UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ
BACHARELADO EM ENGENHARIA DE SOFTWARE

WELLYNGTON AMARAL LEITÃO

ESTUDO EXPLORATÓRIO DE BUGS RELACIONADOS AO TRATAMENTO DE EXCEÇÃO NA PLATAFORMA ANDROID

QUIXADÁ
2016

WELLYNGTON AMARAL LEITÃO

ESTUDO EXPLORATÓRIO DE BUGS RELACIONADOS A TRATAMENTO DE EXCEÇÃO NA PLATAFORMA ANDROID

Trabalho de Conclusão de Curso submetido à Coordenação do Curso Bacharelado em Engenharia de Software da Universidade Federal do Ceará como requisito parcial para obtenção do grau de Bacharel.

Área de concentração: Computação

Orientador Dr. Jefferson da Silva Carvalho

QUIXADÁ
julho-2016

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

L549e Leitão, Wellyngton Amaral.
Estudo Esploratório de Bugs de Tratamento de Exceção na Plataforma Android : Estudo Esploratório /
Wellyngton Amaral Leitão. – 2016.
65 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá,
Curso de Engenharia de Software, Quixadá, 2016.
Orientação: Prof. Dr. Jefferson da Silva Carvalho.

1. Programação orientada a objetos (Computação). 2. Tratamento de exceção. 3. Bug. 4. Código aberto. 5. Android (Recurso eletrônico). I. Título

CDD 621.382

A minha familia...

AGRADECIMENTOS

Quero agradecer primeiramente a Deus, por proporcionar coisas muito felizes na minha vida. Agradecer também a minha familia que sempre se demonstrou presente apesar da distancia.

Agradecer ao professor Dr Jefferson que aceitou o desafio de seguir uma orientação que tinha começado com outro orientado e o projeto já seguia estagnado durante muito tempo, tive total apoio, flexibilidade e compreensão no desenvolvimento da minhas atividades.

Agradecer a todos os educadores que cumpriram seu papel em me passar uma pequena parte do conhecimento que eles obtiveram em todos esses anos de estudo e pesquisa.

E finalmente agradecer aos meus amigos, e companheiros de trabalho que tanto me dão suporte e liberdade, quando aparecem problemas que preciso priorizar em determinados momentos.

"Not all those who wander are lost."
(JRR Tolkien)

RESUMO

A importância de linguagens com paradigma orientada a objetos é uma realidade no meio do desenvolvimento de software. Tais linguagens são bastante robustas se comparadas a linguagens de um paradigma estruturado, por exemplo. Uma das característica que a maioria das linguagens de programação orientada a objetos possue em relação as linguagens do paradigma estrutural, é a capacidade de tratar situações excepcionais no fluxo da lógica do sistema, e se recuperar delas para que o sistema não pare de funcionar. O mecanismo de tratamento de exceção é um recurso que a maioria das linguages de programação orientada a objetos oferem para tratar essas situações excepcionais. Porém apesar da sua importancia, existem *bugs* que estão diretamente relacionados ao código de tratamento de exceção, seja por falta de conhecimento do desenvolvedor sobre a forma correta de utilizar o mecanismo, ou por negligenciar essa parte na lógica do sistema. Este trabalho tem como foco fazer um estudo exploratório em projetos reais de código aberto da plataforma android, com o objetivo de buscar *bugs* que sejam relacionados com o código do mecanismo de tratamento de exceção e catalogá-los para que se possa obter uma visão consolidada de quais os tipos de bugs da plataforma e em quais momentos esses *bugs* aparecem com maior frequencia.

Palavras chave: Bugs. Tratamento de Exceção. Android.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

Uma exceção é uma condição anômala (excepcional) que altera ou interrompe o fluxo de execução normal de um programa (LIGUORI, 2013). Nesse sentido, linguagens de programação modernas como Java e C++ possuem estruturas específicas para lidar com mecanismos que tratam condições excepcionais (GARCIA E RUBIRA, 2001). O recurso utilizado por essas linguagens é conhecido como **tratamento de exceção**. O tratamento de exceção possibilita que o programa detecte e se recupere de erros de entrada e eventos externos inesperados (SOMMERVILLE, 2011). O mecanismo para o tratamento de exceção é uma das técnicas utilizadas na abordagem da programação defensiva. Segundo Engelen (2000) programação defensiva é uma técnica que torna os programas mais robustos para tratar eventos inesperados, fazendo com que uma quantidade relativamente menor de aplicações quebrem em tempo de execução, aumentando assim a qualidade do produto final. Ebert (2013) afirma que a maioria dos desenvolvedores não fazem o tratamento de exceção ou o fazem de maneira incorreta, gerando assim códigos menos robustos e com uma maior tendência a gerar algum tipo de *bug* relacionado ao código de tratamento de exceção.

Ebert (2013) afirma que, as causas mais comuns para *bugs* relacionados a tratamento de exceção são: inexistência do manipulador (que deve existir), exceção não lançada e erro no código que diz respeito ao corpo do block "*catch*" – palavra reservada do Java que serve para definir um bloco no qual determinados tipos de exceções serão tratados.

O estudo de Ebert (2013) revela que o código para tratamento de exceção normalmente tem baixa qualidade devido a negligência por parte dos desenvolvedores, seu trabalho teve como objetivo um estudo exploratório sobre bugs de tratamento de exceção baseado em duas abordagens complementares: uma pesquisa com 154 desenvolvedores e uma análise de 220 bugs dos repositórios do Eclipse e Tomcat, o trabalho mostrou que existe um alto nível de negligencia por parte dos desenvolvedores ao se tratar exceções na plataforma Java. Essa negligência ocorre devido a visão que os desenvolvedores (iniciantes e experientes) possuem em relação ao tratamento de exceção.

> Uma grande parte dos desenvolvedores iniciantes usam o tratamento de exceção na maioria dos casos para *debugging* e/ou por ser requerido pela linguagem, por outro lado, uma grande parte dos desenvolvedores experientes usam o tratamento de exceção para transmitir mensagens de falha compreensíveis (EBERT, 2013).

O estudo de Ebert (2013) também revela que, o tempo de correção dos bugs de tratamento de exceção é significativamente menor do que o de outros tipos de *bugs*. Verificar que existem bugs que quebram a aplicação durante o tempo de execução devido ao uso indevido do mecanismo de tratamento de exceção, pode fazer com que os desenvolvedores passem a utilizar o código do tratamento de exceção com maior prudência, podendo assim diminuir consideravelmente o tempo despendido na fase de debugs e testes.

Este trabalho tem como objetivo gerar dados relacionados a tipos de aplicações diferentes desenvolvidos para a plataforma android estabelecendo uma relação de *bugs* e tratamento de exceção de forma a responder as seguintes perguntas.

- Existem bugs relacionados a tratamento de exceção na plataforma android?
- Quais tipos de bugs relacionados a tratamento de exceção estão presentes na plataforma android?
- Com que frequencia cada tipo de bug ocorre e em qual situação o bug se demonstra mais recorrente.

Nos últimos tempos tem-se notado um crescimento em relação a apllicações *mobile* (móvel) para diversos tipos de dispositivos e plataformas. Estudos comprovam que mais de 3 bilhões de pessoas possuem um dispositivo *mobile*, isso corresponde a aproximadamente metade da população mundial (LECHETA, 2015), devido a esse aumento constante de usuários, foi escolhida uma plataforma *mobile* para conduzir este trabalho. A escolha de qual plataforma *mobile* utilizar foi feita levando em consideração os seguintes fatores.

- Possuir uma grande gama de projetos de código aberto, desenvolvidos para a plataforma
- Ser uma tecnologia estável: que tenha suporte constante e que seja confiável para desenvolvimento de projetos reais.

Android é a primeira plataforma para aplicações móveis completamente livre e *"open source"* (de código aberto) (LECHETA, 2015). Além disso, tem Java como liguagem base para codificação de seus projetos. A linguagem Java é uma tecnologia consolidada no mercado, e fornece recursos e ferramentas robustas para a plataforma android como por exemplo um mecanismo sofisticado para tratamento de erros que produz códigos de manipulação eficientes e organizados, essa parte específica do código é conhecida como tratamento de exceção" (SIERRA, 2008). Outro motivo que levou a escolha da plataforma

android para este trabalho foi a quantidade de projetos *open source* existentes no **Github** (repositório que será utilizado para este trabalho). Java é a segunda linguagem mais utilizada em projetos dentro do Github (ficando atrás apenas do JavaScript).

Para atingirmos o objetivo, será utilizada uma abordagem exploratória para verificar as *issues* (palavra que denomina as atividades que são requisitadas para fazer algum tipo de alteração no projeto, seja para resolver um *bug* ou para adicionar funcionalidades novas) dos projetos do Github, com a intenção de filtrar os *bugs* relacionados ao código de tratamento de exceção e determinar qual tipo específico de *bug* de tratamento de exceção ocorre com maior frequência. Este estudo empírico é baseado em uma análise de repositório de código aberto da plataforma android, já que seguindo essa abordagem temos a possibilidade de capturar os trechos de códigos relacionados ao tratamento de exceção desenvolvidos por terceiros, de forma que eles representam situações reais no desenvolvimento de aplicações para a plataforma.

2 FUNDAMENTAÇÃO TEÓRICA

Para se trabalhar com uma análise de repositórios de código fonte e identificar problemas possivelmente causados pela falha ou até mesmo falta do código necessário relacionado ao tratamento de exceção, é preciso entender o que são *bugs*, identificar quais tipos de *bugs* existem de maneira pragmática, e saber mensurar o impacto que determinados tipos de *bugs* podem causar a um sistema de *software*. Além disso é de fundamental importância conhecer o conceito de exceção, e como tratá-la da maneira adequada. O mecanismo para tratamento de exceção pode gerar *bugs* se não utilizados, ou se utilizados de maneira incorreta, esse tipo específico de *bug* será definido neste trabalho, como "*bug* de tratamento de exceção". Por fim, o trabalho de identificação e análise de *bugs* de tratamento de exceção será feito com repositórios de projetos da plataforma android, o que requer um determinado conhecimento prévio em relação a linguagem base da plataforma Java, assim como conhecimento da tecnologia android e suas especificidades, seus conceitos e sutilezas.

2.1 Bugs

Existe uma diferença classificatória entre as definições para cada possível tipo de *bug* dentro de um sistema de *software*. Segundo a IEEE os *bugs* devem ser divididos em categorias. São elas: defeito, erro, falha, falta.

- **Defeito**: É um evento que altera o estado interno de um sistema de *software* para um estado em que o serviço não corresponde ao esperado, ou desejado, conforme o definido em alguma especificação (BARBOSA, 2012).
- **Erro**: É um estado interno de um sistema de *software* que possibilita a ocorrência de um defeito (BARBOSA, 2012).
- **Falha**: É uma causa hipotética de erro. Falhas podem ser imperfeições ou irregularidades que ocorrem em módulos de hardware ou software (BARBOSA, 2012).
- **Falta**: É a causa física ou algorítmica de um erro. (ROCHA, 2014).

O bug é um estado computacional não previsto que pode levar a uma falha. Uma falha pode causar tanto erros físicos no *hardware*, quanto erros criados a partir de deslizes de programação (SANTOS, 2010). Existem diversos tipos de estados computacionais que podem levar a uma falha, e esses estados computacionais, vão desde o mau funcionamento do sistema operacional, até o sistema responder de forma inesperada a determinadas entradas do usuário

Santos (2010) afirma que os erros/*bugs* podem ser causados por falha do desenvolvedor na fase de implementação. Um caso famoso desse tipo de ocorrência de *bug* aconteceu com o *software* de vôo da sonda Mariner 1, quando uma falha nos cálculos do código, fez com que a nave desviasse de seu curso pré-estabelecido, devido a esse problema, os responsáveis da missão foram obrigados a destruir o foguete, causando prejuízo de milhões a NASA (*National Aeronautics and Space Administration*).

Segundo o Instituto de Pesquisa Científica da IBM o custo para consertar algum tipo de bug que foi descoberto após a versão ter sido publicada em produção é 4 à 5 vezes maior do que um *bug* sendo descoberto durante a fase do *design*, e até 100 vezes mais do que se identificado na fase de manutenção, e com o passar do tempo esse número tende a aumentar, devido ao surgimento de novas tecnologias e ao aumento da complexidade das aplicações.

Proporcionalmente ao aumento da complexidade das aplicações e ao número de componentes utilizados para a construção de um sistema, cresce também, o número de estados computacionais que podem levar a um Erro/*bug* (SANTOS, 2010) e com o advento de novas tecnologias o número de novos tipos de *bugs* relacionado a determinados tipos de situações

computacionais cresce proporcionalmente, dificultando assim a identificação e o tratamento correto para um determinado tipo de *bug*.

O mecanismo para tratamento de exceção é um exemplo de um componente que a partir do momento que foi integrado a determinadas linguagens, foi gerado novos estados computacionais, como consequência disso, novos tipos de *bugs* surgiram, esse grupo específico de *bugs*, passou a ser conhecido como: "*bugs* de tratamento de exceção".

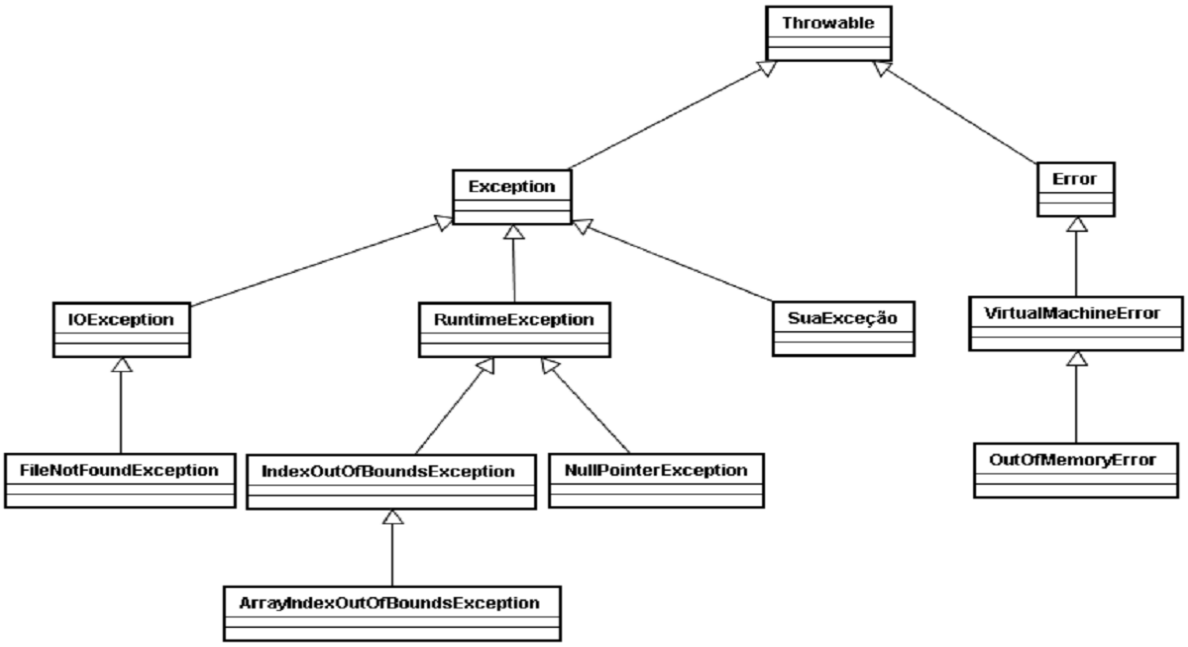
O *bug* de tratamento de exceção é um termo que surgiu para designar falhas em códigos relacionados ao mecanismo de tratamento de exceção. Esse trabalho visa identificar a ocorrência de *bugs* de tratamento de exceção na plataforma android, com o objetivo de diminuir o custo (de tempo e financeiro) relacionado a correção de *bugs* desse tipo, levando em consideração que *bugs* de tratamento de exceção são mais fáceis de serem corrigidos do que outros tipos de *bugs*.

