia que debe tener la ventana:

Cada vez que el usuario presione algunos de los botones, su programa debe incrementar el numero de votos del candidato respectivo y recalcular los porcentajes relativos. En este recuento no se consideran los votos nulos y blancos.

9. Polinomios

La siguiente tabla define las operaciones del Tipo de Dato Abstracto (TDA) en una clase llamada **Polinomio**:

Método	Descripción	Tipo Retorno
Polinomio()	Inicializa polinomio vacío	N/A
Polinomio(int n, double[] a)	Inicializa polinomio de grado n con elementos del arreglo a[]	N/A
P.valor(double x)	Evalua P en x	double
P.suma(Polinomio Q)	P+Q	Polinomio
P.coeficiente(int i, double x)	Asigna x al i-ésimo coeficiente	void
P.grado()	Grado del polinomio	int
P.toString()	Ej: 12.1 x^3 + 5x + 4.9	String

(a) Escribe un programa que use el TDA **Polinomio** para calcular el seno de un ángulo en radianes a través de la fórmula:

$$\textbf{Seno(x)} = x/1! - x^3/3! + x^5/5! - x^7/7! + x^9/9! - x^{11}/11!$$

(b) Escribe el método **suma**, suponiendo la representación (variables de instancia) de **Polinomio**:

```
protected final int N = 10 ;            // grado máximo
protected double[] A = new double[N]    // coeficientes
protected int n = 0                     // grado
```

(c) Define la clase **Parabola** (heredada a partir de **Polinomio**), para la cual deben implementarse los siguientes métodos:

Método	Descripción
Parabola(double a, double b, double c)	Construye una parábola con coeficientes de ax^2+bx+c
boolean imaginaria()	Retorna verdadero si la parábola no corta el eje x

Solución

Parte a: Método del seno

```
public class AlgunPrograma {
      // Método para calcular el factorial
      public int fact(int x) {
            if (x == 0) return 1;
            return x * fact(x-1);
      }

      public double seno(double x) {
            // Declaramos el arreglo de coeficientes
            double a[] = new double[12];
```

```
            // Llenamos los coeficientes del polinomio
            boolean positivo = true;
            for (int i=0; i<12; i++) {
                  if (i % 2 == 0) {
                        // Caso par
                        a[i] = 0;
                  }
                  else {
                        // Caso impar
                        a[i] = 1.0 / fact(i);

                        // le ponemos el signo
                        if (!positivo) {
                              a[i] = a[i] * -1;
                        }

                        // siguiente es de signo cambiado
                        positivo = !positivo;
                  }
            }

            // Creamos el polinomio
            Polinomio p = new Polinomio(11, a);

            // Retornamos la evaluación en x
            return p.valor(x);
      }
}
```

Parte b: Método Suma

```
public class Polinomio {
      protected final int N = 10 ;            // grado máximo
      protected double[] A = new double[N]    // coeficientes
      protected int n = 0                     // grado

      ...

      public Polinomio suma(Polinomio Q) {
            // Como se tienen disponibles los coeficientes de
            // cada polinomio, basta sumarlos en un nuevo arreglo
            double[] B = new double[N];
            for (int i=0; i<N; i++) {
                  B[i] = this.A[i] + Q.A[i];
            }

            // Ahora retornamos un nuevo polinomio con esos valores
            return new Polinomio(N, B);
      }
}
```

Parte c: Clase Parábola

```
public class Parabola extends Polinomio {
      public Parabola(double a, double b, double c) {
            // Asignamos los coeficientes al arreglo
            super.A[2] = a;
```

```
                super.A[1] = b;
                super.A[0] = c;

                // Le asignamos el grado
                super.n = 2;

                // inicializamos con 0 el resto de los coeficientes
                for (int i=3; i<N; i++) {
                        super.A[i] = 0;
                }
        }

        public boolean imaginaria() {
                // Lo único que hay que analizar es si
                // b^2 - 4ac es < 0, ya que eso determina si
                // los valores de x son imaginarios
                return ((Math.pow(super.A[1], 2) - 4 * super.A[2] *
                        super.A[0]) < 0);
        }
}
```

10. Cuadrados y Círculos

(a) Escribe la clase **Circulo** de manera que acepte las siguientes instrucciones

Métodos	Descripción
void asignar(int r, int h, int v)	Asigna radio y coordenadas vertical y horizontal
void dibujar(Graphics G)	Dibuja el circulo en un objeto Graphics
double area()	Retorna el area del circulo

(a) Escribe la clase **CuadradoCircunscrito** que se derive de la clase **Circulo** y que incluya las operaciones

Métodos	Descripción
void dibujar(Graphics G)	Dibujar el cuadrado (y el círculo inscrito)
double area()	Retorna el área del cuadrado

(b) Escribe un programa que reciba un arreglo de objetos, los dibuje y entregue la suma de sus áreas

- **double sumaDibujando(int n, Circulo[] x)**

Solución

```
import java.awt.*;

public class Circulo {
  protected int centroX;
  protected int centroY;
  protected int radio;

  public Circulo(){
       asignar(0,0,0);
  }

  public void asignar(int r, int h, int v){
       radio=r;
       centroX=h;
       centroY=v;
  }


  public void dibujar(Graphics g){
       g.setColor(Color.RED);
       g.drawOval(centroX-radio,centroY-radio, 2*radio,2*radio);
  }


  public double area(){
       return Math.PI*radio*radio;
  }
```

```
  public static double sumaDibujando(int n, Circulo[] x,Graphics g){
       double sumaAreas=0;
       for (int i = 0; i < n; i++) {
              Circulo circulo = x[i];
              circulo.dibujar(g);
              sumaAreas+=circulo.area();
       }
       return sumaAreas;
  }
}


import java.awt.*;

public class CuadradoCircunscrito extends Circulo{

  public CuadradoCircunscrito(){
       asignar(0,0,0);
  }

  public void dibujar(Graphics g) {
       super.dibujar(g);
       g.drawRect(centroX-radio,centroY-radio,2*radio,2*radio);
  }

  public double area() {
       return 4*radio*radio;
  }
}
```

11. Polígonos

La clase **Poligono** usa la clase **Vertice** definida de la siguiente forma:

```
public class Vertice  {
        public int x, y ;

        public Vertice (int x, int y)  {
                this.x = x ;
                this.y = y ;
        }
}
```

Los polígonos tienen una representación interna como un arreglo de vértices, y el largo máximo de vértices que puede tener un polígono es de 100. Además, para que el polígono sea cerrado, debe repetirse el último vértice (igual al primero).

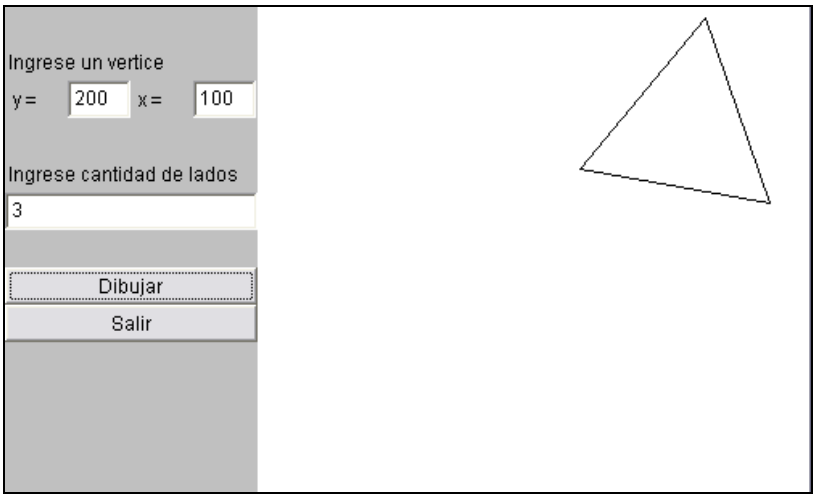
Los métodos de la clase **Poligono** son:

Método	Descripción
Poligono()	Crea un polígono sin vertices
void agregar(int x, int y)	Agrega el vertice (x, y) al Poligono
void eliminar(int n)	Elimina el n-ésimo vértice el polígono, dejando el resto contiguo
void dibujar(Graphics G)	Dibuja el polígono

(a) Implementa la clase **Polígono**

(b) Deriva de **Poligono** una clase **Regular**, que dado un entero **n** < 100 y un vértice, cree un polígono regular de n lados

(c) Escribe un applet que permita al usuario ingresar un vértice y la cantidad de lados , de modo que cada vez que se presione el botón **Dibujar**, un polígono regular se dibuje a partir de dicho vértice. La pantalla debe lucir como la siguiente:



Solución:

La forma de escribir un applet es muy similar a la usada para escribir un frame normal. La utilidad de un applet es que puede ser desplegado dentro de una pagina web. A continuación se presenta el paralelo entre la creación de un frame y unn applet.

	Frame	**Applet**
Declaración:	public class MiFrame extends Frame implements ActionListener	public class MiApplet extends Applet implements ActionListener
Inicialización:	Las variables de instancia se inicializan en el constructor	Las variables de instancia se inicializan en el método "void init()"(no existe constructor)
Ejecución:	Se debe tener un método "main" se ejecuta como cualquier programa java MiFrame	No require de un método main. Para ejecutar un applet se requiere de una página web y utilizar el appletviewer

Ejemplo

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
public class MiFrame extends Frame
implements ActionListener{
  Label rpta;
  Button ok;
  int cont;

  public MiFrame(){
    // inicializar variables
```

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
public class MiApplet extends Applet
implements ActionListener{
  Label rpta;
  Button ok;
  int cont;

  public void init(){
    // inicializar variables
```

```
   cont = 0;
   rpta = new Label();
   ok = new Button("OK");

   // fijar distribucion de los componentes
   this.setLayout(new GridLayout(3,1));
   this.add(new Label("Mi Frame"));
   this.add(rpta);
   this.add(ok);

   ok.addActionListener(this);
 }

 public void actionPerformed(ActionEvent
x){
   rpta.setText(""+cont++);
 }

 public static void main(String[] args){
   MiFrame m = new MiFrame();
   m.pack();
   m.show();
 }
}
```

```
   cont = 0;
   rpta = new Label();
   ok = new Button("OK");

   // fijar distribucion de los componentes
   this.setLayout(new GridLayout(3,1));
   this.add(new Label("Mi Applet"));
   this.add(rpta);
   this.add(ok);

   ok.addActionListener(this);
 }

 public void actionPerformed(ActionEvent
x){
   rpta.setText(""+cont++);
 }


}
```

Ejemplo de Ejecución:

java MiFrame	Se debe tener un archivo html como el siguiente: (MiApplet.html) <html> <body> <applet code="MiApplet.class" width="100" height="100"> </applet> </body> </html> y para ejecutar: appletviewer MiApplet.html

Y ahora resolvamos el problema...

