МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

ГЛАВНОЕ УПРАВЛЕНИЕ ПО ОБРАЗОВАНИЮ

МОГИЛЕВСКОГО ОБЛАСТНОГО ИСПОЛНИТЕЛЬНОГО КОМИТЕТА

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«МОГИЛЕВСКИЙ ГОСУДАРСТВЕННЫЙ ПОЛИТЕХНИЧЕСКИЙ КОЛЛЕДЖ»

**ПРОГРАММИРОВАНИЕ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ**

ДОМАШНЯЯ КОНТРОЛЬНАЯ РАБОТА

Выполнил Учащийся группы ПО-455

О.И. Любаль

Шифр 14

2023

# 11 Перечислите и опишите основные задачи менеджера виртуальных устройств

AVD расшифровывается как Android Virtual Device, а именно виртуальное Андроид устройство. На вашем компьютере вы открываете заранее выбранную и сконфигурированную систему. Это удобно даже когда у вас есть реальный телефон. В эмуляторе можно изменить версию ОС, размер экрана и другие параметры. Это позволяет протестировать то, как работает ваше приложение на разных версиях Android.

В Android Studio всеми виртуальными устройствами управляет AVD manager. Его можно открыть через меню Tools в Android studio, рисунок 1.

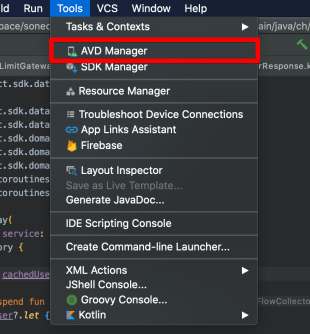


Рисунок 1 Менеджер виртуальных устройств.

Для создания нового Андроид эмулятора в AVD Manager нужно нажать на кнопку создания нового виртуального устройства и выбрать предустановленный шаблон или же создать свой новый. Выберем Pixel XL, рисунок 2.

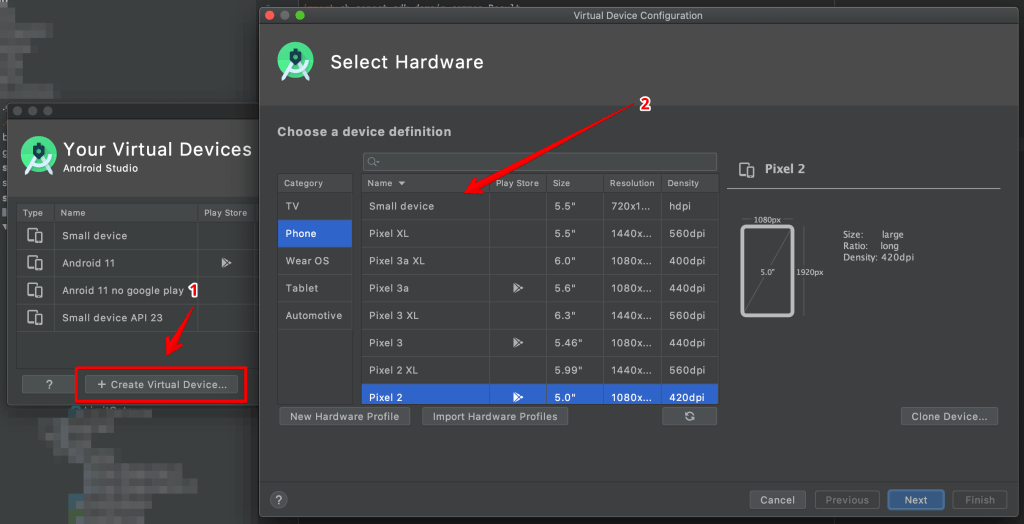


Рисунок 2. Создание устройства, выбор типа.

Следующим пунктом необходимо выбрать образ операционной системы. Это важный шаг, т.к. именно от него будет зависеть, какая версия Android будет установлена на новом виртуальном устройстве, рисунок 3.

