Модуль 1. Основы программирования

Тема 1.7. Циклы. Цикл while. Основы графики в Android*

2 часа

Оглавление

1.7. Циклы. Цикл while

Когда в России в восьмидесятых годах только появились персональные компьютеры, практически каждый школьник - счастливый обладатель электронного чуда сразу писал программу на встроенном в ПЗУ Basic'е:

```
10 PRINT "I'M COOL PROGRAMMER"
20 GOTO 10
```

Те времена прошли. В языке Java, на котором мы пишем программы в IT ШКОЛЕ SAMSUNG, оператора goto вообще нет. Больше того, разработчики языка иронично оставили это слово в списке ключевых слов, так чтобы его практически никак нельзя было использовать в программах. Действительно, оператор безусловного перехода сильно портит читаемость программы. Для того, чтобы «заставить» компьютер повторять последовательность действий нужно использовать циклы.

Если бы циклов не было, то все программы… мгновенно заканчивали свою работу. Например, операционная система, взаимодействует с нами, именно в цикле ожидая наших команд, без него компьютер мгновенно выключался бы сразу после загрузки!

1.7.1. Цикл while

Цикл в программе организовать просто.

По структуре он очень похож на **if**.

```
//ПСЕВДОКОД
while (<условие>){
       <Действия> (выполняются пока условие истинно)
}
```

while в переводе с английского значит "до тех пор", "пока". Программа будет выполнять команды до тех пор, пока условие истинно. Проверяется условие, если оно истинно – выполняются команды и возвращаемся опять на проверку условия.

Блок команд, расположенных после **while**, называют **телом цикла**. Так же как и в условном операторе тело цикла может представлять из себя один оператор либо блок операторов (см. предыдущую тему 1.5).

Обратите внимание, как и в **if** точка с запятой после **условия** в скобках НЕ ставится!

```
while (<условие>)
```

Таким образом, команды могут не выполниться ни разу (если условие сразу ложно) или выполняться сколь угодно много раз. Естественно, предусматривают остановку этого процесса. Обычно условие зависит от значения переменной, а эта переменная изменяется командами.

Например, цикл в реальной программе может выглядеть так:

```
int x = 0;
while (x < 5){
       out.print(x);
       x++;
}
```

Сколько эта программа напечатает чисел?

На самом деле… всего одно число с ведущим нулем: 01234

Для того, чтобы программа напечатала раздельно 5 чисел нужно изменить строку вывода, скажем так:

```
out.print(x + " ");
```

Что напечатает строка, если добавить ее после закрывающей фигурной скобки?

```
out.print("x = " + x);
```

Значение переменной x будет равно 5 после цикла, именно при этом сравнение, которое проверяет цикл станет ложным. Поэтому будет выведена строка:

```
x = 5
```

Условия, как и в **if**, могут быть абсолютно любыми логическими выражениями, например, содержать логические операции **&&** и **||**.

1.7.2. Оператор break

У Java программистов есть способ выйти из цикла без учета условия в while – оператор break.

Например, программа

```
int x = 10;
while (x < 100){
       if (x == 2 * (x / 10) * (x % 10))
       {
            break;
       }
       x++;
}
```

выйдет из цикла, когда найдет первое двузначное число, равное удвоенному произведению своих цифр.

Не стоит злоупотреблять досрочным выходом из цикла! Можно, конечно написать так:

```
int i = 1;
while(true){
        out.println (i);
        i++;
        if (i == 10) break;
}
```

Значительно лучше читается простое и естественное

```
while(i <= 10)
```

без использования **break**. Такой вариант делает программу более понятной.

Следует учитывать, что break прерывает именно цикл. Но использовать его, конечно, стоит только в условном операторе, который находится в цикле.

Для того, чтобы хорошо овладеть применением циклов нужна практика и знание некоторых "секретов".

Например, обычно не стоит использовать условие на точное сравнение с заданным значением. Лучше использовать неравенства, которые задают допустимый интервал значений.

Например, выведем числа от 1 до 99 через пробел:

```
int x = 1;
while (x != 100){
        out.print(x + " ");
        x++;
}
```

Все работает правильно. Когда x станет равен 100 условие станет ложным и мы выйдем из цикла.

Но переформулируем задачу: вывести все **нечетные** числа от 1 до 99.