Parte a: Clase Polígono

```
import java.awt.*;

public class Poligono {
  private Vertice[] vertices;
  private static int MAX_CANTIDAD=100;
  private int cantidad;
```

```
  public Poligono(){
       vertices=new Vertice[MAX_CANTIDAD];
       cantidad=0;
  }

  public void agregar(int x, int y){
       vertices[cantidad++]=new Vertice(x,y);
  }

  public void eliminar(int n){
       for (int i = n-1; i < cantidad-1; i++) {
             vertices[i]=vertices[i+1];
       }
       cantidad--;
  }

  public void dibujar(Graphics G){
       for (int i = 1; i < cantidad; i++) {
             Vertice verticeAnterior = vertices[i-1];
             Vertice verticeActual=vertices[i];
             G.drawLine(verticeAnterior.x,
verticeAnterior.y,verticeActual.x,verticeActual.y);
       }
  }
}
```

Parte b: Clase Regular

```
public class Regular extends Poligono{

  public Regular(int n,Vertice vertice){
       //calculamos un angulo inicial al centro del poligono al azar
       double angulo=Math.random()*2*Math.PI;
       //calculamos una distancia inicial al centro del poligono
       double largo=Math.random()*100;
       //calculamos el angulo del poligono regular
       double deltaAngulo=2*Math.PI/n;
       //coordenadas del centro del poligono
       double centroX=vertice.x-largo*Math.cos(angulo);
       double centroY=vertice.y-largo*Math.sin(angulo);

       agregar(vertice.x,vertice.y);
       for (int i = 0; i < n; i++) {
             angulo+=deltaAngulo;
             double x=centroX +Math.cos(angulo)*largo;
             double y=centroY+Math.sin(angulo)*largo;
             agregar((int) x,(int) y);
       }
       agregar(vertice.x,vertice.y);
  }
}
```

Parte c: Applet

```
import java.awt.*;
```

```
import java.awt.event.*;
import java.applet.Applet;

public class MyApplet extends Applet {
  private Canvas c;
  private Button dibujar;
  private Button salir;
  private TextField vertX;
  private TextField vertY;
  private TextField lados;

  public void init() {
       c = new Canvas();
       setLayout(new BorderLayout());
       setSize(500, 300);

       vertX = new TextField();
       vertY = new TextField();

       Panel numeros = new Panel();
       numeros.setLayout(new GridLayout(1, 4));
       numeros.add(new Label(" y = "));
       numeros.add(vertX);
       numeros.add(new Label(" x = "));
       numeros.add(vertY);

       dibujar = new Button("Dibujar");
       lados = new TextField();
       salir = new Button("Salir");

       Panel west = new Panel();
       west.setBackground(Color.lightGray);
       west.setLayout(new GridLayout(13, 1));
       west.add(new Label(""));
       west.add(new Label("Ingrese un vertice"));
       west.add(numeros);
       west.add(new Label(""));
       west.add(new Label("Ingrese cantidad de lados"));
       west.add(lados);
       west.add(new Label(""));
       west.add(dibujar);
       west.add(salir);
       //agregamos algunos labels para que se vea mejor
       west.add(new Label(""));
       west.add(new Label(""));
       west.add(new Label(""));

       add(BorderLayout.WEST, west);
       add(BorderLayout.CENTER, c);

       //agregamos Listeners
       salir.addActionListener(new QuitListener());
       dibujar.addActionListener(new DrawListener());
  }

  class QuitListener implements ActionListener {
       public void actionPerformed(ActionEvent e) {
              System.exit(0);
       }
  }

  class DrawListener implements ActionListener {
```

```
       public void actionPerformed(ActionEvent e) {
              String textX = vertX.getText();
              String textY = vertY.getText();
              String textL = lados.getText();
              if(textX.equals("")|| textY.equals("")||
textL.equals("")) {
                     return;
              }
              Vertice v = new Vertice(Integer.parseInt(textX),
Integer.parseInt(textY));
              Poligono p = new Regular(Integer.parseInt(textL), v);
              p.dibujar(c.getGraphics());
       }
  }
}
```

12. Cantidad de Habitantes

Escribe un programa que solicite la cantidad de habitantes de cada una de las 13 regiones del país y escriba la siguiente tabla:

```
Region  Población       %
-----------------------------
1
2
...
13
Total                   100.0
```

Solución

```
Console c = new Console();
int[] pobl = new int[13];
int total = 0;

for (int i=0; i<13; i++) {
      c.println("Ingrese la población de la región " + i);
      pobl[i] = c.readInt();
      total += pobl[i];
}

c.println("Region  Población       %");
c.println("-----------------------------");
for (int i=0; i<13; i++) {
      c.println(i + "      " + pobl[i] +
                  "      " + (pobl[i]*100/total));
}
c.println("Total  " + total + "      100.0");
```

13. Frecuencias

Escribe un programa que determine la frecuencia de aparición de cada una de las letras del alfabeto (sin importar mayúsculas / minúsculas) de un archivo de texto:

```
letra   frecuencia
-------------------
a
b
...
z
```

Solución

Definiremos un método que realice y que lo devuelva en un arreglo de enteros, en donde cada una de las 61 letras (a -> z sin contar ñ, ll ni ch) corresponde a un espacio del arreglo. Para ello definiremos una constante String con las letras del abecedario.

```
final String abecedario = "abcdefghijklmnopqrstuvwxyz";
int[] cuentaLetras(String nombreArchivo) {
      BufferedReader f = new BufferedReader(
            new FileReader(nombreArchivo));
      // Se crea e inicializa el arreglo de resultados
      int[] letras = new int[abecedario.length()];
      for (int i=0; i<letras.length; i++)
            letras[i] = 0;
      // Mientras se lee el archivo contaremos las letras
      String linea;
      while( (linea = f.readLine()) !=null) {
            for (int c=0; c<linea.length(); c++) {
                  int p = abecedario.indexOf(
                        linea.charAt(c));
                  // Solo consideramos letras
                  if ( p>=0 )
                        letras[p]++;
            }
      }
      f.close();
      return letras;
}
```

Y el programa principal para que imprima esto:

```
Console C = new Console();
String nombre = c.readLine();
int[] n = cuentaLetras(nombre);
C.println("letra   frecuencia");
C.println("-------------------");
for (int j=0; j<n.length; j++) {
      C.print(abecedario.charAt(j));
      C.println(n[j]);
}
```

14. Perímetros de Figuras y Enlace Dinámico

Dada la siguiente clase Figura:

```
class Figura {
      public Figura() { }
      public double perimetro() { return 0; }
}
```

(a) Implemente una clase Circulo que, usando la siguiente representación interna:

```
class Circulo extends Figura {
      double cx, cy;      // Centro
      double r;           // Radio
      public Circulo(double x, double y, double r) {...}
      public double perimetro() {... }
}
```

permita usar la clase Círculo.

(b) Implemente una clase Cuadrado que, usando la siguiente representación interna:

```
class Cuadrado extends Figura {
      double cx, cy;      // Centro
      double l;           // Lado
      public Cuadrado(double x, double y, double l) {...}
      public double perimetro() {...}
}
```

también permita usar la clase Cuadrado.

(c) Haciendo un paralelo con las clases anteriores, implemente la clase Poligono definida como:

```
class Poligono extends Figura {
      double x[], y[]; // Puntos
      public Poligono(double[] x, double[] y) {...}
      public double perimetro() {...}
}
```

poniendo énfasis en que el perímetro es la suma de las distancias entre los puntos contiguos de un polígono.

(d) Haga un método:

```
      public static double sumaPerimetros(Figura[] fig) {...}
```

que recibe un arreglo de objetos de tipo Figura y que retorna la suma de los perímetros de todas las figuras independientes del Tipo Dinámico.

Solución

Parte a:

```
class Circulo extends Figura {
	double cx, cy;		// Centro
	double r;		// Radio
	public Circulo(double x, double y, double r) {
		super("Circulo");
		this.cx = x;
		this.cy = y;
		this.r = r;
	}
	public double perimetro() {
		return 2 * Math.PI * this.r;
	}
}
```

Parte b:

```
class Cuadrado extends Figura {
	double cx, cy;		// Centro
	double l;	// Lado
	public Cuadrado(double x, double y, double l) {
		super("Cuadrado");
		this.cx = x;
		this.cy = y;
		this.l = l;
	}
	public double perimetro() {
		return 4 * this.l;
	}
}
```

Parte c:

```
class Poligono extends Figura {
	double x[], y[]; // Puntos
	public Poligono(double[] x, double[] y) {
		super("Polígono");
		this.x = new double[x.length];
		this.y = new double[y.length];
		for (int i=0; i<x.length; i++) {
			this.x[i] = x[i];
			this.y[i] = y[i];
		}
	}
	public double perimetro() {
		double suma = 0;
		for (int i=0; i<x.length-1; i++) {
			suma += Math.sqrt(
				Math.pow(x[i+1] - x[i] ,2) +
				Math.pow(y[i+1] - y[i] ,2));
		}
		suma += Math.sqrt(
			Math.pow(x[0] - x[x.length-1] ,2) +
			Math.pow(y[0] - y[x.length-1] ,2));
		return suma;
```

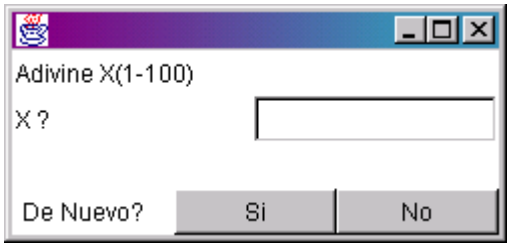
```
        }
}
```

Parte d:

```
        public static double sumaPerimetros(Figura[] fig) {
                double suma = 0;
                for (int i=0; i<fig.length; i++) {
                        suma += fig[i].perimetro();
                }
                return suma;
        }
```

15. Adivinador Gráfico de Números

Escriba una I.G que permita adivinar un N° de acuerdo al siguiente diálogo:



Solución

Importamos los paquetes de clases necesarios para el programa.

```
import java.awt.*;
import java.awt.event.*;
```

Creamos la clase extendiendola de Frame e implementando ActionListener.

```
public class Adivinanza extends Frame implements ActionListener {
```

Definimos las components que utilizaremos y una variable de tipo int para almacenar el numero secreto.

```
      Label         titulo        = new Label("Adivine X(1-100)"),
                    pregunta      = new Label("X ?"),
                    respuesta     = new Label(),
                    denuevo       = new Label(" De Nuevo? ");
      TextField     numero        = new TextField();
      Button        si            = new Button("Si"),
                    no            = new Button("No");
      int x; // NUMERO SECRETO!!!!!!
```

Dentro del constructor llamamos a la funcion anicializar(), encargada de calcular el numero secreto, y dividimos las componentes separándolas en dos paneles.

```
      public Adivinanza(){
            inicializar();

            Panel p1 = new Panel();
            p1.setLayout(new GridLayout(1,2));
            p1.add(pregunta);
            p1.add(numero);

            Panel p2 = new Panel();
            p2.setLayout(new GridLayout(1,3));
            p2.add(denuevo);
            p2.add(si);
```

```
            p2.add(no);
```

agregamos los paneles al frame principal y asociamos el listener que capturará las acciones a los botones y al campo de texto.

```
            setLayout(new GridLayout(4,1));
            add(titulo);
            add(p1);
            add(respuesta);
            add(p2);

            numero.addActionListener(this);
            si.addActionListener(this);
            no.addActionListener(this);
```

El listener que damos por argumento a addActionListener() es this porque nos referimos a la misma clase en que estamos parados, esto lo podemos hacer porque es esta clase la que implementa ActionListener. Como ultimas instrucciones del constructor empacamos la interfaz y la mostramos.

```
            this.pack();
            this.show();
      }
```

Implementamos el metodo actionPerformed() de ActionListener.

```
      public void actionPerformed(ActionEvent e){
```

Si el evento viene del boton si, debemos recalcular el numero secreto.

```
            if(e.getSource() == si)
                  inicializar();
```

Si el evento viene del boton no, debemos terminar el programa.

```
            else
                  if(e.getSource() == no)
                        System.exit(0);
```

Y el ultimo caso de fuente de evento es el campo de texto. En este caso debemos leer el numero ingresado y dar una respuesta de acuerdo a la posición del numero leido respecto del secreto.

```
                  else{
                        int n = Integer.parseInt(
                              numero.getText());
                        if(x == n)
                              respuesta.setText("X = "+n);
                        if(x < n)
                              respuesta.setText("X < "+n);
                        if(x > n)
```

```
                                    respuesta.setText("X > "+n);
                        }
        }
```

Definimos el metodo inicializar() que calcula el numero secreto al azar y setea la interfaz a sus valores iniciales.

```
        public void inicializar(){
                x = 1+ (int)(Math.random()*100);
                respuesta.setText("");
                numero.setText("");
        }
```

Y por ultimo el metodo principal que crea el objeto de la clase.

```
        static public void main(String[] args) {
                Adivinanza n = new Adivinanza();
        }
}
```

16. Conjuntos en Java

Los conjuntos son reales repositorios donde se almacenan distintos tipos de datos. Por ejemplo, existen conjuntos de Números, Letras, Palabras y hasta Objetos (mesa, silla, casa, etc).

