

Лабораторная работа 2.

«Работа с аппаратными датчиками мобильного устройства»

Цель: Изучить принципы взаимодействия мобильных приложений с аппаратными датчиками устройства на платформе Android.

Задание: Разработать Android-приложение, предназначенное для работы с аппаратными датчиками устройства. Приложение должно обеспечивать получение данных с датчика, их обработку, визуализацию и интерактивное взаимодействие с пользователем.

Номер варианта соответствует последней цифре Вашего пароля.

Среда разработки: Android Studio.

Язык: Kotlin.

Вариант 1: Работа с датчиком гироскопа.

Порядок выполнения:

1. Зарегистрируйте `SensorEventListener` для `TYPE_GYROSCOPE`.
2. В `onSensorChanged`:
 - Получайте угловые скорости по осям X, Y, Z.
 - Интегрируйте скорость по оси Z (суммируйте значения).
3. Визуализация:
 - Вращайте `ImageView` (стрелку) через `rotation = -angle`
 - Выводите угол поворота в `TextView`
4. Добавьте кнопку сброс для обнуления угла.

Вариант 2: Работа с датчиком акселерометра.

Порядок выполнения:

1. Зарегистрируйте `SensorEventListener` для `TYPE_ACCELEROMETER`
2. В `onSensorChanged`:
 - Получайте значения по осям X, Y, Z.
 - Рассчитывайте общее ускорение: $\sqrt{x^2 + y^2 + z^2}$.
3. Визуализация:
 - При ускорении $> 15 \text{ m/s}^2$ менять цвет фона на красный.
 - При ускорении $< 5 \text{ m/s}^2$ - на зеленый.
 - Выводить текущее ускорение в `TextView`.
4. Добавьте кнопку сброса показаний.

Вариант 3: Детектор встряхивания.

Порядок выполнения:

1. Зарегистрируйте `SensorEventListener` для `TYPE_ACCELEROMETER`.
2. В `onSensorChanged`:
 - Фиксируйте встряхивание при ускорении $> 20 \text{ m/s}^2$
 - Защита от повторного срабатывания (таймер 1 сек)
3. Визуализация:
 - Выводите счетчик встряхиваний.
 - Мигайте фоном при обнаружении.
4. Добавьте кнопку сброса счетчика.

Вариант 4: Проверка ровности поверхности.

Порядок выполнения:

1. Зарегистрируйте `SensorEventListener` для `TYPE_GRAVITY`.
2. В `onSensorChanged`:
 - Получайте значения силы тяжести по осям X, Y, Z из `event.values`.
 - Рассчитывайте угол наклона устройства относительно горизонта.
3. Визуализация:
 - При угле $< 2^\circ$ по обеим осям показывайте зеленый цвет фона, что означает «Поверхность ровная».
 - При угле $> 5^\circ$ — красный цвет и Toast-уведомление: «Поверхность неровная».
 - Отображайте текущие углы наклона в `TextView`.
4. Добавьте кнопку калибровки для обнуления наклона.

Вариант 5: Работа с сенсором приближения.

Порядок выполнения:

1. Зарегистрируйте `SensorEventListener` для `TYPE_PROXIMITY`.
2. В `onSensorChanged`:
 - Определяйте расстояние до объекта (0-5 см).
 - Фиксируйте время покрытия сенсора.
3. Визуализация:
 - Приближение объекта: изменять цвет фона.
 - Выводить текущий статус и время покрытия в `TextView`.
4. Добавьте регулятор времени блокировки.

Вариант 6: Магнитный компас.

Порядок выполнения:

1. Зарегистрируйте `SensorEventListener` для `TYPE_MAGNETIC_FIELD`.
2. В `onSensorChanged`:
 - Получайте значения по осям X, Y, Z.
 - Рассчитайте направление: $\text{atan2}(y, x) * 180 / \text{PI}$
3. Визуализация:
 - Вращайте `ImageView` со стрелкой компаса.
 - Отображайте текущее направление (N/S/W/E) в `TextView`.
4. Добавьте кнопку "Калибровка", которая обнуляет смещение.

Вариант 7: Работа с датчиком освещенности

Порядок выполнения:

1. Зарегистрируйте `SensorEventListener` для `TYPE_LIGHT`.
2. В `onSensorChanged`:
 - Получайте уровень освещённости (люксы).
 - Добавьте логику для изменения цвета фона при критическом уровне (например, < 50 лк).
3. Визуализация:

- При освещенности < 50 лк: темная тема (черный фон, белый текст)
 - При освещенности ≥ 50 лк: светлая тема
 - Отображайте значение в ProgressBar (0-1000 лк)
4. Добавьте кнопку ручного переключения темы.

Вариант 8: Счетчик шагов.

Порядок выполнения:

1. Зарегистрируйте `SensorEventListener` для `TYPE_ACCELEROMETER`
2. В `onSensorChanged`:
 - Фиксируйте шаг при превышении порога ускорения ($|y| > 12$)
 - Защита от двойного счета (задержка 300 мс)
3. Визуализация:
 - Отображайте количество шагов в `TextView`
 - Изменяйте цвет фона при каждом шаге
4. Добавьте кнопку сброса счетчика.

