Manual Tacnico de Metodos:

En este documento se explicara a mas detalle los metodos mas complejos que contiene el programa de paqueteria.

Empecemos por la clase de controlador de bodega ya que es un de las clases con la funcionalidad mas importante del programa.En esta clase encontramos un total de 2 metodos el primer metodo y son llamados en el implemento de runneable el primero de los dos es el de obtenerPaquetesPorPriorizacion:

```
private ArrayList<Paquete> obtenerPaquetesSegunPriorizacion() {
    ArrayList<Paquete> paquetesOrdenados = new ArrayList<>();
    ArrayList<Paquete> paquetesEnBodega = ControladorDB.obtenerPaquetesPorEstado(1);
    for (int i = 0; i < paquetesEnBodega.size(); i++) {
        if (paquetesEnBodega.get(i).isPriorizado()) {
            paquetesOrdenados.add(paquetesEnBodega.get(i));
        }
    }
    for (int i = 0; i < paquetesEnBodega.size(); i++) {
        boolean aux = true;
        for (int j = 0; j < paquetesOrdenados.size(); j++) {
            if (paquetesEnBodega.get(i).getCodigo() == paquetesOrdenados.get(j).getCodigo()) {
                aux = false;
            }

        }
        if (aux) {
            paquetesOrdenados.add(paquetesEnBodega.get(i));
        }
    }
    return paquetesOrdenados;
}
```

En el que el objetivo es de que se organizen todos los paquetes que esten en bodega para que los que estan priorizados esten de primero en la lista, creado temporalmente un ArrayList de paquetes en donde se guardaran las ordenaciones para luego devolverlo ya ordenado y con todos los paques. Este metodo cuenta con una condicion en el ciclo que verifica si el paquete esta priorizado y si es asi se guarda primer en el ArrayList temporal.
Luego de ordenar los paquetes de la forma posterior se verifica si se pueden processar los paquetes en el orden de priorizacion con el metodo siguiente:

```
private void buscarTransferencias(){
    for (int i = 0; i < paquetes.size(); i++) {
        PuntoDeControl puntoSiguiente = ControladorDB.obtenerPuntosPorRuta(paquetes.get(i).getRuta().getCodigo()).get(0);
        if (TransferenciasDB.obtenerPaquetesPorPunto(puntoSiguiente.getCodigo()
        ).size()<puntoSiguiente.getCapacidad()) {
            TransferenciasDB.ingresarPaqueteDesdeBodega(paquetes.get(i), puntoSiguiente);
        }
    }
}
```

En el que recorremos todo el ArrayLIst de Paquetes que estan en la bodega, obteniendo para cada una un punto de control siguiente y verificamos si ese punto de control esta disponible para procesar.
Todo esto como anteriormente dicho esta la implementacion de runneable lo que significa que los podemos llamar en un hilo diferente a la principal evitando asi la incapacidad de trabajar mientras se ejecuta el metodo.
Para este programa es necesario que los diferentes tipos de objetos tengan una manera de diferenciarse de los demas y para eso tenemos la clase GeneradorDeCodigos con clases estaticas que nos proveen de codigos para diferentes objetos como por ejemoplo:

```
    public static int generarCodigoRuta() {
        int codigo = 110000;
        if (ControladorDB.obtenerCodigoDeRutas() != null) {
            codigo = codigo + ControladorDB.obtenerCodigoDeRutas().size();
        }

        while (ControladorDB.obtenerRutas(codigo) != null) {
            codigo++;
        }
        return codigo;
    }
```

En el imagen superior se genera el codigo para una ruta la cual siempre empezara con 11 , para que no se repita este codigo se verifica la cantidad de rutas que ya existen y verificar que no se repita este codigo, aumentando cada vez que lo hace. Muy similar es el metodo de abajo:

```
public static int generarCodigoPuntoDeControl() {
    int codigo = 220000;
    if (ControladorDB.obtenerCodigoDePuntosDeControl() != null) {
        codigo = codigo + ControladorDB.obtenerCodigoDePuntosDeControl().size();
    }

    while (ControladorDB.obtenerRutas(codigo) != null) {
        codigo++;
    }
    return codigo;
}
```