Казалось бы простейшее изменение

```
int x = 1;
while (x != 100){
        out.print(x + " ");
        x += 2;
}
```

приводит к ошибке! Теперь програма будет выводить нечетные числа практически до бесконечности. А если бы условие изначально было x < 100, программа работала бы корректно.

Чтобы сделать какие-то команды ровно N раз:

```
int i = 0;
while (i < N){
        //… команды выполнятся ровно N раз
        i++;
}
```

Нужно начать с 0 и поставить знак <. Можно и с единицы с операцией <=, но первый вариант используется значительно чаще.

Переменную **i**, которая с нуля увеличивается на единицу до некоторого значения обычно называют **счетчиком**.

Счетчики широко применяются в циклах. Например, часто в задачах необходимо считать число заданное **N раз**. В этом случае необходимо поместить команду считывания в цикл со счетчиком, который выполняется **N** раз.

Пример 1

Для демонстрации всех этих приемов решим задачу, которая предлагалась когда-то на Московской олимпиаде по программированию.

Оргкомитет Московской городской олимпиады решил организовать обзорную экскурсию по Москве для участников олимпиады. Для этого был заказан двухэтажный автобус (участников олимпиады достаточно много и в обычный они не умещаются) высотой 437 сантиметров. На экскурсионном маршруте встречаются N мостов. Жюри и оргкомитет олимпиады очень обеспокоены тем, что высокий двухэтажный автобус может не проехать под одним из них. Им удалось выяснить точную высоту каждого из мостов. Автобус может проехать под мостом тогда и только тогда, когда высота моста превосходит высоту автобуса. Помогите организаторам узнать, закончится ли экскурсия благополучно, а если нет, то установить, где произойдет авария.

Формат входных данных

Во входных данных сначала содержится число *N* (1 <= *N* <= 1000). Далее идут *N* натуральных чисел, не превосходящих 10000 - высоты мостов в сантиметрах в том порядке, в котором они встречаются на пути автобуса.

Формат выходных данных

В единственную строку нужно вывести фразу "`No crash`", если экскурсия закончится благополучно. Если же произойдет авария, то нужно вывести сообщение "`Crash` *k*", где *k* - номер моста, где произойдет авария. Фразы выводить без кавычек ровно с одним пробелом внутри.

Примеры

Входные данные	Выходные данные
1 763	No crash
3 763 245 113	Crash 2
1 437	Crash 1

Решение

В цикле N раз считываем числа и проверяем превосходит ли считанное число 437.

Если нет - выводим ответ и завершаем цикл при помощи break иначе увеличиваем счетчик и идём дальше.

```
int i = 1;
while(i < N){
	t = in.nextInt();
	if (t <= 437){
		out.println("Crash " + i);
		break;
	}
	i++;
}
```

После выхода из цикла нужно опять проверить, почему мы из него вышли (потому что доехали до конца или разбились в дороге):

```
if (t > 437){
	out.println("No crash");
}
```

На informatics эта задача имеет номер **1023**.

Задание 1.7.1

Для того, чтобы освоится с циклами стоит решить задачи informatics под номерами **112204, 112202, 112208, 112213, 112214**.

1.7.3. Основы графики в Android*

Основы рисования на Android. Класс Canvas

Очень помогает пониманию циклов визуализация. Например, рисование узоров. В IT ШКОЛЕ SAMSUNG мы будем рисовать на устройстве Android.

Читая дальше, помните, что программирование невозможно изучать последовательно. Многие вещи, которые могут сейчас показаться непонятными, будут подробно объяснены в дальнейшем.

Обычно, работая с программой на Android пользователь взаимодействует с элементами интерфейса (с объектами View). Многие из них, например, кнопки, списки, поля ввода разработаны создателями Android и программисту нужно просто их использовать в своих программах. Так же программист может создать свои нестандартные элементы. Для этого нужно создать подкласс класса View. Это обычный подход объектно-ориентированного программирования. Взять уже готовый класс за основу и изменить или добавить новые свойства.

Создадим свой собственный элемент интерфейса, на котором сможем рисовать все что угодно. Для этого переопределим в подклассе **View** метод **onDraw**. (Для работы нужно создать еще одну функцию – конструктор, мы поговорим о ней и об ее устройстве позднее). Это так называемый метод обратного вызова. Программисту нужно лишь определить сам метод, описать действия, которые он должен выполнять, но вызывать этот метод сам программист не должен. Ее вызывает система в тот момент, когда это необходимо. Например, функция onDraw вызывается тогда, когда нужно отрисовать объект, скажем, при старте программы.