Вариант 9: Детектор наклона.

Порядок выполнения:

1. Зарегистрируйте `SensorEventListener` для `TYPE_ACCELEROMETER`
2. В `onSensorChanged`:
 - Получайте значения по осям X, Y, Z.
 - Рассчитайте угол наклона: $\text{atan2}(x, z) * 180 / \text{PI}$
3. Визуализация:
 - При угле > 45 менять цвет фона на красный.
 - Поворачивайте `ImageView` (уровень) по оси X.
 - Выводить значение угла в `TextView`.
4. Добавьте кнопку калибровки, которая установит текущее положение как «ровное» положение.

Вариант 0: Работа с датчиком давления.

Порядок выполнения:

1. Зарегистрируйте `SensorEventListener` для `TYPE_PRESSURE`.
2. В `onSensorChanged`:
 - Получайте текущее значение давления.
 - Реализуйте простой прогноз погоды в зависимости от давления.
3. Визуализация:
 - Используйте разные изображения в `ImageView` для отображения прогноза.
 - Используйте `SeekBar` для настройки порога изменения давления, при котором прогноз обновляется.
 - Выводить текущее давление в `TextView`.
4. Добавьте кнопку сброса показаний давления и прогноза.

Методические указания к выполнению лабораторной работы №2

Для выполнения лабораторной работы №2 необходимо изучить лекцию 5. Основные понятия включают использование класса `SensorManager`, интерфейса `SensorEventListener`, методов обработки событий датчика (`onSensorChanged`, `onAccuracyChanged`) и типов датчиков, доступных в системе (например, `TYPE_ACCELEROMETER`, `TYPE_LIGHT`, `TYPE_MAGNETIC_FIELD`). Предполагается, что материал предыдущих лекций уже изучен, а навыки создания мобильного приложения закреплены выполнением работы №1. Для выполнения работы также требуется установка `Android Studio`.

Откройте `Android Studio` и создайте новый проект с шаблоном `Empty Activity`. Назовите его, например, `SensorApp`, и укажите язык `Kotlin` как основной. Предварительно настройте эмулятор или подключите физическое устройство с поддержкой необходимых датчиков.

Минимальный уровень API рекомендуется установить не ниже 21 (`Android 5.0`), чтобы обеспечить совместимость с большинством датчиков. Если вы работаете с эмулятором, убедитесь, что он поддерживает нужный датчик (в настройках `AVD` можно включить/отключить датчики). Для физических устройств проверяйте наличие датчика через `sensorManager.getSensorList(Sensor.TYPE_ALL)`.

Реализацию функционала данного приложения следует проводить по шагам:

1. Для инициализации датчика в активности `MainActivity.kt` получите экземпляр класса `SensorManager`, используя метод `getSystemService(Context.SENSOR_SERVICE)`.

Затем выберите нужный датчик с помощью метода `getDefaultSensor()`, передав в него тип датчика, соответствующий вашему варианту (например, `Sensor.TYPE_GYROSCOPE` для гироскопа или `Sensor.TYPE_LIGHT` для датчика освещенности).

Например, данный код демонстрирует общий алгоритм действий:

```
class MainActivity : AppCompatActivity() {
    // Объявляем переменные для SensorManager и Sensor
    private lateinit var sensorManager: SensorManager
    private var sensor: Sensor? = null // Датчик может отсутствовать на устройстве
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        // Получаем экземпляр SensorManager через системный сервис
        sensorManager = getSystemService(Context.SENSOR_SERVICE) as SensorManager
        // Выбираем датчик в зависимости от варианта задания:
        // Например:
        // - Для гироскопа: Sensor.TYPE_GYROSCOPE
        // - Для акселерометра: Sensor.TYPE_ACCELEROMETER
        // - Для датчика освещённости: Sensor.TYPE_LIGHT
        sensor = sensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE) // Замените на ваш тип датчика
        // Проверка доступности датчика (на случай, если его нет на устройстве)
        if (sensor == null) {
            Toast.makeText(this, "Датчик не найден", Toast.LENGTH_SHORT).show()
        }
    }
}
```

2. Затем необходимо реализовать обработчик событий. Для этого активность должна реализовать интерфейс `SensorEventListener`, который содержит два метода: `onSensorChanged()` и `onAccuracyChanged()`. Метод `onSensorChanged()` будет вызываться каждый раз, когда датчик фиксирует изменение показаний. В нем вы будете получать значения осей (`event.values[0]`, `event.values[1]`, `event.values[2]`) и выполнять необходимые вычисления по варианту. Метод `onAccuracyChanged()` можно оставить без реализации, так как изменение точности датчика не требуется в данном задании.