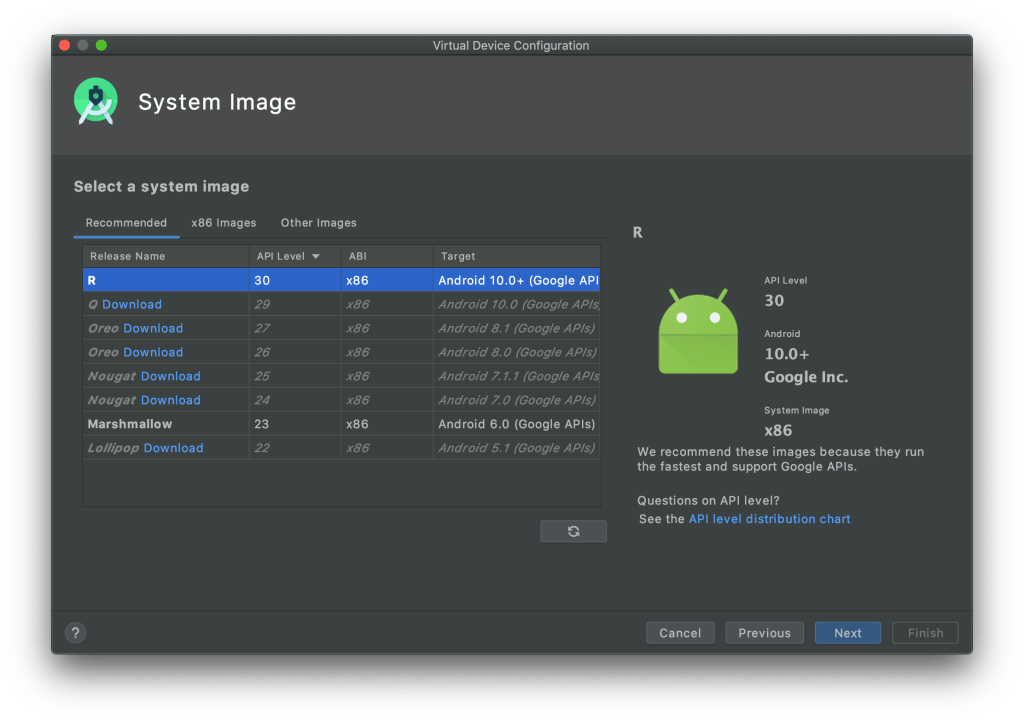


Рисунок 3. Выбор ОС.

Когда все будет готово останется лишь запустить созданный эмулятор Android устройства. Делается это в уже знакомом AVD Manager. Также в нем можно найти возможные настройки и пункты управления девайсом, рисунок 4.



Рисунок 4. Управление устройствами

Вот список возможных опций и что они означают:

| **Название опции** | **Что она означает** |
| --- | --- |
| Duplicate | Дублировать уже созданный эмулятор |
| Wipe Data | Очистить данные устройства. Полезно, если нужен «чистый» девайса с теми же настройками |
| Cold Boot Now | Холодный старт. Бывает, что эмулятор, например, завис. Это способ его перезагрузить |
| Show On Disk | Показать папку, в которой хранятся все файлы, относящиеся к эмулятору |
| View Details | Посмотреть все настройки и конфигурации устройства |
| Delete | Удалить AVD |
| Stop | Остановить уже запущенный эмулятор |

Таким образом, можно разрабатывать под Android не имея реального устройства на руках и запускать приложения для теста на виртуальном устройстве.

# 25 Перечислите компоненты отображения времени. Опишите назначение, свойства, возможности использования

В Java даты и время хранятся несколькими способами, в зависимости от требований.

Длинный тип – это примитивный тип данных, способный хранить количество миллисекунд, прошедших с определенного момента времени (время Unix).

