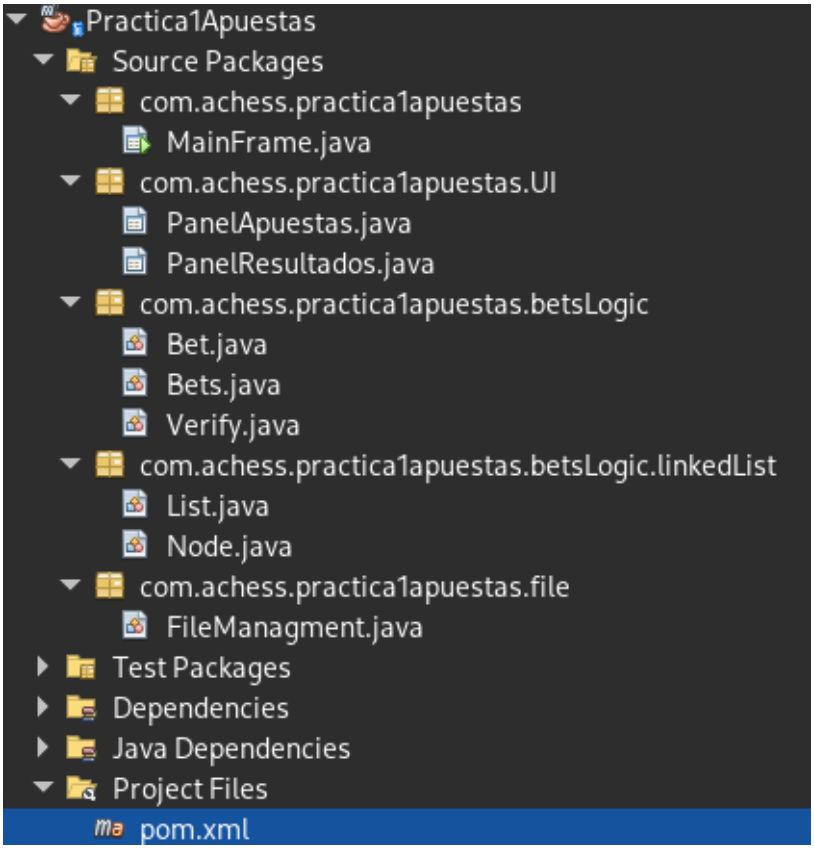
Manual técnico

Información del sistema:

- OS: `Arch Linux`
- Kernel: `x86_64 Linux 5.16.8-arch1-1`
- CPU: `Intel Core i3-4005U @ 4x 1.7GHz`
- GPU: `Intel Corporation Haswell-ULT Integrated Graphics Controller`
- RAM: `7881MiB`
- Versión de java: `11.0.13`
- IDE: `Apache NetBeans IDE 12.6`
- Control de versiones: `git version 2.35.1`

Estructura del proyecto



Estructuras de datos usadas

Arreglos

Los `arreglos` son usados para almacenar las posiciones de los caballos. Para los demás datos se usan listas propias.

Lista enlazada

Para armar la `lista enlazada` se usó una clase `Node` programada usando *Generics* para un uso más expandido. Para la lista se usó la clase `List` encargada de almacenar la cabeza de la lista y el tamaño de cada lista.

Servicios críticos

La mayor parte de estos servicios están en la clase `Bets`, la cual fue programada usando el patrón `Singleton`. En esta clase se encuentran los métodos para ordernar alfabéticamente

y por puntos. También para validar cada una de las apuestas y también, para guardar las apuestas antes de ser validadas y las rechazadas.

Verificación de apuestas

Para validar las apuestas se utiliza el método `validate` ubicado en la clase `Bets`

```
public void validate(){
        if(!validated){
            int stepsCount = 0;
            int maxSteps = stepsCount;
            int minSteps = stepsCount + 10;
            Long start = System.nanoTime();
            Node<Bet> aux = noVerified.pop();
            while(aux != null){
                if(Verify.validate(aux.getData().getPositions())){
                    accepted.push(aux);
                }
                else{
                    rejected.push(aux);
                }
                aux = noVerified.pop();
                int actualSteps = Verify.getSteps();
                stepsCount += actualSteps;
                if(actualSteps > maxSteps){
                    maxSteps = actualSteps;
                }
                if(actualSteps < minSteps){
                    minSteps = actualSteps;
                }
            }
            Long end = System.nanoTime();
            double averageTime = end.doubleValue() - start.doubleValue();
            averageTime = averageTime/noVerified.getLen();
            float averageSteps = stepsCount/noVerified.getLen();
            System.out.println("------VERIFICACIÓN DE RESULTADOS------");
            System.out.println("Tiempo promedio: " + averageTime + "ns");
            System.out.println("Pasos promedio: " + averageSteps + " pasos");
            System.out.println("Mayor cantidad de pasos: " + maxSteps + " pasos");
            System.out.println("Menor cantidad de pasos: " + minSteps + " pasos");
            System.out.println("--------------------------------------");
        }
        validated = true;
        //System.gc();
    }
```

