|  |
| --- |
|  |

Универзитет ,,Св. Кирил и Методиј” – Скопје

Факултет за информатички науки и компјутерско инженерство

**Проект по предметот:**

**Обработка на слики**

**Наслов:**

**Андроид апликација: PhotoEffect**

****

**Изработиле: Професор:**

Владимир Димиќ 111054 Ивица Димитровски

Елена Камчевска 111108

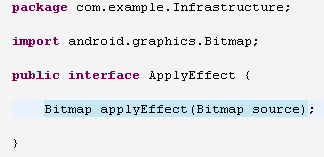
Скопје, 2014

**Опис на апликацијата**

Андроид апликацијата PhotoEffect претставува едноставна апликација за оние корисници кои се љубители на фотографирањето. Оваа апликација на корисниците им овозможува да прават слики со добар квалитет, односно резолуција и на тие слики да применат некои од седумте вградени филтри на самата апликација. Бидејќи голем дел од корисниците на смарт телефоните кои користат ваков тип на апликации, често пати ги користат само филтрите без да ги употребат останатите алатки за едитирање, па така дојде и нашата идеја за ваква апликација која, исто така овозможува сликите да се споделат на некои од социјалните мрежи по желба. Во продолжение на документацијата е објаснет секој сегмент од кодот на апликацијата.

**Составни делови на апликацијата**

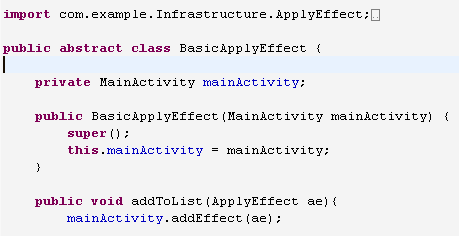
Апликацијата е составена од три дела и тоа: Implementation, Infrastructure и Testing. Implementation ги содржи сите 9 филтри на апликацијата кои можат да се применат од страна на корисникот откако ќе фотографира. Филтрите со кои располага апликацијата се: gamma effect, snow effect, sepia effect, negative effect, reflection effect, black dots effect, greyscale effect, sunny effect и old black white effect. Сите филтри се претставени како посебни класи. Infrastructure го содржи интерфејсот *ApplyEffect* кој се имплементира во сите класи на филтрите и го содржи методот *applyEffect (Bitmap source):*



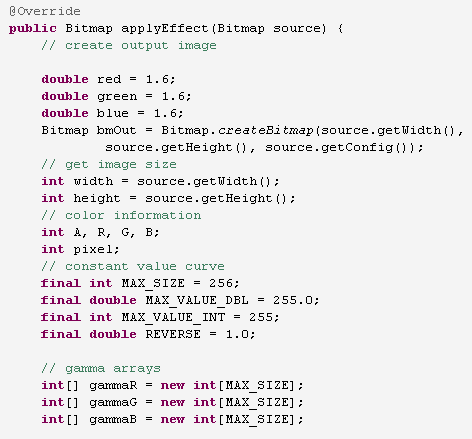
Testing ја содржи *MainActivity* класата за тестирање на филтрите и имплементирање на методите со функционалностите кои ги нуди апликацијата.

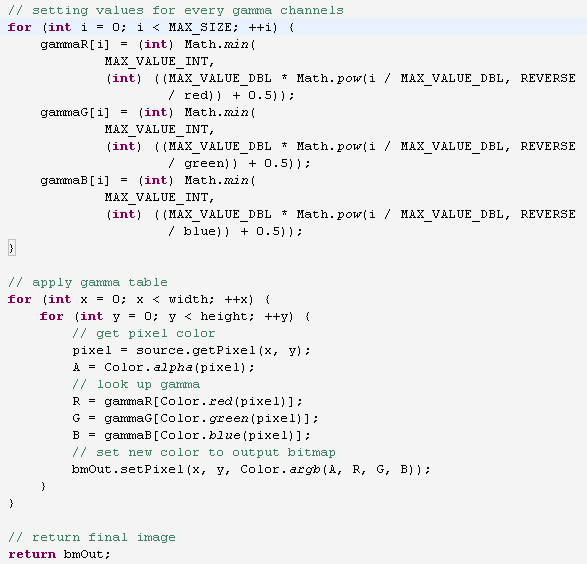
**Implementation**

**BasicApplyEffect** ја претставува класата на basic филтерот. Таа е апстрактна класа која има само една променлива *mainActivity*, конструктор и методата *addToList(ApplyEffect ae)*. Од оваа класа наследуваат сите други класи на филтрите, односно таа претставува супер класа.