2.2 Exceção

O termo exceção significa "condição excepcional", e é uma ocorrência que altera o fluxo normal do programa fazendo com que o programa pare abruptamente. Para que isso possa ser tratado algumas plataformas como java/android oferecem suporte a manipulação de exceções (SIERRA, 2008).

O Java trabalha com uma hierarquia de exceções (Figura 1) onde *Throwable* é a superclasse de todas as outras, Segundo Ricarte (2001) apenas objetos dessa classe ou de suas classes derivadas podem ser gerados, propagados e capturados através do mecanismo de tratamento de exceção. Porém não é correto afirmar por exemplo, que todas as subclasses de *Throwable* são tipos de exceções, já que a classe *Error* também herda de *Throwable* e não é um tipo de exceção.

Figura 1 - Hierarquia de Exceções



Fonte: Lopes, 2009.

A classe *Throwable* é uma classe genérica de grande importância para o java, pois além se ser a classe pai de todas as exceções, ela também fornece métodos que permitem ao desenvolvedor conhecer mais sobre o problema.

A classe *Throwable* é subclassificada duas vezes, tanto pela classe *Exception* quanto pela classe *Error*. É importante frisar que a classe *Error* e suas subclasses não são consideradas tipos de exceções, porém pode-se usar o mesmo mecanismo de tratamento de exceção que é utilizado em exceções para capturá-las.

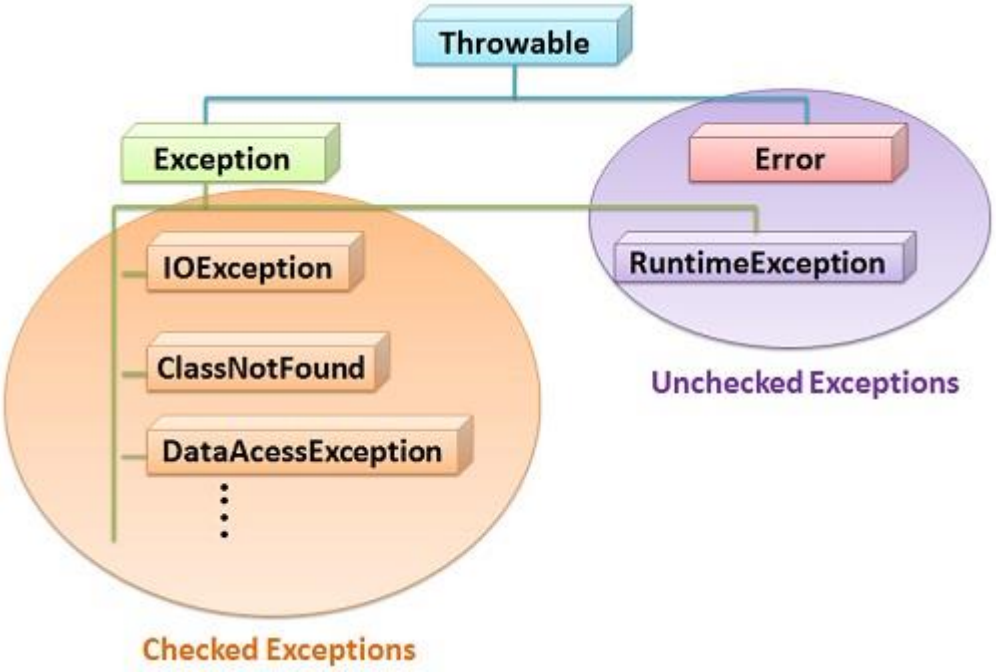
A classe *Exception* e suas subclasses por sua vez, são tipos de exceções. Existem dois tipos de exceções, as verificadas e as não verificadas em tempo execução (Figura 2):

As "*checked exceptions"* (exceções verificadas) são um conjunto especial de exceções que herdam da classe *Exception*. A linguagem Java foi a primeira tecnologia a usar o conceito de exceções verificadas. As exceções verificadas são conhecidas dessa forma pois o compilador verifica o código em tempo de compilação forçando o programador a tratar ou lançar a exceção para que o método chamador a trate. Elas representam erros que podem ocorrer em tempo de execução, mas não dependem da lógica do programa (JONATHAN, 2011). São exemplos de exceções verificadas: *IOException* e *ClassNotFoundException*.

As subclasses de *RuntimeException* são conhecidas como *unchecked exceptions* (exceções não verificadas), pois o compilador não verifica se elas podem ser lançadas, não

sendo obrigatório tratá-las. Segundo Jonathan (2011), as exceções não verificadas, representam erros ou defeitos na lógica do programa que podem causar problemas irrecuperáveis em tempo de execução. *IllegalArgumentException* e *NullPointerException* são exemplos de exceções não verificadas. Segundo Taborda (2007), *IllegalArgumentException* acontece quando se passa um parâmetro para um método e o método não pode usá-lo, já *NullPointerException* acontece quando se tenta acessar uma referência nula.

Figura 2 – *"Checked"* e *"UncheckedExceptions"*



Fonte: Página do Blog TechDiffrences

O tratamento de exceção tem grande relevância para a garantia de que o *software* funcionará corretamente. Tratar as exceções (verificadas e/ou não verificadas) lançadas durante a execução, é uma forma de garantir maior robustez ao código e fazer com que o programa possa se recuperar de alguma "condição excepcional".

2.2.1 Tratamento de Exceção

A captura e o tratamento de exceções em Java se dá através da especificação de blocos *try*, *catch* e *finally*, definidos através destas mesmas palavras reservadas da linguagem (RICARTE, 2001) Conforme descrito no Quadro 1.

O estado inicial do mecanismo de tratamento de exceção no java é o bloco "*try*", se o código lançar um determinado tipo de exceção durante esse bloco, a exceção será automaticamente capturada em tempo de execução pela clausula "*catch*", a clausula "*catch*" por sua vez, terá a missão de tratá-la de acordo com a necessidade do desenvolvedor.

Os blocos "try" "catch" fornecem um ótimo mecanismo para captura e tratamento de exceção, porém no momento em que a exceção é lançada, ela é transferida para fora do bloco "*try*", fazendo com que não seja possível inserir "códigos de limpeza". (SIERRA, 2008). Os códigos de limpeza são utilizados na maioria das vezes para fechar arquivos e conexões que foram abertas durante a execução do bloco "*try*". Segundo Sierra (2008) uma das soluções seria colocar esse código dentro do catch, porém isso é inviável, já que todo tratador precisaria de uma cópia do mesmo código, gerando assim um código redundante.

A clausula "*finally*" surge com o propósito de executar depois que a exceção for tratada (caso não haja *catch*, será executada depois da finalização do bloco *try*) de qualquer maneira, sendo assim você poderá utilizar códigos de limpeza no bloco "*finally*" com a certeza que o fluxo passará pelo mesmo.

Pode acontecer também de um método lançar uma exceção, porém não querer tratá-la, nesse caso o java possuí a palavra reservada "*throws*" que faz com que a exceção seja propagada para seu método chamador, e assim sucessivamente até que algum tratador a capture, caso não aconteça isso, a exceção será lançada. Além do "*throws*" existe o "*throw*" que é responsável por lançar uma determinada exceção no momento em que é acionado.

Quadro 1 - Sintaxe para Tratamento de Exceção (java/android)

Try	Palavra reservada do java conhecida como bloco "protegido" caso ocorra algum problema com código dentro do bloco, o fluxo seguirá para o bloco catch.
Catch	Palavra reservada do java que serve para capturar as exceções, recebe uma exceção como parâmetro para ser tratada pelo desenvolvedor.
Finally	Palavra reservada do java. É um bloco que contém um código que sempre será executado (caso a exceção seja lançada ou não).
Throws	Propaga a exceção para o método chamador tratar caso ele não trate.
Throw	Lança a exceção naquele momento

Fonte: Adaptado da Documentação Java

2.2.2 Bug de Tratamento de Exceção

Ainda não há um termo largamente aceito para a expressão "*Bug* de Tratamento de Exceção" (EBERT, 2013), porém para este trabalho definimos como *bugs* de tratamento de exceção, os tipos de *bugs* que surgem devido a erros dos programadores ao se usar o mecanismo de tratamento de exceção, ou até mesmo, a omissão do tratamento (total ou parcial) por parte do desenvolvedor. Também será considerado um *bug* de tratamento de exceção um código que capture uma exceção diferente da esperada, por exemplo: se uma clausula *catch* capturasse um *NullPointerException* quando deveria ser lançado um *IOException*.

A figura 3 demonstra um exemplo real de *bug* de tratamento de exceção, esse código foi um bug causado no projeto do eclipse, o problema foi reportado como uma exceção que estava sendo lançada, os desenvolvedores descobriram que o problema era com o próprio tratador de exceção, a exceção estava sendo capturada porém no próprio bloco de tratamento da exceção era lançada uma exceção devido a forma incorreta utilizada para a manipulação do *log*.

Figura 3 - Exemplo de Bug de Tratamento de Exceção

```
////////////////// ORIGINAL CODE ////////////////////
            monitor.done();
            fUpdatedExistingClassButton= true;
        } catch (JavaModelException e) {
            JUnitPlugin.log(e);
        }
    }

////////////////// PATCH CODE ////////////////////
            monitor.done();
            fUpdatedExistingClassButton= true;
        } catch (JavaModelException e) {
            String title= WizardMessages.getString
                ("NewTestSuiteWizPage.error_tile"); //$NON-NLS-1$
            String message= WizardMessages.getString
                ("NewTestSuiteWizPage.error_message"); //$NON-NLS-1$
            ExceptionHandler.handle(e, getShell(), title, message);
        }
    }
```

Fonte: Ebert, 2013.

2.3 Android

Android é um sistema operacional criado por um grupo de empresas americanas conhecidas como *Open Handset Alliance*, lideradas pela empresa multinacional Google (LECHETA, 2015). A plataforma android é *open source*, o que sengundo Gargenta (2011) oferece liberdade de acesso para que os desenvolvedores possam acrescentar e/ou alterar suas funcionalidades no desenvolvimento de suas aplicações. Além disso, o android é uma plataforma gratuita, que visa integrar tecnologias como wi-fi, GPS, sensores de velocidade dentre outros.

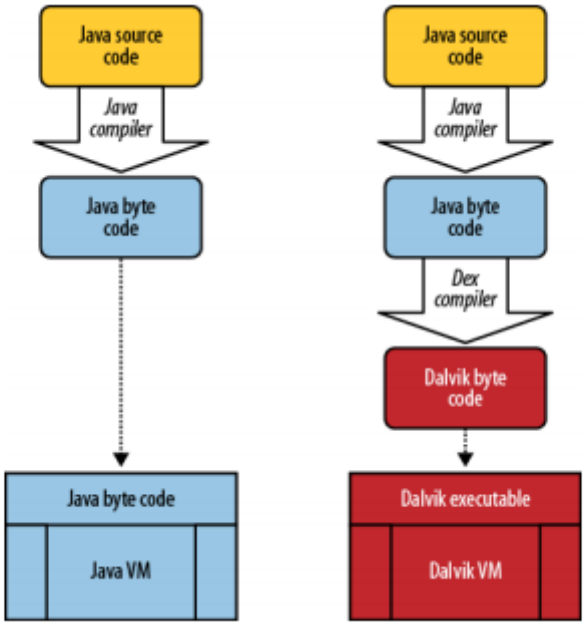
A plataforma android se baseia em uma arquitetura de camadas, onde cada camada é responsável por gerenciar os respectivos processos (LECHETA, 2009). São cinco camadas, divididas da seguinte maneira:

- Camada de Aplicação: Na camada de aplicação são localizadas os aplicativos executados no sistema como: cliente de e-mail, navegador, mapas etc.
- Camada de Framework: Nesse nível estão os programas que gerenciam as aplicações básicas do telefone como a função de realizar chamadas por exemplo.
- Camada de Bibliotecas: É a camada que contém as bibliotecas C/C++ que são utilizadas pelo sistema.

- Camada de Runtime: É a camada onde a VM (virtual machine) é instanciada, porém a máquina virtual utilizada não é a JVM do java e sim a Dalvik VM, uma máquina virtual otimizada para dispositivos com pouca memória.
- Camada Linux Kernel: Essa camada é onde o kernel 2.6 do Linux fica localizado. O android herda várias funções do kernel do Linux, dentre elas o controle de processos e o gerenciamento de memória.

O sistema operacional do android foi baseado no kernel 2.6 do Linux, e é responsável por gerenciar recursos como: memória, processos, threads, segurança dos arquivos e pastas, além de redes e drivers (LECHETA, 2015). A plataforma android utiliza a linguagem Java, como linguagem base, apesar disso, existem algumas diferenças entre as duas plataformas. A figura 4 mostra uma comparação entre o carregamento de uma aplicação na plataforma Java e uma aplicação para a plataforma android.

Figura 4 - Carregamento de Aplicação Java x Android



Fonte: Gargenta, 2011

No Java o código é escrito em sua sintaxe padrão, e convertido para *bytecode* pela JVM – *Java Virtual Machine* (Maquina Virtual Java). Segundo Gargenta (2011) no android após a compilação para o *bytecode* o código é recompilado utilizando o *"Dalvik Compiler"*

(compilador para o *"Dalvik Bytecode"*) e na sequência o *bytecode* é interpretado pela DVM – *Dalvik Virtual Machine* (Maquina Virtual Dalvik), que carrega o aplicativo no sistema.

