Aluno: Luis Gomes
Matrícula: 428223

Refatoração do código para entrega 2

1
classe PNConfiguration.java

método PNConfiguration linha 39

bad smell detectado : Long Identifier [in method PNConfiguration]: The length of the field FILE_MESSAGE_PUBLISH_RETRY_LIMIT is 32.
long identifier = 1
antes da refatoração:

```
        private static final int FILE_MESSAGE_PUBLISH_RETRY_LIMIT = 5;
```

após refatoração:

```
        private static final int MESSAGE_RETRY = 5;
```

bad smell removido

2

classe UnwrapSingleField.java

método deserialize linha 11

Erro detectado : Long Statement [in method deserialize]: The length of the statement "throw new IllegalStateException("Couldn't unwrap field for object containing more than 1 field. Actual number of fields: "; + jsonObject.keySet().size());"
long statemet = 1
antes da refatoração:

```
   public T deserialize(final JsonElement json, final Type typeOfT, final JsonDeserializationContext context) throws JsonParseException {
       final JsonObject jsonObject = json.getAsJsonObject();
       if (jsonObject.keySet().size() != 1) {
           throw new IllegalStateException("Couldn't unwrap field for object containing more than 1 field. Actual number of fields: "; + jsonObject.keySet().size());
       }
       final String key = jsonObject.keySet().toArray(new String[]{})[0];
       final JsonElement element = jsonObject.get(key);
       return context.deserialize(element, typeOfT);
   }
}
```

após refatoração:

```
   public T deserialize(final JsonElement json, final Type typeOfT, final
JsonDeserializationContext context) throws JsonParseException {
       final JsonObject jsonObject = json.getAsJsonObject();
       if (jsonObject.keySet().size() != 1) {
          string messge= "Couldn't unwrap field for object containing more than 1 field. Actual
number of fields: ";
          throw new IllegalStateException(messge + jsonObject.keySet().size());
       }
       final String key = jsonObject.keySet().toArray(new String[]{})[0];
       final JsonElement element = jsonObject.get(key);
       return context.deserialize(element, typeOfT);
   }
}
```

3

classe SquashBacktrace.java

método populateNestedExceptions linha 80

Erro detectado : Complex Conditional [in method populateNestedExceptions]: The conditional expression error == null || error.getCause() == null || error.getCause() == error is complex.

complex conditional = 1
antes da refatoração:

```
  public static void populateNestedExceptions(List<NestedException> nestedExceptions,
     Throwable error) {
   // Only keep processing if the "cause" exception is set and != the "parent" exception.
   if (error == null || error.getCause() == null || error.getCause() == error) {
    return;
   }
   final Throwable cause = error.getCause();
   NestedException doc =
       new NestedException(cause.getClass().getName(), cause.getMessage(),
getBacktraces(cause),
           getIvars(cause));
   nestedExceptions.add(doc);
   // Exceptions all the way down!
   populateNestedExceptions(nestedExceptions, cause);
  }
```

após refatoração:

```
  public static void populateNestedExceptions(List<NestedException> nestedExceptions,
      Throwable error) {
    // Only keep processing if the "cause" exception is set and != the "parent" exception.
    if (error == null  ) {
      return;
    }
    if(error.getCause() == null ){return;}
    if(error.getCause() == error){return;}

    final Throwable cause = error.getCause();
    NestedException doc =
        new NestedException(cause.getClass().getName(), cause.getMessage(),
getBacktraces(cause),
            getIvars(cause));
    nestedExceptions.add(doc);
    // Exceptions all the way down!
    populateNestedExceptions(nestedExceptions, cause);
  }
```

4

classe SquashBackTrace.java

método getStacktraceArray linha 42

Erro detectado : Long Statement [in method getStacktraceArray]: The length of the statement "StackElement elementList=new StackElement(element.getClassName()`element.getFileName()`element.getLineNumber()`element.getMethodName());" is 136.
long statemet = 2

antes da refatoração:

```
private static List<StackElement> getStacktraceArray(Throwable error) {
    List<StackElement> stackElems = new ArrayList<StackElement>();
    for (StackTraceElement element : error.getStackTrace()) {
      StackElement elementList =
          new StackElement(element.getClassName(), element.getFileName(),
element.getLineNumber(),
              element.getMethodName());
      stackElems.add(elementList);
    }
    return stackElems;
  }
```

após refatoração:

```
   private static List<StackElement> getStacktraceArray(Throwable error) {
   List<StackElement> stackElems = new ArrayList<StackElement>();
   for (StackTraceElement element : error.getStackTrace()) {
      String className = element.getClassName();
            String fileName = element.getFileName();
                  int lineNumber = element.getLineNumber();
                        String methodName = element.getMethodName();
     StackElement elementList =
        new StackElement(className, fileName, lineNumber,
           methodName);
     stackElems.add(elementList);
   }
   return stackElems;
 }
```

5

classe SquashBacktrace.java

método populateNestedExceptions linha 80

Erro detectado : Long Statement [in method populateNestedExceptions]: The length of the statement "NestedException doc=new NestedException(cause.getClass().getName()`cause.getMessage()`getBacktraces(cause)`getIvars(cause));" is 124.
long statement = 3

antes da refatoração:

```
   public static void populateNestedExceptions(List<NestedException> nestedExceptions,
     Throwable error) {
   // Only keep processing if the "cause" exception is set and != the "parent" exception.
   if (error == null  ) {
     return;
   }
   if(error.getCause() == null ){return;}
   if(error.getCause() == error){return;}

   final Throwable cause = error.getCause();

   NestedException doc =
      new NestedException(cause.getClass().getName(),cause.getMessage(),
getBacktraces(cause),
          getIvars(cause));
```

```
    nestedExceptions.add(doc);
    // Exceptions all the way down!
    populateNestedExceptions(nestedExceptions, cause);
  }
```

após refatoração:

```
     public static void populateNestedExceptions(List<NestedException> nestedExceptions,
      Throwable error) {
    // Only keep processing if the "cause" exception is set and != the "parent" exception.
    if (error == null  ) {
      return;
    }
    if(error.getCause() == null ){return;}
    if(error.getCause() == error){return;}

    final Throwable cause = error.getCause();
    String name = cause.getClass().getName();
    String message = cause.getMessage();
    NestedException doc =
        new NestedException(name,message, getBacktraces(cause),
            getIvars(cause));
    nestedExceptions.add(doc);
    // Exceptions all the way down!
    populateNestedExceptions(nestedExceptions, cause);
  }
```

6

classe SquashBacktrace.java

método getBacktraces linha 31

Erro detectado : Long Statement [in method getBacktraces]: The length of the statement "final SquashException currentThread=new SquashException(Thread.currentThread().getName()`true`getStacktraceArray(error));" is 121.
long statement = 4
antes da refatoração:

```
      public static List<SquashException> getBacktraces(Throwable error) {
    if (error == null) {
      return null;
    }
    final List<SquashException> threadList = new ArrayList<SquashException>();
    final SquashException currentThread =
```

```
      new SquashException(Thread.currentThread().getName(), true,
getStacktraceArray(error));
   threadList.add(currentThread);
   return threadList;
 }
```

após refatoração:

```
        public static List<SquashException> getBacktraces(Throwable error) {
   if (error == null) {
     return null;
   }
   final List<SquashException> threadList = new ArrayList<SquashException>();
         String name = Thread.currentThread().getName();
         final SquashException currentThread =
      new SquashException(name, true, getStacktraceArray(error));
   threadList.add(currentThread);
   return threadList;
 }
```

7

classe Pubnub.java

método destroy linha 555

Erro detectado :Empty catch clause [in method destroy]: The method has an empty catch block.
empty catch clause : 1
antes da refatoração:

```
     public void destroy() {
       try {
          subscriptionManager.destroy(false);
          retrofitManager.destroy(false);
       } catch (Exception error) {
          //
       }
   }
```

após refatoração:

```
       public void destroy() {
       try {
          subscriptionManager.destroy(false);
          retrofitManager.destroy(false);
```

```
        } catch (Exception error) {
            return ;
        }
    }
```

8

classe Pubnub.java

método forceDestroy linha 567

Erro detectado :Empty catch clause [in method destroy]: The method has an empty catch block.
empty catch clause : 2
antes da refatoração:

```
        public void forceDestroy() {
        try {
            subscriptionManager.destroy(true);
            retrofitManager.destroy(true);
            telemetryManager.stopCleanUpTimer();
        } catch (Exception error) {
            //
        }
    }
```

após refatoração:

```
        public void forceDestroy() {
        try {
            subscriptionManager.destroy(true);
            retrofitManager.destroy(true);
            telemetryManager.stopCleanUpTimer();
        } catch (Exception error) {
            return;
        }
    }
```

9

classe PubNubExceptionTest.java

método testPubnubError linha 38

Erro detectado : Long Statement [in method testPubnubError]: The length of the statement "stubFor(get(urlPathEqualTo("/publish/myPublishKey/mySubscribeKey/0/coolChannel/0/%22

hi%22")).willReturn(aResponse().withStatus(404).withBody("[1`\"Sent\"`\"1459811159531800
3\"]")));" is 181.
long statement: 5

antes da refatoração:
```
  public void testPubnubError() {]

stubFor(get(urlPathEqualTo("/publish/myPublishKey/mySubscribeKey/0/coolChannel/0/%22
hi%22"))

.willReturn(aResponse().withStatus(404).withBody("[1,\"Sent\",\"14598111595318003\"]")));

      int statusCode = -1;

      try {
         instance.channel("coolChannel").message("hi").sync();
      } catch (PubNubException error) {
         statusCode = error.getStatusCode();
      }

      assertEquals(404, statusCode);
   }
```

após refatoração:
```
   public void testPubnubError() {]
      String url = "/publish/myPublishKey/mySubscribeKey/0/coolChannel/0/%22hi%22";
      String sent = "[1,\"Sent\",\"14598111595318003\"]";
      stubFor(get(urlPathEqualTo(url))
            .willReturn(aResponse().withStatus(404).withBody(sent)));

      int statusCode = -1;

      try {
         instance.channel("coolChannel").message("hi").sync();
      } catch (PubNubException error) {
         statusCode = error.getStatusCode();
      }

      assertEquals(404, statusCode);
   }
```