Методы обратного вызова очень часто используются в програмировании под Android. Например, чтобы разместить наш View на Активности (экране приложения) необходимо в методе обратного вызова **onCreate** разместить код

	setContentView(new MyDraw(this)); Сам метод onCreate вызывать нигде не нужно. Система его вызовет сама при создании активности.

Устройство самой функции достаточно просто. Вызывая, система передает в нее объект класса **Canvas** – холст, этот класс и содержит функции рисования. На нем введена декартова система координат, левый верхний угол соответствует точке (0;0), соответственно ось ординат направлена необычно для математиков – вниз. Размер виджета, то есть области рисования, можно узнать вызовом функций getWidth и getHeight (в коде this.getWidth() и this.getHeight()).

Для рисовании линий можно использовать функцию (это текст из документации по классу Canvas на сайте http://developer.android.com, – пожалуй, основном источнике информации Android-разработчика)

void	drawLine(float startX, float startY, float stopX, float stopY, Paint paint) Draw a line segment with the specified start and stop x,y coordinates, using the specified paint.

Она принимает начальную и конечную точки и объект класса Paint, в котором содержаться характерстики линии (цвет, толщина и прочее). Мы подробно рассмотрим его на следующем занятии, а сейчас просто создадим его и передадим последним параметром.

Таким образом, для того, чтобы нарисовать диагональную линию через весь экран, нужно написать в функции **OnDraw** строки

```
Paint paint = new Paint();
canvas.drawLine(0, 0, canvas.getWidth(),canvas.getHeight(),paint);
```

Разлинуем холст в горизонтальную полоску. Для этого используем цикл:

```
int y = 0;
while (y < canvas.getHeight()){
       canvas.drawLine(0, y, this.getWidth(), y, paint);
       y += 10;
}
```

В итоге, полностью код класса **MyDraw**, который разлиновывает себя в горизонтальную полоску, выглядит так:

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

import android.content.Context;
import android.graphics.*;
import android.view.View;

public class MyDraw extends View {
       //конструктор– обсуждается в курсе позднее
       MyDraw(Context context){
```

```
            super(context);
        }
        @Override
        protected void onDraw(Canvas canvas) {
            Paint paint = new Paint();
            int y = 0;
            while (y < canvas.getHeight()){
                canvas.drawLine(0, y,
                this.getWidth(), y, paint);
                 y += 30;
            }
       }
}
```

Обратите внимание на объявление класса

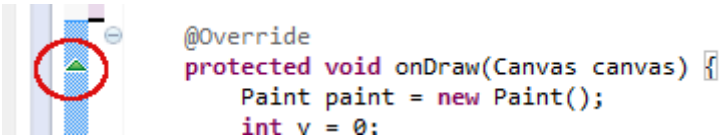
```
public class MyDraw extends View
```

Ключевое слово **extends** говорит о том, что MyDraw наследуется от View, то есть фактически является View, за исключением тех методов, которые переопределены. В нашем случае это View с особым onDraw.

Обратите внимание на строку **@Override**

Она не обязательна, но показывает что метод onDraw существует в суперклассе View – и в этом классе *переопределяется*.

Eclipse обозначает такие методы зеленым треугольником.



Он показывает, что метод именно переопределяется, а не создается новый. Это очень важный знак. Если ошибиться в названии, то будет создан совершенно другой метод и он не будет вызываться системой.

В следующей теме мы рассмотрим некоторые другие команды класса **Canvas**, а также некоторые возможности класса **Paint**, что, например, поможет сделать узоры цветными.

Рисование цветных узоров

Рассмотрим возможности класса *Paint*. До этого мы только создавали объект класса *Paint* и передавали в функцию рисования линии. На этом занятии мы настроим его.

Прежде чем что-то рисовать, нужно определить некоторые параметры: цвет заливки, толщину и вид линии контура и прочее. Эти параметры и содержат в себе объекты класса *Paint*.

Наиболее полезными методами класса *Paint* можно считать методы, задающие цвет, толщину линии контура и способ заливки:

void	setColor(int color) Set the paint's color.
void	setStrokeWidth(float width) Set the width for stroking.
void	setStyle(Paint.Style style) Set the paint's style, used for controlling how primitives' geometries are interpreted (except for drawBitmap, which always assumes Fill).

Например, для того, чтобы, нарисовать зеленую толстую линию, нужно перед рисованием создать объект класса Paint:

```
Paint paint = new Paint();
```

и настроить его так:

```
paint.setColor(Color.GREEN);
paint.setStrokeWidth(5);
```

а после этого рисовать:

```
canvas.drawLine(…,paint);
```

Особых комментариев требует метод **`setStyle`**.

Ему передается число, которое задает способ заливки. Чтобы не запоминать конкретные числовые значения разработчики создали константы с говорящими именами. Эти константы расположены в самом классе Paint в перечисление (enum) Style. Поэтому их полные имена достаточно длинны, например **`Paint.Style.FILL_AND_STROKE`**.

Собственно, имя константы записывается заглавными буквами, это общепринятый стиль для именования констант в языке Java.

Возможные значения:

`Paint.Style.FILL`	рисование с заливкой
`Paint.Style.STROKE`	рисование только контура
`Paint.Style.FILL_AND_STROKE`	рисование и контура и заливки

Заливка используется для рисования замкнутых фигур, например, окружностей или прямоугольников.

Примеры методов рисования фигур класса Canvas

В классе *Canvas*[1] реализовано много различных методов рисования кроме рисования линий.

На этом занятии мы рассмотрим рисование окружностей и прямоугольников, а на одном из следующих поговорим о других возможностях класса Canvas.

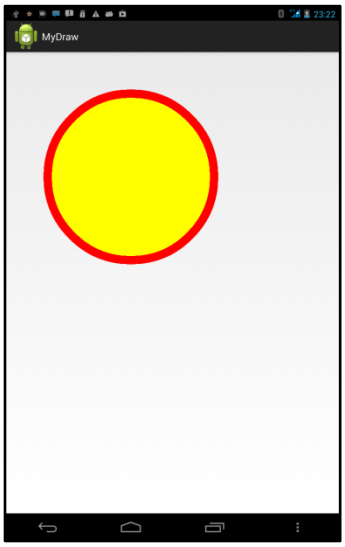
Для рисования окружности можно использовать метод:

void	drawCircle(float cx, float cy, float radius, Paint paint) Draw the specified circle using the specified paint.

Он принимает в качестве параметров координаты центра, радиус и объект класса Paint. Например, чтобы нарисовать желтый круг с красным утолщенным контуром нужно написать:

```
Paint paint = new Paint();
paint.setColor(Color.YELLOW);
paint.setStyle(Paint.Style.FILL);
canvas.drawCircle(300, 300, 200, paint);
paint.setColor(Color.RED);
paint.setStyle(Paint.Style.STROKE);
paint.setStrokeWidth(20);
canvas.drawCircle(300, 300, 200, paint);
```

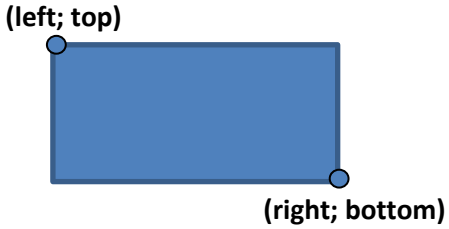
Так же можно использовать цвета, определенные в ресурсах (в xml файлах).



Для рисования прямоугольников можно использовать функцию, которой передаются координаты диагонали прямоугольника:

void	drawRect(float left, float top, float right, float bottom, Paint paint) Draw the specified Rect using the specified paint.

Стороны прямоугольника будут параллельны осям координат. Соответственно, это координаты левой верхней и правой нижней вершины:



[1] Полную документацию по классу *Canvas* можно найти на сайте http://developer.android.com, на странице http://developer.android.com/reference/android/graphics/Paint.html)

Например, нарисовать синий квадрат можно так:

```
Paint paint = new Paint();
paint.setColor(Color.BLUE);
paint.setStyle(Paint.Style.FILL);
canvas.drawRect(100, 100, 300, 300, paint);
```

Задание 1.7.2

1. Разлинуйте экран Android-приложения в диагональную полоску.
2. Придумайте и реализуйте собственный узор из линий.

Благодарности

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