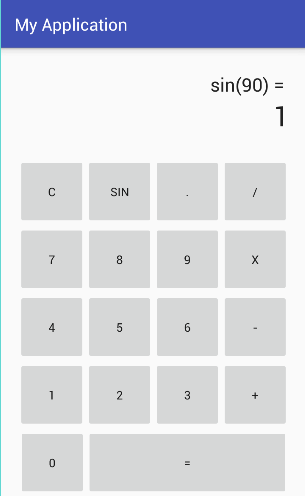
Задание 1

Реализуйте приложение Калькулятор. Имеется набор кнопок, циферблат. Калькулятор позволяет вычислять (сумму, разность, произведение и частное). Предусмотреть обработку ситуации деления на ноль. Кроме этого, добавьте функции вычисления (вариант 1: вычисление синуса). 

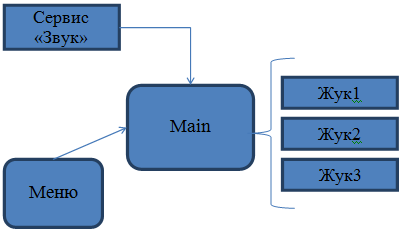


Задание 2:

Необходимо написать игру. Сюжет: множество насекомых ползают по экрану по сложной траектории. Задача игрока уничтожить за отведенное время как можно больше насекомых. За промахи игроку начисляется очки отнимаются. Игра должна сопровождаться звуковыми эффектами. Варианты:

1. В качестве насекомых (осы, и пауки)
2. В качестве насекомых (Божьи коровки, мухи)
3. В качестве насекомых (тараканы, и пчелы)
4. В качестве насекомых (осы, и мухи)
5. В качестве насекомых (майские жуки, и тарантулы)
6. В качестве насекомых (тараканы, и пауки)

Описание выполнения:   
Первым шагом необходимо определить архитектуру приложения. Пусть архитектура будет построена следующим образом.



Звуки и картинки насекомых следует брать в интернете. Следует дополнить этот набор картинок, добавив «красные пятна крови» (эти картинки будут появляться при «попадании» игроком).   
Время раунда игры составляет 2 минуты.   
При запуске приложения открывается меню, позволяющее вести имя игрока, просмотреть рекорд, задать количество жуков и приглашение к началу игры.   
Основной класс приложения (main) будет заниматься прорисовкой всех жуков. Состояние всех жуков хранится в глобальном массиве структур(*Mass*) . Структура должна хранить координаты жука *(x,y)*, направление движения *(dx,dy)*,флаги(*жив/мертв*), и байт, соответствующей отображаемой картинке (т.к. жуки могут быть разными).   
Работа класса происходит в бесконечном цикле, считывание информации из массива и корректное его отображение. Не рекомендуется заниматься прорисовкой из других потоков, т.к. это может привести к ошибкам.   
В других потоках, соответствующих жукам, мы будем заниматься перерасчетом состояний отдельно каждого элемента глобального массива. При смерти жука, объект не уничтожается, а некоторое время просто не отображается на экране. В противном случае нам придется создавать каждый раз новых жуков, что потребует дополнительной памяти.   
Отдельно в классе сервисе следует размещать мелодии в игре. Быстрые мелодии (хруст жука) менее 1-2 сек вполне можно размещать в классе main. Нужно не забыть отключать звуки при остановке приложений и восстанавливать при возобновлении работы приложения(используйте функции *onPause* и *onResume*).   
Используйте класс Preferences для сохранения результатов игры.   
Проверку попадания нажатия по жуку следует сделать так:   
for (int i=0;i<N;i++){ //просматриваем весь массив структур

if (Mass[i].status==Alive){ // Если жук жив и …

if ((Mass[i].x1<e;x)&&(x≤Mass[i].x2)&& (Mass[i].y1≤y)&&(y≤Mass[i].y2))

// координаты нажатия x,y по экрану лежат на картинке, где x1 x2 –границы

// картинки жука и x1≤x≤x2 и y1≤y≤y2

{

//Попали по жуку

}

}

}

Допускается использовать вместо класса Panel, класс SurfaceHolder.

**Задание 3:**

Реализуйте прыгающий <объект>. Начальная позиция появления объекта определяется нажатием пальца по экрану. Объект падает с ускорением вниз, отражаясь от нижней границы экрана, и с незначительным смешением вправо. С каждым разом высота отскока объекта уменьшается, и когда объект практически остановится, он пропадает с экрана. Объектов может быть несколько, поэтому каждый должен обрабатываться в отдельном потоке приложения. (<объект> - Буква Вашей фамилии, после каждого нажития буква меняется на следующую, по окончанию Вы опять возвращаетесь на первую букву).  
Для примера <объект> - цифра восемь, направление падения схематично показано красным.

**Фамилия - Ровнов**

