FACULTY OF INFORMATICS

# Static analysis with Infer

Software Analysis - Assignment 2

Authors FEDERICO
LAGRASTA

April 13, 2022

# 1 Project selection

Games developed without external frameworks have usually an interesting codebase to analyze. As having a wider variety of issues compared to most other projects (ex: a traditional CVM REST API ) they feature an equivalent amount of creative workarounds.
The first project we considered for this assignment was one we had already worked on to refactor https://github.com/OrangoMango/Scapoom an editor to create labyrinth games. The reason we decided against it was that it didn't feature a standardized way to be built (requiring to download JFX and then to write its location on the build file). In addition to that its code quality was a bit too close to the lower end triggering potential errors pretty much everywhere.
With this consideration we veered towards the already tested projects:

**https://github.com/xtaci/algorithms** A collection of algorithms. Algorithms are usually pretty short and don't really have a chance to exploit design patterns. In order to be efficient they are not usually elegantly written and for this reason could trigger many false positives. We didn't judge the code here to be that interesting to analyze.

**https://github.com/Rogiel/torrent4j** No docs whatsoever D:

**https://github.com/Kaysoro/KaellyBot** A discord bot. Infer took over 20m to run on this project and roughly half of Infer positives were related to concurrency which we will potentially dedicate a whole assignment to later on.

**https://github.com/trekawek/coffee-gb** Gameboy Color emulator . Pretty interesting project but all Infer positives were concurrency related...

**https://github.com/Progether/JAdventure** A text based game. All Infer positives were `NullDereference` but on the other hand `SonarQ` reported more.

# 2 Issues

We run the provided image specifying the target output platform as `linux/amd64` since many libraries tend to have issues with the `arm64` achitecture of the new macbooks.
Still running the Infer analyser on projects would result in the following error:

```
Uncaught Internal Error: (Unix.Unix_error "No child
   processes" waitpid "((mode (WNOHANG)) (pid -1))")
```

To solve this issue we forced the analysis to run on a single thread with the option `-j 1` which apparently solved the issue but likely made Infer much slower.

```
docker run --name software_analysis --platform linux/amd64 -it -v \
($PWD):/HWS bugcounting/satools
```

# 3 The project

As per the project description
*JAdventure is a Java (text) based implementation of a Role Playing Game (RPG) - Single Player.*
The code is documented, although not following strict guidelines, pretty decently. There's even a `/documentation` folder describing its architecture automatically making it better than roughly 94.57% of projects.
The project makes use of simple design patterns like `facade` (as a work in progress to split logic and presentation layers from what I can see) and `factory`. It adhers to simple principles like clustering information with classes like `Coordinate`.
At a glance there's seem to be some abuses of techniques like reflection to represent the avalable commands of the games all in one class. The parser for the commands doesn't interpret a DSL and thus loses genericity forcing a lot of information to be hardcoded while some other is properly encoded in `json` files that are turned into repositories. The classes managing these commands could potentially devolve into god-classes were the size of the project to increase.
In general the code has many "quircky" sections like the following code used to unquip to items from a character:

```
private void unequipTwoPlaces(EquipmentLocation
   leftLocation, EquipmentLocation rightLocation) {
    Item left = equipment.get(leftLocation);
    Item right = equipment.get(rightLocation);
    if (left != null) {
        unequipItem(left);
    }
    if (right != null) {
        unequipItem(right);
    }
}
```

why not just overload `unequipItem` to take `List<Item>` instead?
The presence of too many `instance` is also a signal of underlying responsability subversion.

