|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение высшего образования **«МИРЭА  Российский технологический университет»**  **РТУ МИРЭА** |

**Институт информационных технологий (ИИТ)**

**Кафедра математического обеспечения и стандартизации ИТ**

**ОТЧЁТ ПО ПРАКТИЧЕСКОМУ ЗАНЯТИЮ №12**

**по дисциплине**

**«Разработка мобильных приложений»**

|  |  |  |  |
| --- | --- | --- | --- |
| Отчет представлен к  рассмотрению:  Студенты группы ИНБО-04-20 | « » февраля 2022 г. | (подпись) | Ло В.Х. |
|  |  |  |  |
| Преподаватель | « » 2022 г. | (подпись) | Фандеев И.И. |

Москва, 2022г.

**СОДЕРЖАНИЕ**

[ОТЧЕТ ВЫПОЛНЕНИЯ ПРАКТИЧЕСКОЙ РАБОТЫ. 3](#_Toc100742894)

[1. Эффективная работа с изображениями 3](#_Toc100742895)

[1.1 Определить тип и разрешение изображения 3](#_Toc100742896)

[1.2 Обеспечить загрузку уменьшенной версии в память 4](#_Toc100742897)

[2 Обработка изображений в отдельном потоке 6](#_Toc100742898)

[2.1 Реализовать использование AsyncTask 6](#_Toc100742899)

[2.2 Реализовать управление параллельной работой 7](#_Toc100742900)

[3. Кэширование изображений 8](#_Toc100742901)

[3.1 Обеспечить использование кэш-памяти 9](#_Toc100742902)

[3.2 Реализовать использование дискового кэша 11](#_Toc100742903)

[3.3 Реализовать обработку изменений конфигурации 13](#_Toc100742904)

[4. Управление памятью при работе с изображениями 14](#_Toc100742905)

[4.1 Реализовать управление памятью 14](#_Toc100742906)

[4.2 Реализовать сохранение изображений для повторного использования 14](#_Toc100742907)

[4.3 Реализовать использование существующего изображения 15](#_Toc100742908)

[5. Показываем изображение в интерфейсе 16](#_Toc100742909)

[5.1 Реализовать загрузку изображений в компонент ViewPager 16](#_Toc100742910)

[5.2 Реализовать загрузку изображений в компонент GridView 19](#_Toc100742911)

[ВЫВОД 22](#_Toc100742913)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 22](#_Toc100742914)

# ОТЧЕТ ВЫПОЛНЕНИЯ ПРАКТИЧЕСКОЙ РАБОТЫ.

**1.** Эффективная работа с изображениями

Изображения имеют разные формы и размеры. В большинстве случаев они больше, чем требует типовой пользовательский интерфейс. Например, приложение Галерея показывает фотографии, сделанные пользователем на встроенную камеру, которые обычно имеют разрешение выше, чем разрешение экрана устройства.

Учитывая, что объем памяти ограничен, в идеале нужно загружать в память изображения с меньшим разрешением. Такие версии изображений должны подстраиваться под размер компонента, в котором они отображаются. Картинки с большим разрешением не дают видимых преимуществ, но занимают драгоценную память и требуют лишней производительности для вычислений на лету.

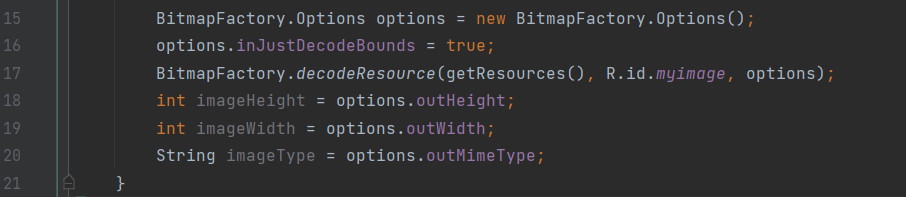
**1.1** Определить тип и разрешение изображения

Класс BitmapFactory предоставляет несколько методов декодирования (decodeByteArray(), decodeFile(), decodeResource(), и.т.д) для создания объекта типа Bitmap из различных ресурсов.

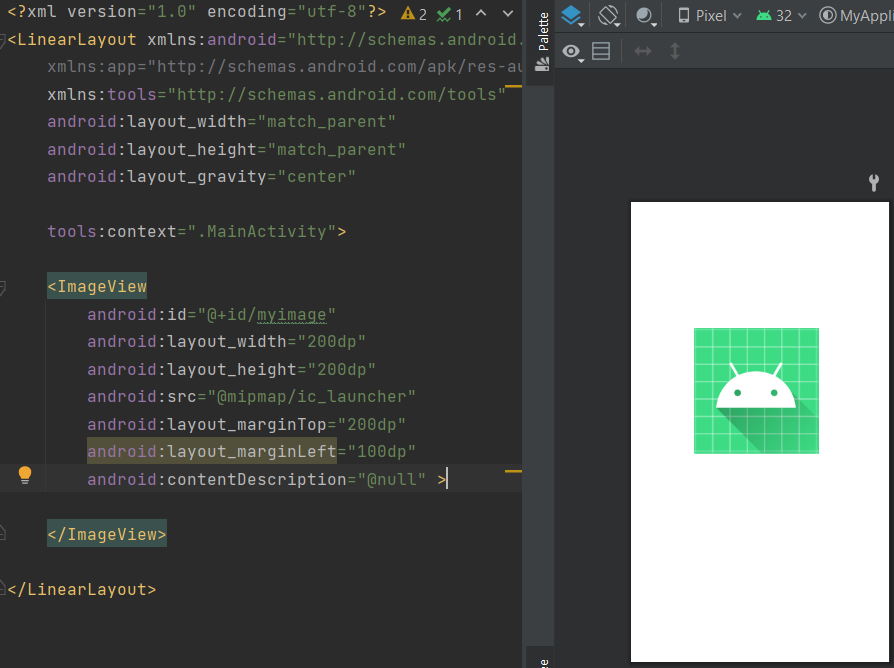
Выберите соответствующий метод, основываясь на вашем источнике данных. Методы пытаются выделить память для построения растрового изображения и запросто могут вызвать исключение OutOfMemory.

Каждый метод имеет дополнительные сигнатуры, которые позволяют задать параметры декодирования с помощью класса BitmapFactory.Options. Установите свойство inJustDecodeBounds в значение true.