Класс Date (java.util.Date) – это служебный класс для хранения даты и времени способом, которым можно разумно манипулировать без необходимости постоянно думать о времени в миллисекундах.

Класс Calendar (java.util.Calendar) – это служебный класс для работы с различными календарями, а также для манипулирования информацией о дате и времени различными способами.

Класс GregorianCalendar (подкласс java.util.Calendar) используется главным образом для манипулирования датами в западном полушарии, где мы используем 12-месячный календарь, с 7 днями до недели и двумя эпохами (BC и AD).

Определение текущей даты и времени

Существует несколько способов определения текущего времени на устройстве Android.

Можно определить необработанные данные даты и времени, используя статический метод, предоставленный в классе System (java.lang.System):

long msTime = System.currentTimeMillis ();

Date curDateTime = новая дата (msTime);

Этот метод основан на времени, когда устройство *думает,* что это так, что может быть или не быть надежным. Другой способ определения текущей даты и времени использует конструктор по умолчанию для класса Date, который создает объект Date с текущей датой и временем:

Date anotherCurDate = new Date ();

Есть и другие способы определения истинного точного времени – компьютеры часто проверяют известные серверы учета времени, чтобы убедиться, что все «синхронизированы».

Устройства Android, которые подключаются к сотовым службам, как правило, имеют точную информацию о дате и времени, так как они могут использоваться для связи с радиомачтами.

Создание даты и времени с нуля

А теперь, что если нужно создать переменную даты или времени с определенной даты, например дня рождения. Для этого нужен класс Calendar:

GregorianCalendar bday = new GregorianCalendar (1977, Calendar.APRIL, 12);

Можно использовать объекты календаря для манипулирования датами и извлечения интересной информации о них. Например, можно использовать метод get (), чтобы определить день недели, дня рождения:

int dayOfWeek = bday.get (Calendar.DAY\_OF\_WEEK); // Возвращает 3 за вторник!

Месяцы календаря основаны на 0, поэтому январь – это месяц 0, а не месяц 1, и, следовательно, апрель – месяц 3, а не 4. Наконец, можно извлечь объект Date из конфигурации даты календаря, используя getTime () метод.

Форматирование даты и времени

Для этого необходимо использовать вспомогательный класс DateFormat (java.text.DateFormat) для преобразования необработанной информации о дате и времени в различные строковые форматы (для разных локалей и т. Д.) Или для анализа строковой информации в соответствующие биты даты и времени для использования в коде. ,

Например, можно использовать класс SimpleDateFormat (подкласс класса DateFormat) для создания пользовательской строки, содержащей информацию о дате и времени. Во-первых, необходимо указать строку формата, используя соответствующие коды для разных битов информации. Затем применим эту информацию к определенной дате или времени. Например:

Date anotherCurDate = new Date ();

SimpleDateFormat formatter = new SimpleDateFormat («EEEE, MMMM d» в «чч: мм а» в году «гггг G»);

String formattedDateString = formatter.format (anotherCurDate);

Если печтатать строку под названием formattedDateString, можно увидеть что-то вроде: «Понедельник, 6 декабря, в 8:34 утра 2010 года нашей эры».

Кстати, также можно использовать классы DateFormat и SimpleDateFormat для разбора строк даты и времени в соответствующий класс Date.

Отображение даты и времени для пользователя

Существует множество способов отображения информации о дате и времени для пользователя. То, как их выбрать, зависит от требований.

Возможно, самый простой способ – просто отобразить определенную дату или время в виде правильно отформатированной строки. Этот метод наиболее удобен для фиксированных дат или времени, таких как дни рождения или события. Просто используем служебный класс DateFormat, чтобы создать соответствующую строку и отобразить ее на экране, как и любой другой текст (например, в элементе управления TextView).

Чтобы показать время, проходящее «в реальном времени», вы можно использовать элементы управления AnalogClock (см. рисунок 5) или DigitalClock.



Рисунок 5. Пример отображения времени.

