IDT Testing API

bu_winter2014

Alexander Meijer and Christopher Rung

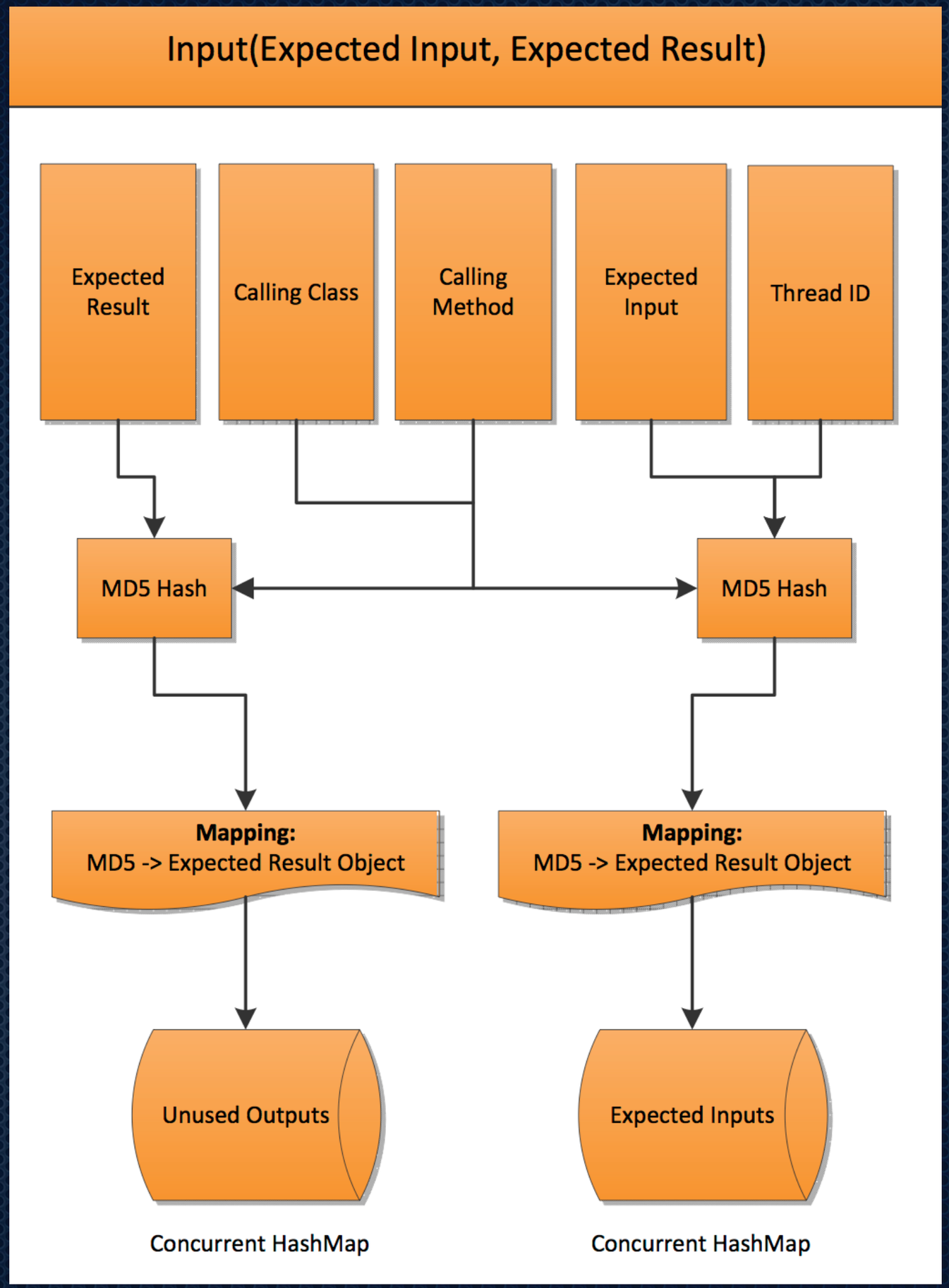
Our API (example usage)

```
public boolean isEven(int numToCheck) throws IOException, UnIdentifiableException {

    Tester.INSTANCE.input(2, true);
    Tester.INSTANCE.input(3, false);
    Tester.INSTANCE.input(4, true);

    Tester.INSTANCE.load(numToCheck);

    // divide the number by 2 and no remainder exists, the number is even
    if (numToCheck % 2 == 0) {
        Tester.INSTANCE.log(true);
        return true;
    } else {
        Tester.INSTANCE.log(false);
        return false;
    }
}
```

