Building stable and flexible libraries

@KeithYokoma - Drivemode, Inc.

potatotips #12

KeithYokoma

Keishin Yokomaku
Drivemode, Inc.
Android Engineer
GitHub: https://github.com/KeithYokoma
e-Book: http://amzn.to/1mZNydv

Agenda

- Stability
- Flexibility

Make libraries STABLE

Make libraries STABLE

- Entity class declaration
- Multi-thread compatibility
- Lifecycle management

Make libraries STABLE

- Entity class declaration
  - Don’t

```
void setToken(String token, String type, String refresh, long by);
```

  - Do

```
void setToken(AccessToken token);
```

Make libraries STABLE

- Entity class declaration

```
void setToken(String token, String type, String refresh, long by);
```

  - Hard to remember the type of args
  - Not Type-Safe(ref. Effective Java)

Make libraries STABLE

- Entity class declaration

```
void setToken(AccessToken token);
```

  - Easy to remember the type of args
  - Type-Safe

Make libraries STABLE

- Multi-thread compatibility
  - Synchronization
  - Immutable entity
  - Thread pool and callback lifecycle
  - Singleton implementation

Make libraries STABLE

- Multi-thread compatibility
  - Synchronization
    - “synchronized” block
    - Synchronization utils(CyclicBarrier, …)
    - Atomicity(AtomicInteger, …)
    - “volatile” field

Make libraries STABLE

- Multi-thread compatibility
  - Immutable entity
    - Immutable entity is thread safe

Make libraries STABLE

- Multi-thread compatibility
  - Thread pool and callback lifecycle
    - Reduce thread initialization cost
    - Align callback lifetime with "Context"
    - Do NOT callback to dead object

Make libraries STABLE

- Multi-thread compatibility
  - Singleton implementation
    - Be aware of “Lazy Initialization”

Case Study
Multi-thread compatibility

```java
// NOT thread safe!!
public class Singleton {
    private static Singleton sInstance;

    public static Singleton getInstance() {
        if (sInstance == null) {
            sInstance = new Singleton();
        }
        return sInstance;
    }
}
```

Make libraries STABLE

- Multi-thread compatibility
    - Singleton implementation
        - “synchronized” block
        - Double checked locking
        - Initialization on demand holder

Case Study
Multi-thread compatibility

```
private static Singleton sInstance;

public static synchronized Singleton getInstance() {
   if (sInstance == null) {
       sInstance = new Singleton();
   }
   return sInstance;
}
```

Case Study
Multi-thread compatibility

```java
private static volatile Singleton sInstance;

public static Singleton getInstance() {
   if (sInstance == null) {
       synchronized (Singleton.class) {
           if (sInstance == null) {
               sInstance = new Singleton();
           }
       }
   }
   return sInstance;
}
```

Case Study
Multi-thread compatibility

```
static class Holder {
   public static final Singleton SINGLETON = new Singleton();
}

public static getInstance() {
   return Holder.SINGLETON;
}
```

Make libraries STABLE

- Lifecycle management
  - Object lifetime alignment

Make libraries STABLE

- Lifecycle management
  - Object lifetime alignment
    - Lifecycle methods of various “Context”
      - onCreate/onDestroy
      - onStart/onStop, onResume/onPause

Make libraries STABLE

- Lifecycle management
  - Object lifetime alignment
    - Naming convention
      - add/remove, register/unregister
      - start/finish, initialize/destroy

Make libraries FLEXIBLE

Make libraries FLEXIBLE

- Annotations vs Listeners
- Customizable resources
- Split package by domain

Make libraries FLEXIBLE

- Annotations
  - ✓ Fast and easy development for client
  - ✓ Automatic code generation(with apt)
  - ✗ Slow(both runtime and apt takes time)
  - ✗ Hard to dig into library itself

Make libraries FLEXIBLE

- Listeners
  - ✓ Faster than annotations(runtime)
  - ✓ Simple architecture
  - ✗ Client should maintain the lifetime

Make libraries FLEXIBLE

- Annotations and Listeners
  - Do NOT call methods of dead object

Make libraries FLEXIBLE

- Customizable resources
  - If the library has UI resources…
    - Theme should be customizable
    - What about layout resources?

Make libraries FLEXIBLE

- Customizable resources
  - At least you need to…
    - Define ID resources that the library uses
    - Otherwise layout may not be customized

Make libraries FLEXIBLE

- Split package by domain
  - Avoid exceeding 65k method limit
  - Less effort to strip out codes not used

Make libraries FLEXIBLE

- Split package by domain
  - e.g. Guava
    - guava, guava-gwt, guava-annotations, …
  - e.g. Google Play Services 6.5
    - play-services, play-services-wearable, …

“Never make the client do anything the library can do for the client.”

–Joshua Bloch

Building stable and flexible libraries

@KeithYokoma - Drivemode, Inc.

potatotips #12