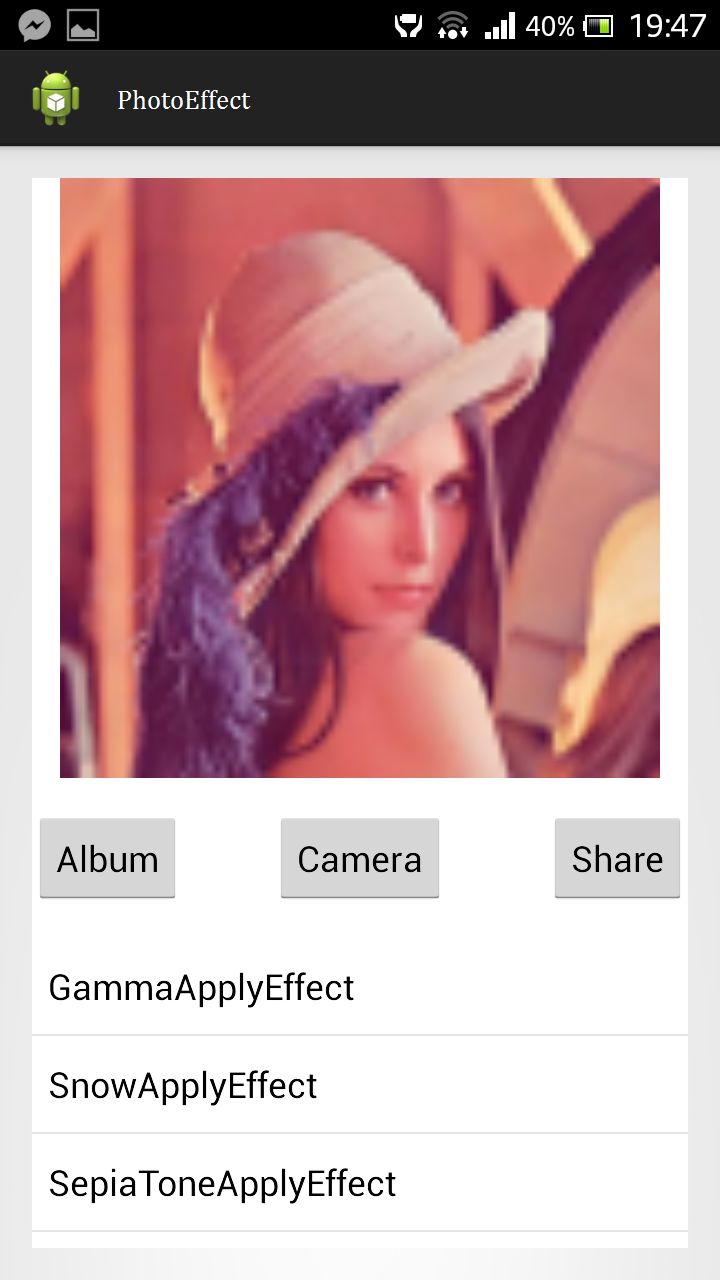
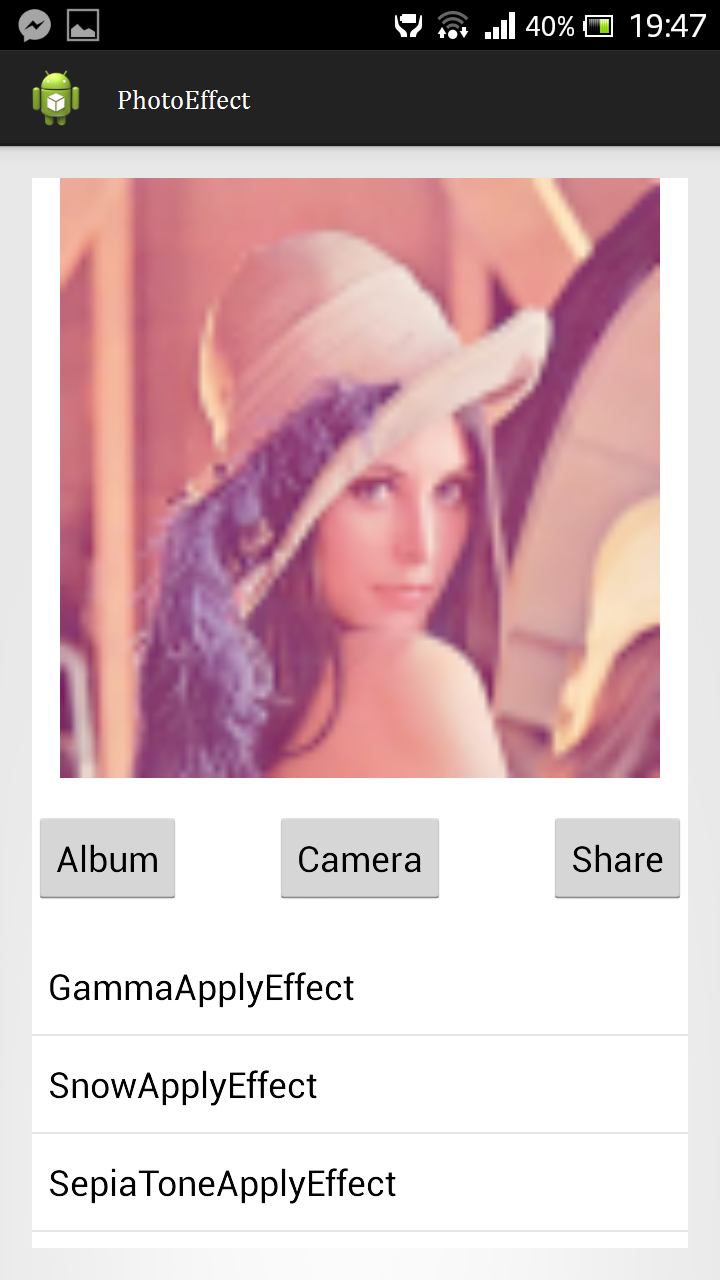


**GammaApplyEffect** ја претставува класата на gamma филтерот. Оваа класа наследува од *BasicApplyEffect* класата, односно ги наследува нејзините променливи и методите, содржи конструктор и го имплементира методот *аpplyEffect* од интерфејсот *ApplyEffect*. Во оваа класа, функцијата на методот *applyEffect* која на влез прима слика е различна од претходните класи. Најпрво, таа креира излезна слика со тоа што го подесува степенот на трите основни бои во неа, red, green и blue. Потоа, ги зема информациите за големината на сликата, боите, константните вредности на кривата и креира три гама низи за red, green и blue. Прво повикува еден for циклус кој ги подесува вредностите за секој гама канал, а потоа два вгнездени for циклуси кои ја применуваат гама табелата, односно го земаат пикселот на бојата, ја бара гама и и поставува нова боја на излезната битмапа. Оваа метода како резултат ја враќа финалната битмапа.

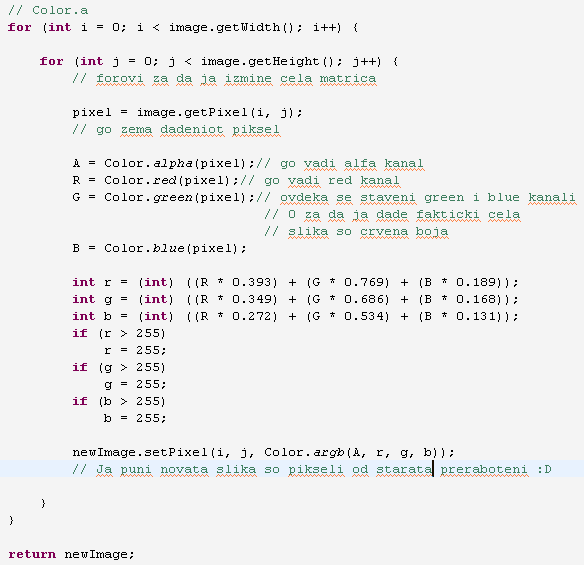




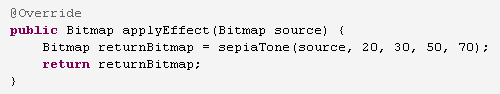
Пример:

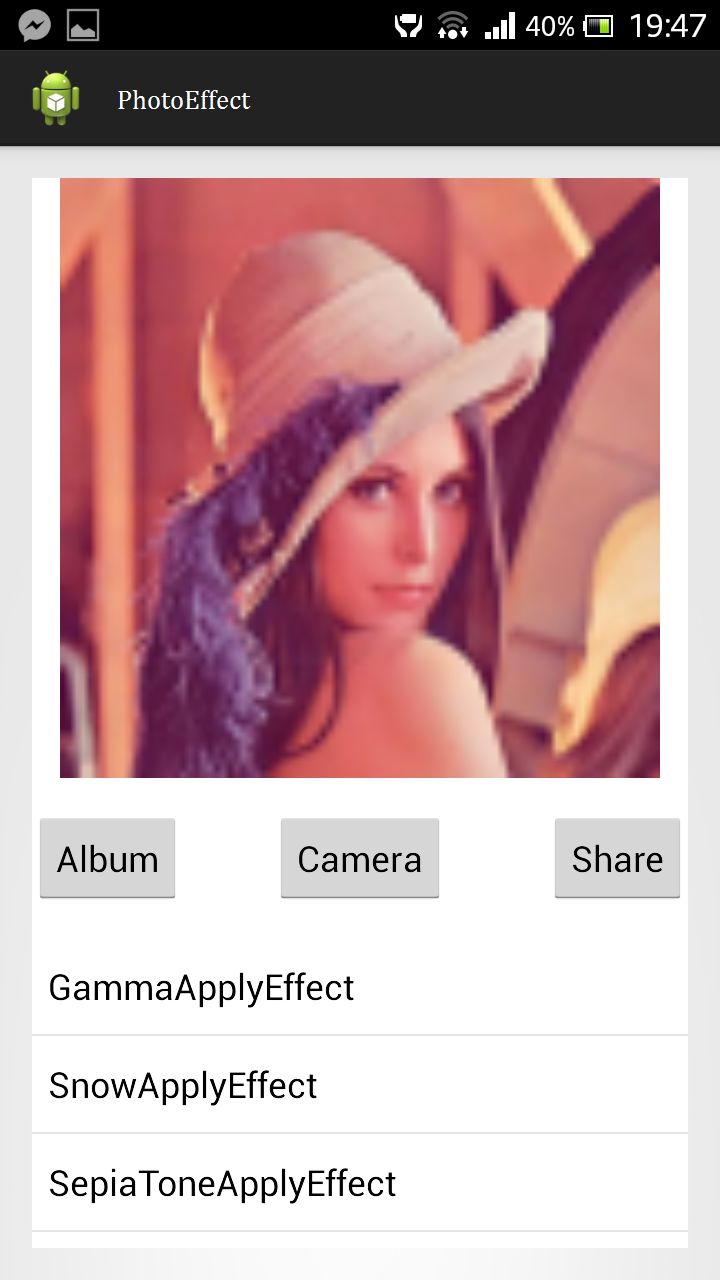
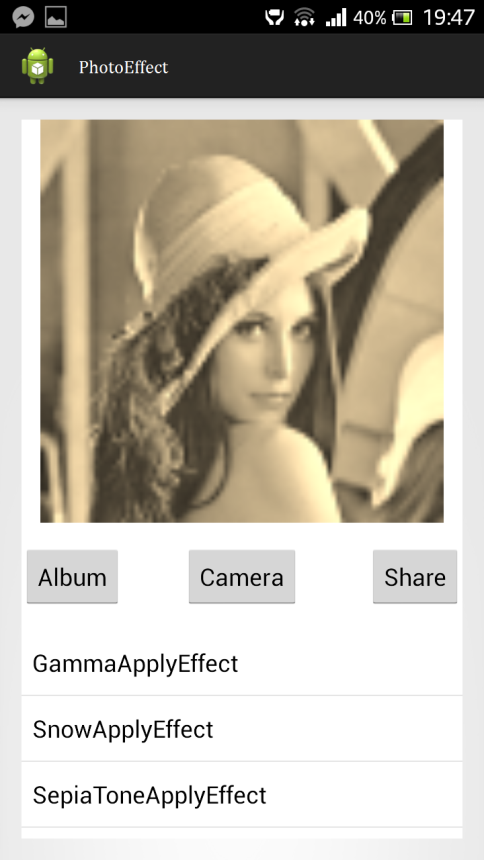
**SepiaToneApplyEffect** ја претставува класата на sepia филтерот. Оваа класа наследува од *BasicApplyEffect* класата, односно ги наследува нејзините променливи и методите, содржи конструктор и го имплементира методот *аpplyEffect* од интерфејсот *ApplyEffect*. Оваа класа покрај *applyEffect* методот, креира уште еден метод *sepiaTone* кој на влез прима 5 аргументи и тоа *image*, *depth*, *red*, *green* и *blue*. Оваа метода креира нова слика *newImage* и два вгнездени for циклуси со кои ја изминува целата матрица при што го зема дадениот пиксел, ги вади alpha и red каналите, а ги става green и blue каналите за да се добие целата слика со црвена боја и ја полни новата слика со пиксели од старата слика кои се преработени. На крај ја враќа новата слика, *newImage. newImage*.



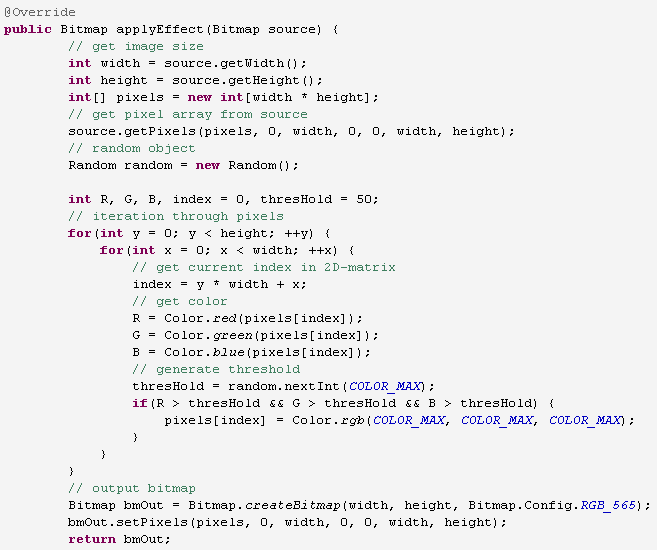
Потоа, методот *sepiaTone* се применува во методот *applyEffect* со дадени вредности и се добива соодветната излезна слика.

****

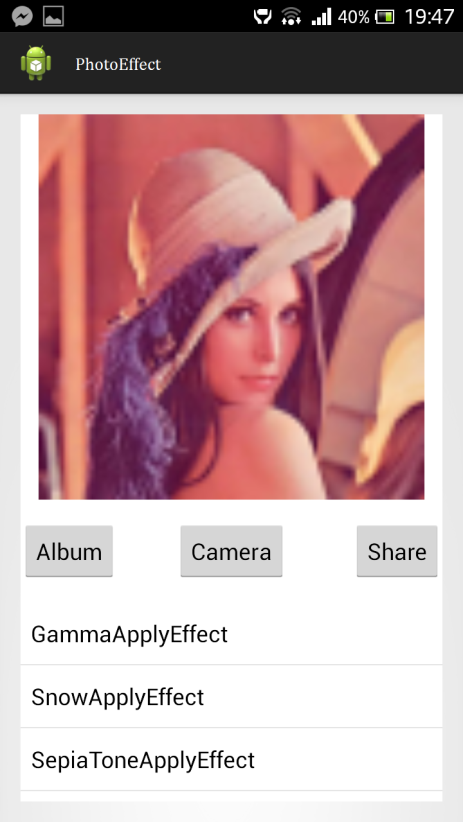
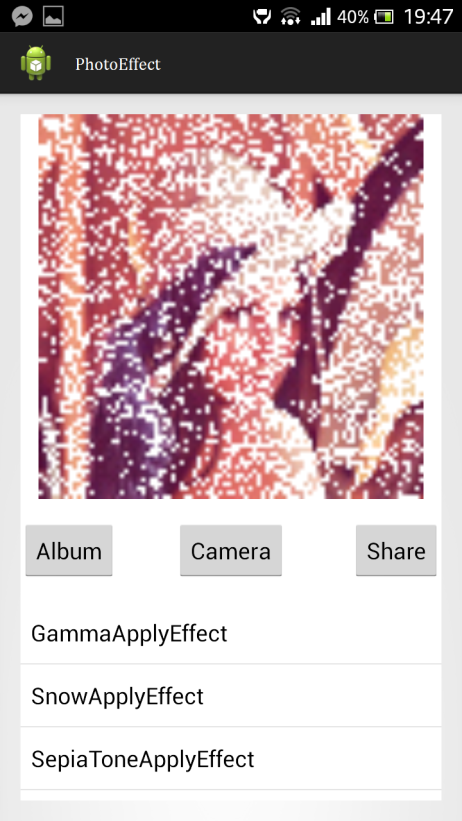
Пример:

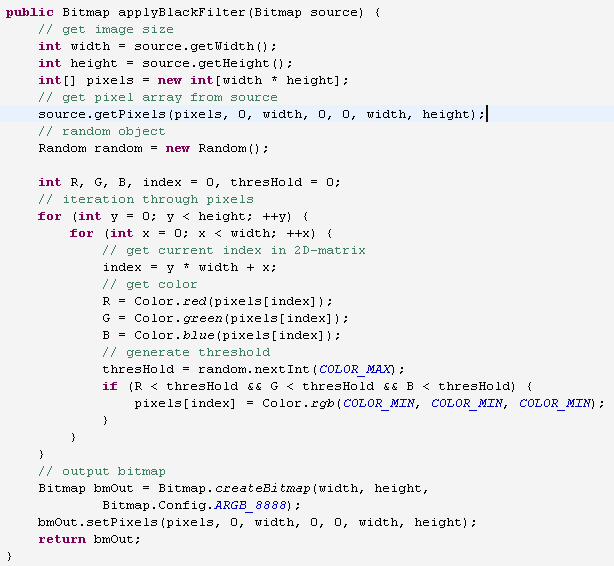
**SnowApplyEffect** ја претставува класата на snow филтерот. Оваа класа наследува од *BasicApplyEffect* класата, односно ги наследува нејзините променливи и методите, содржи конструктор и го имплементира методот *аpplyEffect* од интерфејсот *ApplyEffect*. Во оваа класа на почетокот се декларираат две статички променливи со одредени вредности и тоа *COLOR\_MIN = 0x00* и *COLOR\_MAX = 0xFF* кои ќе се применат во методот *applyEffect*. Методот *applyEffect* кој на влез прима слика, ги зема димензиите на влезната слика, го зема секој пиксел во низа од влезната слика и креира еден рандом објект *random*. Потоа креира два вгнездени for циклуси кои итерираат низ секој пиксел, така што го зема моменталниот индекс во 2D матрица, ја зема бојата и го генерира прагот, односно *threshold*. На крај ја враќа излезната слика.

****

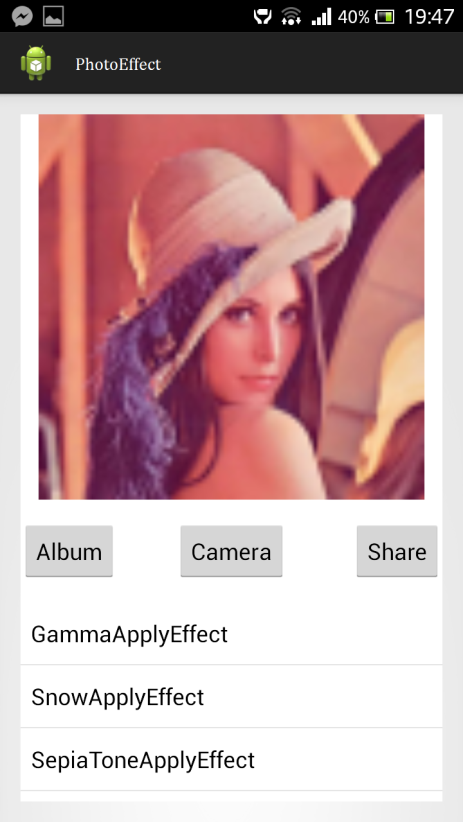
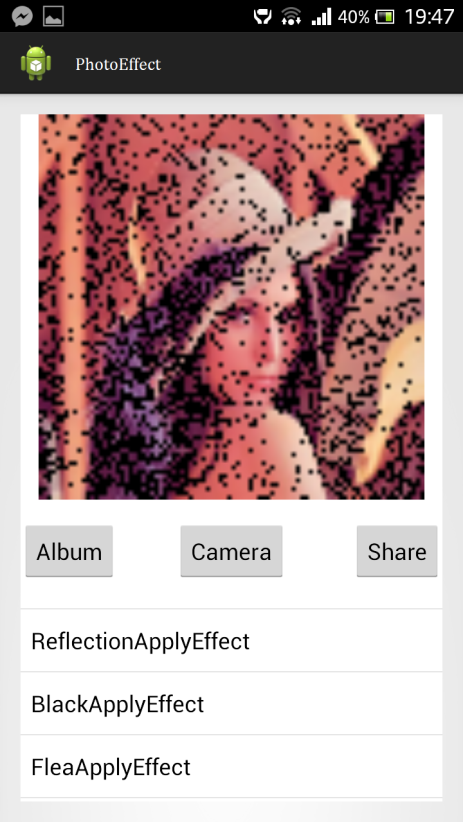
Пример:

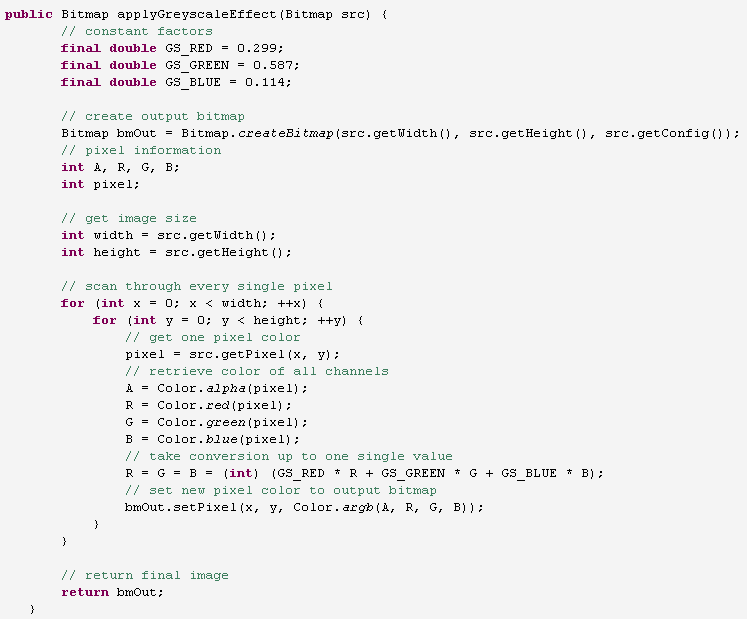
**BlackDotsApplyEffect** ја претставува класата на black dots филтерот. Оваа класа наследува од *BasicApplyEffect* класата, односно ги наследува нејзините променливи и методите, содржи конструктор и го имплементира методот *аpplyEffect* од интерфејсот *ApplyEffect*. Во оваа класа се декларираат две статички променливи со одредени вредности и тоа *COLOR\_MIN = 0x00* и *COLOR\_MAX = 0xFF* кои ќе се применат во методот *applyEffect*. Методот *аpplyEffect* кој на влез прима слика, ги зема димензиите на влезната слика, го зема секој пиксел во низа од влезната слика и креира еден рандом објект *random*. Потоа креира два вгнездени for циклуси кои итерираат низ секој пиксел, така што го зема моменталниот индекс во 2D матрица, ја зема бојата и го генерира прагот, односно *threshold*. На крај ја враќа излезната слика.