3 PROCEDIMENTOS

Este trabalho visa mostrar se existem *bugs* de tratamento de exceção na plataforma android, quais os tipos mais recorrentes e suas principais causas, para isso, será feito um estudo exploratório baseado em uma análise de documentos nos repositórios dos projetos de código aberto (considerando que o código-fonte é um item de configuração, consequentemente um documento). Será levado em consideração a listagem da *issues* e consequentemente o código associado a mesma. Por fim os *bugs* de tratamento de exceção serão catalogados (**ANEXO A**).

3.1 Estudo Exploratório

A investigação dos repositórios de código aberto representa um desafio se levarmos em consideração os seguintes fatores:

Falta de padrão para realizar mudanças no código: Não se tem um padrão de como requisitar uma mudança para o código do projeto, muitas mudanças são realizadas baseadas em *issues* abertas pelos usuários ou responsáveis do projeto, em outros casos o desenvolvedor faz uma *"pull request"* (requisição de mudança) sem necessáriamente ter alguma *issue* aberta para resolver o problema.

Falta de organização de *labels*: Em alguns projetos não se tem muito bem definido as labels para cada atividade que identificam o tipo de recorrência como por exemplo: bug, nova funcionalidade, melhoria etc.

Issues são abertas erroneamente: O que acontece normalmente em projetos *open source* com uma grande quantidade de usuário do produto é a displicência para criação de issues no repositório, o que acaba gerando um grande número de issues que não necessariamente representam um bug ou uma funcionalidade válida para a aplicação, decorrente disso a issue geralmente é fechada sem nenhum tipo de impacto ao código final.

Pouca modularidade nos commits: Um outro grande problema é a falta de modularidade dos commits para o repositório, o que acaba gerando um trabalho mais exaustivo para encontrar o momento em que o tratamento de exceção foi utilizado/modificado, para resolver determinado problema.

Detalhamento informal sobre a issue: O detalhamento muitas vezes é negligenciado e não é auto explicativo, tornando o entendimento do problema mais complexo do que deveria em determinados momentos.

Basedo nesses fatores foi estabelicido um padrão para a realização do processo de coleta e organização dos dados no catálogo.

Para a fase inicial do processo do estudo é necessário escolher qual o repositório que será feito a análise. O Github é o repositório ecolhido para este estudo, por ser um repositório famoso entre os desenvolvedores, e possuir uma grande quantidade de projetos *open source*. O repositório apresenta ainda, uma simples e eficiente busca de projetos e de *issues* relácionadas a determinados projetos.

É necessário em seguida estabelecer padrões e requisitos para a escolha de cada projeto, para que se possa obter um resultado satisfatório de uma pesquisa empírica. Os projetos selecionados para o estudo deverão obedecer a seguinte lista de pré-requisitos:

- O projeto deverá ser de código aberto.
- O projeto deverá ter pelo menos um ano no repositório, desde de sua data de criação.
- O projeto deverá apresentar 100 (cem) ou mais *issues* cadastradas.
- O projeto deverá ter sido publicado na loja de distribuição de apps do android (googleplay).

Para a pesquisa de projetos foi utilizado um projeto do Github que visa mapear aplicativos android *open source* presentes no próprio Github. O projeto divide os aplicativos android em 16 categorias, veja na figura 5.

Figura 5 - Projetos Android Open Source do Github

Name of category	What's inside
Android TV	Apps for Android TV.
Android Wear	Apps for Android Wear.
Communication	Apps like Messager, Hangout, Gmail...
Education	Apps about education.
Finance	Apps about finance.
Game	All games for Android.
Health & Fitness	Apps about health and fitness.
LifeStyle	Apps about our life.
Multi-Media	Apps like Google Play Music, MX Player...
News & Magazines	Apps about news and magazines.
Personalization	Apps about live-wallpaper, launcher...
Productivity	Apps like Any.Do, EverNote...
Social Network	Apps like Twitter, Facebook, Github, Dribble...
Tools	Apps like Clean Master, Barcode Scanner, Keyboard...
Travel & Local	Apps about travel or local things.
Business	Apps for the improvement of your business.

Autor: Contribuidores do Repositório open-source-android-app Github

Para este trabalho foram escolhidos projetos de três diferentes categorias, com o objetivo de obter uma variedade maior de aplicações, as categoria escolhidas são: *Comunication* (Comunicação), *Tools* (Ferramentas) e *Education* (Educação). Outro fator levado em consideração para se obter um objeto de estudo mais variado é a quantidade de *issues* cadastradas por projeto. Os projetos devem ser divididos em 3 tamanhos: **pequeno porte - 100 a 499 *issues***, **médio porte - de 500 a 999**, **grande porte - acima de 1000**. Considera-se a quantidade de *issues* cadastradas no projeto e não o tamanho do projeto própriamente dito. Segue abaixo uma breve descrição dos projetos selecionados em ordem crescente de tamanho.

K-9Mail (Comunicação): é um projeto open-source de um cliente de email com busca, envio de emails via protocolo IMAP, sincronização multipla de pastas dentre outras característica.

Android IMSI Catcher Detector (Ferramentas): AIMSICD é um aplicativo android para detecção de IMSI-Catchers. Estes dispositivos são torres de celulares falsos (estações de base) que atuam entre o celular alvo e as torres reais de prestadores de serviços. Como tal, eles são considerados um (MITM) ataque *"man-in-the-middle"*. Esta tecnologia de vigilância é também conhecido como *"StingRay"* (Cellular Intercepção) e afins.

Anki Android (Educação): AnkiDroid permite que você aprenda com cartões de estudo de forma muito eficiente, mostrando-os imediatamente antes que você os esqueça. Ele é totalmente compatível com o software de repetição espaçada Anki (incluindo a sincronização), que está disponível para Windows, Linux e MacOS.

Serão analisados as *issues* dos 3 projetos, e através dessa análise serão filtradas as issues de maneira que haja apenas *bugs* relacionados a tratamento de exceção. O Github tem pesquisa que filtra as issues por palavras chaves. Para filtrar apenas issues de bugs relacionadas a tratamento de exceção, será utilizada as seguinte query:

handling **OR** exception **OR** catch

Combinações realizadas com a *string* (palavra) ***try*** se mostraram ineficientes devido à presença excessiva da *string* sendo utilizada para comentários que não tinha relação ao código fonte e dificilmente representariam um bug de tratamento de exceção, como a palavra reservada ***try*** sempre deve seguir acompanhada de uma clausula ***catch***, foi considerado que apenas o uso da palavra reservada ***catch*** seria parte da query de busca. Para catalogar as issues e identificar se a mesma possui as características de um bug relacionado a tratamento de exceção, é essencial a leitura tanto da issue quanto dos comentários e o código relacionado a ela (se houver).

Devido a necessidade de se ter um código relacionado a issue para que se possa catalogá-la corretamente, a pesquisa será feita apenas em issues com *"pull requests"* (requisição de alteração) fechados que foram aceitos, ou seja, que estejam com o status *"closed"* (fechado) e *"merged"* (palavra específica para controle de versões que representa o status de que uma alteração já foi incorporada ao código principal do projeto), por ser a única

forma de garantir que a issue tenha um código relacionado a ela, e que a *issue* não seja apenas um dúvida do desenvolvedor/usuário relacionada ao projeto.

3.2 Catálogo de Issues

O catálogo das *issues* será feito seguindo uma estrutura pré-estabelecida para separar os tipos de *issues*. Essa estrutura foi criada baseada nas informações que os repositórios utilizados neste trabalho ofereciam, foi retirado apenas o essencial para a identificação do *bug* de tratamento de exceção. A estrutura utilizada para catalogação das *issues* será a seguinte

- **Título:** Título atribuído ao criar a determinada *issue*.
- **Descrição:** Detalhes da issue fornecido pelo criador da mesma.
- **Categoria:** Categoria da *issue*, segundo definições estabelecidas logo abaixo na Tabela
- **Código:** Código de alteração do mecanismo que demonstra a situação em que o *bug* se manifesta.

Foram delimitados algumas categorias que classificariam o tipo de falha relacionada a exceção, quando esta fosse confirmada (Tabela 1).

Tabela 1 - Categorização dos Bugs de Exceções

Categorização dos Bugs de Exceção	
Categoria	**Descrição**
Exceção não tratada	Exceções lançadas, e que não foram tratadas;
Exceção não lançada	Exceções que não foram lançadas quando deveriam;
Exceção lançada ou capturada incorreta	Contextualizada como exceção errada sendo lançada ou capturada;
Problemas no tratador	Exceções tratadas com erro dentro do corpo do bloco *catch*;
Problema no bloco *finally*	Problemas ocorridos no bloco de código *finally*;

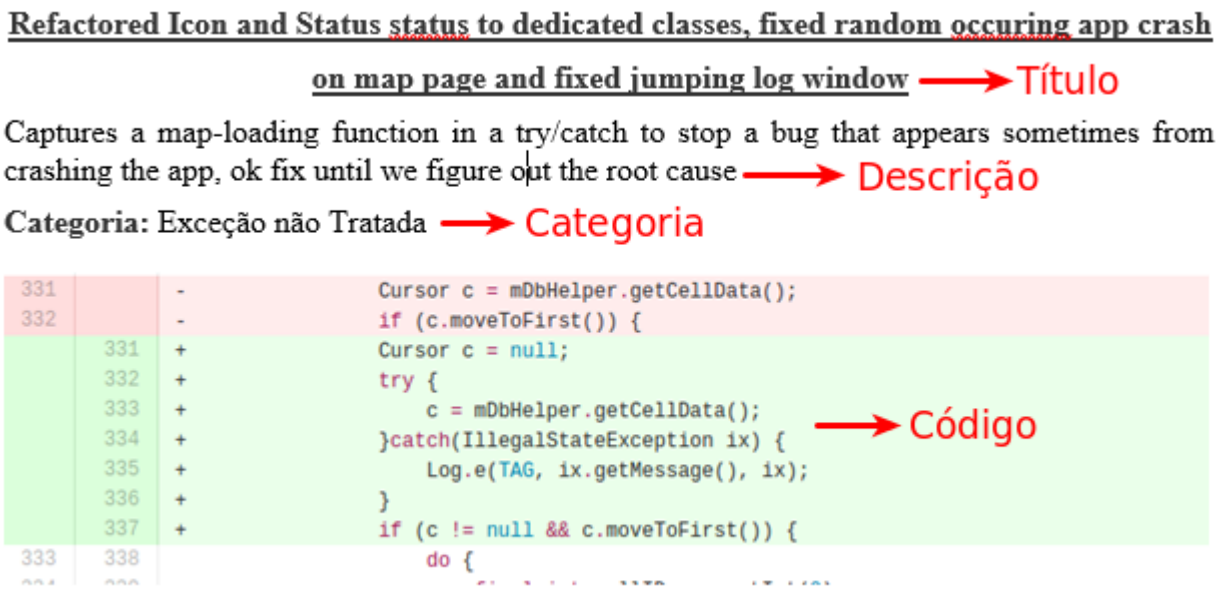
Fonte: Desenvolvido por Felipe Elbert.

Existem casos que tornam algumas *issues* do repositório para este trabalho redundante, já que 2 (duas) *querys* de busca podem retornar a mesma *issue*. As *issues* redundantes devem ser catálogadas como duplicadas, e ignoradas no momento da consolidação dos dados.

É necessário a presença do trecho de código relacionado ao tratamento de exceção, para isso deve ser incorporado ao catálogo uma imagem da *"diff"*(termo em inglês que designa a diferença entre dois estados de um mesmo código, geralmente o antes e o depois da alteração).

Na figura 4 mostra um exemplo de toda a organização estrutural da issue de bug de tratamento de exceção catálogada.

Figura 6 - Estrutura Bug de Tratamento de Exceção Catalogado

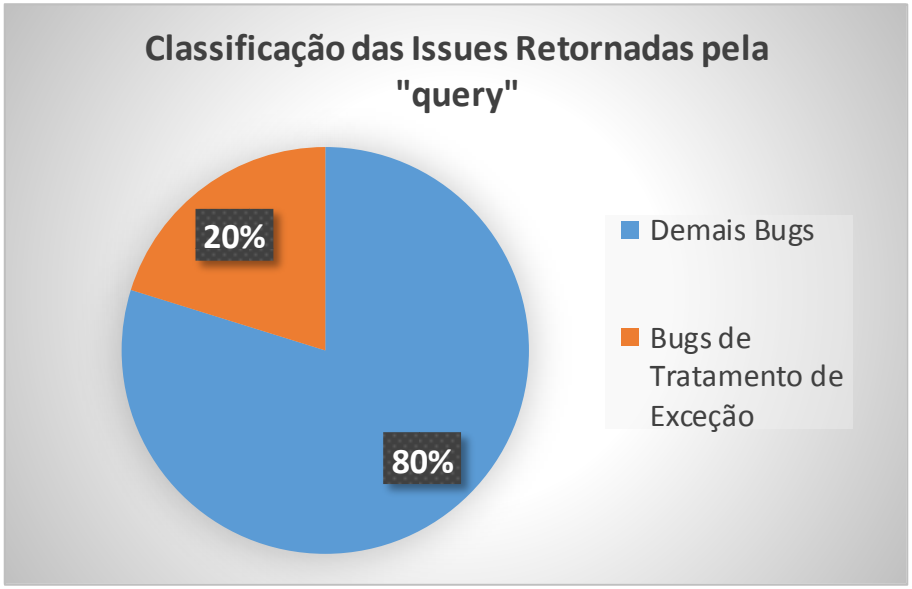


Fonte: elaborada pelo autor.

4 DESENVOLVIMENTO/RESULTADOS

A pesquisa feita levantou alguns resultados dos projetos utilizados como fonte do estudo exploratório de forma a facilitar a demonstração da relação entre *bugs* e o tratamento de exceção na plataforma android. O gráfico 1 mostra a ocorrência de *bugs* relacionados a tratamento de exceção contemplando os 3 (três) projetos utilizados como fonte de pesquisa.