Для выбора времени можно использовать **TimePicker.**

**TimePicker** -- виджет выбора времени, который способен отображать время как в 24-часовом, так и в 12-часовом формате. Рассмотрим его на практике.

В первую очередь, следует отметить один из атрибутов TimePicker -- **timePickerMode**. Он дает возможность менять режим отображения и принимает одно из 2-х значений:

**clock** (результат работы отображается в виде часов);

**spinner** (отображение происходит в виде спиннера).

Методы TimePicker

Что можно выделить среди методов:

int getHour(): для возвращения часов в 24-часовом формате;

int getMinute(): для возвращения минут;

boolean is24HourView(): для возвращения true, когда применяется 24-часовой формат;

void setHour(int hour): производит установку часов для TimePicker;

void setIs24HourView(Boolean is24HourView): производит установку 24-часового формата;

void setMinute(int minute): для установки минут;

void setOnTimeChangedListener(TimePicker.OnTimeChangedListener onTimeChangedListener): для установки слушателя изменения времени для объекта **TimePicker.OnTimeChangedListener**. При этом по умолчанию TimePicker отображается в режиме часов, то есть "clock" (рисунок 6). Но есть и второй режим -- "spinner" (рисунок 7)

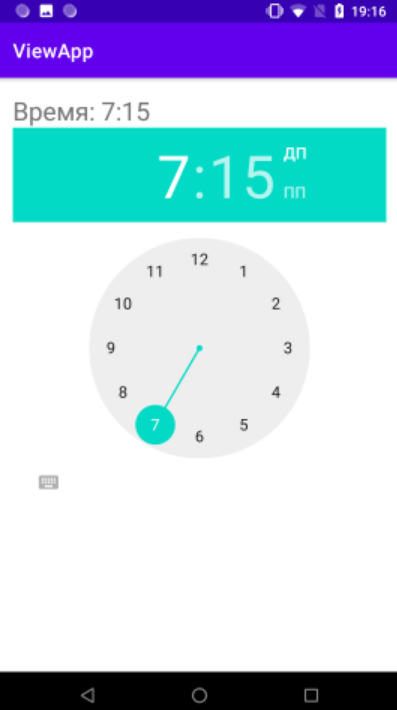


Рисунок 6. TimePicker отображается в режиме часов.

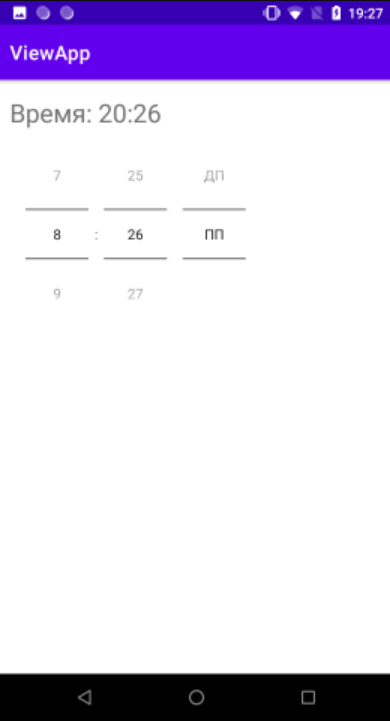


Рисунок 7. TimePicker отображается в режиме Spinner.

# 51 Опишите жизненный цикл приемников широковещательных намерений

**Вещательные приемники** просто отвечают на широковещательные сообщения из других приложений или из самой системы. Эти сообщения иногда называют событиями или намерениями. Например, приложения могут также инициировать широковещательные рассылки, чтобы сообщить другим приложениям, что некоторые данные были загружены на устройство и доступны для их использования, поэтому этот широковещательный приемник будет перехватывать эту связь и инициировать соответствующие действия.

Существует два важных шага, чтобы заставить BroadcastReceiver работать для систем, предназначенных для вещания.

