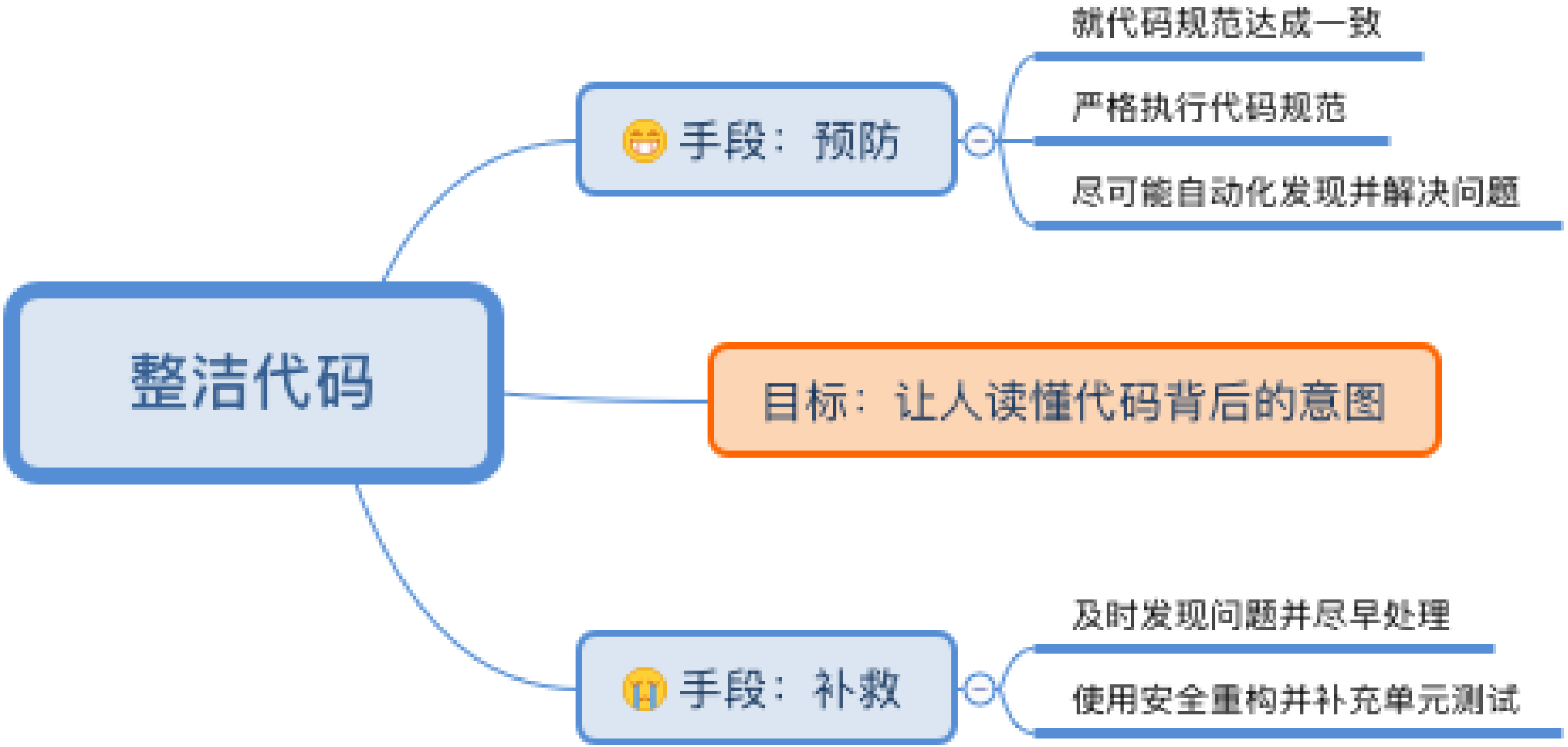
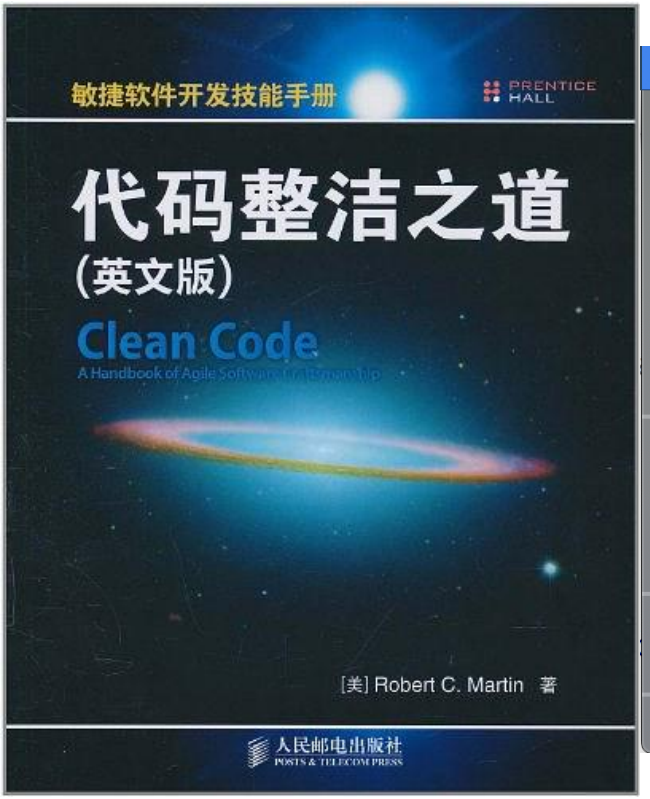