10

classe PubNubErrorBuilder.java

método createCryptoError linha 717

Erro detectado : long statement:6

antes da refatoração:

```
   public static PubNubError createCryptoError(int code, String message) {


       return PubNubError.builder()
               .errorCode(PNERR_CRYPTO_ERROR)
               .errorCodeExtended(code)
               .message("Error while encrypting/decrypting message. Please contact support
with error details. - ".concat(message))
               .build();
   }
```

após refatoração:

```
  public static PubNubError createCryptoError(int code, String message) {

       String err = "Error while encrypting/decrypting message. Please contact support with
error details. - ";
       return PubNubError.builder()
               .errorCode(PNERR_CRYPTO_ERROR)
               .errorCodeExtended(code)
               .message(err.concat(message))
               .build();
   }
```

11

classe Grant.java

método validateParams linha 81

Erro detectado : Complex Conditional [in method validateParams]: The conditional expression (!channels.isEmpty() || !channelGroups.isEmpty()) && !uuids.isEmpty() is complex.
complex conditional: 2

antes da refatoração:

```
     protected void validateParams() throws PubNubException {
       if (this.getPubnub().getConfiguration().getSecretKey() == null || this.getPubnub()
               .getConfiguration()
               .getSecretKey()
               .isEmpty()) {
```

```
            throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SECRET_KEY_
MISSING).build();
        }
        if (this.getPubnub().getConfiguration().getSubscribeKey() == null || this.getPubnub()
                .getConfiguration()
                .getSubscribeKey()
                .isEmpty()) {
            throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SUBSCRIBE_KE
Y_MISSING).build();
        }
        if (this.getPubnub().getConfiguration().getPublishKey() == null || this.getPubnub()
                .getConfiguration()
                .getPublishKey()
                .isEmpty()) {
            throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_PUBLISH_KEY_
MISSING).build();
        }
        if ((!channels.isEmpty() || !channelGroups.isEmpty()) && !uuids.isEmpty()) {
            throw PubNubException.builder()
                    .pubnubError(PubNubErrorBuilder.PNERROBJ_INVALID_ARGUMENTS)
                    .errormsg("Grants for channels or channelGroups can't be changed together
with grants for UUIDs")
                    .build();
        }
        if (!uuids.isEmpty() && authKeys.isEmpty()) {
            throw PubNubException.builder()
                    .pubnubError(PubNubErrorBuilder.PNERROBJ_INVALID_ARGUMENTS)
                    .errormsg("UUIDs grant management require providing non empty authKeys")
                    .build();
        }
    }


após refatoração:
    protected void validateParams() throws PubNubException {
        if (this.getPubnub().getConfiguration().getSecretKey() == null || this.getPubnub()
                .getConfiguration()
                .getSecretKey()
                .isEmpty()) {
            throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SECRET_KEY_
MISSING).build();
        }
        if (this.getPubnub().getConfiguration().getSubscribeKey() == null || this.getPubnub()
                .getConfiguration()
```

```
            .getSubscribeKey()
            .isEmpty()) {
        throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SUBSCRIBE_KE
Y_MISSING).build();
    }
    if (this.getPubnub().getConfiguration().getPublishKey() == null || this.getPubnub()
            .getConfiguration()
            .getPublishKey()
            .isEmpty()) {
        throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_PUBLISH_KEY_
MISSING).build();
    }
    if ((!channels.isEmpty() && !uuids.isEmpty()) {
        something();
    }
    if ((!channelGroups.isEmpty()) && !uuids.isEmpty()) {
        something();
    }
    if (!uuids.isEmpty() && authKeys.isEmpty()) {
        throw PubNubException.builder()
                .pubnubError(PubNubErrorBuilder.PNERROBJ_INVALID_ARGUMENTS)
                .errormsg("UUIDs grant management require providing non empty authKeys")
                .build();
    }
  }
  void something(){
    throw PubNubException.builder()
            .pubnubError(PubNubErrorBuilder.PNERROBJ_INVALID_ARGUMENTS)
            .errormsg("Grants for channels or channelGroups can't be changed together with
grants for UUIDs")
            .build();
  }
```

12

classe Grant.java

método createResponse() linha 196

Erro detectado : Complex Method [in method createResponse]: Cyclomatic complexity of the method is 11
complex method: 1

antes da refatoração:

```
  protected PNAccessManagerGrantResult
createResponse(Response<Envelope<AccessManagerGrantPayload>> input) throws
        PubNubException {
    MapperManager mapperManager = getPubnub().getMapper();
    PNAccessManagerGrantResult.PNAccessManagerGrantResultBuilder
pnAccessManagerGrantResult =
            PNAccessManagerGrantResult.builder();

    if (input.body() == null || input.body().getPayload() == null) {
        throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_PARSING_ERR
OR).build();
    }

    AccessManagerGrantPayload data = input.body().getPayload();
    Map<String, Map<String, PNAccessManagerKeyData>> constructedChannels = new
HashMap<>();
    Map<String, Map<String, PNAccessManagerKeyData>> constructedGroups = new
HashMap<>();

    // we have a case of a singular channel.
    if (data.getChannel() != null) {
        constructedChannels.put(data.getChannel(), data.getAuthKeys());
    }

    if (channelGroups != null) {
        if (channelGroups.size() == 1) {

constructedGroups.put(mapperManager.elementToString(data.getChannelGroups()),
data.getAuthKeys());
        } else if (channelGroups.size() > 1) {
            Iterator<Map.Entry<String, JsonElement>> it =
mapperManager.getObjectIterator(data.getChannelGroups());
            while (it.hasNext()) {
                Map.Entry<String, JsonElement> channelGroup = it.next();
                constructedGroups.put(channelGroup.getKey(),
createKeyMap(channelGroup.getValue()));
            }
        }
    }

    if (data.getChannels() != null) {
        for (String fetchedChannel : data.getChannels().keySet()) {
            constructedChannels.put(fetchedChannel,
data.getChannels().get(fetchedChannel).getAuthKeys());
        }
    }
```

```
        Map<String, Map<String, PNAccessManagerKeyData>> constructedUuids = new
HashMap<>();

        if (data.getUuids() != null) {
            for (String fetchedUuid : data.getUuids().keySet()) {
                constructedUuids.put(fetchedUuid,
data.getUuids().get(fetchedUuid).getAuthKeys());
            }
        }

        return pnAccessManagerGrantResult
                .subscribeKey(data.getSubscribeKey())
                .level(data.getLevel())
                .ttl(data.getTtl())
                .channels(constructedChannels)
                .channelGroups(constructedGroups)
                .uuids(constructedUuids)
                .build();
    }
```

após refatoração:

```
    @Override
    protected PNAccessManagerGrantResult
createResponse(Response<Envelope<AccessManagerGrantPayload>> input) throws
            PubNubException {
        MapperManager mapperManager = getPubnub().getMapper();
        PNAccessManagerGrantResult.PNAccessManagerGrantResultBuilder
pnAccessManagerGrantResult =
                PNAccessManagerGrantResult.builder();

        if (input.body() == null || input.body().getPayload() == null) {
            throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_PARSING_ERR
OR).build();
        }

        AccessManagerGrantPayload data = input.body().getPayload();
        Map<String, Map<String, PNAccessManagerKeyData>> constructedGroups = new
HashMap<>();
        Map<String, Map<String, PNAccessManagerKeyData>> constructedChannels = new
HashMap<>();
        Map<String, Map<String, PNAccessManagerKeyData>> constructedUuids = new
HashMap<>();


        // we have a case of a singular channel.
```

```
        actoin(data,constructedChannels);
        actoin1(data,constructedGroups);
        actoin2(data,constructedChannels);
        actoin3(data,constructedUuids);









        return pnAccessManagerGrantResult
                .subscribeKey(data.getSubscribeKey())
                .level(data.getLevel())
                .ttl(data.getTtl())
                .channels(constructedChannels)
                .channelGroups(constructedGroups)
                .uuids(constructedUuids)
                .build();
    }
        void actoin(AccessManagerGrantPayload data,Map<String, Map<String,
PNAccessManagerKeyData>> constructedChannels){
        //AccessManagerGrantPayload data = input.body().getPayload();
        //Map<String, Map<String, PNAccessManagerKeyData>> constructedChannels = new
HashMap<>();
        if (data.getChannel() != null) {
            constructedChannels.put(data.getChannel(), data.getAuthKeys());
        }
    }
     void actoin1(AccessManagerGrantPayload data, Map<String, Map<String,
PNAccessManagerKeyData>> constructedGroups){
        //AccessManagerGrantPayload data = input.body().getPayload();
        //Map<String, Map<String, PNAccessManagerKeyData>> constructedGroups = new
HashMap<>();
        if (channelGroups != null) {
            if (channelGroups.size() == 1) {

constructedGroups.put(mapperManager.elementToString(data.getChannelGroups()),
data.getAuthKeys());
            } else if (channelGroups.size() > 1) {
                Iterator<Map.Entry<String, JsonElement>> it =
mapperManager.getObjectIterator(data.getChannelGroups());
                while (it.hasNext()) {
                    Map.Entry<String, JsonElement> channelGroup = it.next();
                    constructedGroups.put(channelGroup.getKey(),
createKeyMap(channelGroup.getValue()));
                }
            }
        }
```

```
    }
    void actoin2(AccessManagerGrantPayload data,  Map<String, Map<String,
PNAccessManagerKeyData>> constructedChannels){
        //AccessManagerGrantPayload data = input.body().getPayload();
        //Map<String, Map<String, PNAccessManagerKeyData>> constructedChannels = new
HashMap<>();
        if (data.getChannels() != null) {
            for (String fetchedChannel : data.getChannels().keySet()) {
                constructedChannels.put(fetchedChannel,
data.getChannels().get(fetchedChannel).getAuthKeys());
            }
        }
    }
    void actoin3( AccessManagerGrantPayload data, Map<String, Map<String,
PNAccessManagerKeyData>> constructedUuids){
        //AccessManagerGrantPayload data = input.body().getPayload();
        //Map<String, Map<String, PNAccessManagerKeyData>> constructedUuids = new
HashMap<>();

        if (data.getUuids() != null) {
            for (String fetchedUuid : data.getUuids().keySet()) {
                constructedUuids.put(fetchedUuid,
data.getUuids().get(fetchedUuid).getAuthKeys());
            }
        }
    }
```

