FIT2081 w1

Wednesday, 1 March 2023 7:01 PM

FIT2081

weekly
→ pre reading ( )
→ workshop ( )
→ Lab (thursday)

→ developer.android.com

→ documentation

1.

$$\begin{tikzcd} & \text{Native} \\ \boxed{\text{Apps}} \arrow[ur] \arrow[r] \arrow[dr] & \text{Mobile Web} \\ & \text{Hybrid} \end{tikzcd}$$

2. SDK → Software Development Kit
   - → bundle of software components necessary to develop & deploy on a development platform

3. Application Programming Interface (API) = Class Library
   - → code
   - → not written by the user
   - → can be called to perform common but complicated task

4. Integrated Development Environment (IDE)
   - → software environment
   - → contains all the tools that developers need to develop applications

5. Native Apps

   Characteristics
   - → app's compiled code runs directly on a device's platform
     - → Android, IOS
   - → built using SDK & languages recommended by the vendor
     - → Java + Android Studios
       - → Windows, Mac OSX, Linux
     - → Objective C / Cocoa / Swift + XCode IDE

→ requiring → scarce

→ sophisticated

→ costly

$\boxed{\text{developers,}}$ one set per target platform

↓

have complete access to device features

→ Mac OSX only

6. Mobile Web Apps

→ website designed for smart device display
→ accessed by a device's browser

→ requires → abundant → cheaper / less sophisticated web developers
→ same 'app' functions on all platforms
→ developer access to → device features / Native API depends on
quality, completeness & cross browser consistency of the browsers JS → Native API bridge

7. Hybrid Apps
→ written using a language & development environment other than the recommended languages for the platform
→ deployed as a Native App
eg: ① Thin native shell app + web app
* check slide examples
→ JS → Native API bridge
→ native component + platforms web rendering engine = UI

② Cross compiler
→ convert code into a Native app executable for each required target platform

→ requires abundant, less sophisticated, cheap web devs
→ same app functions on all platforms that the hybrid IDE creates thin clients for
→ developer access to → device features / Native API depends on
quality, completeness & cross browser consistency of the browsers JS → Native API bridge

What 3rd party software provide you
→ single code base to deploy to multiple platforms
→ creates & constantly updates the JS → Native API $\boxed{\text{bridge}}$
↓
$\boxed{\text{quality, completeness + cross browser consistency is critical}}$

8. Table of comparison
9. Issues
i) Cost : write once } cheaper
web dev
↓
however : hybrid apps are at the mercy of quality completeness & consistency of their IDE's thin native clients

