Grado Ingeniería Biomédica

Algorítmos y Estructuras de datos

MEMORIA PRÁCTICA II

Aplicación para gestionar cultivos de bacterias

Sancha Kindler Von Knobloch Luengo

Introducción

Con el propósito de abordar todos las cuestiones planteadas, he decidido estructurar el análisis de mi proyecto de “menor a mayor”. Es decir, empezando por las clases más simples y siguiendo por las clases que tienen como atributo estas clases previas y así sucesivamente. Esto no solamente facilitará el entendimiento general del código al llegar a la clase InputOutput en la que se hace uso de todo el compendio de clases, sino que también reflejará la manera en la que he realizado la practica, fabricando objetos que formarían parte de objetos más completos/desarrollados. Por cada clase, haré una breve explicación de :

1. Uso de estructuras y conceptos de la programación orientada a objetos utilizados
2. Uso de excepciones
3. Lista de fallos conocidos y posibles mejoras
4. Código (sin main ni javadoc)

Asimismo, añadiré un UML que representará la identidad,el estado y el comportamiento del objeto.

Las clases las iré comentando por paquetes. Al principio del paquete explicaré por qué he decidido incluir dichas clases dentro del paquete en cuestión.

Paquete ClasesAuxiliares

En el paquete de ClasesAuxiliares se guardan múltiples clases necesarias para la creacion de un objeto de tipo poblacion de bacteria. Los tres componentes centrales son por un lado la fecha, la comida y la clase Bacteria.

Clase Fecha

1. **Uso de estructuras y conceptos de POO utilizados**

Consiste de tres atributos el dia,mes y año. Los atributos son de tipo final puesto que no se cambiarán mas adelante y por lo tanto la clase no consta de setters. Los atributos tambien son privados consiguiendo un mayor grado de encapsulación, y su acceso se realiza mediante getters. Tiene una función anteriorA que se utiliza a la hora de crear una poblacion de bacterias, que debe tener varios objetos de tipo fecha donde la fecha final debe ser estrictamente mayor a la inicial. A este metodo se le llama desde la clase InputOutput y por esta razón es público.

2. Uso de excepciones

El constructor de la clase lanza una objeto Excepcionfecha, una clase particular creada que hereda de exception (que estrictamente debe ser gestionada) en el caso de que se introduzca una fecha con valores no validos.

3. Lista de fallos conocidos y posibles mejoras

Al haber realizado mi primer práctica con esta clase, he decidido dedicar mas tiempo a mejorar otros aspectos que los de la clase fecha, sin embargo, sería mucho mas eficiente hacer uso de una clase ya creada de Java. Esto se debe principalmente a que clases como LocalDate ya tienen todos los metodos necesarios para poder realizar todos los procedimientos de la practica. Asimismo, hay ciertos metodos que no he podido crear y que por lo tanto he tenido que recurrir a la creación de objetos de tipo LocalDate, como la cantidad de dias entre dos fechas, haciendo el código menos eficiente. Si tuviese que volver a realizar la practica, instanciaría directamente esta clase como atributo de las clases comida.

4. Código

```
public class Fecha {

   private final int dia;
   private final int mes;
   private final int año;

   public Fecha(int dia, int mes, int año) throws Excepcionfecha {

      if (dia > 31 || dia < 1) {
         throw new Excepcionfecha("dia");
      }
      if (mes > 12 || mes < 1) {
         throw new Excepcionfecha("mes");
      }
      //NO PONGO LIMITE SUPERIOR PORQUE PODEMOS ESTAR SIMULANDO UN EXPERIMENTO QUE VAMOS A
      //HACER EN EL FUTURO
      if (año < 2020) {
         throw new Excepcionfecha("año");
      }
      if ((mes == 2 || mes == 4 || mes == 6 || mes == 9 || mes == 11) && dia == 31) {
         throw new Excepcionfecha("combinacion");
      }
      if (mes == 2 && dia > 28) {
         throw new Excepcionfecha("combinacion");
      }
      this.dia = dia;
      this.mes = mes;
      this.año = año;

   }

   public int getDia() {
```

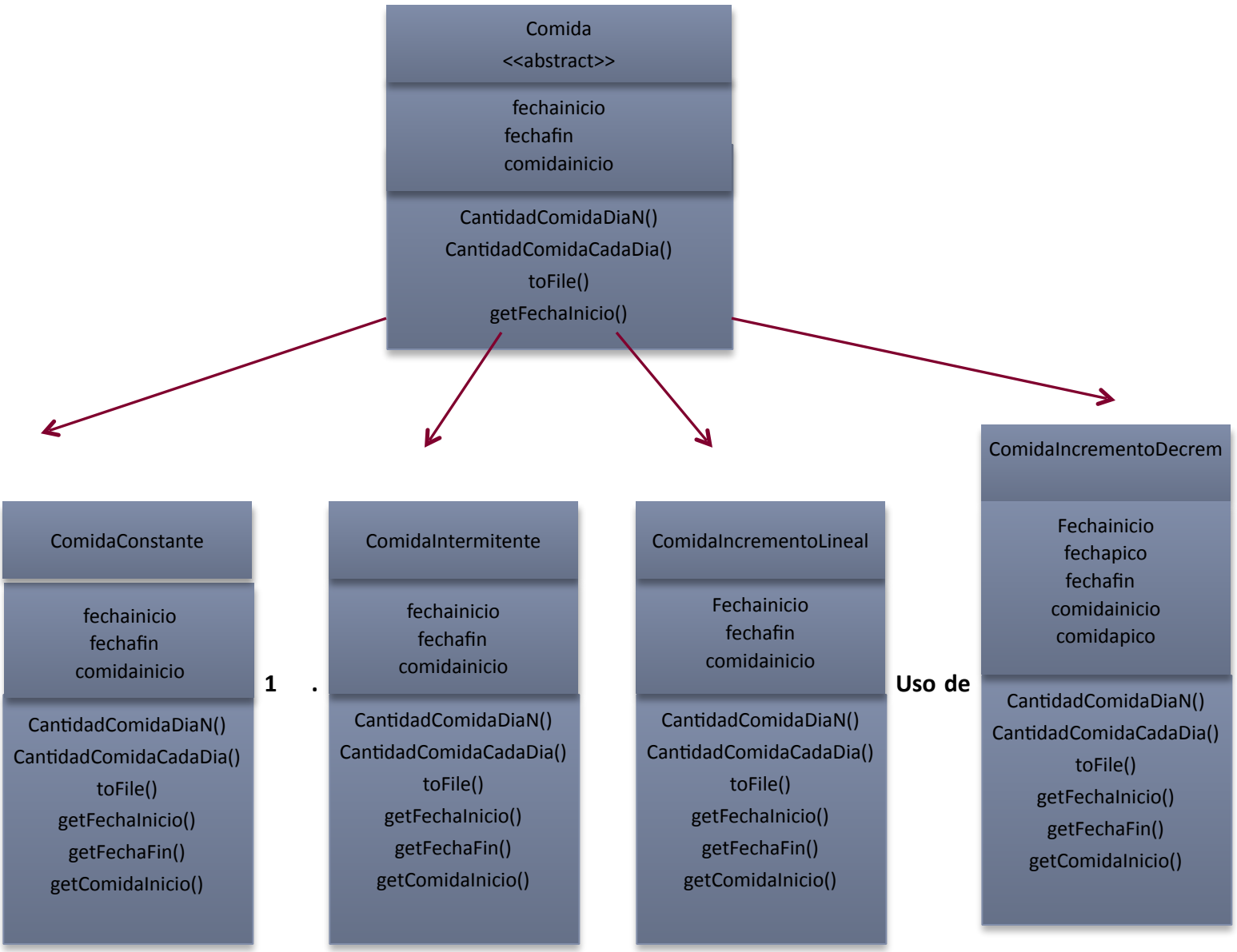
```
    return dia;
  }

  public int getMes() {
     return mes;
  }

  public int getAño() {
     return año;
  }
 public Boolean anteriorA(Fecha mayor) {
     Boolean rango = false;
     if (this.año < mayor.año) {
        rango = true;
        return rango;
     }
     if (this.año > mayor.año) {
        return false;

     }
     if (this.año == mayor.año) {
        if (this.mes > mayor.mes) {
           return false;
        }
        if (this.mes == mayor.mes) {
           if (this.dia > mayor.dia) {
              return false;
           }
           if (this.dia < mayor.dia) {
              rango = true;
              return rango;
           }
           if (this.dia == mayor.dia) {
              return false;
           }

        }
     }
     return rango;
  }
}
```

Clases Comida

Al estar altamente relacionadas, abordaré las clases en conjunto.

CLASE COMIDA

La clase comida es una clase abstracta de la cual heredan las cuatro clases de comida que proporcionan funciones de suministrado de comida durante el periodo de la existencia de una población de bacterias. Existirá un atributo comida en la clase Poblacionbacteria.



estructuras y conceptos de POO utilizados

La diferencia entre crear 4 clases diferentes de comida a crear en una clase comida cuatro funciones diferentes de suministro es que en el caso de las poblaciones de bacterias se desea que se calcule el suministro de comida de una forma única. Esto es significativo a la hora de

1. Calcular la simulación de montecarlo
2. Ver la informacion detallada de una poblacion
3. Pasar el objeto comida a una estructura especifica para guardar en un fichero (metodo toFile).

donde se hará uso de la herencia y la ligadura dinámica llamando al metodo de la subclase como resultado del polimorfismo de asignación. Esto se debe a que cuando creemos una instancia de la clase Poblacionbacteria, haremos un upcasting en el que una de las subclases de comida se utilizará como instancia de la superclase abstracta Comida. La utilización de una clase abstracta comun permite la reutilización de codigo. Las variables fechainicio fechafin y comidainicio son todas variables privadas a las que se accede con getters, encapsulando el codigo.

2. Uso de excepciones

La clase comida hace uso de una única excepcion, la Excepcioncomida que gestiona los valores enteros de comida que se suministrarán a la poblacion de bacterias. Esta excepcion se lanza en los constructores de cada comida y también se utiliza en los metodos privados de la clase InputOutput para asegurarse doblemente de que el valor ingresado esta enre 300000 y 1.

3. Lista de fallos conocidos y posibles mejoras

Una posible mejora que es fácil de ver es que se podría crear un único metodo que devolviese el valor de suministro de comida de una poblacion en un dia, y en caso de que necesitemos todos los dias, llamar a esa función de forma repetida hasta la fecha de finalización de la población. En mi caso, yo he preferido crear dos metodos que se sobreescriben en todas las clases comida hijas que son :

```
@Override
public int CantidadComidaDiaN(int dia) {}
@Override
public int[] CantidadComidaCadaDia() {}
```

En el primer metodo se llama a el segundo metodo que devolverá un array con la comida de suministro de cada día, y el metodo CantidadComidaDiaN cogerá el valor del indice que represente el dia en cuestión. Aunque hago uso de el metodo CantidadComidaDiaN para la funcion de simulacion de Montecarlo, tambien utilizo CantidadComidaCadaDia en la opcion del caso 6 del switch principal(que imprime toda la información de una población de bacterias). Sin embargo, existe la posibilidad de simplificar el codigo realizando una unica función. Otro posible fallo ya mencionado en la clase fecha que tiene una repercusión en la clases comida es la utilización de la clase fecha y no una clase ya creada con múltiples metodos creados como LocalDate para simplificar las

interacciones entre las clases del proyecto. En el caso de la funcion CantidadComidaCadaDia() se hace uso de una función de la clase LocalDate para contar el numero de dias entre dos fechas, y por lo tanto debo crear instancias de LocalDate anteriormente, lo cual implica un gasto innecesario de memoria.