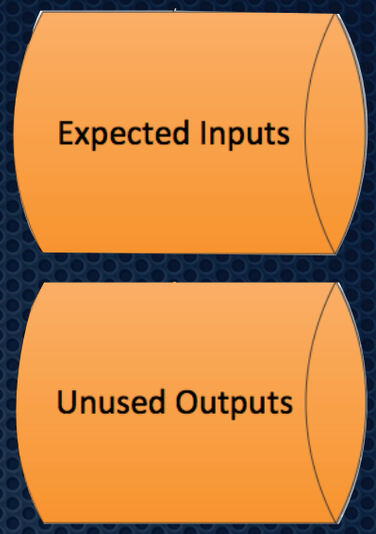
Stores expected input-output relationship

Loads runtime input

Logs runtime output

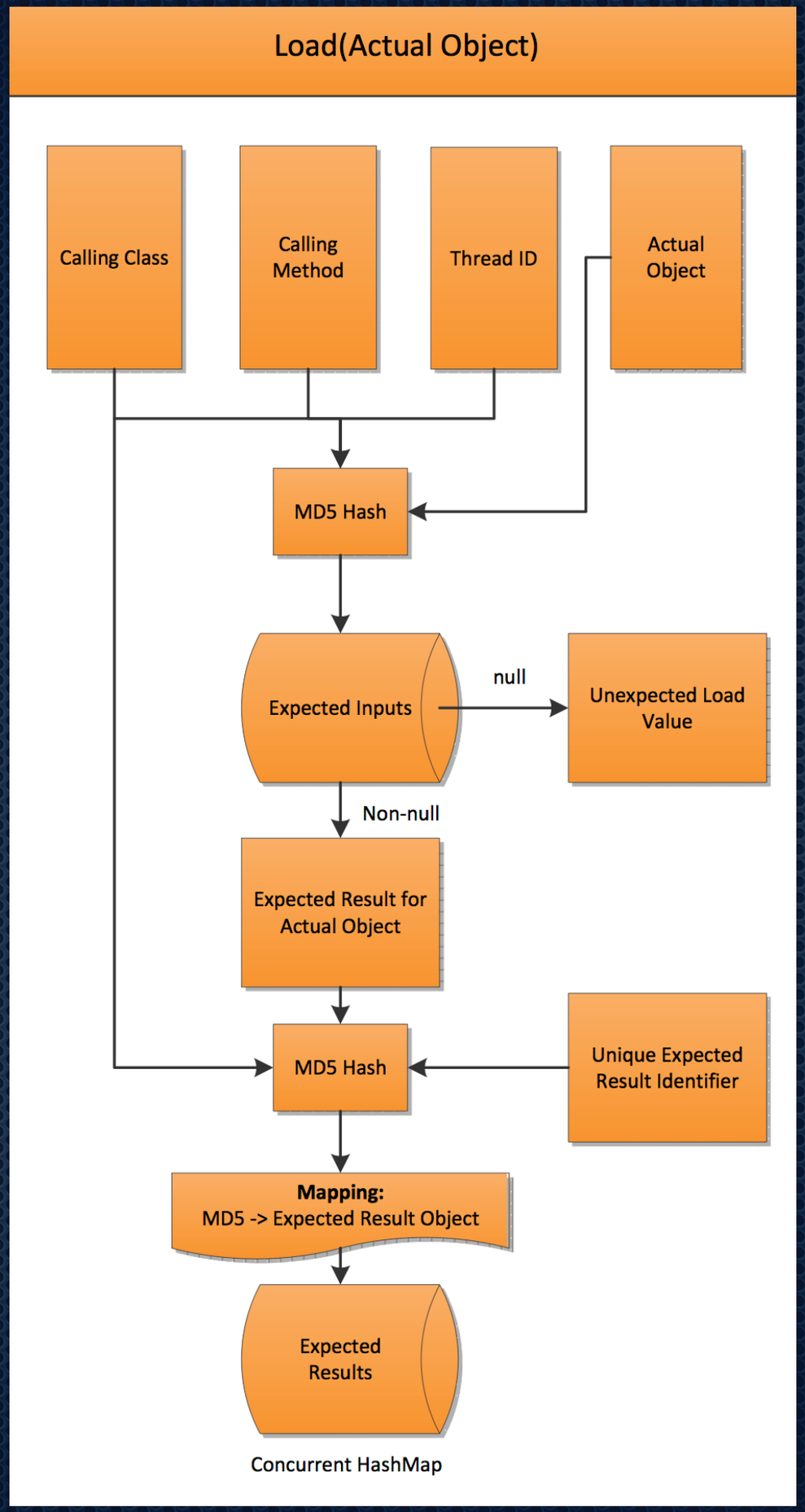

Input(Expected Input, Expected Result)
Expected Result
Calling Class
Calling Method
Expected Input
Thread ID
MD5 Hash
MD5 Hash
Mapping:
MD5 -> Expected Result Object
Mapping:
MD5 -> Expected Result Object
Unused Outputs
Expected Inputs
Concurrent HashMap
Concurrent HashMap