Se desea implementar conjuntos en Java, y se conoce que las funcionalidades de los conjuntos son:

- Unión: Un conjunto A se une con un conjunto B dando como resultado un conjunto C que contiene los elementos de A y B sin repetirlos.
- Intersección: Un conjunto A se intersecta con un conjunto B dando como resultado un conjunto C que contiene solo los elementos repetidos entre A y B.
- Subconjunto: Un conjunto A es subconjunto de B si los elementos de A están también en B, dando como resultado Verdadero o Falso.
- Igualdad: Un conjunto A es igual al conjunto B si A es subconjunto de B y B es subconjunto de A, dando como resultado Verdadero o Falso.
- Cardinal: Retorna un número que indica la cantidad de elementos que tiene el conjunto A.

(a) Implemente una Interface Conjunto que represente estas cuatro funcionalidades como genéricas de los conjuntos.

(b) Se desea implementar una Clase ConjuntoEnteros que represente un Conjunto (implemente la interface) de número enteros. Para ello puede utilizar la siguiente representación interna:

```
public class ConjuntoEntero implements Conjunto {
      protected int[] bolsa;
      public ConjuntoEntero() {...} // Lo crea vacío
      public ConjuntoEntero(int x) {...} // Lo crea con elto x
      public ConjuntoEntero(int[] a) {...} // Lo crea con arreglo a
      ...
}
```

Nota: Se le entregan 3 constructores para que los implemente.

Solución

Parte a:

```
public interface Conjunto {
      public int cardinal(); public Conjunto union (Conjunto B);
      public Conjunto interseccion (Conjunto B);
      public boolean subconjunto (Conjunto B);
      public boolean igualdad (Conjunto B);
}
```

Parte b:

```
public class ConjuntoEntero implements Conjunto {
	protected int[] bolsa;

	public ConjuntoEntero() {
		// No es necesario hacer nada
	}

	public ConjuntoEntero(int x) {
		// Creamos un arreglo de 1 elemento
		this.bolsa = new int[1];
		this.bolsa[0] = x;
	}

	public ConjuntoEntero(int[] a) {
		// Creamos el arreglo con el largo de a
		this.bolsa = new int[a.length];
		for (int i=0; i<a.length; i++)
			this.bolsa[i] = a[i];
	}

	public boolean cardinal () {
		// Retornamos el largo de bolsa
		return this.bolsa.length;
	}

	public Conjunto union (Conjunto B) {
		// Contamos si hay repetidos
		int rep = 0;
		for (int i=0; i<this.bolsa.length; i++) {
			int[] a = new int[1];
			a[0] = this.bolsa[i];
			Conjunto X = new Conjunto(A);
			if (X.subconjunto(B))
				rep++;
		}

		// Creamos un arreglo que contenga todo
		int[] r = new int[this.cardinal()+B.cardinal()-rep];
		int n = 0;

		// Ponemos los elementos que estan en A y
		// no estan en B
		for (int i=0; i<this.bolsa.length; i++) {
			int[] a = new int[1];
			a[0] = this.bolsa[i];
			Conjunto X = new Conjunto(A);
			if (!X.subconjunto(B)) {
				r[n] = this.bolsa[i];
				n++;
			}
		}

		// Ponemos los elementos de B
		for (int i=0; i<B.length; i++) {
			r[n] = B.bolsa[i];
			n++;
		}

		// Retornamos el conjunto creado con r
		return new Conjunto(r);
```

```
	}

	public Conjunto interseccion (Conjunto B) {
		// Contamos si hay repetidos
		int rep = 0;
		for (int i=0; i<this.bolsa.length; i++) {
			int[] a = new int[1];
			a[0] = this.bolsa[i];
			Conjunto X = new Conjunto(A);
			if (X.subconjunto(B))
				rep++;
		}

		// Creamos un arreglo para repetidos
		int[] r = new int[rep]
		int n = 0;

		// Ponemos los elementos que estan en A y
		// que también estan en B
		for (int i=0; i<this.bolsa.length; i++) {
			int[] a = new int[1];
			a[0] = this.bolsa[i];
			Conjunto X = new Conjunto(A);
			if (X.subconjunto(B)) {
				r[n] = this.bolsa[i];
				n++;
			}
		}

		// Retornamos un conjunto creado con r
		return new Conjunto(r);
	}

	public boolean subconjunto (Conjunto B) {
		// Vemos si la instersección es igual a A
		Conjunto C = this.interseccion(B);
		return this.igualdad(C);
	}

	public boolean igualdad (Conjunto B) {
		// Suponemos iguales, y probamos lo contrario
		boolean igual = true;
		for (int i=0; i<this.bolsa.length; i++) {
			boolean esta = false;
			for (int j=0; j<B.cardinal(); j++)
				if (this.bolsa[i] == B[j])
					esta = true;
			igual = igual && esta;
		}
		return igual;
	}
}
```

17. El Acsensor

Se quiere tener una interfaz gráfica que simule el display de un ascensor, de manera que muestre el piso en el que se encuentra el ascensor. Para esto, cree una interfaz gráfia que tenga dos botones (para subir y bajar) y cada vez que el ascensor sube un piso, debe aparecer una nueva barra en el indicador del piso y cuando baja, debe desapaercer una barra.

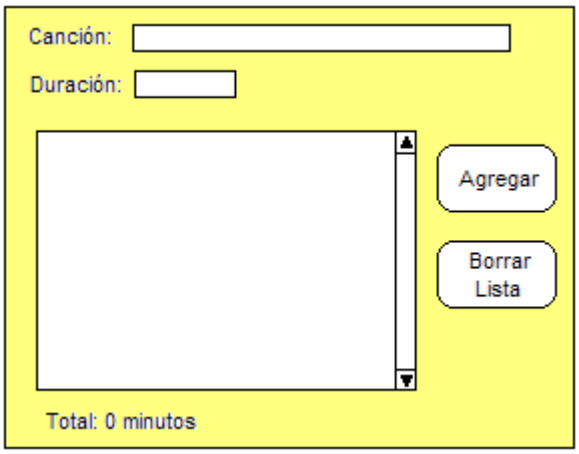
Solución:

```
import java.awt.event.*;
import java.awt.*;
import java.applet.*;

public class Ascensor extends Frame implements ActionListener {

	Button subir, bajar;
	Canvas canvas;
	int piso;

	public Ascensor() {
		bajar = new Button("<< Bajar");
		bajar.addActionListener(this);
		subir = new Button("Subir >>");
		subir.addActionListener(this);

		Panel pan = new Panel();
		pan.add(bajar);
		pan.add(subir);

		canvas = new Canvas();
		canvas.setSize(100,150);

		setLayout(new BorderLayout());
		add("Center", canvas);
		add("South", pan);
```

```
                piso = 1;
        }

// "limpiar" el canvas <=> dibujar un rectangulo relleno de color
//blanco
        public void limpiar() {
                Graphics g = canvas.getGraphics();
                g.setColor(Color.white);
                g.fillRect(0,0,100,150);
        }

        // mostrar el borde de color en donde apareceran las barras
        public void mostrarFondo() {
                Graphics g = canvas.getGraphics();
                g.setColor(Color.green);
                g.drawRect(10,10,80,130);
        }

        // dependiendo del piso en que se encuentra, mostrar las
//barras necesarias
        public void mostrarPiso() {
                Graphics g = canvas.getGraphics();
                g.setColor(Color.blue);

                if (piso >= 1)
                        g.fillRect(15,110,10,30);
                if (piso >= 2)
                        g.fillRect(35,80,10,60);
                if (piso >= 3)
                        g.fillRect(55,50,10,90);
                if (piso >= 4)
                        g.fillRect(75,20,10,120);
        }

        public void actionPerformed(ActionEvent e) {
                if (e.getSource() == bajar)
                        piso--;
                if (e.getSource() == subir)
                        piso++;

                if (piso < 1)
                        piso = 1;
                if (piso > 4)
                        piso = 4;

                limpiar();
                mostrarFondo();
                mostrarPiso();
        }

        public static void main(String[] args) {
                Ascensor a = new Ascensor();
                a.pack();
                a.show();
                a.mostrarFondo();
                a.mostrarPiso()
        }

}
```

18. Musica

Cuando se quiere copiar desde un CD de música a otro CD o a un cassette, uno puede elegir el orden, y las canciones que quiere copiar. Al hacer esto, es importante saber cuanto es el tiempo total de duración de las canciones elegidas. Es por esto, que se necesita hacer un *Frame*, usando las Interfaces Gráficas de Java (AWT). El Frame debe tener la siguiente apariencia:



Al ingresar el nombre de una canción y el tiempo de duración de ésta, debe agregarse al final del listado de canciones, junto a su duración. También es necesario que aparezca el tiempo total de las canciones ingresadas. Otra funcionalidad que debe tener es que se pueda borrar el listado de canciones.

Obs: Se supone que el tiempo de duración de una canción *siempre* viene en formato "min.seg". Por ejemplo, "3.10" representan 3 minutos y 10 segundos.

