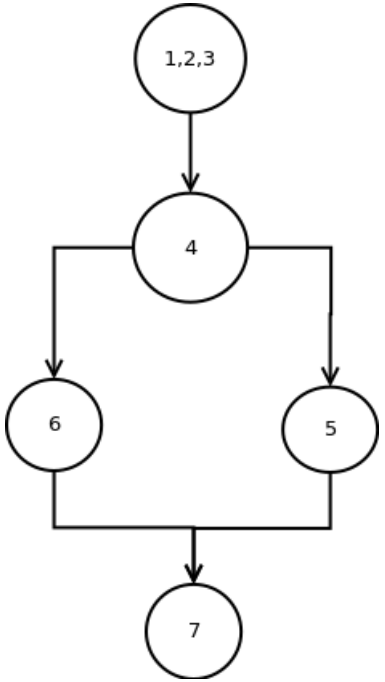
Aclaración: debido a que se utiliza un lenguaje de programación orientado a objetos y la complejidad ciclomática fue desarrollada para ser utilizada en base a un lenguaje de programación estructurado, en aquellas ocasiones en que en algún método aparezca un lanzamiento de excepción la fórmula de la CC tenderá a ser errónea o incluso a dar distintos resultados entre las posibles ecuaciones. Por esto, es que en estos casos, se ignora el resultado de las ecuaciones y se desarrollan los caminos especificados para recorrer completamente el código. En aquellos métodos que surja esta inconveniencia se lo aclarará nuevamente.

Paquete datos

Clase Observación

public int compareTo(Object object)	
{	
int ret;	1
Observacion otra = (Observacion) object;	2
int aux = this.tema.compareTo(otra.tema);	3
if (aux != 0)	4
ret = aux;	5
else	
ret = this.fechaObservacion.compareTo(otra.fechaObservacion);	6
return ret;	7
}	



CC = número de arcos (5) – número de nodos (5) + 2 = 2
CC = nodos condicionales (1) + 1 = 2
CC = número de regiones cerradas (1) + 1 = 2

Camino 1: 1 – 2 – 3 – 4 – 6 – 7
Camino 2: 1 – 2 – 3 – 4 – 5 – 7

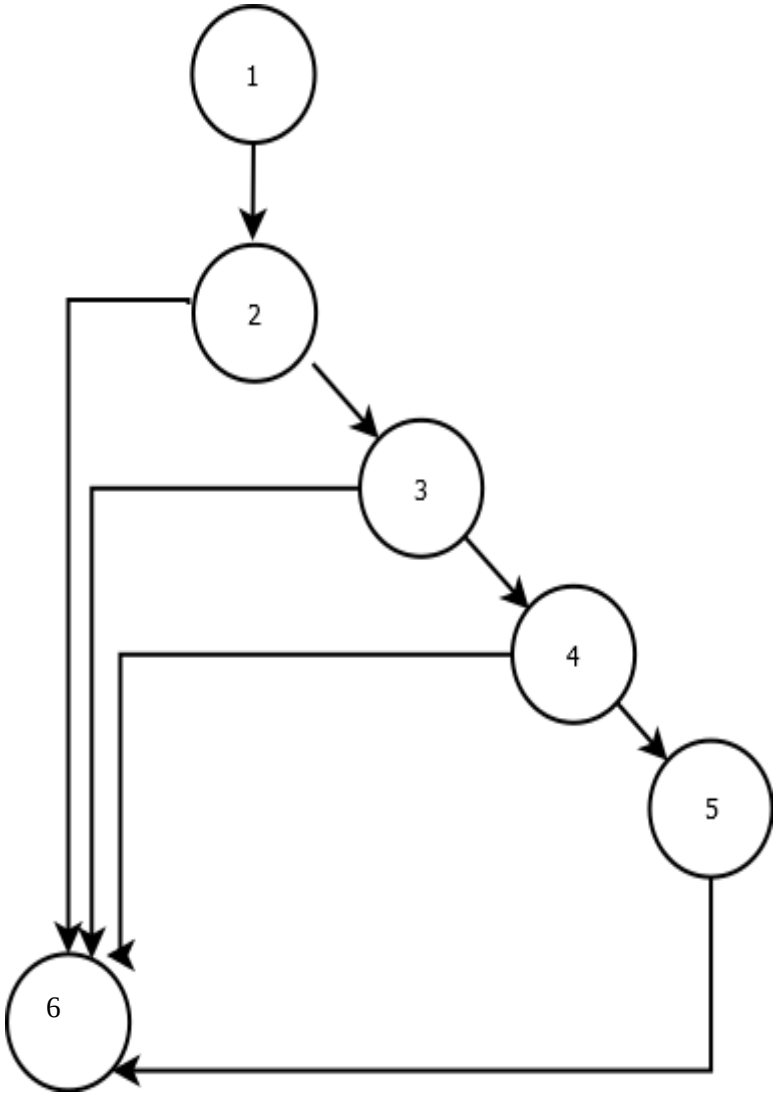
ID	Objetivo de la prueba	Datos de entrada	Procedimiento	Salida esperada	Resultado
T1	Verificar el camino 1	Observación con el mismo tema y fechaObservación menor	• Compara los temas y guarda el resultado en una variable • Como ambos temas son iguales, entra en el else • Compara ambas fechas, y guarda el valor en la variable de retorno • Devuelve el resultado	Como la fecha de la observación de entrada es menor, el método devuelve un valor positivo	Correcto
		Observación con el mismo tema y fechaObservacion igual		Como la fecha de la observación de entrada es igual, el método devuelve un cero	Correcto
		Observación con el mismo tema y fechaObservacion mayor		Como la fecha de la observación de entrada es menor, el método devuelve un valor negativo	Correcto
T2	Verificar el camino 2	Observación con un tema “mayor”	• Compara los temas y guarda el resultado en una variable • Como los temas no son iguales, entra en el if y guarda el valor en la variable de retorno • Devuelve el resultado	Como la observación es considerada “mayor” (en términos de comparación de String) el método devuelve un valor negativo	Correcto
		Observación con un tema “menor”		Como la observación es considerada “menor” (en términos de comparación de String) el método devuelve un valor positivo	Correcto

Clase Pedido

public boolean verificaNull()
{
  return (this.fechaPedidoAceptado != null && this.fechaPropuestaProduccion != null && this.fechaDefinitiva != null);
}
	1

El método anterior se desdobla de la siguiente forma para un mejor análisis de caja blanca:

public boolean verificaNull() {	
boolean ret = false;	1
if(this.fechaPedidoAceptado != null)	2
if(this.fechaPropuestaProduccion != null)	3
if(this.fechaDefinitiva != null)	4
ret = true;	5
return ret;	6
}	



CC = número de arcos (8) – número de nodos (6) + 2 = 4
CC = nodos condicionales (3) + 1 = 4
CC = número de regiones cerradas (3) + 1 = 4

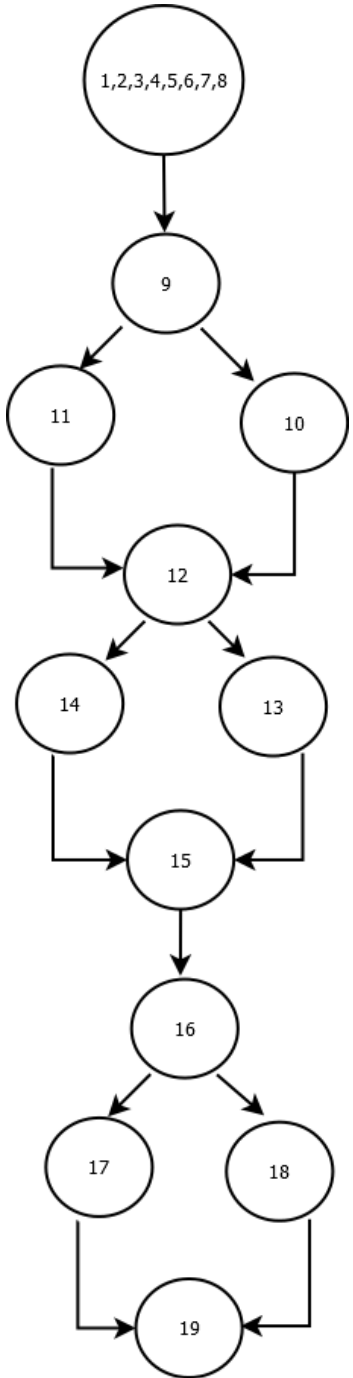
Camino 1: 1-2-6
Camino 2: 1-2-3-6
Camino 3: 1-2-3-4-6
Camino 4: 1-2-3-4-5-6

ID	Objetivo de la prueba	Datos de entrada	Procedimiento	Salida esperada	Resultado
T1	Verificar el camino 1	fechaPedidoAceptado = null	• Verifica si fechaPedidoAceptado != null • Como es null devuelve el valor de ret	false	Correcto
T2	Verificar el camino 2	fechaPedidoAceptado != null fechaPropuestaProduccion = null	• Verifica si fechaPedidoAceptado != null • Como no es null verifica si fechaPropuestaProduccion != null • Como es null devuelve el valor de ret	false	Correcto
T3	Verificar el camino 3	fechaPedidoAceptado != null fechaPropuestaProduccion != null fechaDefinitiva = null	• Verifica si fechaPedidoAceptado != null • Como no es null verifica si fechaPropuestaProduccion != null • Como no es null verifica si fechaDefinitiva != null • Como es null devuelve el valor de ret	False	Correcto
T4	Verificar el camino 4	fechaPedidoAceptado != null fechaPropuestaProduccion != null fechaDefinitiva != null	• Verifica si fechaPedidoAceptado != null • Como no es null verifica si fechaPropuestaProduccion != null • Como no es null verifica si fechaDefinitiva != null • Como no es null devuelve el valor de ret	true	Correcto

```
public String detalles()
{
  String ret = "";

  SimpleDateFormat sdf = new SimpleDateFormat("dd/MMMMM/yyyy");

  ret += "Número de pedido: " + this.númeroPedido;
  ret += "\nFecha de pedido: " + sdf.format(this.fechaPedido.getTime());
  ret += "\nTipo de maquina: " + this.codigoMaquina;
  ret += "\nCantidad a producir: " + this.cantProduccion;
  ret += "\nFecha de entrega solicitada por ventas: " + sdf.format(this.fechaEntregaVentas.getTime());
  ret +=
    "\nFecha propuesta por produccion: " +
    ((this.fechaPropuestaProduccion != null)? sdf.format(this.fechaPropuestaProduccion.getTime()): " - ");
  ret += "\nFecha definitiva: " + ((this.fechaDefinitiva != null)? sdf.format(this.fechaDefinitiva.getTime()): " - ");
  ret +=
    "\nFecha de pedido aceptado: " +
    ((this.fechaPedidoAceptado != null)? sdf.format(this.fechaPedidoAceptado.getTime()): " - ");
  return ret;
}
```

El método anterior se desdobla de la siguiente forma para un mejor análisis de caja blanca:

public String detalles()	
{	
String ret = "";	1
SimpleDateFormat sdf = new SimpleDateFormat("dd/MMMMM/yyyy");	2
ret += "Número de pedido: " + this.númeroPedido;	3
ret += "\nFecha de pedido: " + sdf.format(this.fechaPedido.getTime());	4
ret += "\nTipo de maquina: " + this.codigoMaquina;	5
ret += "\nCantidad a producir: " + this.cantProduccion;	6
ret += "\nFecha de entrega solicitada por ventas: " + sdf.format(this.fechaEntregaVentas.getTime());	7
ret += "\nFecha propuesta por produccion: ";	8
if(this.fechaPropuestaProduccion != null)	9
ret += sdf.format(this.fechaPropuestaProduccion.getTime());	10
else	
ret += " - ";	11
If(this.fechaDefinitiva != null)	12
ret += sdf.format(this.fechaDefinitiva.getTime();	13
else	
ret += " - ";	14
ret +="\nFecha de pedido aceptado: ";	15
if(this.fechaPedidoAceptado != null)	16
ret += sdf.format(this.fechaPedidoAceptado.getTime())	17
else	
ret += " - ";	18
return ret;	19
}	



CC = número de arcos (14) – número de nodos (12) + 2 = 4
CC = nodos condicionales (3) + 1 = 4
CC = número de regiones cerradas (3) + 1 = 4

Camino 1: 1-2-3-4-5-6-7-8-9-11-12-14-15-16-17-19
Camino 2: 1-2-3-4-5-6-7-8-9-10-12-13-15-16-18-19
Camino 3: 1-2-3-4-5-6-7-8-9-11-12-13-15-16-18-19
Camino 4: 1-2-3-4-5-6-7-8-9-11-12-14-15-16-18-19
Camino 5: 1-2-3-4-5-6-7-8-9-10-12-14-15-16-18-19
Camino 6: 1-2-3-4-5-6-7-8-9-10-12-13-15-16-17-19
Camino 7: 1-2-3-4-5-6-7-8-9-10-12-14-15-16-18-19
Camino 8: 1-2-3-4-5-6-7-8-9-11-12-13-15-16-17-19

La complejidad ciclomática está bien calculada, pero debido a algún concepto que desconocemos los caminos a recorrer son 4 más.

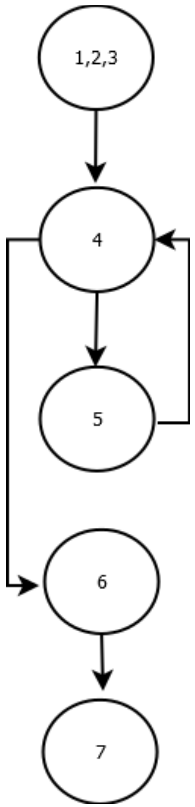
ID	Objetivo de la prueba	Datos de entrada	Procedimiento	Salida esperada	Resultado
T1	Verificar el camino 1	fechaPropuestaProduccion = null fechaDefinitiva = null fechaPedidoAceptado = null	• Se recorren las primeras 8 líneas sin ningún problema • No entra al primer if porque fechaPropuestaProduccion = null, por lo cual entra al else. Ret suma “-“. • No entra al segundo if porque fechaDefinitiva = null. Entra al else y ret suma “-“. • Finalmente tampoco entra al último if porque fechaPedidoAceptado = null, en el else ret suma “-“.	ret = "Número de pedido: " + this.númeroPedido + "\nFecha de pedido: " + sdf.format(this.fechaPedido.getTime()) + "\nTipo de maquina: " + this.codigoMaquina + "\nCantidad a producir: " + this.cantProduccion + "\nFecha de entrega solicitada por ventas: " + sdf.format(this.fechaEntregaVentas.getTime()) + "\nFecha propuesta por produccion: " + " - " + " - " + "\nFecha de pedido aceptado: " + " - ".	Correcto
T2	Verificar el camino 2	fechaPropuestaProduccion != null fechaDefinitiva != null fechaPedidoAceptado != null	• Se recorren las primeras 8 líneas sin ningún problema • Entra al primer if porque fechaPropuestaProduccion != null, ret suma sdf.format(this.fechaPropuestaProduccion.getTime()) • Entra al segundo if porque fechaDefinitiva != null, ret suma sdf.format(this.fechaDefinitiva.getTime() • Finalmente entra al último if porque fechaPedidoAceptado != null, ret suma sdf.format(this.fechaPedidoAceptado.getTime()).	ret = "Número de pedido: " + this.númeroPedido + "\nFecha de pedido: " + sdf.format(this.fechaPedido.getTime()) + "\nTipo de maquina: " + this.codigoMaquina + "\nCantidad a producir: " + this.cantProduccion + "\nFecha de entrega solicitada por ventas: " + sdf.format(this.fechaEntregaVentas.getTime()) + "\nFecha propuesta por produccion: " + sdf.format(this.fechaPropuestaProduccion.getTime()) + " sdf.format(this.fechaDefinitiva.getTime()) + "\nFecha de pedido aceptado: " + sdf.format(this.fechaPedidoAceptado.getTime()).	Correcto
T3	Verificar el camino 3	fechaPropuestaProduccion = null fechaDefinitiva != null fechaPedidoAceptado != null	• Se recorren las primeras 8 líneas sin ningún problema • No entra al primer if porque fechaPropuestaProduccion = null, entra al else entonces ret suma “-“. • Entra al segundo if porque fechaDefinitiva != null, ret suma sdf.format(this.fechaDefinitiva.getTime() • Finalmente entra al último if porque fechaPedidoAceptado != null, ret suma sdf.format(this.fechaPedidoAceptado.getTime()).	ret = "Número de pedido: " + this.númeroPedido + "\nFecha de pedido: " + sdf.format(this.fechaPedido.getTime()) + "\nTipo de maquina: " + this.codigoMaquina + "\nCantidad a producir: " + this.cantProduccion + "\nFecha de entrega solicitada por ventas: " + sdf.format(this.fechaEntregaVentas.getTime()) +	Correcto