Тогда память не будет выделяться, а объект будет равен null, но будут установлены свойства outWidth, outHeight и outMimeType. Это позволит узнать разрешение и тип графических данных до создания объекта Bitmap (и соответственно до выделения памяти).



**Рисунок 1 –** Тип и разрешение изображения

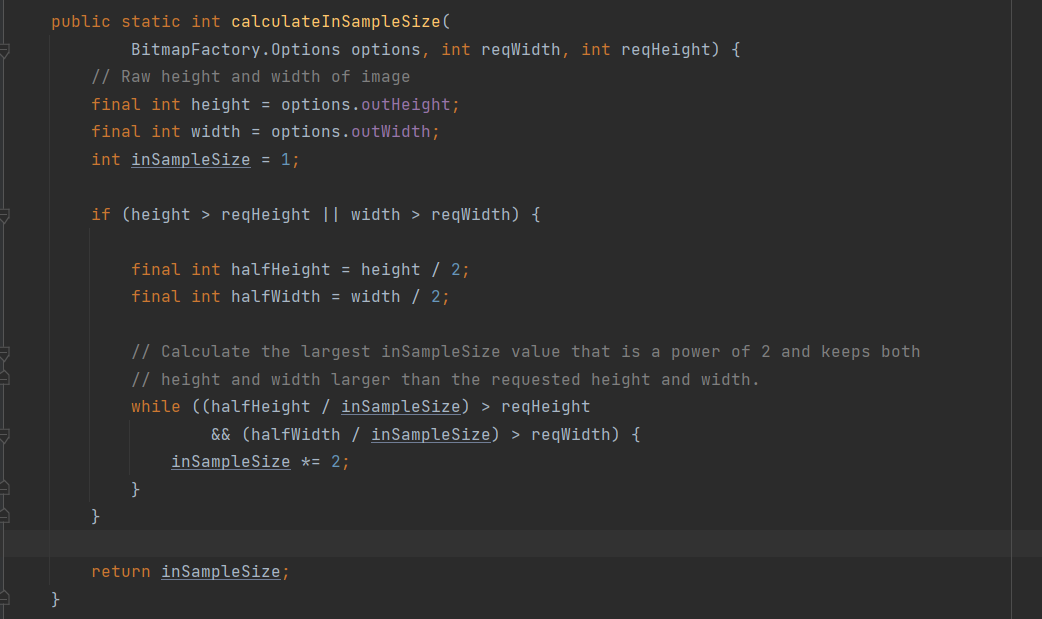
****

**Рисунок 2 – XML-файл**

**1.2 Обеспечить загрузку уменьшенной версии в память**

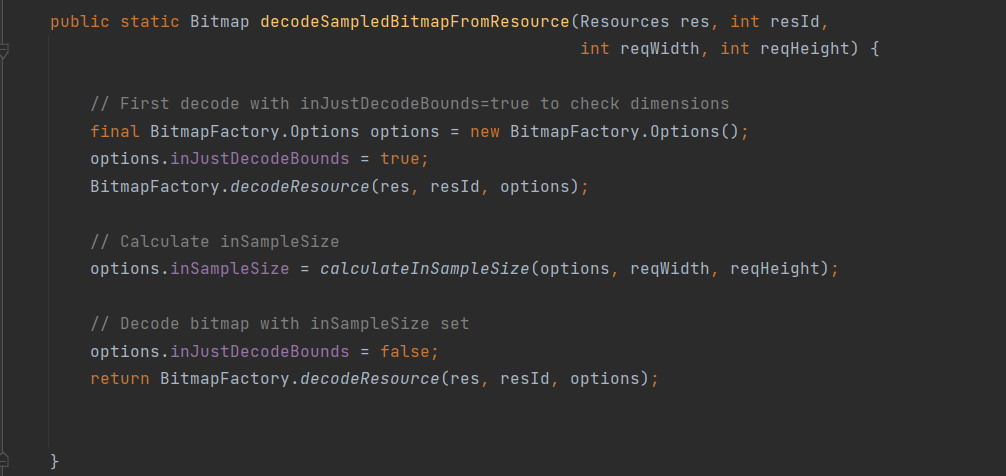
Теперь разрешение картинки известно и может быть использовано для принятия решения: загрузить в память полноразмерную версию картинки или ее уменьшенную версию. Вот некоторые факторы, которые нужно учитывать:

* Расчетное потребление памяти при загрузке полной версии.
* Количество памяти, которое вы готовы потратить на изображение, учитывая общее потребление памяти приложением.
* Разрешение компонента ImageView, который будет отображать загруженные данные.
* Размер и плотность точек экрана на текущем устройстве.

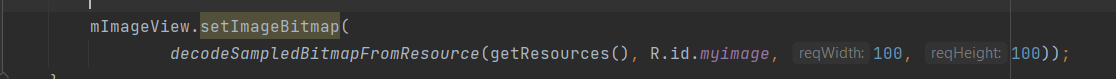
****

**Рисунок 3 – Метод расчета размера образца**

Чтобы использовать описанный метод, сначала вызываем декодер с опцией inJustDecodeBounds равной true для получения параметров изображения. Затем повторно вызываем декодер, используя новое значение inSampleSize и установив inJustDecodeBounds в значение false:

