```java
package Avaliador;

import java.io.BufferedReader;

public class AvaliadorAmbiente extends Avaliador{
    /**
     * Variaveis para as medias dos valores dos sensores
     */
    private double media_escuro;
    private double media_claras;
    private double media_claras_mov;
    private double media_escuro_mov;

    /**
     * Variaveis para guardar os valores correspondentes às leituras dos sensores
     */
    private ArrayList<ValoresSensores> valores_escuro;
    private ArrayList<ValoresSensores> valores_claras;
    private ArrayList<ValoresSensores> valores_claras_mov;
    private ArrayList<ValoresSensores> valores_escuro_mov;

    /**
     * Numero de leituras totais para cada teste
     */
    private static final int trys = 20;

    /**
     * Numero de leitruas para cada movimento em cada teste
     */
    private static final int sub_trys = 4;

    /**
     * Variavel que representa o valor calculado para os limiares entre branco e preto
     */
    private int limiarEsquerda;
    private int limiarDireita;

    /**
     * Variavel para verificar se as medições também são para fazer com o robot parado
     */
    private boolean toMeasureStoped;


    /**
     * Metodo Construtor
     * @param r HemissonRobot a ser avaliado
     */
    public AvaliadorAmbiente(HemissonRobot r) {
        super(r);
        valores_escuro = new ArrayList<ValoresSensores>();
        valores_claras = new ArrayList<ValoresSensores>();
        valores_claras_mov = new ArrayList<ValoresSensores>();
        valores_escuro_mov = new ArrayList<ValoresSensores>();
        this.toMeasureStoped = false;
    }


    /**
     * Metodo Construtor
     * @param r HemissonRobot a ser avaliado
     * @param readStoped boolean para verificar se intressam as medições com o robot
 parado
     */
    public AvaliadorAmbiente(HemissonRobot r, boolean readStoped) {
        super(r);
        valores_escuro = new ArrayList<ValoresSensores>();
```

```
        valores_claras = new ArrayList<ValoresSensores>();
        valores_claras_mov = new ArrayList<ValoresSensores>();
        valores_escuro_mov = new ArrayList<ValoresSensores>();
        this.toMeasureStoped = readStoped;
    }

    /**
     * Metodo que inicia as medições
     */
    public void init() {

        BufferedReader teclado = new BufferedReader (new InputStreamReader(System.in));
        System.out.println("A iniciar medições para ajuste dos sensores");

        if(toMeasureStoped) {
            System.out.println("Coloque o robo sobre uma superficie branca");
            try {
                teclado.readLine(); //Para bloquear
            } catch (IOException e) {
                e.printStackTrace();
            }
            robot.beeper(1);
            readFloorValues(valores_claras);
            robot.beeper(3);
            System.out.println("Coloque o robo sobre uma superficie preta");
            try {
                teclado.readLine(); //Para bloquear
            } catch (IOException e) {
                e.printStackTrace();
            }
            robot.beeper(1);
            readFloorValues(valores_escuro);
            robot.beeper(3);
        }

        System.out.println("Coloque o robo sobre uma superficie clara para mover");
        try {
            teclado.readLine(); //Para bloquear
        } catch (IOException e) {
            e.printStackTrace();
        }
        robot.setSpeed(5, -5);
        robot.beeper(1);
        readFloorValues(valores_claras_mov);
        robot.beeper(3);
        robot.setSpeed(0,0);
        System.out.println("Coloque o robo sobre uma linha escura para mover");
        try {
            teclado.readLine(); //Para bloquear
        } catch (IOException e) {
            e.printStackTrace();
        }

        robot.beeper(1);
        readFloorValuesWithMove(valores_escuro_mov);
        robot.beeper(3);

        robot.setSpeed(0,0);

        if(this.toMeasureStoped) {
            this.media_claras = calculaMediaDesvio(valores_claras);
            this.media_escuro = calculaMediaDesvio(valores_escuro);
        }
        this.media_claras_mov = calculaMediaDesvio(valores_claras_mov);
        this.media_escuro_mov = calculaMediaDesvio(valores_escuro_mov);
```

```
        calcularLimiar();
    }

    /**
     * Metodo que calcula o desnivelamento entre o escuro e o claro
     * @return double desnivelamento
     */
    public double getDesnivelamento() {
        return (this.media_claras_mov + this.media_escuro_mov) / 2;
    }

    /**
     * Metodo que lê os valores dos sensores com movimento
     * @param valores ArrayList para guardar os valores lidos
     */
    private void readFloorValuesWithMove(ArrayList<ValoresSensores> valores) {
        valores.clear();
        robot.setSpeed(3,3);
        boolean added = true;
        for(int i = 0; added && i < trys; i++) {
            for(int j = 0; j < sub_trys && i < trys; j++,i++) {
                int[] values = robot.readSonar();
                added = valores.add(new ValoresSensores(values[5], values[3]));
            }
            int[] speed = robot.readSpeed();
            robot.setSpeed(speed[0] * -1, speed[1] * -1);
        }
    }


    /**
     * Metodo que calcula a media ponderada dos valores
     * @param valores ArrayList que contem os valores para se efectuar a media
     * @return double media ponderada dos valores recebidos em argumento