				"\nFecha propuesta por produccion: " + “-“+ " sdf.format(this.fechaDefinitiva.getTime()) + "\nFecha de pedido aceptado: " + sdf.format(this.fechaPedidoAceptado.getTime()).	
T4	Verificar el camino 4	fechaPropuestaProduccion = null fechaDefinitiva = null fechaPedidoAceptado != null	• Se recorren las primeras 8 líneas sin ningún problema • No entra al primer if porque fechaPropuestaProduccion = null, entra al else entonces ret suma “-“. • No entra al segundo if porque fechaDefinitiva = null, entra al else entonces ret suma “-“ • Finalmente entra al último if porque fechaPedidoAceptado != null, ret suma sdf.format(this.fechaPedidoAceptado.getTime()).	ret = "Número de pedido: " + this.númeroPedido + "\nFecha de pedido: " + sdf.format(this.fechaPedido.getTime()) + "\nTipo de maquina: " + this.codigoMaquina + "\nCantidad a producir: " + this.cantProduccion + "\nFecha de entrega solicitada por ventas: " + sdf.format(this.fechaEntregaVentas.getTime()) + "\nFecha propuesta por produccion: " + “-“+ “–“ + "\nFecha de pedido aceptado: " + sdf.format(this.fechaPedidoAceptado.getTime()).	Correcto
T5	Verificar el camino 5	fechaPropuestaProduccion != null fechaDefinitiva = null fechaPedidoAceptado != null	• Se recorren las primeras 8 líneas sin ningún problema • Entra al primer if porque fechaPropuestaProduccion != null, ret suma sdf.format(this.fechaPropuestaProduccion.getTime()) • No entra al segundo if porque fechaDefinitiva = null, entra al else entonces ret suma “-“. • Finalmente entra al último if porque fechaPedidoAceptado != null, ret suma sdf.format(this.fechaPedidoAceptado.getTime()).	ret = "Número de pedido: " + this.númeroPedido + "\nFecha de pedido: " + sdf.format(this.fechaPedido.getTime()) + "\nTipo de maquina: " + this.codigoMaquina + "\nCantidad a producir: " + this.cantProduccion + "\nFecha de entrega solicitada por ventas: " + sdf.format(this.fechaEntregaVentas.getTime()) + "\nFecha propuesta por produccion: " + sdf.format(this.fechaPropuestaProduccion.getTime()) + “-“ + "\nFecha de pedido aceptado: " + sdf.format(this.fechaPedidoAceptado.getTime()).	Correcto
T6	Verificar el camino 6	fechaPropuestaProduccion != null fechaDefinitiva != null fechaPedidoAceptado = null	• Se recorren las primeras 8 líneas sin ningún problema • Entra al primer if porque fechaPropuestaProduccion != null, ret suma sdf.format(this.fechaPropuestaProduccion.getTime()) • Entra al segundo if porque fechaDefinitiva != null, ret suma sdf.format(this.fechaDefinitiva.getTime() • Finalmente no entra al último if porque fechaPedidoAceptado = null, en el else ret suma “-“.	ret = "Número de pedido: " + this.númeroPedido + "\nFecha de pedido: " + sdf.format(this.fechaPedido.getTime()) + "\nTipo de maquina: " + this.codigoMaquina + "\nCantidad a producir: " + this.cantProduccion + "\nFecha de entrega solicitada por ventas: " + sdf.format(this.fechaEntregaVentas.getTime()) +	Correcto

				"\nFecha propuesta por produccion: " + sdf.format(this.fechaPropuestaProduccion.getTime()) + " sdf.format(this.fechaDefinitiva.getTime()) + "\nFecha de pedido aceptado: " + "-".	
T7	Verificar el camino 7	fechaPropuestaProduccion != null fechaDefinitiva = null fechaPedidoAceptado = null	• Se recorren las primeras 8 líneas sin ningún problema • Entra al primer if porque fechaPropuestaProduccion != null, ret suma sdf.format(this.fechaPropuestaProduccion.getTime()) • No entra al segundo if porque fechaDefinitiva = null, en el else entonces ret suma "-". • Finalmente no entra al último if porque fechaPedidoAceptado = null, en el else ret suma	ret = "Número de pedido: " + this.númeroPedido + "\nFecha de pedido: " + sdf.format(this.fechaPedido.getTime()) + "\nTipo de maquina: " + this.codigoMaquina + "\nCantidad a producir: " + this.cantProduccion + "\nFecha de entrega solicitada por ventas: " + sdf.format(this.fechaEntregaVentas.getTime()) + "\nFecha propuesta por produccion: " + sdf.format(this.fechaPropuestaProduccion.getTime()) + "-" + "\nFecha de pedido aceptado: " + "-".	Correcto
T8	Verificar el camino 8	fechaPropuestaProduccion = null fechaDefinitiva != null fechaPedidoAceptado = null	• Se recorren las primeras 8 líneas sin ningún problema • No entra al primer if porque fechaPropuestaProduccion = null, en el else ret suma "-". • Entra al segundo if porque fechaDefinitiva != null, ret suma sdf.format(this.fechaDefinitiva.getTime() • Finalmente no entra al último if porque fechaPedidoAceptado = null, en el else ret suma "-".	ret = "Número de pedido: " + this.númeroPedido + "\nFecha de pedido: " + sdf.format(this.fechaPedido.getTime()) + "\nTipo de maquina: " + this.codigoMaquina + "\nCantidad a producir: " + this.cantProduccion + "\nFecha de entrega solicitada por ventas: " + sdf.format(this.fechaEntregaVentas.getTime()) + "\nFecha propuesta por produccion: " + "-" + sdf.format(this.fechaDefinitiva.getTime()) + "\nFecha de pedido aceptado: " + "-".	Correcto

Clase TipoProducto

public void generarTipoProd()
{
String aux = Integer.toString(númeroProd);
int longitud = aux.length();
String codigoTipo = "TIP";
for (int i = 0; i < (6 - longitud); i++)
codigoTipo += "0";
codigoTipo += aux;
this.codigoProducto = codigoTipo;
}
	1
	2
	3
	4
	5
	6
	7



CC = número de arcos (5) – número de nodos (5) + 2 = 2
CC = nodos condicionales (1) + 1 = 2
CC = número de regiones cerradas (1) + 1 = 2

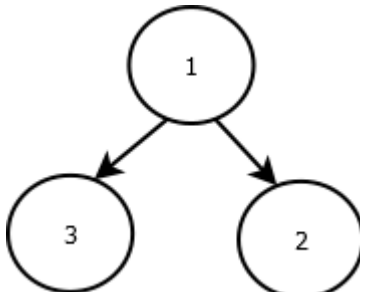
Camino 1: 1-2-3-4-5-6-7
Camino 2: 1-2-3-4-6-7

ID	Objetivo de la prueba	Datos de entrada	Procedimiento	Salida esperada	Resultado
T1	Verificar el camino 1	numeroProd = 11111 longitud = 5 i = 6-longitud = 1	• Calcula la longitud correspondiente al String aux • Entra al ciclo for porque se cumple la condición • Cambia valor de i = 5 • Sale del ciclo	codigoTipo = TIP011111	Correcto
T2	Verificar el camino 2	numeroProd = 111111 longitud = 6 i = 6-longitud = 0	• Calcula la longitud correspondiente al String aux • No entra al ciclo for porque no cumple la condición	codigoTipo = TIP111111	Correcto

Paquete datos.estadosPedido

Clase Evaluación

public void agregarObservacion(Observacion obs)
  throws StateException
{
  if (obs.verificacion())
  this.pedido
  .getListaObservaciones()
  .add(obs);
  else
  throw new StateException("Observacion invalida");
}
	
	1
	2
	
	3



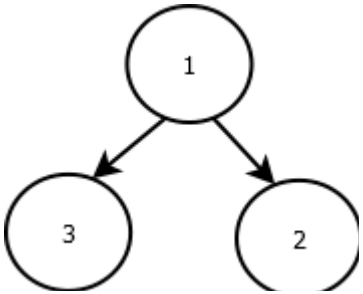
CC = número de arcos (2) – número de nodos (3) + 2 = 1
CC = nodos condicionales (1) + 1 = 2
CC = número de regiones cerradas (1) + 1 = 2

Camino 1: 1-2
Camino 2: 1-3

La complejidad ciclomática no es idéntica en todas las fórmulas debido a que no las ecuaciones no están desarrolladas para el manejo de excepciones. Debido a esto, en este caso se despreciará y se usarán los caminos posibles.

ID	Objetivo de la prueba	Datos de entrada	Procedimiento	Salida esperada	Resultado
T1	Verificar el camino 1	this.pedido.verificaNull()) = true pedido.fechaDefinitiva != null pedido.fechaPedidoAceptado != null pedido.fechaPropuestaProduccion != null	• Corrobora que this.pedido.verificaNull())== true, como lo es entra al if • Sale del método	Cambia el estado del pedido a estado Aceptado	Correcto
T2	Verificar el camino 2	this.pedido.verificaNull()) = false pedido.fechaDefinitiva = null pedido.fechaPedidoAceptado = null pedido.fechaPropuestaProduccion = null	• Como this.pedido.verificaNull())no es true, entra al else • Lanza una excepción debido a que el pedido no está listo para ser aceptado	Excepción StateException	Correcto

public void aceptarPedido()
  throws StateException
{
  if (this.pedido.verificaNull())
  this.pedido.setEstadoActual(new Aceptado(this.pedido));
  else
  throw new StateException("El pedido no está listo para ser aceptado");
}
	
	1
	2
	
	3



CC = número de arcos (2) – número de nodos (3) + 2 = 1
CC = nodos condicionales (1) + 1 = 2
CC = número de regiones cerradas (1) + 1 = 2

Camino 1: 1-2
Camino 2: 1-3

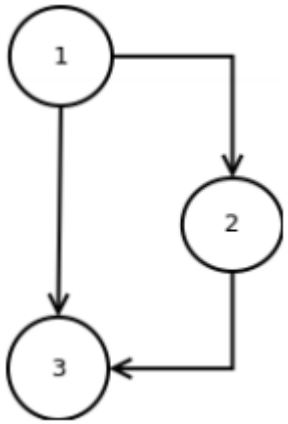
La complejidad ciclomática no es idéntica en todas las fórmulas debido a que no las ecuaciones no están desarrolladas para el manejo de excepciones. Debido a esto, en este caso se despreciará y se usarán los caminos posibles.

ID	Objetivo de la prueba	Datos de entrada	Procedimiento	Salida esperada	Resultado
T1	Verificar el camino 1	this.pedido.verificaNull()) = true	• Corrobora que this.pedido.verificaNull())== true, como lo es entra al if • Sale del método	Cambia el estado del pedido a estado Aceptado	Correcto
T2	Verificar el camino 2	obs.verificacion = false	• Como this.pedido.verificaNull())no es true, entra al else • Lanza una excepción debido a que el pedido no está listo para ser aceptado	Excepción StateException	Correcto

Paquete listas

Clase ListaLotes

public static ListaLotes getInstance()
{
  if (_instance == null)
  _instance = new ListaLotes();
  return _instance;
}
	
	1
	2
	3



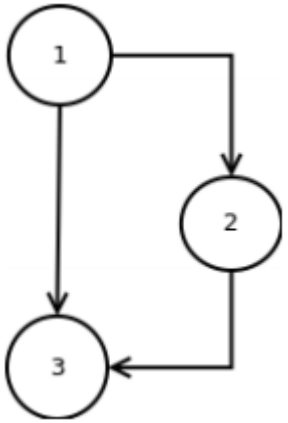
CC = número de arcos(3) – número de nodos(3) +2 = 2
CC = nodos condicionales(1) +1 = 2
CC = número de regiones cerradas(1) +1 =2

Camino 1: 1-3
Camino 2: 1-2-3

ID	Objetivo de la prueba	Datos de entrada	Procedimiento	Salida esperada	Resultado
T1	Verificar el camino 1	Atributo de clase _instance inicializado	• Verifica si el atributo _instance es null • Como no es null, devuelve la referencia	Referencia a la instancia de clase	Correcto
T2	Verificar el camino 2	Atributo de clase _instance sin inicializar	• Verifica si el atributo _instance es null • Como es null, crea una instancia • Devuelve la referencia a la nueva instancia		Correcto

Clase ListaMaterialesStock

public static ListaMaterialesStock getInstance()
{
if (_instance == null)
_instance = new ListaMaterialesStock();
return _instance;
}
	
	1
	2
	3

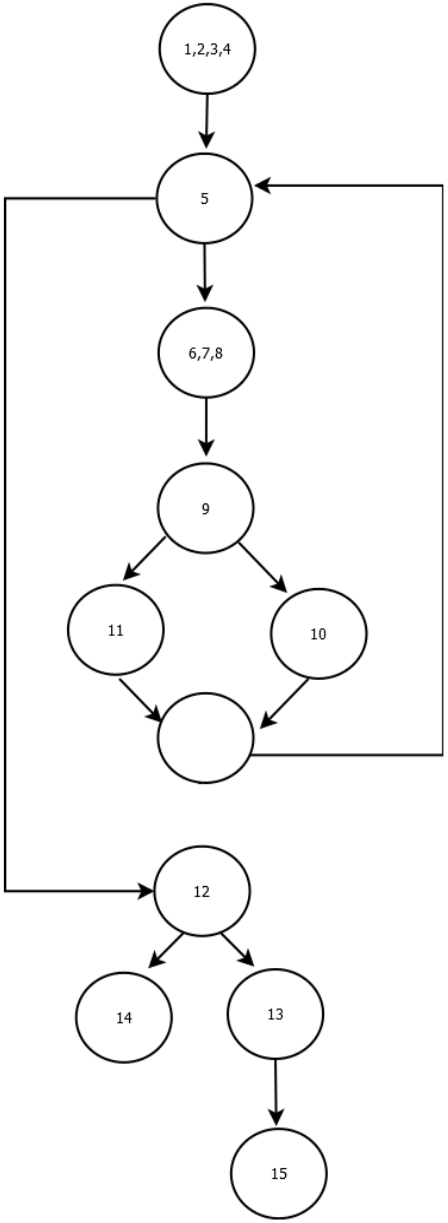


CC = número de arcos(3) – número de nodos(3) +2 = 2
CC = nodos condicionales(1) +1 = 2
CC = número de regiones cerradas(1) +1 =2

Camino 1: 1-3
Camino 2: 1-2-3

ID	Objetivo de la prueba	Datos de entrada	Procedimiento	Salida esperada	Resultado
T1	Verificar el camino 1	Atributo de clase _instance inicializado	• Verifica si el atributo _instance es null • Como no es null, devuelve la referencia	Referencia a la instancia de clase	Correcto
T2	Verificar el camino 2	Atributo de clase _instance sin inicializar	• Verifica si el atributo _instance es null • Como es null, crea una instancia • Devuelve la referencia a la nueva instancia		Correcto

```
  public ListaMateriales verificarExistencias(String tipo, int cantidad)  throws FaltantesException, Exception
 {
  assert Verificaciones.verificaTipoCodigo(tipo) : "Tipo invalido";
  assert Verificaciones.verificaCantProduccion(cantidad) : "Cantidad invalida";

  ListaMateriales listaFinal = new ListaMateriales();                                                                  1
  ListaMateriales listaFaltantes = new ListaMateriales();                                                              2

  ListaMateriales receta = this.recetas.get(tipo).getListaMateriales();                                                3
  Iterator<Material> itReceta = receta.getIterator();                                                                  4

  while (itReceta.hasNext())                                                                                           5
  {
   Material matReceta = itReceta.next();                                                                               6
   Material matExistente = this.listaExistencias.getMaterial(matReceta.getCodigo());                                  7
   float cantidadMaterialNecesaria = matReceta.getCantidad() * cantidad;                                               8
   if (matExistente.getCantidad() >= cantidadMaterialNecesaria)                                                        9
    listaFinal.agregarMaterial(new Material(matReceta.getCodigo(), matReceta.getDescripcion(), cantidadMaterialNecesaria));  10
   else
    listaFaltantes.agregarMaterial(new Material(matReceta.getCodigo(), matReceta.getDescripcion(), cantidadMaterialNecesaria -  11
matExistente.getCantidad()));
  }
  if (listaFaltantes.size() <= 0)                                                                                      12
   ListaMateriales ret = listaFinal                                                                                    13
  else
   throw new FaltantesException("No se cuenta con los suficientes materiales\nEstas son las cantidades faltantes:",listaFaltantes);  14
 return ret;                                                                                                           15
 }
```



CC = número de arcos (13) – número de nodos (11) + 2 = 4
CC = nodos condicionales (3) + 1 = 4
CC = número de regiones cerradas (3) + 1 = 4