Solución:

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;

public class Caratula extends Frame implements ActionListener {

    // componentes graficos
    Button agregar, borrar;
    Label cancion, duracion, total;
    TextField nombre, tiempo;
```

```
    TextArea listado;

    // variable que guarda la suma del tiempo total
    int suma;

    public Caratula() {
        agregar = new Button("Agregar");
        agregar.addActionListener(this);

        borrar = new Button("Borrar Lista");
        borrar.addActionListener(this);

        cancion = new Label("Cancion:");
        duracion = new Label("Duracion:");
        total = new Label("Total:");

        nombre = new TextField(20);
        tiempo = new TextField(5);

        listado = new TextArea(10, 30);

        // inicializo la cantidad de segundos en 0
        suma = 0;

        // panel con los componentes de "arriba"
        Panel arriba = new Panel();
        arriba.setLayout(new GridLayout(2, 2));
        arriba.add(cancion);
        arriba.add(nombre);
        arriba.add(duracion);
        arriba.add(tiempo);

        // panel con los botones
        Panel botones = new Panel();
        botones.setLayout(new GridLayout(2, 1));
        botones.add(agregar);
        botones.add(borrar);

        // agrego los componentes al Frame
        setLayout(new BorderLayout());
        add("North", arriba);
        add("Center", listado);
        add("South", total);
        add("East", botones);
    }

    public void actionPerformed(ActionEvent e) {
        // si apretaronn el boton "agregar"
        if (e.getSource() == agregar) {
// recupero el tiempo que ingreso el usuario y lo convierto a
//segundos
            String tpo = tiempo.getText();
            int segundos = stringASegundos(tpo);

            suma += segundos;

            // creo el String con el nombre de la cancion
            // y su duracion, que voy a agregar al TextArea:
            // por ejemplo: "Hakuna Matata (1.40)"
           String nuevo=nombre.getText()+" (" +tiempo.getText()+")";

            // si el TextArea esta vacio, agrego la nueva linea
```

```
            if (listado.getText().equals(""))
                listado.setText(nuevo);

            // si no esta vacio, agrego un salto de linea ("\n")
            else
                listado.setText(listado.getText() + "\n" + nuevo);

            total.setText("Total:"+segundosAString(suma)+"minutos");
        }
        // si apretaron el boton "borrar"
        else if (e.getSource() == borrar) {
            suma = 0;
            listado.setText("");
           total.setText("Total:"+segundosAString(suma)+" minutos");
        }

        //  limpiar campos del nombre de la cancion y su duracion
        nombre.setText("");
        tiempo.setText("");
    }

    /*
     * Metodo que convierte un string en cantidad de segundos
     * (supongo el formato es "min.seg",
     *por ejemplo "3.20"-> 3 minutos y 20 segundos = 200   segundos)
     */
    public int stringASegundos(String palabra) {
        int pos = palabra.indexOf(".");
        int min = Integer.parseInt(palabra.substring(0, pos));
        int sec = Integer.parseInt(palabra.substring(pos + 1));

        return min * 60 + sec;
    }

    /*
     * Metodo que convierte cierta cantidad de segundos en un String
     * (por ejemplo, 95 -> "1.35")
     */
    public String segundosAString(int segundos) {
      int min=segundos/60;//division entre enteros entrega un entero
        int sec = segundos % 60;
        return min + "." + sec;
    }

    public static void main(String[] args) {
        Caratula disco = new Caratula();
        disco.pack();
        disco.show();
    }
}
```

19. Campeonato de Tenis

A finales de año se quiere realizar un campeonato de tenis. Cada competidor se caracteriza por su nombre, su edad y un número de registro que se le asigna al ingresar al campeonato. Como el campeonato es abierto a todo público, de cualquier edad, se separan a los competidores en distintas categorías, dependiendo de su edad. Estas categorías se especifican como características del campeonato.
Para esto, se tiene la clase *Tenista* que se describe a continuación:

```
public class Tenista {
       int registro;
       String nombre;
       int edad;
       int puntos;

       public Tenista (int registro, String nombre, int edad) {...}
       public void fijarNombre (String nombre) {...}
       public String obtenerNombre () {...}
       public void fijarEdad (int edad) {...}
       public int obtenerEdad () {...}
       public void sumarPuntos (int puntos) {...}
       public int obtenerPuntos () {...}
       public boolean perteneceCategoria (int edadMin, int edadMax)
{...}
}
```

(a) Implemente la clase *Tenista*, usando la descripción anterior.

Para iniciar el campeonato, es necesario que todos los participantes entreguen sus datos personales (nombre y edad) al inicio del campeonato. Al ingresar al campeonato, se les asigna su número de registro, y los jueces, dependiendo de la cantidad de participantes, decidirán cual será la edad mínima para ingresar al campeonato, y cómo serán divididas las categorías. Por ejemplo, si deciden que la edad mínima es 15 años y el intervalo es de 10 años, las categorías serín "menor de 15", "entre 15 y 24", "entre 25 y 34", etc.
Es necesario permitir que se muestre un listado con todos los participantes, y la categoría a la que pertenecen. Al final de cada partido, se ingresará el resultado del match, y el ganador obtendrá 2 puntos, y en caso de empate, cada uno recibirá 1 punto. Finalmente, se desea tener un listado con los resultados finales del campeonato.

(b) Implemente la siguiente clase *Campeonato*:

```
public class Campeonato {
       Tenista[] competidores;
       int edadMinima, intervaloEdad;

       public Campeonato(String[] nombres,int[] edades,int
edadMinima, int intervaloEdad) {...}
       public void mostrarPorCategorias () {...}
       public   void   resultadoPartido   (int   regTenista_1,   int
regTenista_2, int resultado) {...}
```

```
        public void mostrarGanadores () {...}
}
```

Observaciones:

- Al ingresar los tenistas al campeonato, es importante que se agreguen de manera de tener un arreglo ordenado por edad, para hacer más eficientes las búsquedas en el arreglo.
- Si es necesario, cree nuevos métodos que le permitan un código más ordenado.

Solución:

Parte a:

```
public class Tenista {
    int registro;
    String nombre;
    int edad;
    int puntos;

    public Tenista(int registro, String nombre, int edad) {
        this.registro = registro;
        this.nombre = nombre;
        this.edad = edad;
        puntos = 0;
    }

    public void fijarNombre(String nombre) {
        this.nombre = nombre;
    }

    public String obtenerNombre() {
        return nombre;
    }

    public void fijarEdad(int edad) {
        this.edad = edad;
    }

    public int obtenerEdad() {
        return edad;
    }

    public void sumarPuntos(int puntos) {
        this.puntos += puntos;
    }

    public int obtenerPuntos() {
        return puntos;
    }

    public int obtenerRegistro() {
        return registro;
    }

    public boolean perteneceCategoria(int edadMin, int edadMax) {
```

```
        if (edad >= edadMin && edad <= edadMax)
            return true;
        return false;
    }
}
```

Parte b:

```
public class Campeonato {
    Tenista[] competidores;
    int edadMinima, intervaloEdad;
    Console c=new Console();
public Campeonato(String[] nombres,int[] edades,int minima,int
inter) {
        competidores = new Tenista[nombres.length];
        edadMinima = minima;
        intervaloEdad = inter;

        int cantidad = 0;

        for (int i = 0; i < nombres.length; i++) {
            Tenista tenista = new Tenista(i, nombres[i], edades[i]);
            agregarTenista(tenista, cantidad);
            cantidad++;
        }
    }

    public void mostrarPorCategorias() {
        int pos = 0;

        // categorias
        int min = edadMinima;
        while (pos < competidores.length) {
            int max = min + intervaloEdad - 1;

            c.println("Categoria: entre " + min + " y " + max);
            Tenista aux = competidores[pos];
            while (aux.obtenerEdad() <= max) {
                c.println(" - " + aux.obtenerNombre());
                pos++;
                if (pos == competidores.length)
                    break;

                aux = competidores[pos];
            }

            min = min + intervaloEdad;
        }
    }

  public void resultadoPartido(int reg_1,int reg_2, int resultado) {
        int pts_1 = 0;
        int pts_2 = 0;

        if (resultado == -1)
            pts_1 = 2;
        else if (resultado == 1)
            pts_2 = 2;
        else {
            pts_1 = 1;
            pts_2 = 1;
```

```
        }

        for (int i = 0; i < competidores.length; ++i) {
            Tenista actual = competidores[i];

            // si es el primer tenista
            if (actual.obtenerRegistro() == reg_1)
                actual.sumarPuntos(pts_1);

            // si es el segundo
            else if (actual.obtenerRegistro() == reg_2)
                actual.sumarPuntos(pts_2);
        }
    }

    public void mostrarGanadores() {
        int pos = 0;
        int min = edadMinima;
        while (pos < competidores.length) {
            int max = min + intervaloEdad - 1;

            c.println("Categoria: entre " + min + " y " + max);
            Tenista aux = competidores[pos];
            int maxpts = -1;
            String ganadores = "";
            while (aux.obtenerEdad() <= max) {
                if (aux.obtenerPuntos() > maxpts) {
                    maxpts = aux.obtenerPuntos();
                    ganadores = aux.obtenerNombre();
                }
                else if (aux.obtenerPuntos() == maxpts) {
                    ganadores += ", " + aux.obtenerNombre();
                }

                pos++;
                if (pos == competidores.length)
                    break;

                aux = competidores[pos];
            }

            if (!ganadores.equals(""))
                c.println(" Ganadores: " + ganadores);

            min = min + intervaloEdad;
        }

    }

    private void agregarTenista(Tenista tenista, int cantidad) {
        int contador = 0;
        // iterar por los menores que el que quiero ingresar
        while (contador < cantidad) {
            Tenista actual = competidores[contador];
            if (tenista.obtenerEdad() < actual.obtenerEdad())
                break;
            contador++;
        }

        // si es el ultimo
        if (contador == cantidad) {
            competidores[contador] = tenista;
```

```
        }
        // si no, muevo los restantes
        else {
            Tenista actual = competidores[contador];

            competidores[contador] = tenista;
            contador++;

            while (contador < cantidad + 1) {
                Tenista aux = competidores[contador];
                competidores[contador] = actual;
                actual = aux;
                contador++;
            }
        }
    }

    // Prueba del campeonato
    public static void main(String[] args) {
        String[] jugadores = { "juan", "jose", "andres", "pedro" };
        int[] edades = { 17, 10, 15, 18 };

        Campeonato ca = new Campeonato(jugadores, edades, 10, 5);
        ca.mostrarPorCategorias();
        ca.resultadoPartido(2, 3, -1);
        ca.resultadoPartido(2, 0, -1);
        ca.resultadoPartido(0, 3, 1);
        ca.resultadoPartido(2, 3, 1);
        ca.mostrarGanadores();
    }
}
```

20. Empleados

Se tiene la clase *Empleado*, que sirve como clase base para los distintos empleados de una empresa. Esta clase se describe de la siguiente manera:

```
public class Empleado {
	String rut, nombre;
	int sueldo, dias;

	public Empleado (String rut, String nombre, int sueldo, int
dias) {
		this.rut = rut;
		this.nombre = nombre;
		this.sueldo = sueldo;
		this.dias = dias;
	}
	public String obtenerNombre() {
		return nombre;
	}
	public int obtenerSueldo() {
		return sueldo;
	}
	public int cantidadVacaciones() {
		return dias;
	}
}
```

La empresa también tiene *Vendedores*, que además tienen que tener registrado la cantidad de ventas que han realizado, y su sueldo se calcula como una base, más un porcentaje de las ventas que han realizado.

Parte (a):
Usando la clase *Empleado*, escriba la clase *Vendedor*, que además debe tener un método que permita obtener la cantidad de ventas que ha realizado el vendedor.

Parte (b):
Haga un método que reciba un arreglo de objetos de tipo *Empleado*, y muestre en pantalla un resumen de todos los vendedores, mostrando las ventas realizadas, y el sueldo que van a recibir. El encabezado del método es el siguiente:
public void mostrarResumen(Empleado[] empleados) {....}

Parte (c):
Haga un programa que tenga un diálogo con el usuario, en donde pregunte por los datos de un empleado, y en caso que sea un vendedor, le pida los nuevos datos necesarios. Termina cuando el usuario ingresa "fin". Después debe mostrar el resumen de las ventas y sueldo de los vendedores. El programa debe seguir un diálogo similar al siguiente:

```
Ingrese rut: 13655673-2
Ingrese nombre: Juan Perez
Ingrese sueldo: 120000
```

```
Es vendedor? n

Ingrese rut: 9876563-2
Ingrese nombre: Miguel Soto
Ingrese sueldo: 95000
Es vendedor? s
Ingrese porcentaje de ventas: 0.25
Ingrese ventas del mes: 320000

Ingrese rut: fin

Resumen:
--------

Nombre: Miguel Soto
Ventas: $320000
Sueldo: $175000

....
```

Solución:

Parte a:

```
public class Vendedor extends Empleado {
    int ventas;
    double porcentaje;

    public   Vendedor(String   rut,String   nombre,int   sueldo,int
dias,double porcentaje) {
        super(rut, nombre, sueldo, dias);
        this.porcentaje = porcentaje;
        this.ventas = 0;
    }

    public int obtenerSueldo() {
        return sueldo + (int) Math.round(ventas * porcentaje);
    }

    public int obtenerVentas() {
        return ventas;
    }

    public void agregarVentas(int ven) {
             ventas += ven;
      }
}
```

Parte b:

```
public static void mostrarResumen(Empleado[] empleados) {
   for (int i = 0; i < empleados.length; i++) {
      if (empleados[i] instanceof Vendedor) {
         Vendedor ven = (Vendedor) empleados[i];
```

```
            c.println("Nombre: " + ven.obtenerNombre());
            c.println("Ventas: $" + ven.obtenerVentas());
            c.println("Sueldo: $" + ven.obtenerSueldo());
            c.println();
        }
    }
}
```

Parte c:

```
import java.io.*;

public class Programa {
Console c=new Console();
    public static void mostrarResumen(Empleado[] empleados) {
        for (int i = 0; i < empleados.length; i++) {
            if (empleados[i] instanceof Vendedor) {
                Vendedor ven = (Vendedor) empleados[i];

                c.println("Nombre: " +ven.obtenerNombre());
                c.println("Ventas: $"+ven.obtenerVentas());
                c.println("Sueldo: $"+ven.obtenerSueldo());
                c.println();
            }
        }
    }

    public static void main(String[] args) throws IOException {
        // hacemos un arreglo suficientemente "grande"
        Empleado[] emps = new Empleado[1000];

        int pos = 0;
        while (true) {
          c.print ("Ingrese rut: ");
          String rut = c.readLine();

          // si escribe "fin", termino
          if (rut.equals("fin"))
             break;

          c.print ("Ingrese nombre: ");
          String nombre = c.readLine();

          c.print ("Ingrese dias de vacaciones: ");
          int dias = Integer.parseInt(c.readLine());

          c.print ("Ingrese sueldo: ");
          int sueldo = Integer.parseInt(c.readLine());

          c.print ("Es vendedor? ");
          String resp = c.readLine();

          // si es un vendedor, pregunto por los otros datos
          if (resp.equals("s")) {
               c.print ("Ingrese porcentaje de ventas: ");
               double porc = Double.parseDouble(c.readLine());

               c.print ("Ingrese ventas del mes: ");
               int ventas = Integer.parseInt (c.readLine());
```

```
                Vendedor ven = new Vendedor(rut, nombre, sueldo,
dias, porc);
                ven.agregarVentas(ventas);

                emps[pos] = ven;
              }

              else {
                Empleado e = new Empleado(rut, nombre, sueldo, dias);

                emps[pos] = e;
              }

              pos++;

              // escribo una linea en blanco en la pantalla
              c.println();
      }

      c.println();
      c.println("Resumen:");
      c.println("--------");
      c.println();

      mostrarResumen(emps);
   }
}
```

21. Juegos Olímpicos

Para los próximos Juegos Olímpicos, se utilizará un sistema para definir los ganadores de las distintas competencias de manera automática. Es por esto que se diseñó la siguiente clase Competidor:

```
public class Competidor {
    String nombre, nacionalidad;

    public Competidor(String nombre, String nacionalidad) {
        this.nombre = nombre;
        this.nacionalidad = nacionalidad;
    }

    public String getNombre() {
        return nombre;
    }
}
```

Implemente las clases *Nadador* y *LanzadorBala*, quienes se caracterizan por medir su desempeño según el tiempo y la distancia, respectivamente.

Solución:

```
public class Nadador extends Competidor{

    double tiempo;

    public Nadador(String nombre, String nacionalidad){
        super(nombre, nacionalidad);
        tiempo = 0;
    }

    public void setTiempo(double tiempo) {
        this.tiempo = tiempo;
    }

    public double getTiempo() {
        return tiempo;
    }
}
```

```
public class LanzadorBala extends Competidor{

    int distancia;

    public LanzadorBala(String nombre, String nacionalidad) {
        super(nombre, nacionalidad);
        distancia = 0;
    }

    public void setDistancia(int distancia) {
```

```
        this.distancia = distancia;
    }

    public int getDistancia() {
        return distancia;
    }

}
```

22. Juegos Olímpicos II

Para poder describir las competencias de los juegos olímpicos, se creó la clase *Competencia*, la cual permite obtener los ganadores de la competencia actual. Para esto, la clase tiene un arreglo de competidores, que al final de la competencia, lo ordena y al mostrar los ganadores, muestra los primeros tres competidores. Debido a que el ordenamiento del arreglo depende del tipo de competidores, posee un método abstracto `abstract void ordenar()`:

```
abstract class Competencia {
    Competidor[] competidores;

    public Competencia (Competidor[] competidores) {
        this.competidores = competidores;
    }

    public void mostrarGanadores() {
        ordenar();
        if (competidores.length == 0)
            System.out.println("No hay competidores.");
        else
            for (int i=0; i < competidores.length && i < 3; ++i) {
                String nombre = competidores[i].getNombre();
                System.out.println("Posicion " + (i + 1) + ": " +
nombre);
            }
    }
}
```

Implemente las clases CompetenciaNatacion y CompetenciaBala.

Solución:

```
public class CompetenciaNatacion extends Competencia {

    public CompetenciaNatacion(Nadador[] competidores) {
        super(competidores);
    }

    public void ordenar() {
        for (int i = competidores.length - 1; i > 0; i--) {
            for (int j = 0; j < i - 1; j++) {
                double t1 = ((Nadador) competidores[j]).tiempo;
                double t2 = ((Nadador) competidores[j + 1]).tiempo;

                if (t1 > t2) {
                    Competidor auxiliar = competidores[j];
                    competidores[j] = competidores[j + 1];
                    competidores[j + 1] = auxiliar;
                }
            }
        }
    }
}
```

```
public class CompetenciaBala extends Competencia {

    public CompetenciaBala(LanzadorBala[] competidores) {
        super(competidores);
    }

    public void ordenar() {
        for (int i = competidores.length - 1; i > 0; i--) {
            for (int j = 0; j < i - 1; j++) {
                int d1 = ((LanzadorBala) competidores[j]).distancia;
                int d2= ((LanzadorBala)competidores[j+1]).distancia;

                if (d1 < d2) {
                    Competidor auxiliar = competidores[j];
                    competidores[j] = competidores[j + 1];
                    competidores[j + 1] = auxiliar;
                }
            }
        }
    }
}
```

23. Operaciones Bancarias

Se ha fundado un nuevo banco, y para iniciar las operaciones, se requiere un programa que permita retirar dinero del cajero automático. Para esto, se ha definido la siguiente clase,

Encabezamiento	Significado	Ejemplo
Cuenta(int x)	Crea una cuenta, con saldo disponible $ x	Cuenta cta=new Cuenta(80000)
void depositar(int x)	Deposita $ x en la cuenta	cta.depositar(2000)

boolean girar(int x)	Saca $ x de la cuenta. Retorna true si no hay problemas, false si no se puede efectuar la operación.	cta.girar(5000) //true
		cta.girar(120000) //false

int saldo()	retorna saldo disponible actualmente	cta.saldo()

Se le pide a usted que:

a) Implemente la clase Cuenta
b) Como a veces la gente necesita más dinero que el que tiene, el banco decide crear una cuenta corriente, que permita que la gente se endeude (pudiendo retirar cuanto dinero quiera). Para eso, cree una clase CuentaCredito que extienda Cuenta, y redefina el/los métodos apropiados.

Solución:

Parte a:

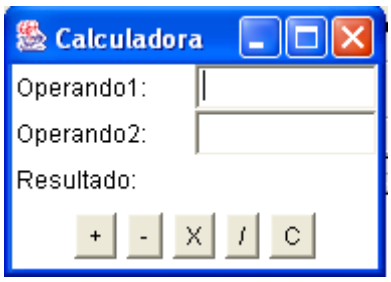
```
public class Cuenta {
	int saldo;
	public Cuenta(int x) {
		this.saldo=x;
	}
	void depositar(int x){
		if (x>0)
			this.saldo+=x;
	}
	boolean girar(int x){
		if (x<0) return false;
		if ((this.saldo-x)<0){
			return false;
		}
		else this.saldo-=x;
		return true;
	}
	int saldo(){
		return saldo;
	}
}
```

Parte b:

```
public class CuentaCredito extends Cuenta {

	public CuentaCredito(int x) {
		super(x);
	}

	boolean girar(int x){
		if (x<0) return false;
		this.saldo-=x; //o super.saldo-=x;
		return true;
	}

}
```

24. Calculadora

Implemente en un frame una calculadora simple que calcule la suma, resta, multiplicación y división de dos números reales. El frame debe debe disponer de dos Textfields para recibir los operandos y de un botón por cada operación, además de un botón para borrar los campos. Guíese por la figura.



Solución:

```
import java.awt.*;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

public class Calculadora extends Frame implements ActionListener {

    Label operando1,operando2,resultado,result;
    TextField op1,op2;
    Button sum,res,mul,div,borrar;

   public Calculadora(){

            super("Calculadora");
        operando1=new Label("Operando1:");
        operando2=new Label("Operando2:");
        resultado=new Label("Resultado:");
        result=new Label();

        op1=new TextField();
        op2=new TextField();

        sum=new Button("+");
        res=new Button("-");
        mul=new Button("X");
        div=new Button("/");
        borrar=new Button("C");
        Panel p1=new Panel();
        Panel p2=new Panel();
        Panel p3=new Panel();
        p1.setLayout(new GridLayout(3,2));
        p1.add(operando1);
        p1.add(op1);
```

```
        p1.add(operando2);
        p1.add(op2);
        p1.add(resultado);
        p1.add(result);

        p2.setLayout(new FlowLayout());
        p2.add(sum);
        p2.add(res);
        p2.add(mul);
        p2.add(div);
        p2.add(borrar);

        setLayout(new BorderLayout());
        add(p1,BorderLayout.CENTER);
        add(p2,BorderLayout.SOUTH);
        pack();
        show();

        sum.addActionListener(this);
        res.addActionListener(this);
        mul.addActionListener(this);
        div.addActionListener(this);
        borrar.addActionListener(this);

    }


     public void actionPerformed(ActionEvent e) {
        double x,y;
        x=Double.parseDouble(op1.getText());
        y=Double.parseDouble(op2.getText());

        if(e.getSource()==sum)
           result.setText((x+y)+"");

        else if(e.getSource()==res)
           result.setText((x-y)+"");

        else if(e.getSource()==mul)
           result.setText((x*y)+"");

        else if(e.getSource()==div)
            result.setText((x/y)+"");

             else {
                    op1.setText("");
                    op2.setText("");
                    result.setText("");
                    }
        }



     public static void main(String[] args) {

        Calculadora calculadora = new Calculadora();

        //esto es solo para que la ventana pueda cerrarse
        calculadora.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
```

```
        });

    }
}
```

Noten que en la definicion del listener de ventanas utilizamos WindowAdapter que es una clase abstracta que implementa WindowListener por lo que solo necesitamos definir el metodo que utilizaremos. La definición de la clase la hemos realizado de forma especial dentro del metodo para agregar la acción, esto se puede hacer con todos los listener pero no es recomendable cuando debamos definir muchos metodos del mismo porque el codigo se vueve demasiado sucio. La forma estandar de hacerlo es definiendo la clase aparte.