Pero en este caso los codigos para puntos de control empezaran con 22.
Y asi tambien podemos ver el siguiente ejemplo en el cual se genera un codigo de punto de control pero al excepcion del anterior, se manda como parametro un ArrayList de Puntos De Control que no son mas que puntos que aun no se guardan en la dase de datos pero que ya existen como objetos en memoria, asi que se debe evitar la repiticion de codigo con estos puntos tambien, como se ve en la figura siguiente:

```java
public static int generarCodigoPuntoDeControl(ArrayList<PuntoDeControl> puntos) {
    int codigo = 220000;
    if (ControladorDB.obtenerCodigoDePuntosDeControl() != null) {
        codigo = codigo + ControladorDB.obtenerCodigoDePuntosDeControl().size();
    }

    while (ControladorDB.obtenerRutas(codigo) != null) {
        codigo++;
    }
    if (puntos != null) {
        boolean auxiliar;
        do {
            auxiliar = false;
            for (int i = 0; i < puntos.size(); i++) {
                if (codigo == puntos.get(i).getCodigo()) {
                    auxiliar = true;
                    codigo++;
                }

            }
        } while (auxiliar);
    }

    return codigo;
}
```

Este misma metodologia de genenracion de codigo lo hacemos para los destinos paquetes y Clientes que son objetos fundamentales que debemos que mencionar.

En conjunto con los metodos de la bodega es de suma importancia de las transferencias de los paquetes de un punto de control a otro, y para ello nos encontramos con la clase TransferenciasDB que no es mas que un controlador de la base de datos, esta clase cuenta con su propia connetion a la DB.
Primero tenemos un metodo que cambia los estados de los paquetes, estos estados representan lo siguiente:
1-Bodega
2-En Ruta
3-En Destino
4-Retirado
Para el cambio de estos estados nos encontramos con el siguiente metodo que usa un PrepareStatment.

```java
public static void moverEstadoDePaquete(int codigo) {
    try {
        PreparedStatement declaracionPreparada = coneccion2.prepareStatement(STATEMENT_SUBIR_ESTADO_POR_CODIGO);
        declaracionPreparada.setString(1, String.valueOf(codigo));
        declaracionPreparada.executeUpdate();
    } catch (SQLException e) {
        System.out.println("Error Al mover");
    }

}
```

Todos los statemente de esta clase tienen una forma similar a la de la sigiente la cuual se usa en el metodo anterior de moverEstadoDePaquete.

```
//Statement Updates
private final static String STATEMENT_PAQUETE_ESTADO_POR_CONTROL = "UPDATE Paquete SET estado =estado+1 WHERE codigo =?";
```

Tambien en esta clase nos podemos topar con el metodo que devuelve todos los paquetes que se encuentran en cola de un punto de control, con el simple hecho de ingresar el codigo del punto y compararlo con el que tiene el paquete, si estos concuerdan el codigo del paquete se envia como parametro para el metodo verificacion de Paquete que devuelve el objeto apquete de ese codigo en especifico para luego guardarlos en un ArrayList de todos los pauqetes de ese PuntoDeControl como se ve en la figura siguiente:

```
public static ArrayList<Paquete> obtenerPaquetesPorPunto(int codigoPunto) {
    ArrayList<Paquete> codigos = new ArrayList();
    try {
        PreparedStatement declaracionPreparada = coneccion2.prepareStatement(STATEMENT_PAQUETE_POR_PUNTO);
        declaracionPreparada.setString(1, String.valueOf(codigoPunto));
        ResultSet resultado2 = declaracionPreparada.executeQuery();
        while (resultado2.next()) {
            codigos.add(verificarPaquete(resultado2.getInt("codigo")));
        }
    } catch (SQLException e) {
        System.out.println("Error SQL");
    }
    return codigos;
}
```

Muy similar es tambien la de obtenerPaquetesActivosPorRuta en la que envez de mandar como parametro un codigo punto con el anterior, se envia un codigo de Ruta y este metodo nos devuelve un ArrayList de Rutas.

