3. ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

3.1 Описание классов

Класс *AudioProcessor* используется для инициализации микрофона и управления процессом обработки аудиосигнала.

Private поля класса	
static final int SAMPLE_RATE = 22050	Частота дискретизации по умолчанию
static final int DEFAULT_BUFF_SIZE = 16384	Размер буфера чтения сигнала с микрофона по умолчанию
static final double ALLOWED_FREQUENCY_DIFFERENCE = 1	Максимальная разница между предыдущей и нынешней частотой, при которой последняя не является шумом
static final int MIN_FREQUENCY = 50;	Минимальное значение распознаваемой частоты
static final int MAX_FREQUENCY = 500;	Максимальное значение распознаваемой частоты
AudioRecord audioRecord;	Стандартный Android класс для работы с микрофоном
FrequencyDetectionListener frequencyDetectionListener	Интерфейс, который связывает модули управления и обработки аудио
double lastComputedFrequency = 1	Значение последней распознанной частоты
int buffSize	Размер буфера
boolean stopFlag = false;	Флаг остановки обработки сигнала
Public методы	
void setFrequencyDetectionListener (FrequencyDetectionListener frequencyDetectionListener)	Задание интерфейса «слушателя», отслеживающего нахождение частоты
void init()	Инициализация микрофона
void stop()	Рутина окончания обработки сигнала
void run()	Запуск обработки сигнала
Private методы	
double[] shortToDouble(short[] source, int length)	Конвертация ИКМ из целочисленного формата(short) к формату с плавающей запятой(double)

Класс *FrequencyDetector* занимается анализом поступившей ИКМ и как результат выдает распознанную частоту.

Private поля класса	
static final int peaksAmount = 15	Число рассматриваемых пиковых значений амплитуды
Public методы	
double findFrequency(double[] soundData, int sampleRate, double minFreq, double maxFreq, FastFourierTransform specificFFT, Window window)	Нахождение доминирующей частоты звучащего сигнала
Private методы	
Pair<Double, Integer> scanSignalIntervals(double[] x, int index, int length, int intervalMin, int intervalMax)	Сканирование промежуточных периодов между дискретными значениями полученных частот, для повышения точности
int[] findPeaks(double[] values, int index, int length, int peaksCount)	Нахождения индексов пиковых амплитуд сигнала

Интерфейс *FrequencyDetectionListener* является вложенным для класса *AudioProcessor* и используется для «подписки» класса *TunerActivity* на обновления текущей частоты.

Методы	
void onFrequencyDetected(double freq);	Рутина, которая будет выполняться при получении нового значения частоты

Интерфейс *Window* описывает некоторую оконную функцию, применяемую ко входной ИКМ, чтобы минимизировать присутствующие шумы и тем самым упростить дальнейшие вычисления.

Методы	
double[] applyWindow(double[] inputData);	Применение оконной функции

Интерфейс *FastFourierTransform* нужен для того чтобы не отталкиваться от конкретных реализация преобразований и чтобы иметь возможность сравнивать результаты различных реализаций в будущем.

Методы	
double[] calculateSpectrogram(double[] inputData);	Рассчитать спектрограмму относительно данной ИКМ

Класс *HammingWindow* реализует интрефейс *Window*, предоставляя оконную функцию Хэмминга.

Public методы	
double[] applyWindow(double[] data)	Реализация метода интерфейса через окно Хэмминга
Private методы	
double iterationHamming(double n, int size)	Итерация оконной функции, при ее наложении

Класс *FFTCooleyTukey* реализует интерфейс *FastFourierTransform*, а именно алгоритм быстро преобразования Фурье по Кули-Тьюки, являющийся одним из самых быстрых алгоритмов БПФ.

Public методы	
double[] calculateSpectrogram(double[] inputData)	Вычисление спектрограммы входного сигнала, представленного ИКМ
Private методы	
int log2(int n)	Расчет логарифма по основанию 2
int reverseBits(int n, int bitsCount)	Применение преобразования «бабочка» ко входным данным
boolean isPowerOfTwo(int n)	Установить, является ли число степенью двойки, и вернуть ближайшее подходящее значение

Класс *Note* служит для хранения информации о ноте.

