

Модуль 1. Основы программирования

*Практикум. Массивы в анимации (к теме 1.8)

Оглавление

Практикум. Массивы в анимации	2
1. Простейшая анимация	2
2. Массивы в анимации	5
Задания.....	8
Благодарности	9

Практикум. Массивы в анимации

Этот материал является дополнительным и по желанию преподавателя может быть использован для проведения практикума в Модуле 1 или самостоятельной работы учащихся.

1. Простейшая анимация

Для того, чтобы добавить в простую графическую программу анимацию, можно использовать таймер.

Таймер в программировании это фактически будильник, который “звонит” через определенные промежутки времени, и это дает возможность делать некоторые операции периодически, например, менять кадры анимации каждую десятую долю секунды.

В Java под Android это можно реализовать при помощи объекта класса-наследника класса `CountDownTimer`.

В классе своего таймера нужно определить функции

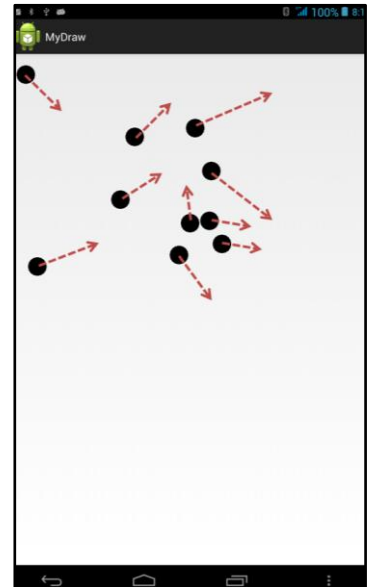
```
public void onTick(long millisUntilFinished) и  
public void onFinish()
```

В них прописываются действия, которые нужно сделать периодически (“каждый tick”) и действия, которые необходимо сделать в конце, когда таймер заканчивает свою работу.



В классе `android.os.CountDownTimer` эти функции объявлены абстрактными. Это значит, что в самом классе `CountDownTimer` фактически объявлены только названия, а функциональность, то есть действия, которые нужно выполнять при их вызове, должны быть объявлены в классе-наследнике.

Подробнее об абстрактных функциях мы поговорим позднее.



Рассмотрим работу таймера на простом примере. Мы будем использовать графический скелетный проект из предыдущих занятий. Его можно скачать с сайта школы и импортировать.

Пусть класс `MyDraw` определен так:

```
public class MyDraw extends View {  
    int x = 0;  
    MyDraw(Context context) {  
        super(context);  
    }  
  
    @Override  
    protected void onDraw(Canvas canvas) {  
        Paint paint = new Paint();  
        canvas.drawCircle(x, 200, 20, paint);  
    }  
}
```

Он рисует небольшой кружок, горизонтальное расположение которого зависит от переменной класса `x`.


Объявим класс таймера не как обычно в отдельном файле, а в необычном месте, *внутри* класса `MyDraw`. Это даст возможность очень просто управлять из функций таймера объектом `MyDraw`.

```
class MyTimer extends CountDownTimer
{
    MyTimer()
    {
        // общее время работы таймера 100 сек,
        // время периодического срабатывания 0.1 сек
        super(100000, 100);
    }


    @Override
    public void onTick(long millisUntilFinished) {
        //вызов функции внешнего класса MyDraw для смены кадра
        nextFrame();
    }

    @Override
    public void onFinish() {
    }
}
```

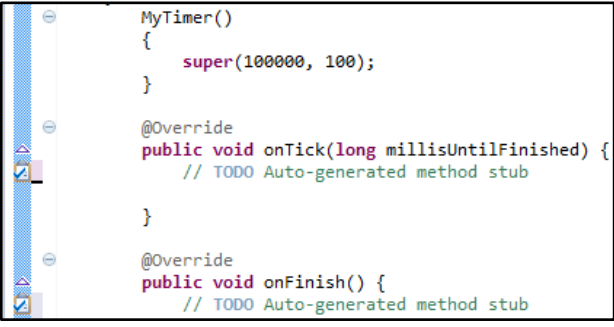
При создании объекта вызывается конструктор суперкласса `CountDownTimer`, при этом устанавливается общее время работы таймера 100 сек, при этом десять раз в секунду будет вызываться функция `onTick`.