Tambien en esta clase nos encontramos con metodos aun mas complejas como la siguiente que es la de ProcesarPaquete que tienen como parametro un objeto Paquete y un Objeto Tarifa que se encarga de procesar el paquete que se envia como parametro a su siguiente punto de control o destino:

```java
public static void procesarPaquete(Paquete paquete, float tarifa) {
    //En caso de que el paquete ya este en su ultimo punto de control
    try {
        coneccion2.setAutoCommit(false);
        PreparedStatement declaracionPaqueteCodigo = coneccion2.prepareStatement(STATEMENT_UPDATE_CODIGO_PUNTO);
        PreparedStatement declaracionPaqueteNumero = coneccion2.prepareStatement(STATEMENT_UPDATE_NUMERO_COLA);
        PreparedStatement declaracionPrecio = coneccion2.prepareStatement(STATEMENT_UPDATE_PRECIO_PERDIDO);

        if (paquete.getPunto().getNumero() == ControladorDB.obtenerPuntosPorRuta(paquete.getRuta().getCodigo()).size()) {
            moverEstadoDePaquete(paquete.getCodigo());
            registrarRetiroDePaquete(paquete.getCodigo());//Obtenemos las paquetes de una ruta en especifico que no esten activo
            declaracionPaqueteCodigo.setString(1, null);//Obtenemos las paquetes de una ruta en especifico que no esten activos
            declaracionPaqueteNumero.setString(1, null);
        } else {
            int codigoSiguientePunto = obtenerSiguientePunto(paquete.getPunto()).getCodigo();
            declaracionPaqueteCodigo.setString(1, String.valueOf(codigoSiguientePunto));
            declaracionPaqueteNumero.setString(1, String.valueOf(obtenerPaquetesPorPunto(codigoSiguientePunto).size() + 1));
        }
        declaracionPrecio.setString(1, String.valueOf(tarifa));
        declaracionPrecio.setString(2, String.valueOf(paquete.getCodigo()));
        declaracionPaqueteCodigo.setString(2, String.valueOf(paquete.getCodigo()));
        declaracionPaqueteNumero.setString(2, String.valueOf(paquete.getCodigo()));
        declaracionPrecio.executeUpdate();
        declaracionPaqueteCodigo.executeUpdate();
        declaracionPaqueteNumero.executeUpdate();
        coneccion2.commit();
        coneccion2.setAutoCommit(true);
    } catch (SQLException ex) {
        System.out.println("Error Al Guardar");
        try {
            coneccion2.rollback();
        } catch (SQLException ex1) {
            System.out.println("Error RollBack");
        }
    }
```

En el que podemos ver que se deben de actualizar varias cosas en la misma connetion asi que temporalmente se ingresa un false al AutoCommit para luego regresarlo a su estado anterior.
Tambien podemos ver en este metodo nos podemos encontrar con la condicion de ubicación del paquete actialmente por ejemploo el paquete puede estar en su ultima fase (Listo para el Destino) en este caso se debe de ingresar los valores de null a los atributos del paquete (numero en cola y codigo de punto) asi como se muestra en la funcion anterior. Si el paquete no esta en la fase anterior se debe de buscar el punto siguiente que tiene la ruta en el que esta nuestro paquete, para hallar el punto siguiente se cuenta con la siguiente funcion que se encuentra en este misma clase:

```java
public static PuntoDeControl obtenerSiguientePunto(PuntoDeControl puntoAnterior) {
    ArrayList<PuntoDeControl> puntosDeRuta = ControladorDB.obtenerPuntosPorRuta(puntoAnterior.getCodigoRuta());
    return puntosDeRuta.get(puntoAnterior.getNumero());
}
```

En el que se necesita un punto anterior para cargar los puntos de esa misma ruta y devolver el punto siguiente a la secuencia de la ruta

En los metodos de la bodega pudimos observar un llamamiento de un metodo con el nombre de ingresoDePaqueteDesdeBodega que se encuentra en esta clase de transferenciaDB ya que la clase ControladorDeBodega no cuenta con una connetion tiene que recorrir a este metodo para registrar un pauete desde la bodega hacia un punto de control que le corresponde con la ayuda del metodo anterior que obtiene el punto

siguiente para enciar este punto como parametro del metodo, este metodo tiene la siguiente estructura:

```java
public static void ingresarPaqueteDesdeBodega(Paquete paquete, PuntoDeControl puntoSiguinte) {
    try {
        coneccion2.setAutoCommit(false);
        moverEstadoDePaquete(paquete.getCodigo());
        //Actualiza el codigo de punto
        PreparedStatement declaracionPaqueteCodigo = coneccion2.prepareStatement(STATEMENT_UPDATE_CODIGO_PUNTO);
        //Ingresa el numero en cola
        PreparedStatement declaracionPaqueteNumero = coneccion2.prepareStatement(STATEMENT_UPDATE_NUMERO_COLA);
        declaracionPaqueteCodigo.setString(1, String.valueOf(puntoSiguinte.getCodigo()));
        declaracionPaqueteNumero.setString(1, String.valueOf(obtenerPaquetesPorPunto(puntoSiguinte.getCodigo()).size() + 1));
        declaracionPaqueteCodigo.setString(2, String.valueOf(paquete.getCodigo()));
        declaracionPaqueteNumero.setString(2, String.valueOf(paquete.getCodigo()));
        declaracionPaqueteCodigo.executeUpdate();
        declaracionPaqueteNumero.executeUpdate();
        coneccion2.commit();
        coneccion2.setAutoCommit(true);
    } catch (SQLException ex) {
        try {
            coneccion2.rollback();
        } catch (SQLException ex1) {
            System.out.println("Error RollBack");
        }
    }
}
```

En el que podemos ver que se deben de actualizar varias cosas en la misma connetion asi que temporalmente se ingresa un false al AutoCommit para luego regresarlo a su estado anterior.