****

**Рисунок 4 – inSampleSize и установив inJustDecodeBounds**

****

**Рисунок 5 – ImageView размером 100x100p**

**2** Обработка изображений в отдельном потоке

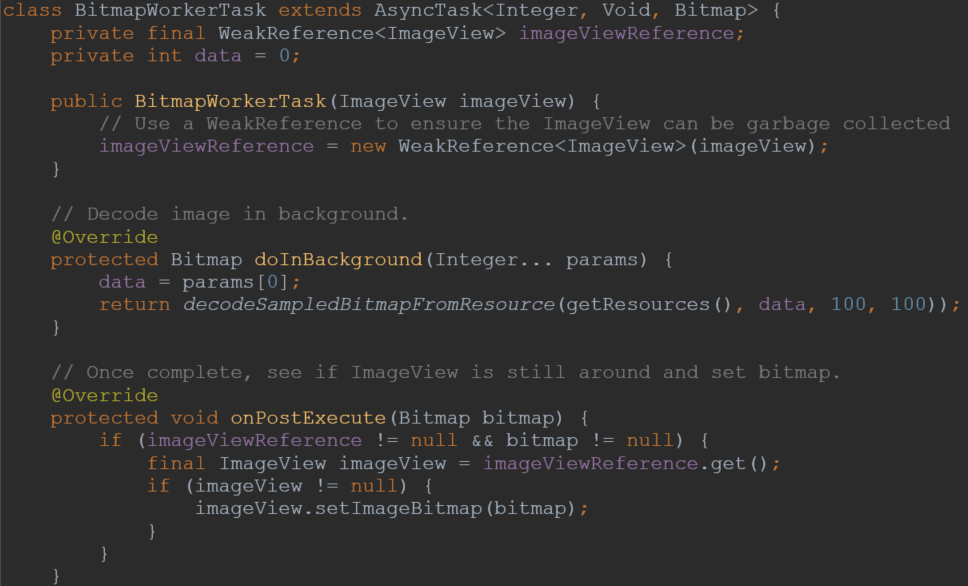
Методы BitmapFactory.decode\*, которые мы обсудили в предыдущем материале, не должны исполняться в основном потоке пользовательского интерфейса, если данные читаются с диска или загружаются из сети (или из любых других источников, кроме памяти).

Нельзя предугадать сколько времени уйдет на загрузку данных, это зависит от различных факторов (скорость чтения с диска, скорость загрузки по сети, размер изображения, мощность процессора, и.т.д).

Если такая задача заблокирует поток пользовательского интерфейса, система отметит ваше приложение как зависшее, и пользователь может закрыть его (читайте подробности в разделе Сохранение отзывчивости приложения).

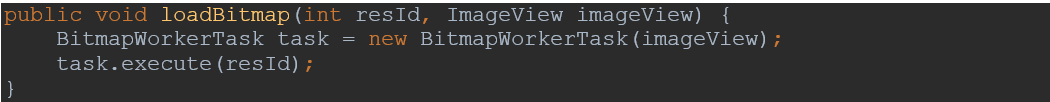
2.1 Реализовать использование AsyncTask

Класс AsyncTask позволяет легко запустить код в отдельном потоке и возвращать результат работы в основной поток пользовательского интерфейса.



**Рисунок 6 – ImageView с использованием AsyncTask и метода decodeSampledBitmapFromResource()**

Чтобы запустить асинхронную загрузку изображения, просто создайте новую задачу и выполните ее:



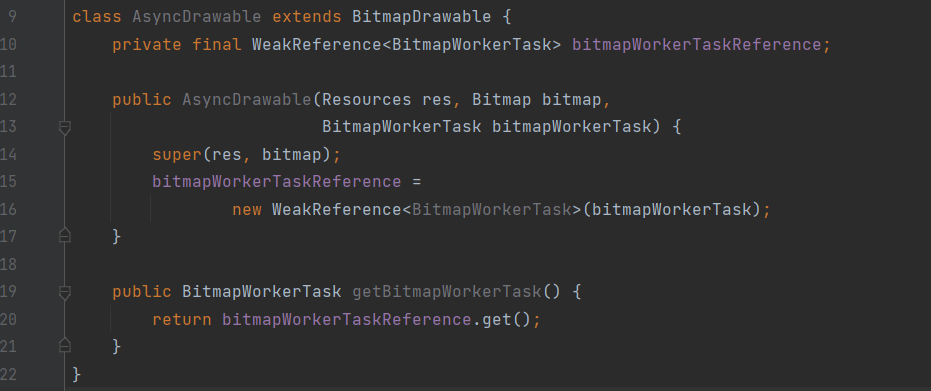
**Рисунок 7 – Запустить асинхронную загрузку изображения**

2.2 Реализовать управление параллельной работой

Чтобы эффективно работать с памятью, эти компоненты удаляют дочерние элементы, вроде полос прокрутки.

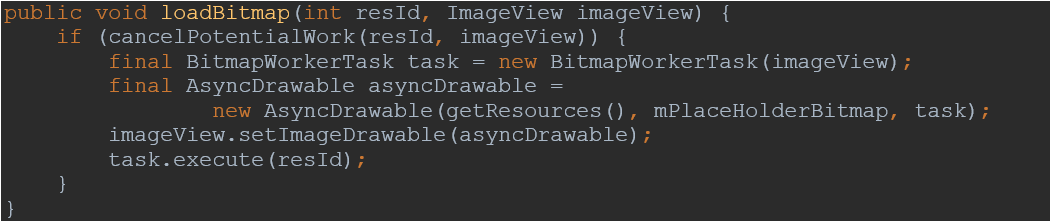
Если каждый дочерний элемент вызывает AsyncTask, нет никакой гарантии, что когда задача завершится, связанный с ней элемент еще не будет удален, чтобы использоваться в другом компоненте.

Более того, нет никакой гарантии, что асинхронные задачи завершаются в таком же порядке, в каком вы их запускаете.



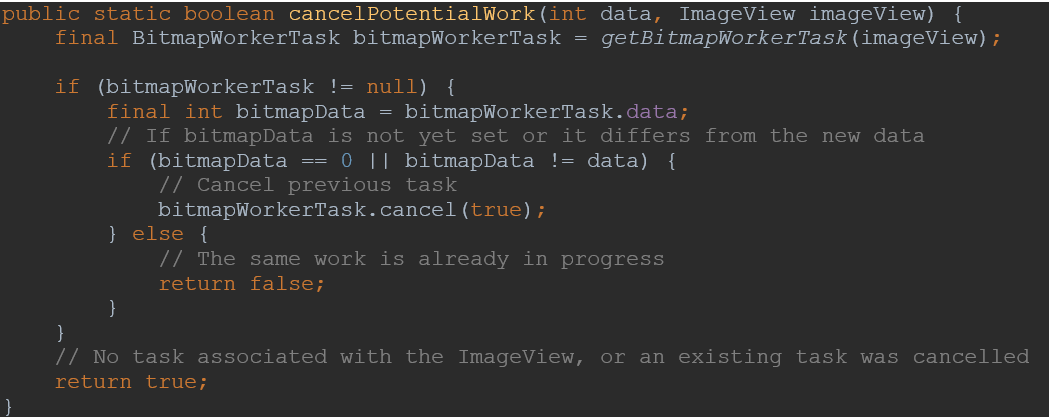
**Рисунок 8 – Подкласс класса Drawable, для хранения ссылки на рабочее задание**