13

classe GrantToken.java

método validateParams linha 71

Erro detectado : Complex Conditional [in method validateParams]: The conditional expression isNullOrEmpty(channels) && isNullOrEmpty(channelGroups) && isNullOrEmpty(uuids) is complex.
complex conditional: 3
antes da refatoração:

```
  protected void validateParams() throws PubNubException {
        if (this.getPubnub().getConfiguration().getSecretKey() == null || this.getPubnub()
                .getConfiguration()
                .getSecretKey()
                .isEmpty()) {
            throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SECRET_KEY_
MISSING).build();
        }
```

```
        if (this.getPubnub().getConfiguration().getSubscribeKey() == null || this.getPubnub()
                .getConfiguration()
                .getSubscribeKey()
                .isEmpty()) {
            throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SUBSCRIBE_KE
Y_MISSING).build();
        }
        if (isNullOrEmpty(channels)
                && isNullOrEmpty(channelGroups)
                && isNullOrEmpty(uuids)) {
            throw PubNubException.builder()
                    .pubnubError(PubNubErrorBuilder.PNERROBJ_RESOURCES_MISSING)
                    .build();
        }
        if (this.ttl == null) {
            throw PubNubException.builder()
                    .pubnubError(PubNubErrorBuilder.PNERROBJ_TTL_MISSING)
                    .build();
        }
    }
```

após refatoração:

```
        protected void validateParams() throws PubNubException {
        if (this.getPubnub().getConfiguration().getSecretKey() == null || this.getPubnub()
                .getConfiguration()
                .getSecretKey()
                .isEmpty()) {
            throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SECRET_KEY_
MISSING).build();
        }
        if (this.getPubnub().getConfiguration().getSubscribeKey() == null || this.getPubnub()
                .getConfiguration()
                .getSubscribeKey()
                .isEmpty()) {
            throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SUBSCRIBE_KE
Y_MISSING).build();
        }
        if (isNullOrEmpty(channels) {
            throw PubNubException.builder()
                    .pubnubError(PubNubErrorBuilder.PNERROBJ_RESOURCES_MISSING)
                    .build();
        }
        if (isNullOrEmpty(channelGroups)) {
```

```
            throw PubNubException.builder()
                    .pubnubError(PubNubErrorBuilder.PNERROBJ_RESOURCES_MISSING)
                    .build();
        }
        if (isNullOrEmpty(uuids)) {
            throw PubNubException.builder()
                    .pubnubError(PubNubErrorBuilder.PNERROBJ_RESOURCES_MISSING)
                    .build();
        }
        if (this.ttl == null) {
            throw PubNubException.builder()
                    .pubnubError(PubNubErrorBuilder.PNERROBJ_TTL_MISSING)
                    .build();
        }
    }
```

14

classe RevokeToken.java

método método validateParams linha 47

Erro detectado : Long Statement [in method validateParams]: The length of the statement "if (this.getPubnub().getConfiguration().getSecretKey() == null || this.getPubnub().getConfiguration().getSecretKey().isEmpty()) {" is 129.
long statement: 7
antes da refatoração:

```
  protected void validateParams() throws PubNubException {
        if (this.getPubnub().getConfiguration().getSecretKey() == null || this.getPubnub()
                .getConfiguration()
                .getSecretKey()
                .isEmpty()) {
            throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SECRET_KEY_
MISSING).build();
        }
        if (this.getPubnub().getConfiguration().getSubscribeKey() == null || this.getPubnub()
                .getConfiguration()
                .getSubscribeKey()
                .isEmpty()) {
            throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SUBSCRIBE_KE
Y_MISSING).build();
        }
        if (this.token == null) {
            throw PubNubException.builder()
                    .pubnubError(PubNubErrorBuilder.PNERROBJ_TOKEN_MISSING)
```

```
                .build();
        }
    }
```

após refatoração:

```
        protected void validateParams() throws PubNubException {
        if (this.getPubnub().getConfiguration().getSecretKey() == null ){
            throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SECRET_KEY_
MISSING).build();
        }
        if(this.getPubnub()
                .getConfiguration()
                .getSecretKey()
                .isEmpty()){
            throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SECRET_KEY_
MISSING).build();
        }

            //a método acima
        if (this.getPubnub().getConfiguration().getSubscribeKey() == null ) {
            throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SUBSCRIBE_KE
Y_MISSING).build();
        }
        if( this.getPubnub()
                .getConfiguration()
                .getSubscribeKey()
                .isEmpty()){
            throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SUBSCRIBE_KE
Y_MISSING).build();
        }
        if (this.token == null) {
            throw PubNubException.builder()
                    .pubnubError(PubNubErrorBuilder.PNERROBJ_TOKEN_MISSING)
                    .build();
        }
    }
```

15

classe RevokeToken.java

método validateParams linha 47

Erro detectado : Long Statement [in method validateParams]: The length of the statement "if (this.getPubnub().getConfiguration().getSubscribeKey() == null || this.getPubnub().getConfiguration().getSubscribeKey().isEmpty()) {" is 135.
long statement: 8
antes da refatoração:

```
   protected void validateParams() throws PubNubException {
      if (this.getPubnub().getConfiguration().getSecretKey() == null || this.getPubnub()
            .getConfiguration()
            .getSecretKey()
            .isEmpty()) {
         throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SECRET_KEY_
MISSING).build();
      }
      if (this.getPubnub().getConfiguration().getSubscribeKey() == null || this.getPubnub()
            .getConfiguration()
            .getSubscribeKey()
            .isEmpty()) {
         throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SUBSCRIBE_KE
Y_MISSING).build();
      }
      if (this.token == null) {
         throw PubNubException.builder()
               .pubnubError(PubNubErrorBuilder.PNERROBJ_TOKEN_MISSING)
               .build();
      }
   }
```

após refatoração:

```
      protected void validateParams() throws PubNubException {
      if (this.getPubnub().getConfiguration().getSecretKey() == null ){
         throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SECRET_KEY_
MISSING).build();
      }
      if(this.getPubnub()
            .getConfiguration()
            .getSecretKey()
            .isEmpty()){
         throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SECRET_KEY_
MISSING).build();
      }
         //a método abaixo
```

```
        if (this.getPubnub().getConfiguration().getSubscribeKey() == null ) {
            throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SUBSCRIBE_KE
Y_MISSING).build();
        }
        if( this.getPubnub()
                .getConfiguration()
                .getSubscribeKey()
                .isEmpty()){
            throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SUBSCRIBE_KE
Y_MISSING).build();
        }
        if (this.token == null) {
            throw PubNubException.builder()
                    .pubnubError(PubNubErrorBuilder.PNERROBJ_TOKEN_MISSING)
                    .build();
        }
    }
```

16

classe RevokeToken.java

método doWork linha 74

Erro detectado : Long Statement [in method doWork]: The length of the statement "return getRetrofit().getAccessManagerService().revokeToken(getPubnub().getConfiguration().getSubscribeKey()`repairEncoding(token)`baseParams);" is 142.
long statement: 9
antes da refatoração:

```
    protected Call<RevokeTokenResponse> doWork(Map<String, String> baseParams)
throws PubNubException {
        return getRetrofit()
                .getAccessManagerService()
                .revokeToken( getPubnub().getConfiguration().getSubscribeKey(),
repairEncoding(token), baseParams);
    }
```

após refatoração:

```
     protected Call<RevokeTokenResponse> doWork(Map<String, String> baseParams)
throws PubNubException {
        PubNub aux = getPubnub().getConfiguration().getSubscribeKey();
        return getRetrofit()
                .getAccessManagerService()
                .revokeToken(aux, repairEncoding(token), baseParams);
```

```
  }
```