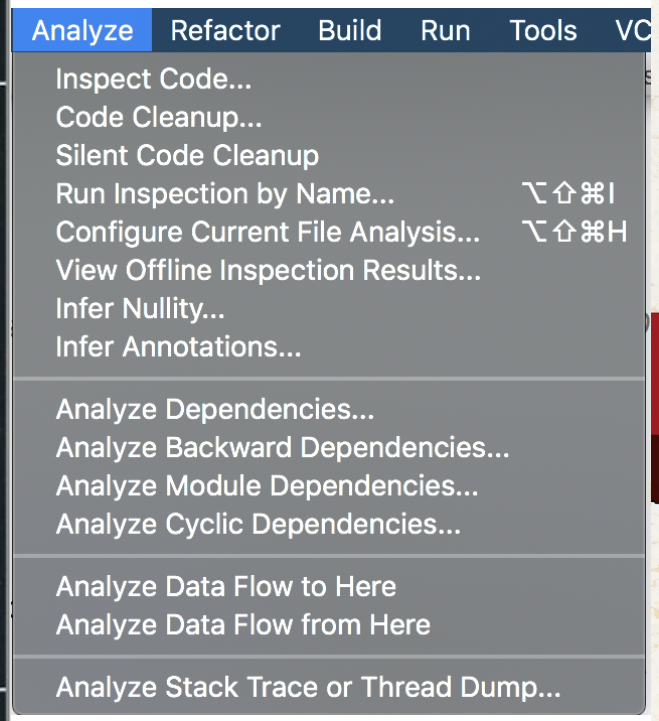
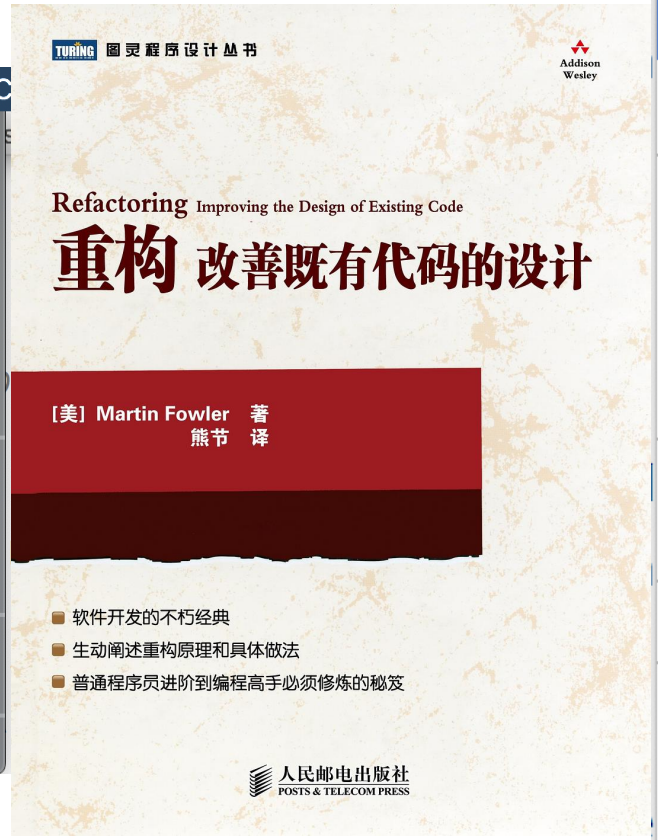
整洁代码&重构基础