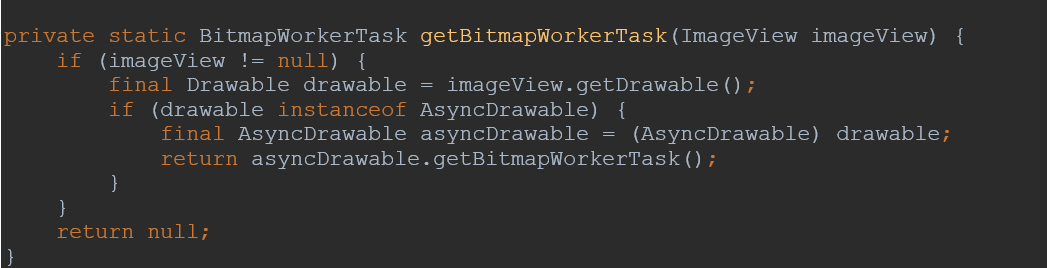
Прежде чем выполнить BitmapWorkerTask, создадим объект AsyncDrawable и свяжем его с ImageView:Прежде чем выполнить BitmapWorkerTask, создадим объект AsyncDrawable и свяжем его с ImageView:



**Рисунок 9 – Объект AsyncDrawable и свяжем его с ImageView**

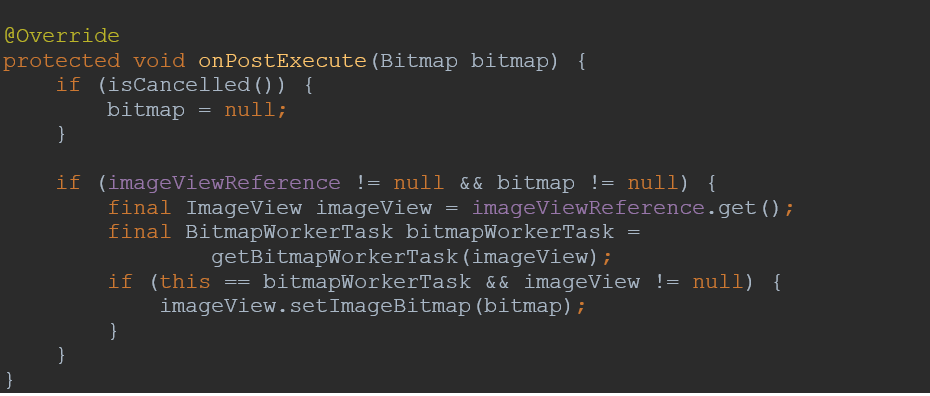
В редких случаях данные новой задачи соответствуют данным существующих задач.

 **Рисунок 10 – МетодаcancelPotentialWork**



**Рисунок 11 – Метод-помощник getBitmapWorkerTask()**

Последним шагом обновим метод onPostExecute() в классе BitmapWorkerTask. Он будет проверять отменена ли задача и ассоциирована ли текущая задача с ImageView:



**Рисунок 12 – Метод onPostExecute() в классе BitmapWorkerTask**

3. Кэширование изображений

Загрузить одну картинку в компонент пользовательского интерфейса просто, однако, все становится сложнее, когда вам нужно загрузить большой набор картинок за раз. Зачастую (в таких компонентах, как RecyclerView, GridView или ViewPager) даже требуются полосы прокрутки, потому что картинок может быть бесконечное множество.

Использование памяти сокращается, если компоненты удаляют дочерние элементы, как только они скрываются с экрана. Сборщик мусора также освобождает загруженные изображения,предполагая, что у вас нет каких-либо долгоживущих ссылок.

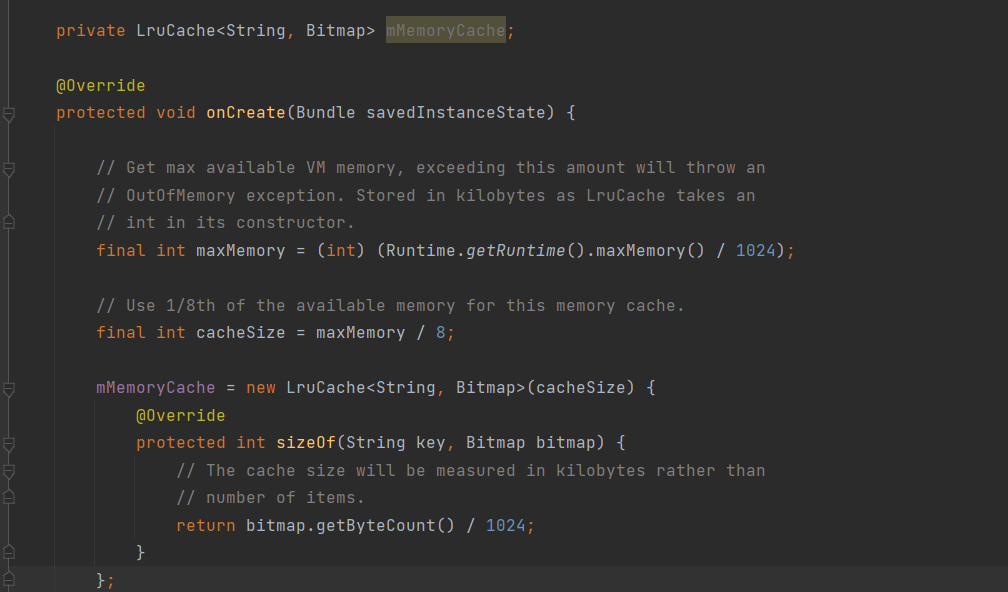
Это все хорошо и прекрасно, но чтобы интерфейс работал быстро и плавно, вы наверняка захотите избежать постоянной обработки этих изображений, когда они возвращаются на экран.

Память и дисковый кэш часто выручают в таких ситуациях, позволяя компонентам быстро подгружать обработанные изображения.