Private поля класса	
double frequency	Частота ноты
String name	Имя ноты
Public методы	
Note(double frequency, String name)	Конструктор
double getFrequency()	Получить частоту ноты
void setFrequency(double frequency)	Задать частоту ноты
String getName()	Получит имя ноты
void setName(String name)	Задать имя ноты

Класс *Tuning* для хранения данных о возможных настройках гитары, а так же для нахождения ближайшей подходящей ноты, отталкиваясь от частоты.

Private поля класса	
String name	Имя настройки
Note[] notes	Ноты настройки
Public методы	
Tuning(String name, Note[] notes)	Конструктор

Note[] getNotes()	Получить массив нот настройки
String getName()	Получить имя настройки
Note closestNote(double frequency)	Получить ближайшую ноту
int closestNoteIndex(double frequency)	Получить индекс ближайшей ноты из текущей настройки
static Tuning getTuning(Context context, String name)	Получить текущую настройку из контекста приложения

Класс *Preferences* предоставляет API для работы с классом *SharedPreferences* данного приложения.

Public методы	
static SharedPreferences getPreferences(Context context)	Получить объект SharedPreferences
static boolean getBoolean(Context context, String key, boolean defaultValue)	Получить boolean значение по ключу
static String getString(Context context, String key, String defaultValue)	Получить строку по ключу

Класс *AppUtilities* используется для предоставления дополнительного функционала программы.

Private поля класса	
static final double LOG2 = Math.log(2)	Логарифм 2
Private методы	
AppUtilities()	Конструктор
Public методы	
static float dpToPixels(Context context, float dp)	Конвертация пикселей, независящих от экрана(dp) в реальные
static int getAttrColor(Context context, int attrId)	Получить цвет атрибута
static double log2(double v)	Логарифм по основанию 2
static void reveal(View view)	Отобразить знак успешной настройки
static void hide(final View view)	Спрятать знак успешной настройки
static void setupActivityTheme(Activity activity)	Установить тему Activity
static boolean checkPermission(Context context, String permission)	Проверить наличие разрешения использования некоторого системного ресурса
static void showSettingsActivity (Context context)	Установить Activity настроек

static void showPermissionDialog(Context context, String message, DialogInterface.OnClickListener listener)	Показать диалоговое окно, для получения разрешения на использование системного ресурса

Класс *TunerActivity* является главным окном приложения и представляет собой управляющий модуль.

Private поля класса	
static final int PERMISSION_REQUEST_RECORD_AUDIO = 443	Номер разрешения для использования микрофона
Tuning tuning	Текущая настройка гитары
AudioProcessor audioProcessor	Класс обработки аудиосигнала
ExecutorService executor = Executors.newSingleThreadExecutor()	Класс, отвечающий за исполнение потока обработки звука
NeedleView needleView	Вид иглы настройки
TuningView tuningView	Вид текущей настройки
TextView frequencyView	Вид текущей частоты
boolean isProcessing = false	Флаг протекания процесса обработки звука
int noteIndex	Индекс ноты из текущей настройки
double lastFrequency	Последняя подсчитанная частота
Public поля класса	
static final String STATE_NEEDLE_POS = "needle_pos"	Ключ для позиции кончика иглы
static final String STATE_NOTE_INDEX = "note_index"	Ключ для индекса ноты
static final String STATE_LAST_FREQ = "last_freq"	Ключ для значения последней подсчитанной частоты
Private методы	
void requestPermissions()	Запросить разрешение у системы
void startAudioProcessing()	Начать обработку аудиосигнала
Protected методы	
void onStart()	Начало работы Activity
void onStop()	Остановка работы Activity
void onResume()	Возобновление работы Activity
void onPause()	Приостановка работы Activity
void onCreate(Bundle savedInstanceState)	Создание Activity
void onSaveInstanceState(Bundle outState)	Сохранение состояния Activity
void onRestoreInstanceState(Bundle savedInstanceState)	Восстановление состояния Activity

Public методы	
void onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults)	Рутина по запросу результата получения разрешения
boolean onOptionsItemSelected(MenuItem item)	Рутина при выборе кнопки «опции»
boolean onCreateOptionsMenu(Menu menu)	Создание меню опций

Класс *SettingsActivity* представляет окно настроек.