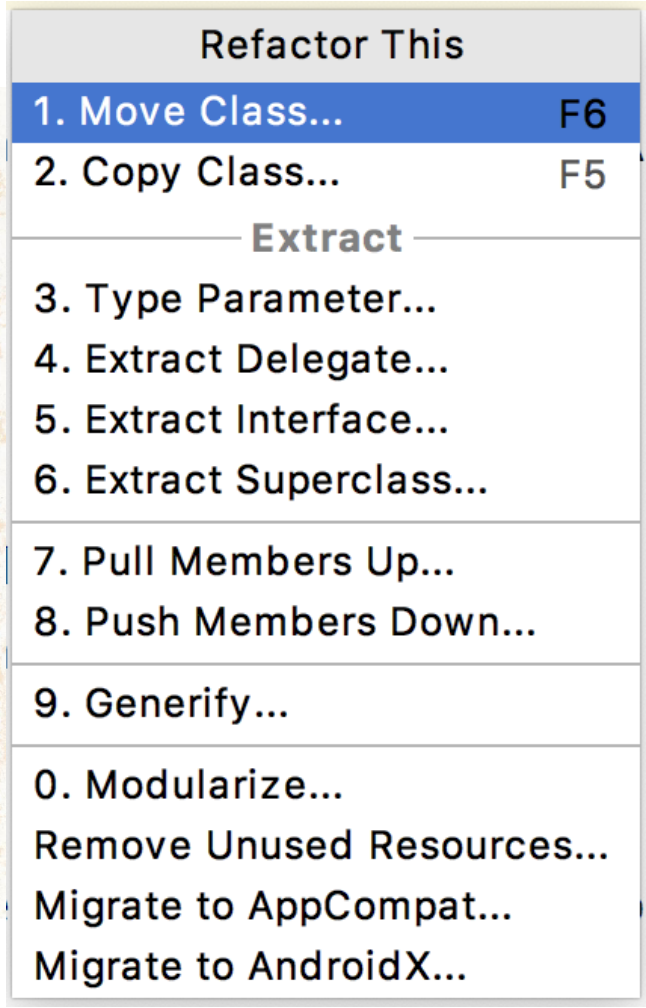
CAC@OPPO by 黄俊彬 & 覃宇


整洁代码
手段：预防
就代码规范达成一致
严格执行代码规范
尽可能自动化发现并解决问题
目标：让人读懂代码背后的意图
手段：补救
及时发现问题并尽早处理
使用安全重构并补充单元测试


敏捷软件开发技能手册
PRENTICE HALL
代码整洁之道
(英文版)
Clean Code
[美] Robert C. Martin 著
人民邮电出版社
POSTS & TELECOM PRESS


Analyze
Refactor
Build
Run
Tools
VC
Inspect Code...
Code Cleanup...
Silent Code Cleanup
Run Inspection by Name... ⌥⇧⌘I
Configure Current File Analysis... ⌥⇧⌘H
View Offline Inspection Results...
Infer Nullity...
Infer Annotations...
Analyze Dependencies...
Analyze Backward Dependencies...
Analyze Module Dependencies...
Analyze Cyclic Dependencies...
Analyze Data Flow to Here
Analyze Data Flow from Here
Analyze Stack Trace or Thread Dump...


TURING 图灵程序设计丛书
Addison Wesley
Refactoring Improving the Design of Existing Code
重构 改善既有代码的设计
[美] Martin Fowler 著
熊节 译
软件开发的不朽经典
生动阐述重构原理和具体做法
普通程序员进阶到编程高手必须修炼的秘笈
人民邮电出版社
POSTS & TELECOM PRESS


Refactor This
1. Move Class... F6
2. Copy Class... F5
Extract
3. Type Parameter...
4. Extract Delegate...
5. Extract Interface...
6. Extract Superclass...
7. Pull Members Up...
8. Push Members Down...
9. Generify...
0. Modularize...
Remove Unused Resources...
Migrate to AppCompat...
Migrate to AndroidX...

大家说...

1. 方法太长，类太大
2. 命名太南了
3. if/else/for嵌套太深
4. 注释和文档问题
5. 滥用设计模式
6. 匿名内部类和回调地狱
7. 并发和多线程问题

方法太长，类太大

- 代码冗余/重复，不是最优实现
- 不敢删除遗留代码，怕出问题
- 复制粘贴造成的帝王类
- 类职责太多(违反 SRP)
- 业务逻辑放在了 UI 中

？过长方法和过大类曾经给你带来什么困扰？通过什么方式解决？

（讨论时间：5分钟）

坏味道在哪❓

```
void parseBetterJSAlternative(String code) {
    String[] REGECES = {};
    String[] statements = code.split(" ");
    String[] tokens = {};
    for(String regex: Arrays.asList(REGECES)) {
        for(String statement:Arrays.asList(statements)) { //... }
    }

    String[] ast={};
    for(String token:Arrays.asList(tokens)) { //lex ...}

    for(String node:Arrays.asList(ast)) { //parse ...}
}
```