```
public static void main(String[] args) {

        Calculadora calculadora = new Calculadora();

        //esto es solo para que la ventana pueda cerrarse
        calculadora.addWindowListener(new escucha());
         class escucha extends WindowAdapter {
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        }

    }
```

25. Vectores

La siguiente tabla define los métodos ofrecidos por la clase Vector que permite realizar operaciones con vectores multi-dimensionales:

Ejemplo	Significado	encabezamiento
new Vector(a,n)	crea vector $\langle a_0, a_1,\ldots,a_{n-1}\rangle$	Vector(int[] x, int n)
a.valor(i)	Entrega valor de la componente de índice i	int valor(int i)
a.suma(b)	Suma vectorial: $\langle a_0+b_0, a_1+b_1,\ldots,a_{n-1}+b_{n-1}\rangle$	Vector suma(Vector x)
a.producto(b)	Producto vectorial: $\langle a_0b_0, a_1b_1,\ldots,a_{n-1}b_{n-1}\rangle$	Vector producto(Vector x)

a) Escribir los métodos de la clase vector de acuerdo a la siguiente declaración:

```
Class Vector{
    // representación
   protected final int N = 100;                 // dimensión máxima
   protected int[] v = new int [N];            // valores components
   protected int n;         // dimensión del vector (N° componentes)
}
```

b) Escribir la clase Vector1, que se derive de la clase Vector, pero que agregue el método productoPunto de acuerdo a la siguiente especificación:

Ejemplo	Significado	encabezamiento
a.productoPunto(b)	$a_0b_0 + a_1b_1 + \ldots + a_{n-1}b_{n-1}$	Int productoPunto(Vector x)

Solución:

Parte a:

```
class Vector{

  // variables de instancia
  protected final int N = 100;
  protected int[] v = new int[N];
  protected int n;

  // constructor
  public Vector(int[]x, int n){
    for (int i=0; i<n; ++i)
      this.v[i] = x[i];
    this.n = n;
  }

  public int valor(int i){
    return this.v[i];
  }
```

```
  public Vector producto(Vector x){
    int []aux = new int[n];
    for (int i=0; i<n; ++i)
      aux[i] = this.v[i] * x.v[i];
    return new Vector(aux,n);
  }
}
```

Parte b:

```
class Vector1 extends Vector{
  public Vector1(int[] x, int n){
    super(x,n);
  }

  public int productoPunto(Vector x){
    int suma = 0;
    Vector aux = this.producto(x);
    for (int i=0; i < n; ++i)
      suma+= aux.valor(i);
    return suma;
  }
}
```

26. El juego de los fósforos

El juego de los 13 fósforos consiste en dejar sobre una mesa 13 fósforos, uno al lado del otro. Dos jugadores van alternadamente sacando 1, 2 o 3 fósforos, hasta que sólo quede 1. Quien tiene que sacar el último fósforo en su turno pierde el juego.

Se le pide que implemente el juego de los 13 fósforos en un frame. Guíese por la siguiente figura:

Notas:

- El botón "Ok!" señala el fin del turno del jugador. Luego debe hacer su jugada el computador.
- Se debe señalar quien gana el juego modificando el label superior.

Solución:

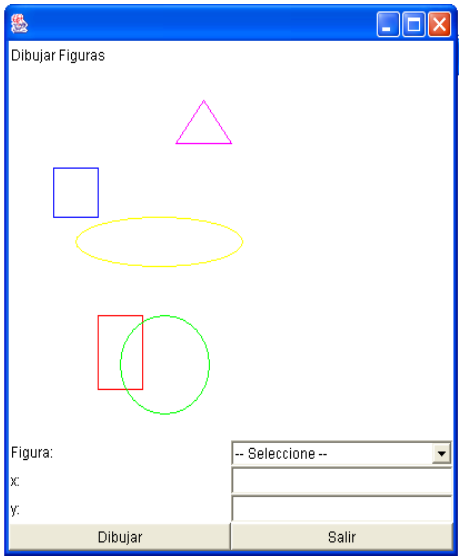
```
import java.awt.*;
import java.awt.event.*;

public class Juego13 extends Frame implements ActionListener {

	Label titulo;     //titulo señala quien ganó
	Button[] botones; //arreglo de botones que indican palitos
	Button listo;             //señala fin del turno del jugador
	int palitosQueQuedan = 13;//al principio todos los palitos

	public Juego13() {

		titulo = new Label("Juego de los 13 fósforos");

		botones = new Button[13];
		for (int i = 0; i < 13; i++)
			botones[i] = new Button("|");

		listo = new Button("Ok!");

		Panel p = new Panel();
		p.setLayout(new FlowLayout());
		for (int i = 0; i < 13; i++)
			p.add(botones[i]);

		this.setLayout(new BorderLayout());
		add(titulo, BorderLayout.NORTH);
		add(p, BorderLayout.CENTER);
		add(listo, BorderLayout.SOUTH);

		this.pack();
		this.show();
```

```
		//Agrego ActionListeners
		for (int i = 0; i < 13; i++)
			botones[i].addActionListener(this);
		listo.addActionListener(this);
	}

	public void actionPerformed(ActionEvent ae) {

	//CASO 1: Turno del Computador (Apretaron botón "Listo")
		if (ae.getSource() == listo) {
			/*
			* Hay 3 casos :
			*	- queda un palito --> el computador perdió
			*	- quedan 4,3,o 2 --> sacando 3,2,o 1
palitos, el computador ganó
			*	- otro --> saco algunos palitos al azar
			*/

			if (palitosQueQuedan == 1) {
				titulo.setText("GANAS TU!");
			}

			if (palitosQueQuedan <= 4) {
				for (int i=0; palitosQueQuedan>1; i++) {
					if
(botones[i].getLabel().equals("|")) {
						botones[i].setLabel("");
						botones[i].setEnabled(false);
						palitosQueQuedan--;
					}
				}
				titulo.setText("GANA EL PC!");
			}

			else {

				int azar=(int)Math.random()*3+1; //1,2,3

				for (int i=0; azar>0; i++) {
				if(botones[i].getLabel().equals("|")) {
						botones[i].setLabel("");
						botones[i].setEnabled(false);
						palitosQueQuedan--;
						azar--;
					} //if
				} //for
			} //else
		}
		//CASO 2: TURNO DEL JUGADOR
		else {
			/* Hay 2 casos :
			*	- queda un palito --> perdió el jugador
			*	- quedan mas --> el jugador saca palitos
			*		(confiamos en que saca a lo más 3)
			*/
			//Jugada persona
			if (palitosQueQuedan==1) {
				titulo.setText("GANA EL PC");
			}
			for (int i = 0; i < 13; i++) {
				if (ae.getSource() == botones[i]) {
					botones[i].setLabel("");
```

```
                                botones[i].setEnabled(false);
                                palitosQueQuedan--;
                            }
                        }
                }

        }

        public static void main(String args[]){
                Juego13 j=new Juego13();
        }

}
```

27. Tablero de Dibujo

Se desea tener una interfaz gráfica que permita al usuario dibujar distintas figuras geométricas. Para esto la interfaz debe permitir al usuario seleccionar qué figura desea dibujar e ingresar las coordenadas donde dibujarlo. El frame debe tener la apariencia de la figura.



Al presionar el boton "Dibujar" se dibujará la figura seleccionada en las coordenadas ingresadas por el usuario.

Indicación:

- La función *void drawLine(int x1, int y1, int x2, int y2)* dibuja una línea entre (x1,y1) y (x2,y2).
- La función void *drawRect(int x1, int y1, int a, int l)* dibuja un rectángulo cuyo vértice superior izquierdo es el punto (x1,y1) con ancho *a* y largo *l*.
- La función *void drawOval(int x1, int y1, int a, int l)* dibuja un óvalo que llena el rectángulo definido por el vértice superior izquierdo (x1,y1) , el ancho *a* y el largo *l*.

Solución:

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;

public class Dibujador extends Frame implements ActionListener {

	// componentes graficos
	Canvas canvas;
	Button dibujar;
	Button salir;
	Choice opciones;
	TextField x;
	TextField y;
	Label figura;
	Label tituloX;
	Label tituloY;

	public Dibujador() {
		// inicializar variables

```

```
            // boton
            dibujar = new Button("Dibujar");

            dibujar.addActionListener(this);
            salir = new Button("Salir");
            salir.addActionListener(this);

            // canvas
            canvas = new Canvas();
            canvas.setSize(400, 300);


            // opciones (combo box)
            opciones = new Choice();
            opciones.add("-- Seleccione --");
            opciones.add("Cuadrado");
            opciones.add("Rectangulo");
            opciones.add("Circulo");
            opciones.add("Ovalo");
            opciones.add("Triángulo");

            // label
            figura = new Label("Figura: ");
            tituloX = new Label("x: ");
            tituloY = new Label("y: ");

            // textfield
            x = new TextField(5);
            y = new TextField(5);


            // diagramar frame
            Panel p = new Panel();
            p.setLayout(new GridLayout(4, 2));


            // agregar componentes al panel
            p.add(figura);
            p.add(opciones);
            p.add(tituloX);
            p.add(x);
            p.add(tituloY);
            p.add(y);
            p.add(dibujar);
            p.add(salir);



            // fijar layout general
            setLayout(new BorderLayout());
            add(new Label("Dibujar Figuras"),BorderLayout.NORTH);
            add(p,BorderLayout.CENTER);
            add(canvas,BorderLayout.SOUTH);

      }

      public void actionPerformed(ActionEvent e) {
            // si se apreto boton salir

            if (e.getSource() == salir) {
                  System.exit(0);
```

```
            }

            // si se apreto boton dibujar

            else if (e.getSource() == dibujar) {

                  // obtener objeto graphic del canvas
                  Graphics graphics = canvas.getGraphics();

                  // obtener coordenadas
                  int valorX = Integer.parseInt(x.getText());
                  int valorY = Integer.parseInt(y.getText());

                  // obtener figura a dibujar
                  int eleccion = opciones.getSelectedIndex();
                  switch (eleccion) {
                        case 1 : // Cuadrado
                              graphics.setColor(Color.blue);
                        graphics.drawRect(valorX,valorY, 40, 40);
                              break;

                        case 2 : // Rectangulo
                              graphics.setColor(Color.red);
                        graphics.drawRect(valorX, valorY, 40, 60);
                              break;

                        case 3 : // Circulo
                              graphics.setColor(Color.green);
                        graphics.drawOval(valorX, valorY, 80, 80);
                              break;

                        case 4 : // Ovalo
                              graphics.setColor(Color.yellow);
                  graphics.drawOval(valorX, valorY, 150, 40);
                              break;

                        case 5 : // Triangulo
                              graphics.setColor(Color.magenta);
                              graphics.drawLine(valorX, valorY,
valorX+50, valorY);
                              graphics.drawLine(valorX+50,
valorY, valorX+25, valorY-35);
                              graphics.drawLine(valorX+25,
valorY-35, valorX, valorY);
                              break;


                  }
            }
      }



      public static void main(String[] args) {

            Dibujador disco = new Dibujador();
            disco.pack();
            disco.show();

      }
}
```