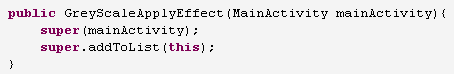
Пример:

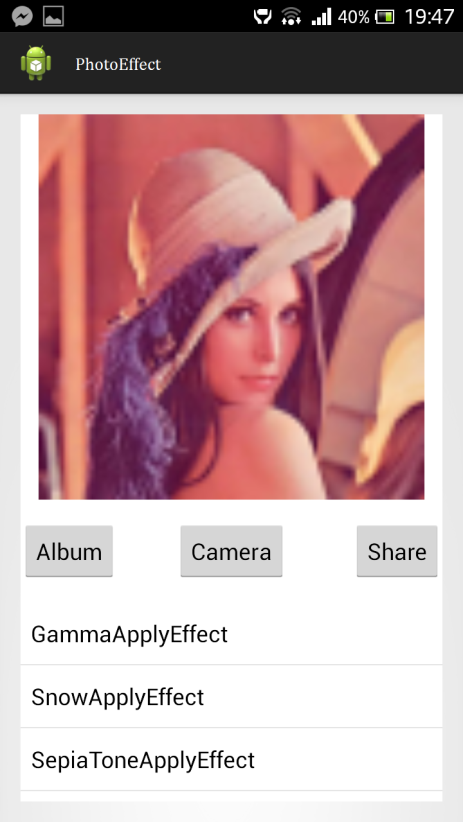
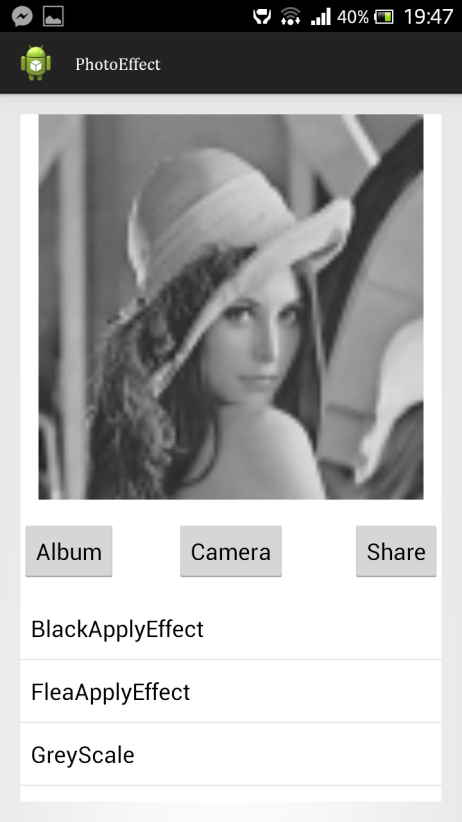
**GrayScaleApplyEffect** ја претставува класата на grey scale филтерот. Оваа класа наследува од *BasicApplyEffect* класата, односно ги наследува нејзините променливи и методите, содржи конструктор и го имплементира методот *аpplyEffect* од интерфејсот *ApplyEffect*. Оваа класа покрај *applyEffect* методот, креира уште еден метод *appyGrayscaleEffect* кој на влез прима слика. Овој метод декларира три константни фактори *GS\_RED = 0.299, GS\_GREEN = 0.587, GS\_BLUE = 0.114*. Потоа, креира излезна битмапа, декларира променливи во кои ќе се сместат информациите за пикселите и ги зема димензиите на влезната слика. Креира два вгнездени for циклуси со кои го скенира секој пиксел, односно зема еден пикел по боја, ја враќа бојата за секој канал, превзема интеракција за секоја една вредност и го поставува новиот пиксел на бојата во излезната битмапа. На излез, оваа метода враќа финална слика.



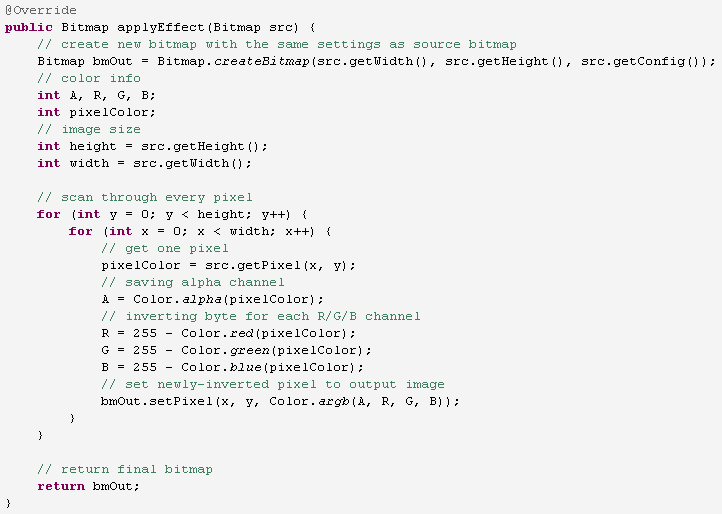
Потоа, методот *applyGrayscaleEffect* се применува во методот *applyEffect* и се добива соодветната излезна слика.



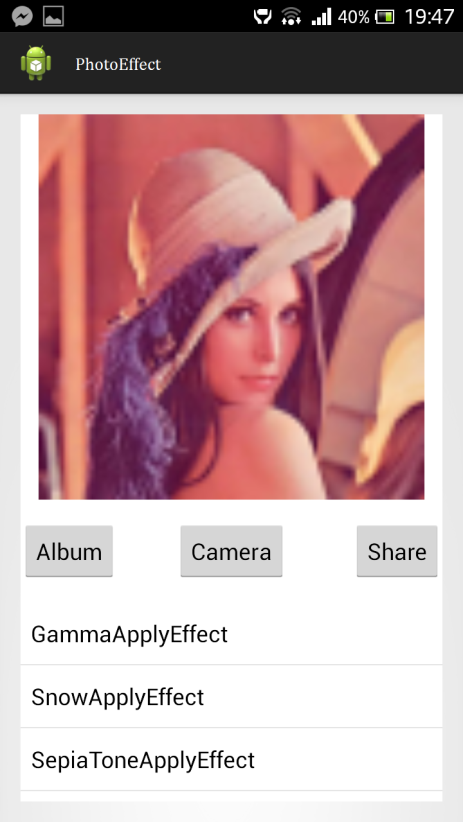
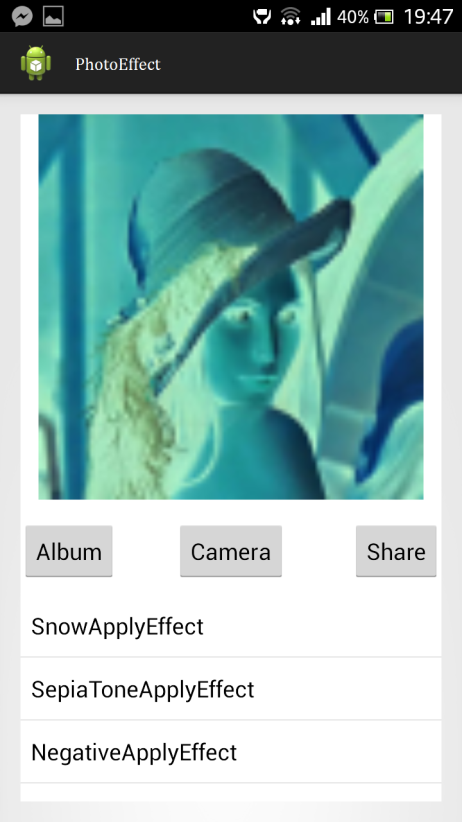
Пример:

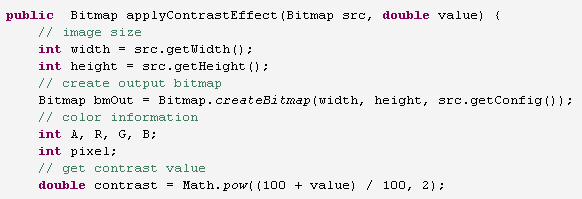
**NegativeApplyEffect** ја претставува класата на negative филтерот. Оваа класа наследува од *BasicApplyEffect* класата, односно ги наследува нејзините променливи и методите, содржи конструктор и го имплементира методот *аpplyEffect* од интерфејсот *ApplyEffect*. Методот *applyEffect* кој прима на влез слика, креира нова битмапа со истите карактеристики (settings) како и изворната битмапа., декларира променливи во кои ќе се чуваат информациите за бојата и ги зема димензиите на влезната слика. Потоа, креира два внгездени for циклуси со кои скенира низ секој пиксел на сликата со тоа што зема еден пиксел, го зачувува alpha каналот, инвертира бајт по бајт за секој red, green и blue канал и поставува нов, инвертиран пиксел во излезната битмапа и така се повторуваат чекорите за секој пиксел од сликата. На крај, оваа метода враќа финална битмапа.