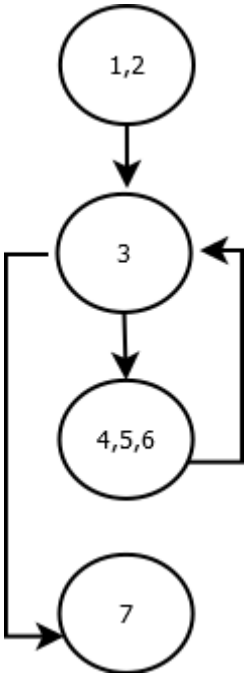
Camino 1: 1-2-3-4-5-6-7-8-9-11-5-12-14
Camino 2: 1-2-3-4-5-6-7-8-9-10-5-12-13-15
Camino 3: 1-2-3-4-5-12-13-15
Camino 4: 1-2-3-4-5-12-14

ID	Objetivo de la prueba	Datos de entrada	Procedimiento	Salida esperada	Resultado
T1	Verificar el camino 1	receta != null itReceta != null itReceta posee algún elemento más matReceta != null matExistente != null matExistente.getCantidad()<cantidadMaterialNecesaria listaFaltantes posee algún elemento	• Obtiene la lista de recetas del producto especificado • Obtiene el iterador de esa lista correspondiente (itReceta) • Como itReceta tiene un elemento más, entra al while • Obtiene el material siguiente en el itReceta y a su vez obtiene también el mismo material pero de la lista de existencias (matReceta y matExistente respectivamente) • Compara si las cantidades del material existente son mayores que las del necesario (matExistente.getCantidad() >= cantidadMaterialNecesaria). • Como no se cumple la condición entra al else y agrega al material a la lista de faltantes (listaFaltantes) • Sale del ciclo • Como listaFaltantes posee un elemento no entra al if, en el else tira una excepcion	Excepción de FaltantesException	Correcto
T2	Verificar el	receta != null	• Obtiene la lista de recetas del		

	camino 2	itReceta != null itReceta posee algún elemento más matReceta != null matExistente != null matExistente.getCantidad()>=cantidadMaterialNecesaria listaFaltantes no posee ningún elemento	producto especificado • Obtiene el iterador de esa lista correspondiente (itReceta) • Como itReceta tiene un elemento más, entra al while • Obtiene el material siguiente en el itReceta y a su vez obtiene también el mismo material pero de la lista de existencias (matReceta y matExistente respectivamente) • Compara si las cantidades del material existente son mayores que las del necesario (matExistente.getCantidad() >= cantidadMaterialNecesaria). • Como se cumple la condición entra al if y agrega a la listaFinal el material. • Sale del ciclo • Como listaFaltantes no posee ningún elemento, entra al if.	ListaFinal	Correcto
T3	Verificar el camino 3	receta != null itReceta != null itReceta no posee ningún elemento listaFaltantes no posee ningún elemento	• Obtiene la lista de recetas del producto especificado • Obtiene el iterador de esa lista correspondiente (itReceta) • Como itReceta no tiene ningún elemento, no entra al while • Como listaFaltantes no posee	listaFinal	Correcto

			ningún elemento, entra al while		
T4	Verificar el camino 4	receta != null itReceta != null itReceta no posee ningún elemento listaFaltantes posee algún elemento	• Obtiene la lista de recetas del producto especificado • Obtiene el iterador de esa lista correspondiente (itReceta) • Como itReceta no tiene ningún elemento, no entra al while • Como listaFaltantes posee por lo menos un elemento, no entra al if • Desde el else lanza una excepción	Excepción faltantesException	Correcto

```
  public void actualizarExistencias(TipoProducto tipo)
{
  assert tipo != null : "Producto nulo";

  ListaMateriales lista = tipo.getListaMateriales();                                                    1
  Iterator<Material> it = lista.getIterator();                                                          2
  while (it.hasNext())                                                                                  3
  {
    Material mat = it.next();                                                                           4
    try
    {                                                                                                   5
      float cant1 = this.listaExistencias.getMaterial(mat.getCodigo()).getCantidad();                   6
      this.listaExistencias.getMaterial(mat.getCodigo()).setCantidad(cant1 - mat.getCantidad());
    }
    catch (Exception e)
    {
    }
  }
}                                                                                                       7
```



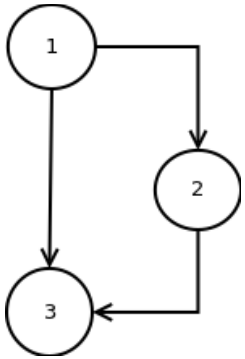
CC = número de arcos (4) – número de nodos (4) + 2 = 2
CC = nodos condicionales (1) + 1 = 2
CC = número de regiones cerradas (1) + 1 = 2

Camino 1: 1-2-3-4-5-6-7
Camino 2: 1-2-3-7

ID	Objetivo de la prueba	Datos de entrada	Procedimiento	Salida esperada	Resultado
T1	Verificar el camino 1	lista != null it != null it posee algún elemento	• Entra al while porque it.hasNext() == true • Modifica el valor de it, it.hasNext() == false • Sale del ciclo	listaExistencias actualizada	Correcto
T2	Verificar el camino 2	lista != null it != null it no posee ningún elemento	• No entra al ciclo while porque no cumple la condición	listaExistencas actualizada	Correcto

Clase ListaEmpleado

public static ListaEmpleados getInstance()
{
  if (_instance == null)
  _instance = new ListaEmpleados();
  return _instance;
}
	
	1
	2
	3



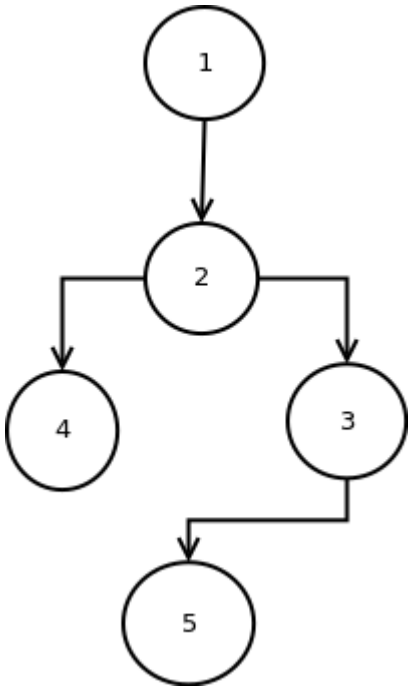
CC = número de arcos (3) – número de nodos (3) + 2 = 2
CC = nodos condicionales (1) + 1 = 2
CC = número de regiones cerradas (1) + 1 = 2

Camino 1: 1 – 3
Camino 2: 1 – 2 – 3

ID	Objetivo de la prueba	Datos de entrada	Procedimiento	Salida esperada	Resultado
T1	Verificar el camino 1	Atributo de clase _instance inicializado	• Verifica si el atributo _instance es null • Como no es null, devuelve la referencia	Referencia a la instancia de clase	Correcto
T2	Verificar el camino 2	Atributo de clase _instance sin inicializar	• Verifica si el atributo _instance es null • Como es null, crea una instancia		Correcto

			• Devuelve la referencia a la nueva instancia		

public Empleado buscar(String legajo) throws Exception {	
Empleado ret = null;	1
if (this.empleados.containsKey(legajo))	2
ret = this.empleados.get(legajo);	3
else	
throw new Exception("El empleado no es parte de la empresa");	4
return ret;	5
}	



CC = número de arcos (4) – número de nodos (5) + 2 = 1
CC = nodos condicionales (1) + 1 = 2
CC = número de regiones cerradas (1) + 1 = 1

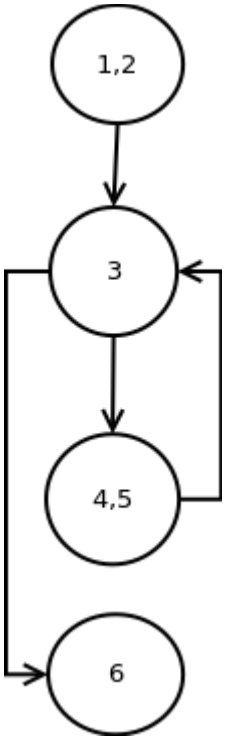
Como la complejidad ciclotímica no contempla el caso de ramas que producen excepciones, no se tendrá en cuenta el valor obtenido mediante la formula y se tomaran los siguientes caminos:

Camino 1: 1 – 2 – 4
Camino 2: 1 – 2 – 3 – 5

ID	Objetivo de la prueba	Datos de entrada	Procedimiento	Salida esperada	Resultado
T1	Verificar el camino 1	Legajo contenido en la lista de empleados	• Crea una variable de retorno en null • Verifica si el elemento esta contenido en la lista • Como esta contenido, guarda la referencia al elemento en la variable de retorno • Se devuelve el elemento	Referencia a la instancia del elemento buscado	Correcto
T2	Verificar el camino 2	Legajo no contenido en la lista de empleados	• Crea una variable de retorno en null • Verifica si el elemento esta contenido en la lista • Como el elemento no se encuentra en la lista, informa del error	Error	Correcto

Clase ListaMateriales

```
public String detalles()
{
  Iterator<Material> it = this.getIterator();                    1
  String aux = "";                                               2
  while (it.hasNext())                                           3
  {
    Material mat = it.next();                                    4
    aux = aux + mat.detalles() + " unidades\n";                  5
  }
  return aux;                                                    6
}
```

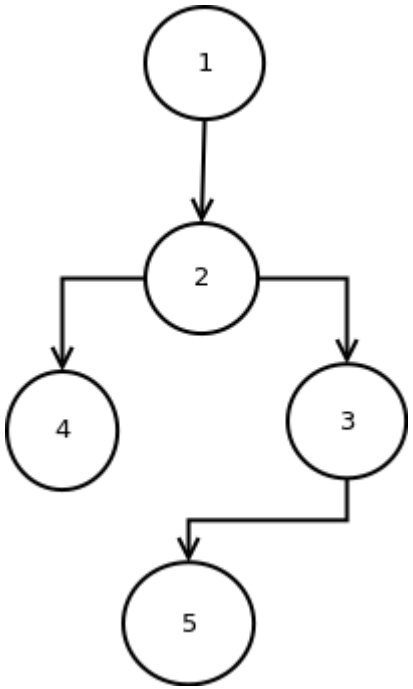


CC = número de arcos (4) – número de nodos (4) + 2 = 2
CC = nodos condicionales (1) + 1 = 2
CC = número de regiones cerradas (1) + 1 = 2

Camino 1: 1 – 2 – 3 - 6
Camino 2: 1 – 2 – 3 – 4 – 5 – 6

ID	Objetivo de la prueba	Datos de entrada	Procedimiento	Salida esperada	Resultado
T1	Verificar el camino 1	Iterator de Materiales vacío	• Inicializa una variable para retorno como String vacío. • Obtiene el iterator con los datos • Mientras el iterator tiene elementos, los agrega al retorno. Como el iterator esta vacío, no agrega nada. • Devuelve el String vacío	Devuelve un String vacío ya que el iterator no tiene elementos	Correcto
T2	Verificar el camino 2	Iterator de Materiales con un solo elemento	• Inicializa una variable para retorno como String vacío. • Obtiene el iterator con los datos • Mientras el iterator tiene elementos, los agrega al retorno. Como el iterator es de longitud 1, agrega un solo detalle • Devuelve el String de retorno con un único detalle.	Devuelve un String con un solo detalle, ya que el iterator es de longitud 1	Correcto

```
 public Material getMaterial(String código)
 throws Exception
{
 Material ret = null;                                                    1
 if (this.lista.containsKey(código))                                     2
   ret = this.lista.get(código);                                         3
 else
   throw new Exception("El material no se encuentra en la lista");       4
 return ret;                                                             5
}
```



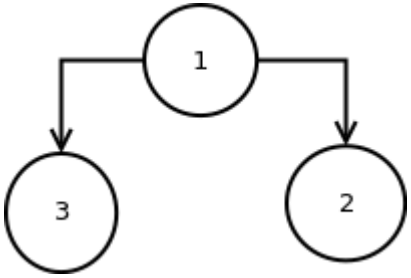
CC = número de arcos (4) – número de nodos (5) + 2 = 1
CC = nodos condicionales (1) + 1 = 2
CC = número de regiones cerradas (1) + 1 = 1

Como la complejidad ciclotímica no contempla el caso de ramas que producen excepciones, no se tendrá en cuenta el valor obtenido mediante la formula y se tomaran los siguientes caminos:

Camino 1: 1 – 2 – 4
Camino 2: 1 – 2 – 3 – 5

ID	Objetivo de la prueba	Datos de entrada	Procedimiento	Salida esperada	Resultado
T1	Verificar el camino 1	Código contenido en la lista de materiales	• Crea una variable de retorno en null • Verifica si el elemento esta contenido en la lista • Como esta contenido, guarda la referencia al elemento en la variable de retorno • Se devuelve el elemento	Referencia a la instancia del elemento buscado	Correcto
T2	Verificar el camino 2	Código no contenido en la lista de materiales	• Crea una variable de retorno en null • Verifica si el elemento esta contenido en la lista • Como el elemento no se encuentra en la lista, informa del error	Error	Correcto

public void borrarMaterial(String código)
  throws Exception
{
  if (this.lista.containsKey(código))
  this.lista.remove(código);
  else
  throw new Exception("Campo vacio");
}
	
	1
	2
	
	3



CC = número de arcos (2) – número de nodos (3) + 2 = 1
CC = nodos condicionales (1) + 1 = 2
CC = número de regiones cerradas (0) + 1 = 1

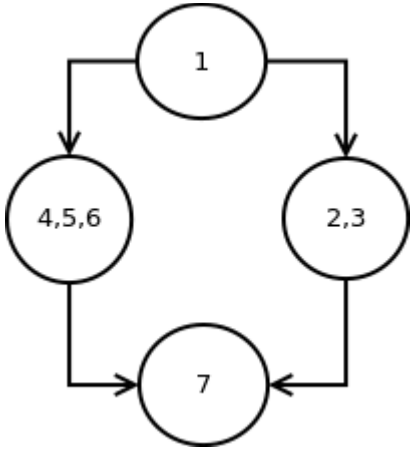
Como la complejidad ciclotímica no contempla el caso de ramas que producen excepciones, no se tendrá en cuenta el valor obtenido mediante la formula y se tomaran los siguientes caminos:

Camino 1: 1 – 2 – 3
Camino 2: 1 – 4

ID	Objetivo de la prueba	Datos de entrada	Procedimiento	Salida esperada	Resultado
T1	Verificar el camino 1	Código contenido en la lista de materiales para borrar	• Verifica si el elemento esta contenido en la lista • Como esta contenido, lo borra de la lista	Elemento eliminado de la lista	Correcto

T2	Verificar el camino 2	Código no contenido en la lista de materiales	• Verifica si el elemento esta contenido en la lista • Como el elemento no se encuentra en la lista, informa del error	Error	Correcto

public void agregarMaterial(String código, String descripcion, float cantidad)
  throws LengthException
{
  if (!lista.containsKey(código))
  {
  Material mat = new Material(código, descripcion, cantidad);
  this.agregarMaterial(mat);
  }
  else
  {
  Material mat = lista.get(código);
  mat.setCantidad(mat.getCantidad() + cantidad);
  mat.setDescripcion(descripcion);
  }
}
	
	1
	
	2
	3
	
	4
	5
	6
	
	7



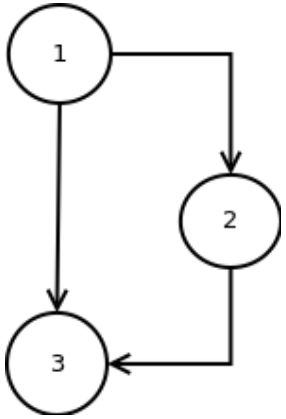
CC = número de arcos (4) – número de nodos (4) + 2 = 2
CC = nodos condicionales (1) + 1 = 2
CC = número de regiones cerradas (1) + 1 = 2

Camino 1: 1 – 2 – 3 – 7
Camino 2: 1 – 4 – 5 – 6 – 7

ID	Objetivo de la prueba	Datos de entrada	Procedimiento	Salida esperada	Resultado
T1	Verificar el camino 1	Un código no contenido en la lista de materiales, descripción y cantidad a elegir	• Verifica si el código ya esta contenido en la lista, y como no esta entra en la rama de true • Crea el nuevo material con los datos indicados y lo agrega a la lista	Nuevo material agregado en la lista	Correcto
T2	Verificar el camino 2	Un código ya contenido en la lista de materiales, descripción y cantidad a elegir	• Verifica si el código ya esta contenido en la lista, y como esta entra en la rama de false • Obtiene le material indicado por el código y modifica sus datos.	Material ya existente en la lista modificado	Correcto

Clase ListaPedidos

public static ListaPedidos getInstance() {
  if (_instance == null)
  _instance = new ListaPedidos();
  return _instance;
}
	
	1
	2
	3



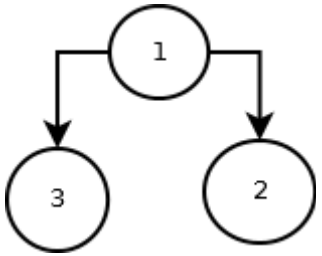
CC = número de arcos (3) – número de nodos (3) + 2 = 2
CC = nodos condicionales (1) + 1 = 2
CC = número de regiones cerradas (1) + 1 = 2

Camino 1: 1 – 3
Camino 2: 1 – 2 – 3

ID	Objetivo de la prueba	Datos de entrada	Procedimiento	Salida esperada	Resultado
T1	Verificar el camino 1	Atributo de clase _instance inicializado	• Verifica si el atributo _instance es null • Como no es null, devuelve la referencia	Referencia a la instancia de clase	Correcto
T2	Verificar el camino 2	Atributo de clase _instance sin inicializar	• Verifica si el atributo _instance es null • Como es null, crea una instancia • Devuelve la referencia a la nueva instancia		Correcto

Clase Controlador

public Pedido getPedidoActual() throws Exception
{
  if(this.pedidoActual != null)
  return this.pedidoActual;
  else
  throw new Exception("El pedido es incorrecto");
}
	
	1
	2
	
	3



CC = número de arcos (2) – número de nodos (3) + 2 = 1
CC = nodos condicionales (1) + 1 = 2
CC = número de regiones cerradas (0) + 1 = 1

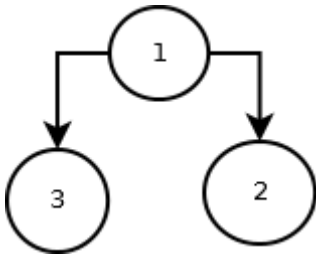
Como el método posee excepciones y en el análisis de caja blanca no esta contemplado, se añadirán todos los caminos que sean necesarios para cubrir el código.