# 4 Infer analysis

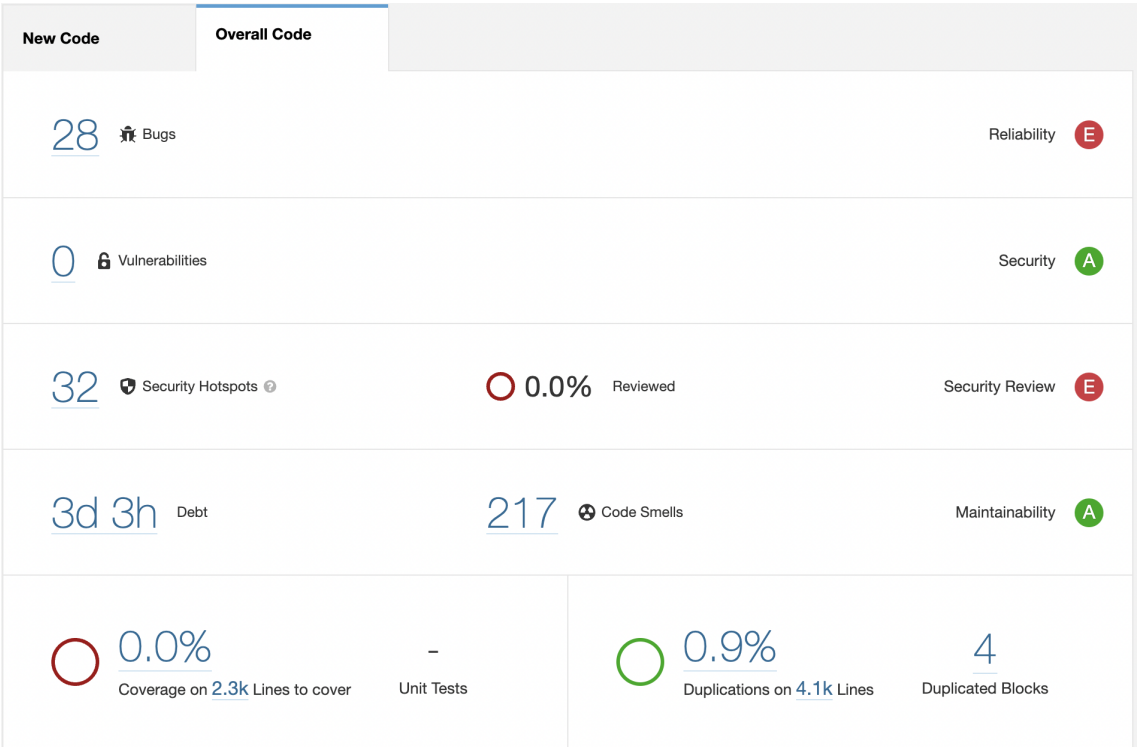
Running infer

```
infer run -j 1 --default-checkers -- mvn clean compile
```

on the project produces 10 issues all of type `NULL_DEREFERENCE`.

# 5 Sonaqube analysis

The following is an overview of the results produced by Sonarqube:




(Following section taken from a previous project)
There are three types of issues:

**Bug** A coding mistake that can lead to an error or unexpected behavior at runtime.

**Vulnerability** A point in your code that's open to attack.

**Code Smell** A maintainability issue that makes your code confusing and difficult to maintain.

Each issue has one of five severities:

**BLOCKER** Bug with a high probability to impact the behavior of the application in production: memory leak, unclosed JDBC connection, .... The code MUST be fixed immediately.

**CRITICAL** Either a bug with a low probability to impact the behavior of the application in production or an issue which represents a security flaw: empty catch block, SQL injection, ... The code MUST be immediately reviewed.

**MAJOR** Quality flaw which can highly impact the developer productivity: uncovered piece of code, duplicated blocks, unused parameters, ...

**MINOR** Quality flaw which can slightly impact the developer productivity: lines should not be too long, "switch" statements should have at least 3 cases, ...

**INFO** Neither a bug nor a quality flaw, just a finding.

Here we report all the BLOCKER issues found, all the CRITICAL issues found and the MAJOR issues relative to null pointer dereference.

| Rule | Type | Matches |
|---|---|---|
| Loops should not be infinite | Bug | 1 |
| Child class fields should not shadow parent class fields | Smell | 2 |
| String literals should not be duplicated | Smell | 21 |
| Cognitive Complexity of methods should not be too high | Smell | 9 |
| "Random" objects should be reused | Bug | 8 |
| "switch" statements should have "default" clauses | Smell | 5 |
| Instance methods should not write to "static" fields | Smell | 3 |
| "String#replace" should be preferred to "String#replaceAll" | Smell | 2 |
| "static" base class members should not be accessed via derived types | Smell | 2 |
| Try-with-resources should be used | Smell | 2 |
| Assertions comparing incompatible types should not be made | Bug | 1 |
| Methods should not be empty | Smell | 1 |
| Null pointers should not be dereferenced | Bug | 3 |

