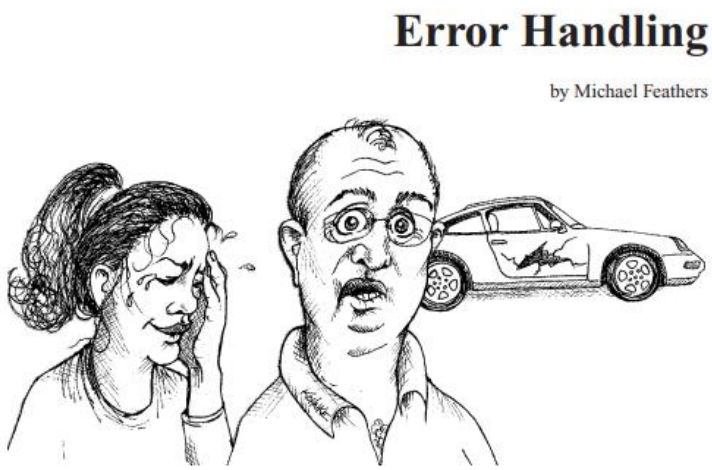
1/14/2023

Clean code

Ch7 Error Handling



Abdurahman Gamal Ahmed
NO CAMPANY

Table of Contents

1-intro:

عم بوب بيبداء الشبتر وبيقلك يمكن حاجه غريبه ان يكون في شابتر كامل عن الError handling ف كتاب بتكلم عن الclean code .

ب الError handling هنا حاجه بتعملها مع الكود وممكن يكون ال input غلط والبرنامج يضرب ف دورنا كمبرمجين ان نصلح الدنيا ونتاكد ان الكود بيقوم بدوره تمم .

- ف الشابتر ده هنتكلم عن طرق واساليب نكتب بيها error handling بشكل clean وقوى .

2- Use Exceptions Rather Than Return Codes :

الاحسن نستخدام ال Exceptions بدال منرجع error code . زمان كان فيه لوغات كتير مفهاش Exceptions والطرق الى كانت مستخدامه عشان تهندل ال error كانت محدوده لما بتحط error flag اوبترجع error code زى الاسكرين دى:

Listing 7-1
DeviceController.java

```
public class DeviceController {
  ...
  public void sendShutDown() {
    DeviceHandle handle = getHandle(DEV1);
    // Check the state of the device
    if (handle != DeviceHandle.INVALID) {
      // Save the device status to the record field
      retrieveDeviceRecord(handle);
      // If not suspended, shut down
      if (record.getStatus() != DEVICE_SUSPENDED) {
        pauseDevice(handle);
        clearDeviceWorkQueue(handle);
        closeDevice(handle);
      } else {
        logger.log("Device suspended.  Unable to shut down");
      }
    } else {
      logger.log("Invalid handle for: " + DEV1.toString());
    }
  }
  ...
}
```

- الاسكرين دى بتوضح الطرق الى كانت بستخدامه زمان لعمل هاندل للerror : المشكله ف الطريقه دى انها بتلخبط الى هيستخدام الطرق دى .

والاحسن ان نعمل throw exception زى الصوره الى جايه :والطريقه الى جايه احسن عشان خلصتنا من مشكلتين ف الكود الى فات

1- ان فصلت ال logic بتاعنا عن ال error handling ف كدة نقدر نشوف كل واحد فيهم على حدا ونفهمه لواحده بطريقه منفصله.
2- انها خلت شكل الكود احسن ومنظم اكتر .
3- وطبعا الدوال بقت اصغر .

الجزء الى جاى انا الى بقوله مش عم بوب: هو مشرحش الاسكرين عشان كان شرحها قبل كده وكل الى فيها ان خلى ال bodyبتاع ال try و ال catch فانكشن بيعمها call .