28. Encuesta Televisiva.

Se ha realizado una encuesta sobre televisión, y se necesita publicar los resultados en el diario de mañana. Sin embargo, sólo se tiene un archivo de texto (llamado "encuesta.txt"), donde cada línea contiene el nombre del encuestado (columnas 1 a 20), su sexo (M o F, en la columna 22) y su respuesta (columna 23). Las respuestas son un número 1, 2 o 3, correspondiente a Sí, No, y No sabe/No responde, respectivamente.

Se le pide que escriba un programa (usando arreglos) que imprima una tabla con la siguiente información:

Grupo	Sí	No	No sabe/No responde
Mujeres	45	25	5
Hombres	60	12	3

Solución:

```
import java.io.*;

public class Encuesta {

      public static void main(String args[]) throws IOException{
                  int resultados[][]=new int[2][3];//2 filas(M y F),3
                                          //columnas (Si, No, NS/NR)

            //Inicializo todos los valores como 0
            for (int i=0; i<resultados.length; i++)
                  for (int j=0; j<resultados[0].length; j++)
                        resultados[i][j]=0;
            String linea="";

            //Leo archivo y lleno arreglo
   BufferedReaderb=newBufferedReader(newFileReader("encuesta.txt"));
            while ((linea=b.readLine())!=null) {
                  int fila;
                  if (linea.charAt(21)=='M') //hombre
                        fila=0;
                  else fila=1;
   resultados[fila][Integer.parseInt(linea.substring(22,23))-  1]++;
            }

            //Imprimo resultados
            Console c=new Console();
            c.println("Grupo\t Si\t No\t NS/NR");
            c.print("Hombres\t");
            for (int j=0; j<resultados[0].length; j++) {
                  c.print(resultados[0][j]+"\t");
            }
            c.println();
            c.print("Mujeres\t");
            for (int j=0; j<resultados[0].length; j++) {
                  c.print(resultados[1][j]+"\t");
            }
      }
```

```
}
```

29. Pares Ordenados

Una secuencia de pares ordenados (x1,y1), (x2,y2), ... se puede representar a través de la clase Tabla que tiene definidos los siguientes métodos.

ejemplo	significado	encabezamiento
a.agregar(x,y)	agrega el par (x,y) a la tabla a	void agregar(int x, int y)
a.existe(y)	entrega true si existe un par con un valor y en la segunda coordenada	boolean existe(int y)
a.obtenerx(y)	entrega el valor de x del par (x,y)	int obtenerx(int y)
Tabla a=new Tabla()	crea un objeto (sin pares)	Tabla()

a) escriba un segmento de programa que utilice la clase anterior para:

- construir una tabla con los pares (1,1),(2,4),(3,9),(4,16), ...,(100,10000).
- leer un N° desde el teclado y escribir el valor de su raíz entera. Por ejemplo, si el N° es 36 escribe 6, pero si el N° es 37 escribe la frase SIN RAIZ ENTERA

b) Escriba el método obtenerx, suponiendo la siguiente representación:

```
       class Tabla{
       private final int N=1000;        //cantidad máxima de pares
       private int[  ] x=new int[N], y=new int[N]; //arreglos para
                                                    //pares ordenados
       private int n=0;                 //cantidad efectiva de pares
       . . .
}
```

Solución:

Parte a:

```
class TestTabla{

  public static void main(String[] args){

    // construir tabla

    Tabla t = new Tabla();
    for (int i=0; i < 100; ++i){
      t.agregar(i, i*i);
    }

    Console C = new Console();
    C.print("Ingrese numero: ");
    int num = C.readInt();
```

```
    if (t.existe(num)){

      C.println(t.obtenerx(num));

    }
    else{

      C.println("SIN RAIZ ENTERA");

    }
  }

}
```

Parte b:

```
class Tabla{

  private final int N=1000;     //cantidad máxima de pares

  private int[] x=new int[N], y=new int[N]; //arreglos para pares
//ordenados
  private int n=0;        //cantidad efectiva de pares

  public Tabla(){
    for (int i=0; i < N; ++i){
      x[i] = 0;
      y[i] = 0;
    }
    n = 0;
  }

  public void agregar(int a, int b){
    if (n < N){
      this.x[n] = a;
      this.y[n] = b;
      n++;
    }
  }

  public boolean existe(int b){
    for (int i=0; i < n; ++i){
      if (this.y[i] == b)
        return true;
    }
    return false;
  }

  public int obtenerx(int b){
    for (int i=0; i < n; ++i){
      if (this.y[i] == b)
        return this.x[i];
    }
    return -1;

  }
}
```

30. Mas Arreglos

a) Escriba un método de encabezamiento ***void insertar(String x,String[]y,int max,int n***) que inserte ***x*** en el lugar que le corresponde en el arreglo ***y*** que está ordenado ascendentemente. Los parámetros ***max*** y ***n*** representan la cantidad máxima y actual de elementos del arreglo ***y***.

Ejemplo: String[]a = new String[4]; a[0]="A"; a[1]="C"; a[2]="E";
insertar("B",a,4,3); //deja a={"A","B","C","E"}
insertar("D",a,4,4); //deja a={"A","B","C","D"}y se pierde el último valor ("E")

b)Utilice el método anterior en un programa que lea el archivo "palabras.txt" que contiene una cantidad indeterminada de palabras (cada una en una línea) y escriba las primeras 100 según el orden alfabético. Suponga que el archivo tiene al menos 100 líneas.

Solución:

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class Ordenar {

 public static void insertar(String x, String[] y, int max, int n) {
      //max -> cantidad máxima de elementos (tamaño del arreglo)
      //n--> numero actual de elementos del arreglo y

      //arreglo auxiliar del mismo tamaño que y
      String[] aux = new String[max];

      int i=0;

      //recorro todos los elementos del arreglo inicial que sean
     //menores que x
            for (i = 0; i < n; i++) {
              if (x.compareTo(y[i]) <= 0) //x < y --> debo insertar
                        break; //debo insertar x
              else
                        aux[i] = y[i];
            }

            // si i<max significa que "cabe" x
            if (i < max) {
                  //inserto x y muevo i en 1
                  aux[i++] = x;
                  //recorro el resto del arreglo
                  for (; i < Math.min(n+1,max); i++) {
                        aux[i] = y[i - 1];
                  }
            }
            //copio el arreglo de vuelta a y
            for (i = 0; i < Math.min(n+1,max); i++)
                  y[i] = aux[i];
```

```
	}

	public static void main(String args[]) throws IOException {
 BufferedReader in=newBufferedReader(newFileReader("palabras.txt"));
		Console c=new Console();
		String linea;
		String arreglo[]=new String[100];
		int contador=0;
		while((linea=in.readLine())!=null){
			insertar(linea,arreglo,100,contador);
			if (contador<100) contador++;
		}
		for (int i=0; i<100; i++)
			c.println(arreglo[i]);

	}
}
```

31.Ranking de Notas

Suponga que existe la clase Nota con la siguiente representación:

```
class Nota{

      public double nota;
      public String apellido;
      pulbic String nombre;

…//constructor y métodos
}
```

Los profesores de Computación desean llevar un ranking de los alumnos de su sección, según la nota promedio que llevan hasta el momento en el ramo. Los profesores tienen un arreglo con las notas de sus alumnos, sin embargo éste no tiene ningún orden en particular.
Implemente el método *void ordenar(Nota x[])* que ordena el arreglo que recibe como parámetro en orden decreciente de notas. Los alumnos que tengan la misma nota deben quedar ordenados alfabéticamente, primero por apellido y luego por nombre.
Ejemplo: En el arreglo de notas se tienen los objetos definidos por los valores:

```
3,5#Pérez#Juan
4,5#Aceval#Domingo
4,0#Carrasco#Marcela
4,5#Toledo#Rodrigo
6,0#Gómez#Jorge
4,0#Carrasco#Armando
```

Luego de aplicar el método ordenar, el arreglo queda:

```
6,0#Gómez#Jorge
4,5#Aceval#Domingo
4,5#Toledo#Rodrigo
4,0#Carrasco#Armando
4,0#Carrasco#Marcela
3,5#Pérez#Juan
```

Solución:

El metodo que utilizaremos para ordenar, tiene por nombre BubbleSort o algoritmo de la burbuja. Consiste en tomar dos valores del arreglo comparandolos entre si, en caso que el elemento que estamos comparando sea mayor a otro se intercambian los lugares y se continua. Al fin de una iteracion tendremos garantizado que al menos un elemento del arreglo estará ordenado y continuamos ordenando el resto del arreglo. Asi el arreglo estará ordenado en n iteraciones, donde n es el tamaño del arreglo.
Veamos un ejemplo:
Supongamos que nuetro arreglo es

4 – 3 - 5 – 2 – 1

Tenemos 5 elementos. Comenzamos comparando el primer elemento con el segundo. 4 es mayor que 3, asi que intercambiamos. Ahora tenemos:

3 – 4 – 5 – 2 – 1

ahora comparamos el segundo con el tercero. 4 es menor que 5, asi que no hacemos nada. Continuamos con el tercero y cuarto: 5 es mayor que 2. Intercambiamops y obtenemos:

3 – 4 – 2 – 5 – 1

Comparamos el cuarto con el quinto: 5 es mayor que 1. Intercambiamos nuevamente:

3 – 4 – 2 – 1 – 5

Ahora tenemos en la ultima posición de nuestro arreglo al numero mayor. Y continuamos ordenando el la parte del arreglo que falta (ya no llegamos hasta la quinta posición porque esta ordenada). Si seguimos aplicando este metodo obtenemos los siguientes cambios:

3 - 2 - 1 - 4 - 5

2 - 1 - 3 - 4 - 5

1 - 2 - 3 - 4 - 5

Y dejamos el arreglo ordenado.

En codigo java este algoritmo :