4. Código

CLASE Comida

```
public abstract class Comida {

public Fecha fechainicio;
public Fecha fechafin;
public int comidainicio;

public int CantidadComidaDiaN(int dia) {
int cantidadcomida = 0;
return cantidadcomida;
}

public int[] CantidadComidaCadaDia() {
int[] comidas = null;
return comidas;
}

public String toFile() {
String a = "";
return a;
}

public Fecha getFechaInicio() {
return fechainicio;
}

public Fecha getFechaFin() {
return fechafin;
}

public int getComidaInicio() {
return comidainicio;
}
}
```

CLASE ComidaConstante

```
public class ComidaConstante extends Comida {
public ComidaConstante(int cantidadcomidaprimerdia, Fecha fechainicio, Fecha fechafin) throws Excepcioncomida {
    if (cantidadcomidaprimerdia > 300000 || cantidadcomidaprimerdia < 1) {
       throw new Excepcioncomida("rango");
    }
    this.fechainicio = fechainicio;
```

```
    this.fechafin = fechafin;
    this.comidainicio = cantidadcomidaprimerdia;
  }

  @Override
  public int getComidaInicio() {
    return this.comidainicio;
  }

  @Override
  public Fecha getFechaInicio() {
    return this.fechainicio;
  }

  @Override
  public Fecha getFechaFin() {
    return this.fechafin;
  }

  @Override
  public int CantidadComidaDiaN(int dia) {
    return comidainicio;
  }

  @Override
  public int[] CantidadComidaCadaDia() {
    int[] comida = new int[1];
    comida[0] = comidainicio;
    return comida;
  }
@Override
  public String toFile() {
    return "ComidaConstante" + "," + fechainicio.toFile() + "," + fechafin.toFile() + "," + comidainicio;
  }

  @Override
  public String toString() {
    return "Fecha inicial:    " + fechainicio
        + "\nFecha final:    " + fechafin
        + "\nComida inicial:    " + comidainicio;
  }

}
```

CLASE ComidaIntermitente

```
public class ComidaIntermitente extends Comida {
public ComidaIntermitente(Fecha fechainicio, Fecha fechafin, int cantidadcomidaprimerdia) throws Excepcioncomida {
    if (cantidadcomidaprimerdia > 300000 || cantidadcomidaprimerdia < 1) {
      throw new Excepcioncomida("rango");
    }
    this.fechainicio = fechainicio;
    this.fechafin = fechafin;
    this.comidainicio = cantidadcomidaprimerdia;
```

```
}

public int getComidainicial() {
    return this.comidainicio;
}

@Override
public Fecha getFechaInicio() {
    return this.fechainicio;
}

@Override
public Fecha getFechaFin() {
    return this.fechafin;
}

@Override
public int CantidadComidaDiaN(int dia) {
    int[] arraycomidas = CantidadComidaCadaDia();
    return arraycomidas[dia];
}

@Override
public int[] CantidadComidaCadaDia() {
    LocalDate fechainicial = LocalDate.of(fechainicio.getAño(), fechainicio.getMes(), fechainicio.getDia());
    LocalDate fechafinal = LocalDate.of(fechafin.getAño(), fechafin.getMes(), fechafin.getDia());
    int numerodedias = (int) ChronoUnit.DAYS.between(fechainicial, fechafinal);
    System.out.println("El numero de dias es" + numerodedias);
    int[] comidas = new int[numerodedias + 1];
    comidas[0] = comidainicio;
    for (int i = 1; i < comidas.length; i++) {
        if (i % 2 == 0) {
            comidas[i] = comidainicio;
        } else {
            comidas[i] = 0;
        }
    }
    return comidas;
}
@Override
public String toFile() {
    return "ComidaIntermitente" + "," + fechainicio.toFile() + "," + fechafin.toFile() + ","
            + comidainicio;
}


@Override
public String toString() {
    return "Fecha inicial:    " + fechainicio
            + "\nFecha final:    " + fechafin
            + "\nComida inicial:    " + comidainicio;
}
```

CLASE ComidaIncrementoDecremento

```
public class ComidaIncrementoDecremento extends Comida {

    private int comidapico; //comida en el ultimo dia de incremento de comida
    private int comidafin;
    private Fecha fechapico; //fecha del ultimo dia de incremento de comida

    public ComidaIncrementoDecremento(Fecha fechainicio, Fecha fechapico, Fecha fechafin, int comidainicio, int comidapico, int comidafin) throws Excepcioncomida {
        if (comidainicio < 0 || comidainicio > 300000) {
            throw new Excepcioncomida("rango");
        }
        this.fechainicio = fechainicio;
        this.fechapico = fechapico;
        this.fechafin = fechafin;
        this.comidainicio = comidainicio;
        this.comidapico = comidapico;
        this.comidafin = comidafin;
    }

    @Override
    public int getComidaInicio() {
        return this.comidainicio;
    }

    public int getComidaFin() {
        return this.comidafin;
    }

    public int getComidaPico() {
        return this.comidapico;
    }

    @Override
    public Fecha getFechaInicio() {
        return this.fechainicio;
    }

    @Override
    public Fecha getFechaFin() {
        return this.fechafin;
    }

    public Fecha getFechaPico() {
        return this.fechapico;
    }

    @Override
    public int CantidadComidaDiaN(int dia) {
        int[] arraycomidas = CantidadComidaCadaDia();
        return arraycomidas[dia];
```

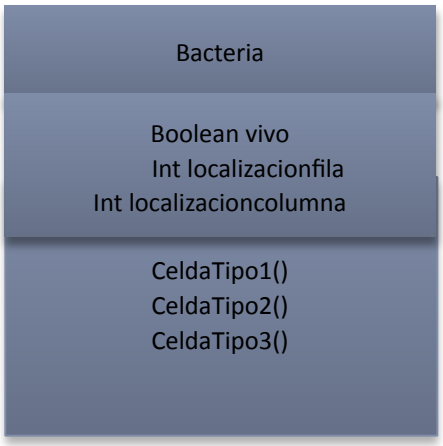
```
}

  @Override
  public int[] CantidadComidaCadaDia() {
    LocalDate fechainicial = LocalDate.of(fechainicio.getAño(), fechainicio.getMes(), fechainicio.getDia());
    LocalDate fechadepico = LocalDate.of(fechapico.getAño(), fechapico.getMes(), fechapico.getDia());
    LocalDate fechafinal = LocalDate.of(fechafin.getAño(), fechafin.getMes(), fechafin.getDia());
    int numerodias = (int) ChronoUnit.DAYS.between(fechainicial, fechafinal);
    int[] comidastotal = new int[numerodias + 1];
    int numerodiasincremento = (int) ChronoUnit.DAYS.between(fechainicial, fechadepico);
    float dosisincremento = (float) (comidapico - comidainicio) / numerodiasincremento;
    for (int i = 0; i <= numerodiasincremento; i++) {
      int valorcomida = (int) (this.comidainicio + (i * dosisincremento));
      comidastotal[i] = valorcomida;
    }

    int numerodiasdecremento = (int) ChronoUnit.DAYS.between(fechadepico, fechafinal);
    float dosisdecremento = (float) (comidafin - comidapico) / (numerodiasdecremento);
    //creamos otro bucle for para rellenar los dias desde el dia siguiente
    //al ultimo dia de incremento de la comida hasta el dia final
    for (int i = numerodiasincremento + 1, j = 1; i < comidastotal.length; i++, j++) {
      comidastotal[i] = (int) (this.comidapico + (j * dosisdecremento));
    }

    return comidastotal;
  }

@Override
  public String toFile() {
    return "ComidaIncrementoDecremento" + "," + fechainicio.toFile() + "," + fechapico.toFile() + "," + fechafin.toFile() + ","
        + comidainicio + "," + comidapico + "," + comidafin;
  }


  @Override
  public String toString() {
    return "Fecha inicial:    " + fechainicio
        + " \nFecha pico:    " + fechapico
        + "\nFecha final:    " + fechafin
        + "\nComida inicial:    " + comidainicio
        + "\nComida pico:    " + comidapico
        + "\nComida fin:    " + comidafin;
  }

}
```

CLASE ComidaIncrementoLineal

```
public class ComidaIncrementoLineal extends Comida {

    private int comidafin;
public ComidaIncrementoLineal(Fecha fechainicio, Fecha fechafin, int cantidadcomidaprimerdia, int cantidadcomidaultimodia) throws Excepcioncomida {

        if (cantidadcomidaprimerdia > 300000 || cantidadcomidaprimerdia < 1 || cantidadcomidaultimodia > 300000 || cantidadcomidaultimodia < 1) {
            throw new Excepcioncomida("rango");
        }
        this.fechainicio = fechainicio;
        this.fechafin = fechafin;
        this.comidainicio = cantidadcomidaprimerdia;
        this.comidafin = cantidadcomidaultimodia;
    }

    public int getComidainicial() {
        return this.comidainicio;
    }

    public int getComidaFin() {
        return this.comidafin;
    }

    @Override
    public Fecha getFechaInicio() {
        return this.fechainicio;
    }

    @Override
    public Fecha getFechaFin() {
        return this.fechafin;
    }

    @Override
    public int CantidadComidaDiaN(int dia) {
        int[] arraycomidas = CantidadComidaCadaDia();
        return arraycomidas[dia];
    }

    @Override
    public int[] CantidadComidaCadaDia() {
        LocalDate fechainicial = LocalDate.of(fechainicio.getAño(), fechainicio.getMes(), fechainicio.getDia());
        LocalDate fechafinal = LocalDate.of(fechafin.getAño(), fechafin.getMes(), fechafin.getDia());
        int numerodedias = (int) ChronoUnit.DAYS.between(fechainicial, fechafinal);
        int[] comidas = new int[numerodedias + 1];
        float dosisincremento = (float) (comidafin - comidainicio) / numerodedias;
        //con la dosis de incremento, hacemos un bucle que nos calcule la comida
        //para cada dia hasta el ultimo dia de incremento de comida.
        //esto lo guardaremos en el array de tipo int que despues
        //devolveremos.
        for (int i = 0; i <= numerodedias; i++) {
            int valorcomida = (int) (this.comidainicio + (i * dosisincremento));
            comidas[i] = valorcomida;
        }
```

```
        return comidas;
    }
@Override
    public String toFile() {
        return "ComidaIncrementoLineal" + "," + fechainicio.toFile() + "," + fechafin.toFile() + ","
                + comidainicio + "," + comidafin;
    }
@Override
    public String toString() {
        return "Fecha inicial:    " + fechainicio
                + "\nFecha final:    " + fechafin
                + "\nComida inicial:    " + comidainicio
                + "\nComida fin:    " + comidafin;
    }
}
```

CLASE Bacteria



1. **Uso de estructuras y conceptos de POO utilizados**

La clase bacteria contiene tres atributos dos de ellos, la localizacionfila y la localizacioncolumna sirven en conjunto, para representar la celda en la que se encuentra la bacteria en cualquier momento del experimento. Asimismo contiene un boolean que representa el estado de la bacteria, si esta viva o no. Estos tres valores son utiles en la clase Poblacionbacteria para realizar la simulación de Montecarlo. Sabiendo si la bacteria esta viva o no, sabemos si podemos removerla del conjunto LinkedList de tipo bacteria. Asimismo, sabiendo la celda en la que se encuentra podemos saber a donde se tiene que mover o la cantidad de comida que debe comer.