Listing 7-2
DeviceController.java (with exceptions)

```
public class DeviceController {
  ...

  public void sendShutDown() {
    try {
      tryToShutDown();
    } catch (DeviceShutDownError e) {
      logger.log(e);
    }
  }

  private void tryToShutDown() throws DeviceShutDownError {
    DeviceHandle handle = getHandle(DEV1);
    DeviceRecord record = retrieveDeviceRecord(handle);

    pauseDevice(handle);
    clearDeviceWorkQueue(handle);
    closeDevice(handle);
  }

  private DeviceHandle getHandle(DeviceID id) {
    ...
    throw new DeviceShutDownError("Invalid handle for: " + id.toString());
    ...
  }

  ...
}
```

2- Write Your Try-Catch-Finally Statement First:

من افضل الحجات ف الexception ان ليه scope يعنى ال body بتاعه بيكون جوه {} . وال try بتكون لوحدها وال catch برضو بتكون لوحدها وال catch لازم تسيب البرنامج ف حاله شغاله وتمام مهما كان الى حصل ف ال try .

تعاله ناخد مثال : ف المثال ده الكود بيعمل access لfile وبيقره شويه serialized object والصورة دى كمان فيها unit test بتقول exception لو ال file مش موجود . طبعا الاسكرين دى مش كامله ف انا قلتلك عم بوب عاوز يعمل ايه.

```
@Test(expected = StorageException.class)
public void retrieveSectionShouldThrowOnInvalidFileName() {
  sectionStore.retrieveSection("invalid - file");
}
```

The test drives us to create this stub:

```
public List<RecordedGrip> retrieveSection(String sectionName) {
  // dummy return until we have a real implementation
  return new ArrayList<RecordedGrip>();
}
```

- تانى عشان تفهم الاسكرين الى فوق دى عم بوب بيقلك الى مفروض تعمله بس هى مش كامله هى مفروض بتقدر حاجه من file .
طيب دلوقتى لو ال file مش موجود البرنامج هيضرب ويقف . ف عوزين عمل حل احين من كده ف هنستخدام ال exception .

- ده ال refactoring للكود الى فات : كل الحكايه ان ضفت كود ال try/catch وطبعا ال file Stream دة الى بيقره من file ف لو شم موجود هيرمى exception .

```
public List<RecordedGrip> retrieveSection(String sectionName) {
  try {
    FileInputStream stream = new FileInputStream(sectionName)
  } catch (Exception e) {
    throw new StorageException("retrieval error", e);
  }
  return new ArrayList<RecordedGrip>();
}
```

- نقدر نحسن الكود الى فات ونحدد نوع ال exception المحتمل الى ممكن يحصل . زى الاسكرين دى كل الى اتغير ان شلت كلمه exception وخليتها ادق او محدده اكتر FileNotFoundException وده احسن

```
public List<RecordedGrip> retrieveSection(String sectionName) {
  try {
    FileInputStream stream = new FileInputStream(sectionName);
    stream.close();
  } catch (FileNotFoundException e) {
    throw new StorageException("retrieval error", e);
  }
  return new ArrayList<RecordedGrip>();
}
```

3- Use Unchecked Exceptions:

السكشن ده بيتكلم عن حاجه اسمها Unchecked Exceptions ودى موجوده ف الجافا بس اما ال c++ و #cو c مفهاش

ال Unchecked Exceptions ف انا مش هكتب السكشن ده وهو مش مهم من وجه نظرى انا قريته ولقيته مش مهم .

وكل الى فيه بيقلك متستخدامش ال Unchecked Exceptions .والاسباب الى مخليه المضوع مش كويس او مش مفيد

4- Provide Context with Exceptions:

كل exception بتعمله throw لازم بيقاله context يعنى سياق بيحدد مكان ومصدر ال error ف الجافا ممكن تشوف ال stack trace .بس برضو ال stack trace مش هيقدر يقلك غرض العمليه الى فشلت.ف الاحسن تعمل error massage ومعبره وبعتهم مع exception وحدد المشكله كانت ايه وايه نوعها . ولو بتعمل logging ابعت معلومات كافيه عشان تقدر تعمل logging ف ال catch .