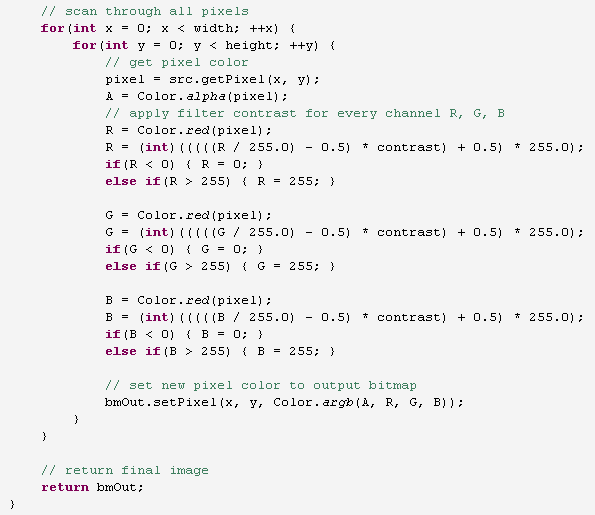


Пример:

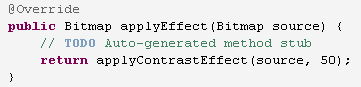
 

**OldBlackWhiteApplyEffect** ја претставува класата на old back white филтерот. Оваа класа наследува од *BasicApplyEffect* класата, односно ги наследува нејзините променливи и методите, содржи конструктор и го имплементира методот *аpplyEffect* од интерфејсот *ApplyEffect*. Оваа класа покрај *applyEffect* методот, креира уште еден метод *applyContrastEffect* кој на влез прима две променливи и тоа *Bitmapa src* и *double value*. Оваа метода, ги зема димензиите на влезната слика, креира излезна битмапа, декларира променливи во кои ќе се чуваат информациите за бојата и ја зема вредноста на контрастот. Потоа, креира две вгнездени for циклуси со кои скенира низ сите пиксели со тоа што го зема пикселот на бојата, го применува филтерот контраст за секој канал red, green и blue и поставува нов пиксел на бојата во излезната битмапа. На крај, оваа метода враќа финална слика.

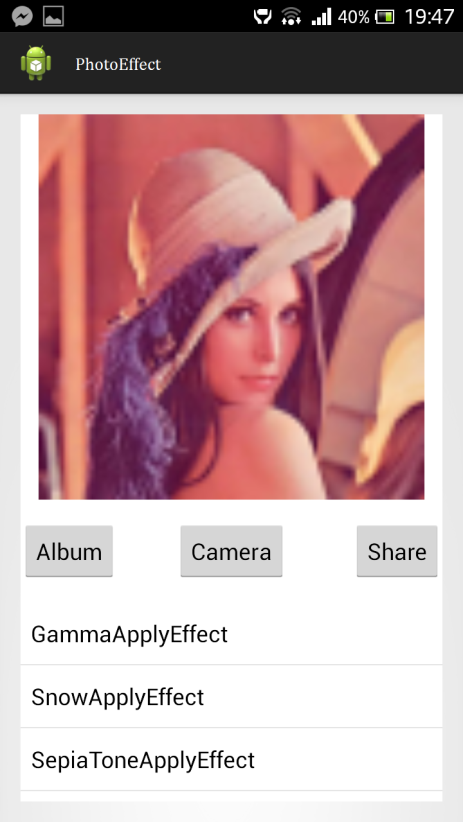
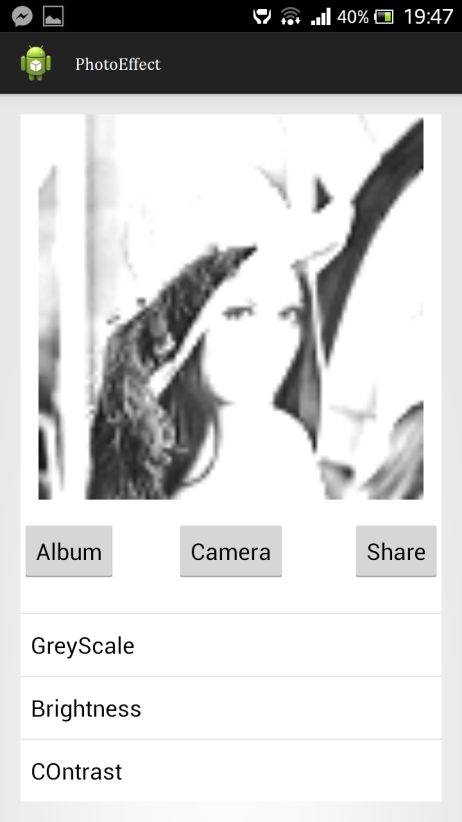




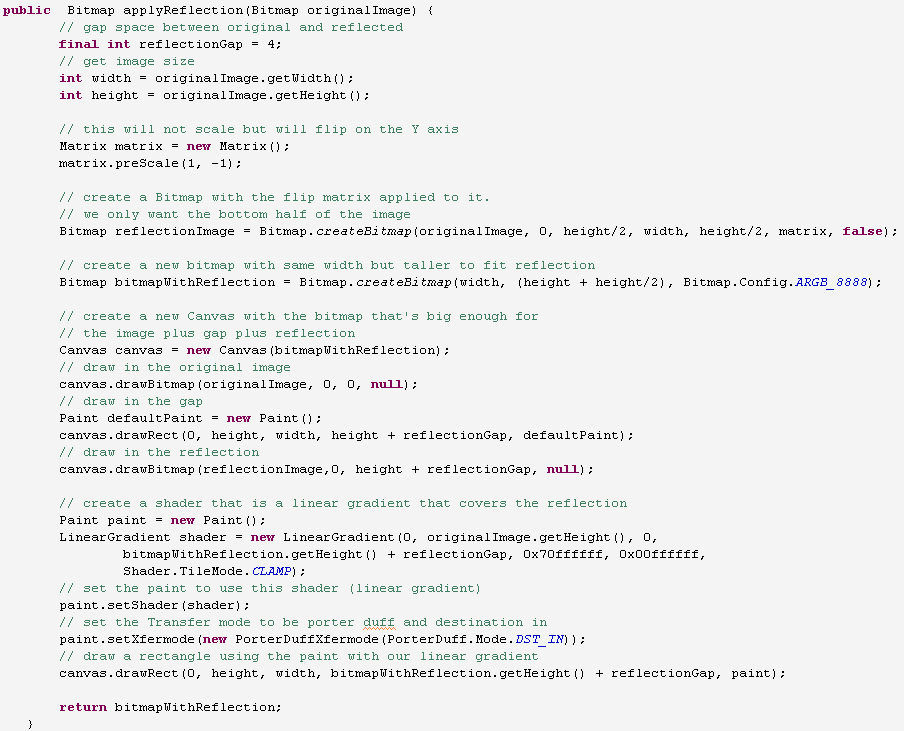
Потоа, во методот *applyEffect* се применува методот *applyContrastEffect* со дадени вредности и се добива соодветната излезна слика.