# 6 Infer vs Sonarqube

Infer seems to be focused on resolving a number of issues with its checkers specifically about potential errors while Sonarqube covers a larger scope that includes code smells and antipatterns beside pure potential bugs.
Annotations could potentially make infer more useful than it is, for example the following code would trigger a `CHECKERS_CALLS_EXPENSIVE_METHOD`

```
class C {
  @PerformanceCritical
  void perfCritical() {
    expensive();
  }

  @Expensive
  void expensive() {}
}
```

Regarding potential null dereferences infer finds 10 while Sonarqube only 3. Of the 10 results from infer all are true positives while of the 3 of sonarqube only 1 is a true positive.
Overall Sonarqube seems to be a far better tool but Infer could still retain a niche given its higher recall on its checkers.
The following possible dereference is missed by Infer but not Sonarqube: as `new ServerSocket(port);` could throw an `IoException` thus `listener.close();` would be a dereference.

```
while (true) {
    ServerSocket listener = null;
```

```
    try {
        listener = new ServerSocket(port);
        while (true) {
            Socket server = listener.accept();
            Runnable r = new MainMenu(server, mode);
            new Thread(r).start();
        }
    } catch (IOException c) {
        c.printStackTrace();
    } finally {
        try {
                        listener.close();
                } catch (IOException e) {
                        e.printStackTrace();
                }
    }
```

# 7 Fixing the code

## 7.1

```
#0
src/main/java/com/jadventure/game/prompts/
   BackpackDebugPrompt.java:42: error: Null Dereference
  object `appendItem` last assigned on line 41 could be
     null and is dereferenced at line 42.

  40.       try {
  41.           Item appendItem = itemRepo.getItem(
     command.substring(3).trim());
  42. >         if (appendItem.getName() != null)
  43.               player.addItemToStorage(appendItem);
  44.       } catch (RepositoryException ex) {
```

Reasonably one would think that the `itemRepo` would always throw a `RepositoryException` when it can't provide an object instead it only does so when the given item id is non `null` and has length greater than 0 as we can see in the following code:

```
public Item getItem(String id) {
    if (id == null || id.trim().length() == 0) {
        return null;
    }
        if (!itemMap.containsKey(id)) {
           throw new RepositoryException("Argument 'id'
              with value '" + id + "' not found in
              repository.");
         }
```

```
        return itemMap.get(id);
}
```

The proposed solution is to just throw in any of the three cases instead of returning `null`.
The method has 39 usages through the code and each should be evaluated in a potential refactor.
This is a true positive for Infer and a false negative for Sonarqube.

## 7.2

```
#1
src/main/java/com/jadventure/game/Game.java:51: error:
   Null Dereference
  object `locationRepo` last assigned on line 50 could be
      null and is dereferenced at line 51.
  49.           player.setName(userInput);
  50.           LocationRepository locationRepo =
     GameBeans.getLocationRepository(player.getName());
  51. >         this.player.setLocation(locationRepo.
     getInitialLocation());
  52.           player.save();
  53.           QueueProvider.offer("Welcome to Silliya,
     " + player.getName() + ".");
```

If we look down in the code we find the code instanciating the `LocationRepository` singleton:

```
public static LocationRepository createRepo(String
   profileName) {
    if ("".equals(profileName)) {
        return instance;
    }
    if (instance == null) {
        instance = new LocationRepository(profileName);
    } else if (!instance.getFileName().contains(
       profileName)) {
        instance = new LocationRepository(profileName);
    }
    return instance;
}
```

Since the `instance` field is `null` at the start and this code returns it when the name is empty this is a true positive for infer and a false positive for sonarqube. Also what happens when `profileName` is `null`?
The proposed solution is to throw an `InvalidArgumentException` when `profileName` is empty and a `NullPointerException` when it's `null`.