Gráfico 1 - Classificação das Issues Retornadas pela Query

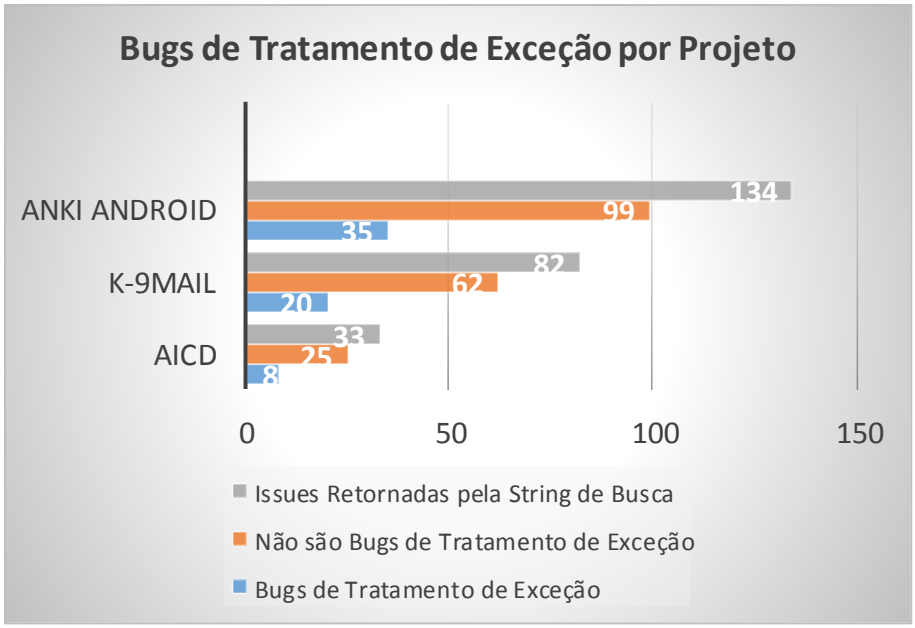


Segundo especificado neste trabalho a *string* de busca (utilizada para filtrar os *bugs* relacionados a tratamento de exceção) poderia retornar 2 (dois) resultados possíveis: **bug de tratamento de exceção** e **demais *bugs*** (não é bug de tratamento de exceção).

Podemos concluir que dos bugs retornados, 20% representavam bugs causados pelo código de tratamento de exceção, deixando claro que apesar de não representar a maioria dos *bugs* de um projeto, os bugs de tratamento de exceção são recorrentes na plataforma android.

O gráfico 2 demonstra os resultados obtidos por cada projeto com uma maior riqueza de detalhes.

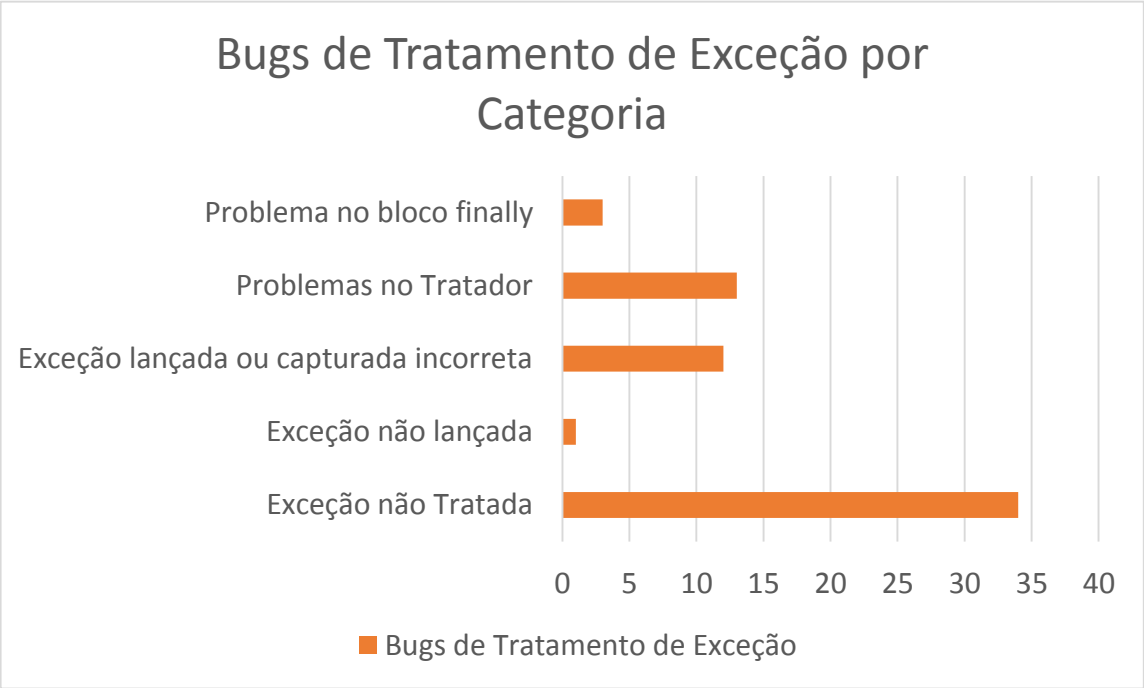
Gráfico 2 - Bugs de Tratamento de Exceção por Projeto



Este trabalho foi conduzido utilizando projetos de dimensões (tamanho do projeto) e categorias diferentes. Podemos verificar que a medida que o projeto cresce de dimensão, cresce também a quantidade de bugs retornados e consequentemente a quantidade de bugs relacionado ao código de tratamento de exceção utilizado.

Para cada *bug* de tratamento de exceção foi associado uma categoria, que especificava qual a natureza do tipo de *bug* reportado, o gráfico 3 mostra quais tipos de bugs mais recorrentes observados com este estudo.

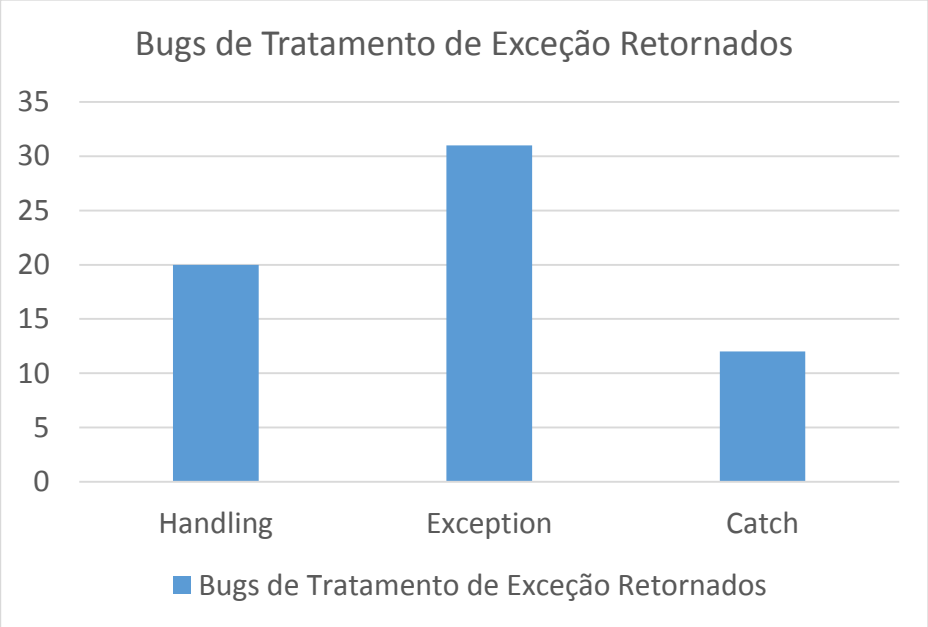
Gráfico 3 - bugs de Tratamento de Exceção por Categoria



Os tipo de bug com maior recorrência nos projetos pesquisados foi “Exceção não Tratada”, seguido de: “Problemas no Tratador”, “Exceção Lançada ou Capturada Incorreta”, “Problema no Bloco Finally” e por último “Exceção não Lançada”.

4.1 String de Busca

Embora não seja possível observar a eficiência da *query* de busca relacionada a outras *query*, apenas com os dados coletados neste projeto, podemos utilizar os dados para entender qual string foi responsável por coletar uma maior quantidade de bugs de tratamento de exceção. O gráfico 3 mostra quantos bugs foram catalogados para cada *string.*



As *query* utilizada para a busca de issues no repositório apresentava uma quantidade relevante de resultados repetidos entre as *strings*, e como as issues repetidas não foram catalogádas, não se pode ter uma precisão exata de qual string obteve a maior quantidade de bugs de tratamento de exceção, porém pode-se ter uma idéia se levar em consideração a quantidade de issues repetidas para cada string.

5 DISCUSSÃO

A abordagem exploratória utilizada para esse trabalho foi desafiadora, pois pesquisar em projetos *open source* que geralmente não possuem padrão e sua documentação é escassa, obriga ao pesquisador estabelecer padrões bem restritos para a coleta dos dados.

Apesar da dificuldade que o trabalho apresentou, pode-se comprovar que existem bugs que ocorrem na plataforma android devido ao código de tratamento de exceção, e que esses bugs acontecem em geral quando a exceção não é tratada por algum motivo, seja por negligência do desenvolvedor ou até mesmo falta de conhecimento em relação ao recurso da linguagem e como utilizá-lo da melhor maneira.

Seja em projetos de pequeno, médio ou grande porte, os *bugs* de tratamento de exceção existem, e aumentam o número de orrencias a medida que o número de *issues* cresce. Sendo

assim os bugs acontecem em situações reais, e um determinado tempo é despendido para a correção desses bugs.

6 CONSIDERAÇÕES FINAIS

A criação de uma biblioteca que executasse pré-validações durante a fase de desenvolvimento, poderia gerar um código mais robusto, com cada vez menos aplicações quebrando em tempo de execução, consequentemente despertaria um maior interesse dos desenvolvedores para que não seja necessário um retrabalho analisando novamente o mesmo código, sempre que o validador entrasse em ação.

Em um cenário ideal podemos reduzir à risco zero (ou nos aproximar desse valor), a chance de um aplicativo quebrar em tempo de execução com um *bug* de tratamento de exceção, melhorando a qualidade do produto e diminuindo o tempo despendido para análise e correção desses bugs, consequentemente diminuindo o custo do projeto.

Outro fator interessante que poderia ser estudado seria a quantidade de tempo despendida para a resolução de *bugs* relacionados ao tratamento de exceção na plataforma android, o que reforçaria a ideia de que o tratamento de excecão jamais deve ser negligenciado.

REFERÊNCIAS

ROCHA, Lincoln S. **Tratamento de Exceção em Sistemas Ubíquos: Evolução do Tratamento e Requisitos Desafiadores**. 2011. 10 f. Dissertação (Qualificação Doutorado) - Ufc, [S. l.], 2011.

BARBOSA, Eiji Adachi Medeiros. **Sistema de Recomendação para Código de Tratamento de Exceções**. 2012. 125 f. Dissertação (Mestrado) - Puc-rio, Rio de Janeiro, 2012.

EBERT, F. CASTOR, F. **A Study on Developer's Perception about Exception Handling Bugs**. Universidade Federal de Pernambuco – UFP. Pernambuco, 2013.

FILHO, F. C., GARCIA, A., RUBIRA, C. M. **Error Handling as an Aspect**. In: Proceedings of the 2nd Workshop on Best Practices in Applying Aspect-Oriented Software Development (BPAOSD'07), Vancouver, ACM, 2007. ISBN: 978-1-59593-662-2 doi>10.1145/1229485.1229486.

JONATHAN, M. **Exceções em Java**. Universidade Federal do Rio de Janeiro – UFRJ. Rio de Janeiro, 2011.

LECHETA, R. **Google Android - aprenda a criar aplicações para dispositivos móveis com o android SDK** – 2 ed – São Paulo. Novatec Editora, 2010.

LIGUORI, R. **Java 7 – Pocket Guide** – 2 ed – Sebastopol (Califórnia). O'Reilly Media, 2013.

LOPES, R**. Tratamento de Erros com Exceções.** Universidade Federal de Campina Grande – Laboratório de Sistemas Distribuídos 2009. Disponível em: <http://www.dsc.ufcg.edu.br/~raquel/p2-2009.1/excecoes.htm>. Acesso em: 22 out. 2013.

RICARDO, L. **Google Android – Aprenda a Criar Aplicações para Dispositivos Móveis Com o Android SDK –** 4 ed **-** Novatec, 2015.

RICARTE, M. **Programação Orientada a Objetos - Uma Abordagem com Java.** 2001 Universidade Estadual de Campinas - Departamento de Engenharia de Computação e Automação Industrial. Faculdade de Engenharia Elétrica e de Computação.

ROBERT, A. **Exception Handling -** FSU Department of Computer Science, 2000.

SANTOS, M. **Estendendo a Ferramenta SAFE para JBOSS AOP**. Rio de Janeiro, 2010. 147p. Dissertação de Mestrado – Departamento de Informática, Pontifícia. Universidade Católica do Rio de Janeiro.

SIERRA, K. **Certificação Sun para Programador Java 6**. Alta Books, 2008.

SOMMERVILLE, I. **Engenharia de Software.** Pearson Education, 9ª edição – 2011.

TABORDA, S. **Exceções Conceitos**. 2007. Disponível em: <http://sergiotaborda.wordpress.com/desenvolvimento-de-software/java/trabalhando-com-excecoes-conceitos/>. Acesso em: 25 nov. 2013.

GARGENTA, M. **Learning Android.** United States of America USA**,** 2011. O'Reilly Media, Inc., 1005 Gravenstein Highway North.