17

classe ListFiles.java

método validateParams linha 60

Erro detectado : Complex Conditional [in method validateParams]: The conditional expression next != null && (next.getHash() == null || next.getHash().isEmpty()) is complex.
complex conditional: 4
antes da refatoração:

```
    protected void validateParams() throws PubNubException {
      if (this.getPubnub().getConfiguration().getSubscribeKey() == null
            || this.getPubnub().getConfiguration().getSubscribeKey().isEmpty()) {
         throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SUBSCRIBE_KE
Y_MISSING).build();
      }

      if (channel == null || channel.isEmpty()) {
         throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_CHANNEL_MIS
SING).build();
      }

      if (limit != null && !(MIN_LIMIT <= limit && limit <= MAX_LIMIT)) {
         throw PubNubException.builder()
               .pubnubError(PubNubErrorBuilder.PNERROBJ_INVALID_ARGUMENTS)
               .errormsg("Limit should be in range from 1 to 100 (both inclusive)")
               .build();
      }

      if (next != null && (next.getHash() == null || next.getHash().isEmpty())) {
         throw PubNubException.builder()
               .pubnubError(PubNubErrorBuilder.PNERROBJ_INVALID_ARGUMENTS)
               .errormsg("Next should not be an empty string")
               .build();
      }
   }
```

após refatoração:

```
      protected void validateParams() throws PubNubException {
      if (this.getPubnub().getConfiguration().getSubscribeKey() == null
            || this.getPubnub().getConfiguration().getSubscribeKey().isEmpty()) {
```

```
            throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SUBSCRIBE_KE
Y_MISSING).build();
        }

        if (channel == null || channel.isEmpty()) {
            throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_CHANNEL_MIS
SING).build();
        }

        if (limit != null && !(MIN_LIMIT <= limit && limit <= MAX_LIMIT)) {
            throw PubNubException.builder()
                    .pubnubError(PubNubErrorBuilder.PNERROBJ_INVALID_ARGUMENTS)
                    .errormsg("Limit should be in range from 1 to 100 (both inclusive)")
                    .build();
        }

        if (next != null && next.getHash() == null) {
            toDo();
        }
        if (next != null && next.getHash().isEmpty()) {
            toDo();
        }
    }
    void toDO(){
        String msg = "";
        throw PubNubException.builder()
                .pubnubError(PubNubErrorBuilder.PNERROBJ_INVALID_ARGUMENTS)
                .errormsg("Next should not be an empty string")
                .build();
    }
```

18

classe ListFiles.java

método validadeParams linha 60

Erro detectado : Long Statement [in method validateParams]: The length of the statement "if (this.getPubnub().getConfiguration().getSubscribeKey() == null || this.getPubnub().getConfiguration().getSubscribeKey().isEmpty()) {" is 135.
long statement: 10
antes da refatoração:

```
    protected void validateParams() throws PubNubException {
        if (this.getPubnub().getConfiguration().getSubscribeKey() == null
                || this.getPubnub().getConfiguration().getSubscribeKey().isEmpty()) {
```

```
            throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SUBSCRIBE_KE
Y_MISSING).build();
        }

        if (channel == null || channel.isEmpty()) {
            throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_CHANNEL_MIS
SING).build();
        }

        if (limit != null && !(MIN_LIMIT <= limit && limit <= MAX_LIMIT)) {
            throw PubNubException.builder()
                    .pubnubError(PubNubErrorBuilder.PNERROBJ_INVALID_ARGUMENTS)
                    .errormsg("Limit should be in range from 1 to 100 (both inclusive)")
                    .build();
        }

        if (next != null && next.getHash() == null) {
            toDo();
        }
        if (next != null && next.getHash().isEmpty()) {
            toDo();
        }
    }
```

após refatoração:

```
        protected void validateParams() throws PubNubException {
        if (this.getPubnub().getConfiguration().getSubscribeKey() == null
                || this.getPubnub().getConfiguration().getSubscribeKey().isEmpty()) {
            throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SUBSCRIBE_KE
Y_MISSING).build();
        }

        if (channel == null || channel.isEmpty()) {
            throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_CHANNEL_MIS
SING).build();
        }

        if (limit != null && !(MIN_LIMIT <= limit && limit <= MAX_LIMIT)) {
            String str = "Limit should be in range from 1 to 100 (both inclusive)";
            throw PubNubException.builder()
                    .pubnubError(PubNubErrorBuilder.PNERROBJ_INVALID_ARGUMENTS)
                    .errormsg(str)
                    .build();
```

```
        }

        if (next != null && next.getHash() == null) {
            toDo();
        }
        if (next != null && next.getHash().isEmpty()) {
            toDo();
        }
    }
```

19

classe History.java

método doWork Linha 80

Erro detectado : Complex Conditional [in method doWork]: The conditional expression count != null && count > 0 && count <= MAX_COUNT is complex.
  complex conditional: 6

```
 protected Call<JsonElement> doWork(Map<String, String> params) {

        if (reverse != null) {
            params.put("reverse", String.valueOf(reverse));
        }

        if (includeTimetoken != null) {
            params.put("include_token", String.valueOf(includeTimetoken));
        }

        if (includeMeta) {
            params.put("include_meta", String.valueOf(includeMeta));
        }

        if (count != null && count > 0 && count <= MAX_COUNT) {
            params.put("count", String.valueOf(count));
        } else {
            params.put("count", "100");
        }

        if (start != null) {
            params.put("start", Long.toString(start).toLowerCase());
        }
        if (end != null) {
            params.put("end", Long.toString(end).toLowerCase());
        }
```

```
        return
this.getRetrofit().getHistoryService().fetchHistory(this.getPubnub().getConfiguration()
                .getSubscribeKey(), channel, params);
    }
```

após refatoração:

20

classe FetchMessages.java

método validateParams linha 80

Erro detectado : Complex Conditional [in method validateParams]: The conditional expression maximumPerChannel == null || maximumPerChannel < 1 || maximumPerChannel > MAX_MESSAGES_WITH_ACTIONS is complex.
complex conditional: 7
antes da refatoração:

```
        protected void validateParams() throws PubNubException {
        if (this.getPubnub().getConfiguration().getSubscribeKey() == null
                || this.getPubnub().getConfiguration().getSubscribeKey().isEmpty()) {
            throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SUBSCRIBE_KEY_MISSING).build();
        }

        if (channels == null || channels.size() == 0) {
            throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_CHANNEL_MISSING).build();
        }

        if (includeMeta == null) {
            includeMeta = false;
        }

        if (includeMessageActions == null) {
            includeMessageActions = false;
        }

        if (!includeMessageActions) {
            if (channels.size() == 1) {
                if (maximumPerChannel == null || maximumPerChannel < 1) {
                    maximumPerChannel = SINGLE_CHANNEL_DEFAULT_MESSAGES;
```

```
                log.info("maximumPerChannel param defaulting to " + maximumPerChannel);
            } else if (maximumPerChannel > SINGLE_CHANNEL_MAX_MESSAGES) {
                maximumPerChannel = SINGLE_CHANNEL_MAX_MESSAGES;
                log.info("maximumPerChannel param defaulting to " + maximumPerChannel);
            }
        } else {
            if (maximumPerChannel == null || maximumPerChannel < 1) {
                maximumPerChannel = MULTIPLE_CHANNEL_DEFAULT_MESSAGES;
                log.info("maximumPerChannel param defaulting to " + maximumPerChannel);
            } else if (maximumPerChannel > MULTIPLE_CHANNEL_MAX_MESSAGES) {
                maximumPerChannel = MULTIPLE_CHANNEL_MAX_MESSAGES;
                log.info("maximumPerChannel param defaulting to " + maximumPerChannel);
            }
        }
    } else {

    if (maximumPerChannel == null || maximumPerChannel < 1 || maximumPerChannel >
MAX_MESSAGES_WITH_ACTIONS){
                maximumPerChannel = DEFAULT_MESSAGES_WITH_ACTIONS;
        log.info("maximumPerChannel param defaulting to " + maximumPerChannel);
    }

  }


após refatoração:
    protected void validateParams() throws PubNubException {
    if (this.getPubnub().getConfiguration().getSubscribeKey() == null
            || this.getPubnub().getConfiguration().getSubscribeKey().isEmpty()) {
        throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SUBSCRIBE_KE
Y_MISSING).build();
    }

    if (channels == null || channels.size() == 0) {
        throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_CHANNEL_MIS
SING).build();
    }

    if (includeMeta == null) {
        includeMeta = false;
    }

    if (includeMessageActions == null) {
        includeMessageActions = false;
    }
```

```
        if (!includeMessageActions) {
            if (channels.size() == 1) {
                if (maximumPerChannel == null || maximumPerChannel < 1) {
                    maximumPerChannel = SINGLE_CHANNEL_DEFAULT_MESSAGES;
                    log.info("maximumPerChannel param defaulting to " + maximumPerChannel);
                } else if (maximumPerChannel > SINGLE_CHANNEL_MAX_MESSAGES) {
                    maximumPerChannel = SINGLE_CHANNEL_MAX_MESSAGES;
                    log.info("maximumPerChannel param defaulting to " + maximumPerChannel);
                }
            } else {
                if (maximumPerChannel == null || maximumPerChannel < 1) {
                    maximumPerChannel = MULTIPLE_CHANNEL_DEFAULT_MESSAGES;
                    log.info("maximumPerChannel param defaulting to " + maximumPerChannel);
                } else if (maximumPerChannel > MULTIPLE_CHANNEL_MAX_MESSAGES) {
                    maximumPerChannel = MULTIPLE_CHANNEL_MAX_MESSAGES;
                    log.info("maximumPerChannel param defaulting to " + maximumPerChannel);
                }
            }
        } else {
            doIt(maximumPerChannel);
        }
    }
    void doIt(Integer maximumPerChannel){
        maximumPerChannel = DEFAULT_MESSAGES_WITH_ACTIONS;
        if (maximumPerChannel == null ){

            log.info("maximumPerChannel param defaulting to " + maximumPerChannel);
        }
        if(maximumPerChannel < 1){

            log.info("maximumPerChannel param defaulting to " + maximumPerChannel);
        }
        if(maximumPerChannel > MAX_MESSAGES_WITH_ACTIONS){
            log.info("maximumPerChannel param defaulting to " + maximumPerChannel);
        }
    }
```