     */
    private double calculaMediaDesvio(ArrayList<ValoresSensores> valores) {
        double somatorio = 0;
        for(int i = 0; i<valores.size(); i++)
            somatorio +=((ValoresSensores)valores.get(i)).getDiferencaAbsoluta();

        return somatorio / valores.size();
    }

    /**
     * Metodo que lê os valores dos sensores
     * @param valores ArrayList para guardar os valores lidos
     */
    private void readFloorValues(ArrayList<ValoresSensores> valores) {
        valores.clear();
        boolean added = true;
        for(int i = 0; added && i < trys; i++) {
            int[] values = robot.readSonar();
            added = valores.add(new ValoresSensores(values[5], values[3]));
        }
    }

    /**
     * Metodo que guarda em ficheiro os dados recolhidos
     * @param filename String com o nome do ficheiro para os dados
     */
    public void dumpDados(String filename) {
        try {
            FileWriter f = new FileWriter(filename);
            if(this.toMeasureStoped) {
                f.write("Media às claras = " + this.media_claras + "\n");
                f.write("Media às escuras = " + this.media_escuro + "\n");
```

```
            }
            f.write("Media às claras em movimento = " + this.media_claras_mov + "\n");
            f.write("Media às escuras em movimento = " + this.media_escuro_mov + "\n");
            f.write("\n");

            if(this.toMeasureStoped) {
                f.write("Valores às claras:\n");

                for(int i = 0; i<this.valores_claras.size(); i++) {
                    f.write(((ValoresSensores) valores_claras.get(i)).toString());
                }
                f.write("\n");

                f.write("Valores às Escuras:\n");
                for(int i = 0; i<this.valores_escuro.size(); i++) {
                    f.write(((ValoresSensores) valores_escuro.get(i)).toString());
                }
                f.write("\n");
            }

            f.write("Valores às claras Com movimentação:\n");
            for(int i = 0; i<this.valores_claras_mov.size(); i++) {
                f.write(((ValoresSensores) valores_claras_mov.get(i)).toString());
            }
            f.write("\n");

            f.write("Valores às escuras Com movimentação:\n");
            for(int i = 0; i<this.valores_escuro_mov.size(); i++) {
                f.write(((ValoresSensores) valores_escuro_mov.get(i)).toString());
            }
            f.write("\n");

            f.flush();
            f.close();
        } catch (IOException e) {
            e.printStackTrace();
        }

    }

    /**
     * Metodo que devolve o limiar à esquerda
     * @return int valor limiar à esquerda
     */
    public int getLimiarEsquerda()  {
        return this.limiarEsquerda;
    }

    /**
     * Metodo que devolve o limiar à direita
     * @return int valor limiar à direita
     */
    public int getLimiarDireita() {
        return this.limiarDireita;
    }

    /**
     * Metodo que calcula os limiares
     */
    public void calcularLimiar() {
        int minClarasEsquerda = findMinimo(valores_claras_mov,0);
        int maxEscuroEsquerda = findMaximo(valores_escuro_mov,0);
        int minClarasDireita = findMinimo(valores_claras_mov,1);
        int maxEscuroDireita = findMaximo(valores_escuro_mov,1);

        this.limiarEsquerda = (minClarasEsquerda + maxEscuroEsquerda)/2;
```

```
        this.limiarDireita = (minClarasDireita + maxEscuroDireita)/2;
    }

    /**
     * Metodo que procura o maximo dado o lado
     * @param valores ArrayList com os valores lidos
     * @param lado inteiro correspondente ao lado que queremos o maximo
     * @return int valor maximo encontrado
     */
    private int findMaximo(ArrayList<ValoresSensores> valores, int lado) {
        int max;
        if(lado == 0)
            max = ((ValoresSensores)valores.get(0)).getEsquerda();
        else
            max = ((ValoresSensores)valores.get(0)).getDireita();

        for(int i = 1; i<valores.size(); i++)
            if(lado == 0)
            {
                if(max < ((ValoresSensores)valores.get(i)).getEsquerda())
                    max = ((ValoresSensores)valores.get(i)).getEsquerda();
            }
            else
            {
                if(max < ((ValoresSensores)valores.get(i)).getDireita())
                    max = ((ValoresSensores)valores.get(i)).getDireita();
            }
        return max;
    }


    /**
     * Metodo que procura o minimo dado o lado
     * @param valores ArrayList com os valores lidos
     * @param lado inteiro correspondente ao lado que queremos o minimo
     * @return int valor minimo encontrado
     */
    private int findMinimo(ArrayList<ValoresSensores> valores, int lado) {
        int min;
        if(lado == 0)
            min = ((ValoresSensores)valores.get(0)).getEsquerda();
        else
            min = ((ValoresSensores)valores.get(0)).getDireita();

        for(int i = 1; i<valores.size(); i++)
            if(lado == 0)
            {
                if(min > ((ValoresSensores)valores.get(i)).getEsquerda())
                    min = ((ValoresSensores)valores.get(i)).getEsquerda();
            }
            else
            {
                if(min > ((ValoresSensores)valores.get(i)).getDireita())
                    min = ((ValoresSensores)valores.get(i)).getDireita();
            }
        return min;
    }
}
```