坏味道：方法抽象超过一层。补救办法：提取方法

```
String[] tokenize(String code) {
    String[] REGECES = {};
    String[] statements = code.split(" ");
    String[] tokens = {};
    for(String regex: Arrays.asList(REGECES)) {
        for(String statement:Arrays.asList(statements)) { //tokens push }
    }
    return tokens;
}

String[] lexer(String[] tokens) {
    String[] ast = {};
    for(String token:Arrays.asList(tokens)) { //ast push }
    return ast;
}

void parseBetterJSAlternative(String code) {
    String[] tokens = tokenize(code);
    String[] ast = lexer(tokens);
    for(String node:Arrays.asList(ast)) { //parse ...}
}
```

坏味道在哪❓

```
void showDeveloperList(List<Developer> developers) {
    for(Developer developer:developers) {
        render(new Data(developer.expectedSalary, developer.experience, developer.githubLink));
    }
}

void showManagerList(List<Manager> managers) {
    for(Manager manager:managers) {
        render(new Data(manager.expectedSalary, manager.experience, manager.portfolio));
    }
}
```

坏味道：重复代码。补救办法：消除重复

```
void showList(List<Employee> employees) {
    for(Employee employee:employees) {
        Data data=new Data(employee.expectedSalary,employee.experience,employee.githubLink);
        String portfolio=employee.portfolio;
        if("manager".equals(employee)){
            portfolio=employee.portfolio;
        }
        data.portfolio=portfolio;
        render(data);
    }
}
```

坏味道在哪❓

```
@Deprecated
void oldRequestModule(String url) {
    // ...
}

void newRequestModule(String url) {
    // ...
}

String req = newRequestModule;
inventoryTracker("apples", req, "www.inventory-awesome.io");
```

坏味道：僵尸代码。解救办法：安全删除

```
void newRequestModule(String url) {
    // ...
}

String req = newRequestModule;
inventoryTracker("apples", req, "www.inventory-awesome.io");
```

坏味道在哪❓

```
class UserSettings {
    User user;

    void changeSettings(UserSettings settings) {
        if (this.verifyCredentials()) {
            // ...
        }
    }

    void verifyCredentials() {
        // ...
    }
}
```

坏味道：违反单一职责原则。补救办法：提取类、移动方法

```
User user;
UserAuth auth;

public UserSettings(User user) {
    this.user = user;
    this.auth = new UserAuth(user);
}

void changeSettings(UserSettings settings) {
    if (this.auth.verifyCredentials()) {
        // ...
    }
}
```

预防措施

代码规范

- 方法长度限制
- 类长度限制
- 接口废弃约定（@Deprecated）

CheckStyle

- Maximum File Length（最大文件长度）
- Maximum Line Length（最大行长度）
- Maximum Method Length（最大方法长度）
- Strict Duplicate Code（严格重复代码）
- Designed For Extension（设计扩展性）

重构手法

- 提取方法（Extract Method）
- 移动变量（Move...）
- 移动方法（Move...）
- 安全删除（Safe Delete）

命名太  了

- Chinglish，英文水平参差不齐
- 方法参数太多，命名随意，无法判断参数副作用
- 滥用缩写
- 实现修改了，命名没有修改
- 害怕散弹式修改，不敢重命名或者修改参数

❓你是否为曾经为命一个好名称而感到头痛，你有什么好的方式解决?

（讨论时间：5分钟）

坏味道在哪❓

```
String yyyymmdstr = new SimpleDateFormat("YYYY/MM/DD").format(new Date());
```

坏味道：命名无意义。补救办法：使用有意义并且可读的变量名称

```
String currentDate = new SimpleDateFormat("YYYY/MM/DD").format(new Date());
```

坏味道在哪❓

```
// 86400000 是什么鬼?
setTimeout(blastOff, 86400000);
```

坏味道：魔法值。补救办法：使用可搜索的名称

```
// 将它们声明为全局常量。
public static final int MILLISECONDS_IN_A_DAY = 86400000;
setTimeout(blastOff, MILLISECONDS_IN_A_DAY);
```

坏味道在哪❓

```
String address = "One Infinite Loop, Cupertino 95014";
String cityZipCodeRegex = "/^[^,\\\\]+[,\\\\\\s]+(.+?)\\s*(\\d{5})?$/";

saveCityZipCode(address.split(cityZipCodeRegex)[0],
address.split(cityZipCodeRegex)[1]);
```