Meet the Data Structures:

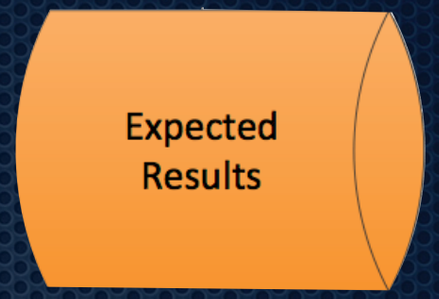


- Expected inputs:
  - Stores all expected inputs and the outputs that they correspond to.
  - Exists in code as `input_expected`
    - `ConcurrentHashMap` object

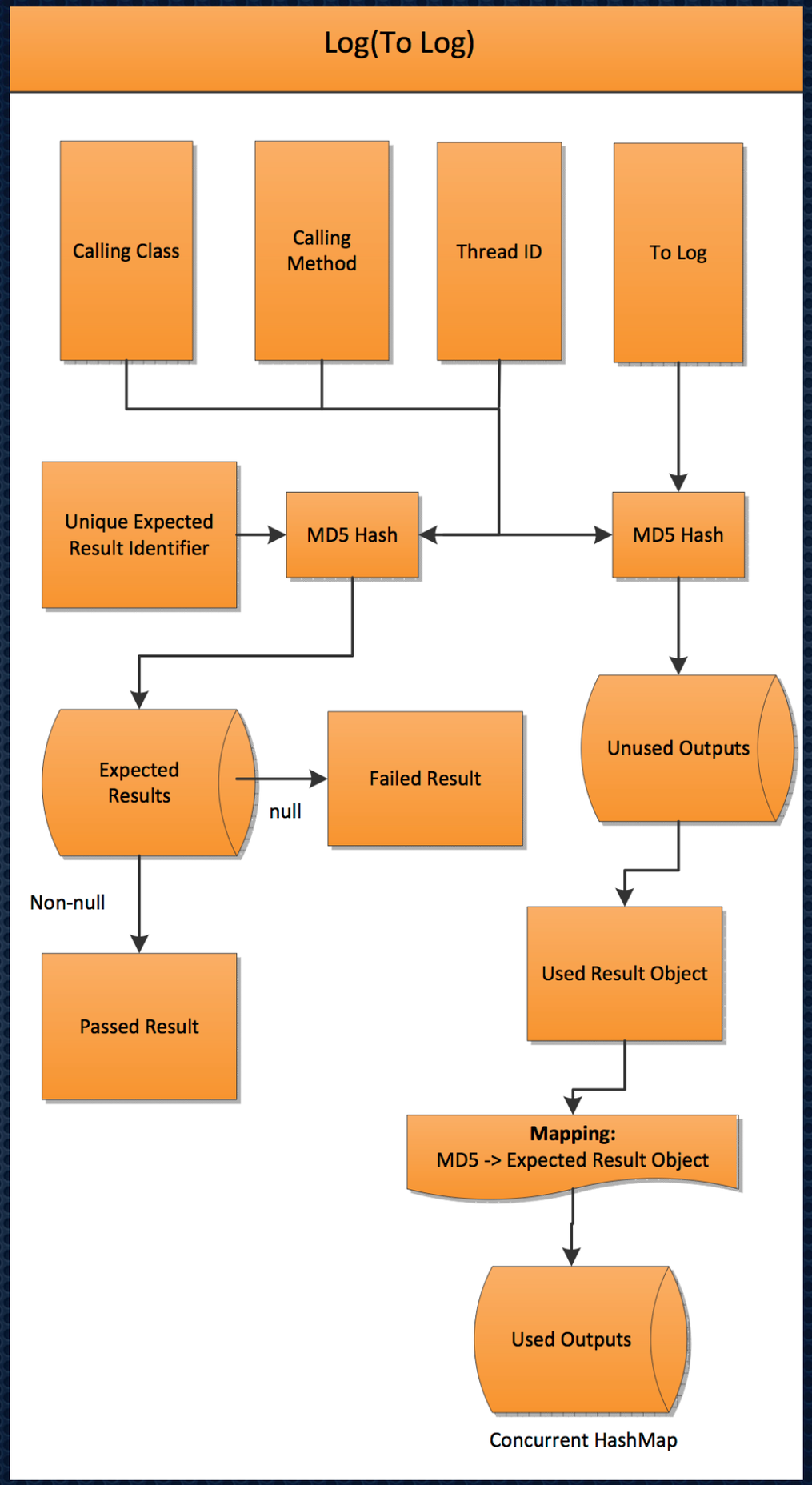
- Unused outputs:
  - Stores all outputs that have not yet been logged by the Tester
  - Exists in code as `notUsed`
    - `ConcurrentHashMap` object