1

classe PNConfiguration.java

método PNConfiguration linha 39

bad smell detectado : Long Identifier [in method PNConfiguration]: The length of the field FILE_MESSAGE_PUBLISH_RETRY_LIMIT is 32.

long identifier = 1

antes da refatoração:

```
    private static final int FILE_MESSAGE_PUBLISH_RETRY_LIMIT = 5;
```

após refatoração:
   private static final int MESSAGE_RETRY = 5;

bad smell removido

2

classe UnwrapSingleField.java

método deserialize linha 11

Erro detectado : Long Statement [in method deserialize]: The length of the statement "throw new IllegalStateException("Couldn't unwrap field for object containing more than 1 field. Actual number of fields: "; + jsonObject.keySet().size());"
long statemet = 1
antes da refatoração:

```
        public T deserialize(final JsonElement json, final Type typeOfT, final
JsonDeserializationContext context) throws JsonParseException {
        final JsonObject jsonObject = json.getAsJsonObject();
        if (jsonObject.keySet().size() != 1) {
        throw new IllegalStateException("Couldn't unwrap field for object containing more
than 1 field. Actual number of fields: "; + jsonObject.keySet().size());
        }
        final String key = jsonObject.keySet().toArray(new String[]{})[0];
        final JsonElement element = jsonObject.get(key);
        return context.deserialize(element, typeOfT);
        }
}
```

após refatoração:

```
        public T deserialize(final JsonElement json, final Type typeOfT, final
JsonDeserializationContext context) throws JsonParseException {
        final JsonObject jsonObject = json.getAsJsonObject();
        if (jsonObject.keySet().size() != 1) {
        string messge= "Couldn't unwrap field for object containing more than 1 field. Actual
number of fields: ";
        throw new IllegalStateException(messge + jsonObject.keySet().size());
        }
        final String key = jsonObject.keySet().toArray(new String[]{})[0];
        final JsonElement element = jsonObject.get(key);
        return context.deserialize(element, typeOfT);
        }
```

}

3

classe SquashBacktrace.java

método populateNestedExceptions linha 80

Erro detectado : Complex Conditional [in method populateNestedExceptions]: The conditional expression error == null || error.getCause() == null || error.getCause() == error is complex.

complex conditional = 1
antes da refatoração:

```
  public static void populateNestedExceptions(List<NestedException> nestedExceptions,
          Throwable error) {
          // Only keep processing if the "cause" exception is set and != the "parent" exception.
          if (error == null || error.getCause() == null || error.getCause() == error) {
          return;
          }
          final Throwable cause = error.getCause();
          NestedException doc =
          new NestedException(cause.getClass().getName(), cause.getMessage(),
getBacktraces(cause),
          getIvars(cause));
          nestedExceptions.add(doc);
          // Exceptions all the way down!
          populateNestedExceptions(nestedExceptions, cause);
  }
```

após refatoração:

```
  public static void populateNestedExceptions(List<NestedException> nestedExceptions,
          Throwable error) {
          // Only keep processing if the "cause" exception is set and != the "parent" exception.
          if (error == null  ) {
          return;
          }
          if(error.getCause() == null ){return;}
          if(error.getCause() == error){return;}

          final Throwable cause = error.getCause();
          NestedException doc =
          new NestedException(cause.getClass().getName(), cause.getMessage(),
getBacktraces(cause),
          getIvars(cause));
```

```
        nestedExceptions.add(doc);
        // Exceptions all the way down!
        populateNestedExceptions(nestedExceptions, cause);
 }
```

4

classe SquashBackTrace.java

método getStacktraceArray linha 42

Erro detectado : Long Statement [in method getStacktraceArray]: The length of the statement "StackElement elementList=new StackElement(element.getClassName()`element.getFileName()`element.getLineNumber()`element.getMethodName());" is 136.
long statemet = 2

antes da refatoração:

```
private static List<StackElement> getStacktraceArray(Throwable error) {
        List<StackElement> stackElems = new ArrayList<StackElement>();
        for (StackTraceElement element : error.getStackTrace()) {
        StackElement elementList =
        new StackElement(element.getClassName(), element.getFileName(),
element.getLineNumber(),
        element.getMethodName());
        stackElems.add(elementList);
        }
        return stackElems;
 }
```

após refatoração:

```
        private static List<StackElement> getStacktraceArray(Throwable error) {
        List<StackElement> stackElems = new ArrayList<StackElement>();
        for (StackTraceElement element : error.getStackTrace()) {
        String className = element.getClassName();
                String fileName = element.getFileName();
                int lineNumber = element.getLineNumber();
                        String methodName = element.getMethodName();
        StackElement elementList =
        new StackElement(className, fileName, lineNumber,
        methodName);
        stackElems.add(elementList);
        }
```

```
        return stackElems;
 }
```

5

classe SquashBacktrace.java

método populateNestedExceptions linha 80

Erro detectado : Long Statement [in method populateNestedExceptions]: The length of the statement "NestedException doc=new NestedException(cause.getClass().getName()`cause.getMessage()`getBacktraces(cause)`getIvars(cause));" is 124.
long statement = 3

antes da refatoração:

```
        public static void populateNestedExceptions(List<NestedException>
nestedExceptions,
        Throwable error) {
        // Only keep processing if the "cause" exception is set and != the "parent" exception.
        if (error == null  ) {
        return;
        }
        if(error.getCause() == null ){return;}
        if(error.getCause() == error){return;}

        final Throwable cause = error.getCause();

        NestedException doc =
        new NestedException(cause.getClass().getName(),cause.getMessage(),
getBacktraces(cause),
        getIvars(cause));
        nestedExceptions.add(doc);
        // Exceptions all the way down!
        populateNestedExceptions(nestedExceptions, cause);
 }
```

após refatoração:

```
        public static void populateNestedExceptions(List<NestedException>
nestedExceptions,
        Throwable error) {
        // Only keep processing if the "cause" exception is set and != the "parent" exception.
        if (error == null  ) {
        return;
```

```
        }
        if(error.getCause() == null ){return;}
        if(error.getCause() == error){return;}

        final Throwable cause = error.getCause();
        String name = cause.getClass().getName();
        String message = cause.getMessage();
        NestedException doc =
        new NestedException(name,message, getBacktraces(cause),
        getIvars(cause));
        nestedExceptions.add(doc);
        // Exceptions all the way down!
        populateNestedExceptions(nestedExceptions, cause);
 }
```

6

classe SquashBacktrace.java

método getBacktraces linha 31

Erro detectado : Long Statement [in method getBacktraces]: The length of the statement "final SquashException currentThread=new SquashException(Thread.currentThread().getName()`true`getStacktraceArray(error));" is 121.
long statement = 4
antes da refatoração:

```
        public static List<SquashException> getBacktraces(Throwable error) {
        if (error == null) {
        return null;
        }
        final List<SquashException> threadList = new ArrayList<SquashException>();
        final SquashException currentThread =
        new SquashException(Thread.currentThread().getName(), true,
getStacktraceArray(error));
        threadList.add(currentThread);
        return threadList;
 }
```

após refatoração:

```
        public static List<SquashException> getBacktraces(Throwable error) {
        if (error == null) {
        return null;
        }
        final List<SquashException> threadList = new ArrayList<SquashException>();
```

```
            String name = Thread.currentThread().getName();
    final SquashException currentThread =
            new SquashException(name, true, getStacktraceArray(error));
            threadList.add(currentThread);
            return threadList;
  }
```

7

classe Pubnub.java

método destroy linha 555

Erro detectado :Empty catch clause [in method destroy]: The method has an empty catch block.
empty catch clause : 1
antes da refatoração:

```
            public void destroy() {
            try {
            subscriptionManager.destroy(false);
            retrofitManager.destroy(false);
            } catch (Exception error) {
            //
            }
            }
```

após refatoração:

```
            public void destroy() {
            try {
            subscriptionManager.destroy(false);
            retrofitManager.destroy(false);
            } catch (Exception error) {
            return ;
            }
            }
```

8

classe Pubnub.java

método forceDestroy linha 567

Erro detectado :Empty catch clause [in method destroy]: The method has an empty catch block.
empty catch clause : 2
antes da refatoração:

```
public void forceDestroy() {
try {
subscriptionManager.destroy(true);
retrofitManager.destroy(true);
telemetryManager.stopCleanUpTimer();
} catch (Exception error) {
//
}
}
```

após refatoração:

```
public void forceDestroy() {
try {
subscriptionManager.destroy(true);
retrofitManager.destroy(true);
telemetryManager.stopCleanUpTimer();
} catch (Exception error) {
return;
}
}
```

9

classe PubNubExceptionTest.java

método testPubnubError linha 38

Erro detectado : Long Statement [in method testPubnubError]: The length of the statement "stubFor(get(urlPathEqualTo("/publish/myPublishKey/mySubscribeKey/0/coolChannel/0/%22hi%22")).willReturn(aResponse().withStatus(404).withBody("[1`\"Sent\"`\"14598111595318003\"]")));" is 181.
long statement: 5

antes da refatoração:

```
  public void testPubnubError() {]

stubFor(get(urlPathEqualTo("/publish/myPublishKey/mySubscribeKey/0/coolChannel/0/%22hi%22"))

.willReturn(aResponse().withStatus(404).withBody("[1,\"Sent\",\"14598111595318003\"]")));

        int statusCode = -1;
```

```
        try {
        instance.channel("coolChannel").message("hi").sync();
        } catch (PubNubException error) {
        statusCode = error.getStatusCode();
        }

        assertEquals(404, statusCode);
        }
```

após refatoração:

```
        public void testPubnubError() {]
        String url = "/publish/myPublishKey/mySubscribeKey/0/coolChannel/0/%22hi%22";
        String sent = "[1,\"Sent\",\"14598111595318003\"]";
        stubFor(get(urlPathEqualTo(url))
                .willReturn(aResponse().withStatus(404).withBody(sent)));

        int statusCode = -1;

        try {
        instance.channel("coolChannel").message("hi").sync();
        } catch (PubNubException error) {
        statusCode = error.getStatusCode();
        }

        assertEquals(404, statusCode);
        }
```