Например, данный код показывает общую реализацию на этом шаге:

```
// Обработка данных с датчика
override fun onSensorChanged(event: SensorEvent?) {
    event?.let { sensorEvent ->
        // Получение значений по осям (X, Y, Z или специфичные для датчика)
        val axisX = sensorEvent.values[0]
        val axisY = sensorEvent.values[1]
        val axisZ = sensorEvent.values[2]

        // Логика обработки под ваш вариант:
        when (sensorEvent.sensor.type) {
            Sensor.TYPE_GYROSCOPE -> {
                // Интеграция угловой скорости для угла поворота (вариант 1)
            }
            Sensor.TYPE_ACCELEROMETER -> {
                // Расчет ускорения, угла наклона или шагов (варианты 2, 3, 8, 9)
            }
            Sensor.TYPE_LIGHT -> {
                // Изменение фона (вариант 7)
            }
            Sensor.TYPE_MAGNETIC_FIELD -> {
                // Расчет направления (вариант 6)
            }
            Sensor.TYPE_PRESSURE -> {
                // Прогноз погоды (вариант 0)
            }
            Sensor.TYPE_PROXIMITY -> {
                // Определения расстояния, времени покрытия (вариант 5)
            }
            Sensor.TYPE_GRAVITY -> {
                // Определение наклона (вариант 4)
            }
        }
        // Обновление UI (замените на ваши элементы из layout)
        findViewById<TextView>(R.id.textView).text = "X: $axisX\nY: $axisY\nZ: $axisZ"
    }
}

// onAccuracyChanged не требует обязательной реализации
override fun onAccuracyChanged(sensor: Sensor?, accuracy: Int) = Unit
}
```

Остановимся подробнее на фрагменте с обновлением UI из представленного кода. В нем `findViewById` — это метод, который находит элемент интерфейса (например, `TextView`, `Button`) по его идентификатору (ID) из XML-разметки (`activity_main.xml`). `<TextView>` — указывает тип элемента, который мы ищем (в данном случае текстовое поле). `R.id.textView` — ссылка на ID элемента,

определённый в файле `res/values/ids.xml` или автоматически сгенерированная Android Studio. Например, если в вашем XML есть:

```
<TextView
    android:id="@+id/textView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```

То `R.id.textView` будет указывать на этот элемент.

Далее, `.text = "X: $axisX\nY: $axisY\nZ: $axisZ"` означает следующее:

- `.text` — свойство `TextView`, которое устанавливает отображаемый текст.
- `"X: $axisX\nY: $axisY\nZ: $axisZ"` — строка, в которую подставляются текущие значения осей датчика (`axisX`, `axisY`, `axisZ`).
- `\n` — символ перевода строки (переводит значения на новую строку).
- `$axisX` — синтаксис Kotlin для вставки переменной в строку (интерполяция).

Таким образом, вам нужно заменить `R.id.textView` на ID элемента из вашего XML-макета.

3. Для получения данных с датчика необходимо зарегистрировать слушатель в методе `onResume()` активности, используя метод `registerListener()`. В параметрах укажите ссылку на текущую активность, выбранный датчик и частоту обновления показаний (например, `SensorManager.SENSOR_DELAY_NORMAL`). Чтобы избежать потребления ресурсов устройства, отмените регистрацию слушателя в методе `onPause()` с помощью `unregisterListener()`:

```
override fun onResume() {
    super.onResume()
    // Регистрация слушателя при активности
    sensor?.let {
        sensorManager.registerListener(
            this, // Ссылка на текущую активность (SensorEventListener)
            it, // Выбранный датчик
            SensorManager.SENSOR_DELAY_NORMAL // Частота обновления данных
        )
    }
}
override fun onPause() {
    super.onPause()
    // Отмена регистрации слушателя для экономии ресурсов
    sensorManager.unregisterListener(this)
}
```

4. На данном шаге осуществляется визуализация данных, полученных ранее в методе `onSensorChanged()` согласно требованиям вашего варианта. Результаты обработки данных выводите в интерфейс приложения. Для элементов управления, таких как кнопки сброса или калибровки, реализуйте обработчики событий в методе `setOnClickListener()`.
5. После реализации основной функциональности протестируйте приложение на эмуляторе или физическом устройстве. Убедитесь, что данные с датчика корректно считываются и отображаются в интерфейсе. Используйте класс `Log` для вывода значений в консоль `Logcat`, что поможет отладить алгоритмы

обработки данных. Проверьте, как приложение реагирует на изменение положения устройства, уровень освещенности или другие внешние факторы, в зависимости от типа датчика.

Требования к отчету

Результаты выполненной лабораторной работы должны быть оформлены в формате текстового редактора Word, размером шрифта 14 пунктов и включать:

1. Титульный лист.
2. Текст задания, соответствующий Вашему варианту.
3. Описание структуры проекта, алгоритма выполнения, используемых функций.
4. Результаты работы программы в виде скриншотов работающего приложения.
5. Программный код с комментариями.
6. Ссылки на источники внешней информации, которые были использованы при выполнении заданий.

Объем пояснительной записки должен включать в себя не менее 5 страниц поясняющего текста (не считая исходного кода программ).

Список дополнительных источников

1. Фримен Э. Паттерны проектирования : [пер. с англ.] / Элизабет Фримен, Эрик Фримен ; пер. с англ. — Санкт-Петербург : Питер, 2018. — 656 с.
2. Количниченко Д.Н. Программирование для Android. 3-е изд. — Издательство БХВ, 2021 г. — 288 с.