App programming

Programming a smartphone app is a bit different from PC programming.

- We don't have a compiler on the phone.
- There is no keyboard.
- We can use a variety of sensors (camera, accelerometer, compas, light sensor).

Cross-compiling

Write source code on PC.

Use a special compiler running on PC to create code for phone.

Cross-compiler

Package to install on the phone.

A cross-compiler is a compiler that generates code for a different platform from the one the compiler is running on.

Same for Arduino and other embedded platforms.

Compiled code needs to be copied to the phone to run.

Testing

It's rather annoying to test your code on the phone. There is no console output, no keyboard, and every time you change the code, you have to copy it to the phone again.

Solution: Use an emulator for most testing. It provides a simulation of the phone environment on the PC, so that you can use `println` for debugging.

CS109 Mini-App framework

The iOS and Android software development kits (SDKs) are very large, complicated, and not suitable for this course.

Instead, we will use the CS109 mini-app framework. It is really simple to use, but teaches many concepts about app programming for smartphones.

- Currently only available for Android (a beta version of a Kotlin compiler for iOS exists, so in the future maybe...)
- Your mini-apps cannot be installed directly on the phone. You will need the CS109 App to run them.
- Only canvas-based mini-apps are possible, and many phone features are currently not accessible (e.g. camera).

How to compile, test, and install

Write your source code: `basic.kt`

Cross-compile: `ktc-dex basic.kt`

This creates two files: `basic.dex` (for the phone) and `basic.jar` (for the emulator).

Test on the emulator: `kt-emulator basic.jar`

On the phone, install the CS109 App (search for `KAIST CS109` on Play store). You need Android 4.2 or higher.

Transfer `basic.dex` to the phone (e.g. mail it to yourself).

In CS109 App, choose `Load new mini-app` from menu, and pick the dex file.

Security

Your mini-app runs on the phone hardware, just like any other Android app. It has full access to Android features (only restricted by the CS109 App permissions).

Therefore: do not run DEX files you get from unrealiable sources using the CS109 App. Use the app only to run your own mini-apps.

The CS109 App has very limited permissions, so not much can go wrong.

The mini-app framework

Smallest possible mini-app (`basic.kt`):

```
class Main(val ctx: Context) : MiniApp {
  init {
    ctx.setTitle("Demo #1")
  }
  override fun onDraw(canvas: Canvas) {
    canvas.clear(Color(255, 255, 192))
    canvas.setColor(Color.BLUE)
    canvas.setFont(48.0)
    canvas.drawText("CS109", canvas.width / 2.0,
      200.0, TextAlign.CENTER)
    canvas.drawCircle(canvas.width / 2.0, 400.0,
      60.0)
  }
}
```

`Main`, `MiniApp`: required names

`Context`: object providing access to Android features

The Main class

The `Main` class must `override` the `onDraw` method. It's purpose is to draw the graphics for the app.

The `Context` provides some services, for instance:

- `setTitle` to set (and change) the title of the mini-app;
- `width` and `height` to obtain the size of the screen;
- `toast` to show a message;
- `update` to make sure that the screen will be drawn again;
- `onTap`, `onDoubleTap`, `onFling` to handle finger input;
- `onGravity` and `onLight` to use sensors.

Other `Context` methods (see documentation):

- `showMessage`, `askYesNo`, `inputString` for dialogs;
- `after` for animation;
- `createMenu` to make a menu.

Finger input — event processing

You cannot simply use `waitMouse` to wait patiently—you need to be prepared to handle finger taps at any time.

```
class Main(val ctx: Context) : MiniApp {
  private var lastX = 0.0; private var lastY = 0.0
  init {
    ctx.setTitle("Tap and fling demo")
    ctx.onTap { x, y -> tapped(x, y) }
  }
  fun tapped(x: Double, y: Double) {
    lastX = x; lastY = y
    ctx.update()
  }
  override fun onDraw(canvas: Canvas) {
    canvas.clear(Color(255, 255, 192))
    canvas.setColor(Color.BLUE)
    canvas.drawCircle(lastX, lastY, 30.0)
  }
}
```

function object

tell system that screen needs to be drawn again

graphics depends on tap

Using sensors

Smartphones contain a number of sensors. Mini-apps can use light sensor (not in all phones) and accelerometer.

```
class Main(val ctx: Context) : MiniApp {
  var gravity = arrayOf(0.0, 0.0, 0.0)
  init {
    ctx.setTitle("Gravity sensor demo #1")
    ctx.onGravity { x, y, z -> updateGravity(x, y, z) }
  }

  fun updateGravity(x: Double, y: Double, z: Double) {
    gravity = arrayOf(x, y, z)
    ctx.update()
  }

  override fun onDraw(canvas: Canvas) {
    // show gravity value on screen
  }
}
```

function object is called every time new sensor value is available