10

classe PubNubErrorBuilder.java

método createCryptoError linha 717

Erro detectado : long statement:6

antes da refatoração:

```
        public static PubNubError createCryptoError(int code, String message) {


        return PubNubError.builder()
                .errorCode(PNERR_CRYPTO_ERROR)
                .errorCodeExtended(code)
                .message("Error while encrypting/decrypting message. Please contact
support with error details. - ".concat(message))
                .build();
```

```
        }
```

após refatoração:

```
   public static PubNubError createCryptoError(int code, String message) {

        String err = "Error while encrypting/decrypting message. Please contact support with
error details. - ";
        return PubNubError.builder()
                .errorCode(PNERR_CRYPTO_ERROR)
                .errorCodeExtended(code)
                .message(err.concat(message))
                .build();
        }
```

11

classe Grant.java

método validateParams linha 81

Erro detectado : Complex Conditional [in method validateParams]: The conditional expression (!channels.isEmpty() || !channelGroups.isEmpty()) && !uuids.isEmpty() is complex.
complex conditional: 2

antes da refatoração:

```
        protected void validateParams() throws PubNubException {
        if (this.getPubnub().getConfiguration().getSecretKey() == null || this.getPubnub()
                .getConfiguration()
                .getSecretKey()
                .isEmpty()) {
        throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SECRET_KEY_
MISSING).build();
        }
        if (this.getPubnub().getConfiguration().getSubscribeKey() == null || this.getPubnub()
                .getConfiguration()
                .getSubscribeKey()
                .isEmpty()) {
        throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SUBSCRIBE_KE
Y_MISSING).build();
        }
        if (this.getPubnub().getConfiguration().getPublishKey() == null || this.getPubnub()
```

```
                .getConfiguration()
                .getPublishKey()
                .isEmpty()) {
        throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_PUBLISH_KEY_
MISSING).build();
        }
        if ((!channels.isEmpty() || !channelGroups.isEmpty()) && !uuids.isEmpty()) {
        throw PubNubException.builder()
                .pubnubError(PubNubErrorBuilder.PNERROBJ_INVALID_ARGUMENTS)
                .errormsg("Grants for channels or channelGroups can't be changed together
with grants for UUIDs")
                .build();
        }
        if (!uuids.isEmpty() && authKeys.isEmpty()) {
        throw PubNubException.builder()
                .pubnubError(PubNubErrorBuilder.PNERROBJ_INVALID_ARGUMENTS)
                .errormsg("UUIDs grant management require providing non empty authKeys")
                .build();
        }
        }


após refatoração:
        protected void validateParams() throws PubNubException {
        if (this.getPubnub().getConfiguration().getSecretKey() == null || this.getPubnub()
                .getConfiguration()
                .getSecretKey()
                .isEmpty()) {
        throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SECRET_KEY_
MISSING).build();
        }
        if (this.getPubnub().getConfiguration().getSubscribeKey() == null || this.getPubnub()
                .getConfiguration()
                .getSubscribeKey()
                .isEmpty()) {
        throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SUBSCRIBE_KE
Y_MISSING).build();
        }
        if (this.getPubnub().getConfiguration().getPublishKey() == null || this.getPubnub()
                .getConfiguration()
                .getPublishKey()
                .isEmpty()) {
        throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_PUBLISH_KEY_
MISSING).build();
```

```
        }
        if ((!channels.isEmpty() && !uuids.isEmpty()) {
        something();
        }
        if ((!channelGroups.isEmpty()) && !uuids.isEmpty()) {
        something();
        }
        if (!uuids.isEmpty() && authKeys.isEmpty()) {
        throw PubNubException.builder()
                .pubnubError(PubNubErrorBuilder.PNERROBJ_INVALID_ARGUMENTS)
                .errormsg("UUIDs grant management require providing non empty authKeys")
                .build();
        }
        }
        void something(){
        throw PubNubException.builder()
                .pubnubError(PubNubErrorBuilder.PNERROBJ_INVALID_ARGUMENTS)
                .errormsg("Grants for channels or channelGroups can't be changed together
with grants for UUIDs")
                .build();
        }
```

12

classe Grant.java

método createResponse() linha 196

Erro detectado : Complex Method [in method createResponse]: Cyclomatic complexity of the method is 11
complex method: 1

antes da refatoração:

```
   protected PNAccessManagerGrantResult
createResponse(Response<Envelope<AccessManagerGrantPayload>> input) throws
        PubNubException {
        MapperManager mapperManager = getPubnub().getMapper();
        PNAccessManagerGrantResult.PNAccessManagerGrantResultBuilder
pnAccessManagerGrantResult =
                PNAccessManagerGrantResult.builder();

        if (input.body() == null || input.body().getPayload() == null) {
        throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_PARSING_ERR
OR).build();
        }
```

```
        AccessManagerGrantPayload data = input.body().getPayload();
        Map<String, Map<String, PNAccessManagerKeyData>> constructedChannels = new
HashMap<>();
        Map<String, Map<String, PNAccessManagerKeyData>> constructedGroups = new
HashMap<>();

        // we have a case of a singular channel.
        if (data.getChannel() != null) {
        constructedChannels.put(data.getChannel(), data.getAuthKeys());
        }

        if (channelGroups != null) {
        if (channelGroups.size() == 1) {

constructedGroups.put(mapperManager.elementToString(data.getChannelGroups()),
data.getAuthKeys());
        } else if (channelGroups.size() > 1) {
                Iterator<Map.Entry<String, JsonElement>> it =
mapperManager.getObjectIterator(data.getChannelGroups());
                while (it.hasNext()) {
                Map.Entry<String, JsonElement> channelGroup = it.next();
                constructedGroups.put(channelGroup.getKey(),
createKeyMap(channelGroup.getValue()));
                }
        }
        }

        if (data.getChannels() != null) {
        for (String fetchedChannel : data.getChannels().keySet()) {
                constructedChannels.put(fetchedChannel,
data.getChannels().get(fetchedChannel).getAuthKeys());
        }
        }

        Map<String, Map<String, PNAccessManagerKeyData>> constructedUuids = new
HashMap<>();

        if (data.getUuids() != null) {
        for (String fetchedUuid : data.getUuids().keySet()) {
                constructedUuids.put(fetchedUuid,
data.getUuids().get(fetchedUuid).getAuthKeys());
        }
        }

        return pnAccessManagerGrantResult
                .subscribeKey(data.getSubscribeKey())
                .level(data.getLevel())
```

```
                .ttl(data.getTtl())
                .channels(constructedChannels)
                .channelGroups(constructedGroups)
                .uuids(constructedUuids)
                .build();
        }
```

após refatoração:

```
        @Override
        protected PNAccessManagerGrantResult
createResponse(Response<Envelope<AccessManagerGrantPayload>> input) throws
        PubNubException {
        MapperManager mapperManager = getPubnub().getMapper();
        PNAccessManagerGrantResult.PNAccessManagerGrantResultBuilder
pnAccessManagerGrantResult =
                PNAccessManagerGrantResult.builder();

        if (input.body() == null || input.body().getPayload() == null) {
        throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_PARSING_ERR
OR).build();
        }

        AccessManagerGrantPayload data = input.body().getPayload();
        Map<String, Map<String, PNAccessManagerKeyData>> constructedGroups = new
HashMap<>();
        Map<String, Map<String, PNAccessManagerKeyData>> constructedChannels = new
HashMap<>();
        Map<String, Map<String, PNAccessManagerKeyData>> constructedUuids = new
HashMap<>();


        // we have a case of a singular channel.
        actoin(data,constructedChannels);
        actoin1(data,constructedGroups);
        actoin2(data,constructedChannels);
        actoin3(data,constructedUuids);




        return pnAccessManagerGrantResult
                .subscribeKey(data.getSubscribeKey())
                .level(data.getLevel())
```

```
                .ttl(data.getTtl())
                .channels(constructedChannels)
                .channelGroups(constructedGroups)
                .uuids(constructedUuids)
                .build();
        }
        void actoin(AccessManagerGrantPayload data,Map<String, Map<String,
PNAccessManagerKeyData>> constructedChannels){
        //AccessManagerGrantPayload data = input.body().getPayload();
        //Map<String, Map<String, PNAccessManagerKeyData>> constructedChannels =
new HashMap<>();
        if (data.getChannel() != null) {
        constructedChannels.put(data.getChannel(), data.getAuthKeys());
        }
        }
        void actoin1(AccessManagerGrantPayload data, Map<String, Map<String,
PNAccessManagerKeyData>> constructedGroups){
        //AccessManagerGrantPayload data = input.body().getPayload();
        //Map<String, Map<String, PNAccessManagerKeyData>> constructedGroups = new
HashMap<>();
        if (channelGroups != null) {
        if (channelGroups.size() == 1) {

constructedGroups.put(mapperManager.elementToString(data.getChannelGroups()),
data.getAuthKeys());
        } else if (channelGroups.size() > 1) {
                Iterator<Map.Entry<String, JsonElement>> it =
mapperManager.getObjectIterator(data.getChannelGroups());
                while (it.hasNext()) {
                Map.Entry<String, JsonElement> channelGroup = it.next();
                constructedGroups.put(channelGroup.getKey(),
createKeyMap(channelGroup.getValue()));
                }
        }
        }
        }
        void actoin2(AccessManagerGrantPayload data,  Map<String, Map<String,
PNAccessManagerKeyData>> constructedChannels){
        //AccessManagerGrantPayload data = input.body().getPayload();
        //Map<String, Map<String, PNAccessManagerKeyData>> constructedChannels =
new HashMap<>();
        if (data.getChannels() != null) {
        for (String fetchedChannel : data.getChannels().keySet()) {
                constructedChannels.put(fetchedChannel,
data.getChannels().get(fetchedChannel).getAuthKeys());
        }
        }
        }
```

```
	void actoin3( AccessManagerGrantPayload data, Map<String, Map<String,
PNAccessManagerKeyData>> constructedUuids){
	//AccessManagerGrantPayload data = input.body().getPayload();
	//Map<String, Map<String, PNAccessManagerKeyData>> constructedUuids = new
HashMap<>();

	if (data.getUuids() != null) {
	for (String fetchedUuid : data.getUuids().keySet()) {
		constructedUuids.put(fetchedUuid,
data.getUuids().get(fetchedUuid).getAuthKeys());
	}
	}
	}
```