坏味道：直接用表达式传递变量。补救办法：增加解释性变量

```
String address = "One Infinite Loop, Cupertino 95014";
String cityZipCodeRegex = "/^[^, \\\\]+[, \\\\\\s]+(.+?)\\s*(\\d{5})?$/";

String city = address.split(cityZipCodeRegex)[0];
String zipCode = address.split(cityZipCodeRegex)[1];

saveCityZipCode(city, zipCode);
```

坏味道在哪❓

```
String[] l = {"Austin", "New York", "San Francisco"};

for (int i = 0; i < l.length; i++) {
    String li = l[i];
    doStuff();
    doSomeOtherStuff();
    // ...
    // ...
    // ...
    // Wait, what is `$li` for again?
    dispatch(li);
}
```

坏味道：使用隐晦的缩写。补救办法：使用显示命名

```
String[] locations = {"Austin", "New York", "San Francisco"};

for (String location : locations) {
    doStuff();
    doSomeOtherStuff();
    // ...
    // ...
    // ...
    dispatch(location);
}
```

坏味道在哪❓

```
void createMenu(String title, String body, String buttonText, boolean cancellable) {}
```

坏味道：超长参数列表（超过2个）。补救办法：封装参数对象

```
class MenuConfig {
    String title;
    String body;
    String buttonText;
    boolean cancellable;
}
void createMenu(@Nullable MenuConfig menuConfig) {}
```

预防措施

代码规范

- 常量使用（位置、命名）
- 使用标记注解（android.support.annotations）
- 建立业务术语表（中英文对照）

CheckStyle

- Member Names（成员名称）
- Method Names（方法名称）
- Maximum Parameters（最大参数数量）

参考别人是怎么命名的 https://unbug.github.io/codelf/

利用 Android Studio 提示参数名

重构手法

- 重命名（Rename）
- 提取常量（Extract Constant）
- 提取变量（Extract Variable）
- 提取方法参数对象（Extract Parameter）
- 修改方法签名（Change Signature）

if/else/for嵌套

- 缩进不统一，怕影响 blame 不敢改
- 分支丢失，缺少 else

？为什么 if-else 不是好代码?

（讨论时间：5分钟）

坏味道在哪❓

```
if (fsm.state.equals("fetching") && listNode.isEmpty()) {
    //...
}
```

坏味道：条件语句过长。补救办法：封装条件语句

```
void shouldShowSpinner(Fsm fsm, String listNode) {
    return fsm.state.equals("fetching") && listNode.isEmpty();
}

if (shouldShowSpinner(fsmInstance, listNodeInstance)) {
    // ...
}
```

坏味道在哪❓

```
void isDOMNodeNotPresent(Node node) {
    // ...
}

if (!isDOMNodeNotPresent(node)) {
    // ...
}
```

坏味道：负面条件判断语句。补救办法：使用正面判断条件

```
void isDOMNodePresent(Node node) {
    // ...
}

if (isDOMNodePresent(node)) {
    // ...
}
```

坏味道在哪❓

```
double disablityAmount() {
    if (_seniority < 2)
        return 0;

    if (_monthsDisabled > 12)
        return 0;

    if (_isPartTime)
        return 0;

    //do somethig
    }
```

坏味道：相同结果的条件过渡拆分。补救办法：合并条件表达式

```
double disablityAmount(){
    if (_seniority < 2 || _monthsDisabled > 12 || _isPartTime)
        return 0;
    //do somethig
}
```

坏味道在哪❓

```
double getPayAmount() {
    double result;
    if (_isDead) {
        result = deadAmount();
    } else {
        if (_isSeparated) {
            result = separatedAmount();
        }
        else {
            if (_isRetired) {
                result = retiredAmount();
            else {
                result = normalPayAmount();
            }
        }
    }
    return result;
}
```

坏味道：if-else嵌套没有关联性。补救办法：将包含关系改为平行关系

```
double getPayAmount(){
    if (_isDead)
        return deadAmount();

    if (_isSeparated)
        return separatedAmount();

    if (_isRetired)
        return retiredAmount();

    return normalPayAmount();
}
```

坏味道在哪❓

```
/* 查找年龄大于18岁且为男性的学生列表 */
public ArrayList<Student> getStudents(int uid){
        ArrayList<Student> result = new ArrayList<Student>();
        Student stu = getStudentByUid(uid);
        if (stu != null) {
            Teacher teacher = stu.getTeacher();
            if (teacher != null) {
                ArrayList<Student> students = teacher.getStudents();
                if (students != null) {
                    for(Student student : students){
                        if(student.getAge() > = 18 && student.getGender() == MALE){
                            result.add(student);
                        }
                    }
                } else {
                    logger.error("获取学生列表失败");
                }
            } else {
                logger.error("获取老师信息失败");
            }
        } else {
            logger.error("获取学生信息失败");
        }
        return result;
    }
```