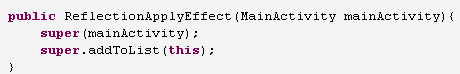
Пример:

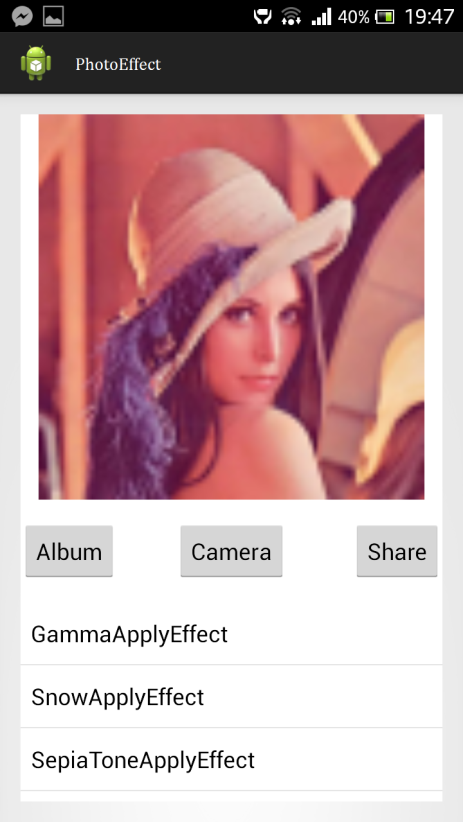
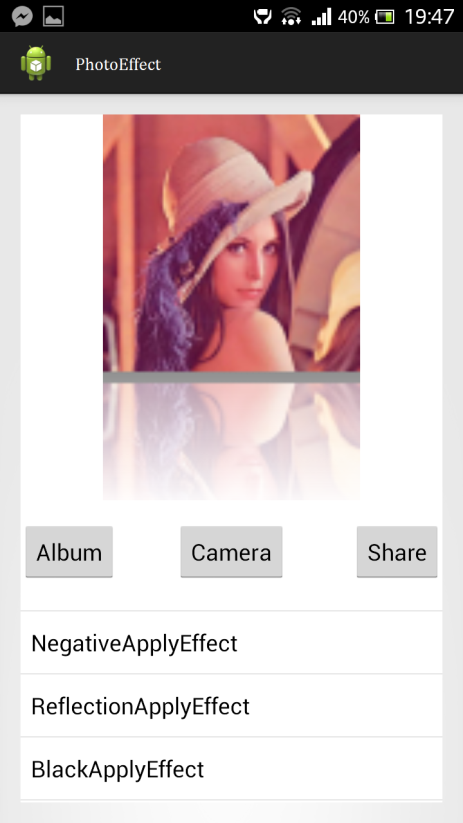
**ReflectionApplyEffect** ја претставува класата на reflection филтерот. Оваа класа наследува од *BasicApplyEffect* класата, односно ги наследува нејзините променливи и методите, содржи конструктор и го имплементира методот *аpplyEffect* од интерфејсот *ApplyEffect*. Оваа класа покрај *applyEffect* методот, креира уште еден метод *applyReflection* кој на влез прима слика. Овој метод, декларира променлива во која се чува растојанието помеѓу оригиналната слика и рефлектираната, ги зема димензиите на влезната слика и креира матрица со која нема да се скалира, но ќе се допре на y оската. Потоа, креира битмапа со примена на флип матрицата, со тоа што ја сакаме само долната половина на сликата. Креира уште една нова битмапа со истата ширина, но повисока за да одговара на рефлекцијата. Исто така, креира канвас со битмапата кој е доволно голем за сликата заедно со растојанието и рефлекцијата, го исцртува во оригиналната слика и во растојанието и во рефлекцијата. Потоа, креира линеарен градиент кој на покрива рефлекцијата, ја поставува бојата за да го користи градиентот, го поставува трансфер моделот да биде porter duff и крајна дестинација и црта правоаголник користејќи ја бојата со линеарниот градиент. На крај, оваа метода враќа рефлектирана битмапа.



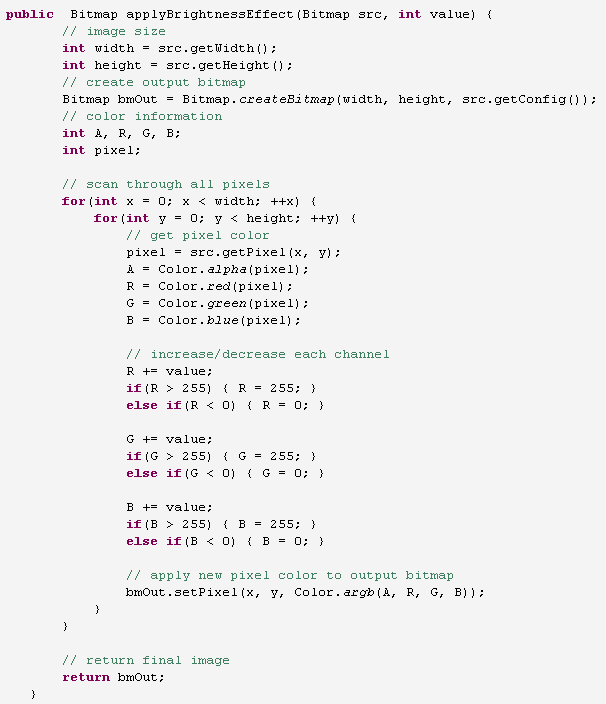
Потоа, во методот *applyEffect* се применува методот *applyReflection* и се добива соодветната излезна слика.



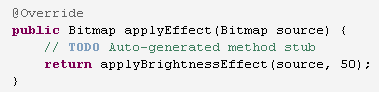
Пример:

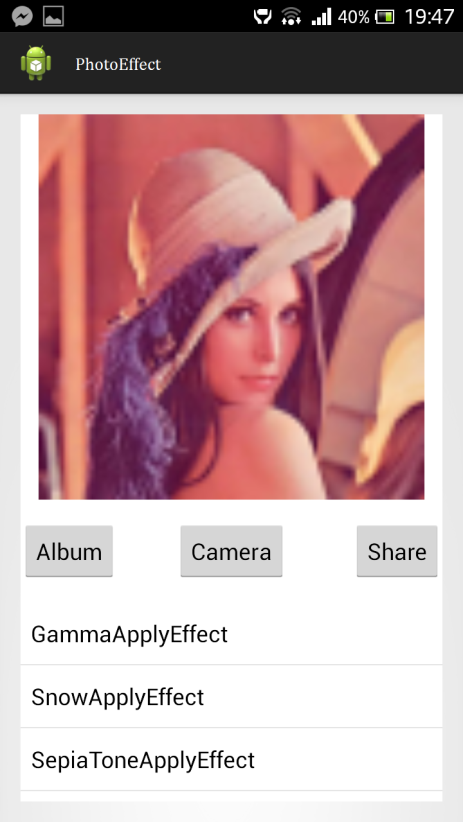
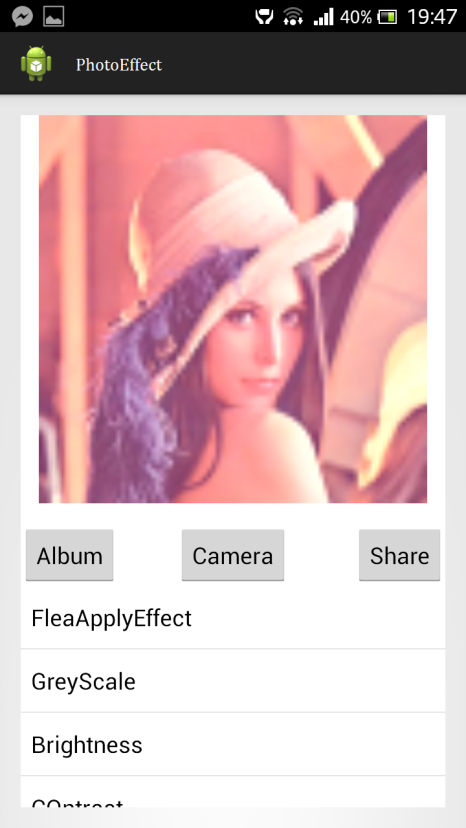
**SunnyApplyEffect** ја претставува класата на sunny филтерот. Оваа класа наследува од *BasicApplyEffect* класата, односно ги наследува нејзините променливи и методите, содржи конструктор и го имплементира методот *аpplyEffect* од интерфејсот *ApplyEffect*. Оваа класа покрај *applyEffect* методот, креира уште еден метод *applyBrightnessEffect* кој на влез прима две променливи и тоа *Bitmap src* и *int value*. Овој метод, ги зема димензиите на влезната слика, креира излезна битмапа и декларира променливи во кои ќе се чуваат информациите за бојата. Потоа, креира две вгнездени for циклуси со кои скенира низ сите пиксели со тоа што го зема пикселот на бојата, го зголемува / намалува соодветно секој канал и го применува новиот пиксел на бојата во излезната битмапа и така сите овие чекори се повоторуваат за сите пиксели на сликата. На крај, оваа метода враќа финална слика.