13

classe GrantToken.java

método validateParams linha 71

Erro detectado : Complex Conditional [in method validateParams]: The conditional expression isNullOrEmpty(channels) && isNullOrEmpty(channelGroups) && isNullOrEmpty(uuids) is complex.
complex conditional: 3
antes da refatoração:

```
  protected void validateParams() throws PubNubException {
	if (this.getPubnub().getConfiguration().getSecretKey() == null || this.getPubnub()
		.getConfiguration()
		.getSecretKey()
		.isEmpty()) {
	throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SECRET_KEY_
MISSING).build();
	}
	if (this.getPubnub().getConfiguration().getSubscribeKey() == null || this.getPubnub()
		.getConfiguration()
		.getSubscribeKey()
		.isEmpty()) {
	throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SUBSCRIBE_KE
Y_MISSING).build();
	}
	if (isNullOrEmpty(channels)
		&& isNullOrEmpty(channelGroups)
		&& isNullOrEmpty(uuids)) {
	throw PubNubException.builder()
		.pubnubError(PubNubErrorBuilder.PNERROBJ_RESOURCES_MISSING)
```

```
                .build();
        }
        if (this.ttl == null) {
        throw PubNubException.builder()
                .pubnubError(PubNubErrorBuilder.PNERROBJ_TTL_MISSING)
                .build();
        }
        }
```

após refatoração:

```
        protected void validateParams() throws PubNubException {
        if (this.getPubnub().getConfiguration().getSecretKey() == null || this.getPubnub()
                .getConfiguration()
                .getSecretKey()
                .isEmpty()) {
        throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SECRET_KEY_
MISSING).build();
        }
        if (this.getPubnub().getConfiguration().getSubscribeKey() == null || this.getPubnub()
                .getConfiguration()
                .getSubscribeKey()
                .isEmpty()) {
        throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SUBSCRIBE_KE
Y_MISSING).build();
        }
        if (isNullOrEmpty(channels) {
        throw PubNubException.builder()
                .pubnubError(PubNubErrorBuilder.PNERROBJ_RESOURCES_MISSING)
                .build();
        }
        if (isNullOrEmpty(channelGroups)) {
        throw PubNubException.builder()
                .pubnubError(PubNubErrorBuilder.PNERROBJ_RESOURCES_MISSING)
                .build();
        }
        if (isNullOrEmpty(uuids)) {
        throw PubNubException.builder()
                .pubnubError(PubNubErrorBuilder.PNERROBJ_RESOURCES_MISSING)
                .build();
        }
        if (this.ttl == null) {
        throw PubNubException.builder()
                .pubnubError(PubNubErrorBuilder.PNERROBJ_TTL_MISSING)
                .build();
```

```
        }
        }
```

14

classe RevokeToken.java

método método validateParams linha 47

Erro detectado : Long Statement [in method validateParams]: The length of the statement "if (this.getPubnub().getConfiguration().getSecretKey() == null || this.getPubnub().getConfiguration().getSecretKey().isEmpty()) {" is 129.
long statement: 7
antes da refatoração:

```
  protected void validateParams() throws PubNubException {
        if (this.getPubnub().getConfiguration().getSecretKey() == null || this.getPubnub()
                .getConfiguration()
                .getSecretKey()
                .isEmpty()) {
        throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SECRET_KEY_
MISSING).build();
        }
        if (this.getPubnub().getConfiguration().getSubscribeKey() == null || this.getPubnub()
                .getConfiguration()
                .getSubscribeKey()
                .isEmpty()) {
        throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SUBSCRIBE_KE
Y_MISSING).build();
        }
        if (this.token == null) {
        throw PubNubException.builder()
                .pubnubError(PubNubErrorBuilder.PNERROBJ_TOKEN_MISSING)
                .build();
        }
        }
```

após refatoração:

```
        protected void validateParams() throws PubNubException {
        if (this.getPubnub().getConfiguration().getSecretKey() == null ){
        throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SECRET_KEY_
MISSING).build();
        }
```

```
        if(this.getPubnub()
                .getConfiguration()
                .getSecretKey()
                .isEmpty()){
        throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SECRET_KEY_
MISSING).build();
        }

   //a método acima
        if (this.getPubnub().getConfiguration().getSubscribeKey() == null ) {
        throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SUBSCRIBE_KE
Y_MISSING).build();
        }
        if( this.getPubnub()
                .getConfiguration()
                .getSubscribeKey()
                .isEmpty()){
        throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SUBSCRIBE_KE
Y_MISSING).build();
        }
        if (this.token == null) {
        throw PubNubException.builder()
                .pubnubError(PubNubErrorBuilder.PNERROBJ_TOKEN_MISSING)
                .build();
        }
        }
```

15

classe RevokeToken.java

método validateParams linha 47

Erro detectado : Long Statement [in method validateParams]: The length of the statement "if (this.getPubnub().getConfiguration().getSubscribeKey() == null || this.getPubnub().getConfiguration().getSubscribeKey().isEmpty()) {" is 135.
long statement: 8
antes da refatoração:

```
        protected void validateParams() throws PubNubException {
        if (this.getPubnub().getConfiguration().getSecretKey() == null || this.getPubnub()
                .getConfiguration()
                .getSecretKey()
                .isEmpty()) {
```

```
	throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SECRET_KEY_
MISSING).build();
	}
	if (this.getPubnub().getConfiguration().getSubscribeKey() == null || this.getPubnub()
		.getConfiguration()
		.getSubscribeKey()
		.isEmpty()) {
	throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SUBSCRIBE_KE
Y_MISSING).build();
	}
	if (this.token == null) {
	throw PubNubException.builder()
		.pubnubError(PubNubErrorBuilder.PNERROBJ_TOKEN_MISSING)
		.build();
	}
	}
```

após refatoração:

```
	protected void validateParams() throws PubNubException {
	if (this.getPubnub().getConfiguration().getSecretKey() == null ){
	throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SECRET_KEY_
MISSING).build();
	}
	if(this.getPubnub()
		.getConfiguration()
		.getSecretKey()
		.isEmpty()){
	throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SECRET_KEY_
MISSING).build();
	}
   //a método abaixo
	if (this.getPubnub().getConfiguration().getSubscribeKey() == null ) {
	throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SUBSCRIBE_KE
Y_MISSING).build();
	}
	if( this.getPubnub()
		.getConfiguration()
		.getSubscribeKey()
		.isEmpty()){
	throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SUBSCRIBE_KE
Y_MISSING).build();
```

```
        }
        if (this.token == null) {
        throw PubNubException.builder()
                .pubnubError(PubNubErrorBuilder.PNERROBJ_TOKEN_MISSING)
                .build();
        }
        }
```

16

classe RevokeToken.java

método doWork linha 74

Erro detectado : Long Statement [in method doWork]: The length of the statement "return getRetrofit().getAccessManagerService().revokeToken(getPubnub().getConfiguration().getSubscribeKey()`repairEncoding(token)`baseParams);" is 142.
long statement: 9
antes da refatoração:

```
  protected Call<RevokeTokenResponse> doWork(Map<String, String> baseParams)
throws PubNubException {
        return getRetrofit()
                .getAccessManagerService()
                .revokeToken( getPubnub().getConfiguration().getSubscribeKey(),
repairEncoding(token), baseParams);
        }
```

após refatoração:

```
        protected Call<RevokeTokenResponse> doWork(Map<String, String> baseParams)
throws PubNubException {
        PubNub aux = getPubnub().getConfiguration().getSubscribeKey();
        return getRetrofit()
                .getAccessManagerService()
                .revokeToken(aux, repairEncoding(token), baseParams);
        }
```