坏味道："箭头型"代码。补救办法：异常条件先退出，保持主干流程是核心流程

```java
/* 查找年龄大于18岁且为男性的学生列表 */
public ArrayList<Student> getStudents(int uid){
        ArrayList<Student> result = new ArrayList<Student>();
        Student stu = getStudentByUid(uid);
        if (stu == null) {
            logger.error("获取学生信息失败");
            return result;
        }

        Teacher teacher = stu.getTeacher();
        if (teacher == null) {
            logger.error("获取老师信息失败");
            return result;
        }

        ArrayList<Student> students = teacher.getStudents();
        if (students == null) {
            logger.error("获取学生列表失败");
            return result;
        }

        for(Student student : students) {
            if (student.getAge() > 18 && student.getGender() == MALE){
                result.add(student);
            }
        }
        return result;
    }
```

坏味道在哪❓

```
class Airplane {
    int getCurisingAltitude() {
        switch(this.type) {
            case "777":
                return this.getMaxAltitude() - this.getPassengerCount();
            case "Air Force One":
                return this.getMaxAltitude();
            case "Cessna":
                return this.getMaxAltitude() - this.getFuelExpenditure();
        }
    }
}
```

坏味道：条件分支太多。补救办法：运用多态

```
class Airplane {
    // ...
}

class Boeing777 extends Airplane {
    // ...
    int getCruisingAltitude() {
        return this.getMaxAltitude() - this.getPassengerCount();
    }
}

class AirForceOne extends Airplane {
    // ...
    int getCruisingAltitude() {
        return this.getMaxAltitude();
    }
}

class Cessna extends Airplane {
    // ...
    int getCruisingAltitude() {
        return this.getMaxAltitude() - this.getFuelExpenditure();
    }
}
```

预防措施

代码规范

- 统一缩进（使用 Space，利用保存时自动格式化功能）
- 限制方法复杂度（不超过 7）

CheckStyle

- Nested For Depth（for嵌套深度）
- Nested If Depth（if嵌套深度）
- Simplify Boolean Expression（简化布尔表达式）
- Cyclomatic Complexity（圈复杂度）

`git blame -w -M` 或者 git-hyper-blame

重构手法

- 提取变量（Extract Variable）
- 提取方法（Extract Method）
- 移动代码块（Move...）

文档和注释问题

- 重要的方法没有注释，如关键算法、BUG 修改
- 无意义的注释太多
- 重要的接口缺少文档
- 接口文档没有集中管理，搜索如大海捞针

❓那些注释或文档是必需要写的?

(讨论时间：5分钟)

坏味道在哪里？

```
void hashIt(String data) {
    // The hash
    long hash = 0;

    // Length of string
    int length = data.length();

    // Loop through every character in data
    for (int i = 0; i < length; i++) {
        // Get character code.
        char mChar = data.charAt(i);
        // Make the hash
        hash = ((hash << 5) - hash) + mChar;
        // Convert to 32-bit integer
        hash &= hash;
    }
}
```

坏味道：无意义的注释太多。补救办法：仅对包含复杂业务逻辑进行注释

```
void hashIt(String data) {
    long hash = 0;
    int length = data.length();

     for (int i = 0; i < length; i++) {
         char mChar = data.charAt(i);
         hash = ((hash << 5) - hash) + mChar;

        // Convert to 32-bit integer
        hash &= hash;
    }
}
```

坏味道在哪里？

```
doStuff();
// doOtherStuff();
// doSomeMoreStuff();
// doSoMuchStuff();
```

坏味道：不要在代码库中保存注释掉的代码。补救办法：把老代码留在版本控制里面

```
    doStuff();
```

坏味道在哪里？

```
/**
 * 2016-12-20: Removed monads, didn't understand them (RM)
 * 2016-10-01: Improved using special monads (JP)
 * 2016-02-03: Removed type-checking (LI)
 * 2015-03-14: Added combine with type-checking (JR)
 */
void combine(String a, String b) {
    return a + b;
}
```

坏味道：不要有日志式的注释。补救办法：移除，使用 `git log` 来获取历史记录

```
void combine(String a, String b) {
    return a + b;
}
```

坏味道在哪里？

```
////////////////////////////////////////////////////////////////////////
// Scope Model Instantiation
////////////////////////////////////////////////////////////////////////
String[] model = {"foo","bar"};

////////////////////////////////////////////////////////////////////////
// Action setup
////////////////////////////////////////////////////////////////////////
void action(){
    //...
}
```