En este método se valida que no se repita ninguna posición. Si se repite se guarda en la lista `accepted`. Si es rechazado se guarda en la lista `rejected`. Para validar esto se usa el

método `validate` que se encuentra en la clase `Verify`. Este método realiza 10 o menos pasos. Por lo tanto es un $O(1)$.

```
public static DefaultTableModel getModel(List<Bet> list){
        Node<Bet> aux = list.getHead();
        DefaultTableModel model = new DefaultTableModel();
        model.addColumn("No.");
        model.addColumn("Nombre");
        model.addColumn("Puntos");
        Integer line = 1;
        while(aux != null){
            Bet bet = aux.getData();
            Object[] data = {line, bet.getGamblerName(), bet.getPoints()};
            model.addRow(data);
            line++;
            aux = aux.getNext();
        }
        return model;
    }
```

Al final el método para validar todas las apuestas será un $O(n)$.

Cálculo de los resultados al finalizar la carrera

Para calcular los resultados se recorren todas las apuestas con el fin de asignar los puntos a cada apuesta. Si se acierta la primera posición se le asignarán 10 puntos, para la segunda 9 puntos y así, restanto los puntos, hasta la última posición.

Para esto se usan dos métodos:

```
public void calculateRetults(){
        if(!calculated){
            Long start = System.nanoTime();
            Double average = 0.0;
            Node<Bet> aux = accepted.getHead();
            while(aux != null){
                getPoints(aux.getData());
                aux = aux.getNext();
            }
            average = average / accepted.getLen();
            //System.out.printf("Tiempo promedio: %d ns", average);
            Long end = System.nanoTime();
            double averageTime = 0.0;
            averageTime = (end.doubleValue() - start.doubleValue());
            averageTime = averageTime/accepted.getLen();
            System.out.println("---------CÁLCULO DE RESULTADOS--------");
            System.out.println("Tiempo promedio: "+ averageTime+"ns");
            System.out.println("Promedio de pasos: 10 pasos");
            System.out.println("Mayor: 10 pasos");
```

```
            System.out.println("Menor: 10 pasos");
            System.out.println("-------------------------------------");
        }
        calculated = true;
    }
```

El primero que se encarga de recorrer todas las apuestas $O(n)$ y el segundo que calcula los puntos por cada posición. Al final son 10 posiciones por lo tanto siempre es un $O(1)$. Al final la complejidad queda como un $O(n)$

Ordenamiento de los resultados

Los resultados se pueden ordenar de dos maneras: por puntos y por orden alfabético. Para ordenar las lista se usó el método conocido como `MergeSort` que en el peor de los casos tiene una complejidad de $O(nlog(n))$.

```
private Node<Bet> mergeSort(Node<Bet> head, boolean points){
        steps++;
        if(head == null || head.getNext() == null){
            return head;
        }
        Node middle = getMiddle(head);
        Node nextoMiddle = middle.getNext();
        middle.setNext(null);
        Node left = mergeSort(head, points);
        Node right = mergeSort(nextoMiddle, points);
        if(points) return SortedMergeForPoints(left, right);
        return SortedMergeForNames(left, right);

    }
```

Este es el método principal. Básicamente, se encarga de retornar la nueva cabeza de la lista ya ordenada. Consiste en dividir la lista en dos partes y ordenar esas dos partes por separado utlizando el mismo método.

```
private Node getMiddle(Node head){
        steps++;
        if(head == null){
            return head;
        }

        Node slow = head, fast = head;
        while(fast.getNext() != null && fast.getNext().getNext() != null){
            slow = slow.getNext();
            fast = fast.getNext().getNext();
            steps++;
```

```
        }
        return slow;
    }
```

Este método se encarga de obtener el nodo que se encuentra en la mitad de la lista.

```
private Node<Bet> SortedMergeForPoints(Node<Bet> a, Node<Bet> b){
        steps++;
        Node<Bet> result = null;
        if(a == null){
            return b;
        }
        if(b == null){
            return a;
        }
        if(a.getData().getPoints() >= b.getData().getPoints()){
            result = a;
            result.setNext(SortedMergeForPoints(a.getNext(), b));
        }else{
            result = b;
            result.setNext(SortedMergeForPoints(a, b.getNext()));
        }
        return result;
    }

private Node<Bet> SortedMergeForNames(Node<Bet> a, Node<Bet> b){
        steps++;
        Node<Bet> result = null;
        if(a == null){
            return b;
        }
        if(b== null){
            return a;
        }
        if(a.getData().getGamblerName().compareToIgnoreCase(b.getData().getGamblerName()) <= 0){
            result = a;
            result.setNext(SortedMergeForNames(a.getNext(), b));
        }else{
            result = b;
            result.setNext(SortedMergeForNames(a, b.getNext()));
        }
        return result;
    }
```

Estos dos métodos hacen lo mismo, pero uno es para ordenar por puntos y el otro para ordenar alfabéticamente. Básicamente recibe dos nodos cabeza. Compara las cabezas y según sea el criterio, una de las dos cabezas sera la cabeza de la nueva lista (producto de la unión las dos). Para obtener el siguiente de la cabeza se vuelve a llamar al mismo método.