При создании класса `MyTimer` Eclipse указывает на необходимость определения абстрактных функций



и предлагает добавить реализацию.



Заголовки определенных абстрактных функций помечаются белыми треугольничками слева от названия.

Благодаря, тому что класс **MyTimer** объявлен внутри класса **MyDraw**, то есть является внутренним классом, можно просто вызвать функцию внешнего класса **nextFrame**. (Во внутреннем классе - в таймере - нет функции **nextFrame**, поэтому вызывается функция из внешнего класса **MyDraw**).

Сама функция **nextFrame** может выглядеть так:

```
void nextFrame()
{
    // увеличиваем координату кружка
    x += 5;
    // вызываем перерисовку
    invalidate();
}
```



*Обратите внимание! Мы не вызываем функцию **onDraw** сами, а отдаем системе заказ на перерисовку. Функция **onDraw** вызовется системой.*

Функцию **onFinish** нужно тоже определить, но тело ее можно оставить пустым.

Остается последнее: создать и запустить таймер. Лучше всего это сделать при создании **MyDraw** в конструкторе:

```
MyDraw(Context context) {
    super(context);
    MyTimer timer = new MyTimer();
    timer.start();
}
```

Анимация готова! Шарик должен улететь вправо.

Таким образом весь код класса **MyDraw** будет выглядеть так:

```
package ru.samsung.itschool.mydraw;

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Paint;
import android.os.CountDownTimer;
import android.view.View;

public class MyDraw extends View {
    int x = 0;
    MyDraw(Context context) {
        super(context);
        MyTimer timer = new MyTimer();
        timer.start();
    }

    @Override
    protected void onDraw(Canvas canvas) {
        Paint paint = new Paint();
        canvas.drawCircle(x, 200, 20, paint);
    }

    void nextFrame()
    {

```

```
        x += 5;
        invalidate();
    }

    // MyTimer - внутренний класс
    class MyTimer extends CountdownTimer
    {
        MyTimer()
        {
            super(100000, 100);
        }

        @Override
        public void onTick(long millisUntilFinished) {
            nextFrame();
        }

        @Override
        public void onFinish() {
        }
    }
}
```



Еще раз обратите внимание. Функции класса `MyTimer` не вызываются программистом. Это функции обратного вызова и они вызываются системой.

2. Массивы в анимации

Массивы позволяют очень легко создать и управлять множеством объектов, в нашем случае шариков.

Вместо одной переменной `x` создадим массивы абсцисс и ординат, а также массивы скоростей, чтобы шарик двигался в разные стороны.

```
public class MyDraw extends View {
    int N = 10; // количество шариков
    int[] x = new int[N];
    int[] y = new int[N];
    int[] vx = new int[N];
    int[] vy = new int[N];
    ...
}
```

Теперь нужно заполнить их случайными числами. Чтобы не дублировать код, напомним функцию, которая заполняет переданный ей массив случайными числами в диапазоне от `min` до `max`.

Удобно для этого создать функцию, которая будет выдавать (возвращать) случайные целые числа в заданном отрезке:

```
int rand(int min, int max)
{
    return (int)(Math.random() * (max - min + 1)) + min;
}
```

Эта функции передаются границы отрезка **min** и **max** и она возвращает значение при помощи конструкции **return**.

Теперь функция заполнения массива записывается очень понятно:

```
void fillArrayRandom(int[] a, int min, int max)
{
    for (int i= 0; i < a.length; i++)
    {
        a[i] = rand(min, max);
    }
}
```

Достаточно четыре раза ее вызвать, чтобы полностью подготовить шарики.

Лучше этот код вынести тоже в отдельную функцию.

```
void makeBalls()
{
    fillArrayRandom(x, 0, 500);
    fillArrayRandom(y, 0, 500);
    fillArrayRandom(vx, -10, 10);
    fillArrayRandom(vy, -10, 10);
}
```

Вызвать ее можно, например, в конструкторе **MyDraw**.