[i-1][j-1]	[i-1][j]	[i-1][j+1]
[i][j-1]	[i][j]	[i][j+1]
[i+1][j-1]	[i+1][j]	[i+1][j+1]

Uso de excepciones

En la clase Bacteria se lanza un objeto (excepcion) de tipo CeldaNoExisteException en los tres metodos auxiliares que tienen para poder realizar la simulación en la clase bacteria. En el caso de que una bacteria deba ser movida a una celda contigua que este fuera de plato de cultivo se lanzará esta excepción. Tanto las funciones CeldaTipo1, como CeldaTipo2 y CeldaTipo3 se basan en el siguiente estructura, donde i representa el numero de la fila y j la columna, donde la i y la j son la localizacionfila y la localizacion columna de la bacteria en cuestion. En el caso de que i-1 o j-1 o ambos sea menor a 0 o j+1 o i+1 o ambos sea mayor a 20, entonces saltará la excepción.

3. Código

```
public class Bacteria {

   private boolean vivo;
   private int localizacionfila;
   private int localizacioncolumna;

public Bacteria(int i, int j) {
      this.localizacionfila = i;
      this.localizacioncolumna = j;
      this.vivo = true;
   }

   public int getFilai() {
      return localizacionfila;
   }

   public boolean getVivo() {
      return vivo;
   }

   public void setVivo(boolean estado) {
      vivo = estado;
   }

   public int getColumnaj() {
      return localizacioncolumna;
   }

   public void setFilai(int i) {
      localizacionfila = i;

   }
```

```
  public void setColumnaj(int j) {
    localizacioncolumna = j;
  }

  public void setCantidadComidaDiaria(int comida) {

  }

public void CeldaTipo1(int i, int j) throws NoExisteCeldaException {
    Random randomnumber = new Random();
    int valorandom = randomnumber.nextInt(100);
    if (valorandom < 3) {
      vivo = false;
    }
    if (valorandom >= 60 && valorandom < 100) {
      if (valorandom >= 60 && valorandom < 65) {
        if (i - 1 < 0 || j - 1 < 0) {
          throw new NoExisteCeldaException();
        } else {
          localizacionfila = i - 1;
          localizacioncolumna = j - 1;
        }
      }
      if (valorandom >= 65 && valorandom < 70) {
        if (i - 1 < 0) {
          throw new NoExisteCeldaException();
        } else {
          localizacionfila = i - 1;
        }
      }
      if (valorandom >= 70 && valorandom < 75) {
        if (i - 1 < 0 || j + 1 > 20) {
          throw new NoExisteCeldaException();
        } else {
          localizacionfila = i - 1;
          localizacioncolumna = j - 1;
        }
      }
      if (valorandom >= 75 & valorandom < 80) {
        if (j - 1 < 0) {
          throw new NoExisteCeldaException();
        } else {
          localizacioncolumna = j - 1;
        }
      }
      if (valorandom >= 85 && valorandom < 90) {
        if (i + 1 > 20 || j - 1 < 0) {
          throw new NoExisteCeldaException();
        } else {
          localizacionfila = i + 1;
          localizacioncolumna = j - 1;
        }
      }
      if (valorandom >= 80 && valorandom < 85) {
        if (j + 1 > 20) {
          throw new NoExisteCeldaException();
        } else {
          localizacioncolumna = j + 1;
        }
      }
      if (valorandom >= 95 && valorandom < 100) {
```

```
            if (i + 1 > 20 || j + 1 > 20) {
               throw new NoExisteCeldaException();
            } else {
               localizacionfila = i + 1;
               localizacioncolumna = j + 1;
            }
         }
         if (valorandom >= 90 && valorandom < 95) {
            if (i + 1 > 20) {
               throw new NoExisteCeldaException();
            } else {
               localizacionfila = i + 1;
            }
         }

      }
   }

public void CeldaTipo2(int i, int j) throws NoExisteCeldaException {
      Random randomnumber = new Random();
      int valorandom = randomnumber.nextInt(100);
      if (valorandom < 6) {
         vivo = false;
      }
      if (valorandom >= 20 && valorandom < 30) {
         if (valorandom >= 60 && valorandom < 65) {
            if (i - 1 < 0 || j - 1 < 0) {
               throw new NoExisteCeldaException();
            } else {
               localizacionfila = i - 1;
               localizacioncolumna = j - 1;
            }
         }
         if (valorandom >= 30 && valorandom < 40) {
            if (i - 1 < 0) {
               throw new NoExisteCeldaException();
            } else {
               localizacionfila = i - 1;
            }
         }
         if (valorandom >= 40 && valorandom < 50) {
            if (i - 1 < 0 || j + 1 > 20) {
               throw new NoExisteCeldaException();
            } else {
               localizacionfila = i - 1;
               localizacioncolumna = j - 1;
            }
         }
         if (valorandom >= 50 & valorandom < 60) {
            if (j - 1 < 0) {
               throw new NoExisteCeldaException();
            } else {
               localizacioncolumna = j - 1;
            }
         }
         if (valorandom >= 70 && valorandom < 80) {
            if (i + 1 > 20 || j - 1 < 0) {
               throw new NoExisteCeldaException();
            } else {
               localizacionfila = i + 1;
               localizacioncolumna = j - 1;
```

```
            }
        }
        if (valorandom >= 60 && valorandom < 70) {
            if (j + 1 > 20) {
                throw new NoExisteCeldaException();
            } else {
                localizacioncolumna = j + 1;
            }
        }
        if (valorandom >= 90 && valorandom < 100) {
            if (i + 1 > 20 || j + 1 > 20) {
                throw new NoExisteCeldaException();
            } else {
                localizacionfila = i + 1;
                localizacioncolumna = j + 1;
            }
        }
        if (valorandom >= 80 && valorandom < 90) {
            if (i + 1 > 20) {
                throw new NoExisteCeldaException();
            } else {
                localizacionfila = i + 1;
            }

        }
    }
}

public void CeldaTipo3(int i, int j) throws NoExisteCeldaException {
    Random randomnumber = new Random();
    int valorandom = randomnumber.nextInt(100);
    if (valorandom < 20) {
        vivo = false;
    }
    if (valorandom >= 60 && valorandom < 100) {
        if (valorandom >= 60 && valorandom < 65) {
            if (i - 1 < 0 || j - 1 < 0) {
                throw new NoExisteCeldaException();
            } else {
                localizacionfila = i - 1;
                localizacioncolumna = j - 1;
            }
        }
        if (valorandom >= 65 && valorandom < 70) {
            if (i - 1 < 0) {
                throw new NoExisteCeldaException();
            } else {
                localizacionfila = i - 1;
            }
        }
        if (valorandom >= 70 && valorandom < 75) {
            if (i - 1 < 0 || j + 1 > 20) {
                throw new NoExisteCeldaException();
            } else {
                localizacionfila = i - 1;
                localizacioncolumna = j - 1;
            }
        }
        if (valorandom >= 75 & valorandom < 80) {
            if (j - 1 < 0) {
                throw new NoExisteCeldaException();
```

```
            } else {
                localizacioncolumna = j - 1;
            }
        }
        if (valorandom >= 85 && valorandom < 90) {
            if (i + 1 > 20 || j - 1 < 0) {
                throw new NoExisteCeldaException();
            } else {
                localizacionfila = i + 1;
                localizacioncolumna = j - 1;
            }
        }
        if (valorandom >= 80 && valorandom < 85) {
            if (j + 1 > 20) {
                throw new NoExisteCeldaException();
            } else {
                localizacioncolumna = j + 1;
            }
        }
        if (valorandom >= 95 && valorandom < 100) {
            if (i + 1 > 20 || j + 1 > 20) {
                throw new NoExisteCeldaException();
            } else {
                localizacionfila = i + 1;
                localizacioncolumna = j + 1;
            }
        }
        if (valorandom >= 90 && valorandom < 95) {
            if (i + 1 > 20) {
                throw new NoExisteCeldaException();
            } else {
                localizacionfila = i + 1;
            }
        }

    }
  }

}
```

Paquete Excepciones

El paquete de excepciones _contiene todas las clases creadas que heredan de la clase Exception y que controlan las excepciones de la practica. Estas son :

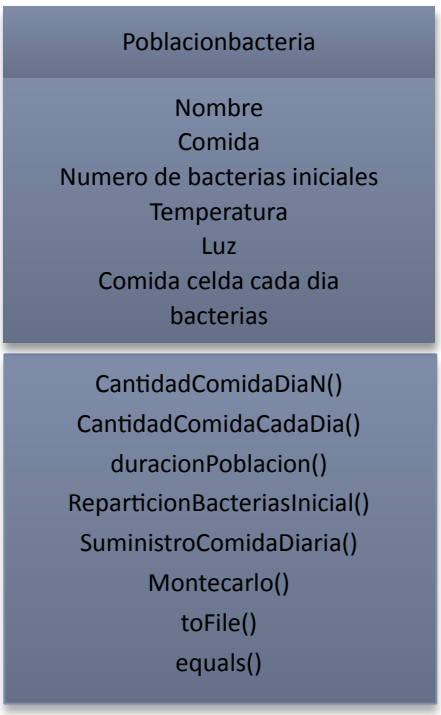
1. ExcepcionEstructuraIncorrecta
2. Excepcioncomida
3. Excepcionfecha
4. MismaPoblacionException : en el caso de que se ingrese una poblacion a un experimento que ya se haya ingresado previamente
5. NoExisteCeldaException
6. NoExistePoblacion : si la poblacion que se esta buscando entre las poblaciones de un experimento no existe.
7. NumeroInicialBacteriasNovalidoException : Si se ingresa un numero negativo de bacterias

En multiples de las clases en su constructor, se les pasa un String que se introduce en el super (llamada a la clase padre). Una posible mejora del paquete de excepciones sería reducir las excepciones por clase, y que las excepciones que se lanzen en una misma clase formen una enumeracion dentro de una clase, reduciendo la cantidad de codigo.

Paquete practica

En este paquete se guardan las clases mas significativas de la practica, la poblacionBacteria el experimento que sirve para guardar las poblaciones de bacterias y dos clases de funciones que manipulan poblaciones de bacterias.

CLASE Poblacionbacteria



1. Uso de estructuras y conceptos de POO utilizados

La clase Poblacionbacteria representa una población. Utilizo un linkedlist de la clase bacteria al incrementar y decrementar constantemente, por lo que es mas eficiente utilizar este tipo de estructura para guardar bacterias de forma dinamica. Todos los atributos son privados, que consiguen mantener el concepto de encapsulación. He decidido sobreescribir los metodos hashCode y equals, aunque solo tengo en consideracion que dos poblaciones son iguales por el nombre, lo cual ya hacía previamente, porque me resulta mas util si ya realizo funciones de comparacion de poblaciones directamente, asi en el caso de que quiera que dos clases sean iguales si tienen mas atributos parecidos, solo hay que cambiar los metodos hashCode y equals.

2. Uso de excepciones

La clase bacteria hace uso de la excepcion NumeroInicialBacteriasNoValidoException, que lanza el mismo constructor en el caso de que el numero de bacterias ingresado sea menor a 0.

3. Lista de fallos conocidos y posibles mejoras

Uno de los fallos principales de la clase es que para realizar la simulación, he creado algunos atributos en la clase cuando en realidad basta con crearlos en la función Montecarlo en si. En caso de tener mas tiempo, haría que la función realizase todos los calculos sin tener atributos en la clase que solo sirvan como soporte para devolver la cantidad de comida en cada celda cada día. Otra posible mejora sería reducir el numero de funciones (duracionPoblacion(),ReparticionBacteriasInicial(),SuministroComidaDiaria que dan soporte a la función Montecarlo para así solamente tener una función que haga el simulacro y abstraer más el código(aunque las demas funciones sean privadas y no se pueden acceder desde otra clase). Tambien se podría hacer la función estática, pasandole una población de bacterias por parametro y devolviendo, mediante el array tridimensional, la cantidad de bacterias por celda por dia. De esta forma, no tendríamos que acceder a la funcion por medio del experimento y despues de la población.