Load(Actual Object)
Calling Class
Calling Method
Thread ID
Actual Object
MD5 Hash
Expected Inputs
null
Unexpected Load Value
Non-null
Expected Result for Actual Object
MD5 Hash
Unique Expected Result Identifier
Mapping:
MD5 -> Expected Result Object
Expected Results
Concurrent HashMap

Meet the Data Structures:



- Expected Results:
  - Stores the Results to be logged
    - Matching inputs have been loaded
  - Exists in code as `expectedHash_inObject`
    - `ConcurrentHashMap` object


Log(To Log)
Calling Class
Calling Method
Thread ID
To Log
Unique Expected Result Identifier
MD5 Hash
MD5 Hash
Expected Results
null
Failed Result
Unused Outputs
Non-null
Passed Result
Used Result Object
Mapping:
MD5 -> Expected Result Object
Used Outputs
Concurrent HashMap

Meet the Data Structures:

- Used Outputs:
  - Contains all expected outputs that have been seen at runtime
  - Exists in code as `used`
    - `ConcurrentHashMap` object

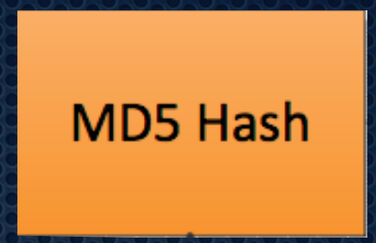
ConcurrentHashMap

- O(1) search running time
  - This allows great scalability
- Not just synchronized; is concurrent
  - multiple threads can access map at the same time
    - leads to performance increase

`ConcurrentHashMap` Drawbacks

- In `ConcurrentHashMap`s, sections are locked by threads
- If threads attempt to access similar keys, it is still possible for one thread to block another
- How do we reduce the probability of this happening?

MD5



- Inherently unstable
  - Small variance in input creates very different outputs
  - Similar objects hash to wildly different values
    - Threads are less likely to block each other, even if similar objects are input
- Popularly used as a checksum
- We use MD5 so heavily that we switched to a fast MD5 library - our only external library
  - The library we use is ~48x faster than Java's default
  - Runs native code (on supported OSs)

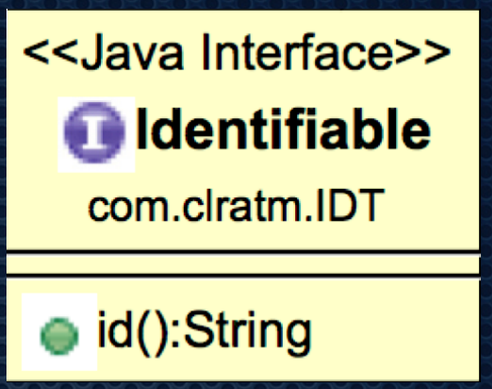
Demo: TesterHealthCheck

The “Multi-Singleton”

- Singletons are beneficial, but only one can exist by definition
- Using Java enums, we are able to provide multiple singleton Tester Instances
  - Hence the term “multi-singleton”
- Our Tester ships with up to 6 instances, but this can be increased arbitrarily

```
Tester.INSTANCE.enable(true);
Tester.INSTANCE_1.enable(true);
Tester.INSTANCE_2.enable(true);

Tester.INSTANCE.input(_byte, _byte);
Tester.INSTANCE_1.input(_str, _str);
Tester.INSTANCE_2.input(_ints, _ints);
```

The `Identifiable` Interface



- Tester will refuse any object that it cannot reliably identify
  - Throws `UnIdentifiableException`
- The `Identifiable` interface permits our Tester to be used by any object that implements it
- Designed to require minimal effort to incorporate into code under test

Demo: Identifiable Interface

API Contraints

- Can’t handle multiple values
  - Overcome by creating array of values

```
Tester.INSTANCE.input(new Object[] { (byte) 5, (int) 1, (boolean) true }, "10");
Tester.INSTANCE.load(new Object[] { (byte) b, (int) placesToShift, (boolean) left });
```

- Repeated calls to `load()` function may result in nondeterministic behavior

Challenges

- Thread safety
  - Limited experience with thread safety in a project

- Runtime-aware computing
  - Used knowledge of calling class and function combined with parameters

Takeaways

- Refactoring
  - Eliminated hundreds of lines of code
  - Eliminated tens of unnecessary methods
  - Critical for long-term projects

- Git was instrumental during development
  - We were in different states while working during 80% of development
  - Git’s branching feature was used to develop new features away from ‘production’ code

Questions?