5- Define Exception Classes in Terms of a Caller's Needs:

ف طرق كتير لتصنيف ال error ممكن نصنفهم بناء على المصدر يعنى هم من component x اوcomponent y من . او ممكن نصنفهم عن طريق نوعهم هلى هم network filer ولا device frailer . بس الاهم من كل ده ان لما نعمل exception class لازم نعرف ازاى نعملهم catch . لو مش فاهم تاله ناخد مثال .

```
ACMEPort port = new ACMEPort(12);

try {
  port.open();
} catch (DeviceResponseException e) {
  reportPortError(e);
  logger.log("Device response exception", e);
} catch (ATM1212UnlockedException e) {
  reportPortError(e);
  logger.log("Unlock exception", e);
} catch (GMXError e) {
  reportPortError(e);
  logger.log("Device response exception");
} finally {
  …
}
```

- ف الاسكرين دى احنا بنغطى كل ال exception الى ممكن تحصل بس المشكله ان فيه كود متكرر كتير زى
  - reportPortError
  - logger.log
- وموضع التكرار ده حاجه مش مفاجئه يعنى ف اغلب حلات ال exception handling الى بنعمله بيكون متشابه بغض النظر عن السبب الفعلى.
- ف عشان احنا عرفين ان الى بنعمله متسابه بغض النظر عن ال exception نقدر نبسط الكود خاالص عن طريق ان نعمل كلاس ونحط فيه الAcmePort الى ف الصورة ونادم داله ال openالى جوه كلاس AcmePort جوه الكلاس الجديد اللى هنعمله .
- وده شكل الجديد للكود الى فوق ف واضح اكيد هو ازاى مختصر ومنظم طيب السوال هو الكلاس الجديد اللى هنعمله ده شكله ازاى ؟ يعنى كلاس localPort جواه ايه ؟

```
LocalPort port = new LocalPort(12);
try {
  port.open();
} catch (PortDeviceFailure e) {
  reportError(e);
  logger.log(e.getMessage(), e);
} finally {
  …
}
```

- ده شكل الكلاس الجديد الى هنعمله : هيكون فيه object من ACMEPort . وكمان هيكون فيه داله اسمها open نفس اسم الداله اللى ف كلاس ACMEPort ف داله open الجديده هنادم داله open القديمه ونحطها ف try/catch .

```
public class LocalPort {
  private ACMEPort innerPort;

  public LocalPort(int portNumber) {
    innerPort = new ACMEPort(portNumber);
  }

  public void open() {
    try {
      innerPort.open();
    } catch (DeviceResponseException e) {
      throw new PortDeviceFailure(e);
    } catch (ATM1212UnlockedException e) {
      throw new PortDeviceFailure(e);
    } catch (GMXError e) {
```

طيب انا كده استفت ايه ؟

- اول حاجه كده عم بوب عمل كلاس اسمه reportDrviceFailure ده بيعمله throwمع كل ال error الى هتحصل . وممكن تعمل اكتر من كلاس وحسب ال exception هتعمل throw لكلاس معين .

```
public class LocalPort {
  private ACMEPort innerPort;

  public LocalPort(int portNumber) {
    innerPort = new ACMEPort(portNumber);
  }

  public void open() {
    try {
      innerPort.open();
    } catch (DeviceResponseException e) {
      throw new PortDeviceFailure(e);
    } catch (ATM1212UnlockedException e) {
      throw new PortDeviceFailure(e);
    } catch (GMXError e) {
```

- تانى حاجه كده انت خليت ال logic بتاع الكود بتاعك بعيد عن الجزء بتاع ال handling error والكود بقى اصغر واوضح .