* Создание приемника вещания.
* Регистрация приемника вещания

Есть один дополнительный шаг, если вы необходимо реализовать свои собственные намерения, тогда вам придется создавать и передавать эти намерения.

## Создание приемника вещания

Приемник широковещания реализован как подкласс класса **BroadcastReceiver** и переопределяет метод onReceive (), где каждое сообщение принимается в качестве параметра объекта **Intent** .

открытый класс MyReceiver extends BroadcastReceiver {

@Override

public void onReceive (Контекст контекста, Намерение намерения) {

Toast.makeText (контекст, «Обнаружено намерение.», Toast.LENGTH\_LONG) .show ();

}

}

## Регистрация приемника вещания

Приложение прослушивает определенные намерения вещания, регистрируя получателя вещания в файле *AndroidManifest.xml* . Предположим, что мы собираемся зарегистрировать *MyReceiver* для сгенерированного системой события ACTION\_BOOT\_COMPLETED, которое запускается системой, когда система Android завершила процесс загрузки.

#### Broadcast-приемник

<application

Android: icon = "@ extract / ic\_launcher"

android: label = "@ string / app\_name"

android: theme = "@ style / AppTheme">

<reciever android: name = "MyReceiver">

<intent filter>

<action android: name = "android.intent.action.BOOT\_COMPLETED">

</ Действие>

</ intent filter>

</ Reciever>

</ application>

Теперь, когда ваше Android-устройство загружается, оно будет перехвачено BroadcastReceiver *MyReceiver,* и будет реализована реализованная логика внутри *onReceive ()* .

Существует несколько сгенерированных системой событий, определенных как окончательные статические поля в классе **Intent** . В следующей таблице перечислены несколько важных системных событий.

**android.intent.action.BATTERY\_CHANGED**

Липкая трансляция, содержащая состояние зарядки, уровень и другую информацию о батарее.

**android.intent.action.BATTERY\_LOW**

Указывает на низкий уровень заряда батареи на устройстве.

**android.intent.action.BATTERY\_OKAY**

Указывает на то, что батарея в порядке после низкого уровня заряда.

**android.intent.action.BOOT\_COMPLETED**

Это транслируется один раз после завершения загрузки системы.

**android.intent.action.BUG\_REPORT**

Показать активность для сообщения об ошибке.

**android.intent.action.CALL**

Выполните звонок кому-либо, указанному в данных.

**android.intent.action.CALL\_BUTTON**

Пользователь нажал кнопку «вызов», чтобы перейти на номеронабиратель или другой соответствующий пользовательский интерфейс для совершения вызова.

**android.intent.action.DATE\_CHANGED**

Дата изменилась.

**android.intent.action.REBOOT**

Перезагрузите устройство.

# 67 Охарактеризуйте класс Camera, перечислите и опишите основные методы класса, укажите их назначение

Все операции с камерой мобильного устройства проводятся с помощью класса *Camera*. Необходимо завести переменную

Camera camera;

и инициализировать ее

camera = Camera.open();

После завершения работы с камерой необходимо сделать

camera.release();

в противном случае камера останется заблокированной и недоступной для других приложений.Для обычных приложений типа фотокамеры инициализацию лучше всего производить в *onResume*, а освобождение в *onPause*.  
Обязательным условием при работе с камерой является создание окна предпросмотра (preview). Это окно должно являться объектом класса *Surface*и для отображения на экране подходит *SurfaceView*.  
Объявим

SurfaceView preview;

Чтобы задать preview, необходимо вызвать метод *setPreviewDisplay*, параметром которого является объект класса *SurfaceHolder*.