Camino 1: 1 – 2
Camino 2: 1 – 3

ID	Objetivo de la prueba	Datos de entrada	Procedimiento	Salida esperada	Resultado
T1	Verificar el camino 1	Atributo pedidoActual != null	• Se verifica que el atributo de clase pedidoActual sea distinto de null • Como lo es, se devuelve una referencia a él	Referencia al pedido actual	Correcto
T2	Verificar el camino 2	Atributo pedidoActual == null	• Se verifica que el atributo de clase pedidoActual sea distinto de null • Como no lo es, se informa	Excepción	Correcto

			mediante una excepcion		

public TipoProducto getProductoActual() throws Exception
{
  if(this.productoActual != null)
  return productoActual;
  else
  throw new Exception("El producto es incorrecto");
}
	
	1
	2
	
	3



CC = número de arcos (2) – número de nodos (3) + 2 = 1

CC = nodos condicionales (1) + 1 = 2

CC = número de regiones cerradas (0) + 1 = 1

Como el método posee excepciones y en el análisis de caja blanca no esta contemplado, se añadirán todos los caminos que sean necesarios para cubrir el código.

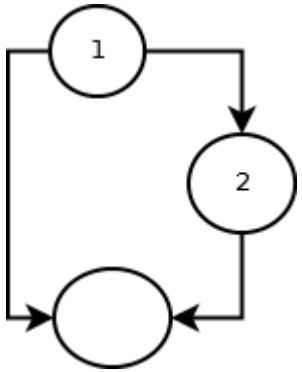
Camino 1: 1 – 2

Camino 2: 1 – 3

ID	Objetivo de la prueba	Datos de entrada	Procedimiento	Salida esperada	Resultado
T1	Verificar el camino 1	Atributo productoActual != null	• Se verifica que el atributo de clase productoActual sea distinto de null • Como lo es, se devuelve una referencia a él	Referencia al producto actual	Correcto
T2	Verificar el camino 2	Atributo productoActual == null	• Se verifica que el atributo de clase productoActual sea distinto de null • Como no lo es, se informa	Excepción	Correcto

			mediante una excepcion		

public void removePedido()
{
  if(this.pedidoActual != null)
  this.pedidos.borrarPedido(this.pedidoActual);
}
	
	1
	2



CC = número de arcos (3) – número de nodos (3) + 2 = 2

CC = nodos condicionales (1) + 1 = 2

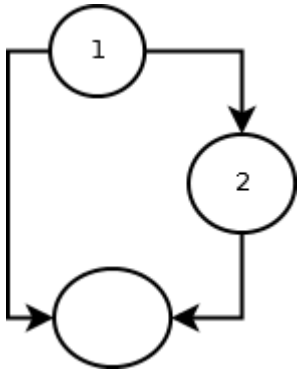
CC = número de regiones cerradas (1) + 1 = 2

Camino 1: 1 – 2 – return

Camino 2: 1 – return

ID	Objetivo de la prueba	Datos de entrada	Procedimiento	Salida esperada	Resultado
T1	Verificar el camino 1	Atributo pedidoActual != null	• Se verifica que el atributo de clase pedidoActual sea distinto de null • Como lo es, se elimina de la lista de pedidos el pedido actual	Pedido actual eliminado de la lista de pedidos	Correcto
T2	Verificar el camino 2	Atributo pedidoActual == null	• Se verifica que el atributo de clase pedidoActual sea distinto de null • Como no lo es, se sale del método sin realizar nada	Ningún cambio se efectuá	Correcto

public void removeLote()
{
  if (this.loteActual != null)
  this.lotes.borrarLote(this.loteActual);
}
	
	1
	2
	3



CC = número de arcos (3) – número de nodos (3) + 2 = 2

CC = nodos condicionales (1) + 1 = 2

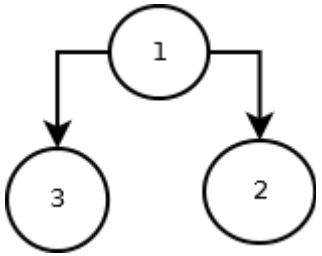
CC = número de regiones cerradas (1) + 1 = 2

Camino 1: 1 – 2 – return

Camino 2: 1 – return

ID	Objetivo de la prueba	Datos de entrada	Procedimiento	Salida esperada	Resultado
T1	Verificar el camino 1	Atributo loteActual != null	• Se verifica que el atributo de clase loteActual sea distinto de null • Como lo es, se elimina de la lista de lotes el lote actual	Lote actual eliminado de la lista de lotes	Correcto
T2	Verificar el camino 2	Atributo loteActual == null	• Se verifica que el atributo de clase loteActual sea distinto de null • Como no lo es, se sale del método sin realizar nada	Ningún cambio se efectuá	Correcto

public void borrarMaterial(String codigo)
  throws Exception
{
  if (Verificaciones.verificaCodigo(codigo))
  this.productoActual.getListaMateriales().borrarMaterial(codigo);
  else
  throw new Exception("Codigo invalido");
}
	
	1
	2
	
	3



CC = número de arcos (2) – número de nodos (3) + 2 = 1
CC = nodos condicionales (1) + 1 = 2
CC = número de regiones cerradas (0) + 1 = 1

Como el método posee excepciones y en el análisis de caja blanca no esta contemplado, se añadirán todos los caminos que sean necesarios para cubrir el código.

Camino 1: 1 – 2
Camino 2: 1 – 3

ID	Objetivo de la prueba	Datos de entrada	Procedimiento	Salida esperada	Resultado
T1	Verificar el camino 1	Código valido	• Se verifica que le código sea valido. Como lo es, entra en el if. • Se borra de la lista de materiales del producto actual el material indicado por el código.	Material eliminado de la lista	Correcto
T2	Verificar el camino 2	Código invalido	• Se verifica que le código sea valido. • Como no lo es, entra en el else	Excepción	Correcto

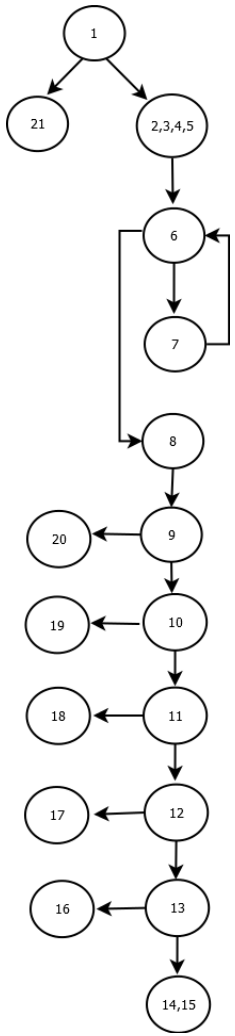
			y se notifica el error mediante una excepción.		

```
public void crearNuevoPedido(Calendar fechaPedido, String tipoMaquina, int cantProducir,
                    Calendar fechaSolicitadaVentas)
  throws Exception
{
  if (Verificaciones.verificaTipoProducto(tipoMaquina))                                                    1
  {
    String maquina = ListaMaterialesStock.getInstance().getCodigo(tipoMaquina);                            2
    String aux = Integer.toString(this.pedidos.getProximoNumeroPedido());                                 3
    int longitud = aux.length();                                                                           4
    String numeroPedido = "PED";                                                                           5
    for (int i = 0; i < (6 - longitud); i++)                                                               6
      numeroPedido += "0";                                                                                 7
    numeroPedido += aux;                                                                                   8
    if (Verificaciones.verificaTipoCodigo(maquina) && Verificaciones.verificaNumeroPedido(numeroPedido) &&  9
       fechaPedido != null && Verificaciones.verificaCantProduccion(cantProducir) && fechaSolicitadaVentas != null)
    {
      Pedido nuevo = new Pedido(numeroPedido, fechaPedido, fechaSolicitadaVentas, maquina, tipoMaquina, cantProducir);
      pedidos.agregarNuevo(nuevo);
    }                                                                                                      10
    else
      throw new Exception("Pedido invalido");                                                              11
  }
  else
    throw new Exception("Tipo producto invalido");                                                         12
}

                                                                                                           13
```

El código anterior se desdobla de la siguiente forma para un mejor análisis de caja blanca:

```
public void crearNuevoPedido(Calendar fechaPedido, String tipoMaquina, int cantProducir,
                        Calendar fechaSolicitadaVentas)
  throws Exception
{
  if (Verificaciones.verificaTipoProducto(tipoMaquina))                                                                    1
  {
    String maquina = ListaMaterialesStock.getInstance().getCodigo(tipoMaquina);                                            2
    String aux = Integer.toString(this.pedidos.getProximoNumeroPedido());                                                 3
    int longitud = aux.length();                                                                                           4
    String numeroPedido = "PED";                                                                                           5
    for (int i = 0; i < (6 - longitud); i++)                                                                               6
      numeroPedido += "0";                                                                                                 7
    numeroPedido += aux;                                                                                                   8
    if (Verificaciones.verificaTipoCodigo(maquina))                                                                        9
        if(Verificaciones.verificaNumeroPedido(numeroPedido))                                                             10
           if(fechaPedido != null)                                                                                        11
              if(Verificaciones.verificaCantProduccion(cantProducir))                                                     12
                 if(fechaSolicitadaVentas != null)                                                                        13
                 {
                     Pedido nuevo = new Pedido(numeroPedido, fechaPedido, fechaSolicitadaVentas, maquina, tipoMaquina, cantProducir);   14
                     pedidos.agregarNuevo(nuevo);                                                                        15

                 }
                else
                    throw new Exception("Pedido invalido");                                                            16
              else
                  throw new Exception("Pedido invalido");                                                              17
           else
```

```
                throw new Exception("Pedido invalido");                    18
          else
       throw new Exception("Pedido invalido");                             19
    else
      throw new Exception("Pedido invalido");                              20
   }
   else
     throw new Exception("Tipo producto invalido");                        21
}
```



CC = número de arcos (17) – número de nodos (17) + 2 = 2
CC = nodos condicionales (7) + 1 = 8
CC = número de regiones cerradas (7) + 1 = 8

Camino 1: 1-2-3-4-5-6-7-8-9-10-11-12-13-14-15
Camino 2: 1-2-3-4-5-6-7-8-9-20
Camino 3: 1-2-3-4-5-6-8-9-10-11-12-13-14-15
Camino 4: 1-2-3-4-5-6-8-9-20
Camino 5: 1-21

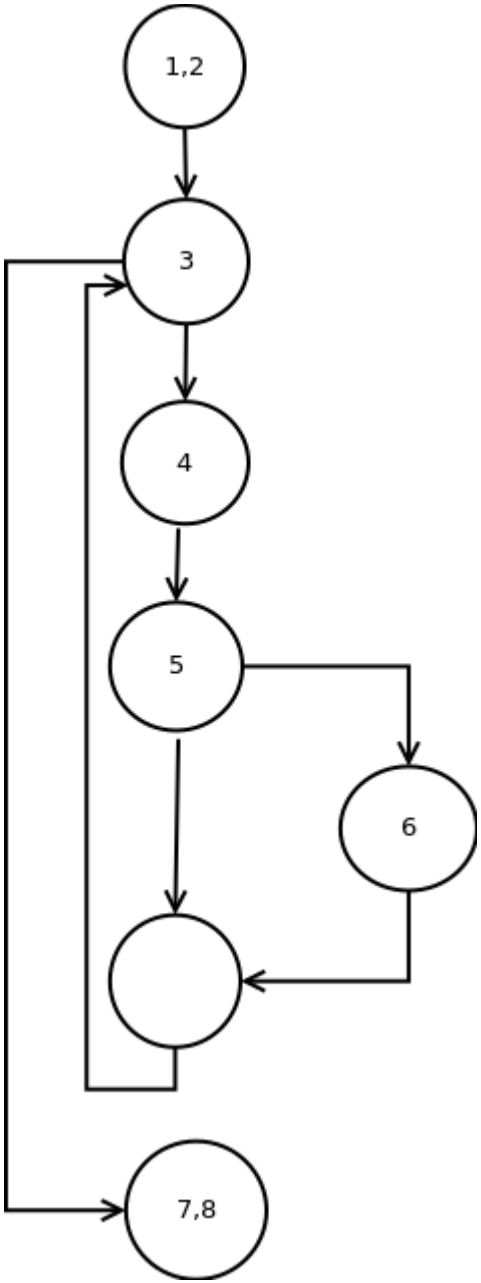
La complejidad ciclomática no es idéntica en todas las fórmulas debido a que las ecuaciones no están desarrolladas para el manejo de excepciones. Debido a esto, en este caso se despreciará y se usarán los caminos posibles.
Si cualquiera de los valores de los if pertenecientes a las líneas 9,10,11,12,13 fallan, se lanza la misma excepción.
Entonces para simplificar el análisis, se tomará un caso en el que falla el primero (línea 9) y otro en el que todos son correctos. Por esto es que solo se desarrollaron los 5 caminos posibles de arriba, y no las demás posibilidades.

ID	Objetivo de la prueba	Datos de entrada	Procedimiento	Salida esperada	Resultado
T1	Verificar el camino 1	Verificaciones.verificaTipoProducto(tipoMaquina) = true longitud = 5 i = 6-longitud = 1 Verificaciones.verificaTipoCodigo(maquina) = true Verificaciones.verificaNumeroPedido(numeroPedido) = true fechaPedido != null Verificaciones.verificaCantProduccion(cantProducir) = true fechaSolicitadaVentas != null	• Verifica que tipoMaquina sea correcto, como lo es entra al if • Obtiene el código correspondiente a ese tipo de máquina (maquina) • Obtiene el número correspondiente al siguiente pedido (aux) • Calcula la longitud de aux • Como i= 4, entra al ciclo for • Verifica que maquina sea correcto, como también el numeroPedido, como cantProducir. Y que fechaPedido y fechaSolicitadaVentas sean distintas de null. Como se cumple todo entra a los if • Crea un nuevo pedido	Creación de nuevo pedido y agregación del mismo a la lista de pedidos	Correcto
T2	Verificar el camino 2	Verificaciones.verificaTipoProducto(tipoMaquina) = true longitud = 5 i = 6-longitud = 1 Verificaciones.verificaTipoCodigo(maquina) = false	• Verifica que tipoMaquina sea correcto, como lo es entra al if • Obtiene el código correspondiente a ese tipo de máquina (maquina)	Excepción “Pedido invalido”	Correcto