4. Código

```
public class Poblacionbacteria {

    private String nombre = "";
    private final Comida comidapoblacion;
    private final int numerobacteriasinicial;
    private final float temperatura;
    private luminosidad luz;
    private int[][] comidacelda = new int[20][20];
    private LinkedList<Bacteria> grupobacterias = new LinkedList<Bacteria>();


    public enum luminosidad {
        Alta("La luminosidad es alta"),
        Media("la luminosidad es media"),
        Baja("La luminosidad es baja");
        private String luz;

        luminosidad(String luz) {
            this.luz = luz;
```

```
    }

    public String getLuminosidad() {
      return this.luz;
    }
  };

public Poblacionbacteria(String nombre, Comida comidapoblacion, int numerobacteriasinicial, float temperatura,
      luminosidad luz) throws NumeroInicialBacteriasNoValidoException {
    if (numerobacteriasinicial < 0) {
      throw new NumeroInicialBacteriasNoValidoException();
    }
    this.nombre = nombre;
    this.comidapoblacion = comidapoblacion;
    this.numerobacteriasinicial = numerobacteriasinicial;
    this.temperatura = temperatura;
    this.luz = luz;
  }


  private int duracionPoblacion() {
    LocalDate fechainicial = LocalDate.of(comidapoblacion.fechainicio.getAño(), comidapoblacion.fechainicio.getMes(), comidapoblacion.fechainicio.getDia());
    LocalDate fechafinal = LocalDate.of(comidapoblacion.fechafin.getAño(), comidapoblacion.fechafin.getMes(), comidapoblacion.fechafin.getDia());
    int numerodedias = (int) ChronoUnit.DAYS.between(fechainicial, fechafinal);
    return numerodedias;
  }

private void ReparticionBacteriasInicial() {
    int numeroporcelda = numerobacteriasinicial / 16;
    for (int fila = 8; fila <= 11; fila++) {
      for (int columna = 8; columna <= 11; columna++) {
        for (int numerobacteriasencelda = 0; numerobacteriasencelda < numeroporcelda; numerobacteriasencelda++) {
          Bacteria bacteriameter = new Bacteria(fila, columna);
          grupobacterias.add(bacteriameter);
        }
      }
    }
  }

private void SuministroComidaDiaria(int dia) {
    int comidasuministrar = comidapoblacion.CantidadComidaDiaN(dia);
    int comidaporcelda = comidasuministrar / 400;
    for (int fila = 0; fila < 20; fila++) {
      for (int columna = 0; columna < 20; columna++) {
        comidacelda[fila][columna] += comidaporcelda;
      }
    }
  }

public int[][][] Montecarlo() {
int numerobacteriasporceldapordia[][] = new int[20][20];
    int duracionpoblacion = duracionPoblacion();
    int[][][] bacteriaspordiaarray = new int[duracionpoblacion][20][20];
    ReparticionBacteriasInicial();
   int[] bacteriaspordia = new int[duracionpoblacion];
    //int[][][] bacteriasycomidarestantecadadia = new int[duracionpoblacion][20][20];
    //reparto las bacterias en las dieciseis celdas centrales
    ReparticionBacteriasInicial();
    for (int dia = 0; dia < duracionpoblacion; dia++) {
      //reparto la comida del dia
      SuministroComidaDiaria(dia);
```

```
int cantidadbacteriasprincipiodia = grupobacterias.size();
//metemos el numero de bacterias al iniciar el dia, porque despues se suman las bacterias
//hijas que se crean al grupobacterias.size() pero nosotros solamente queremos calcular para
//las bacterias con las que empezamos el dia.
//las bacterias hijas seran manipuladas al dia siguiente
for (int bacteria = 0; bacteria < cantidadbacteriasprincipiodia; bacteria++) {
   int repeticion = 0;
   int comebacteria = 0;
   boolean celdaencontrada;
   do {
      celdaencontrada = false;
      for (int fila = 0; fila < 20 && celdaencontrada == false; fila++) {
         for (int columna = 0; columna < 20 && celdaencontrada == false; columna++) {
            if (grupobacterias.get(bacteria).getFilai() == fila && grupobacterias.get(bacteria).getColumnaj() == columna) {
               celdaencontrada = true;
               int opcion = 0;
               if (comidacelda[fila][columna] >= 100) {
                  opcion = 1;
               }
               if (comidacelda[fila][columna] > 9 && comidacelda[fila][columna] < 100) {
                  opcion = 2;
               }
               if (comidacelda[fila][columna] < 9) {
                  opcion = 3;
               }
               switch (opcion) {
                  case 1: {
                     comebacteria += 20;
                     comidacelda[fila][columna] -= 20;
                     try {
                        grupobacterias.get(bacteria).CeldaTipo1(fila, columna);
                     } catch (NoExisteCeldaException ex) {
                        //quiere decir que no se puede mover a la celda indicada porque esta fuera del 20x20
                        //por lo que se queda en su celda.
                        grupobacterias.get(bacteria).setFilai(fila);
                        grupobacterias.get(bacteria).setColumnaj(columna);
                     }
                     break;
                  }
                  case 2: {
                     comebacteria += 10;
                     comidacelda[fila][columna] -= 10;
                     try {
                        grupobacterias.get(bacteria).CeldaTipo2(fila, columna);
                     } catch (NoExisteCeldaException ex) {
                        grupobacterias.get(bacteria).setFilai(fila);
                        grupobacterias.get(bacteria).setColumnaj(columna);
                     }
                     break;
                  }
                  case 3: {
                     try {
                        grupobacterias.get(bacteria).CeldaTipo3(fila, columna);
                     } catch (NoExisteCeldaException ex) {
                        grupobacterias.get(bacteria).setFilai(fila);
                        grupobacterias.get(bacteria).setColumnaj(columna);
                     }
                     break;
                  }
               } //switch
            } //if
```

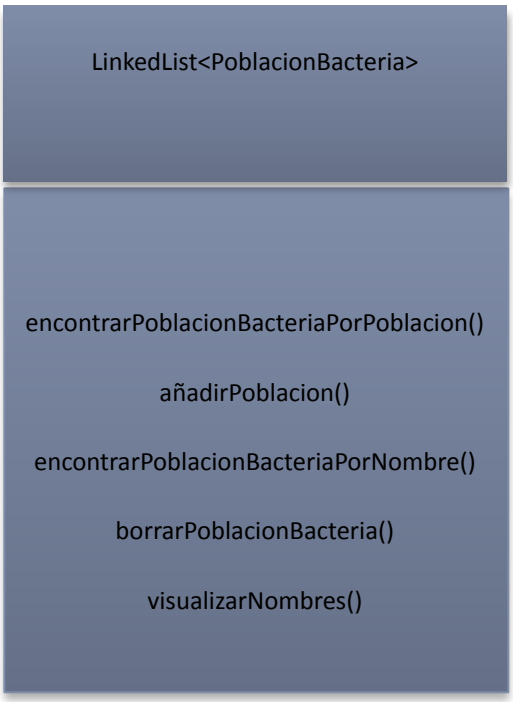
```
            } //forcolumnas
          } //for filas
          repeticion++;
        } while (grupobacterias.get(bacteria).getVivo() && repeticion < 10);
        //con la cantidad de comida de bacteriacomida si la bacteria sigue vivo
        if (grupobacterias.get(bacteria).getVivo() == true) {
          if (comebacteria > 150) {
            grupobacterias.add(new Bacteria(grupobacterias.get(bacteria).getFilai(), grupobacterias.get(bacteria).getColumnaj()));
            grupobacterias.add(new Bacteria(grupobacterias.get(bacteria).getFilai(), grupobacterias.get(bacteria).getColumnaj()));
            grupobacterias.add(new Bacteria(grupobacterias.get(bacteria).getFilai(), grupobacterias.get(bacteria).getColumnaj()));
          }
          if (comebacteria > 100 && comebacteria < 150) {
            grupobacterias.add(new Bacteria(grupobacterias.get(bacteria).getFilai(), grupobacterias.get(bacteria).getColumnaj()));
            grupobacterias.add(new Bacteria(grupobacterias.get(bacteria).getFilai(), grupobacterias.get(bacteria).getColumnaj()));
          }
          if (comebacteria > 50 && comebacteria < 100) {
            grupobacterias.add(new Bacteria(grupobacterias.get(bacteria).getFilai(), grupobacterias.get(bacteria).getColumnaj()));
          }
        }
      }
      //eliminamos las bacterias muertas
      for (int bacteria = 0; bacteria < grupobacterias.size(); bacteria++) {
        if (grupobacterias.get(bacteria).getVivo() == false) {
          grupobacterias.remove(bacteria);
        }
      }

          int cantidadbacteriasencelda;
     for(int fila=0;fila<20;fila++){
      for(int columna=0;columna<20;columna++){
        cantidadbacteriasencelda =0;
        for(int i=0;i<grupobacterias.size();i++){
        if((grupobacterias.get(i).getFilai()==fila) && (grupobacterias.get(dia).getColumnaj()==columna)){
          cantidadbacteriasencelda++;
        }
         }

        numerobacteriasporceldapordia[fila][columna]=cantidadbacteriasencelda;
      }
      }
       for(int fila=0;fila<20;fila++){
        for(int columna=0;columna<20;columna++){
      bacteriaspordiaarray[dia][fila][columna]= numerobacteriasporceldapordia[fila][columna];
          }
      }


      //contamos la cantidad de bacterias en el dia
      bacteriaspordia[dia] = grupobacterias.size();

    }
    //return bacteriaspordia;
    return bacteriaspordiaarray;
}

public Comida getComida() {
    return comidapoblacion;
}

public String getNombre() {
    return this.nombre;
```

```
    }

    public int getNumeroBacteriasInicial() {
        return this.numerobacteriasinicial;
    }

    public String toFile() {
        return nombre + "," + numerobacteriasinicial + "," + temperatura + "," + luz + "," + comidapoblacion.toFile();
    }

@Override
    public String toString() {
        return "NOMBRE:    " + nombre + "\n\nNUMERO DE BACTERIAS:    " + numerobacteriasinicial + ""
                + "\n\nTEMPERATURA(EN GRADOS)    :" + temperatura + "\n\nLUMINOSIDAD:    " + luz.getLuminosidad() + "\n\n  COMIDA: \n\n" + comidapoblacion;
    }

    @Override
    public int hashCode() {
        int hash = 5;
        hash = 59 * hash + Objects.hashCode(this.nombre);
        return hash;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (obj == null) {
            return false;
        }
        if (getClass() != obj.getClass()) {
            return false;
        }
        final Poblacionbacteria other = (Poblacionbacteria) obj;
        if (!Objects.equals(this.nombre, other.nombre)) {
            return false;
        }
        return true;
    }


}
```

CLASE Experimento

La clase experimento sirve para almacenar un grupo de poblaciones de bacterias (clase Poblacionbacteria), tambien contiene funciones para poder añadir, borrar y visualizar los nombres de las poblaciones.

Experimento



1. **Uso de estructuras y conceptos de POO utilizados**

En la clase Experimento he decidido utilizar una estructura de datos de tipo LinkedList para almacenar las poblaciones de bacterias porque teniendo en cuenta la variación de tamaño del experimento, me ha parecido mas eficiente una linkedlist que tiene una complejidad computacional de borrado e insercion constante.