## 7.3

```
#2
src/main/java/com/jadventure/game/entities/Player.java
   :92: error: Null Dereference
  object returned by `com.jadventure.game.entities.Player
     .characterLevels.get(characterType)` could be null
     and is dereferenced at line 92.
  90.
  91.       public int getCharacterLevel(String
     characterType) {
  92. >         int characterLevel = this.characterLevels
     .get(characterType);
  93.           return characterLevel;
  94.       }
```

There are 2 possible `null` dereferences here: one because an `HashMap` in java could contain null values (why???) thus dereferencing null to `int`. The second because the map itself could be null given the following setter:

```
public void setCharacterLevels(HashMap<String, Integer>
   newCharacterLevels) {
    this.characterLevels = newCharacterLevels;
}
```

which at any rate should do a copy of the given map...
This is a true positive for infer and false negative for sonarqube.
The proposed solution is to throw when there's an attempt to set the map to null or when the map returns null on a key. An improvement would be to use a map that does not allow null values and keys.

## 7.4

```
#3
src/main/java/com/jadventure/game/conversation/
   ConversationManager.java:103: error: Null Dereference
  object returned by `line.get("player")` could be null
     and is dereferenced at line 103.
  101.                }
  102.            }
  103. >          String playerPrompt = line.get("player")
     .getAsString();
  104.            String text = line.get("text").
     getAsString();
  105.            String[] con = line.get("condition").
     getAsString().split("=");
```

The conversations for the game are stored in a .Json file but never validated. We can add an exception here in case on of the required properties is missing but in general there should be a validation step on read before the game starts.
This is a true positive for Infer and a false negative for Sonarqube.

## 7.5

```
#4
src/main/java/com/jadventure/game/Trading.java:116: error
   : Null Dereference
  object returned by `itemValues.get(itemIds.get(itemName
     ))` could be null and is dereferenced at line 116.
  114.
  115.            if (itemIds.containsKey(itemName)) {
  116. >              int itemValue = itemValues.get(
     itemIds.get(itemName));
  117.               Item item = itemIdtoItem.get(itemIds
     .get(itemName));
  118.
```

Here we can throw an `IllegalArgumentException` if the value of the item is not known.
This is a true positive for Infer and a false negative for Sonarqube.

## 7.6

```
#5
src/main/java/com/jadventure/game/conversation/
   ConversationManager.java:172: error: Null Dereference
  object `requiredItem` last assigned on line 171 could
     be null and is dereferenced by call to `hasItem(...)
     ` at line 172.
  170.                    ItemRepository itemRepo =
     GameBeans.getItemRepository();
  171.                    Item requiredItem = itemRepo.
     getItem(line.getConditionParameter());
  172. >                  return player.hasItem(
     requiredItem);
  173.                case CHAR_TYPE:
  174.                    String charType = line.
     getConditionParameter();
```

Here we have to look at the code of Player#hasitem() to confirm the potential dereference:

```
    public boolean hasItem(Item item) {
        List<Item> searchEquipment = searchEquipment(item
           .getName(), getEquipment());
        List<Item> searchStorage = searchItem(item.
           getName(), getStorage());
        return !(searchEquipment.size() == 0 &&
           searchStorage.size() == 0);
    }
```

an exception could be thrown when the item is null or false should be returned..
This is a true positive for Infer and a false negative for Sonarqube.

## 7.7

```
#6
src/main/java/com/jadventure/game/entities/Player.java
   :245: error: Null Dereference
  object `weapon` last assigned on line 244 could be null
      and is dereferenced at line 245.
  243.        public void getStats(){
  244.            Item weapon = itemRepo.getItem(getWeapon
     ());
  245. >          String weaponName = weapon.getName();
  246.            if (weaponName.equals(null)) {
  247.                weaponName = "hands";
```

...uh
yeah for once that should be `==` instead of `.equals`.
In Sonarqube this gets listed under the rule: `Silly equality checks should not be made`.
The null dereference is at the previous line tho, it's solved by throwing in the item repository when an item is not present.
This is a true positive for Infer and a false negative for Sonarqube.