			• Obtiene el número correspondiente al siguiente pedido (aux) • Calcula la longitud de aux • Como i= 4, entra al ciclo for • Verifica que maquina sea correcto, como no lo es, se lanza una excepción en la línea 20		
T3	Verificar el camino 3	Verificaciones.verificaTipoProducto(tipoMaquina) = true longitud = 6 i = 6-longitud = 0 Verificaciones.verificaTipoCodigo(maquina) = true Verificaciones.verificaNumeroPedido(numeroPedido) = true fechaPedido != null Verificaciones.verificaCantProduccion(cantProducir) = true fechaSolicitadaVentas != null	• Verifica que tipoMaquina sea correcto, como lo es entra al if • Obtiene el código correspondiente a ese tipo de máquina (maquina) • Obtiene el número correspondiente al siguiente pedido (aux) • Calcula la longitud de aux • Como i= 5, no entra al ciclo for, pero el numeroPedido ya fue generado en alguna iteración anterior • Verifica que maquina sea correcto, como también el numeroPedido, como cantProducir. Y que fechaPedido y fechaSolicitadaVentas sean	Creación de nuevo pedido y agregación del mismo a la lista de pedidos	Correcto

			distintas de null. Como se cumple todo entra a los if • Crea un nuevo pedido		
T4	Verificar el camino 4	Verificaciones.verificaTipoProducto(tipoMaquina) = true longitud = 6 i = 6-longitud = 0 Verificaciones.verificaTipoCodigo(maquina) = false	• Verifica que tipoMaquina sea correcto, como lo es entra al if • Obtiene el código correspondiente a ese tipo de máquina (maquina) • Obtiene el número correspondiente al siguiente pedido (aux) • Calcula la longitud de aux • Como i= 5, no entra al ciclo for, pero el numeroPedido ya fue generado en alguna iteración anterior • Verifica que maquina sea correcto, como no lo es, se lanza una excepción en la línea 20	Excepción “Pedido invalido	Correcto
T5	Verificar el camino 5	Verificaciones.verificaTipoProducto(tipoMaquina) = false	• Verifica que tipoMaquina sea correcto, como no lo es no entra al if, lanza una excepción desde la línea 21	Excepción ”Tipo producto invalido”	Correcto

```
public Iterator<Pedido> getPedidosEvaluacion()
{
  Iterator<Pedido> it = this.pedidos.getIterator();                  1
  ArrayList<Pedido> lotesEv = new ArrayList<>();                     2
  while (it.hasNext())                                               3
  {
    Pedido lot = it.next();                                          4
    if (lot.isEnEvaluacion())                                        5
      lotesEv.add(lot);                                              6
  }
  it = lotesEv.iterator();                                           7
  return it;                                                         8
}
```



CC = número de arcos (8) – número de nodos (7) + 2 = 3

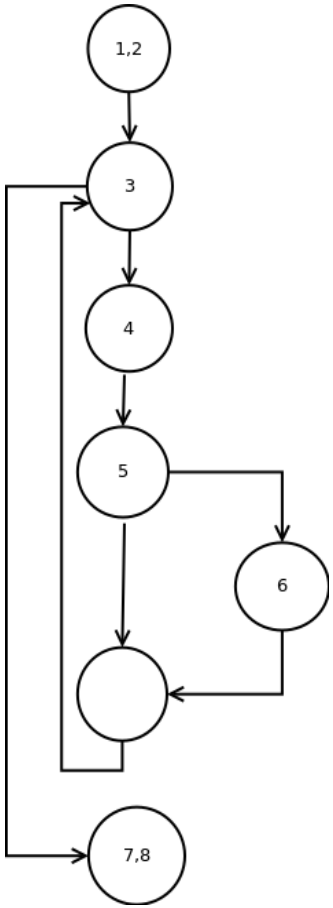
CC = nodos condicionales (2) + 1 = 3
CC = número de regiones cerradas (2) + 1 = 3

Camino 1: 1 – 2 – 3 – 7 – 8
Camino 2: 1 – 2 – 3 – 4 – 5 – 3 – 7 – 8
Camino 3: 1 – 2 – 3 – 4 – 5 – 6 – 3 – 7 – 8

ID	Objetivo de la prueba	Datos de entrada	Procedimiento	Salida esperada	Resultado
T1	Verificar el camino 1	Iterator vacío	• Inicializa una variable para retorno como ArrayList vacío. • Obtiene el iterator con los datos • Mientras el iterator tiene elementos, los agrega al retorno. Como el iterator esta vacío, no agrega nada. • Devuelve el iterator del ArrayList, que estará vacío	Iterator vacío	Correcto
T2	Verificar el camino 2	Iterator con un solo pedido que no esta en evaluación	• Inicializa una variable para retorno como ArrayList vacío. • Obtiene el iterator con los datos • Mientras el iterator tiene elementos, los agrega al retorno. Como el pedido no esta en evaluación, no lo agrega al retorno • Devuelve el iterator del ArrayList, que estará vacío	Iterator vacío	Correcto
T3	Verificar el camino 3	Iterator con un solo pedido que esta en evaluación	• Inicializa una variable para retorno como ArrayList vacío. • Obtiene el iterator con los datos • Mientras el iterator tiene elementos, los agrega al retorno. Como el pedido esta en evaluación, lo agrega al retorno • Devuelve el iterator del ArrayList, que tendrá un	Iterator con un único elemento	Correcto

			elemento		

```
  public Iterator<Pedido> getPedidosIniciados()
{
  Iterator<Pedido> it = this.pedidos.getIterator();                    1
  ArrayList<Pedido> lotesEv = new ArrayList<>();                       2
  while (it.hasNext())                                                 3
  {
    Pedido lot = it.next();                                            4
    if (lot.isIniciado())                                              5
      lotesEv.add(lot);                                                6
  }
  it = lotesEv.iterator();                                             7
  return it;                                                           8
}
```



CC = número de arcos (8) – número de nodos (7) + 2 = 3

CC = nodos condicionales (2) + 1 = 3
CC = número de regiones cerradas (2) + 1 = 3

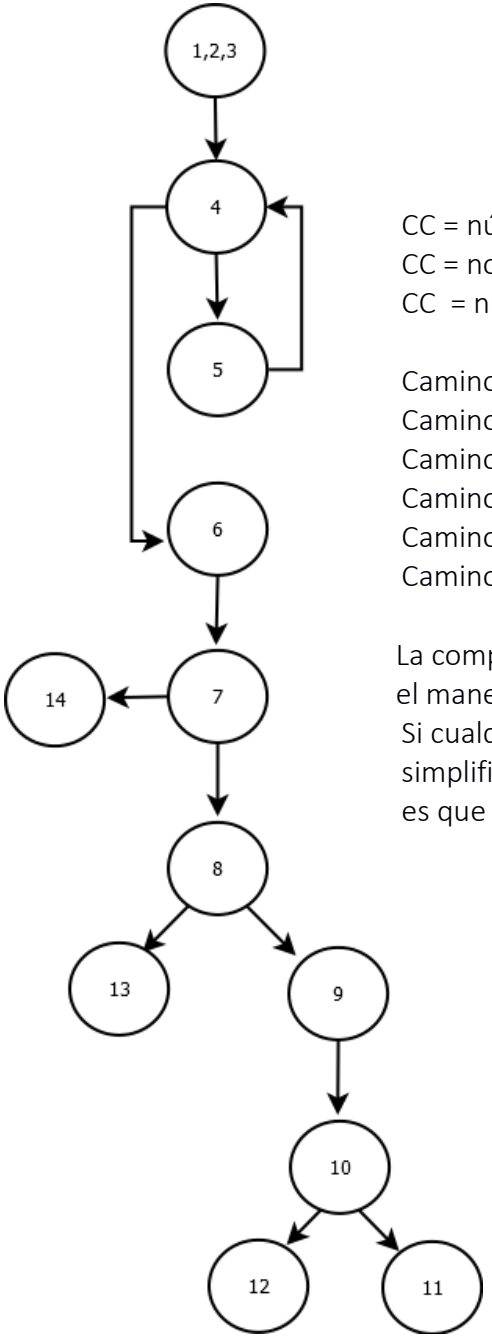
Camino 1: 1 – 2 – 3 – 7 – 8
Camino 2: 1 – 2 – 3 – 4 – 5 – 3 – 7 – 8
Camino 3: 1 – 2 – 3 – 4 – 5 – 6 – 3 – 7 – 8

ID	Objetivo de la prueba	Datos de entrada	Procedimiento	Salida esperada	Resultado
T1	Verificar el camino 1	Iterator vacío	• Inicializa una variable para retorno como ArrayList vacío. • Obtiene el iterator con los datos • Mientras el iterator tiene elementos, los agrega al retorno. Como el iterator esta vacío, no agrega nada. • Devuelve el iterator del ArrayList, que estará vacío	Iterator vacío	Correcto
T2	Verificar el camino 2	Iterator con un solo pedido que no esta iniciado	• Inicializa una variable para retorno como ArrayList vacío. • Obtiene el iterator con los datos • Mientras el iterator tiene elementos, los agrega al retorno. Como el pedido no esta iniciado, no lo agrega al retorno • Devuelve el iterator del ArrayList, que estará vacío	Iterator vacío	Correcto
T3	Verificar el camino 3	Iterator con un solo pedido que esta iniciado	• Inicializa una variable para retorno como ArrayList vacío. • Obtiene el iterator con los datos • Mientras el iterator tiene elementos, los agrega al retorno. Como el pedido esta iniciado, lo agrega al retorno • Devuelve el iterator del ArrayList, que tendrá un elemento	Iterator con un único elemento	Correcto

```
public void generarLote()
  throws Exception
{
  String aux = Integer.toString(this.lotes.getProximoNumeroLote());                    1
  int longitud = aux.length();                                                          2
  String numeroLote = "LOT";                                                            3
  for (int i = 0; i < (6 - longitud); i++)                                              4
    numeroLote += "0";                                                                  5
  numeroLote += aux;                                                                    6
  if (this.pedidoActual != null && Verificaciones.verificaNumeroLote(numeroLote))       7
  {
    Lote lote = new Lote(this.pedidoActual, numeroLote);                                8
    if (lote != null)                                                                   9
      this.lotes.agregarNuevo(lote);                                                   10
    else
      throw new Exception("El lote no se creo");                                       11
  }
  else
    throw new Exception("El lote es invalido");                                        12
}
```

El código anterior se desdobla de la siguiente forma para un mejor análisis de caja blanca:

```
public void generarLote()                                                                    
  throws Exception                                                                           
{                                                                                            
  String aux = Integer.toString(this.lotes.getProximoNumeroLote());                    1    
  int longitud = aux.length();                                                          2    
  String numeroLote = "LOT";                                                            3    
  for (int i = 0; i < (6 - longitud); i++)                                              4    
    numeroLote += "0";                                                                  5    
  numeroLote += aux;                                                                    6    
  if (this.pedidoActual != null)                                                        7    
    if(Verificaciones.verificaNumeroLote(numeroLote))                                   8    
    {                                                                                        
      Lote lote = new Lote(this.pedidoActual, numeroLote);                              9    
      if (lote != null)                                                                 10   
          this.lotes.agregarNuevo(lote);                                                11   
      else                                                                                   
         throw new Exception("El lote no se creo");                                     12   
    }                                                                                        
    else                                                                                     
       throw new Exception("El lote es invalido");                                      13   
  else                                                                                       
     throw new Exception(“El lote es invalido”)                                        14   
}
```



CC = número de arcos (12) – número de nodos (12) + 2 = 2
CC = nodos condicionales (4) + 1 = 5
CC = número de regiones cerradas (4) + 1 = 5

Camino 1: 1-2-3-4-5-6-7-8-9-10-11
Camino 2: 1-2-3-4-6-7-8-9-10-11
Camino 3: 1-2-3-4-5-6-7-14
Camino 4: 1-2-3-4-5-6-7-8-9-10-12
Camino 5: 1-2-3-4-6-7-14
Camino 6: 1-2-3-4-6-7-8-9-10-12

La complejidad ciclomática no es idéntica en todas las fórmulas debido a que las ecuaciones no están desarrolladas para el manejo de excepciones. Debido a esto, en este caso se despreciará y se usarán los caminos posibles.
Si cualquiera de los valores de los if pertenecientes a las líneas 7,8 fallan, se lanza la misma excepción. Entonces para simplificar el análisis, se tomará un caso en el que falla el primero (línea 7) y otro en el que todos son correctos. Por esto es que solo se desarrollaron los 6 caminos posibles de arriba, y no las demás posibilidades

ID	Objetivo de la prueba	Datos de entrada	Procedimiento	Salida esperada	Resultado
T 1	Verificar el camino 1	longitud = 5 i = 6-longitud = 1 pedidoActual != null Verificaciones.verificaNumeroLote(numeroLote) = true lote != null	• Obtiene el número del siguiente lote a generar (aux) • Obtiene la longitud de aux (longitud) • Como i=4, entra al ciclo for • Verifica si pedidoActual es distinto de null. Como si lo es, entra al if • Verifica si numeroLote es correcto, como lo es entra al if • Verifica si lote es distinto de null, como lo es entra al if • Agrega un nuevo lote a la lista de lotes	Creación de un nuevo lote y agregación del mismo a la lista de lotes	Correcto
T 2	Verificar el camino 2	longitud = 6 i = 6-longitud = 0 pedidoActual != null Verificaciones.verificaNumeroLote(numeroLote) = true lote != null	• Obtiene el número del siguiente lote a generar (aux) • Obtiene la longitud de aux (longitud) • Como i=5, no entra al ciclo for. Pero el numeroLote se generó en alguna iteración anterior • Verifica si pedidoActual es distinto de null. Como si lo	Creación de un nuevo lote y agregación del mismo a la lista de lotes	Correcto

			es, entra al if • Verifica si numeroLote es correcto, como lo es entra al if • Verifica si lote es distinto de null, como lo es entra al if • Agrega un nuevo lote a la lista de lotes		
T 3	Verificar el camino 3	longitud = 5 i = 6-longitud = 1 pedidoActual = null	• Obtiene el número del siguiente lote a generar (aux) • Obtiene la longitud de aux (longitud) • Como i=4, entra al ciclo for • Verifica si pedidoActual es distinto de null. Como no lo es, no entra al if y lanza una excepción desde la línea 14	Excepción “El lote es invalido”	Correcto
T 4	Verificar el camino 4	longitud = 5 i = 6-longitud = 1 pedidoActual != null Verificaciones.verificaNumeroLote(numeroLote) = true lote = null	• Obtiene el número del siguiente lote a generar (aux) • Obtiene la longitud de aux (longitud) • Como i=4, entra al ciclo for • Verifica si pedidoActual es distinto de null. Como si lo es, entra al if • Verifica si numeroLote es correcto, como lo es entra al	Excepción “El lote no se creo”	Correcto

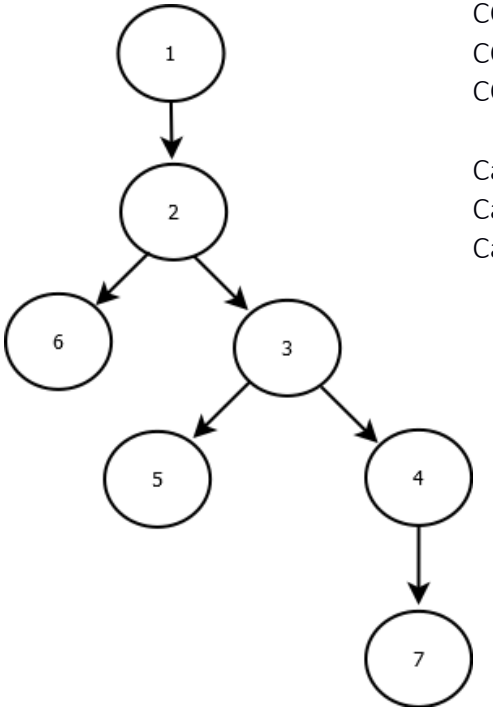
			if • Verifica si lote es distinto de null, como no lo es, no entra al if y lanza una excepción desde la línea 12		
T 5	Verificar el camino 5	longitud = 6 i = 6-longitud = 0 pedidoActual = null	• Obtiene el número del siguiente lote a generar (aux) • Obtiene la longitud de aux (longitud) • Como i=5, no entra al ciclo for. Pero el numeroLote se generó en alguna iteración anterior • Verifica si pedidoActual es distinto de null. Como no lo es, no entra al if y lanza una excepción desde la línea 14	Excepción ”El lote es invalido”	Correcto
T 6	Verificar el camino 6	longitud = 6 i = 6-longitud = 0 pedidoActual != null Verificaciones.verificaNumeroLote(numeroLote) = true lote = null	• Obtiene el número del siguiente lote a generar (aux) • Obtiene la longitud de aux (longitud) • Como i=5, no entra al ciclo for. Pero el numeroLote se generó en alguna iteración anterior • Verifica si pedidoActual es distinto de null. Como si lo	Excepción “El lote no se creo”	Correcto

			es, entra al if • Verifica si numeroLote es correcto, como lo es entra al if • Verifica si lote es distinto de null, como no lo es, no entra al if y lanza una excepción desde la línea 12		

```
public ListaMateriales verificaExistencias(String tipo)                                  
  throws FaltantesException, Exception                                                    
{                                                                                         
  ListaMateriales list;                                                                   1
  if (Verificaciones.verificaTipoCodigo(tipo) &&                                          2
      Verificaciones.verificaCantidad(this.pedidoActual.getCantProduccion()))             
   list = this.stock.verificarExistencias(tipo, this.pedidoActual.getCantProduccion());  3
  else                                                                                    
    throw new Exception("Cantidad o producto invalido");                                  4
  return list;                                                                            5
}                                                                                         
```