Ya que es de suma importancia el manejo de los ingresos y gastos el metodo para obtener la ganancia en una ruta en especifco es la siguiente

```java
public static float obtenerGananciasPorRuta(int codigoRuta) {
    float ganancias = 0;
    try {
        PreparedStatement declaracionPreparada = coneccion2.prepareStatement(STATEMENT_PAQUETE_POR_RUTA_ACTIVOS);
        declaracionPreparada.setString(1, String.valueOf(codigoRuta));
        ResultSet resultado2 = declaracionPreparada.executeQuery();
        while (resultado2.next()) {
            ganancias = ganancias + verificarPaquete(resultado2.getInt("codigo")).getPrecioPagado();
        }
        declaracionPreparada = coneccion2.prepareStatement(STATEMENT_PAQUETE_POR_RUTA_SALIDOS);
        declaracionPreparada.setString(1, String.valueOf(codigoRuta));
        ResultSet resultado3 = declaracionPreparada.executeQuery();
        while (resultado3.next()) {
            ganancias = ganancias + verificarPaquete(resultado3.getInt("codigo")).getPrecioPagado();
        }
    } catch (SQLException e) {
        System.out.println("Error SQL");
    }
    return ganancias;
}
```

En el cual devolvemos un float la cual es la suma de todos los precios pagados de los paquetes de dicha ruta. En el que podemos ver que se deben de actualizar varias cosas en la misma connetion asi que temporalmente se ingresa un false al AutoCommit para luego regresarlo a su estado anterior. Algo siminlar es su metodo opues que el de obtener las perdidas de una ruta en especifica pero a excepcion del anterior se otendra el precioPerdido de cada Paquete como se ve en el imagen siguiente:

```java
public static float obtenerPerdidaPorRuta(int codigoRuta) {
    float perdida = 0;
    try {
        PreparedStatement declaracionPreparada = coneccion2.prepareStatement(STATEMENT_PAQUETE_POR_RUTA_ACTIVOS);
        declaracionPreparada.setString(1, String.valueOf(codigoRuta));
        ResultSet resultado2 = declaracionPreparada.executeQuery();
        while (resultado2.next()) {
            perdida = perdida + verificarPaquete(resultado2.getInt("codigo")).getPrecioPerdido();
        }
        declaracionPreparada = coneccion2.prepareStatement(STATEMENT_PAQUETE_POR_RUTA_SALIDOS);
        declaracionPreparada.setString(1, String.valueOf(codigoRuta));
        ResultSet resultado3 = declaracionPreparada.executeQuery();
        while (resultado3.next()) {
            perdida = perdida + verificarPaquete(resultado3.getInt("codigo")).getPrecioPerdido();
        }
    } catch (SQLException e) {
        System.out.println("Error SQL");
    }
    return perdida;
}
```

En varias ocaciones debemos de filtrar estos mismo resultado según fechas que ingresa el usuario y para ello contamos con el siguietne metodo en el cual enviamos la fecha de inicio y la final como parametro y estas fechas las comparamos con la fecha de ingreso de cada paquete, y si esta fecha se encuentra en medio de las filtraciones se agregar al float ganancia que se devuelve al recorrer todos los pauqetes de dicha ruta.

```java
public static float obtenerGananciasPorRutaFecha(int codigoRuta, LocalDateTime inicio, LocalDateTime fin) {
    float ganancias = 0;
    ArrayList<Paquete> paquetesFecha = obtenerPaquetesPorRutaFecha(codigoRuta, inicio, fin);
    for (int i = 0; i < paquetesFecha.size(); i++) {
        ganancias = ganancias + paquetesFecha.get(i).getPrecioPagado();
    }
    return ganancias;
}
```