ii) UI Look & Feel (L & F)
→ web LF ≠ Android LF

→ different user experience

iii) Offline storage

→ inbrowser : limited storage span < local file storage

↓

local storage exists & is improving

↓

HTML5 有 application caching

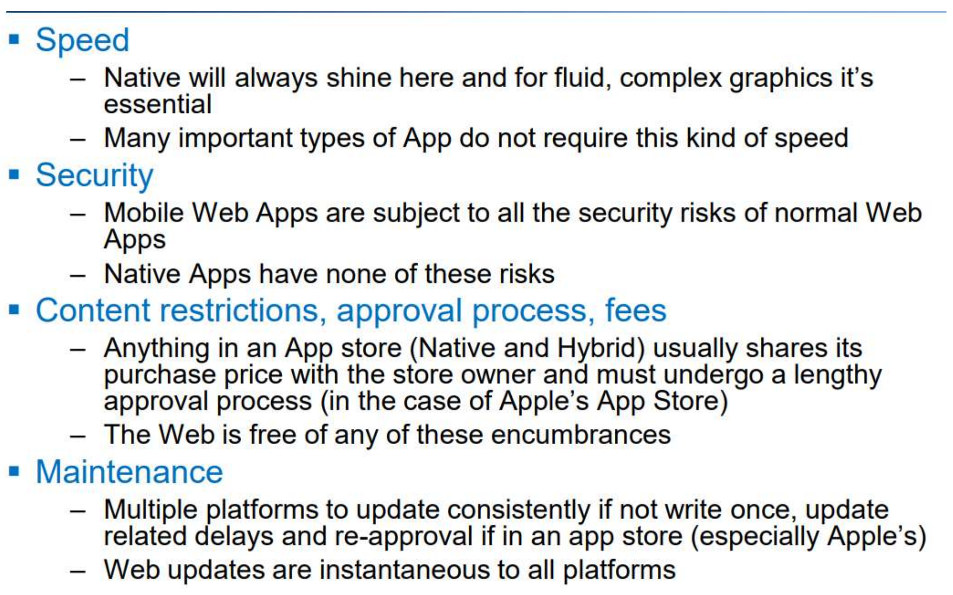
iv) Discoverability & installation

→ home screen bookmarks

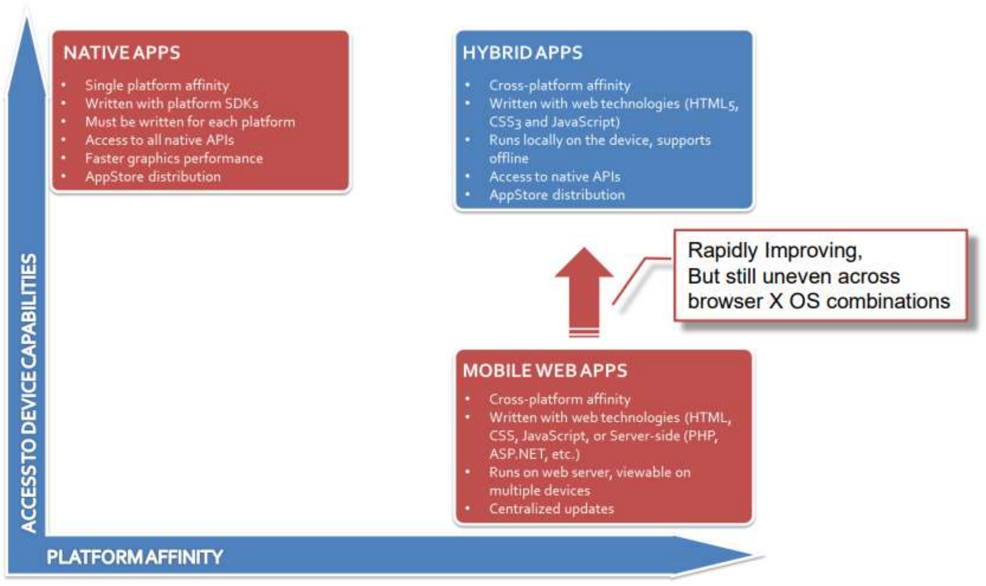
→ installation vs no installation

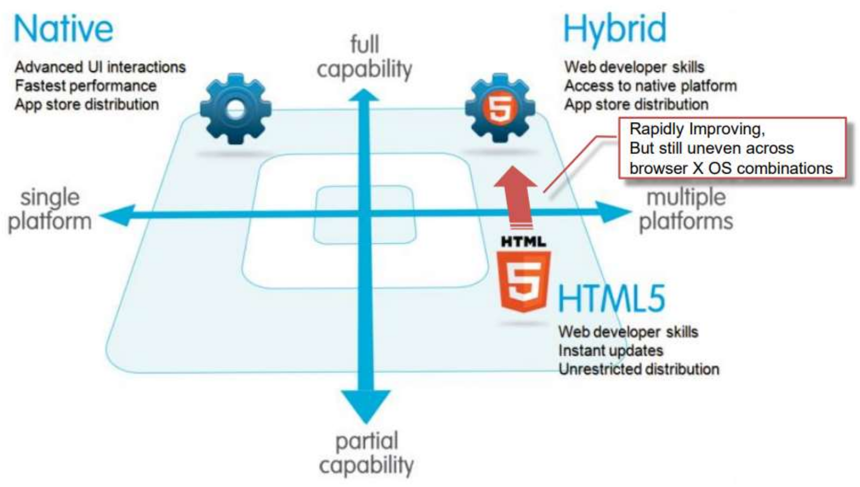
→ app store vs URLs

v) Speed



- Speed
  - Native will always shine here and for fluid, complex graphics it's essential
  - Many important types of App do not require this kind of speed
- Security
  - Mobile Web Apps are subject to all the security risks of normal Web Apps
  - Native Apps have none of these risks
- Content restrictions, approval process, fees
  - Anything in an App store (Native and Hybrid) usually shares its purchase price with the store owner and must undergo a lengthy approval process (in the case of Apple's App Store)
  - The Web is free of any of these encumbrances
- Maintenance
  - Multiple platforms to update consistently if not write once, update related delays and re-approval if in an app store (especially Apple's)
  - Web updates are instantaneous to all platforms