6- Define the Normal Flow:

لو انت عملت زى ملقنا ف السكشن الى فاتت ف هتعرف تفصل كويس بين business logic وال error handling والكود هيكون احسن بس العمليه دى . بس ف حلات معينه موضوع انك تعمل exception ده مش اصح حاجه وتعاله ناخد المثال :

- ف الاسكرين الى جايه دى ال exception جزء من ال logic وده مش صح ومش احسن استغلال للexception

```
try {
  MealExpenses expenses = expenseReportDAO.getMeals(employee.getID());
  m_total += expenses.getTotal();
} catch(MealExpensesNotFound e) {
  m_total += getMealPerDiem();
}
```

- زى ما واضح ف الاسكرين لما يحصل exception ال total هيكون فيه قيمه جايه من malePerDiem ولو محصلش exception ف ال total هيكون فيها Get Total .طيب هيحصل exception امتى لو ال male مكنتش `expense` . يعنى كلاس `expenseReportDAO` ممكن يرجع `object` من نوع `expense` وفى الحاله دى هيكون فيه داله `getTotal` فمش هيضرب `exception` وممكن يرجع حاجه مفهاش داله`get Total` .
- معنى كلمه malePerDiem هو بدال الوجبه اليومى

- ف الاحسن من العك اللى ف الاسكرين الى فاتت ان نخلى expenseReportDAO يرجع دايما object من نوع expense وانت اصلا بترجع object حسب ال id ف افرض ان id ده كان ل object من نوع غير ال expense ؟ الحل ان هنرجع object احنا ظبطينه يكون فيه داله getTotal ولما نادى على ال getToale بتاعته يرجع malePerDiem .
والحل ده اسمه special Case pattern وده هيكون شكل الكود الجديد .

```
MealExpenses expenses = expenseReportDAO.getMeals(employee.getID());
m_total += expenses.getTotal();
```

- طيب تعاله اشرحلك ال special Case pattern وده شرحى انا مش عم بوب :
  - معنى special Case pattern ان اعمل كلاس بحيث يهندل special Case وسعتها ال client كود مش هيبقا بيتعامل مع ال special Case دى و special Case هيكون حصلها encapsulate جوه special Case object .
  - وده مثال على special Case pattern : هنفرض ان عندى كلاس اسمه person وفى احتمال ان ال person ده فى مره يرجع فاضى ومره تانى يرجع مثلا ب null ف ما يرجع ب null او فاضى دى special Case . ف عشان نحل ال special Case فهنعمل كلاس اسمه emptyPerson و NullPerson والاتنين دول ممكين يكونوا وارثيين من person وهيكون فيهم ال behaviorالى انا عاوزه لما يرجع person null او empty person .

7- Don't Return Null:

اى نقاش على الاerror handling لازم نذكر الحجات الى بتسبب ال error :

1- اول حاجه هى انك ترجع null دى الصوره دى كده

```
public void registerItem(Item item) {
  if (item != null) {
    ItemRegistry registry = peristentStore.getItemRegistry();
    if (registry != null) {
      Item existing = registry.getItem(item.getID());
      if (existing.getBillingPeriod().hasRetailOwner()) {
        existing.register(item);
      }
    }
  }
}
```

- ولو انت فاكر ان الاكواد الى شبه الاسكرين الى فاتت حلوه او كويسه "وقصدى بالاكواد الى شبه الاسكرين الى فاتت ان كود بيعمل check على null " تبقا غلطان والسبب ان لو نسيت تعمل checkواحده بس ف الكود هيضرب .
- واصلا الكود الى فات دة ممكن يضرب عشان مفيش check على حوار ال null على الى راجع من تانى سطر

```
public void registerItem(Item item) {
  if (item != null) {
    ItemRegistry registry = peristentStore.getItemRegistry();
    if (registry != null) {
      Item existing = registry.getItem(item.getID());
      if (existing.getBillingPeriod().hasRetailOwner()) {
        existing.register(item);
      }
    }
  }
}
```

- ف الكود ده مشكلته انه ناقص null check بس المشكله الحقيقه ان فيه null check كتير . ولو انت هترجع null من function ف الاحسن انك ترجع special case object زى الى شرحته او انك تعمل throw for exception . يعنى عم بوب بيقلك ابعد عن انك تعمل function بترجع null وعمل زى الاسكرين الى جايه بس الاسكرين الى جايه على مثال غير الى فات

