Мобильная разработка

Урок 3

**Циклы**

Для того чтобы «заставить» компьютер повторять последовательность действий, нужно использовать циклы. Цикл в программе организовать просто. По структуре он очень похож на if.

В Java выделяют 4 вида циклов:

1. Цикл while
2. Цикл do…while
3. Цикл for
4. Цикл for each**.** Его пока рассматривать не будем, рассмотрим не следующем уроке.

**Цикл while**

Слово 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 программа переходит опять на проверку условия.

Блок команд, расположенных после while, называют телом цикла. Т

Запомните: суть оператора while: он повторяет действие(-я) до тех пор, пока условие истинно.

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

Пример бесконечного цикла

*while (true) {*

*System.out.print(i + " ");*

*i++;*

*}*

**Цикл do...while**

цикл while сначала проверяет условие, потом выполняет код (тело цикла), а do..while сначала выполняет тело цикла, а потом проверяет условие, поэтому тело цикла выполнится, как минимум, 1 раз.

### **Цикл for**

Цикл for еще называют цикл "со счетчиком". Когда Вы заранее знаете сколько раз повторится одно и то же действие, необходимо использовать цикл for.

Первая часть объявления цикла - int i = 0 создает и инициализирует счетчик i. Счетчик необязательно должен представлять тип int. Это может быть и любой другой числовой тип, например, float. Перед выполнением цикла значение счетчика будет равно 1. В данном случае это то же самое, что и объявление переменной.

Вторая часть - условие, при котором будет выполняться цикл. В данном случае цикл будет выполняться, пока i не достигнет 9.

И третья часть - приращение счетчика на единицу. Опять же нам необязательно увеличивать на единицу. Можно уменьшать: i--.

Пример цикла без инициализации и счестчика:

### Операторы continue и break

Оператор break позволяет выйти из цикла в любой его момент, даже если цикл не закончил свою работу. Оператор break не проектировался в качестве нормального средства завершения цикла. Эту цель обслуживает условное выражение заголовка цикла. Оператор break следует использовать для прерывания цикла только тогда, когда возникают некоторые специальные ситуации.

Оператор continue позволяет перейти к следующей итерации цикла в любой момент текущей итерации:

Иногда возникают ситуации, когда требуется выход из многократно вложенных циклов.

Для решения подобных задач в Java применяется расширенная форма инструкции break, используя которую можно выйти за пределы одного или нескольких блоков кода.

Выход из вложенных циклов с помощью break с отметкой:

**Вложенные циклы**

Циклы можно вкладывать друг в друга. С помощью вложенных циклов решаются самые разные задачи.

Для наглядности, давайте выведем таблицу умножения.

Пример кода:

*public static void main(String[] args) {*

*for (int i = 1; i < 10; i++) {*

*for (int j = 1; j < 10; j++) {*

*System.out.printf("%2d ", i \* j);*

*}*

*System.out.println();*

*} }*

**Область видимости переменной цикла**

Пример кода:

*for (int i = 10; i > 0; i--){*

*out.println(i);*

*}*

*out.println(i); // ОШИБКА! Переменная i здесь не существует*

**Практическое занятие**

Scanner sc = new Scanner(System.in);

int a = sc.nextInt();

int b = sc.nextInt();

int c = 0;

int i = 0;

while(i < abs(b)){

c = c + abs(a);

i++;

}

System.out.println(a \* b > 0 ? c : -c);

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.

Создаем свой класс-канву для рисования. Класс является наследником View, т. е. При создании нужно указать SuperClass. После создания класса, AndroidStudio предложит создать конструктор — создаем.

3 В нашем классе MyView нужно создать метод, в котором будут написаны код для отрисовки чего-либо. У таких методов название всегда стандартное.

@Override

protected void onDraw(Canvas canvas) {

super.onDraw(canvas); //Здесь располагаются команды рисования

}

4 Чтобы разместить наш View на активности (экране приложения), необходимо в методе обратного вызова onCreate() класса главной активности MainActivity

разместить код:

setContentView(new MyDraw(this));

onCreate() и onDraw() — так называемые методы обратного вызова. Программисту нужно лишь определить сам метод, описать действия, которые он должен выполнять, но вызывать этот метод сам программист не должен. Его вызывает система в тот момент, когда это необходимо. Например, метод onDraw() вызывается тогда, когда нужно отрисовать объект, скажем, при старте программы.

Методы обратного вызова очень часто используются в программировании под Android. Например, cам метод onCreate() вызывать нигде не нужно. Система его вызовет сама при создании активности.

5 Устройство самой функции onDraw() 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 холста, то есть области рисования, можно узнать вызовом функций getWidth() и getHeight().

Для рисовании линий можно использовать метод:

@Override

protected void onDraw(Canvas canvas) {

canvas.drawCircle(x, 300, 20, paint);

// готовим x для следующего кадра

x += 0.5f;

invalidate();

}