Именно потому, что мы будем вызывать функцию *makeBalls* из конструктора, используется константа 500 вместо привычных *this.getWidth* и *this.getHeight*. Дело в том, что в момент создания наш *View* еще не добавлен на экран, а значит у него не установлены правильные размеры, и эти функции возвратят ноль. Использование констант в таком качестве, вообще говоря, плохая идея. Улучшить код можно несколькими способами. Например, завести в классе логическую переменную *started*, равную *false* в начале и вызывать ее один раз, например, в функции рисования:

```
protected void onDraw(Canvas canvas) {
    if (!started){
        makeBalls();
        started = true;
    }
    //...
```

Движение шариков также стоит оформить отдельным методом

```
void moveBalls()
{
    for (int i = 0; i < N; i++)
    {
        x[i] += vx[i];
        y[i] += vy[i];
    }
}
```

Благодаря этому метод **nextFrame** останется таким же коротким. И даже более понятным, чем был:

```
void nextFrame()
{
```

```
        moveBalls();  
        invalidate();  
    }
```

Остается добавить в функцию рисования цикла

```
@Override  
protected void onDraw(Canvas canvas) {  
    Paint paint = new Paint();  
    for (int i = 0; i < N; i++)  
    {  
        canvas.drawCircle(x[i], y[i], 20, paint);  
    }  
}
```

и по экрану полетит много шариков.

Полностью класс MyDraw с анимацией с использованием массивов выглядит так:

```
public class MyDraw extends View {  
  
    int N = 10;  
    int[] x = new int[N];  
    int[] y = new int[N];  
    int[] vx = new int[N];  
    int[] vy = new int[N];  
  
    int rand(int min, int max)  
    {  
        return (int) (Math.random() * (max - min + 1)) + min;  
    }  
  
    void fillArrayRandom(int[] a, int min, int max)  
    {  
        for (int i = 0; i < a.length; i++)  
        {  
            a[i] = rand(min, max);  
        }  
    }  
  
    void makeBalls()  
    {  
        fillArrayRandom(x, 0, 500);  
        fillArrayRandom(y, 0, 500);  
        fillArrayRandom(vx, -10, 10);  
        fillArrayRandom(vy, -10, 10);  
    }  
  
    void moveBalls()  
    {  
        for (int i = 0; i < N; i++)  
        {  
            x[i] += vx[i];  
            y[i] += vy[i];  
        }  
    }  
  
    MyDraw(Context context) {  
        super(context);  
  
        makeBalls();  
    }  
}
```

```
        MyTimer timer = new MyTimer();
        timer.start();
    }

    @Override
    protected void onDraw(Canvas canvas) {
        Paint paint = new Paint();
        for (int i = 0; i < N; i++) {
            canvas.drawCircle(x[i], y[i], 20, paint);
        }
    }

    void nextFrame()
    {
        moveBalls();
        invalidate();
    }

    // класс MyTimer остался без изменения
    // ...
}
```

Этот код выглядит простым и понятным именно благодаря использованию функций.

Задания

Улучшить программу можно разными способами, например,

1. Добавить цвет и размер шарикам (для этого следует создать еще два массива)
2. Нетрудно сделать так, чтобы шарики не вылетали за пределы экрана, для этого нужно добавить проверки в функцию `moveBalls` и менять скорости на противоположные:

```
//...
if (x[i] < 0 || x[i] > this.getHeight()){
    vx[i] = - vx[i];
}
// ...аналогично для y
```

или вылетевшие шарики “возвращать” на экран (генерировать координаты заново)

```
//...
if (x[i] < 0 || x[i] > this.getHeight()){
    vx[i] = rand(0, this.getWidth());
    vy[i] = rand(0, this.getHeight());
}
```

в этом коде уже можно использовать функции получения размера, потому что они вызываются в процессе работы, когда View уже получил размеры.

3. Замените окружности, на другие фигуры. Представьте, например, как будет выглядеть анимация, если функцию рисования изменить так:


```
protected void onDraw(Canvas canvas) {  
    Paint paint = new Paint();  
    for (int i = 0; i < N - 1; i++) {  
        canvas.drawLine(x[i], y[i], x[i + 1], y[i + 1], paint);  
    }  
}
```

проверьте результат на практике.

4. Можно эффектно использовать изменение цвета, например, прозрачность, чтобы шарики, улетаая “растворялись” или просто плавно менять фон.
5. ... и да эта программа может стать началом для игры, или моделирования идеального газа.

Благодарности

Компания Samsung Electronics выражает благодарность за участие в подготовке данного материала преподавателю ИТ ШКОЛЫ SAMSUNG Ильину Владимиру Владимировичу