El código anterior se desdobla de la siguiente forma para un mejor análisis de caja blanca:

```
public ListaMateriales verificaExistencias(String tipo)                                  
  throws FaltantesException, Exception                                                    
{                                                                                         
  ListaMateriales list;                                                                   1
  if (Verificaciones.verificaTipoCodigo(tipo))                                            2
    if(Verificaciones.verificaCantidad(this.pedidoActual.getCantProduccion()))            3
        list = this.stock.verificarExistencias(tipo, this.pedidoActual.getCantProduccion());  4
    else                                                                                  
       throw new Exception("Cantidad o producto invalido");                               5
  else                                                                                    
     throw new Exception("Cantidad o producto invalido");                                 6
  return list;                                                                            7
}                                                                                         
```



CC = número de arcos (6) – número de nodos (7) + 2 = 1
CC = nodos condicionales (4) + 1 = 5
CC = número de regiones cerradas (4) + 1 = 5

Camino 1: 1-2-3-4-7
Camino 2: 1-2-3-5
Camino 3: 1-2-6

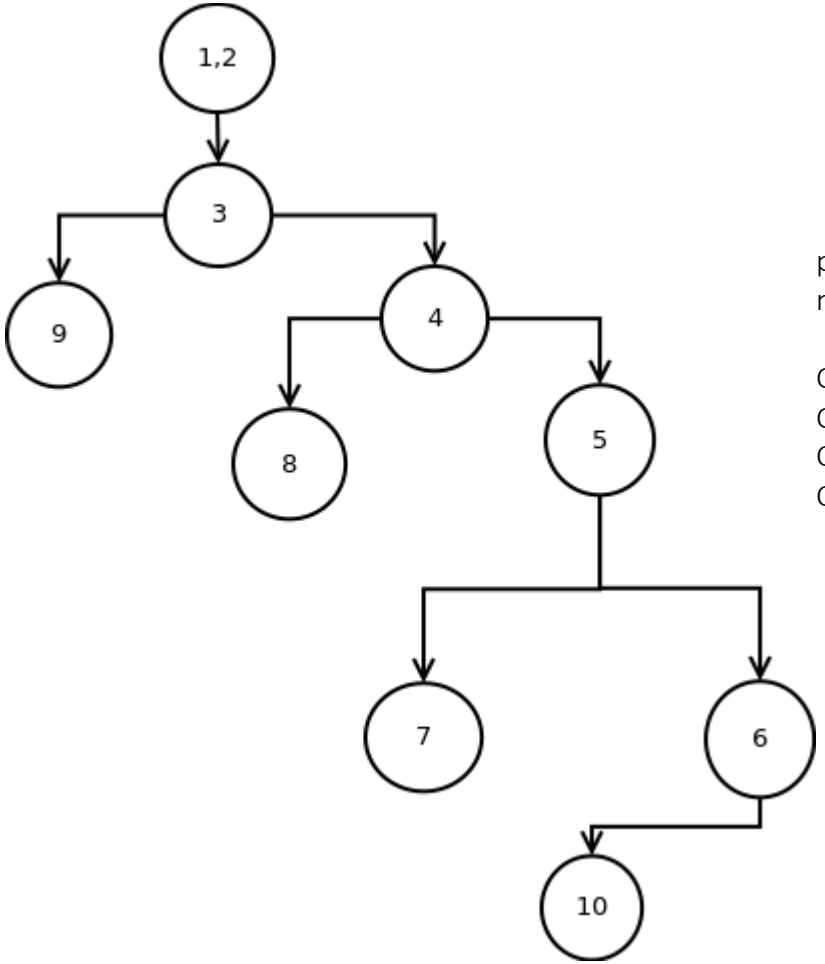
ID	Objetivo de la prueba	Datos de entrada	Procedimiento	Salida esperada	Resultado
T 1	Verificar el camino 1	Verificaciones.verificaTipoCodigo(tipo) = true Verificaciones.verificaCantidad(this.pedidoActual.getCantProduccion()) = true	• Verifica que tipo sea correcto, como lo es, entra al if • Verifica que la cantidad de producción del pedido actual sea correcta (this.pedidoActual.getCantProduccion()), como lo es entra al if • Llama al correspondiente método de ListaMaterialesStock que verifica existencias	Lista final de materiales o lista de materiales faltantes	Correcto
T 2	Verificar el camino 2	Verificaciones.verificaTipoCodigo(tipo) = true Verificaciones.verificaCantidad(this.pedidoActual.getCantProduccion()) = false	• Verifica que tipo sea correcto, como lo es, entra al if • Verifica que la cantidad de producción del pedido actual sea correcta (this.pedidoActual.getCantProduccion()), como no lo es no entra al if y lanza una excepción desde la línea 5	Excepción "Cantidad o producto invalido"	Correcto
T 3	Verificar el camino 3	Verificaciones.verificaTipoCodigo(tipo) = false	• Verifica que tipo sea correcto, como no lo es no entra al if, lanza una excepción desde la línea 6	Excepción "Cantidad o producto invalido"	Correcto

```
  public Observacion crearObservacion(String temaIngresado, String texto)
  throws Exception
{
  Observacion obs = null;                                                                    1
  String númeroLegajo = this.empleadoActual.getLegajo();                                     2
  if (temaIngresado != null && Verificaciones.verificanúmeroLegajo(númeroLegajo) &&          3
      Verificaciones.verificaTexto(texto))
    obs = new Observacion(temaIngresado, GregorianCalendar.getInstance(), númeroLegajo, texto);  4
  else
    throw new Exception("La observacion es invalida");                                       5
  return obs;                                                                              6
}
```

El método anterior se desdobla de la siguiente forma para un mejor análisis de caja blanca

```
public Observacion crearObservacion(String temaIngresado, String texto)
  throws Exception
{
  Observacion obs = null;                                                                    1
  String númeroLegajo = this.empleadoActual.getLegajo();                                     2
  if (temaIngresado != null)                                                                 3
    if(Verificaciones.verificanúmeroLegajo(númeroLegajo))                                    4
      if(Verificaciones.verificaTexto(texto))                                              5
        obs = new Observacion(temaIngresado, GregorianCalendar.getInstance(), númeroLegajo, texto);  6
      else
        throw new Exception("La observacion es invalida");                                 7
    else
      throw new Exception("La observacion es invalida");                                   8
  else
    throw new Exception("La observacion es invalida");                                     9
```

return obs; }	10



CC = número de arcos (8) – número de nodos (9) + 2 = 1
CC = nodos condicionales (3) + 1 = 4
CC = número de regiones cerradas (0) + 1 = 1

Como la complejidad ciclotímica no contempla el caso de ramas que producen excepciones, no se tendrá en cuenta el valor obtenido mediante la formula y se tomaran los siguientes caminos:

Camino 1: 1 – 2 – 3 – 9
Camino 2: 1 – 2 – 3 – 4 – 8
Camino 3: 1 – 2 – 3 – 4 – 5 – 7
Camino 4: 1 – 2 – 3 – 4 – 5 – 6 - 10

ID	Objetivo de la prueba	Datos de entrada	Procedimiento	Salida esperada	Resultado
T1	Verificar el camino 1	TextoIngresado = null	• Se crea una nueva observación en null • Se obtiene el legajo actual • Como el texto ingresado es null, se informa el error	Error	Correcto
T2	Verificar el camino 2	TextoIngresado != null numeroLegajo invalido	• Se crea una nueva observación en null • Se obtiene el legajo actual • Como el texto ingresado es ditinto de null, se verifica el numero de legajo • Como el numero de legajo es invalido, se informa el error	Error	Correcto
T3	Verificar el camino 3	TextoIngresado != null numeroLegajo valido texto invalido	• Se crea una nueva observación en null • Se obtiene el legajo actual • Como el texto ingresado es ditinto de null, se verifica el numero de legajo • Como el numero de legajo es valido, se verifica el texto • Como el texto es invalido, se informa el error	Error	Correcto
T4	Verificar el camino 4	TextoIngresado != null numeroLegajo valido texto valido	• Se crea una nueva observación en null • Se obtiene el legajo actual • Como el texto ingresado es ditinto de null, se verifica el numero de legajo • Como el numero de legajo es valido, se verifica el texto	Nueva observación agregada	Correcto

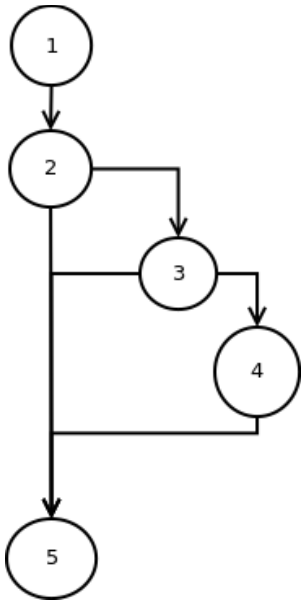
			• Como el texto es valido, se agrega la nueva observación		

Clase Verificaciones

```
 public static boolean verificaCantidad(double cantidad)
{
  return (cantidad > 0.0 && cantidad <= 999.9999);
}
```

El método anterior se desdobla de la siguiente forma para un mejor análisis de caja blanca

```
  public static boolean verificaCantidad(double cantidad)
{
  boolean ret = false;                                          1
  if (cantidad > 0.0)                                           2
     if (cantidad <= 999.9999)                                  3
        ret = true;                                             4
  return ret;                                                   5
}
```



CC = número de arcos (6) – número de nodos (5) + 2 = 3
CC = nodos condicionales (2) + 1 = 3
CC = número de regiones cerradas (2) + 1 = 3

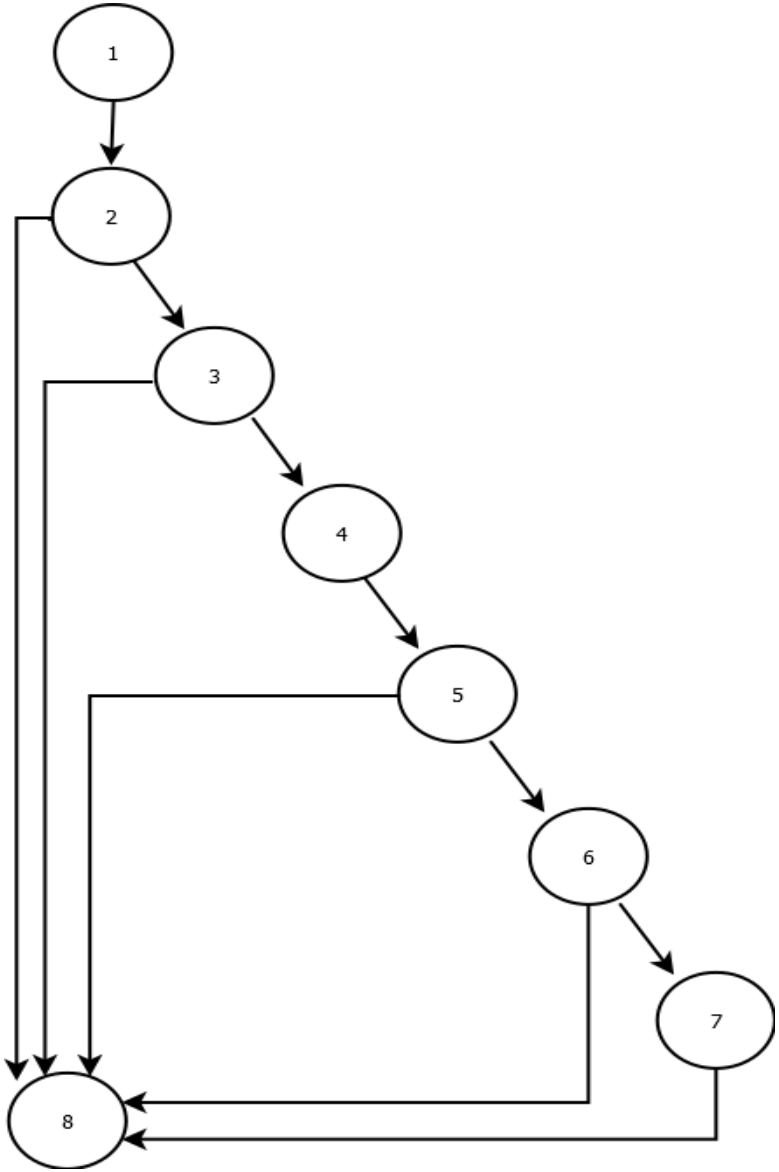
Camino 1: 1 – 2 - 5
Camino 2: 1 – 2 – 3 – 5
Camino 3: 1 – 2 – 3 – 4 – 5

ID	Objetivo de la prueba	Datos de entrada	Procedimiento	Salida esperada	Resultado
T1	Verificar el camino 1	Cantidad <= 0	• Se crea una variable boolean de retorno inicializada en false • Se verifica que la cantidad sea positiva • Como la cantidad no es positiva, se devuelve false	false	Correcto
T2	Verificar el camino 2	Cantidad > 0 Cantidad > 999.9999	• Se crea una variable boolean de retorno inicializada en false • Se verifica que la cantidad sea positiva • Como la cantidad es positiva, se verifica que la cantidad sea menor que 999.9999 • Como la cantidad es mayor que 999.9999, se devuelve false	false	Correcto
T3	Verificar el camino 3	ntidad > 0 Cantidad <= 999.9999	• Se crea una variable boolean de retorno inicializada en false • Se verifica que la cantidad sea positiva • Como la cantidad es positiva, se verifica que la cantidad sea menor que 999.9999 • Como la cantidad es menor o igual que 999.9999, se devuelve true	true	Correcto

```
public static boolean verifica(String str)
{
  boolean ret = false;                                                  1
  if (str != null)                                                      2
    if (str.length() == 9)                                              3
    {
      int num = Integer.parseInt(str.substring(3).trim());              4
      if (num >= 0 && num <= 999999)                                    5
        ret = true;                                                     6
    }
  return ret;                                                           7
}
```

El método anterior se desdobla de la siguiente forma para un mejor análisis de caja blanca:

```
public static boolean verifica(String str)
{
  boolean ret = false;                                                  1
  if (str != null)                                                      2
    if (str.length() == 9)                                              3
    {
      int num = Integer.parseInt(str.substring(3).trim());              4
      if (num >= 0)                                                     5
        if(num <= 999999)                                               6
            ret = true;                                                 7
    }
  return ret;                                                           8
}
```



CC = número de arcos (11) – número de nodos (8) + 2 = 5
CC = nodos condicionales (4) + 1 = 5
CC = número de regiones cerradas (4) + 1 = 5

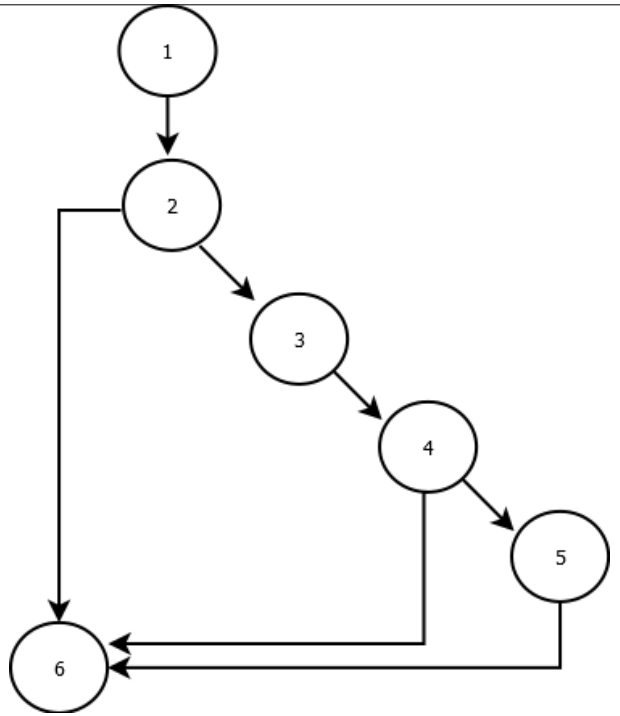
Camino 1: 1-2-3-4-5-6-7-8
Camino 2: 1-2-3-4-5-6-8
Camino 3: 1-2-3-4-5-8
Camino 4: 1-2-3-7
Camino 5: 1-2-8

ID	Objetivo de la prueba	Datos de entrada	Procedimiento	Salida esperada	Resultado
T1	Verificar el camino 1	str != null str.length = 9 0 <= num <=999999	• Compara si str != null, como lo es entra al if • Compara si la longitud de str es igual a 9, como lo es entra al if • Saca el substring correspondiente desde la tercera posición hasta la última del String str, y lo convierte a entero (num) • Verifica si num se encuentra entre 0 y 999999, como si se encuentra, entra a ambos if • Cambia el valor de ret a true	true	Correcto
T2	Verificar el camino 2	str != null str.length = 9 num >= 0 num > 999999	• Compara si str != null, como lo es entra al if • Compara si la longitud de str es igual a 9, como lo es entra al if • Saca el substring correspondiente desde la tercera posición hasta la última del String str, y lo convierte a entero (num) • Verifica si num es mayor o igual a 0, como lo es entra al	false	Correcto

			if • Verifica si num es menor o igual a 999999, como no lo es y no posee else va directamente hacia la línea 8 • Retorna ret		
T3	Verificar el camino 3	str != null str.length = 9 num < 0	• Compara si str != null, como lo es entra al if • Compara si la longitud de str es igual a 9, como lo es entra al if • Saca el substring correspondiente desde la tercera posición hasta la última del String str, y lo convierte a entero (num) • Verifica si num es mayor o igual a 0, como no lo es no entra al if y como no posee else va directamente a la línea 8 • Retorna ret	false	Correcto
T4	Verificar el camino 4	str != null str.length != 9	• Compara si str != null, como lo es entra al if • Compara si la longitud de str es igual a 9, como no lo es y no posee else el if, va directamente a la línea 8 • Retorna ret	false	Correcto