2. **Uso de excepciones**

Esta clase lanza dos excepciones, la primera, NoExistePoblacion en el caso de las funciones que requieren buscar una poblacion y no se encuentra dentro de la lista y la excepcion MismaPoblacionException que gestiona el caso en el que se desea añadir una poblacion que ya existe en la lista.

3. **Lista de fallos conocidos y posibles mejoras**

Una clara mejora en el código es utilizar simplemente una función de busqueda, sin embargo, yo he creado dos porque a pesar de que me interesaría tener una única a la que se le pasase un objeto de tipo poblacion y lo compare mediante el metodo equals y el metodo hashCode, resulta más tedioso pedirle al usuario cada variable de la población a borrar. En realidad, ambos metodos realizan lo mismo puesto que al sobreescribir los metodos en la clase Poblacionbacteria, solamente puse la igualdad en base al nombre. Sin embargo, en caso de que se realizase una búsqueda más especifica y que el equals y el hashCode esten basados en todos los aspectos, la funcion busqueda con una Poblacionbacteria como parametro sería la ideal.

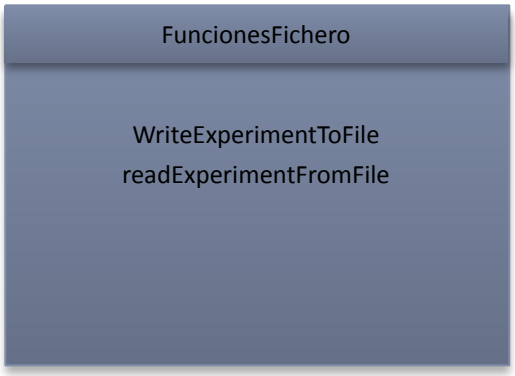
4. Código

```
public class Experimento {

   private LinkedList<Poblacionbacteria> poblacion1 = new LinkedList<Poblacionbacteria>();

public Experimento(Poblacionbacteria primerapoblacion) {
      this.poblacion1.add(primerapoblacion);
   }

   //para recoger datos de fichero, que no queremos meter una poblacioninicial;
   public Experimento() {
   }

   public LinkedList<Poblacionbacteria> getPoblacionesBacteria() {
      return poblacion1;
   }

   public Poblacionbacteria getPoblacionBacteria(String nombrepoblacion) throws NoExistePoblacion {
      return poblacion1.get(encontrarPoblacionBacteriaPorNombre(nombrepoblacion));
   }

private int encontrarPoblacionBacteriaPorPoblacion(Poblacionbacteria poblacion) throws NoExistePoblacion {
      int i = 0;
      boolean poblacionencontrada = false;
      for (i = 0; i < poblacion1.size() && poblacionencontrada == false; i++) {
         if ((poblacion1.get(i)).equals(poblacion) && poblacion1.get(i).hashCode() == poblacion1.get(i).hashCode()) {
            poblacionencontrada = true;
            return i;
         }
      }
      if (poblacionencontrada == false) {
         throw new NoExistePoblacion();
      }
      return i;
   }

public void añadirPoblacion(Poblacionbacteria poblacion) throws MismaPoblacionException {
      try {
         encontrarPoblacionBacteriaPorPoblacion(poblacion);
         throw new MismaPoblacionException();
      } catch (NoExistePoblacion e) {
         poblacion1.add(poblacion);
      }
   }

private int encontrarPoblacionBacteriaPorNombre(String nombrepoblacion) throws NoExistePoblacion {
      int encontrado = 0;
      int i;
      for (i = 0; i < poblacion1.size() && encontrado == 0; i++) {
         if (nombrepoblacion.equals(poblacion1.get(i).getNombre())) {
            encontrado = 1;
            return i;
```

```
        }
      }
      if (i == poblacion1.size()) {
        throw new NoExistePoblacion();
      }
      return i;
    }

public void borrarPoblacionBacteria(String nombrepoblacion) throws NoExistePoblacion {
      int i = encontrarPoblacionBacteriaPorNombre(nombrepoblacion);
      poblacion1.remove(poblacion1.get(i));
    }

public String[] visualizarNombres() {
      String nombrespoblaciones[] = new String[poblacion1.size()];
      for (int i = 0; i < poblacion1.size(); i++) {
        nombrespoblaciones[i] = poblacion1.get(i).getNombre();
      }
      return nombrespoblaciones;
    }
}
```

CLASE FuncionesFichero

La clase FuncionesFichero contiene los metodos para guardar un experimento en un fichero y devolver un experimento de un fichero. Consta de dos funciones estaticas : readExperimentFromFile y WriteExperimentToFile.



1. Uso de estructuras y conceptos de POO utilizados

En esta clase se hace uso de la función to file, que separa cada atributo de cada clase por comas. En el caso de el atributo comida de una población, dependiendo de la clase que sea, llamará a una función u otra. Para meter un experimento de fichero en el experimento actual que estemos manipulando, he introducido un string con el nombre de la clase dentro del metodo toFile para que al llegar al String, sepa cuantos atributos quedan por almacenar y que tipo de comida crear para mas tarde ingresar en una poblacion.

2. Uso de excepciones y lista de fallos conocidos y posibles mejoras

Esta clase puede lanzar la que mas excepciones, puesto que estamos haciendo uso de los constructores de la clase fecha, comida, poblacionbacteria que todos tienen una excepcion particular. Sin embargo, estas nunca se lanzaran porque las poblaciones que se han guardado en el fichero ya se habrían comprobado a la hora de añadirlas al linkedlist del experimento. Estas excepciones son la : Excepcionfecha, Excepcioncomida,NumeroInicialBacteriasNoValidoException. Una forma de hacer que no sea imperativo incluir que esta función no lanze estas excepciones, es hacer que estas clases creadas no hereden de Exception sino de RunTimeException lo que hace que puedan ser unchecked. Esta sería una posible mejora. Una excepcion que solamente se lanza en esta clase es la ExcepcionEstructuraIncorrecta, en caso de que se abra un fichero que no tiene unos atributos en orden para realizar una población se lanzará esta excepcion. Tambien se lanza la IOException. Una forma en la que me ahorro codigo es incluyendo el toFile de la comida como el ultimo atributo, por lo que todos los demas se pueden recoger en conjunto hasta llegar al último tramo.

3. Código

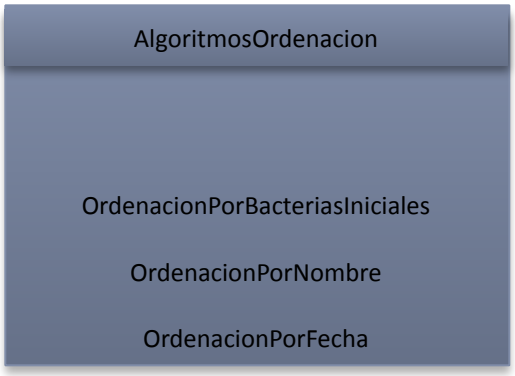
```
public class FuncionesFichero {

public static Experimento readExperimentFromFile(File fichero) throws FileNotFoundException, MismaPoblacionException, IOException,
ExcepcionEstructuraIncorrecta, Excepcionfecha, Excepcioncomida, NumeroInicialBacteriasNoValidoException {
    Experimento experimentodevolver = new Experimento();
    BufferedReader in = new BufferedReader(new FileReader(fichero));
    String line;
    while ((line = in.readLine()) != null) {
      String valores[] = line.split(",");
      String nombrepoblacion = valores[0];
      int numerobacteriasinicial = Integer.parseInt(valores[1]);
      float temperatura = Float.parseFloat(valores[2]);
      String luminosidadvalor = (valores[3]);
      //lo inicicializamos con un valor cualquiera
      Poblacionbacteria.luminosidad luz = luminosidad.Alta;
      if (luminosidadvalor.equals("Alta")) {
        luz = Poblacionbacteria.luminosidad.Alta;
      }
      if (luminosidadvalor.equals("Media")) {
        luz = Poblacionbacteria.luminosidad.Media;
      }
      if (luminosidadvalor.equals("Baja")) {
        luz = Poblacionbacteria.luminosidad.Baja;
      }
      String tipocomida = valores[4];
      Comida comidapoblacion = null;
      Poblacionbacteria poblacionarchivo;

      if (tipocomida.equals("ComidaIncrementoDecremento")) {
        int diainicial = Integer.parseInt(valores[5]);
        int mesinicial = Integer.parseInt(valores[6]);
        int añoinicial = Integer.parseInt(valores[7]);
        Fecha fechainicial = new Fecha(diainicial, mesinicial, añoinicial);
        int ultimodiaincremento = Integer.parseInt(valores[8]);
```

```
        int ultimomesincremento = Integer.parseInt(valores[9]);
        int ultimoañoincremento = Integer.parseInt(valores[10]);
        Fecha fechaincremento = new Fecha(ultimodiaincremento, ultimomesincremento, ultimoañoincremento);
        int diafinal = Integer.parseInt(valores[11]);
        int mesfinal = Integer.parseInt(valores[12]);
        int añofinal = Integer.parseInt(valores[13]);
        Fecha fechafinal = new Fecha(diafinal, mesfinal, añofinal);
        int comidainicial = Integer.parseInt(valores[14]);
        int comidaultimodiaincremento = Integer.parseInt(valores[15]);
        int comidafinal = Integer.parseInt(valores[16]);
        comidapoblacion = new ComidaIncrementoDecremento(fechainicial,fechaincremento ,fechafinal, comidainicial, comidaultimodiaincremento, comidafinal);
        poblacionarchivo = new Poblacionbacteria(nombrepoblacion, comidapoblacion, numerobacteriasinicial, temperatura, luz);
    }
    if (tipocomida.equals("ComidaIncrementoLineal")) {
        int diainicial = Integer.parseInt(valores[5]);
        int mesinicial = Integer.parseInt(valores[6]);
        int añoinicial = Integer.parseInt(valores[7]);
        Fecha fechainicial = new Fecha(diainicial, mesinicial, añoinicial);
        int diafinal = Integer.parseInt(valores[8]);
        int mesfinal = Integer.parseInt(valores[9]);
        int añofinal = Integer.parseInt(valores[10]);
        Fecha fechafinal = new Fecha(diafinal, mesfinal, añofinal);
        int comidainicial = Integer.parseInt(valores[11]);
        int comidafinal = Integer.parseInt(valores[12]);
        comidapoblacion = new ComidaIncrementoLineal(fechainicial, fechafinal, comidainicial, comidafinal);
        poblacionarchivo = new Poblacionbacteria(nombrepoblacion, comidapoblacion, numerobacteriasinicial, temperatura, luz);
    }

    if (tipocomida.equals("ComidaConstante")) {
        int diainicial = Integer.parseInt(valores[5]);
        int mesinicial = Integer.parseInt(valores[6]);
        int añoinicial = Integer.parseInt(valores[7]);
        Fecha fechainicial = new Fecha(diainicial, mesinicial, añoinicial);
        int diafinal = Integer.parseInt(valores[8]);
        int mesfinal = Integer.parseInt(valores[9]);
        int añofinal = Integer.parseInt(valores[10]);
        Fecha fechafinal = new Fecha(diafinal, mesfinal, añofinal);
        int comidainicial = Integer.parseInt(valores[11]);
        comidapoblacion = new ComidaConstante(comidainicial, fechainicial, fechafinal);
        poblacionarchivo = new Poblacionbacteria(nombrepoblacion, comidapoblacion, numerobacteriasinicial, temperatura, luz);

    }
    if (tipocomida.equals("ComidaIntermitente")) {
        int diainicial = Integer.parseInt(valores[5]);
        int mesinicial = Integer.parseInt(valores[6]);
        int añoinicial = Integer.parseInt(valores[7]);
        Fecha fechainicial = new Fecha(diainicial, mesinicial, añoinicial);
        int diafinal = Integer.parseInt(valores[8]);
        int mesfinal = Integer.parseInt(valores[9]);
        int añofinal = Integer.parseInt(valores[10]);
        Fecha fechafinal = new Fecha(diafinal, mesfinal, añofinal);
        int comidainicial = Integer.parseInt(valores[11]);
        comidapoblacion = new ComidaIntermitente(fechainicial, fechafinal, comidainicial);
```

```
            poblacionarchivo = new Poblacionbacteria(nombrepoblacion, comidapoblacion, numerobacteriasinicial, temperatura, luz);
        }
        poblacionarchivo = new Poblacionbacteria(nombrepoblacion, comidapoblacion, numerobacteriasinicial, temperatura, luz);
        experimentodevolver.añadirPoblacion(poblacionarchivo);
      }
      if (in != null) {
        in.close();
      }
      return experimentodevolver;
    }

    public static void WriteExperimentToFile(File fichero, Experimento experimentometer) throws IOException {
      if (!fichero.exists()) {
        fichero.createNewFile();
      }
      PrintWriter out = new PrintWriter(new FileWriter(fichero, false));
      for (int i = 0; i < experimentometer.getPoblacionesBacteria().size(); i++) {
        out.println(experimentometer.getPoblacionesBacteria().get(i).toFile());
      }
      out.close();
    }

}
```