ANEXOS

ANEXO A – Catálogo de Issues das Aplicações Utilizadas para Estudo

Android IMSI Catcher Detector

330 Pull Requests

Handling - 10 (Pull Requests)

7 Não são bugs de tratamento de exceção

Rotation handling and prevent crash on some devices

Added some exception handling to prevent location manager crashing if a provider is not available

Categoria: Exceção não Tratada

```
51      -       lm.requestLocationUpdates(LocationManager.GPS_PROVIDER, GPS_MIN_UPDATE_TIME,
52      -               GPS_MIN_UPDATE_DISTANCE, mLocationListener);
53      -       lm.requestLocationUpdates(LocationManager.PASSIVE_PROVIDER, GPS_MIN_UPDATE_TIME,
54      -               GPS_MIN_UPDATE_DISTANCE, mLocationListener);
55      -       lm.requestLocationUpdates(LocationManager.NETWORK_PROVIDER, GPS_MIN_UPDATE_TIME,
56      -               GPS_MIN_UPDATE_DISTANCE, mLocationListener);
    51  +
    52  +       try {
    53  +           lm.requestLocationUpdates(LocationManager.GPS_PROVIDER, GPS_MIN_UPDATE_TIME,
    54  +                   GPS_MIN_UPDATE_DISTANCE, mLocationListener);
    55  +       } catch (IllegalArgumentException e) {
    56  +           // provider doesn't exist, so ignore
    57  +       }
    58  +
    59  +       try {
    60  +           lm.requestLocationUpdates(LocationManager.PASSIVE_PROVIDER, GPS_MIN_UPDATE_TIME,
    61  +                   GPS_MIN_UPDATE_DISTANCE, mLocationListener);
    62  +       } catch (IllegalArgumentException e) {
    63  +           // provider doesn't exist, so ignore
    64  +       }
    65  +
    66  +       try {
    67  +           lm.requestLocationUpdates(LocationManager.NETWORK_PROVIDER, GPS_MIN_UPDATE_TIME,
    68  +                   GPS_MIN_UPDATE_DISTANCE, mLocationListener);
    69  +       } catch (IllegalArgumentException e) {
    70  +           // provider doesn't exist, so ignore
    71  +       }
```

Fix for crash of Cell Info on Android 4.2

Fix for crash of Cell Info on Android 4.2

Categoria: Exceção não Tratada

```
12      -           buildConfigField 'String', 'OPEN_CELLID_API_KEY', '\"' + open_cellid_api_key + '\"'
    12  +           try {
    13  +               buildConfigField 'String', 'OPEN_CELLID_API_KEY', '\"' + open_cellid_api_key + '\"'
    14  +           } catch (MissingPropertyException e) {
    15  +               buildConfigField 'String', 'OPEN_CELLID_API_KEY', '"NA"'
    16  +           }
13  17          }
```

SIM Methods Exception Handling

Seems that the SIM specific TelephonyManager methods can cause some serious issues with some devices so all accesses have now been wrapped in try catch blocks to provide stability to these devices.

Categoria: Exceção não Tratada

```
358         -           mSimCountry = tm.getSimCountryIso();
    358     +           try {
    359     +               mSimCountry = tm.getSimCountryIso();
    360     +           } catch (Exception e) {
    361     +               //SIM methods can cause Exceptions on some devices
    362     +           }
359 363             }
```

Exception - 20 (Pull Requests)

15 Não são bugs de tratamento de exceção

1 Repetidas

Minor update to stop the app from crashing

The app was crashing when cliking the 'credits' button.

Categoria: Problemas no Tratador

```
46  46          } catch (Exception ee) {
47      -           log.error(ee.getMessage());
    47  +           log.error(ee.toString());
48  48          } finally {
49  49              if(reader != null) {
50  50                  try {
```

Rotation handling and prevent crash on some devices

Added some exception handling to prevent location manager crashing if a provider is not available

Categoria: Exceção não Tratada

```
51      -       lm.requestLocationUpdates(LocationManager.GPS_PROVIDER, GPS_MIN_UPDATE_TIME,
52      -               GPS_MIN_UPDATE_DISTANCE, mLocationListener);
53      -       lm.requestLocationUpdates(LocationManager.PASSIVE_PROVIDER, GPS_MIN_UPDATE_TIME,
54      -               GPS_MIN_UPDATE_DISTANCE, mLocationListener);
55      -       lm.requestLocationUpdates(LocationManager.NETWORK_PROVIDER, GPS_MIN_UPDATE_TIME,
56      -               GPS_MIN_UPDATE_DISTANCE, mLocationListener);
        51  +
        52  +       try {
        53  +           lm.requestLocationUpdates(LocationManager.GPS_PROVIDER, GPS_MIN_UPDATE_TIME,
        54  +                   GPS_MIN_UPDATE_DISTANCE, mLocationListener);
        55  +       } catch (IllegalArgumentException e) {
        56  +           // provider doesn't exist, so ignore
        57  +       }
        58  +
        59  +       try {
        60  +           lm.requestLocationUpdates(LocationManager.PASSIVE_PROVIDER, GPS_MIN_UPDATE_TIME,
        61  +                   GPS_MIN_UPDATE_DISTANCE, mLocationListener);
        62  +       } catch (IllegalArgumentException e) {
        63  +           // provider doesn't exist, so ignore
        64  +       }
        65  +
        66  +       try {
        67  +           lm.requestLocationUpdates(LocationManager.NETWORK_PROVIDER, GPS_MIN_UPDATE_TIME,
        68  +                   GPS_MIN_UPDATE_DISTANCE, mLocationListener);
        69  +       } catch (IllegalArgumentException e) {
        70  +           // provider doesn't exist, so ignore
        71  +       }
57      72      }
```

Refactored Icon and Status status to dedicated classes, fixed random occuring app crash on map page and fixed jumping log window

Captures a map-loading function in a try/catch to stop a bug that appears sometimes from crashing the app, ok fix until we figure out the root cause

Categoria: Exceção não Tratada

```
331     -               Cursor c = mDbHelper.getCellData();
332     -               if (c.moveToFirst()) {
        331 +           Cursor c = null;
        332 +           try {
        333 +               c = mDbHelper.getCellData();
        334 +           }catch(IllegalStateException ix) {
        335 +               Log.e(TAG, ix.getMessage(), ix);
        336 +           }
        337 +           if (c != null && c.moveToFirst()) {
333     338                 do {
```

Updated build for Android Studio v1.0/Gradle v2.2, implemented "Request OCID key"

The reason for the crash is homeOperator.substring(0, 3). There is no check if the homeOperator value is set properly. This will be fixed.

Categoria: Exceção não Tratada

```
101     -                       if (mCell.getMCC() == Integer.MAX_VALUE)
102     -                           mCell.setMCC(Integer.valueOf(homeOperator.substring(0, 3)));
103     -                       if (mCell.getMNC() == Integer.MAX_VALUE)
104     -                           mCell.setMNC(Integer.valueOf(homeOperator.substring(3, 5)));
        101 +                   try {
        102 +                       if (mCell.getMCC() == Integer.MAX_VALUE)
        103 +                           mCell.setMCC(Integer.valueOf(homeOperator.substring(0, 3)));
        104 +                       if (mCell.getMNC() == Integer.MAX_VALUE)
        105 +                           mCell.setMNC(Integer.valueOf(homeOperator.substring(3, 5)));
        106 +                   } catch (Exception e) {
        107 +                       Log.i(TAG, "homeOperator parse exception - " + e.getMessage());
        108 +                   }
105     109               }
```

Catch - 9 (Pull Requests)

3 Não são bugs de tratamento de exceção

5 Repetidas

Add debugging code to dump full sms in pdu format

Added try-catch so not to crash app if Exception.

Categoria: Exceção não Tratada

```
32      -               for(int xx = 0; xx < smsPdu.length;xx++) {
33      -                   String test = Integer.toHexString(smsPdu[xx] & 0xff);
34      -                   if (test.length() <= 1){test = "0"+test;}
35      -                   sb.append(test);
36      -               }full_pdu_string = sb.toString();
37      -
        33  +           try{
        34  +
        35  +                   for(int xx = 0; xx < smsPdu.length;xx++)
        36  +                   {
        37  +                       String test = Integer.toHexString(smsPdu[xx] & 0xff);
        38  +                       if (test.length() <= 1){test = "0"+test;}
        39  +                       sb.append(test);
        40  +                   }full_pdu_string = sb.toString();
        41  +
        42  +               }catch (Exception err) {
        43  +                   Log.e("SmsReceiver", "Exception Pdu smsReceiver" + err);
        44  +               }
38      45                /**
```

K-9Mail

765 Pull Requests

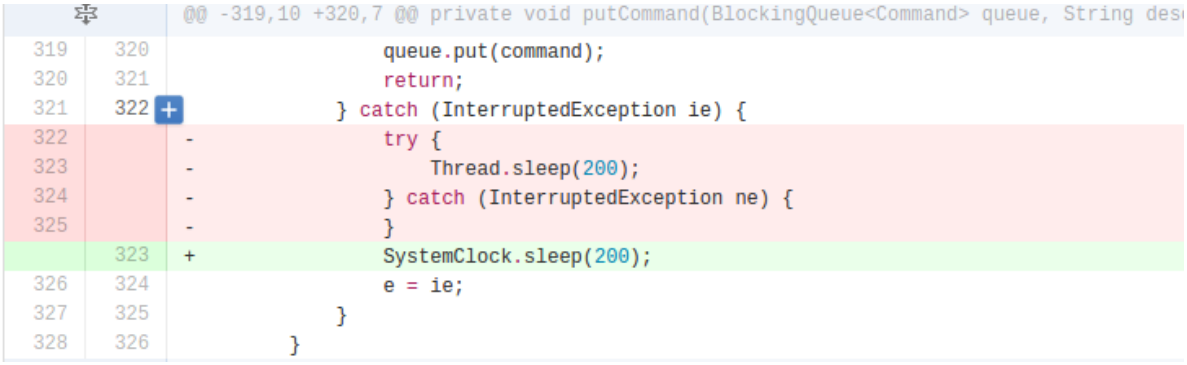
Handling - 31 (Pull Requests)

23 Não são bugs de tratamento de exceção

Replace Thread.sleep() with SystemClock.sleep()

Less LOC and correct handling of InterruptedException

Categoria: Problemas no tratador

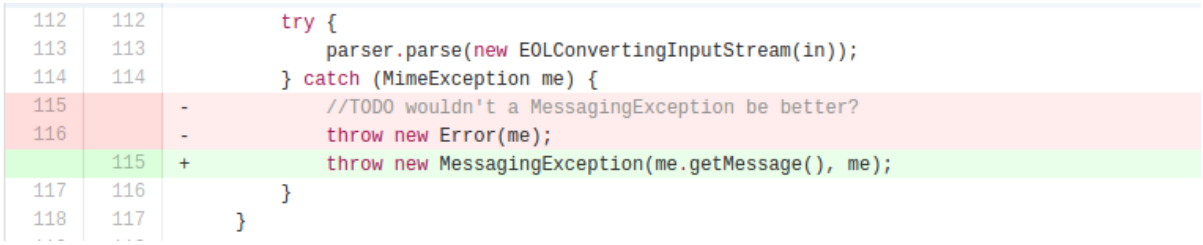


```
@@ -319,10 +320,7 @@ private void putCommand(BlockingQueue<Command> queue, String des
319 320                     queue.put(command);
320 321                     return;
321 322                 } catch (InterruptedException ie) {
322     -                   try {
323     -                       Thread.sleep(200);
324     -                   } catch (InterruptedException ne) {
325     -                   }
    323 +                   SystemClock.sleep(200);
326 324                     e = ie;
327 325                 }
328 326             }
```

Better handling of bad MIME messages

Part 1. Switch to MimeExceptions which currently get handled as synchronization failures rather than application crashes.
Part 2 should be to mark a message that couldn't be constructed as a bad message, allowing the user to user delete it.

Categoria: Problemas no tratador



```
112 112             try {
113 113                 parser.parse(new EOLConvertingInputStream(in));
114 114             } catch (MimeException me) {
115     -               //TODO wouldn't a MessagingException be better?
116     -               throw new Error(me);
    115 +           throw new MessagingException(me.getMessage(), me);
117 116             }
118 117         }
```

Improve handling if crypto provider is not available

This improves handling of error states in the compose dialog, allowing the user to retry if the connection to the crypto provider fails and displaying error messages for various error states. Fixes #1050, among other things. I'm still not 100% happy with the error handling, but this should be okay for a beta release.

Categoria: Exceção não Tratada

```
259       -       Cursor cursor = getContext().getContentResolver().query(queryUri, PROJECTION_CRYPTO_STATUS, null,
260       -               recipientAddresses, null);
          260 +   try {
          261 +       cursor = getContext().getContentResolver().query(queryUri, PROJECTION_CRYPTO_STATUS, null,
          262 +               recipientAddresses, null);
          263 +   } catch (SecurityException e) {
          264 +       return;
          265 +   }
261       266
262       267       initializeCryptoStatusForAllRecipients(recipientMap);
```

Break migration down into methods, clean up some warnings in mailstore

What the title says. this pr contains no actual semantic changes, I only moved code around, got rid of warnings along the way, and made some methods static and reduced visibility here and there, while reading up on LocalFolder and StoreSchemaDefinition

Categoria: Exceção lançada ou capturada incorreta

```
186       -       } catch (UnavailableStorageException e) {
187       -           mReadLock.unlock();
188       -           throw e;
189       -       } catch (RuntimeException e) {
          186 +   } catch (UnavailableStorageException | RuntimeException e) {
190       187           mReadLock.unlock();
191       188           throw e;
192       189       }
```

Improve send failure handling

MessagingController doesn't need to know the details about which SMTP reply codes correspond to permanent errors. The code in place wasn't working as intended for quite a while now. Overall the code is still a mess, but the error handling should be a little bit better and the code a bit more readable now.