3.1 Обеспечить использование кэш-памяти

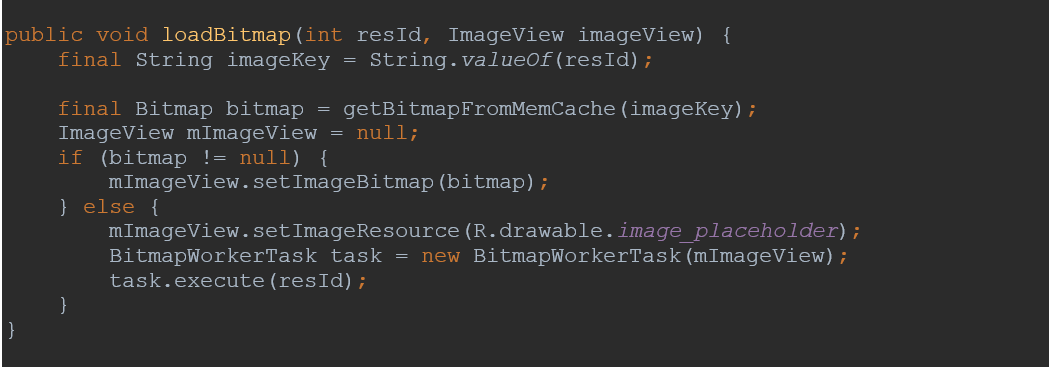
Кэш-память предлагает быстрый доступ к изображениям за счет использования ценной для приложения памяти.

Класс LruCache (также доступный в библиотеке поддержки) особенно хорошо подходит для задачи кэширования изображений, сохраняя недавние объекты в сильных ссылках LinkedHashMap и выталкивая объекты, которые в последнее время использовались редко до того, как кэш превысит заданный размер.

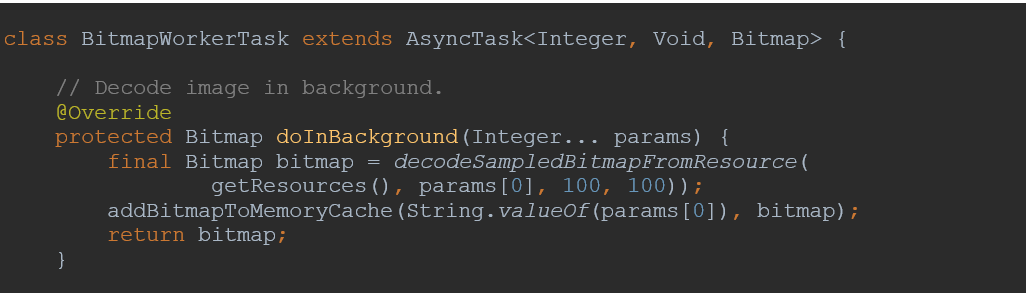


**Рисунок 13 – Использование класса LruCache для изображений**

Если запись найдена, то она немедленно используется для обновления ImageView, иначе начнется обработка изображения в отдельном потоке:



**Рисунок 14 – Начнется обработка изображения**



**Рисунок 15 – Добавлять записи в кэш-память**

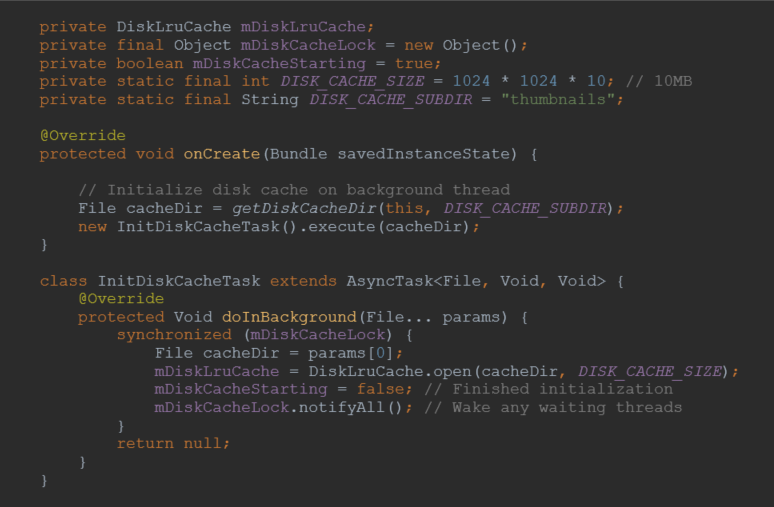
3.2 Реализовать использование дискового кэша

Кэш в памяти полезен для мгновенного доступа к недавно просмотренным изображениям, однако вы не можете полагаться на то, что изображения могут быть доступны в кэше.

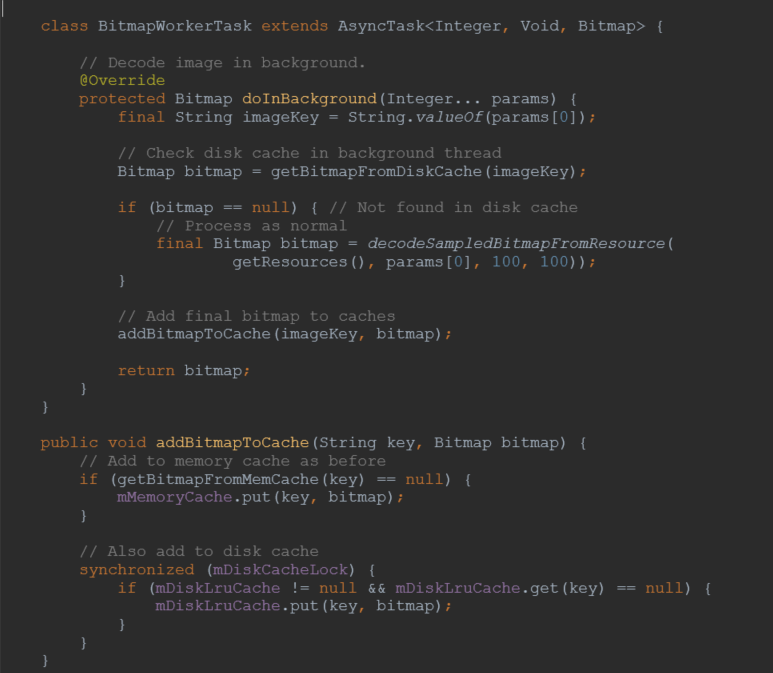
Компоненты вроде GridView с большим набором данных обычно забивают кэш-память.