Потоа, во методот applyEffect се применува методот applyBrightnessEffect со одредени вредности и се добива соодветната излезна слика.

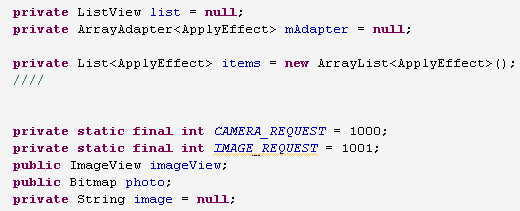


Пример:

**Testing**

Testing ја содржи **MainActivity** класата за тестирање на филтрите и имплементирање на методите со функционалностите кои ги нуди апликацијата. *MainActivity* класата наследува од **Activity** класата и во оваа класа се импортираат сите класи на филтрите кои ги објаснивме претходно. Глобални променливи кои се декларирани во *MainActivity* класата се *list* со вредност null, *mAdapter* од податочен тип ArrayAdapter со вредност null, *items* од податочен тип List која претставува празна листа на елементи, статички променливи *CAMERA\_REQUEST = 1000* и *IMAGE\_REQUES = 1001, imageView* од податочен тип ImageView, *photo* од податочен тип Bitmap и *image* од податочен тип String со вредност null.

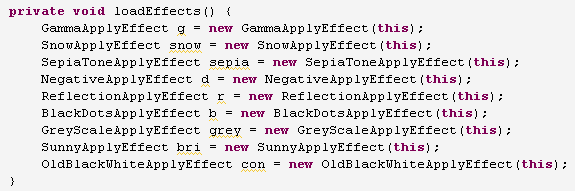


Методите кои ги содржи класата *MainActivity* се a*ddEffect(), loadEffects(), onCreate(), changePhoto(), share(), getPicture(), firstImage(), getImage(), onActivityResult(), onCreateOptionsMenu()* и *onOptionsItemSelected()*. Секој од овие методите ќе биде посебно објаснет како работи.

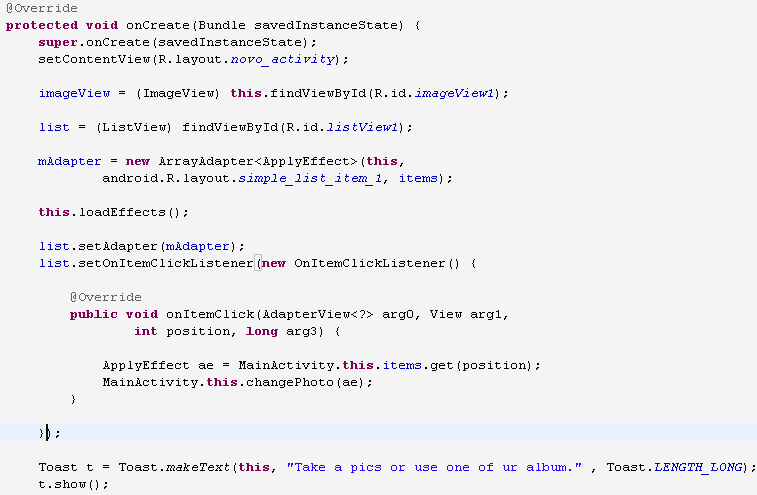
**addEffect()** методот служи за динамички секоја од класите ( филтри ) да се додадат во листата на филтри во апликацијата, односно во List<ApplyEffect> items.

Cap_2.bmp

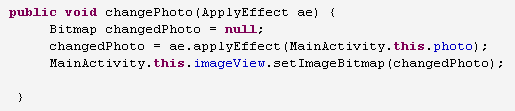
**loadEffects()** методот креира инстанци од секоја од класите на филтрите посебно. Се препраќа this објектот за да класите сами се додадат себеси со користење на предходната метода **addEffect()** во листата на филтри во aпликацијата.



**onCreate() е** метод во кој се иницијализира апликацијата. Најважно е тоа што во оваа метода се повикува setContentView() со layout ресурс дефинирајќи го UI и користејќи го findViewById() за да врати widgets во тој UI кои се потребни за програмска интеракција. Се повикува кога активноста е прво креирана.



**changePhoto()** е метод кој ја креира новата филтрирана слика.



**share()** е метод кој овозможува сликата која е фотографирана и на која е применет некој од филтрите да може да се сподели на некоја од социјалните мрежи зависно од тоа која од нив ги користи корисникот.



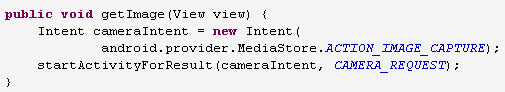
**getPicture()** е метод со кој што може да се земе слика од албумот на самиот телефон и да се обработи со помош на некој од филтрите.



**firstImage()** е метод кој откако ќе се вклучи апликацијата, мора прво да се фотографира, инаку ја прикажува фотографијата во првичната состојба, на која нема никаков филтер.



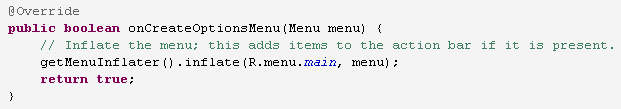
**getImage()** е метод со кој се отвора апликацијата апарат (camera) и може да се слика слика со помош на камерата на телефонот и потоа таа да се обработува.



**onActivityResult()** методот служи за апликацијата да комуницира со другите апликации кои се веќе инсталирани на телефонот, со тоа што тука доаѓа резултатот од комуникацијата. Ако е успешен ќе се изврши кодот, а со тоа ќе биде преземена сликата од камерата или од албумот.



**onCreateOptionsMenu()** се повикува само еднаш, односно првиот пат кога менито со опции е прикажано и всушност ни ги прикажува опциите кои може да се користат, а во овој случај е само една и тоа опцијата firstImage.



**onOptionsItemSelected()** е метод кој што проверува која опција е кликната, во нашиот случај имаме само една, а таа е firstImage.