T5	Verificar el camino 5	str = null	• Compara si str != null, como no lo es no entra al if, como no posee else, va directamente a la línea 8 • Retorna ret	false	

public static boolean verificaNumeroLote(String numeroLote)
{
boolean ret = false;
if (numeroLote != null)
{
String aux = numeroLote.substring(0, 3);
if (aux.compareTo("LOT") == 0)
ret = verifica(numeroLote);
}
return ret;
}
	
	1
	2
	
	3
	4
	5
	
	6



CC = número de arcos (7) – número de nodos (6) + 2 = 3
CC = nodos condicionales (2) + 1 = 3
CC = número de regiones cerradas (2) + 1 = 3

Camino 1: 1-2-3-4-5-6
Camino 2: 1-2-3-4-6
Camino 3: 1-2-6

ID	Objetivo de la prueba	Datos de entrada	Procedimiento	Salida esperada	Resultado
T1	Verificar el camino 1	numeroLote != null numeroLote = “LOT0000001” aux = “LOT” verifica(numeroLote) = false	• Compara si numeroLote != null, como lo es entra al if • Saca el substring correspondiente desde la primera posición hasta la tercera del String numeroLote (aux) • Compara si aux es igual a “LOT”, como lo es entra al if • Se le asigna a ret el valor que devuelve la función verifica(numeroLote) • Retorna ret	false	Correcto
		numeroLote != null numeroLote = “LOT000001” aux = “LOT” verifica(numeroLote) = true		true	

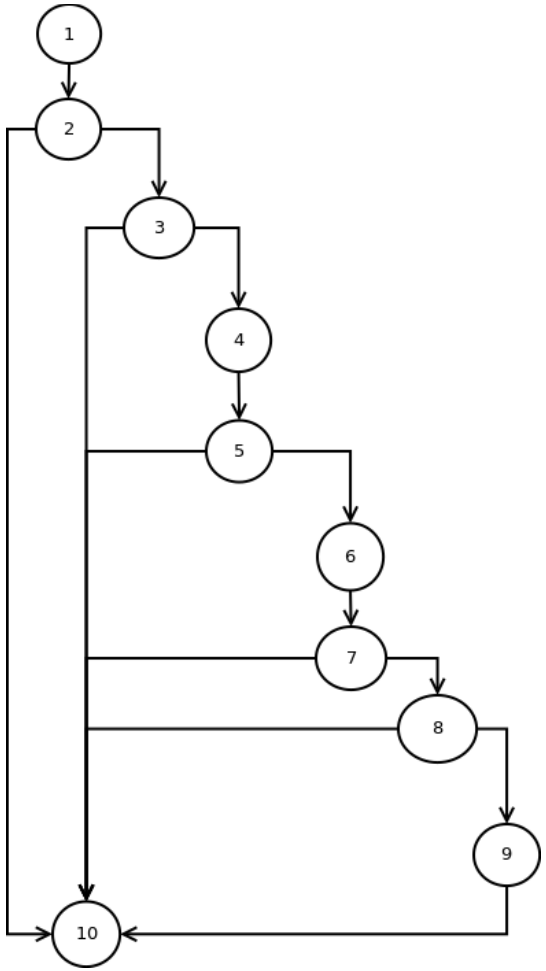
T2	Verificar el camino 2	numeroLote != null aux != “LOT”	• Compara si numeroLote != null, como lo es entra al if • Saca el substring correspondiente desde la primera posición hasta la tercera del String numeroLote (aux) • Compara si aux es igual a “LOT”, como no lo es no entra al if, y como el if no posee else, va directo a la línea 6 • Retorna ret	false	Correcto
T3	Verificar el camino 3	numeroLote = null	• Compara si str != null, como no lo es no entra al if, como el if no posee else va directo a la línea 6 • Retorna ret	false	Correcto

```
 public static boolean verificaCodigo(String codigo)
{
  boolean ret = false;
  if (codigo != null)
    if (codigo.length() == 8)
    {
      String aux = codigo.substring(0, 3);
      if (aux.compareTo("MAT") == 0)
      {
        int num = Integer.parseInt(codigo.substring(3).trim());
        if (num >= 0 && num <= 99999)
          ret = true;
      }
    }
  return ret;
}
```

El método anterior se desdobla de la siguiente forma para un mejor análisis de caja blanca:

```
    public static boolean verificaCodigo(String codigo) {
      boolean ret = false;                                                    1
      if (codigo != null)                                                     2
         if (codigo.length() == 8) {                                          3
            String aux = codigo.substring(0, 3);                              4
            if (aux.compareTo("MAT") == 0) {                                  5
               int num = Integer.parseInt(codigo.substring(3).trim());        6
               if (num >= 0)                                                  7
                  if (num <= 99999)                                           8
                     ret = true;                                              9
            }
         }
      return ret;                                                            10
```

}	



CC = número de arcos (14) – número de nodos (10) + 2 = 6
CC = nodos condicionales (5) + 1 = 6
CC = número de regiones cerradas (5) + 1 = 6

Camino 1: 1 – 2 – 10
Camino 2: 1 – 2 – 3 – 10
Camino 3: 1 – 2 – 3 – 4 – 5 – 10
Camino 4: 1 – 2 – 3 – 4 – 5 – 6 – 7 – 10
Camino 5: 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 10
Camino 6: 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – 10

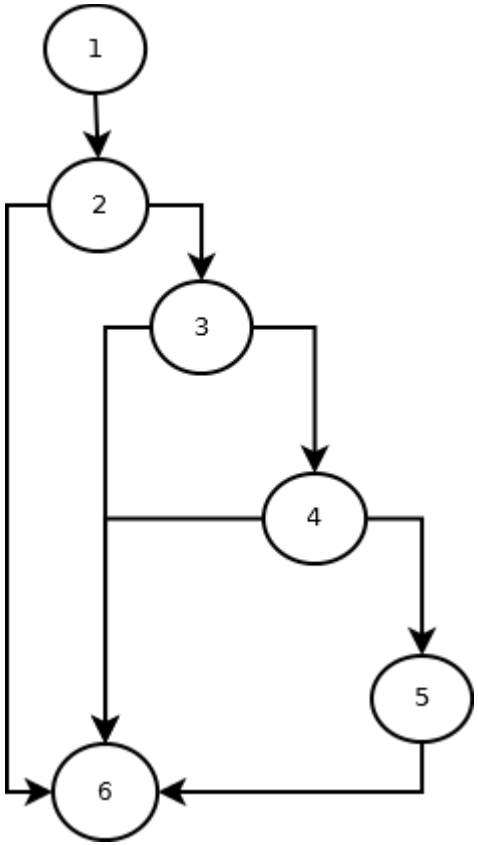
ID	Objetivo de la prueba	Datos de entrada	Procedimiento	Salida esperada	Resultado
T1	Verificar camino 1	Codigo == null	• Verifica que el codigo sea distinto de null • Como no lo es, devuelve false	false	Correcto
T2	Verificar camino 2	Codigo != null codigo.length != 8	• Verifica que el codigo sea distinto de null • Como lo es, verifica que su longitud sea igual a 8 • Como no lo es, devuelve false	false	Correcto
T3	Verificar camino 3	Codigo != null codigo.length == 8 Sin el prefijo MAT	• Verifica que el codigo sea distinto de null • Como lo es, verifica que su longitud sea igual a 8 • Como lo es, verifica que comience con el prefijo “MAT” • Como no cumple esto, devuelve false	false	Correcto
T4	Verificar camino 4	Codigo != null codigo.length == 8 Con el prefijo MAT numero < 0	• Verifica que el codigo sea distinto de null • Como lo es, verifica que su longitud sea igual a 8 • Como lo es, verifica que comience con el prefijo “MAT” • Como cumple esto, verifica que su numero sea positivo	false	Correcto

			• Como no lo es, devuelve false		
T5	Verificar camino 5	Codigo != null codigo.length == 8 Con el prefijo MAT numero > 0 numero > 999	• Verifica que el codigo sea distinto de null • Como lo es, verifica que su longitud sea igual a 8 • Como lo es, verifica que comience con el prefijo “MAT” • Como cumple esto, verifica que su numero sea positivo • Como lo es, verifica que el numero sea menor que 99999 • Como no lo cumple, devuelve false	false	Correcto
T6	Verificar camino 6	Codigo != null codigo.length == 8 Con el prefijo MAT numero > 0 numero < 999	• Verifica que el codigo sea distinto de null • Como lo es, verifica que su longitud sea igual a 8 • Como lo es, verifica que comience con el prefijo “MAT” • Como cumple esto, verifica que su numero sea positivo • Como lo es, verifica que el numero sea menor que 99999 • Como lo cumple, devuelve true	true	Correcto

```
public static boolean verificaDescripcion(String descripcion)
{
  boolean ret = false;
  if (descripcion != null)
    if (descripcion.length() <= 100 && descripcion.length() > 0)
      ret = true;
  return ret;
}
```

El método anterior se desdobla de la siguiente forma para un mejor análisis de caja blanca:

```
public static boolean verificaDescripcion(String descripcion) {
      boolean ret = false;                                          1
      if (descripcion != null)                                      2
         if (descripcion.length() <= 100)                           3
            if (descripcion.length() > 0)                           4
               ret = true;                                          5
      return ret;                                                   6
   }
```



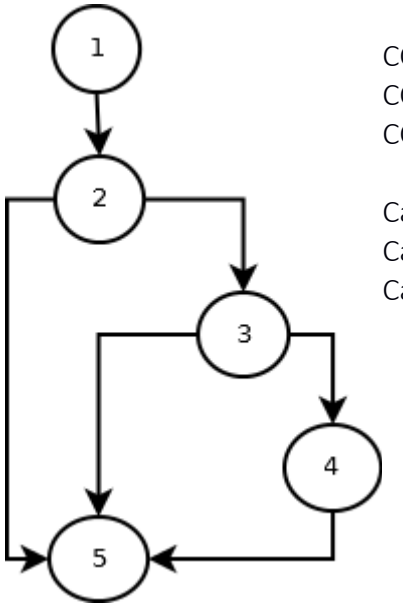
CC = número de arcos (8) – número de nodos (6) + 2 = 4
CC = nodos condicionales (3) + 1 = 4
CC = número de regiones cerradas (3) + 1 = 4

Camino 1: 1-2-3-4-5-6
Camino 2: 1-2-3-4-6
Camino 3: 1-2-3-6
Camino 3: 1-2-6

ID	Objetivo de la prueba	Datos de entrada	Procedimiento	Salida esperada	Resultado
T1	Verificar el camino 1	Descripcion != null descripcion.length > 0 descripcion.length <= 100	• Verifica si la descirpcion es distinta de null, como lo es entra al if • Verifica si descripción es mayor que 0, como lo es entra al if • Verifica si descripción es menor o igual a 100, como lo es entra al if • Cambia el valor a ret por true • Retorna ret	true	Correcto
T2	Verificar el camino 2	Descripcion != null descripcion.length > 0 descripcion.length > 100	• Verifica si descripción es mayor que 0, como lo es entra al if • Verifica si descripción es menor o igual a 100, como o lo es no entra al if y como no posee else va directo a la línea 5 • Retorna ret	false	Correcto
T3	Verificar el camino 3	Descripcion != null descripcion.length == 0	• Verifica si descripción es mayor que 0, como no lo es no entra al if y como no posee else va directamente a la línea 5	false	Correcto

T4	Verifica el camino 4	Descripcion == null	• Verifica que la descripción sea distinta de null • Como no lo es, devuelve false	false	

```
public static boolean verificaSector(String sector) {
   boolean ret = false;                                                                    1
   if (sector != null)                                                                     2
      if (sector == ListaEmpleados.VENTAS || sector == ListaEmpleados.CONTABILIDAD ||      3
         sector == ListaEmpleados.INSPECCION || sector == ListaEmpleados.PRODUCCION)
         ret = true;                                                                       4
   return ret;                                                                             5
}
```

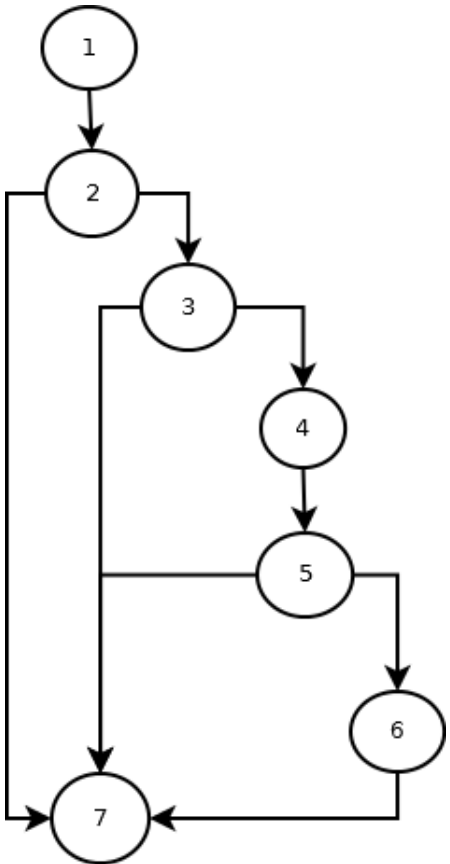


CC = número de arcos (6) – número de nodos (5) + 2 = 3
CC = nodos condicionales (2) + 1 = 3
CC = número de regiones cerradas (2) + 1 = 3

Camino 1: 1-2-3-4-5
Camino 2: 1-2-3-5
Camino 3: 1-2-5

ID	Objetivo de la prueba	Datos de entrada	Procedimiento	Salida esperada	Resultado
T1	Verificar el camino 1	Sector != null sector = “Ventas” || sector = “Contabilidad” || sector = “Inspeccion y Calidad” || sector = “Produccion”	• Verifica que el sector sea distinto de null, y como lo es entra en el if • Verifica que sector sea alguno de los valores correctos, como lo es entra al if • Modifica el valor de ret a true • Retorna ret	true	Correcto
T2	Verificar el camino 2	Sector != null sector != “Ventas” || sector != “Contabilidad” || sector != “Inspeccion y Calidad” || sector != “Produccion”	• Verifica que el sector sea distinto de null, y como lo es entra en el if • Verifica que sector sea alguno de los valores correctos, como no es igual a ninguno no entra al if, como el mismo no posee else, va directamente a la línea 5 • Retorna ret	false	Correcto
T3	Verifica el camino 3	Sector == null	• Verifica que el sector sea distinto de null • Como no lo es va por el else a la linea 5 y retorna false	false	Correcto

```
   public static boolean verificaNumeroPedido(String numeroPedido) {
    boolean ret = false;                                                    1
    if (numeroPedido != null)                                               2
       if (numeroPedido.length() == 9) {                                    3
          String aux = numeroPedido.substring(0, 3);                        4
          if (aux.compareTo("PED") == 0) {                                  5
             ret = Verificaciones.verifica(numeroPedido);                   6
          }
       }
    return ret;                                                             7
}
```



CC = número de arcos (9) - número de nodos (7) + 2 = 4
CC = nodos condicionales (3) + 1 = 4
CC = número de regiones cerradas (3) + 1 = 4

Camino 1: 1 – 2 - 7
Camino 2: 1 – 2 – 3 – 7
Camino 3: 1 – 2 – 3 – 4 – 5 – 7
Camino 4: 1 – 2 – 3 – 4 – 5 – 6 – 7