17

classe ListFiles.java

método validateParams linha 60

Erro detectado : Complex Conditional [in method validateParams]: The conditional expression next != null && (next.getHash() == null || next.getHash().isEmpty()) is complex.
complex conditional: 4

antes da refatoração:

```
	protected void validateParams() throws PubNubException {
	if (this.getPubnub().getConfiguration().getSubscribeKey() == null
		|| this.getPubnub().getConfiguration().getSubscribeKey().isEmpty()) {
	throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SUBSCRIBE_KE
Y_MISSING).build();
	}

	if (channel == null || channel.isEmpty()) {
	throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_CHANNEL_MIS
SING).build();
	}

	if (limit != null && !(MIN_LIMIT <= limit && limit <= MAX_LIMIT)) {
	throw PubNubException.builder()
		.pubnubError(PubNubErrorBuilder.PNERROBJ_INVALID_ARGUMENTS)
		.errormsg("Limit should be in range from 1 to 100 (both inclusive)")
		.build();
	}

	if (next != null && (next.getHash() == null || next.getHash().isEmpty())) {
	throw PubNubException.builder()
		.pubnubError(PubNubErrorBuilder.PNERROBJ_INVALID_ARGUMENTS)
		.errormsg("Next should not be an empty string")
		.build();
	}
	}
```

após refatoração:

```
	protected void validateParams() throws PubNubException {
	if (this.getPubnub().getConfiguration().getSubscribeKey() == null
		|| this.getPubnub().getConfiguration().getSubscribeKey().isEmpty()) {
	throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SUBSCRIBE_KE
Y_MISSING).build();
	}

	if (channel == null || channel.isEmpty()) {
	throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_CHANNEL_MIS
SING).build();
	}

	if (limit != null && !(MIN_LIMIT <= limit && limit <= MAX_LIMIT)) {
	throw PubNubException.builder()
```

```
                .pubnubError(PubNubErrorBuilder.PNERROBJ_INVALID_ARGUMENTS)
                .errormsg("Limit should be in range from 1 to 100 (both inclusive)")
                .build();
        }

        if (next != null && next.getHash() == null) {
        toDo();
        }
        if (next != null && next.getHash().isEmpty()) {
        toDo();
        }
        }
        void toDO(){
        String msg = "";
        throw PubNubException.builder()
                .pubnubError(PubNubErrorBuilder.PNERROBJ_INVALID_ARGUMENTS)
                .errormsg("Next should not be an empty string")
                .build();
        }
```

18

classe ListFiles.java

método validadeParams linha 60

Erro detectado : Long Statement [in method validateParams]: The length of the statement "if (this.getPubnub().getConfiguration().getSubscribeKey() == null || this.getPubnub().getConfiguration().getSubscribeKey().isEmpty()) {" is 135.
long statement: 10
antes da refatoração:

```
        protected void validateParams() throws PubNubException {
        if (this.getPubnub().getConfiguration().getSubscribeKey() == null
                || this.getPubnub().getConfiguration().getSubscribeKey().isEmpty()) {
        throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SUBSCRIBE_KEY_MISSING).build();
        }

        if (channel == null || channel.isEmpty()) {
        throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_CHANNEL_MISSING).build();
        }

        if (limit != null && !(MIN_LIMIT <= limit && limit <= MAX_LIMIT)) {
        throw PubNubException.builder()
                .pubnubError(PubNubErrorBuilder.PNERROBJ_INVALID_ARGUMENTS)
```

```
                .errormsg("Limit should be in range from 1 to 100 (both inclusive)")
                .build();
        }

        if (next != null && next.getHash() == null) {
        toDo();
        }
        if (next != null && next.getHash().isEmpty()) {
        toDo();
        }
        }
```

após refatoração:

```
        protected void validateParams() throws PubNubException {
        if (this.getPubnub().getConfiguration().getSubscribeKey() == null
                || this.getPubnub().getConfiguration().getSubscribeKey().isEmpty()) {
        throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SUBSCRIBE_KE
Y_MISSING).build();
        }

        if (channel == null || channel.isEmpty()) {
        throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_CHANNEL_MIS
SING).build();
        }

        if (limit != null && !(MIN_LIMIT <= limit && limit <= MAX_LIMIT)) {
        String str = "Limit should be in range from 1 to 100 (both inclusive)";
        throw PubNubException.builder()
                .pubnubError(PubNubErrorBuilder.PNERROBJ_INVALID_ARGUMENTS)
                .errormsg(str)
                .build();
        }

        if (next != null && next.getHash() == null) {
        toDo();
        }
        if (next != null && next.getHash().isEmpty()) {
        toDo();
        }
        }
```

19

classe History.java

método doWork Linha 80

Erro detectado : Complex Conditional [in method doWork]: The conditional expression count != null && count > 0 && count <= MAX_COUNT is complex.
  complex conditional: 6
 protected Call<JsonElement> doWork(Map<String, String> params) {

```
        if (reverse != null) {
        params.put("reverse", String.valueOf(reverse));
        }

        if (includeTimetoken != null) {
        params.put("include_token", String.valueOf(includeTimetoken));
        }

        if (includeMeta) {
        params.put("include_meta", String.valueOf(includeMeta));
        }

        if (count != null && count > 0 && count <= MAX_COUNT) {
        params.put("count", String.valueOf(count));
        } else {
        params.put("count", "100");
        }

        if (start != null) {
        params.put("start", Long.toString(start).toLowerCase());
        }
        if (end != null) {
        params.put("end", Long.toString(end).toLowerCase());
        }

        return
this.getRetrofit().getHistoryService().fetchHistory(this.getPubnub().getConfiguration()
                .getSubscribeKey(), channel, params);
        }
```

após refatoração:

20

classe FetchMessages.java

método validateParams linha 80

Erro detectado : Complex Conditional [in method validateParams]: The conditional expression maximumPerChannel == null || maximumPerChannel < 1 || maximumPerChannel > MAX_MESSAGES_WITH_ACTIONS is complex.
complex conditional: 7
antes da refatoração:

```
	protected void validateParams() throws PubNubException {
	if (this.getPubnub().getConfiguration().getSubscribeKey() == null
		|| this.getPubnub().getConfiguration().getSubscribeKey().isEmpty()) {
	throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SUBSCRIBE_KE
Y_MISSING).build();
	}

	if (channels == null || channels.size() == 0) {
	throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_CHANNEL_MIS
SING).build();
	}

	if (includeMeta == null) {
	includeMeta = false;
	}

	if (includeMessageActions == null) {
	includeMessageActions = false;
	}

	if (!includeMessageActions) {
	if (channels.size() == 1) {
		if (maximumPerChannel == null || maximumPerChannel < 1) {
		maximumPerChannel = SINGLE_CHANNEL_DEFAULT_MESSAGES;
		log.info("maximumPerChannel param defaulting to " + maximumPerChannel);
		} else if (maximumPerChannel > SINGLE_CHANNEL_MAX_MESSAGES) {
		maximumPerChannel = SINGLE_CHANNEL_MAX_MESSAGES;
		log.info("maximumPerChannel param defaulting to " + maximumPerChannel);
		}
	} else {
		if (maximumPerChannel == null || maximumPerChannel < 1) {
		maximumPerChannel = MULTIPLE_CHANNEL_DEFAULT_MESSAGES;
		log.info("maximumPerChannel param defaulting to " + maximumPerChannel);
		} else if (maximumPerChannel > MULTIPLE_CHANNEL_MAX_MESSAGES)
{
		maximumPerChannel = MULTIPLE_CHANNEL_MAX_MESSAGES;
		log.info("maximumPerChannel param defaulting to " + maximumPerChannel);
		}
	}
```

```
        } else {

        if (maximumPerChannel == null || maximumPerChannel < 1 || maximumPerChannel
> MAX_MESSAGES_WITH_ACTIONS){
         maximumPerChannel = DEFAULT_MESSAGES_WITH_ACTIONS;
        log.info("maximumPerChannel param defaulting to " + maximumPerChannel);
        }

        }
```

após refatoração:

```
        protected void validateParams() throws PubNubException {
        if (this.getPubnub().getConfiguration().getSubscribeKey() == null
                || this.getPubnub().getConfiguration().getSubscribeKey().isEmpty()) {
        throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_SUBSCRIBE_KE
Y_MISSING).build();
        }

        if (channels == null || channels.size() == 0) {
        throw
PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_CHANNEL_MIS
SING).build();
        }

        if (includeMeta == null) {
        includeMeta = false;
        }

        if (includeMessageActions == null) {
        includeMessageActions = false;
        }

        if (!includeMessageActions) {
        if (channels.size() == 1) {
                if (maximumPerChannel == null || maximumPerChannel < 1) {
                maximumPerChannel = SINGLE_CHANNEL_DEFAULT_MESSAGES;
                log.info("maximumPerChannel param defaulting to " + maximumPerChannel);
                } else if (maximumPerChannel > SINGLE_CHANNEL_MAX_MESSAGES) {
                maximumPerChannel = SINGLE_CHANNEL_MAX_MESSAGES;
                log.info("maximumPerChannel param defaulting to " + maximumPerChannel);
                }
        } else {
                if (maximumPerChannel == null || maximumPerChannel < 1) {
                maximumPerChannel = MULTIPLE_CHANNEL_DEFAULT_MESSAGES;
                log.info("maximumPerChannel param defaulting to " + maximumPerChannel);
```

```
                } else if (maximumPerChannel > MULTIPLE_CHANNEL_MAX_MESSAGES)
{
                maximumPerChannel = MULTIPLE_CHANNEL_MAX_MESSAGES;
                log.info("maximumPerChannel param defaulting to " + maximumPerChannel);
                }
        }
        } else {
        doIt(maximumPerChannel);
        }
        }
        void doIt(Integer maximumPerChannel){
        maximumPerChannel = DEFAULT_MESSAGES_WITH_ACTIONS;
        if (maximumPerChannel == null ){

        log.info("maximumPerChannel param defaulting to " + maximumPerChannel);
        }
        if(maximumPerChannel < 1){

        log.info("maximumPerChannel param defaulting to " + maximumPerChannel);
        }
        if(maximumPerChannel > MAX_MESSAGES_WITH_ACTIONS){
        log.info("maximumPerChannel param defaulting to " + maximumPerChannel);
        }
        }
```