Categoria: Exceção lançada ou capturada incorreta

```
504       503       if (mLargestAcceptableMessage > 0 && message.hasAttachments()) {
505       504           if (message.calculateSize() > mLargestAcceptableMessage) {
506       -               MessagingException me = new MessagingException("Message too large for server");
507       -               //TODO this looks rather suspicious... shouldn't it be true?
508       -               me.setPermanentFailure(possibleSend);
509       -               throw me;
          505 +           throw new MessagingException("Message too large for server", true);
510       506           }
511       507       }
```

Addition of number format handling to Storage.java

Amended Storage.java for issue 3714 to add NumberFormatException protection

Categoria: Exceção não Tratada

```
324     -       return Float.parseFloat(val);
        324 +   try{
        325 +          return Float.parseFloat(val);
        326 +   } catch (NumberFormatException nfe){
        327 +       return defValue;
        328 +   }
325     329     }
```

Issue 1303: can't send mail get "no route to host" error

Catch SocketException instead of ConnectException => cover "no route to host". Do the same handling of multiple IPv6+IPv4 adresses we do in SMTP in IMAP too

Categoria: Exceção lançada ou capturada incorreta

```
2135    -
2136    -           SocketAddress socketAddress = new InetSocketAddress(mSettings.getHost(), mSettings.getPort());
2137    -
2138    -           if (K9.DEBUG)
2139    -               Log.i(K9.LOG_TAG, "Connection " + getLogId() + " connecting to " + mSettings.getHost() + " @ IP addr
2140    -
2141    -           if (mSettings.getConnectionSecurity() == CONNECTION_SECURITY_SSL_REQUIRED ||
2142    -                   mSettings.getConnectionSecurity() == CONNECTION_SECURITY_SSL_OPTIONAL) {
2143    -               SSLContext sslContext = SSLContext.getInstance("TLS");
2144    -               final boolean secure = mSettings.getConnectionSecurity() == CONNECTION_SECURITY_SSL_REQUIRED;
2145    -               sslContext.init(null, new TrustManager[] {
2146    -                               TrustManagerFactory.get(mSettings.getHost(), secure)
2147    -                       }, new SecureRandom());
2148    -               mSocket = sslContext.getSocketFactory().createSocket();
2149    -               mSocket.connect(socketAddress, SOCKET_CONNECT_TIMEOUT);
2150    -           } else {
2151    -               mSocket = new Socket();
2152    -               mSocket.connect(socketAddress, SOCKET_CONNECT_TIMEOUT);
        2136 +         // -> try all IPv4 and IPv6 addresses of the host.
        2137 +         int mConnectionSecurity = mSettings.getConnectionSecurity();
        2138 +         InetAddress[] addresses = InetAddress.getAllByName(mSettings.getHost());
        2139 +          for (int i = 0; i < addresses.length; i++) {
        2140 +              try {
        2141 +                 if (K9.DEBUG && K9.DEBUG_PROTOCOL_IMAP) {
        2142 +                         Log.d(K9.LOG_TAG, "connecting to " + mSettings.getHost() + " as " + addresses[i]);
        2143 +                 }
        2144 +                  SocketAddress socketAddress = new InetSocketAddress(addresses[i], mSettings.getPort());
        2145 +                  if (mConnectionSecurity == CONNECTION_SECURITY_SSL_REQUIRED ||
        2146 +                          mConnectionSecurity == CONNECTION_SECURITY_SSL_OPTIONAL) {
        2147 +                      SSLContext sslContext = SSLContext.getInstance("TLS");
        2148 +                      boolean secure = mConnectionSecurity == CONNECTION_SECURITY_SSL_REQUIRED;
        2149 +                      sslContext.init(null, new TrustManager[] {
        2150 +                                      TrustManagerFactory.get(mSettings.getHost(), secure)
        2151 +                              }, new SecureRandom());
        2152 +                      mSocket = sslContext.getSocketFactory().createSocket();
        2153 +                      mSocket.connect(socketAddress, SOCKET_CONNECT_TIMEOUT);
        2154 +                  } else {
        2155 +                      mSocket = new Socket();
        2156 +                      mSocket.connect(socketAddress, SOCKET_CONNECT_TIMEOUT);
        2157 +                  }
        2158 +              } catch (SocketException e) {
        2159 +                  if (i < (addresses.length - 1)) {
        2160 +                      // there are still other addresses for that host to try
        2161 +                      continue;
        2162 +                  }
        2163 +                  throw new MessagingException("Cannot connect to host", e);
        2164 +              }
        2165 +              break; // connection success
```

Improve handling of From: Headers without parseable email addresses

Improve handling of From: Headers without parseable email addresses (fixes 3814)

Categoria: Problemas no tratador

```
150  150          } catch (MimeException pe) {
151  151              Log.e(K9.LOG_TAG, "MimeException in Address.parse()", pe);
     152  +           //but we do an silent failover : we just use the given string as name with empty address
     153  +           addresses.add(new Address("", addressList));
152  154          }
153  155          return addresses.toArray(EMPTY_ADDRESS_ARRAY);
154  156      }
```

Exception - 40 (Pull Requests)

27 Não são bugs de tratamento de exceção

5 Repetidas

Crash app when migrations fail in debug build

I don't want to change the current behavior of recreating the database in case of an unexpected exception during migration. However, when using debug builds we rather want the app to crash than our database containing the data to trigger the error wiped away.

Categoria: Problemas no tratador

```
437      -           } catch (Throwable t) {
438      -               Log.e(K9.LOG_TAG, "Error while testing settings", t);
439      -               showErrorDialog(
440      -                       R.string.account_setup_failed_dlg_server_message_fmt,
441      -                       (t.getMessage() == null ? "" : t.getMessage()));
         437  +       } catch (Exception e) {
         438  +           Log.e(K9.LOG_TAG, "Error while testing settings", e);
         439  +           String message = e.getMessage() == null ? "" : e.getMessage();
         440  +           showErrorDialog(R.string.account_setup_failed_dlg_server_message_fmt, message);
442      441          }
```

Prevent exceptions in test when account is deleted

No argument provided

Categoria: Exceção não Tratada

```
3597  3600              } catch (MessagingException me) {
3598  3601                  Log.e(K9.LOG_TAG, "Count not get unread count for account " +
3599  3602                          account.getDescription(), me);
      3603  +           } catch (RuntimeException e) {
      3604  +               e.addSuppressed(eCallingThread);
      3605  +               throw e;
3600  3606  +         }
3601  3607
3602  3608          }
```

Don't throw CertificateValidationException for all SSLExceptions

An interrupted connection attempt to the server yields an SSLException as well

Categoria: Exceção lançada ou capturada incorreta

```
2619      -              throw new CertificateValidationException(e.getMessage(), e);
     2620 +              if (e.getCause() instanceof CertificateException) {
     2621 +                  throw new CertificateValidationException(e.getMessage(), e);
     2622 +              } else {
     2623 +                  throw e;
     2624 +              }
2620 2625           } catch (GeneralSecurityException gse) {
2621 2626               throw new MessagingException(
2622 2627                   "Unable to open connection to IMAP server due to security error.", gse);
```

Some kernels have frandom and erandom

When a users kernel has frandom and erandom this would cause the whole app to crash. When your kernel supports this frandom will replace random and erandom will replace urandom. The advantage here is that it's faster and that erandom doesn't use up any entropy at all. This helps to remove jank from Android.

Categoria: Problemas no tratador

```
217  217            } catch (IOException e) {
218       -              throw new SecurityException(
219       -                      "Failed to mix seed into " + URANDOM_FILE, e);
     218 +              mSeeded = true;
220  219            }
```

Fix third party app access to provider

K-9 4.3xx have broken third party unread count readers, such as TeslaUnread/Nova Launcher and MailListWidget for K-9 . This fixes the issue and also fixes a leaked semaphore.

Categoria: Problema no bloco finally

```
828       -          final Cursor cursor;
829       -          cursor = mDelegate.query(uri, projection, selection, selectionArgs, sortOrder);
     833 +          Cursor cursor = null;
     834 +          try {
     835 +              cursor = mDelegate.query(uri, projection, selection, selectionArgs, sortOrder);
     836 +          } finally {
     837 +              if (cursor == null) {
     838 +                  mSemaphore.release();
     839 +              }
     840 +          }
```

StrictMode stacktrace since stream was not closed properly

When the device lost connectivity, a StrictMode error could be thrown since the stream had not been closed.

Categoria: Exceção não Tratada

```
39  39  +       InputStream is = null;
40  40  +       FileInputStream fis = null;
39  41          try {
40      -           return new BinaryTempFileBodyInputStream(new FileInputStream(mFile));
    42  +           fis = new FileInputStream(mFile);
    43  +           is = new BinaryTempFileBodyInputStream(fis);
    44  +           return is;
41  45          } catch (IOException ioe) {
    46  +           try { if (fis != null) { fis.close(); } } catch (IOException e) { }
    47  +           try { if (is != null) { is.close(); } } catch (IOException e) { }
42  48              throw new MessagingException("Unable to open body", ioe);
43  49          }
```

Close resources properly

Java has exceptions and resources should always be closed

Categoria: Problema no bloco finally

```
503     -           FileOutputStream out = new FileOutputStream(to);
504     -           byte[] buffer = new byte[1024];
505     -           int count = -1;
506     -           while ((count = in.read(buffer)) > 0) {
507     -               out.write(buffer, 0, count);
    503 +           try {
    504 +               FileOutputStream out = new FileOutputStream(to);
    505 +               try {
    506 +                   byte[] buffer = new byte[1024];
    507 +                   int count = -1;
    508 +                   while ((count = in.read(buffer)) > 0) {
    509 +                       out.write(buffer, 0, count);
    510 +                   }
    511 +               } finally {
    512 +                   out.close();
    513 +               }
    514 +           } finally {
    515 +               try { in.close(); } catch (Throwable ignore) {}
508 516             }
509     -           out.close();
510     -           in.close();
511 517             from.delete();
512 518             return true;
513 519         } catch (Exception e) {
```

Introduce and use Utility.closeQuietly(Cursor)

This helper is in the same spirit as IOUtils.closeQuietly.

Categoria: Problema no bloco finally

```
231  231          } catch (Exception e) {
232  232              Log.e(K9.LOG_TAG, "Failed to get email data", e);
233  233          } finally {
234       -          if (cursor != null) {
235       -              cursor.close();
236       -          }
237       -          if (cursor2 != null) {
238       -              cursor2.close();
239       -          }
       234  +          Utility.closeQuietly(cursor);
       235  +          Utility.closeQuietly(cursor2);
240  236          }
```

Catch - 24 (Pull Requests)

12 Não são bugs de tratamento de exceção

8 Repetidas

Make the migration more resilient

If a single mail migration fails, that's no reason to reset the entire database. I added a catch-all, and an upper limit of failed mails before the migration routine gives up. A fixed limit might not be the best option here, but it's simple and I couldn't think of a very good less-simple solution.

Categoria: Exceção não Tratada

```
86   89  +              try {
87       -                  updateFlagsForMessage(db, messageId, messageFlags, migrationsHelper);
88       -                  MimeHeader mimeHeader = loadHeaderFromHeadersTable(db, messageId);
       90  +              structureState = MimeStructureState.getNewRootState();
89   91
90       -                  MimeStructureState structureState = MimeStructureState.getNewRootState();
       92  +              MimeHeader mimeHeader = loadHeaderFromHeadersTable(db, messageId);
91   93
92   94                  boolean messageHadSpecialFormat = false;
93   95
     @@ -117,14 +119,21 @@ public static void db51MigrateMessageFormat(SQLiteDatabase db, MigrationsHelper
117  119                                          htmlContent, textContent, mimeHeader, structureState);
118  120                      }
119  121                  }
       122  +          } catch (Exception e) {
       123  +              Log.e(K9.LOG_TAG, "error inserting into database! (removing remnant message parts)", e);
       124  +
```

Improve behaviour when an error occurs extracting part of a message

Change the behaviour of MessageExtractor to no longer absorb MessagingExceptions.
Change the behaviour of MessageExtractor to no longer catch OOM exceptions.
Change the behaviour of MessageExtractor to wrap IOExceptions it generates in a MessagingException instead of absorbing them

Categoria: Problemas no tratador

```
88  -
89  -       } catch (OutOfMemoryError oom) {
90  -           /*
91  -            * If we are not able to process the body there's nothing we can do about it. Return
92  -            * null and let the upper layers handle the missing content.
93  -            */
94  -           Log.e(LOG_TAG, "Unable to getTextFromPart " + oom.toString());
95  -       } catch (Exception e) {
96  -           /*
97  -            * If we are not able to process the body there's nothing we can do about it. Return
98  -            * null and let the upper layers handle the missing content.
99  -            */
    92  +       } catch (IOException e) {
100 93            Log.e(LOG_TAG, "Unable to getTextFromPart", e);
    94  +           throw new MessagingException("Unable to get text from part: "+part, e);
101 95        }
102 -       return null;
```

In saslAuthPlain: if AUTHENTICATE PLAIN fails, try LOGIN

Authentication with just username for the personal mailbox works fine but not for the special mailbox with the backslash syntax above.

Categoria: Exceção não Tratada

```
366 366
    367 +   protected void saslAuthPlainWithLoginFallback() throws IOException, MessagingException {
    368 +       try {
    369 +           saslAuthPlain();
    370 +       } catch (AuthenticationFailedException e) {
    371 +           login();
    372 +       }
    373 +   }
    374 +
367 375       protected ImapResponse readContinuationResponse(String tag)
```

Catch IllegalCharsetNameException causing force-close on unsupported japa

Catch IllegalCharsetNameException causing force-close on unsupported japanese charsets (issue 3272)

Categoria: Exceção não Tratada

```
1384 1386         */
1385 -       if (!Charset.isSupported(charset)) {
     1387 +       try {
     1388 +           if (!Charset.isSupported(charset)) {
     1389 +               Log.e(K9.LOG_TAG, "I don't know how to deal with the charset " + charset +
     1390 +               ". Falling back to US-ASCII");
     1391 +               charset = "US-ASCII";
     1392 +           }
     1393 +       } catch (IllegalCharsetNameException e) {
1386 1394            Log.e(K9.LOG_TAG, "I don't know how to deal with the charset " + charset +
1387 1395            ". Falling back to US-ASCII");
1388 1396            charset = "US-ASCII";
1389 1397        }
```

Anki Android

1255 Pull Requests

Handling - 40 (Pull Requests)

31 Não são bugs de tratamento de exceção

Check for IllegalStateException when opening DB due to 1MB cursor limit

Fix for java.lang.IllegalStateException: Couldn't read row 0, col 0 from CursorWindow. Make sure the Cursor is initialized correctly before accessing data from it.

Categoria: Exceção não Tratada

```
138 +    public synchronized boolean exceededCursorSizeLimit(Context context) {
139 +        try {
140 +            getCol(context);
141 +        } catch (IllegalStateException e) {
142 +            return true;
143 +        }
144 +        return false;
145 +    }
146 +
```

Don't report duplicate caught crashes

Each time a caught exception is up for reporting, we check if the hash already exists, and if it does we skip sending it to our server. If not, it's added to the cache. Uncaught exceptions are unaffected as that would require modifications to Acra, and it's probably OK to prioritize those crashes anyway.

Categoria: Exceção não Tratada

```
249 +        SharedPreferences prefs = getSharedPrefs(getInstance());
250 +        // Only send report if we have not sent an identical report before
251 +        try {
252 +            JSONObject sentReports = new JSONObject(prefs.getString("sentExceptionReports", "{}"));
253 +            String hash = getExceptionHash(e);
254 +            if (sentReports.has(hash)) {
255 +                Timber.i("The exception report with hash %s has already been sent from this device", hash);
256 +                return;
257 +            } else {
258 +                sentReports.put(hash, true);
259 +                prefs.edit().putString("sentExceptionReports", sentReports.toString()).apply();
260 +            }
261 +        } catch (JSONException e1) {
262 +            Timber.i(e1, "Could not get cache of sent exception reports");
263 +        }
```

Show a media check option on failed media sync

Fix for java.lang.RuntimeException: java.lang.RuntimeException: SyncError:Can't add 0 byte file.

at com.ichi2.libanki.sync.MediaSyncer.sync(MediaSyncer.java:227)

at com.ichi2.async.Connection.doInBackgroundSync(Connection.java:393)

Categoria: Problemas no Tratador

```
219  217          } catch (JSONException e) {
220  218              throw new RuntimeException(e);
221       -      } catch (APIVersionException e) {
222       -          UnsupportedSyncException ee = new UnsupportedSyncException("Cannot sync
223       -          Timber.e(e.getMessage());
224       -          throw ee;
225  219          } catch (Exception e) {
```

Purge temporary media sync files and improve stream management

Better exception handling. Don't catch-all or swallow exceptions.