CLASE AlgoritmosOrdenacion

La clase AlgoritmosOrdenacion proporciona metodos para ordenar el LinkedList de poblaciones de bacterias de un experimento en base a distintos requerimientos : fecha, numero de bacterias iniciales, y nombre.



1. **Uso de estructuras y conceptos de POO utilizados**

He decidido utilizar la burbuja mejorada en todos los casos, puesto que es un codigo que he podido utilizar facilmente en los tres casos y me ha ahorrado tiempo. Asimismo, la utilización de la burbuja mejorada tiene una complejidad computacional normalmente hasta dos veces menor al metodo de la burbuja normal. He decidido realizar las funciones de ordenacion en una clase separada para reducir la cantidad de código de cada clase y hacer que el código sea mas visible y fácil de seguir.

2. **Uso de excepciones**

Esta clase no consta de excepciones

3. **Código**

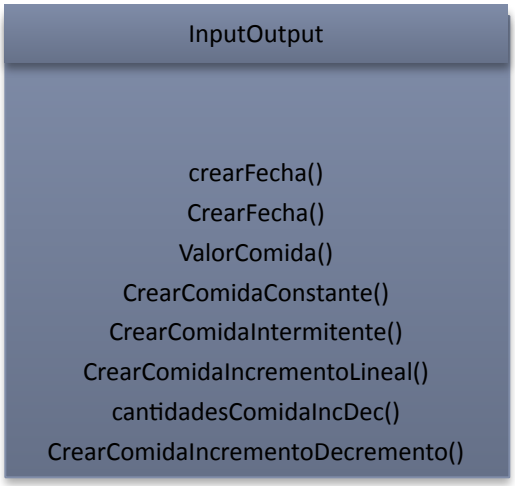
```
public class AlgoritmosOrdenacion {

public static void OrdenacionPorNombre(LinkedList<Poblacionbacteria> poblacion) {
      int tamaño = poblacion.size();
      Poblacionbacteria auxiliar;
      int ordenado;
      for (int posicionactual = 0; posicionactual < tamaño - 1; posicionactual++) {
         ordenado = 0;
         for (int posicioncomparacion = 0; posicioncomparacion < (tamaño - 1 -
posicionactual); posicioncomparacion++) {
            if
((poblacion.get(posicioncomparacion).getNombre()).compareToIgnoreCase(poblacion.
get(posicioncomparacion + 1).getNombre()) > 0) {
               auxiliar = poblacion.get(posicioncomparacion + 1);
               poblacion.set((posicioncomparacion + 1),
poblacion.get(posicioncomparacion));
               poblacion.set((posicioncomparacion), auxiliar);
               ordenado = 1;
            }
         }
         if (ordenado == 0) {
            break;
         }
      }
   }

public static void OrdenacionPorBacteriasIniciales(LinkedList<Poblacionbacteria>
poblacion) {
      int tamaño = poblacion.size();
      Poblacionbacteria auxiliar;
      int ordenado;
      for (int posicionactual = 0; posicionactual < tamaño - 1; posicionactual++) {
         ordenado = 0;
         for (int posicioncomparacion = 0; posicioncomparacion < (tamaño - 1 -
posicionactual); posicioncomparacion++) {
            if (poblacion.get(posicioncomparacion).getNumeroBacteriasInicial() >
(poblacion.get(posicioncomparacion + 1).getNumeroBacteriasInicial())) {
               auxiliar = poblacion.get(posicioncomparacion + 1);
               poblacion.set((posicioncomparacion + 1),
poblacion.get(posicioncomparacion));
               poblacion.set((posicioncomparacion), auxiliar);
               ordenado = 1;
            }
         }
         if (ordenado == 0) {
            break;
         }
      }
   }

public static void OrdenacionPorFecha(LinkedList<Poblacionbacteria> poblacion) {
      int tamaño = poblacion.size();
```

Paquete InputOutput

Este paquete consta de dos clases, que son las únicas que mantienen la relación con el usuario.

CLASE InputOutput

La clase InputOutput es la encargada de realizar instancias de cada clase. Consta de múltiples metodos para crear los distintos objetos. Todas las funciones estaticas son privadas menos la de crearPoblacion que se utilizará despues en la clase main.



1. **Uso de estructuras y conceptos de POO utilizados y posibles mejoras**

Aunque haya intentado reutilizar codigo con la funcion ValorComida(), es cierto que tengo muchas funciones para realizar menos cosas. Por lo tanto una mejora sería reducir la cantidad de funciones para que el programa fuese mas comprensible.