## 7.8

```
#7
src/main/java/com/jadventure/game/prompts/
   CommandCollection.java:290: error: Null Dereference
  object `locationRepo` last assigned on line 289 could
     be null and is dereferenced at line 290.
  288.        public void command_teleport(String arg) {
  289.            LocationRepository locationRepo =
     GameBeans.getLocationRepository(player.getName());
  290. >          ILocation newLocation = locationRepo.
     getLocation(new Coordinate(arg));
  291.            ILocation oldLocation = player.
     getLocation();
  292.            try {
```

Here we have a similar problem as before as the repository can return a null value.
We can solve it by throwing when coordinates are null.
This is a true positive for Infer and a false negative for Sonarqube.

## 7.9

```
#8
src/main/java/com/jadventure/game/entities/Entity.java
   :317: error: Null Dereference
  object `hands` last assigned on line 304 could be null
     and is dereferenced at line 317.
  315.                locations.put(EquipmentLocation.FEET
     , "Feet");
  316.                for (Map.Entry<EquipmentLocation,
     Item> item : equipment.entrySet()) {
  317. >                  if (item.getKey() != null && !
     hands.equals(item.getValue()) && item.getValue() !=
     null) {
  318.                         QueueProvider.offer(
     locations.get(item.getKey()) + " - " + item.getValue
     ().getName());
  319.                    } else {
```

This is a true positive for Infer and a false negative for Sonarqube. It should be solved by the fact that we now throw in the repository when a requested item is not present to correct another potential dereference.

### 7.10

```
#9
src/main/java/com/jadventure/game/entities/Player.java
   :376: error: Null Dereference
  object `weapon` last assigned on line 375 could be null
      and is dereferenced at line 376.
  374.                Item itemToDrop = itemRepo.getItem(
     item.getId());
  375.                Item weapon = itemRepo.getItem(
     getWeapon());
  376. >              String wName = weapon.getName();
  377.
  378.                if (itemName.equals(wName)) {

Found 10 issues
          Issue Type(ISSUED_TYPE_ID): #
  Null Dereference(NULL_DEREFERENCE): 10
```

This should be solved by the previous correction again.

## 8 After

There is one possible dereference found by Infer after the previous changes:

```
src/main/java/com/jadventure/game/Trading.java:119: error
   : Null Dereference
```

```
object returned by `itemValues.get(itemIds.get(itemName
   ))` could be null and is dereferenced at line 119.
117.                     throw new
   IllegalArgumentException("Unknown item value");
118.                 }
119. >               int itemValue = itemValues.get(
   itemIds.get(itemName));
120.                 Item item = itemIdtoItem.get(itemIds
   .get(itemName));
121.
```

this is a false positive since an exception will be thrown.

# 9 The scourge of `null`

Null references have been defined by Tony Hoare[1] as "the billion dollar mistake" and for a good reason.
With `null` being an option the question is where to we check for it? If we check every possible (even if technically unfeasible) dereference of it the code becomes incredibly bloated while if we don't we are bound to use it in a case where a dereference is actually feasible. In general manually checking if a dereference is possible at a point in the code becomes increasibly difficult when the code presents a high level of nesting.
More modern languages like Kotlin are statically null-safe but with Java what's the best practice to deal with it? Methods should guarantee returning a non `null` value. Methods that can return `null` should instead wrap it in an `Optional` to force the calling method to handle it. We can further use the null object pattern [2] to further cull `null`.
In the case of the analyzed code in most cases a null dereference would not have been feasible assuming the game data stored in the .Json files was properly formed but this is not really an assumption that makes sense.
In other cases an empty string being received in input would have crashed the whole game assuming it wasn't filtered before reaching the game engine. Again since there is even an explicit attempt to separate the game engine from the presentation layer this is not a reasonable assumption.
Correcting single instances of possible dereferences doesn't make much sense what instead should be done is a change in how `null` is handled as previously described at least in a high-level language such as Java.

[1] https://www.infoq.com/presentations/Null-References-The-Billion-Dollar-Mistake-Tony-Hoare/

[2] https://en.wikipedia.org/wiki/Null_object_pattern