Trabajar con bytes en fichero bmp

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;


public class App {
    public static void main(String[] args) throws IOException {
        File archivo = new File("unaimagen.bmp");
        FileInputStream fis = new FileInputStream(archivo);


        // Saltar los primeros 2 bytes
        fis.read();
        fis.read();


        // Leer los siguientes 4 bytes que contienen el  tamaño de archivo  de la imagen
        byte[] bytesTamano = new byte[4];
        for (int i = 0; i < 4; i++) {
            bytesTamano[i] = (byte) fis.read();
        }
        int tamano = byteArrayToInt(bytesTamano, 0);
        System.out.println("El tamaño del fichero BMP es: " + tamano + " Kbytes");


        // Salta 12 bytes para posicionarnos en  anchura
        for (int i = 0; i < 12; i++) {
            fis.read();
        }


        // Leer los siguientes 8 bytes que contienen la anchura y altura de la imagen
        byte[] bytesAnchuraYAltura = new byte[8];
        for (int i = 0; i < 8; i++) {
            bytesAnchuraYAltura[i] = (byte) fis.read();
        }


        int anchura = byteArrayToInt(bytesAnchuraYAltura, 0);
        int altura = byteArrayToInt(bytesAnchuraYAltura, 4);


        System.out.println("La anchura de la imagen BMP es: " + anchura + " píxeles");
        System.out.println("La altura de la imagen BMP es: " + altura + " píxeles");


        fis.close();
    }


    public static int byteArrayToInt(byte[] bytes, int offset) {
        // Convertir  4 bytes de un array de bytes a un valor entero
       //si offset es 0 se trabaja con los bytes 3,2,1 y 0 del array
        return (bytes[offset + 3] & 0xFF) << 24 | (bytes[offset + 2] & 0xFF) << 16 | (bytes[offset + 1] & 0xFF) << 8
                | (bytes[offset] & 0xFF);
    }
}
```

Ordenando articulos(kOTLIN)

```
data class Articulo(val id: Int, val nombre: String, val precio: Double, val cantidad: Int)
val listaArticulos= listOf(
    Articulo(1, "Coca Cola", 8.0, 10),
    Articulo(2, "Pepsi", 8.0, 10),
    Articulo(3, "Fanta", 7.0, 10),
    Articulo(4, "Sprite", 6.0, 20),
    Articulo(5, "Manzanita", 5.0, 25),
    Articulo(6, "7up", 6.0, 30),
    Articulo(7, "Mirinda", 10.0, 25),

)

fun main() {
    //Ordenar e imprimir
    listaArticulos
        .filter { it.cantidad<=25 }
        .sortedWith(compareBy({it.cantidad},{it.precio},{it.nombre}))
        .forEach(::println)

}
```

FISGEITOR

Como sólo vamos a tener una clase Observable (la clase Fisgeitor), para simplificar, no hacemos un interface Observable aunque si se hace, mejor que mejor.

```
import java.util.ArrayList;
import java.util.List;

// Clase observable
class Fisgeitor {
    private List<UnidadEmergencia> unidadesEmergenciaNaranja = new ArrayList<>();
    private List<UnidadEmergencia> unidadesEmergenciaRoja = new ArrayList<>();

    public void registrarUnidadEmergenciaNaranja(UnidadEmergencia unidad) {
        unidadesEmergenciaNaranja.add(unidad);
    }

    public void registrarUnidadEmergenciaRoja(UnidadEmergencia unidad) {
        unidadesEmergenciaRoja.add(unidad);
    }

    public void eliminarUnidadEmergenciaNaranja(UnidadEmergencia unidad) {
        unidadesEmergenciaNaranja.remove(unidad);
    }

    public void eliminarUnidadEmergenciaRoja(UnidadEmergencia unidad) {
        unidadesEmergenciaRoja.remove(unidad);
    }

    public void notificarNaranja() {
        for (UnidadEmergencia unidad : unidadesEmergenciaNaranja) {
            unidad.actualizar();
        }
    }

    public void notificarRoja() {
        for (UnidadEmergencia unidad : unidadesEmergenciaRoja) {
            unidad.actualizar();
        }
    }


    public void generarAlerta(String codigoAlerta) {
        System.out.println("------------------");
        if(codigoAlerta.startsWith("Naranja")){
            System.out.println("Se ha detectado una alerta "+codigoAlerta+".");
            notificarNaranja();


        }else{
            System.out.println("¡¡peligro!!  alerta "+codigoAlerta+".");
            notificarRoja();
        }

    }

}

// estructura Observadores
interface UnidadEmergencia {
    void actualizar();
}

// Implementación de unidades de emergencia
// todos los observadores deberían sobreescribir  equals y hashcode para eliminar como sugiere test
```

```
//para simplificar líneas sólo se hace en clase Psicologos que es la única que lo requiere en los test
class Policia implements UnidadEmergencia {
    @Override
    public void actualizar() {
        System.out.println("Policía en camino.");
    }
}

class Ambulancia implements UnidadEmergencia {
    @Override
    public void actualizar() {
        System.out.println("Ambulancia en camino.");
    }
}

class Bomberos implements UnidadEmergencia {
    @Override
    public void actualizar() {
        System.out.println("Bomberos en camino.");
    }
}

class Geos implements UnidadEmergencia {
    @Override
    public void actualizar() {

        System.out.println("Geos en camino.");
    }
}

class Psicologos implements UnidadEmergencia {
    String nombreGabinete;
    public Psicologos(String nombreGabinete){
        this.nombreGabinete=nombreGabinete;
    }
    @Override
    public void actualizar() {

        System.out.println("Psicologos en camino.");
    }
    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + ((nombreGabinete == null) ? 0 : nombreGabinete.hashCode());
        return result;
    }
    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Psicologos other = (Psicologos) obj;
        if (nombreGabinete == null) {
            if (other.nombreGabinete != null)
                return false;
        } else if (!nombreGabinete.equals(other.nombreGabinete))
            return false;
        return true;
    }

}
```

PULPO A FARTAR

esta pregunta se presta para que haya muchas soluciones y todas igual de buenas, por ejemplo, la siguiente solución usa una cola pero también pudimos haber usado en lugar de la cola un array y haberlo tratado circularmente.

```
import java.util.Arrays;
import java.util.LinkedList;
import java.util.Queue;
import java.util.Scanner;
import java.util.TreeSet;


public class App {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String platos = sc.nextLine();


        String[] personas = sc.nextLine().split(" ");
        Queue<String> cola = new LinkedList<>(Arrays.asList(personas));
        TreeSet<String> comieronCabeza = new TreeSet<>();


        // recorrer platos
        for (int i = 0; i < platos.length(); i++) {
            char plato = platos.charAt(i);
            String comensal = cola.remove();
            if (plato == 'c') {
                comieronCabeza.add(comensal);
            }
            cola.add(comensal);
        }
        System.out.println(comieronCabeza);
    }


}
```