2. **Código**

```
public class InputOutput {

//CREAR FECHA : mediante la funcion crearFecha() y CrearFecha()
```

```
static private Fecha crearFecha() throws Excepcionfecha{
    boolean fechanovalida = false;
    int dia = 0;
    int mes = 0;
    int año = 0;
    do {
        try {
            System.out.println("Introduzca el dia");
            Scanner diascanner = new Scanner(System.in);
            dia = diascanner.nextInt();

            if (dia > 31 || dia < 1) {
                throw new Excepcionfecha("dia");
            } else {
                fechanovalida = true;
            }
        } catch (InputMismatchException excepcion) {
            System.out.println("Tiene que ser un numero entero");
        }
    } while (!fechanovalida);

    fechanovalida = false;
    do {
        try {
            System.out.println("Introduzca el mes");
            Scanner messcanner = new Scanner(System.in);
            mes = messcanner.nextInt();
            if (mes > 12 || mes < 1) {
                throw new Excepcionfecha("mes");
            }
            if ((mes == 2 || mes == 4 || mes == 6 || mes == 9 || mes == 11) && dia == 31) {
                throw new Excepcionfecha("combinacion");
            } else {
                fechanovalida = true;
            }
        } catch (InputMismatchException excepcion) {
            System.out.println("Tiene que ser un numero entero");
        }
    } while (!fechanovalida);

    fechanovalida = false;
    do {
        System.out.println("Introduzca el año");
        Scanner añoscanner = new Scanner(System.in);
        año = añoscanner.nextInt();
        if (año < 1900) {
            throw new Excepcionfecha("año");
        } else {
            fechanovalida = true;
        }
    } while (!fechanovalida);
    Fecha fecha1 = new Fecha(dia, mes, año);
```

```
    return fecha1;
  }

static private Fecha CrearFecha() {
    boolean noexcepcion = true;
    Fecha fecha1 = null;
    do {
      try {
        fecha1 = crearFecha();
        noexcepcion = true;
      } catch (Excepcionfecha excepcion) {
        if (excepcion.getMessage() == "dia") {
          System.out.println("El dia debe ser un entero entre 1 y 31");
          noexcepcion = false;
        }
        if (excepcion.getMessage() == "mes") {
          System.out.println("El mes debe ser un entero entre 1 y 12");
          noexcepcion = false;
        }
        if (excepcion.getMessage() == "año") {
          System.out.println("El año debe ser mayor a 2020");
          noexcepcion = false;
        }
        if (excepcion.getMessage() == "combinacion") {
          System.out.println("Los meses de febrero,abril,junio,septiembre y noviembre no tienen 31 dias");
          noexcepcion = false;
        }
      }
    } while (false == noexcepcion);
    return fecha1;
  }
  //meter un valor valido en el atributo comida

static private int ValorComida() throws Excepcioncomida {
    int comidainicial = 0;
    boolean comidaesentero = true;
    do {
      try {
        System.out.println("Introduzca el valor de la comida");
        Scanner scannercomida = new Scanner(System.in);
        comidainicial = scannercomida.nextInt();
        if (comidainicial >= 300000 || comidainicial <= 0) {
          throw new Excepcioncomida("rango");
        }
      } catch (InputMismatchException ex) {
        System.out.println("Debe ser un entero");
        comidaesentero = false;
      }
    } while (comidaesentero == false);
    return comidainicial;
  }

  //CREAMOS COMIDA CONSTANTE
```

```
static private ComidaConstante CrearComidaConstante() {
      Fecha fechainicio = null;
      Fecha fechafin = null;
      int comida = 0;
      boolean comidanovalida = true;
      boolean fechasnovalidas = true;
      do {
         System.out.println("Introducir la fecha de inicio de la poblacion");
         fechainicio = CrearFecha();
         System.out.println("Introducir la fecha de finalizacion de la poblacion");
         fechafin = CrearFecha();
         if (fechainicio.anteriorA(fechafin) == false) {
            fechasnovalidas = false;
            System.out.println("La fecha de finalizacion debe ser posterior a la de inicio");
         };

      } while (fechasnovalidas = false);
      ComidaConstante comida1 = null;
      do {
         try {
            comida = ValorComida();
         } catch (Excepcioncomida ex) {
            System.out.println("Debe ser un valor entero entre 1 y 300000");
            comidanovalida = true;
         }

      } while (comidanovalida == false);
      try {
         comida1 = new ComidaConstante(comida, fechainicio, fechafin);
      } catch (Excepcioncomida ex) {
         //ya esta mas que comprobado, pero al ponerlo que el constructor
         //lanza la excepcion debo comprobarla otra vez.
         ex.printStackTrace();
      }
      return comida1;
   }

static private ComidaIntermitente CrearComidaIntermitente() {
      Fecha fechainicio = null;
      Fecha fechafin = null;
      int comida = 0;
      boolean comidanovalida = true;
      boolean fechanovalida;
      do {
         fechanovalida = true;
         System.out.println("Introducir la fecha de inicio de la poblacion");
         fechainicio = CrearFecha();
         System.out.println("Introducir la fecha de finalizacion de la poblacion");
         fechafin = CrearFecha();
         if (fechainicio.anteriorA(fechafin) == false) {
            fechanovalida = false;
            System.out.println("La fecha de finalizacion debe ser posterior a la de inicio");
         };
```

```
    } while (fechanovalida == false);
    ComidaIntermitente comida1 = null;
    do {
        try {
            System.out.println("Introduzca el valor de la comida");
            comida = ValorComida();
        } catch (Excepcioncomida ex) {
            System.out.println("Debe ser un valor entero entre 1 y 300000");
            comidanovalida = true;
        }
    } while (comidanovalida == false);
    try {
        comida1 = new ComidaIntermitente(fechainicio, fechafin, comida);
    } catch (Excepcioncomida ex) {
        //ya esta mas que comprobado, pero al ponerlo que el constructor
        //lanza la excepcion debo comprobarla otra vez.
        ex.printStackTrace();
    }
    return comida1;
}

/**
 * La funcion CrearComidaIncrementoLineal crea un objeto de tipo
 * ComidaIncrementoLineal
 *
 * @return el objeto creado de tipo ComidaIncrementoLineal
 */
static private ComidaIncrementoLineal CrearComidaIncrementoLineal() {
    Fecha fechainicio = null;
    Fecha fechafin = null;
    int comidainicial = 0;
    int comidafinal = 0;
    boolean comidanovalida = true;
    boolean fechanovalida = true;
    ComidaIncrementoLineal comida1 = null;
    do {
        fechanovalida = true;
        System.out.println("Introducir la fecha de inicio de la poblacion");
        fechainicio = CrearFecha();
        System.out.println("Introducir la fecha de finalizacion de la poblacion");
        fechafin = CrearFecha();
        if (fechainicio.anteriorA(fechafin) == false) {
            fechanovalida = false;
            System.out.println("La fecha de finalizacion debe ser posterior a la de inicio");
        };
    } while (fechanovalida == false);
    do {
        comidanovalida = true;
        try {
            System.out.println("introduzca la comida del primer dia");
            comidainicial = ValorComida();
        } catch (Excepcioncomida ex) {
            System.out.println("Debe ser un valor entero entre 1 y 300000");
```

```
            comidanovalida = false;
        }
        try {
            System.out.println("introduzca la comida del ultimo dia");
            comidafinal = ValorComida();
        } catch (Excepcioncomida ex) {
            System.out.println("Debe ser un valor entero entre 1 y 300000");
            comidanovalida = false;
        }
        if (comidainicial >= comidafinal) {
            comidanovalida = false;
        }
    } while (comidanovalida == false);
    try {
        comida1 = new ComidaIncrementoLineal(fechainicio, fechafin, comidainicial, comidafinal);
    } catch (Excepcioncomida ex) {
        ex.printStackTrace();
    }
    return comida1;
}

static private int[] cantidadesComidaIncDec() throws Excepcioncomida {
    boolean comidanovalida = false;
    int comidainicial = 0;
    int comidapico = 0;
    int comidafinal = 0;
    int comidainfo[] = new int[3];
    do {
        try {
            System.out.println("Introduzca la cantidad de comida inicial");
            Scanner scannercomidainicial = new Scanner(System.in);
            comidainicial = scannercomidainicial.nextInt();
            if (comidainicial >= 300000 || comidainicial <= 0) {
                throw new Excepcioncomida("rango");
            } else {
                comidanovalida = true;
            }
        } catch (InputMismatchException excepcion) {
            System.out.println("Tiene que ser un valor de tipo entero");
        }
    } while (false == comidanovalida);
    comidanovalida = false;
    do {
        try {
            System.out.println("Introduzca la cantidad de comida final");
            Scanner scannercomidafinal = new Scanner(System.in);
            comidafinal = scannercomidafinal.nextInt();
            if (comidafinal >= 300000 || comidafinal <= 0) {
                throw new Excepcioncomida("rango");
            } else {
                comidanovalida = true;
            }
        } catch (InputMismatchException excepcion) {
```

```
            System.out.println("Tiene que ser un valor de tipo entero");
        }
    } while (false == comidanovalida);
    comidanovalida = false;
    do {
        try {
            System.out.println("Introduzca la cantidad de comida el ultimo dia que esta incrementa");
            Scanner scannercomidapico = new Scanner(System.in);
            comidapico = scannercomidapico.nextInt();
            if (comidapico >= 300000 || comidapico <= 0) {
                throw new Excepcioncomida("rango");
            }
            if (comidapico <= comidafinal) {
                throw new Excepcioncomida("comidafinal");
            }
            if (comidapico <= comidainicial) {
                throw new Excepcioncomida("comidainicial");
            } else {
                comidanovalida = true;
            }
        } catch (InputMismatchException excepcion) {
            System.out.println("Tiene que ser un valor de tipo entero");
        }
    } while (false == comidanovalida);

    comidainfo[0] = comidainicial;
    comidainfo[1] = comidapico;
    comidainfo[2] = comidafinal;

    return comidainfo;
}

static private ComidaIncrementoDecremento CrearComidaIncrementoDecremento() {
    Fecha fechainicio;
    Fecha fechapico;
    Fecha fechafin;
    int comidainicial;
    int comidapico;
    int comidafinal;
    int comidainfo[] = new int[3];
    boolean noexcepcion = false;
    do {
        try {
            comidainfo = cantidadesComidaIncDec();
            noexcepcion = true;
        } catch (Excepcioncomida excepcion) {
            if (excepcion.getMessage() == "rango") {
                System.out.println("La comida debe ser un numero entero entre 1 y 300,000 ");
                noexcepcion = false;
            }
            if (excepcion.getMessage() == "comidafinal") {
                System.out.println("La comida final debe ser menor a la comida del ultimo dia de incremento de comida");
                noexcepcion = false;
```

```
            }
            if (excepcion.getMessage() == "comidainicial") {
               System.out.println("La comida inicial debe ser menor a la comida del ultimo dia de incremento de comida");
               noexcepcion = false;
            }
         }
      } while (noexcepcion == false);
//los valores del array se vuelven a trasladar a variables de tipo int
      comidainicial = comidainfo[0];
      comidapico = comidainfo[1];
      comidafinal = comidainfo[2];
//estas variables servirán para instanciar la clase Comida.
      System.out.println("Introduzca cuando la fecha en la que empezó la poblacion");
      fechainicio = CrearFecha();
      fechafin = CrearFecha();
      //fechafin = fechainicio.fechaFinal();
      System.out.println("Introduzca la fecha del ultimo dia en la que se incrementa la comida de la poblacion");
      fechapico = CrearFecha();
      //comprobamos que tanto la fecha pico es mayor a la fecha inicial como que la fecha final es mayor a la pico
      do {
         noexcepcion = true;
         if (fechainicio.anteriorA(fechapico) == false) {
            noexcepcion = false;
            System.out.println("La fecha del ultimo dia en la que se incrementa la comida de la poblacion tiene que"
                  + "estar entre la fecha" + fechainicio + "y" + fechafin);
            fechapico = CrearFecha();
         }
         if (fechapico.anteriorA(fechafin) == false) {
            noexcepcion = false;
            System.out.println("La fecha del ultimo dia en la que se incrementa la comida de la poblacion tiene que"
                  + "estar entre la fecha" + fechainicio + "y" + fechafin);
            fechapico = CrearFecha();
         }
      } while (noexcepcion == false); //mientras que la fechapico(fecha que representa el ultimo dia de incremento
      //de comida no este entre la fecha en la que comienza la poblacion y la fecha en la que termina.
      ComidaIncrementoDecremento comida1 = null;
      try {
         comida1 = new ComidaIncrementoDecremento(fechainicio, fechapico, fechafin, comidainicial, comidapico, comidafinal);
         return comida1;
      } catch (Excepcioncomida ex) {
         ex.printStackTrace();
      }
      return comida1;
   }

   //CREAR ATRIBUTO ENUMERACION (despues se introducirá en una poblacion de bacterias).
static private String ComprobarEnumeracion() throws Excepcionenumeracion {
      Boolean valido = false;
      String opcionluminosidad = "";
      do {
         try {
            System.out.println("Introduzca la luminosidad: (Alta, Media o Baja)");
            Scanner luminosidadpoblacion1 = new Scanner(System.in);
```

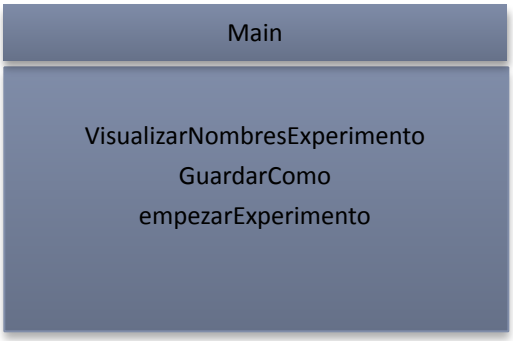
```
            opcionluminosidad = luminosidadpoblacion1.nextLine();
            if (opcionluminosidad.equals("Alta") || opcionluminosidad.equals("Media")
                || opcionluminosidad.equals("Baja")) {
              valido = true;
            } else {
              throw new Excepcionenumeracion(opcionluminosidad);
            }
          } catch (InputMismatchException excepcion) {
            System.out.println("Debe ser una cadena de caracteres");
            valido = false;
          }
        } while (valido == false);
        return opcionluminosidad;
    }

//CREAR POBLACION
    static public Poblacionbacteria CrearPoblacion() {
        String nombre = "";
        int numerobacteriasinicial = 0;
        float temperatura = 0F;
        String opcionluminosidad = "";
        luminosidad luz = luminosidad.Alta;
        Boolean valido = false;
        Comida comidapoblacion = null;
        do {
          try {
            System.out.println("Introduzca el nombre de la poblacion");
            Scanner nombrepoblacion = new Scanner(System.in);
            nombre = nombrepoblacion.nextLine();
            valido = true;
          } catch (InputMismatchException excepcion) {
            System.out.println("Tiene que ser una cadena de caracteres");
            valido = false;
          }
        } while (valido == false);
        // Temperatura
        valido = false;

        do {
          try {
            System.out.println("Introduzca la temperatura");
            Scanner temperaturapoblacion = new Scanner(System.in);
            temperatura = temperaturapoblacion.nextFloat();
            valido = true;
          } catch (InputMismatchException excepcion) {
            System.out.println("Tiene que introducir un valor de tipo real");
            valido = false;
          }
        } while (valido == false);

        valido = false;
        //Luminosidad
        do {
```

```
        try {
            opcionluminosidad = ComprobarEnumeracion();
            valido = true;
        } catch (Excepcionenumeracion excepcion) {
            System.out.println("Tiene que una de las tres opciones:Alta,Media,Baja");
            valido = false;
        }
    } while (valido == false);
    switch (opcionluminosidad) {
        case "Alta": {
            luz = luminosidad.Alta;
            break;
        }
        case "Media": {
            luz = luminosidad.Media;
            break;
        }
        case "Baja": {
            luz = luminosidad.Baja;
            break;
        }
    }
    //Comida de la poblacion;
    boolean numeroescogido = true;
    int numero = 0;
    do {
        try {
            System.out.println("Que formato de suministro de comida desea que tenga la poblacion:");
            System.out.println("1)constante");
            System.out.println("2)Incremento Lineal");
            System.out.println("3)Incremento hasta una fecha pico y despues decremento hasta el final");
            System.out.println("4)Intermitente");
            Scanner s = new Scanner(System.in);
            numero = s.nextInt();
        } catch (InputMismatchException e) {
            System.out.println("Debe ser un numero entero");
        }
        if (numero == 1 || numero == 2 || numero == 3 || numero == 3) {
            numeroescogido = true;
        }
    } while (numeroescogido == false);
    switch (numero) {
        case 1: {
            comidapoblacion = CrearComidaConstante();
            break;
        }
        case 2: {
            comidapoblacion = CrearComidaIncrementoLineal();
            break;
        }
        case 3: {
            comidapoblacion = CrearComidaIncrementoDecremento();
            break;
```

```
        }
        case 4: {
            comidapoblacion = CrearComidaIntermitente();
            break;
        }
    }
    Poblacionbacteria poblacion = null;
    boolean bacteriasnovalido;
    do {
        try {
            //numero de bacterias inicial
            System.out.println("Introduzca el numero de bacterias inicial");
            Scanner numerobacterias = new Scanner(System.in);
            numerobacteriasinicial = numerobacterias.nextInt();
            bacteriasnovalido = true;
        } catch (InputMismatchException excepcion) {
            System.out.println("Tiene que ser un numero entero");
            bacteriasnovalido = false;
        }
        try {
            poblacion = new Poblacionbacteria(nombre, comidapoblacion, numerobacteriasinicial, temperatura, luz);
            bacteriasnovalido = true;
        } catch (NumeroInicialBacteriasNoValidoException ex) {
            System.out.println("El numero inicial de bacterias no es valido");
            bacteriasnovalido = false;
        }
    } while (bacteriasnovalido == false);
    return poblacion;
  }

}
```

CLASE Main

La clase main tiene tres funciones, una que hace comenzar el programa y que tiene un switch con las operaciones que se pueden realizar empezarExperimento() y las Funciones GuardarComo para la reutilizacón de codigo y VisualizarNombresExperimento. Ambas se llaman dentro de la funcion empezarExperimento(). He decidido incluir estas funciones en una clase separada con el fin de hacer una separacion entre el inputoutput con una finalidad de crear objetos, y el inputoutput de las opciones generales.