Categoria: Exceção lançada ou capturada incorreta

```
239       -          } catch (Exception e) {
      243 +          } catch (IOException e) {
      244 +              Timber.e(e, "Error downloading media files");
      245 +              throw new RuntimeException(e);
      246 +          } catch (UnknownHttpResponseException e) {
240   247              Timber.e(e, "Error downloading media files");
241   248              throw new RuntimeException(e);
242   249          }
```

Db error handling

Don't bother trying to auto-repair as it wasn't working well. Only show "repair" option in dialog when sqlite3 installed. Reword error dialog

Categoria: Exceção não Tratada

```
93  +      boolean sqliteInstalled = false;
94  +      try {
95  +          sqliteInstalled = Runtime.getRuntime().exec("sqlite3 --version").waitFor() == 0;
96  +      } catch (IOException e) {
97  +          e.printStackTrace();
98  +      } catch (InterruptedException e) {
99  +          e.printStackTrace();
100 +      }
```

Switch error reporting over to ACRA @ ankidroid.org

This PR switches the AnkiDroid error reporting over to ACRA / Acralyzer on ankidroid.org and makes the error reporting occur automatically in the background by default (a toast is shown so the user knows). This is not quite ready to merge as I need to finalize some details about the couchdb authentication with@agrueneberg

Categoria: Exceção não Tratada

```
+            try {
+                if (value.equals(Feedback.REPORT_ALWAYS)) {
+                    ACRA.getConfig().setMode(ReportingInteractionMode.TOAST);
+                    ACRA.getConfig().setResToastText(R.string.feedback_auto_toast_text);
+                } else if (value.equals(Feedback.REPORT_ASK)) {
+                    ACRA.getConfig().setMode(ReportingInteractionMode.DIALOG);
+                    ACRA.getConfig().setResToastText(R.string.feedback_manual_toast_text);
+                }
+            } catch (ACRAConfigurationException e) {
+                Log.e(AnkiDroidApp.TAG, "Could not set ACRA report mode");
+            }
```

Media sync protocol

A change to disable media syncing on Android 2.2 and below if any of the media files are non-ascii. Since we're already going through the compat class, it was relatively unobtrusive to bubble up a checked exception to the syncer and disable syncing form there.

Categoria: Exceção não Tratada

```
2234        -        return AnkiDroidApp.getCompat().nfcNormalized(answerText);
        2235 +        try {
        2236 +            return AnkiDroidApp.getCompat().nfcNormalized(answerText);
        2237 +        } catch (APIVersionException e) {
        2238 +            return answerText;
        2239 +        }
2235    2240      }
```

Add odid fix to db check

Added another missing but similar block in the integrity check. The addition of this fix allows us to remove a workaround in Sched. When this problem is now triggered, it will bubble up to the general-purpose exception handler in the reviewer which offers the option to do a database check where it can be fixed. I've tested this with a collection that exhibits this particular issue. The fix here is fine but the dialog for handling the error is broken, so I'll see what I can do with that.

Categoria: Problemas no tratador

```
1788          } catch (JSONException e) {
-                 if (!mCol.getDecks().isDyn(card.getDid()) && card.getODid() != 0) {
-                         // workaround, if a card's deck is a normal deck, but odid != 0
-                         card.setODue(0);
-                         card.setODid(0);
-                         AnkiDroidApp.saveExceptionReportFile(e, "fixedODidInconsistencyInSched_lapseConf");
-                         // return proper value after having fixed the problem
-                         return _lapseConf(card);
-                 } else {
-                     throw new RuntimeException(e);
-                 }
1789 +               throw new RuntimeException(e);
1790          }
1791      }
```

Guard against deck deletion crashes; speed up cram/filter deck deletion

there was in fact an issue in the code path now skipped. It came down to trying to access a sql result that was null. In the exception handler, I added a specific check to see if a null was encountered, and if so attempt to simply move on.

Categoria: Exceção não Tratada

```
201       -               // The magical line. Almost as illegible as python code ;)
202       -               results.add(type.cast(Cursor.class.getMethod(methodName, int.class).invoke(cursor, column)));
      201 +               try {
      202 +                   // The magical line. Almost as illegible as python code ;)
      203 +                   results.add(type.cast(Cursor.class.getMethod(methodName, int.class).invoke(cursor, column)));
      204 +               } catch (InvocationTargetException e) {
      205 +                   // this is just a guard against null generated crashes, but
      206 +                   // it does not ensure the query isn't expecting to also receive null results for processing
      207 +                   if (cursor.getType(column) == Cursor.FIELD_TYPE_NULL) {
      208 +                       Log.d(AnkiDroidApp.TAG, "AnkiDb.queryColumn: Located null during exception. Attempting to skip r
      209 +                       continue; // quite likely a null issue, just skip it
      210 +                   } else {
      211 +                       throw new RuntimeException(e);
      212 +                   }
      213 +               }
203   214           }
```

Exception - 68 (Pull Requests)

33 Não são bugs de tratamento de exceção

16 Repetidas

addNewDeck() should throw exception if deckName already exists

I made that change regarding Decks.id() accepting a desc - seems much better.

Categoria: Exceção não lançada

```
935     -               did = col.getDecks().id(deckName);
936     -               JSONObject deck = col.getDecks().get(did);
937     -               if (deck != null) {
938     -                   try {
939     -                       Integer deckDyn = values.getAsInteger(FlashCardsContract.Deck.DECK_DYN);
940     -                       if (deckDyn != null) {
941     -                           deck.put("dyn", deckDyn);
942     -                       }
943     -                       String deckDesc = values.getAsString(FlashCardsContract.Deck.DECK_DESC);
944     -                       if (deckDesc != null) {
945     -                           deck.put("desc", deckDesc);
946     -                       }
947     -                   } catch (JSONException e) {
948     -                       Timber.e(e, "Could not set a field of new deck %s", deckName);
949     -                       return null;
950     -                   }
        935 +               did = col.getDecks().id(deckName, false);
        936 +               if (did != null) {
        937 +                   throw new IllegalArgumentException("Deck name already exists: " + deckName);
        938 +               }
        939 +               did = col.getDecks().id(deckName, values.getAsString(FlashCardsContract.Deck.DECK_DESC));
```

Fix bug in getNote() that meant it only ever returned null or threw exception

Fixed silly bug in getNote()

Categoria: Exceção não Tratada

```
236     -       String[] selectionArgs = {String.format("%s=%d", FlashCardsContract.Note._ID, noteId)};
237     -       Cursor cursor = mResolver.query(FlashCardsContract.Note.CONTENT_URI, PROJECTION, null, selectionArgs, null);
        236 +       Uri noteUri = Uri.withAppendedPath(FlashCardsContract.Note.CONTENT_URI, Long.toString(noteId));
        237 +       Cursor cursor = mResolver.query(noteUri, PROJECTION, null, null, null);
238     238         if (cursor == null) {
239     239             return null;
240     240         }
241     241         try {
        242 +           if (!cursor.moveToNext()) {
        243 +               return null;
        244 +           }
242     245             return NoteInfo.buildFromCursor(cursor);
243     246         } finally {
244     247             cursor.close();
```

Ignore exceptions caused by the widget

I saw this in the beta3 logs

java.lang.NullPointerException: Attempt to invoke virtual method 'long java.lang.Long.longValue()' on a null object reference

Categoria: Exceção não Tratada

```
92      -           mSmallWidgetStatus = getCounts(context);
        92  +       try {
        93  +           updateCounts(context);
        94  +       } catch (Exception e) {
        95  +           Timber.e(e, "Could not update widget");
        96  +       }
93      97          return context;
94      98      }
```

Support Custom URLs

Catch ActivityNotFoundException exceptions, rather than crashing

Categoria: Exceção não Tratada

```
1522      -              startActivityWithoutAnimation(intent);
      1517 +              try {
      1518 +                  startActivityWithoutAnimation(intent);
      1519 +              } catch(ActivityNotFoundException e) {
      1520 +                  e.printStackTrace(); // Don't crash if the intent is not handled
      1521 +              }
1523  1522                  return true;
```

Catch runtime exceptions when trying to enable cookies

It seems some devices crash when trying to enable cookies. A fix has been pushed into 2.4.3 a while ago, this adds it to 2.5

Categoria: Exceção não Tratada

```
14        -          CookieManager.setAcceptFileSchemeCookies(true);
      16  +          try {
      17  +              CookieManager.setAcceptFileSchemeCookies(true);
      18  +          } catch (Throwable e) {
      19  +              Timber.e(e, "Runtime exception enabling cookies");
      20  +          }
15    21        }
```

Fix progress dialog dismiss

Try to fix NPE and WindowLeaked exceptions due to progress dialog.

Categoria: Exceção lançada ou capturada incorreta

```
1328  1331              try {
1329      -               if (mProgressDialog != null) {
      1332  +           if (mProgressDialog != null && mProgressDialog.isShowing()) {
1330  1333                  mProgressDialog.dismiss();
1331  1334              }
1332  1335          } catch (IllegalArgumentException e) {
```

Catch exceptions with ShowcaseView

There seem to be a bunch of Gingerbread users experiencing problems with ShowcaseView.

Categoria: Exceção não Tratada

```
522     -           ActionItemTarget target = new ActionItemTarget(this, R.id.action_add_decks);
523     -           mShowcaseDialog = new ShowcaseView.Builder(this).setTarget(target)
524     -                   .setContentTitle(res.getString(R.string.studyoptions_welcome_title))
525     -                   .setStyle(R.style.ShowcaseView_Light).setShowcaseEventListener(this)
526     -                   .setOnClickListener(new OnClickListener() {
527     -                       @Override
528     -                       public void onClick(View v) {
529     -                           mShowcaseDialog.hide();
530     -                           Intent helpIntent = new Intent("android.intent.action.VIEW", Uri.parse(res
531     -                                   .getString(R.string.link_manual_getting_started)));
532     -                           startActivityWithoutAnimation(helpIntent);
533     -                       }
534     -                   }).setContentText(res.getString(R.string.add_content_showcase_text)).hideOnTouchOutside().build();
535     -           mShowcaseDialog.setButtonText(getResources().getString(R.string.help_title));
    522 +           try {
    523 +               ActionItemTarget target = new ActionItemTarget(this, R.id.action_add_decks);
    524 +               mShowcaseDialog = new ShowcaseView.Builder(this).setTarget(target)
    525 +                       .setContentTitle(res.getString(R.string.studyoptions_welcome_title))
    526 +                       .setStyle(R.style.ShowcaseView_Light).setShowcaseEventListener(this)
    527 +                       .setOnClickListener(new OnClickListener() {
    528 +                           @Override
    529 +                           public void onClick(View v) {
    530 +                               mShowcaseDialog.hide();
    531 +                               Intent helpIntent = new Intent("android.intent.action.VIEW", Uri.parse(res
    532 +                                       .getString(R.string.link_manual_getting_started)));
    533 +                               startActivityWithoutAnimation(helpIntent);
    534 +                           }
    535 +                       }).setContentText(res.getString(R.string.add_content_showcase_text)).hideOnTouchOutside().build(
    536 +               mShowcaseDialog.setButtonText(getResources().getString(R.string.help_title));
    537 +           } catch (Exception e) {
    538 +               Timber.e(e, "Error showing ShowcaseView");
    539 +               Themes.showThemedToast(this, res.getString(R.string.add_content_showcase_text), false);
    540 +           }
```

Send exception to server when apkg import fails due to content provider

Hopefully this will allows us to narrow down the issue a bit more

Categoria: Exceção não Tratada

```
93      -                   e.printStackTrace();
    92  +                   AnkiDroidApp.saveExceptionReportFile(e, "IntentHandler.java", "apkg import failed: " + filename)
94  93                  } catch (IOException e2) {
95  94                      errorMessage=e2.getLocalizedMessage();
96      -                   e2.printStackTrace();
    95  +                   AnkiDroidApp.saveExceptionReportFile(e2, "IntentHandler.java", "apkg import failed" + filename);
```

Fix crash when play button is pressed twice

Tricky beast: if the Play button is pressed without recording something first, the playback will fail and the state will change to STOPPED. When Play is clicked again, it will try to seek to 0, and throw an exception because there is no media file. The exception is caught, but doesn't contain a message, which crashesLog.d.

Categoria: Problemas no tratador

```
238 238                     } catch (Exception e) {
239 239                         Log.e("AudioView", e.getMessage());
240 240                         showToast(gtxt(R.string.multimedia_editor_audio_view_playing_failed));
241     -                       mStatus = Status.STOPPED;
    241 +                       mStatus = Status.IDLE;
242 242                     }
```

Catch timeout exceptions on login

Fix Catch timeout exceptions on login

Categoria: Exceção lançada ou capturada incorreta

```
308  298          } catch (UnknownHttpResponseException e) {
309  299              data.success = false;
310  300              data.result = new Object[] { "error", e.getResponseCode(), e.getMessage() };
311  301              return data;
     302  +       } catch (Exception e2) {
     303  +           // Ask user to report all bugs which aren't timeout errors
     304  +           if (!timeoutOccured(e2)) {
     305  +               AnkiDroidApp.saveExceptionReportFile(e2, "doInBackgroundLogin");
     306  +           }
     307  +           data.success = false;
     308  +           data.result = new Object[] {"connectionError" };
     309  +           return data;
312  310          }
313  311          String hostkey = null;
```

Fix IOException bug in media sync

Currently fails to complete media sync if a user has manually a deleted a media file... It doesn't crash, but an unhandled exception dialog is shown, and the user isn't given any clues on how to solve the problem.

Categoria: Problemas no tratador

```
841  841          } catch (IOException e) {
     842  +           // Probably a file was manually deleted from the media folder
842  843              Log.e(AnkiDroidApp.TAG, "Failed to create media changes zip", e);
     844  +           // mark the media db as new so that next sync the media database is rebuilt
     845  +           mCol.getMedia().getDb().execute("update meta set dirmod=0");
843  846              throw new RuntimeException(e);
844  847          } finally {
845  848              if (cur != null) {
```

Small changes before beta

Don't swallow an exception. Might help us narrow down a media sync error.