Private поля класса	
boolean shouldRestart = false;	Флаг перезагрузки
SharedPreferences.OnSharedPreferenceChangeListener onSharedPreferenceChangeListener = new SharedPreferences.OnSharedPreferenceChangeListener()	Слушатель изменений данных в SharedPreferences
Private методы	
void setupActionBar()	Установка ActionBar
Protected методы	
void onCreate(Bundle savedInstanceState)	Создание Activity
void onPause()	Приостановка Activity
Public методы	
boolean onMenuItemSelected(int featureId, MenuItem item)	Рутина при выборе элемента меню
void onBackPressed()	Рутина при нажатии кнопки «назад»

Класс *NeedleView* используется для отображения арки и иглы настройки.

Private поля класса	
double angle	Угол поворота илы
Paint paint	Класс, занимающийся рисованием
float strokeWidth	Толщина stroke, для работы с размерами фигур, независимыми от реальных пикселей
float textStrokeWidth	Толщина stroke, для работы с размерами текста, независимого от реальных пикселей
float tickLabelTextSize	Размер текстового Label
float arcOffset	Отступ арки
float tickLength	Длина палочки арки
int needleColor	Цвет иглы

int smallTicksColor	Цвет малой палочки арки
int bigTicksColor	Цвет большой палочки арки
int textColor	Цвет текста
Map<Float, String> tickLabels = new HashMap<>()	Карта, где ключ – это позиция Label, а значение – строка с нужным текстом
Private методы	
void drawTickLabels(Canvas canvas, int width, int height)	Отрисовка текста над аркой
void drawNeedle(Canvas canvas, int width, int height, float tickLabelHeight)	Отрисовка иглы
void drawTicks(Canvas canvas, int width, int height, float tickLabelHeight)	Отрисовка палочек арки
void drawSmallTick(Canvas canvas, float height, float tickLabelHeight, float cx, float cy, float angle)	Отрисовка малых палочек арки
void drawBigTick(Canvas canvas, float height, float tickLabelHeight, float cx, float cy, float angle)	Отрисовка больших палочек арки
protected void onDestroy()	Уничтожение View
Protected методы	
void onDraw(Canvas canvas)	Рутина отрисовки View
Public методы	
NeedleView(Context context)	Конструктор
NeedleView(Context context, AttributeSet attrs)	Конструктор
NeedleView(Context context, AttributeSet attrs, int defStyleAttr)	Конструктор
void animateTip(float toPos)	Анимировать поворот иглы
double getAngle()	Получить угол поворота иглы
float getTipPosition()	Получить позицию кончика иглы
void setTipPosition(float pos)	Задать позицию кончика иглы
void setTickLabel(float pos, String label)	Задать Labels над аркой

Класс *TuningView* нужен для анимации изменений в текущей настройке гитары

Private поля класса	
int selectedIndex	Выбранный индекс в настройке
Tuning tuning	Текущая настройка

float tuningItemWidth	Ширина элемента настройки
Paint paint = new Paint()	Класс, занимающийся рисованием
Rect tempRect = new Rect()	Прямоугольник для отрисовки элемента настройки
int normalTextColor	Цвет обычного текста
int selectedTextColor	Цвет выбранного элемента
float offset = 0	Отступ элемента
ValueAnimator offsetAnimator = null	Аниматор
Private методы	
void init(Context context, AttributeSet attrs, int defStyleAttr)	Инициализация объекта класса
void stopAnimation()	Остановка анимации
Protected методы	
void onSizeChanged(int w, int h, int oldw, int oldh)	Рутина при изменении параметров экрана
void onDraw(Canvas canvas)	Рутина отрисовки View
Public методы	
TuningView(Context context)	Конструктор
TuningView(Context context, AttributeSet attrs)	Конструктор
TuningView(Context context, AttributeSet attrs, int defStyleAttr)	Конструктор
int getSelectedIndex()	Получить выбранный индекс
setSelectedIndex(int selectedIndex, boolean animate)	Задать текущий выбранный индекс с параметрами анимации
void setSelectedIndex(int selectedIndex)	Задать текущий выбранный индекс
float getTextSize()	Получить размер текста
void setTextSize(float textSize)	Задать размер текста
int getNormalTextColor()	Получить цвет обычного текста
void setNormalTextColor(int color)	Задать цвет обычного текста
int getSelectedTextColor()	Получить цвет выбранного элемента
void setSelectedTextColor(int selectedTextColor)	Задать цвет выбранного элемента
Tuning getTuning()	Получить текущую настройку
void setTuning(Tuning tuning)	Задать текущую настройку
float getTuningItemWidth()	Получить ширину элемента настройки
void setTuningItemWidth(float tuningItemWidth)	Задать ширину элемента настройки