1. Código

```
public class Main {
  //Funcion visualizar nombres

public static void VisualizarNombresExperimento(Experimento experimento) {
    int opcion = 0;
    do {
      try {
        System.out.println("Desea visualizar los nombres de las poblaciones del experimento segun:");
        System.out.println("1) Orden alfabetico");
        System.out.println("2) Por fecha : desde la fecha de comienzo mas antigua hasta la mas reciente");
        System.out.println("3) Por numero de bacterias iniciales");
        Scanner s = new Scanner(System.in);
        opcion = s.nextInt();
      } catch (InputMismatchException e) {
        System.out.println("Debe ser un numero entero");
      }
    } while (opcion != 1 && opcion != 2 && opcion != 3);
    switch (opcion) {
      case 1: {
        OrdenacionPorNombre(experimento.getPoblacionesBacteria());
        break;
      }
      case 2: {
        OrdenacionPorFecha(experimento.getPoblacionesBacteria());

        break;
      }
      case 3: {
        OrdenacionPorBacteriasIniciales(experimento.getPoblacionesBacteria());
        break;
      }

    }
    for (String e : experimento.visualizarNombres()) {
      System.out.println(e);
    }
  }
public static void GuardarComo(Experimento experimento1, File file1) {
    System.out.println("Introduzca el nombre del fichero en el que quiere guardar el experimento");
    Scanner nombre = new Scanner(System.in);
    String nombrefichero = nombre.nextLine();
    file1 = new File(nombrefichero);
    try {
      WriteExperimentToFile(file1, experimento1);
    } catch (IOException a) {
      System.out.println("no se ha podido guardar el experimento en el fichero");
    } catch (NullPointerException b) {
      System.out.println("No hay poblaciones que añadir");
```

```
    }
  }

  //FUNCION PARA COMENZAR A CREAR EXPERIMENTOS

  static public void empezarExperimento() {
    Experimento experimento1 = null;
    int opcion = 0;
    boolean numerovalido = true;
    File file1 = null;
    do {
      menu();
      Scanner opcionmenu = new Scanner(System.in);
      opcion = opcionmenu.nextInt();
      if (opcion < 1 || opcion > 10) {
        System.out.println("Las opciones validas abarcan desde el 1-8, si desea cerrar el programa pulse 0");
        opcion = opcionmenu.nextInt();
        if (opcion == 0) {
          exit(0);
        }
      }
      switch (opcion) {
        case 1: {

          System.out.println("Introduzca el nombre del fichero al que quiere acceder (ingresar ruta completa)");
          Scanner nombre = new Scanner(System.in);
          String nombrefichero = nombre.nextLine();
          File filenuevo = new File(nombrefichero);
          file1 = filenuevo;
          try {
            experimento1 = readExperimentFromFile(file1);
          } catch (FileNotFoundException a) {
            System.out.println("el fichero ingresado no se ha encontrado");
            //ARREGLAR POBLACION MISMO NOMBRE
          } catch (IOException b) {
            System.out.println("Ha habido un problema con la lectura del archivo, por lo que sigue ingresado el experimento anterior, o ninguno en el caso de que
no se haya hecho o ingresado uno previamente");
          } catch (ExcepcionEstructuraIncorrecta c) {
            System.out.println("Ha abierto un fichero que contiene informacion que es un experimento con poblaciones de bacterias");
          } catch (MismaPoblacionException ex) {
            Logger.getLogger(InputOutput.class.getName()).log(Level.SEVERE, null, ex);
          } catch (Excepcionfecha ex) {
            Logger.getLogger(InputOutput.class.getName()).log(Level.SEVERE, null, ex);
          } catch (Excepcioncomida ex) {
            Logger.getLogger(InputOutput.class.getName()).log(Level.SEVERE, null, ex);
          } catch (NumeroInicialBacteriasNoValidoException ex) {
            Logger.getLogger(InputOutput.class.getName()).log(Level.SEVERE, null, ex);
          }
          break;
        }
        case 2: {
          //utilizamos el constructor que obliga al experimento a crear meter una poblacion de bacterias inicial
          experimento1 = new Experimento(CrearPoblacion());
```

```
        break;
    }
    case 3: {
        if (experimento1 == null) {
            System.out.println("Debe haber un experimento abierto para ingresar una poblacion");
            break;
        }
        Poblacionbacteria poblacionañadir = CrearPoblacion();
        try {
            experimento1.añadirPoblacion(poblacionañadir);
        } catch (MismaPoblacionException excepcion) {
            System.out.println("La poblacion a añadir ya existe");
        }
        break;
    }
    case 4: {
        try {
            VisualizarNombresExperimento(experimento1);
        } catch (NullPointerException excepcion) {
            System.out.println("No se ha creado el experimento aun");
        }
        break;
    }
    case 5: {
        System.out.println("NOMBRES DE LAS POBLACIONES DEL EXPERIMENTO");
        try {
            for (String e : experimento1.visualizarNombres()) {
                System.out.println(e);
            }
            System.out.println("Introduzca el nombre de la poblacion "
                    + "de bacterias que desea eliminar");
            Scanner nombrepoblacion = new Scanner(System.in);
            String nombre = nombrepoblacion.nextLine();
            try {
                experimento1.borrarPoblacionBacteria(nombre);
            } catch (NoExistePoblacion ex) {
                System.out.println("No existe una poblacion con este nombre");
            }
        } catch (NullPointerException excepcion) {
            System.out.println("No se ha creado el experimento aun");
        } catch (InputMismatchException e) {
            System.out.println("El nombre debe ser una cadena de caracteres");
        }
        break;
    }
    case 6: {
        String nombrepoblacion;
        Poblacionbacteria poblacionimprimir;
        try {
            System.out.println("Introduzca el nombre de la poblacion");
            Scanner nombre = new Scanner(System.in);
            nombrepoblacion = nombre.nextLine();
            poblacionimprimir = experimento1.getPoblacionBacteria(nombrepoblacion);
```

```
            System.out.println("El experimento tiene una poblacion de bacterias y es la siguiente: \n" + poblacionimprimir);
            int valorescomida[] = poblacionimprimir.getComida().CantidadComidaCadaDia();
            for (int i = 0; i < valorescomida.length; i++) {
                System.out.println("La comida el dia" + (i + 1) + "es:   " + valorescomida[i]);
            }
        } catch (NullPointerException excepcion) {
            System.out.println("No se ha creado el experimento aun");
        } catch (InputMismatchException e) {
            System.out.println("El nombre debe ser una cadena de caracteres");
        } catch (NoExistePoblacion e) {
            System.out.println("No existe una poblacion con este nombre");
        }
        break;
    }
    case 7: {
        try {
            String nombrepoblacion;
            Poblacionbacteria poblacionimprimir;
            System.out.println("Introduzca el nombre de la poblacion");
            Scanner nombre = new Scanner(System.in);
            nombrepoblacion = nombre.nextLine();
            int bacteriascadadia[][][] = experimento1.getPoblacionBacteria(nombrepoblacion).Montecarlo();
            System.out.println("HOLA"+bacteriascadadia.length);
            for(int dia=0;dia<bacteriascadadia.length;dia++){
                for(int fila=0;fila<20;fila++){
                    for(int columna=0;columna<20;columna++){
                        System.out.println("El dia "+dia+1+"hay"+bacteriascadadia[dia][fila][columna]+"en la celda"+"["+fila+1+"]"+"["+columna+1+"]");
                    }
                }
            }

            //for (int i = 0; i < bacteriascadadia.length; i++) {
              // System.out.println(bacteriascadadia[i]);
            //}

        } catch (NoExistePoblacion ex) {
            System.out.println("No existe una poblacion con este nombre");
        }catch(NullPointerException e){
            System.out.println("Primero se debe abrir un experimento con poblaciones");
        }
        break;
    }
    case 8: {
        if (experimento1 == null) {
            System.out.println("Se debe tener un experimento abierto para guardar informacion");
            break;
        }
        if (file1 != null) {
            try {
                WriteExperimentToFile(file1, experimento1);
            } catch (IOException ex) {
                System.out.println("ha ocurrido un error escribiendo en el archivo");
            } catch (NullPointerException b) {
```

```
                System.out.println("No hay poblaciones que añadir");
                }
            } else {
                GuardarComo(experimento1, file1);
            }

            break;
        }
        case 9: {
            GuardarComo(experimento1, file1);
            break;
        }
        case 10: {
            exit(0);
        }
      }
    } while (true);
  }

  /**
   * Funcion menu() imprime las opciones que hay para hacer en la practica.
   */
  static public void menu() {
    System.out.println("MENU \n");
    System.out.println("1.Abrir un archivo que contenga un experimento \n");
    System.out.println("2. Crear un nuevo experimento \n");
    System.out.println("3. Crear una población de bacterias y añadirla al experimento actual \n");
    System.out.println("4. Visualizar los nombres de todas las poblaciones de bacterias del experimento actual\n");
    System.out.println("5. Borrar una población de bacterias del experimento actual \n");
    System.out.println("6. Ver informacion detallada de una poblacion de bacterias del experimento actual \n");
    System.out.println("7. Realizar y visualizar la simulación correspondiente con una de las poblaciones\n"
        + "de bacterias del experimento \n");
    System.out.println("8. Guardar (se supone que para usar esta opción previamente hemos abierto un archivo \n");
    System.out.println("9. Guardar como \n");
    System.out.println("10. Cerrar programa");
  }

  /**
   * @param args the command line arguments
   */
  public static void main(String[] args) {

    empezarExperimento();
  }
}
```

Conclusion

En conclusión, ha sido un proyecto que definitivamente me ha exigido un alto nivel de abstracción y que por lo tanto me ha resultado complejo. En cuanto a la simulación, he tenido que abstraerme y pensar en las formas

de representar celdas dentro de un plato de cultivo, y como relacionarlo con un objeto de tipo bacteria. En cuanto a horas de trabajo, considero que ha debido exceder las 15 horas divididas entre múltiples semanas.