坏味道：使用占位符。补救办法：移除，使用缩进和格式化；有意义的命名

```
String[] model = {"foo", "bar"};
void actionSetup(){
    //...
}
```

预防措施

代码规范

- 公共接口必须规范注释（JavaDoc 规范）
- 复杂算法加上合适的注释
- 用有意义的名字代替注释

CheckStyle

- Style Javadoc（风格注释）
- Trailing Comment（行尾注释）

滥用设计模式

- 单例模式初始化顺序引起 NPE
- 单例带来的内存问题，尤其使用 list 或者 map

❓在项目中，经常使用那些了那些设计模式，为什么使用？使用时有什么地方需要注意吗？

（讨论时间：5分钟）

坏味道在哪里？

```
public class CommUtil {
    private static CommUtil instance;
    private Context context;

    private CommUtil(Context context) {
        this.context = context;
    }

    public static CommUtil getInstance(Context mcontext) {
        if (instance == null) {
            instance = new CommUtil(mcontext);
        }
        return instance;
    }
}
```

坏味道：单例造成的内存泄漏。补救办法：绑定引用的生命周期或及时释放引用

```
public class CommUtil {
    private static CommUtil instance;
    private Context context;

    private CommUtil(Context context) {
        this.context = context;
    }

    public static CommUtil getInstance(Context mcontext) {
        if (instance == null) {
            instance = new CommUtil(mcontext.getApplicationContext());
        }
        return instance;
    }
}
```

预防措施

CheckStyle

- Explicit Initialization（显式初始化）
- Class Data Abstraction Coupling（类的数据抽象耦合）

leakcanary

- debug notify

依赖注入

- dagger2
- koin

匿名内部类和回调地狱

- 随意使用匿名内部类
- 回调地狱，可读性极差

？为什么会“偷懒”使用匿名内部类?

（讨论时间：5分钟）

坏味道在哪里？

```
new AsyncTask<Void, Void, Void>() {
    @Override
    protected Void doInBackground(Void... params) {
        SystemClock.sleep(10000);
        return null;
    }
}.execute();

new Thread(new Runnable() {
    @Override
    public void run() {
        SystemClock.sleep(10000);
    }
}).start();
```

坏味道：匿名内部类导致内存泄漏。补救办法：使用静态内部类

```
static class MyAsyncTask extends AsyncTask<Void, Void, Void> {
    private WeakReference<Context> weakReference;

    public MyAsyncTask(Context context) {
        weakReference = new WeakReference<>(context);
    }

    @Override
    protected Void doInBackground(Void... params) {
        SystemClock.sleep(10000);
        return null;
    }

    @Override
    protected void onPostExecute(Void aVoid) {
        super.onPostExecute(aVoid);
        MainActivity activity = (MainActivity) weakReference.get();
        if (activity != null) {
            //...
        }
    }
}

static class MyRunnable implements Runnable {
    @Override
    public void run() {
        SystemClock.sleep(10000);
    }
}

new Thread(new MyRunnable()).start();
new MyAsyncTask(this).execute();
```

坏味道在哪里？

```
new Thread(new Runnable() {
    @Override
    public void run() {
        try {
            Bitmap qrCode = CodeCreator.createQRCode(ShareActivity.this, SHARE_QR_CODE);
            runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    img_qr_code.setImageBitmap(qrCode);
                }
            });
        } catch (WriterException e) {
            e.printStackTrace();
        }
    }
}).start();
```

坏味道：回调地狱。补救办法1：使用链式调用

```
Observable.just(SHARE_QR_CODE)
        .map(new Function<String, Bitmap>() {
            @Override
            public Bitmap apply(String s) throws Exception {
                return CodeCreator.createQRCode(ShareActivity.this, s);
            }
        })
        .subscribe(new Consumer<Bitmap>() {
            @Override
            public void accept(Bitmap bitmap) throws Exception {
                img_qr_code.setImageBitmap(bitmap);
            }
        });
```

坏味道：回调地狱。补救办法2：使用 lambda

```
Observable.just(SHARE_QR_CODE)
                .map(s -> CodeCreator.createQRCode(ShareActivity.this, s))
                .subscribe(bitmap -> img_qr_code.setImageBitmap(bitmap));
```

坏味道：回调地狱。补救办法3：使用 coroutine

```
val job = launch(Background) {
    val bitmap = CodeCreator.createQRCode(ShareActivity.this, SHARE_QR_CODE)
    launch(UI) {
        img_qr_code.setImageBitmap(bitmap)
    }
}
```