- ده المثال الجديد الى هنشتغل عليه هنشوف المشكله الاول وبعد كده هنحلها بستخدام special case object :

وهنا كنا خيفين ال getEmployeesترجعلنا null ف عشان كدة عملنا nullCheck وقلنا ان دة مش احسن حل تعاله نشوف الحل الاحسن

```
List<Employee> employees = getEmployees();
if (employees != null) {
  for(Employee e : employees) {
    totalPay += e.getPay();
  }
}
```

- ف هنخلى ال getEmployees ترجع empty List وده يعتبر ال special case object ف لما يرجع empty List ف كدة ال loop مش هتشتغل ف مش هيحصل exception ومش هنحتاج ال nullechck الى كانت موجوده ف الاسكرين اللى فاتت .

```
List<Employee> employees = getEmployees();
for(Employee e : employees) {
  totalPay += e.getPay();
}
```

- ف لو بتسئل نفسك ازاى هخلى ال getEmployees ترجع empty list ف سهله كدة والكود ده جافا .

```
public List<Employee> getEmployees() {
  if( .. there are no employees .. )
    return Collections.emptyList();
}
```

8- Don't Pass Null:

ف السكشن ده بيقلك لو انت بترجع null من function ودى حاجه وحشه ف انك تعمل pass ل null دى حاجه اوحش بكتير الا لو كنت بتكلم api وهو متوقع كده .بس برضو لازم تقلل حوار انك تبعت null

وتعاله ناخد مثال زى ما تعودنا يحب .

- ف الكود الى جاى ده لو حد بعتله null اكيد هيضرب InvalidArgumentException

```
public class MetricsCalculator
{
  public double xProjection(Point p1, Point p2) {
    return (p2.x - p1.x) * 1.5;
  }
  …
}
```

- لو مش فاهم يعنى ايه بعت null ف شوف الاسكرين دى :

```
calculator.xProjection(null, new Point(12, 13));
```

- طيب ايه الحل لو حد بعت null ؟احنا قلنا السكشن الى فات حل التعامل مع ال null لما special case object او نرمى exception . ف المثال ده هنجرب نرمى exception .

```
public double xProjection(Point p1, Point p2) {
  if (p1 == null || p2 == null) {
    throw InvalidArgumentException(
      "Invalid argument for MetricsCalculator.xProjection");
  }
  return (p2.x - p1.x) * 1.5;
}
}
```

- ف حاجه هى شبه انك تعمل null check بس اوضح ف الكتابه واسهل وهى ال assertion ف شكل الكود هيبقا كده

```
public class MetricsCalculator
{
  public double xProjection(Point p1, Point p2) {
    assert p1 != null : "p1 should not be null";
    assert p2 != null : "p2 should not be null";
    return (p2.x - p1.x) * 1.5;
  }
}
```

- لو مكنتش فاكر ال assert ف هيا لو الشرط الى بعديها طلع غلط بترمى exception من نواع AssertionError

وده شكل العام ليها ودى ف لغه الجافا

```
assert expression1 : expression2;
```

وف العموما ف معظم لغات البرمجه مفيش طريقه معينه للتعامل مع ال null الى بيتبعت لfunction ف لما تكون بتكتب كود وانت عارف ان ممكن تسبب مشكله ف اكيد اخطاءك هتبقا قليله وهتعرف تهندل الموضوع .

- ف خلاصه السكشن متبعت null لفانكشن او مترجعش null من فانكشن والحل
  1- لما ارمى اكسبشن
  2- استخدام ال special case class

9- Conclusion:

اكتر سكشن بحبه الزتونه :

ال clean code هو حاجه readable وف ناس الوقت لازم تبقا حاجه قويه ودول هدفين مش عكس بعض.

فهتقدر تكتب كود clean وقوى لما تعامل ال error handling على انه Concept منفصل عن ال business logic بتاع الكود .وكل متقدر تفكر بشكل مستقل عن الاتنين دول هتقدر تحافظ عل الكود clean .