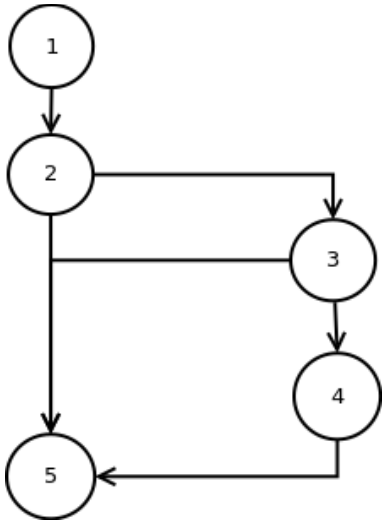
ID	Objetivo de la prueba	Datos de entrada	Procedimiento	Salida esperada	Resultado
T1	Verificar el camino 1	NumeroPedido == null	• Se crea una variable boolean de retorno inicializada en false • Verifica que el numero de pedido no sea null • Como es null, devuelve false	false	Correcto
T1	Verificar el camino 1	NumeroPedido != null NumeroPedido de longitud distinta de 9	• Se crea una variable boolean de retorno inicializada en false • Se verifica que el pedido sea disntito de null, cosa que cumple • Se verifica que la longitud del • numero de pedido sea igual a 9 • Como la longitud no es 9, devuelve false	false	Correcto
T2	Verificar el camino 2	NumeroPedido != null NumeroPedido de longitud 9 Sin prefijo PED	• Se crea una variable boolean de retorno inicializada en false • Se verifica que el pedido sea disntito de null, cosa que cumple • Se verifica que la longitud del numero de pedido sea igual a 9 • Como la longitud es 9, se obtienen los 3 primeros caracteres del numero y se verifica que sean “PED” • Como no se verifica que sean “PED”, devuelve false	false	Correcto
T3	Verificar el camino 3	NumeroPedido != null NumeroPedido de longitud 9 Con prefijo PED	• Se crea una variable boolean de retorno inicializada en false • Se verifica que el pedido sea disntito de null, cosa que cumple	true	Correcto

			• Se verifica que la longitud del numero de pedido sea igual a 9 • Como la longitud es 9, se obtienen los 3 primeros caracteres del numero y se verifica que sean “PED” • Como se verifica que sean “PED”, devuelve true		

```
public static boolean verificaCantProduccion(int cantProduccion)
{
  boolean ret = false;
  if (cantProduccion > 0 && cantProduccion < 999)
    ret = true;
  return ret;
}
```

El método anterior se desdobla de la siguiente forma para un mejor análisis de caja blanca

```
public static boolean verificaCantProduccion(int cantProduccion)
{
  boolean ret = false;                                  1
  if (cantProduccion > 0)                               2
     if (cantProduccion < 999)                          3
        ret = true;                                     4
  return ret;                                           5
}
```

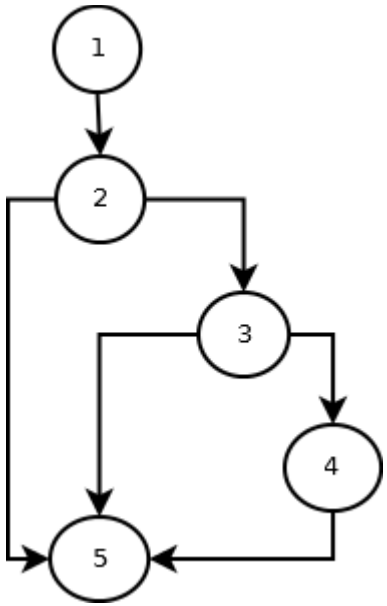


CC = número de arcos (6) - número de nodos (5) + 2 = 3
CC = nodos condicionales (2) + 1 = 3
CC = número de regiones cerradas (2) + 1 = 3

Camino 1: 1 – 2 – 5
Camino 2: 1 – 2 – 3 – 5
Camino 3: 1 – 2 – 3 – 4 – 5

ID	Objetivo de la prueba	Datos de entrada	Procedimiento	Salida esperada	Resultado
T1	Verificar el camino 1	CantProduccion <= 0	• Se crea una variable boolean de retorno inicializada en false • Se verifica que la cantidad a producir sea positiva • Como no es positiva, devuelve false	false	Correcto
T2	Verificar el camino 2	CantProduccion > 0 CantProduccion >= 999	• Se crea una variable boolean de retorno inicializada en false • Se verifica que la cantidad a producir sea positiva • Como es positiva, se verifica que sea menor a 999 • Como es mayor a 999, devuelve falseSe crea una variable boolean de retorno inicializada en false • Se verifica que la cantidad a producir sea positiva • Como es positiva, se verifica que sea menor a 999 • Como es mayor a 999, devuelve false	false	Correcto
T3	Verificar el camino 3	CantProduccion > 0 CantProduccion < 999	• Se crea una variable boolean de retorno inicializada en false • Se verifica que la cantidad a producir sea positiva • Como es positiva, se verifica que sea menor a 999 • Como es menor a 999, devuelve true	true	Correcto

public static boolean verificaTexto(String texto) {
  boolean ret = false;
  if (texto != null)
  if (texto.length() <= 500)
  ret = true;
  return ret;
}
	
	1
	2
	3
	4
	5

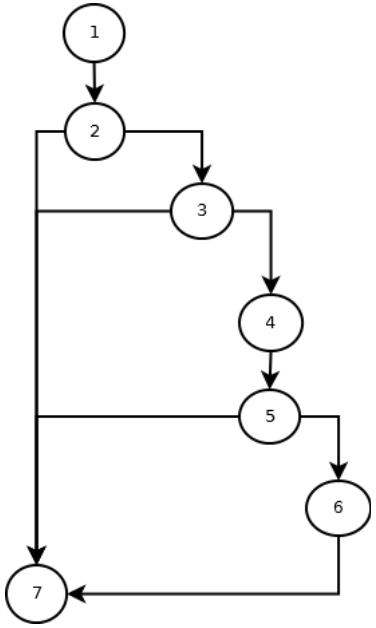


CC = número de arcos (6) - número de nodos (5) + 2 = 3
CC = nodos condicionales (2) + 1 = 3
CC = número de regiones cerradas (2) + 1 = 3

Camino 1: 1 – 2 – 5
Camino 2: 1 – 2 – 3 – 5
Camino 3: 1 – 2 – 3 – 4 – 5

ID	Objetivo de la prueba	Datos de entrada	Procedimiento	Salida esperada	Resultado
T1	Verificar el camino 1	Texto == null	• Se crea una variable boolean de retorno inicializada en false • Se verifica que el texto sea disntito de null. Como no cumple esto, devuelve false	false	Correcto
T2	Verificar el camino 2	Texto != null Texto con longitud mayor a 500	• Se crea una variable boolean de retorno inicializada en false • Se verifica que el texto sea disntito de null, condicion que se cumple • Se verifica si la longitud del texto es menor o igual a 500 • Como no verifica, se devuelve falseSe crea una variable boolean de retorno inicializada en false • Se verifica si la longitud del texto es menor o igual a 500 • Como no verifica, se devuelve false	false	Correcto
T3	Verificar el camino 3	Texto != null Texto con longitud menor a 500	• Se crea una variable boolean de retorno inicializada en false • Se verifica que el texto sea disntito de null, condicion que se cumple • Se verifica si la longitud del texto es menor o igual a 500 • Como se verifica, se devuelve true	true	Correcto

public static boolean verificaNumeroLegajo(String codigo)
{
boolean ret = false;
 	
if (codigo != null)
if (codigo.length() == 9)
{
String aux = codigo.substring(0, 3);
if (aux.compareTo("LEG") == 0)
{
ret = Verificaciones.verifica(codigo);
}
}
return ret;
}
	
	1
	
	2
	3
	
	4
	5
	
	6
	
	7



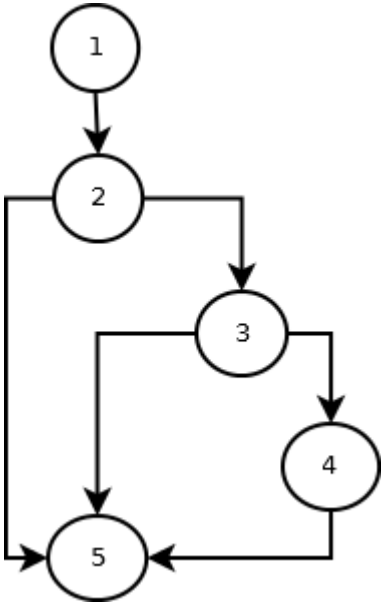
CC = número de arcos (9) - número de nodos (7) + 2 = 4
CC = nodos condicionales (3) + 1 = 4
CC = número de regiones cerradas (3) + 1 = 4

Camino 1: 1 – 2 – 7
Camino 2: 1 – 2 – 3 – 7
Camino 3: 1 – 2 – 3 – 4 – 5 – 7
Camino 4: 1 – 2 – 3 – 4 – 5 – 6 – 7

ID	Objetivo de la prueba	Datos de entrada	Procedimiento	Salida esperada	Resultado
T1	Verificar el camino 1	NumeroLegajo == null	• Se crea una variable boolean de retorno inicializada en false • Se verifica si el numero de legano es distinto de null • Como no verifica la condición anterior, devuelve false	false	
T2	Verificar el camino 2	NumeroLegajo de longitud distinta de 9	• Se crea una variable boolean de retorno inicializada en false • Se verifica si el numero de legajo es distinto de null, condición que cumple. • Se verifica que la longitud del numero de legajo sea igual a 9 • Como la longitud no es 9, devuelve false	false	Correcto
T3	Verificar el camino 3	NumeroLegajo de longitud 9 Sin prefijo LEG	• Se crea una variable boolean de retorno inicializada en false • Se verifica si el numero de legajo es distinto de null, condición que cumple. • Se verifica que la longitud del numero de legajo sea igual a 9 • Como la longitud es 9, se obtienen los 3 primeros caracteres del numero y se verifica que sean “LEG” • Como no se verifica que sean “LEG”, devuelve false	false	Correcto
T4	Verificar el camino 4	NumeroLegajo de longitud 9 Con prefijo LEG	• Se crea una variable boolean de retorno inicializada en false • Se verifica si el numero de legajo es distinto de null,	true	Correcto

			condición que cumple. • Se verifica que la longitud del numero de legajo sea igual a 9 • Como la longitud es 9, se obtienen los 3 primeros caracteres del numero y se verifica que sean “LEG” • Como se verifica que sean “LEG”, devuelve true		

public static boolean verificaNombreyApellido(String nya) {	
boolean ret = false;	1
if (nya != null)	2
if (nya.length() > 0 && nya.length() <= 100)	3
ret = true;	4
return ret;	5
}	



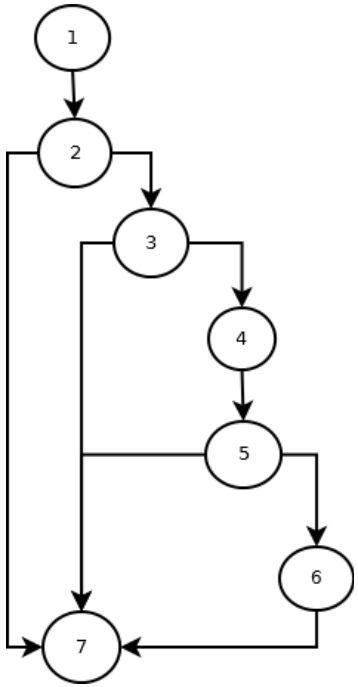
CC = número de arcos (6) - número de nodos (5) + 2 = 3
CC = nodos condicionales (2) + 1 = 3
CC = número de regiones cerradas (2) + 1 = 3

Camino 1: 1 – 2 – 5
Camino 2: 1 – 2 – 3 – 5
Camino 3: 1 – 2 – 3 – 4 – 5

ID	Objetivo de la prueba	Datos de entrada	Procedimiento	Salida esperada	Resultado
T1	Verificar el camino 1	Nya == null	• Se crea una variable boolean de retorno inicializada en false • Se verifica que el nombre y apellido sea distinto de null • Como no lo cumple, devuelve false	false	Correcto
T2	Verificar el camino 2	Nya != null Nya de longitud <= 0	• Se crea una variable boolean de retorno inicializada en false • Se verifica que el nombre y apellido sea distinto de null, condicion que se cumple • Se verifica que la longitud del nombre y apellido • Como la longitud es menor o igual a cero, devuelve false	false	Correcto
T3	Verificar el camino 3	Nya != null Nya de longitud > 0 Nya de longitud > 100	• Se crea una variable boolean de retorno inicializada en false • Se verifica que el nombre y apellido sea distinto de null, condicion que se cumple • Se verifica que la longitud del nombre y apellido • Como la longitud es mayor a cero, se verifica que sea menor o igual a 100 • Como no verifica, devuelve false	false	Correcto
T4	Verificar el camino 4	Nya != null Nya de longitud > 0 Nya de longitud <= 100	• Se crea una variable boolean de retorno inicializada en false • Se verifica que el nombre y apellido sea distinto de null, condicion que se cumple • Se verifica que la longitud del	true	Correcto

			nombre y apellido • Como la longitud es mayor a cero, se verifica que sea menor o igual a 100 • Como verifica, devuelve true		

public static boolean verificaTipoCodigo(String tipoCodigo) {	
boolean ret = false;	1
if (tipoCodigo != null)	2
if (tipoCodigo.length() == 9) {	3
String aux = tipoCodigo.substring(0, 3);	4
if (aux.compareTo("TIP") == 0) {	5
ret = Verificaciones.verifica(tipoCodigo);	6
}	
}	
return ret;	7
}	



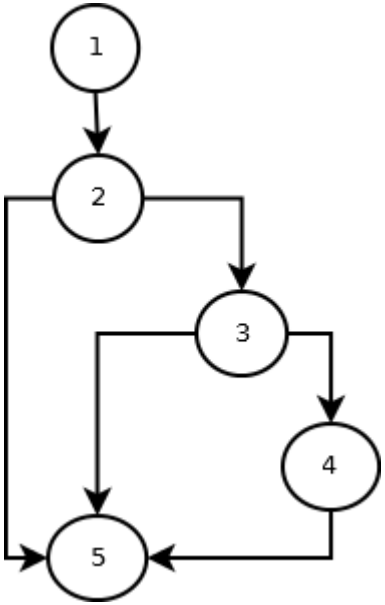
CC = número de arcos (9) - número de nodos (7) + 2 = 4
CC = nodos condicionales (3) + 1 = 4
CC = número de regiones cerradas (3) + 1 = 4

Camino 1: 1 – 2 – 7
Camino 2: 1 – 2 – 3 – 7
Camino 3: 1 – 2 – 3 – 4 – 5 – 7
Camino 4: 1 – 2 – 3 – 4 – 5 – 6 – 7

ID	Objetivo de la prueba	Datos de entrada	Procedimiento	Salida esperada	Resultado
T1	Verificar el camino 1	Codigo == null	• Se crea una variable boolean de retorno inicializada en false • Se verifica que el codigo sea distinto de null • Como no se cumple la condicionale, devuelve false	false	Correcto
T2	Verificar el camino 2	Codigo != null código de longitud distinta de 9	• Se crea una variable boolean de retorno inicializada en false • Se verifica que el codigo no sea null, condicion que se cumple • Se verifica que la longitud del código sea igual a 9 • Como la longitud no es 9, devuelve false	false	Correcto
T3	Verificar el camino 3	Codigo != null código de longitud 9 Sin prefijo TIP	• Se crea una variable boolean de retorno inicializada en false • Se verifica que el codigo no sea null, condicion que se cumple • Se verifica que la longitud del código sea igual a 9 • Como la longitud es 9, se obtienen los 3 primeros caracteres del numero y se verifica que sean “TIP” • Como no se verifica que sean “TIP”, devuelve false	false	Correcto
T4	Verificar el camino 4	Codigo != null código de longitud 9 Con prefijo TIP	• Se crea una variable boolean de retorno inicializada en false • Se verifica que el codigo no sea null, condicion que se cumple • Se verifica que la longitud del código sea igual a 9 • Como la longitud es 9, se	true	Correcto

			obtienen los 3 primeros caracteres del numero y se verifica que sean “TIP” • Como se verifica que sean “TIP”, devuelve true		

```
public static boolean verificaTipoProducto(String tipoProducto) {
   boolean ret = false;                                                                  1
   if (tipoProducto != null)                                                             2
      if (ListaMaterialesStock.getInstance().getCodigoProd().containsKey(tipoProducto))  3
         ret = true;                                                                     4
   return ret;                                                                           5
}
```



CC = número de arcos (6) - número de nodos (5) + 2 = 3
CC = nodos condicionales (2) + 1 = 3
CC = número de regiones cerradas (2) + 1 = 3

Camino 1: 1 – 2 – 5
Camino 2: 1 – 2 – 3 – 5
Camino 3: 1 – 2 – 3 – 4 – 5

ID	Objetivo de la prueba	Datos de entrada	Procedimiento	Salida esperada	Resultado
T1	Verificar el camino 1	tipoProducto == null	• Se crea una variable boolean de retorno inicializada en false • Se verifica que el tipo de producto sea distinto de null • Como no se cumple la condicionale, devuelve false	false	Correcto
T2	Verificar el camino 2	TipoProducto != null Producto no contenido en la lista de stock	• Se crea una variable boolean de retorno inicializada en false • Se verifica que el tipo de producto sea distinto de null • Como cumple la condicion anterior, se verifica que el tipo de producto este contenido en la lista de materiales stock • Como no cumple la condicion, devuelve false	false	Correcto
T3	Verificar el camino 3	TipoProducto != null Producto contenido en la lista de stock	• Se crea una variable boolean de retorno inicializada en false • Se verifica que el tipo de producto sea distinto de null • Como cumple la condicion anterior, se verifica que el tipo de producto este contenido en la lista de materiales stock • Como cumple la condicion, devuelve true	true	Correcto