预防措施

库/语言特性

- Java 8 lambda（Android Studio 可以把 Java 8 之前的单方法接口显示成 lambda）
- RxJava

Lint

- Android/Performace
- Java/Memory

leakcanary

- debug notify

并发和多线程问题

- 只会使用 synchronized 解决同步问题
- 随意 new Thread 或者 new AsyncTask

❓你遇到过什么并发的坑？如何爬出来的？

（讨论时间：5分钟）

坏味道在哪里？

```
public class Counter{
    private int value;
    public synchronized int getValue() {
        return value;
    }
    public int getNextValue() {
        return value++;
    }

    public int getPreviousValue() {
        return value--;
    }
}
```

坏味道：随意使用synchronized。改进办法：使用无锁算法

```
public class AtomicCounter {
    private final AtomicInteger value = new AtomicInteger(0);

    public int getValue() {
        return value.get();
    }

    public int getNextValue() {
        return value.incrementAndGet();
    }

    public int getPreviousValue() {
        return value.decrementAndGet();
    }

}
```

坏味道在哪里？

```
new Thread(new Runnable() {
    @Override
    public void run() {
        // TODO Auto-generated method stub
    }
}).start();
```

坏味道：直接 new Thread。补救办法：使用线程池管理

```
ExecutorService singleThreadExecutor = Executors.newSingleThreadExecutor();

singleThreadExecutor.execute(new Runnable() {
    @Override
    public void run() {
        // TODO Auto-generated catch block
    }
});
```

预防措施

Alibaba Coding Gudelines

- Threads should be provided by thread pools
- A thread pool should be created by ThreadPoolExecutor rather than Executors.

库/语言特性

- RxJava
- Kotlin 协程

不积跬步，无以至千里

不积小流，无以成江海

10,000 hours to master your craft

Better get started!

课后作业

至少完成 4 处产品代码中坏味道的重构（坏味道不限于上述7类）。

要求：

1. 尽可能采用 Android Studio 的安全重构手段小步迭代前进。
2. 每处坏味道说明：视为坏味道的原因；重构前后代码 Diff；重构时掉过的坑。
3. 作业以文档形式提交到石墨
4. 完成时间截止2020年3月17日早上10点
5. 作业可以分步提交（建议完成一处提交一处）

课后作业

积分规则：

1. 按时按照要求完成作业提交即获得 3 个基础积分。
2. 教练对作业进行 Review，评选出优秀的重构每处获得 2 个积分。
3. 学员获得的总分 15 分封顶。

课后作业

获得优秀重构的秘籍：

1. 只用一次快捷键就完成的重构大概率不会获得优秀
2. 巧妙地解决了不能自动化重构的问题（在作业中描述）
3. 重构时补充了单元测试（测试代码附在作业中）

课后分享

- 如无特别说明，内容必须和当期作业有关。
- 每人10分钟，控制好时间（不超过12分钟）。
- 每人分享完之后，教练点评（一共 2 分钟)。

> 把作业以及分享都和实际工作相结合，用最低的成本换来最高的价值

积分规则

在当期学员全部分享完毕，由正式学员和教练（旁听同事不参与）现场投票。学员每人有 3 票，教练每人有 6 票，一票只能投给一人。得票最多的前三名（名次允许并列）分别获得7个、5个、3个积分，其它当期分享过的学员获得1个积分。

> 因为学员人数较多，每一期只能让部分学员（约10人）分享；每一期分享都按照主动报名顺序（只取前10人，截止当期分享开始前均可报名）进行，但我们一定保证全部学员轮转分享一次后，再进行下一次轮转。(例如A学员第一次分享之后，第二次依然第一个主动报名，也必须等到第一次没有分享的同事分享完之后，才会轮到A)

第一次分享出场顺序

邱盛、汪航、马干宣、刘剑、马明星、黄细闽、何辉、吴俊逸、姚伟和、陈桂鹏、周伍润、陈蓉、张烨、彭颖、何清丰。

> 如缺席、请假或者不方便接入则顺延到下一位

第一次作业总结

邱盛：组合模式以及适用场景介绍。汪航：MsgSyncAgent 不同版本 SDK 抽象。马干宣：DebugLog.d 和拼装字符串优化，LiveTemplate 技巧。刘剑：WeatherSupplier 三步走分成四个类，支持不同版本天气接口。黄细闽：使用组件化隔离内外销地址常量。马明星：使用 HashMap 提到 Switch 语句，用内存换取可读性。陈桂鹏：提取复杂布局计算方法，提高可读性。张烨：使用提取方法简化 Switch 语句。何清丰：提取四个检查方法，消除if嵌套逻辑。

第二次分享出场顺序

何辉、吴俊逸、姚伟和、周伍润、陈蓉、彭颖