```
public static void ordenarNotasBurbuja(Nota[] a){

      Nota aux;
      for(int i=a.length-1; i>0; i--)
            for(int j=0; j<i; j++)
                  if(a[j].nota<a[j+1].nota ||
                      (a[j].nota==a[j+1].nota &&
                      (a[j].apellido.compareTo(a[j+1].apellido)>0
                    ||(a[j].apellido.compareTo(a[j+1].apellido)==0
                    && a[j].nombre.compareTo(a[j+1].nombre)>0))))){

                        aux=a[j];
                        a[j]=a[j+1];
                        a[j+1]=aux;
                  }

      }
```

```
}
```

Para la comparacion de los elementos utilizamos las reglas descritas en el enunciado. Primero comparamos por el valor de la nota, luego por apellido y en caso que fueran iguales comparamos por nombre. Noten que en cada iteración el valor de i va disminuyendo y la comparacion en el ciclo de j llega hasta el valor de i por lo que se va disminuyendo el tamaño del arreglo en que comparamos.
En principio en este control no se piden algoritmos eficientes de ordenamiento pero nunca esta de mas tenerlo solo en caso de necesitarlo.

32. Impuestos a productos (Pregunta 1 Control 2, 2003)

La siguiente tabla define los impuestos que se aplican a un producto para calcular su precio final:

Tipo de Producto	Arancel Importación	Impuesto alcoholes	IVA
Nacional	0%	0%	18%
Alcohol Nacional	0%	10%	18%
Importado	*%	0%	18%
Alcohol Importado	*%	10%	18%

* significa que el valor es distinto para cada producto.

De acuerdo a la tabla, el precio final de un alcohol importado se calcula aplicando primero el arancel de importación, segundo el impuesto a alcoholes y finalmente el IVA. Por ejemplo, un alcohol importado de precio inicial 100 y de arancel de importación de un 20% tendrá un precio final de 155 que se calcula ((100*1.20)*1.10)*1.18.
La siguiente clase permite almacenar y realizar operaciones con objetos que representan productos nacionales:

```
public class ProductoNacional {
	public double IVA = 0.18;
	protected int precioInicial; //protected se puede omitir
	protected String nombre, codigo;
	public ProductoNacional(String x, String y, int z){
		nombre = x;
		codigo = y;
		precioInicial = z;
	}
	public int obtenerPrecioInicial(){
		return precioInicial;
	}
	public int obtenerPrecioFinal(){
		return (int)(precioInicial*(1+ IVA));
	}
}
```

a) Declare clases distintas que extiendan la clase ProductoNacional de modo que permitan almacenar y operar con cada uno de los otros productos de la tabla.
b) Escriba un metodo que reciba un arreglo de productos y entrege la suma de los precios finales. Note que el arreglo contiene objetos de cualquiera de las clases.

Solución:

Parte a:

Clase alcoholNacional: solo necesitamos una variable adicional para almacenar el valor del impuesto a alcoholes el que debemos agregar en el calculo del precio final.

```
public class alcoholNacional extends ProductoNacional {
	protected double impAlcoholes=0.1;
	public alcoholNacional(String x, String y, int z){
		super(x,y,z);
	}
	public int obtenerPrecioInicial(){
		return precioInicial;
	}
	public int obtenerPrecioFinal(){
		return(int)((precioInicial*(1+impAlcoholes))*(1+ IVA));
	}
}
```

Clase importado: en esta clase necesitamos una varible para guardar el arancel de importación y dado que este valor depende del producto lo inicializaremos dentro del constructor de la clase. Ademas debemos agregar este impuesto al calculo del precio final del producto.

```
public class importado extends ProductoNacional {
	protected double arancelImportacion;
	public importado(String x, String y, int z, int arcl){
		arancelImportado=arcl/100.0;
		super(x,y,z);
	}
	public int obtenerPrecioInicial(){
		return precioInicial;
	}
	public int obtenerPrecioFinal(){
	  return (int)((precioInicial*(1+arancelImportado))*(1+ IVA));
	}
}
```

Clase alcoholImportado: en esta clase necesitamos variables para el impuesto a alcoholes y otra para el arancel de importación. Al igual que en la clase anterior el arancel es una cantidad variable por lo que la inicializamos en el constructor.

```
public class alcoholImportado extends ProductoNacional {
	protected double impAlcoholes=0.1;
	protected double arancel;
	public ProductoNacional(String x, String y, int z,int arcl){
		arancel=arcl/100.0;
		 super(x,y,z);
	}
	public int obtenerPrecioInicial(){
		return precioInicial;
	}
	public int obtenerPrecioFinal(){
return(int)(((precioInicial*(1+arancel))*(1+impAlcoholes))*(1+
IVA));
	}
}
```

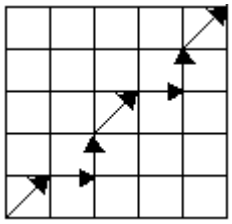
Parte b:

La clave de este problema es saber que metodo utilizar para obtener el precio final. La solucion es bastante sencilla, dado que creamos tres clases que extienden de una, definimos el arreglo recibido como parametro de tipo ProductoNacional y utilizamos el operador "intanceof" para verificar si el objeto almacenado se trata de uno de las clases extendidas. Asi hacemos un cast al objeto obtenido y llamamos al metodo obtenerPrecioFinal() correspondiente a su clase. Si probamos todos los casos de las clases hijas, entonces el objeto es de la clase productoNacional por lo que no necesitamos realizar un cast.

```
Public int sumaPrecios (ProductoNacional [] p){
   Int suma=0;
     for (int i=0 ; i<p.length ;i++){
      if (p[i] instanceof alcoholNacional)
            suma+=((alcoholNacional)p[i]).obtenerPrecioFinal();
      else if (p[i] instanceof importado)
                  suma+=((importado)p[i]).obtenerPrecioFinal();
      else if (p[i] instanceof alcoholImportado)
            suma+=((alcoholImportado)p[i]).obtenerPrecioFinal();
      else
                  suma+=p[i].obtenerPrecioFinal();
     }
    return suma;
}
```

33. Carrera de Flechas (Pregunta 2 Control 2, 2003)

Un juego llamado "carrera de flechas" se juega entre dos personas sobre un papel cuadriculado en el cual alternadamente los jugadores van dibujndo una flecha desde un punto al siguiente mas proximo ya sea hacia la derecha, hacia arriba o en diagonal (arriba derecha). Cada jugador debe dibujar su flecha desde el punto donde el jugador anterior termino de dibujar la suya. El jugador que parte lo hace dibujando una flecha desde el extremo inferior izquierdo. Gana el jugador que dibuja la flecha que llega al extremo superior derecho del cuadriculado.



a) La siguiente clase permite manipular y dibujar flechas:

```
public class Flecha{
       protected int x0, y0; //coordenadas de origen de la flecha
                             //protected se puede omitir
       public Flecha( int x , int y){
              x0=x;
              y0=y;
       }
       public void dibujar(int x , int y , G g){
       //G=Graphics o Console
       .
       .
       .
       }
}
```

Complete el metodo dibujar, de modo que trace una linea desde el origen (x0,y0) hasta (x , y) y cambie el origen a (x ,y). Nota: no necesita dibujar la punta de la flecha.

b) (Propuesto) Escriba un programa que utilizando la clase anterior, implemente una interfaz gráfica para jugar "carrera de flechas" entre dos personas en un cuadriculado de 10x10 cuadrados de 10x10 pixeles cada uno. Además debe proveer tres botones con las leyendas "horizontal", "vertical" y "diagonal" que permiten al jugador de turno dibujar la flecha que estime conveniente. El programa debe terminar cuando algún jugador llega al extremo superior derecho.
Nota: El cuadriculado y los botones pueden aparecer en la misma o en distintas ventanas, y en cualquier ubicación.

Solución:

Parte a:

```
class flecha{
.
.
.
public void dibujar(int x , int y , G g){
       g.drawLine(x0,y0,x,y);
              x0=x;
              y0=y;
}
}
```

34. Emparejando el mundo (Pregunta 3 Control 2, 2003)

a) Escriba el metodo emparejar que roerdena los elementos de un arreglo de strings produciendo los resultados indicados en los siguientes ejemplos:

String [] a={"2A","1B","1C","2D","2E","2F","1G","1H"};
emparejar(a); //resultado: a={"1B","2A","1C","2D","1G","2E","1H","2F"};

String []b ={"1A","2B","2C","2D","1E","2F"};
emparejar(b); //resultado: b={"1A","2B","1E","2C","2D","2F"};

Note que:

-Los elementos del arreglo son strings que comienzan con 1 o 2.
-El método reordena los elementos formando una secuencia de parejas de strings que comienzan con 1 y 2.
-Los elementos sin pareja deben quedar al final.
- Se respeta el orden relativo de los elementos que comienzan con 1 o 2 entre sí.

Indicacion: puede utilizar arreglos auxiliares.

b) Escriba un programa que utilice el metodo anterior para formar parejas entre personas de distinto sexo. El criterio para definir las parejas es que la mas pequeña de las mujeres se empareje con el mas pequeño de los hombres, y así sucesivamente.
La informacion de las personas esta contenida en el archivo "Personas.txt". Cada linea contiene un caracter que identifica el sexo (1 femenino, 2 masculino) y a continuación el nombre. Las personas estan ordenadas segun su estatura. El programa debe mostrar en cada linea los nombres de los integrantes de cada pareja. Suponga que hay igual número de hombres que de mujeres.

Solución:

Parte a:

```
Public void emparejar( String [] a){
      //separamos en u gurpo que comienza con 1 y otro con 2
      String [] aux1= new  String [a.length];
      String [] aux2= new String [a.length];
      Int f=0,m=0
      for (int i =0; i<a.length;i++){
            String valor=a[i];
```

```
                if(valor.substring(0,1).equals("1")){
                        aux1[f]=valor;
                        f++;
                }
                else {
                        aux2[m]=valor;
                        m++;
                }
        }
        //formamos las parejas
        int f1=0, m1=0,i;
        for(i=0; i< a.length; i++){
                if (f!=0 && i%2==0){
                        a[i]=aux1[f1];
                        f1++;
                        f--;
                }
                else if (m!=0 && i%2==1){
                        a[i]=aux2[m1];
                        m1++;
                        m--;
                }
                else break;
        }
        //imprimimos lo que falto
        if (f!=0)
                for( int j=f1; j<f+f1; i++)
                        a[i]=aux1[j];
                }
        else if (m!=0)
                for(int j=m1; j<m+m1 ;j++)
                        a[i]=aux2[j];
                        i++;
                }
}
```

Parte b:

```
class parejas{
        public static void main (String [] args)throws IOException {
                Console c=new Console();

        BufferedReader bf=new BufferedReader(new
FileReader("Personas.txt"));
                //cuento cuantas lineas tiene el archivo
                int cont=0;
                while((String linea=bf.readLine())!=null)
                        cont++;
                //creo el arreglo del largo debido
                String [] personas=new String [cont];
                bf=new BufferedReader (new FileReader("Personas.txt"));
                //lleno el arreglo
                cont=0;
                 while((String linea=bf.readLine())!=null){
                        personas[cont]=linea;
                        cont++;
                }
                //llamo la funcion y muestro las parejas
                emparejar(personas);
```

```
            for (int i=0;i+1<personas.length;i++)

            c.println(personas[i].substring(1)+"  es  pareja  de  "+
personas[i+1]);

      }
}
```