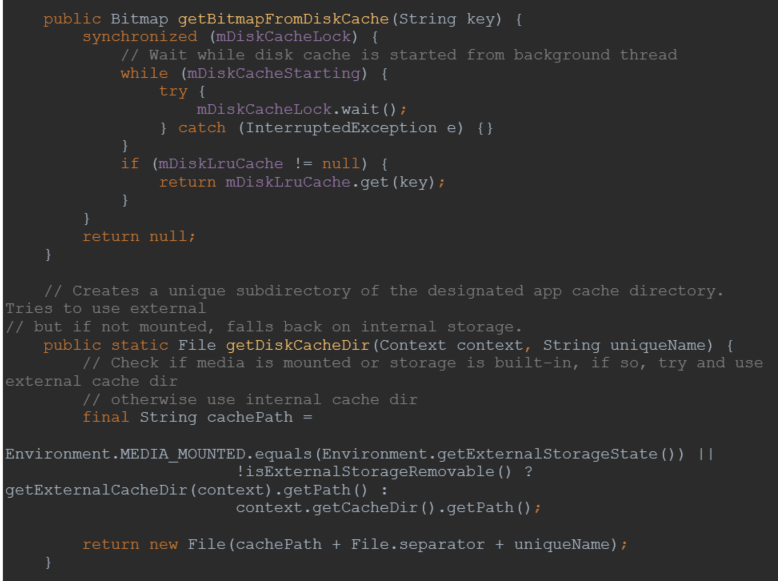
Выполнение ваше приложения может быть прервано, например входящим звонком, и пока оно остановлено, кэш-память может быть уничтожена.

Дисковый кэш может использоваться в данном случае для хранения обработанных изображений и помочь уменьшить время загрузки изображений, которые уже недоступны в кэш-памяти.



**Рисунок 16.1 – Дополнение к существующей кэш-памяти**

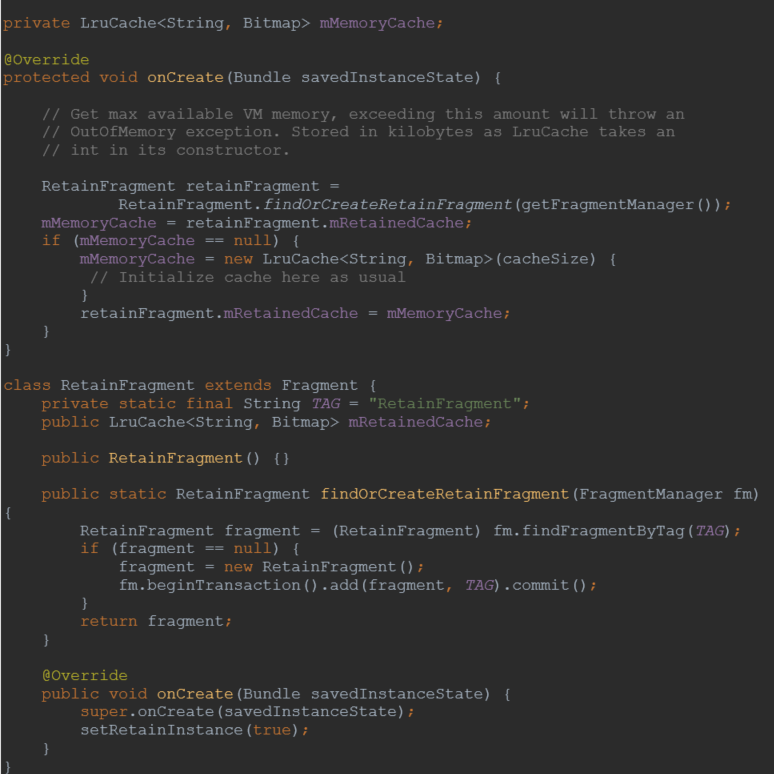




**Рисунок 16.2 – Дополнение к существующей кэш-памяти**

**3.3 Реализовать обработку изменений конфигурации**

Кэш может передаваться новому экземпляру явления при использовании фрагментов, которые сохраняются при вызове метода setRetainInstance(true). После пересоздания явления, сохраненный фрагмент присоединяется, и вы получаете доступ к существующим объектам в кэше. Это позволяет быстро перебрать изображения и поместить их в соответствующие компоненты ImageView.



**Рисунок 17 – Объекта LruCache при изменении конфигурации, с использованием фрагментов**

**4. Управление памятью при работе с изображениями**

4.1 Реализовать управление памятью

В Android 3.0(API 11) появилась опция BitmapFactory.Options.inBitmap. Если опция включена, методы декодирования попытаются использовать существующий объект при загрузке данных.Это означает, что память с загруженными изображениями используется повторно, что в результате повышает производительность и убирает лишние операции выделения и освобождения памяти.

4.2 Реализовать сохранение изображений для повторного использования

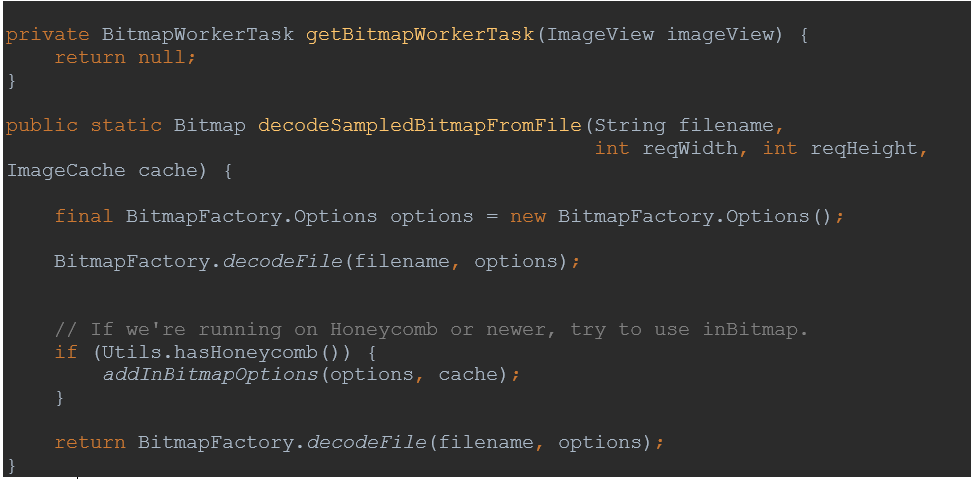
При запуске на Android 3.0 и выше и использовании LruCache, мягкие ссылки на изображение размещены в объекте типа HashSet, чтобы использовать его совместно с inBitmap:



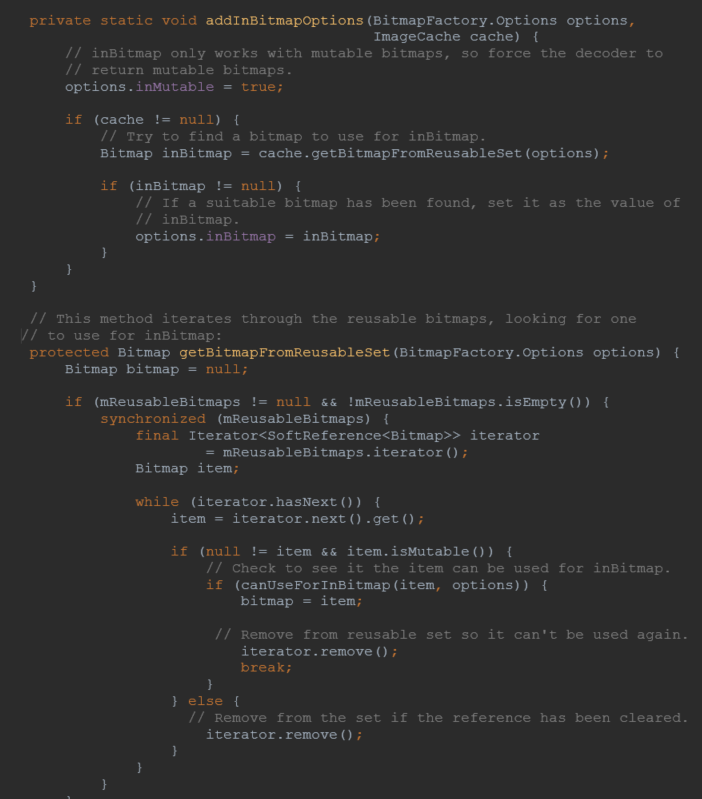
**Рисунок 18 –** Реализовать сохранение изображений

**4.3 Реализовать использование существующего изображения**

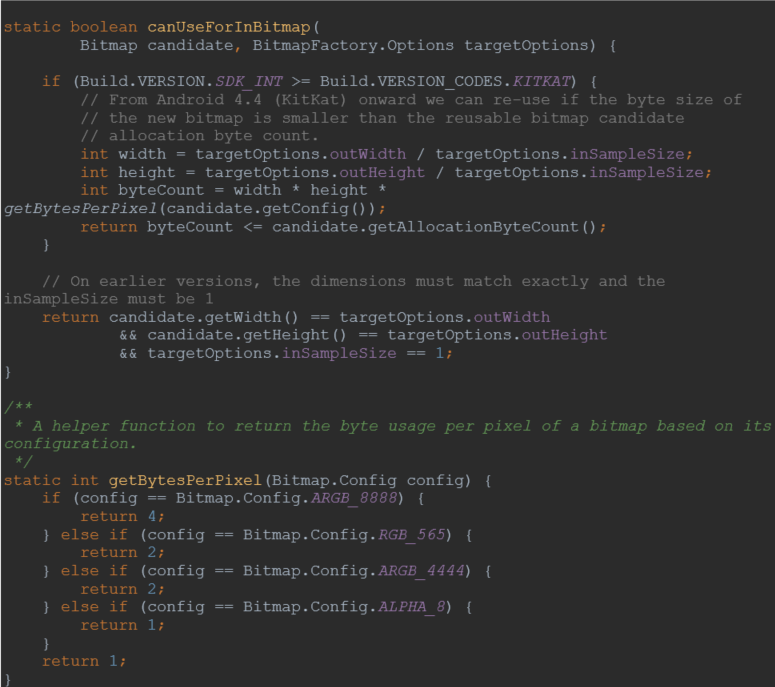
Использование существующего изображения



**Рисунок 19 – Использование существующего изображения**



**Рисунок 20 – метод addInBitmapOptions()**



**Рисунок 21 – Наконец последний метод, который определяет, удовлетворяет ли кандидат размерам**

**5. Показываем изображение в интерфейсе**

5.1 Реализовать загрузку изображений в компонент ViewPager

Паттерн свайп отличный выбор для реализации детального просмотра изображений в галерее. Однако, лучше подходит адаптер подкласса FragmentStatePagerAdapter, который автоматически уничтожает и сохраняет состояние фрагментов в компонентеViewPager, как только они скрываются с экрана, снижая расход памяти.



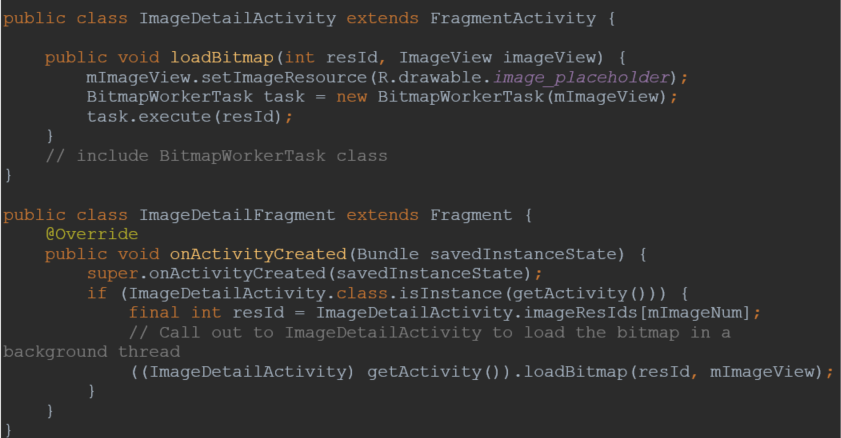
**Рисунок 22 – Реализация ViewPager с дочерними компонентами ImageView.**

Далее приведена реализация фрагмента, в котором находятся компоненты ImageView.