SurfaceHolder surfaceHolder;

surfaceHolder = preview.getHolder();

camera.setPreviewDisplay(surfaceHolder);

Чтобы включить отображение preview, вызываем

camera.startPreview();

Если этого не сделать, то камера не сможет делать снимки.  
Собственно для того, чтобы сделать снимок, необходимо вызвать метод

void takePicture(Camera.ShutterCalback shutter, Camera.PictureCallback raw, Camera.PictureCallback postview, Camera.PictureCallback jpg);

С помощью параметров (кстати, любой из них может быть null) задаются обработчики разных событий:

* shutter — вызывается в момент получения изображения с матрицы
* raw — программе передаются для обработки raw данные (если поддерживается аппаратно)
* postview — программе передаются полностью обработанные данные (если поддерживается аппаратно)
* jpg — программе передается изображение в виде jpg. Здесь можно организовать запись изображения на карту памяти.

Вызов *takePicture* можно поместить непосредственно в обработчик *onClick* кнопки — в этом случае фотографирование произойдет сразу после нажатия на нее, но можно и воспользоваться предварительной автофокусировкой.

В этом случае задается обработчик *Camera.AutoFocusCallback*, в котором необходимо реализовать метод:

public void onAutoFocus(boolean paramBoolean, Camera paramCamera);

Тогда после вызова в обработчике нажатия на кнопку *camera.autoFocus()*, однократно будет вызван обработчик, в котором мы уже и примем решение об удачной фокусировке и необходимости сделать снимок.

Для работы с SurfaceHolder можно задать *SurfaceHolder.Callbac*k

surfaceHolder.addCallback();

В этом случае необходимо реализовать методы

public void surfaceCreated(SurfaceHolder holder);

public void surfaceChanged(SurfaceHolder holder, int format, int width, int height);

public void surfaceDestroyed(SurfaceHolder holder);

C помощью них приложению будет сообщаться о том, что Surface успешно создано, если оно изменено или то, что оно удалено.

Размер нашего preview можно менять в процессе выполнения программы:

LayoutParams lp = preview.getLayoutParams();

lp.width = задаваемая ширина;

lp.height = задаваемая высота;

preview.setLayoutParams(lp);

Для приложения камеры удобнее всего сразу задать расположение экрана как

setRequestedOrientation(ActivityInfo.SCREEN\_ORIENTATION\_LANDSCAPE);

В противном случае нам придется, например, в *surfaceCreated* проверять расположение экрана и поворачивать preview с помощью, например, camera.setDisplayOrientation(0).

Это не очень удобно, потому что поворот экрана занимает какое-то время. В этот момент происходит вызов onPause и onResume, пересоздается Surface.

Также имеется возможность объявить обработчик *Camera.PreviewCallback*, с помощью которого путем реализации метода

void onPreviewFrame(byte[] paramArrayOfByte, Camera paramCamera);

можно получать и обрабатывать каждый кадр, отображаемый в preview.

И последний важный момент. Чаще всего получается так, что отношение сторон SurfaceView отличается от отношения сторон в preview камеры. Поэтому для того, чтобы избежать искажений изображения на экране, необходимо подкорректировать размер отображаемого окна предпросмотра.  
В манифест необходимо добавить permission

<uses-permission android:name="android.permission.CAMERA" />

# Список использованных источников

1. Васильева, И.Н. Криптографические методы защиты информации :учеб. и практикум / И.Н. Васильева. М. :Юрайт, 2020. 349 с.
2. Малюк, А.А. Информационная безопасность: концептуальные и методологические основы защиты информации : учеб. пособие / А.А. Малюк. М. : Горячая линия-Телеком, 2015. 280 с.
3. Безбогов, А.А. Методы и средства защиты компьютерной информации / А.А. Безбогов, А.В. Яковлев, В.Н. Шамкин. – Тамбов, 2006.
4. Математические и компьютерные основы криптологии: учеб. пособие / Ю.С. Харин [и др.]. – Минск, 2003.
5. Петров, А.А. Компьютерная безопасность. Криптографические методы защиты / А.А.Петров. – М., 2000.
6. Тимошенко, А.А. Защита информации в специализированнных информационно-телекоммуникационных системах / А.А.Тимошенко. –Киев, 2010.