Categoria: Exceção lançada ou capturada incorreta

```
222  222          } catch (IOException e) {
223  223              Log.e(AnkiDroidApp.TAG, "BasicHttpSyncer.sync: IOException", e);
224       -           return null;
     224  +           throw new RuntimeException(e);
225  225          } catch (JSONException e) {
226  226              throw new RuntimeException(e);
227  227          } finally {
```

Support sticky fields

I track references for almost all my cards. When I'm adding cards while reading a book the reference will stay mostly the same with exception of the page number. This feature saves a lot of typing / copy-pasting.

Categoria: Exceção não Tratada

```
225 +                try {
226 +                    flds = mEditorNote.model().getJSONArray("flds");
227 +                    if (oldNote != null) {
228 +                        for (int fldIdx = 0; fldIdx < flds.length(); fldIdx++) {
229 +                            if (flds.getJSONObject(fldIdx).getBoolean("sticky")) {
230 +                                mEditFields.get(fldIdx).setText(oldNote.getFields()[fldIdx]);
231 +                            }
232 +                        }
233 +                    }
234 +                } catch (JSONException e) {
235 +                    throw new RuntimeException();
236 +                }
```

Prepare error report when null encountered in queryColumn

Nulls have been encountered in the results of queries in the queryColumn function, and they are not supported there. This change prepares an error report that the user can send when they next start Anki - which keeps them from getting prompted multiple times. Thanks to @timrae for info on generating the report.

Categoria: Exceção não Tratada

```
198     -                // The magical line. Almost as illegible as python code ;)
199     -                results.add(type.cast(Cursor.class.getMethod(methodName, int.class).invoke(cursor, column)));
    201 +                try {
    202 +                    // The magical line. Almost as illegible as python code ;)
    203 +                    results.add(type.cast(Cursor.class.getMethod(methodName, int.class).invoke(cursor, column)));
    204 +                } catch (InvocationTargetException e) {
    205 +                    if (cursor.isNull(column)) { // null value encountered
    206 +                        nullExceptionCount++;
    207 +                        if (nullExceptionCount == 1) { // Toast and error report first time only
    208 +                            nullException = e;
```

Guard against null crash in generic column query.

Without this fix, the code has crashed when encountering nulls. This code path does not allow the returning of null values to the caller, so skipping nulls that cause a crash is a safe operation in that the caller is not expecting to see nulls, so to filter them out does not deprive the caller of data they intend to handle.

Categoria: Exceção não Tratada

```
201     -               // The magical line. Almost as illegible as python code ;)
202     -               results.add(type.cast(Cursor.class.getMethod(methodName, int.class).invoke(cursor, column)));
    202 +               try {
    203 +                   // The magical line. Almost as illegible as python code ;)
    204 +                   results.add(type.cast(Cursor.class.getMethod(methodName, int.class).invoke(cursor, column)));
    205 +               } catch (InvocationTargetException e) {
    206 +                   if (cursor.getType(column) == Cursor.FIELD_TYPE_NULL) { // null value encountered
    207 +                       Log.d(AnkiDroidApp.TAG, "AnkiDb.queryColumn (column " + column + "): Located null during excepti
    208 +                       // TODO: maybe implement reporting to notify devs of null issues, without burdening users
    209 +                       continue; // attempt to skip this null record
    210 +                   } else {
    211 +                       throw new RuntimeException(e);
    212 +                   }
    213 +               }
```

Refinements on Browser Second Column

Added more column possibilities, and added a basic check for OutOfMemory errors

Categoria: Problemas no tratador

```
775 775             } catch (OutOfMemoryError e){
776 776                 // TODO: Check if this is actually effective at dealing with the error, maybe the ArrayList has grown too l
777 777                 Log.e(AnkiDroidApp.TAG, "OutOfMemoryError rendering the Q&A for browser... probably too many cards");
    778 +               return null;
778 779             }
```

disableWriteAheadLogging() is not supported on API < 16

This commit attempts to fix startup crash on API levels less than 16 as, according to documentation, SQLiteDatabase::disableWriteAheadLogging() was introduced in JELLY_BEAN. Probably, it should be rechecked on Nook as this code was added here for that device for the first place.

Categoria: Exceção lançada ou capturada incorreta

```
316     -           db.disableWriteAheadLogging(); // call necessary on some Nook devices
317     -       } catch (Exception e) {
    316 +           //This call is required on some with custom Android versiones
    317 +           //for example on Nook, where it was backported from Jelly Bean.
    318 +           //So, we can't rely here on API version, to check for availability.
    319 +           //That's why we just catching the error if method is not available.
    320 +           db.disableWriteAheadLogging();
    321 +       } catch (Throwable e) {
318 322             Log.e(AnkiDroidApp.TAG, "AnkiDb - setDeleteJournalMode, attempting to proceed after following exception: "
```

Issue1820

Since yet another clipboard exception is crashing AnkiDroid, I've decided to just catch all exceptions coming from the clipboard rather than guess what else it might throw so we don't keep getting these.

Categoria: Exceção lançada ou capturada incorreta

```
1331      -          } catch (NullPointerException e) {
1332      -              // Workaround for https://code.google.com/p/ankidroid/issues/detail?id=1746
1333      -              // Some devices end up with an unusable clipboard. If so, we must disable it or AnkiDroid will
1334      -              // crash if it tries to use it.
     1331 +          } catch (Exception e) {
     1332 +              // https://code.google.com/p/ankidroid/issues/detail?id=1746
     1333 +              // https://code.google.com/p/ankidroid/issues/detail?id=1820
     1334 +              // Some devices or external applications make the clipboard throw exceptions. If this happens, we
     1335 +              // must disable it or AnkiDroid will crash if it tries to use it.
```

Convert steps to array of ints and floats, not strings

The StepsPreference currently returns a JSONArray of strings, and we end up storing them that way in the collection. While this works on AnkiDroid, it breaks the desktop client during review since it's expecting numerical types. This will ensure we return a JSONArray with numerical types.

Categoria: Exceção lançada ou capturada incorreta

```
 85   84              try {
 86        -              float f = Float.parseFloat(s);
 87        -              // Steps can't be 0, negative, or non-serializable (too large), so it's invalid.
 88        -              if (f <= 0 || Float.isInfinite(f)) {
 89        -                  return null;
 90        -              }
 91        -              // Use whole numbers if we can (but still allow decimals)
 92        -              int i = (int) f;
 93        -              if (i == f) {
 94        -                  sb.append(i).append(" ");
 95        -              } else {
 96        -                  sb.append(f).append(" ");
      85   +              for (int i = 0; i < ja.length(); i++) {
      86   +                  sb.append(ja.getString(i)).append(" ");
 97   87              }
 98        -          } catch (NumberFormatException e) {
 99        -              return null;
      88   +              return sb.toString().trim();
      89   +          } catch (JSONException e) {
      90   +              throw new RuntimeException(e);
100   91          }
```

Catch - 66 (Pull Requests)

35 Não são bugs de tratamento de exceção

24 Repetidas

Lint warnings with automated fixes

Some of the lint warnings have automated fixes. Here is a batch that fixes these:

Categoria: Exceção lançada ou capturada incorreta

```
271       -          } catch (IOException e) {
      269 +          } catch (IOException | InterruptedException e) {
272   270                Timber.e("repairCollection - error: " + e.getMessage());
```

Catch all simple interface exceptions

Fix for issue 2503 and potentially any other crashes that occur because of simple interface. To be honest, I think we should just remove simple interface entirely... It's not really necessary with the power of modern Android devices, and we could always add it back eventually as a plugin

Categoria: Exceção não Tratada

```
3030        -       return Html.fromHtml(text, mSimpleInterfaceImagegetter, mSimpleInterfaceTagHandler);
    3030    +       try {
    3031    +           return Html.fromHtml(text, mSimpleInterfaceImagegetter, mSimpleInterfaceTagHandler);
    3032    +       } catch (Exception e) {
    3033    +           Timber.e(e, "Error converting to simple interface");
    3034    +           return null;
    3035    +       }
```

Fix a bug where app crashes if language code three letters

No description provided.

Categoria: Exceção não Tratada

```
43          -           locale = new Locale(localeCode.substring(0, 2), localeCode.substring(3, 5));
    43      +           try {
    44      +               locale = new Locale(localeCode.substring(0, 2), localeCode.substring(3, 5));
    45      +           } catch (StringIndexOutOfBoundsException e) {
    46      +               locale = new Locale(localeCode);
    47      +           }
```

Add global try/catch around compile method in Mustache

Show a card template error on the card itself when *ANY* error occurs in Mustache compile.

Categoria: Exceção não Tratada

```
87        -         // a hand-rolled parser; whee!
88        -         Accumulator accum = new Accumulator(compiler);
89        -         char start1 = '{', start2 = '{', end1 = '}', end2 = '}';
90        -         int state = TEXT;
91        -         StringBuilder text = new StringBuilder();
92        -         int line = 1;
93        -         boolean skipNewline = false;
94        -         boolean skippedExtraBracket = false;
95        -
96        -         while (true) {
97        -             char c;
98        -             try {
99        -                 int v = source.read();
100       -                 if (v == -1) {
101       -                     break;
          87        +         try {
          88        +             // a hand-rolled parser; whee!
          89        +             Accumulator accum = new Accumulator(compiler);
          90        +             char start1 = '{', start2 = '{', end1 = '}', end2 = '}';
          91        +             int state = TEXT;
          92        +             StringBuilder text = new StringBuilder();
          93        +             int line = 1;
          94        +             boolean skipNewline = false;
          95        +             boolean skippedExtraBracket = false;
          96        +
          97        +             while (true) {
          98        +                 char c;
          99        +                 try {
          100       +                     int v = source.read();
          101       +                     if (v == -1) {
          102       +                         break;
          103       +                     }
          104       +                     c = (char) v;
          105       +                 } catch (IOException e) {
          106       +                     throw new MustacheException(e);
102       107               }
```

Catch all sync errors

This PR fixes issue 2324 and makes sure that we catch all sync errors. It also adds a finally block at the end of the sync which closes the collection, as per Anki Desktop, which should ensure that the Collection is rolled back to its previous state properly when an error occurs.

Categoria: Problemas no Tratador

```
742 +       } catch (Exception e) {
743 +           // Global error catcher.
744 +           // Try to give a human readable error, otherwise print the raw error message
745 +           Log.e(AnkiDroidApp.TAG, "doInBackgroundSync error");
746 +           e.printStackTrace();
747             data.success = false;
    -           data.result = new Object[] { "noChanges" };
    -           return data;
    -       } else {
    -           data.success = true;
    -           Object[] dc = col.getSched().deckCounts();
    -           data.result = dc[0];
    -           data.data = new Object[] { conflictResolution, col, dc[1], dc[2], mediaError };
748 +           String msg = e.getMessage();
749 +           if (msg.contains("UnknownHostException") || msg.contains("HttpHostConnectException")) {
750 +               data.result = new Object[] {"connectionError" };
751 +           } else {
752 +               AnkiDroidApp.saveExceptionReportFile(e, "doInBackgroundSync");
753 +               data.result = new Object[] {e.getLocalizedMessage()};
754 +           }
755             return data;
756 +       } finally {
757 +           col.close(false);
758         }
```

Quick fix: bugID 6548138210885632

Quick fix: throw non-file URLs into an Intent.

Categoria: Exceção não Tratada

```
454     -               if (!AnkiDroidApp.colIsOpen()) {
455     -                   closeStudyOptions();
456     -               } else if (!mFragmented) {
457     -                   ((OnStudyOptionsReloadListener) getActivity()).loadStudyOptionsFragment();
    454 +               try {
    455 +                   if (!AnkiDroidApp.colIsOpen()) {
    456 +                       closeStudyOptions();
    457 +                   } else if (!mFragmented) {
    458 +                       ((OnStudyOptionsReloadListener) getActivity()).loadStudyOptionsFragment();
    459 +                   }
    460 +               } catch (NullPointerException e) {
    461 +                   if (!isAdded()) {
    462 +                       // if the fragment is no longer attached to activity an NPE can arise (for
    463 +                   } else {
    464 +                       throw e;
    465 +                   }
```

Issue 1784: Catch template parsing errors and show useful message

Show a useful error message with some instructions when the template cannot be parsed. It ends up looking like this:

Categoria: Exceção não Tratada

```
943     -           d.put("id", Long.toString((Long) data[0]));
944     -           String qfmt = template.getString("qfmt");
945     -           String afmt = template.getString("afmt");
946     -           String html;
947     -           String format;
948     -
949     -           // runFilter mungeFields for type "q"
950     -           Models.fieldParser fparser = new Models.fieldParser(fields);
951     -           Matcher m = fClozePattern.matcher(qfmt);
952     -           format = m.replaceFirst(String.format(Locale.US, "{{cq:%d:", ((Integer) data[4]) + 1));
953     -           m = fAltClozePattern.matcher(format);
954     -           format = m.replaceFirst(String.format(Locale.US, "<%%cq:%d:",((Integer) data[4]) + 1));
955     -           html = mModels.getCmpldTemplate(format).execute(fparser);
956     -           html = (String) AnkiDroidApp.getHooks().runFilter("mungeQA", html, "q", fields, model, data, this);
957     -           d.put("q", html);
958     -           // empty cloze?
959     -           if (model.getInt("type") == Sched.MODEL_CLOZE) {
960     -               if (getModels()._availClozeOrds(model, (String) data[6], false).size() == 0) {
961     -                   d.put("q", "Please edit this note and add some cloze deletions.");
        947 +           try {
        948 +               d.put("id", Long.toString((Long) data[0]));
        949 +               String qfmt = template.getString("qfmt");
        950 +               String afmt = template.getString("afmt");
        951 +               String html;
        952 +               String format;
        953 +
        954 +               // runFilter mungeFields for type "q"
        955 +               Models.fieldParser fparser = new Models.fieldParser(fields);
        956 +               Matcher m = fClozePattern.matcher(qfmt);
        957 +               format = m.replaceFirst(String.format(Locale.US, "{{cq:%d:", ((Integer) data[4]) + 1));
        958 +               m = fAltClozePattern.matcher(format);
        959 +               format = m.replaceFirst(String.format(Locale.US, "<%%cq:%d:",((Integer) data[4]) + 1));
        960 +               html = mModels.getCmpldTemplate(format).execute(fparser);
        961 +               html = (String) AnkiDroidApp.getHooks().runFilter("mungeQA", html, "q", fields, model, data, this);
        962 +               d.put("q", html);
        963 +               // empty cloze?
        964 +               if (model.getInt("type") == Sched.MODEL_CLOZE) {
        965 +                   if (getModels()._availClozeOrds(model, (String) data[6], false).size() == 0) {
        966 +                       d.put("q", "Please edit this note and add some cloze deletions.");
        967 +                   }
962     968                 }
```