Ademas de filtrar los resultados según fechas tambien en ocaciones debemos de filtrarlos por su codigo en especifico por ejemplo en el siguiente metodo debemos de ordenar el ArrayList puntos que es una lista de puntos conforme al codigo ingresado en este caso el String palabra y para ello creamos un arrayList Temporal que sera el de aux que nos sirvira para guardar los puntos que entran en la condicion de tener el mismo codigo que se ingresa como ‘palabra’.
Ya con la lista aux ordenada simplemente se iguala a la original para luego mostrar los PuntosDeControl en una tabla, como se muestra en la siguiente figura:

```java
private void buscar(String palabra) {
    if (!"".equals(palabra)) {
        ArrayList<PuntoDeControl> aux = new ArrayList<>();
        for (int i = 0; i < puntos.size(); i++) {
            if (String.valueOf(puntos.get(i).getCodigo()).startsWith(palabra)) {
                aux.add(puntos.get(i));
            }
        }
        boolean auxiliar;
        for (int i = 0; i < puntos.size(); i++) {
            auxiliar = true;
            for (int j = 0; j < aux.size(); j++) {
                if (puntos.get(i).getCodigo() == aux.get(j).getCodigo()) {
                    auxiliar = false;
                }
            }
            if (auxiliar) {
                aux.add(puntos.get(i));
            }
        }
        puntos = aux;
    }
}
```

Ya que es de suma importancia el manejo de los ingresos y gastos el metodo para obtener la ganancia en una ruta en especifco es la siguiente en el cual devolvemos un float la cual es la suma de todos los precios pagados de los paquetes de dicha ruta. En el que podemos ver que se deben de actualizar varias cosas en la misma connetion asi que temporalmente se ingresa un false al AutoCommit para luego regresarlo a su estado anterior.

```java
public static float obtenerGananciasPorRuta(int codigoRuta) {
    float ganancias = 0;
    try {
        PreparedStatement declaracionPreparada = coneccion2.prepareStatement(STATEMENT_PAQUETE_POR_RUTA_ACTIVOS);
        declaracionPreparada.setString(1, String.valueOf(codigoRuta));
        ResultSet resultado2 = declaracionPreparada.executeQuery();
        while (resultado2.next()) {
            ganancias = ganancias + verificarPaquete(resultado2.getInt("codigo")).getPrecioPagado();
        }
        declaracionPreparada = coneccion2.prepareStatement(STATEMENT_PAQUETE_POR_RUTA_SALIDOS);
        declaracionPreparada.setString(1, String.valueOf(codigoRuta));
        ResultSet resultado3 = declaracionPreparada.executeQuery();
        while (resultado3.next()) {
            ganancias = ganancias + verificarPaquete(resultado3.getInt("codigo")).getPrecioPagado();
        }
    } catch (SQLException e) {
        System.out.println("Error SQL");
    }
    return ganancias;
}
```

Ademas de TranseferenciaDB tambien contamos con otra clase que tiene un connetion que tambien se encarga de controlar la base de datos y esta es la clase ControladorDB

estas acciones en la base de datos estan mas asociados a la consulta y al insert de elementos de nuestra base de datos, por ejemplo una de sus funciones mas importantes es la de guardar los paquetes en nuestra DB y para esto cuenta con la funcion estatica de guardarPaquete.
GuardarPaquete es un metodo que recibe como parametro un objeto paquete para luego hacer un insert de todos los valores de dicho paquete que nos podran servir mas adelante, la estructura de este metodo es la siguiente:

```java
public static void guardarPaquete(Paquete paquete) {
    try {
        PreparedStatement declaracionPreparada = coneccion.prepareStatement(STATEMENT_GUARDAR_PAQUETE);
        declaracionPreparada.setString(1, String.valueOf(paquete.getCodigo()));
        declaracionPreparada.setString(2, String.valueOf(paquete.getPeso()));
        declaracionPreparada.setString(3, String.valueOf(paquete.getRuta().getCodigo()));
        declaracionPreparada.setString(4, String.valueOf(paquete.getCliente().getCodigo()));
        if (paquete.isPriorizado()) {
            declaracionPreparada.setString(5, "1");
        } else {
            declaracionPreparada.setString(5, "0");
        }
        declaracionPreparada.setString(6, String.valueOf(paquete.getFechaIngresado()));
        declaracionPreparada.setString(7, null);
        declaracionPreparada.setString(8, String.valueOf(paquete.getNumeroEnCola()));
        declaracionPreparada.setString(9, null);
        declaracionPreparada.setString(10, String.valueOf(paquete.getEstado()));
        declaracionPreparada.setString(11, String.valueOf(paquete.getPrecioPerdido()));
        declaracionPreparada.setString(12, String.valueOf(paquete.getPrecioPagado()));
        declaracionPreparada.executeUpdate();
    } catch (SQLException e) {
        System.out.println("Error Al Guardar");
    }
}
```