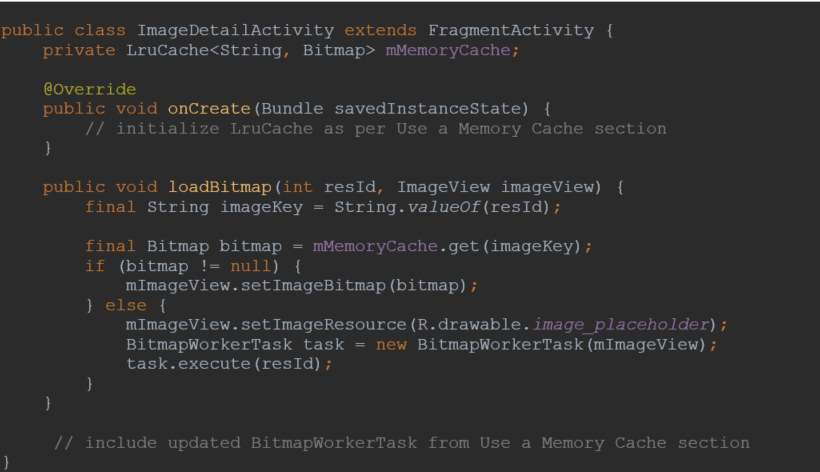
**Рисунок 23 – Реализация фрагмента, в котором находятся компоненты ImageView.**

Используйте объект типа AsyncTask, как описано в материале Обработка изображений в отдельном потоке., чтобы убрать загрузку и обработку изображений в отдельный поток:



**Рисунок 24 – Убрать загрузку и обработку изображений в отдельный поток**

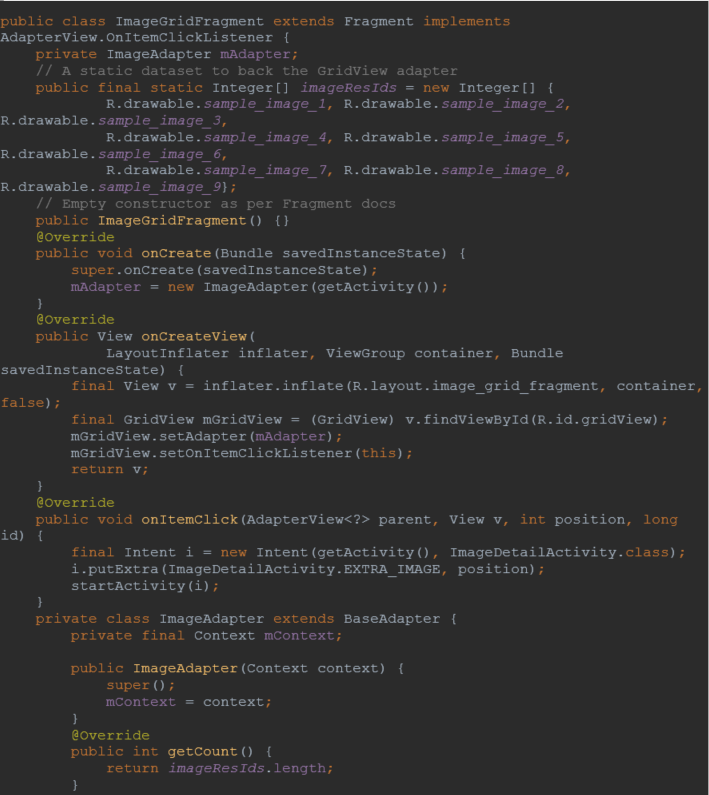
Любая дополнительная обработка (изменение размеров или загрузка изображений по сети) может выполняться в классе BitmapWorkerTask (который мы создали в прошлых уроках) не причиняя ущербу отзывчивости пользовательского интерфейса.

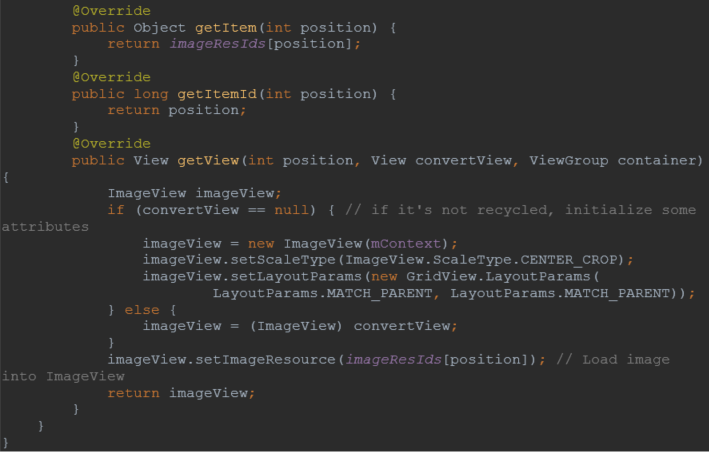


**Рисунок 25 – Добавление кэш-памяти или дискового кэша**

5.2 Реализовать загрузку изображений в компонент GridView

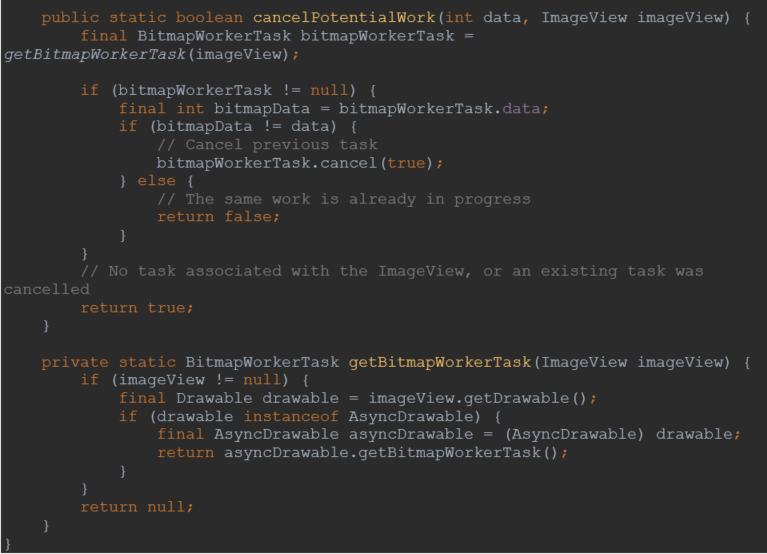
Паттерн сетчатые списки удобен для отображения наборов изображений, и для его реализации может использоваться компонент GridView, в котором несколько изображений показано на экране и еще некоторое количество готово к тому, что пользователь будет листать экран вверх или вниз.Для начала возьмем стандартный GridView с дочерними ImageView элементами, размещенными внутри фрагмента.





**Рисунок 26 – стандартный GridView с дочерними ImageView элементами, размещенными внутри фрагмента.**





**Рисунок 27 – Устранить возникшую проблему**

# ВЫВОД

В ходе выполнения практической работы были изучены:

* Методические указания Графика и анимация

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Лекции по дисциплине «Разработка мобильных приложений» / И. В. Синицын, МИРЭА — Российский технологический университет, 2022.