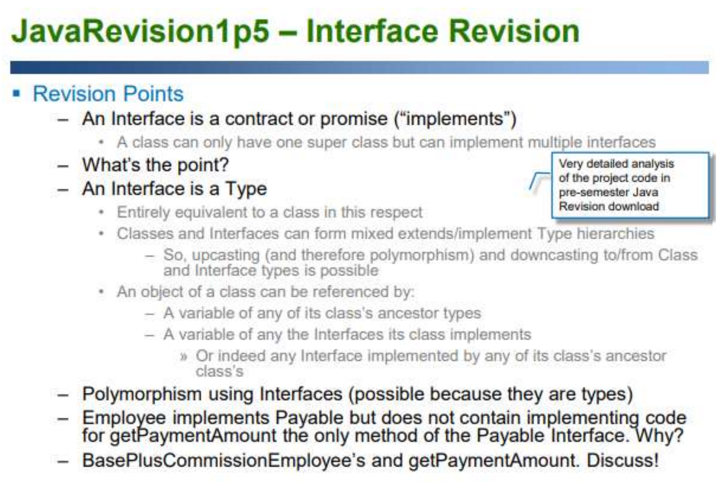

NATIVE APPS
Single platform affinity
Written with platform SDKs
Must be written for each platform
Access to all native APIs
Faster graphics performance
AppStore distribution
HYBRID APPS
Cross-platform affinity
Written with web technologies (HTML5, CSS3 and JavaScript)
Runs locally on the device, supports offline
Access to native APIs
AppStore distribution
Rapidly Improving, But still uneven across browser X OS combinations
MOBILE WEB APPS
Cross-platform affinity
Written with web technologies (HTML, CSS, JavaScript, or Server-side (PHP, ASP.NET, etc.)
Runs on web server, viewable on multiple devices
Centralized updates
ACCESS TO DEVICE CAPABILITIES
PLATFORM AFFINITY



JavaRevision1 – Inheritance Revision

- Revision Points
  - Super and Sub Classes ("extends")
  - Abstract classes and methods
  - @Override
    - And overriding itself
    - Calling super
    - Calling super.someOverridenMethod(…)
      - Leveraging possibilities
  - Instance variables should be private
    - They are (directly) accessible in subclasses
    - But public methods are so their accessors and mutators can be called in subclasses to manipulate them
  - A class inherits from all its ancestors not just its parent
  - toString(…)
  - Polymorphism using Inheritance (Upcasting)
    - Code
  - Downcasting

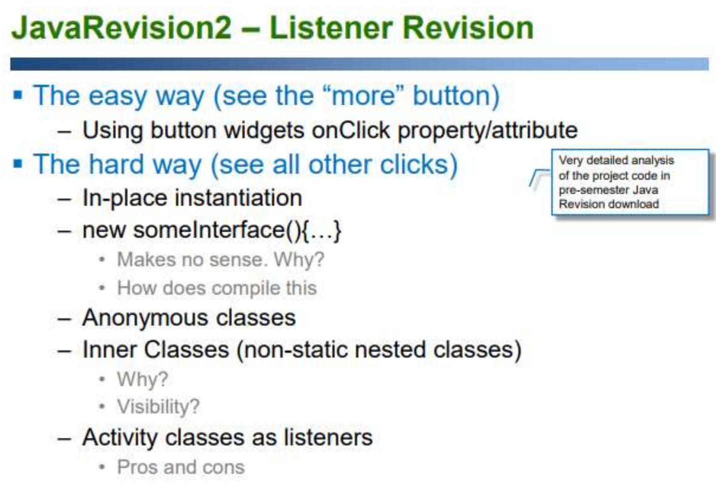
Very detailed analysis of the project code in pre-semester Java Revision download



JavaRevision1p5 – Interface Revision

- Revision Points
  - An Interface is a contract or promise (“implements”)
    - A class can only have one super class but can implement multiple interfaces
  - What’s the point?
  - An Interface is a Type
    - Entirely equivalent to a class in this respect
    - Classes and Interfaces can form mixed extends/implement Type hierarchies
      - So, upcasting (and therefore polymorphism) and downcasting to/from Class and Interface types is possible
    - An object of a class can be referenced by:
      - A variable of any of its class’s ancestor types
      - A variable of any the Interfaces its class implements
        - » Or indeed any Interface implemented by any of its class’s ancestor class’s
  - Polymorphism using Interfaces (possible because they are types)
  - Employee implements Payable but does not contain implementing code for getPaymentAmount the only method of the Payable Interface. Why?
  - BasePlusCommissionEmployee’s and getPaymentAmount. Discuss!

Very detailed analysis of the project code in pre-semester Java Revision download



JavaRevision2 – Listener Revision

- The easy way (see the “more” button)
  - Using button widgets onClick property/attribute
- The hard way (see all other clicks)
  - In-place instantiation
  - new someInterface(){…}
    - Makes no sense. Why?
    - How does compile this
  - Anonymous classes
  - Inner Classes (non-static nested classes)
    - Why?
    - Visibility?
  - Activity classes as listeners
    - Pros and cons

Very detailed analysis of the project code in pre-semester Java Revision download