Testing Object Oriented Software 2010

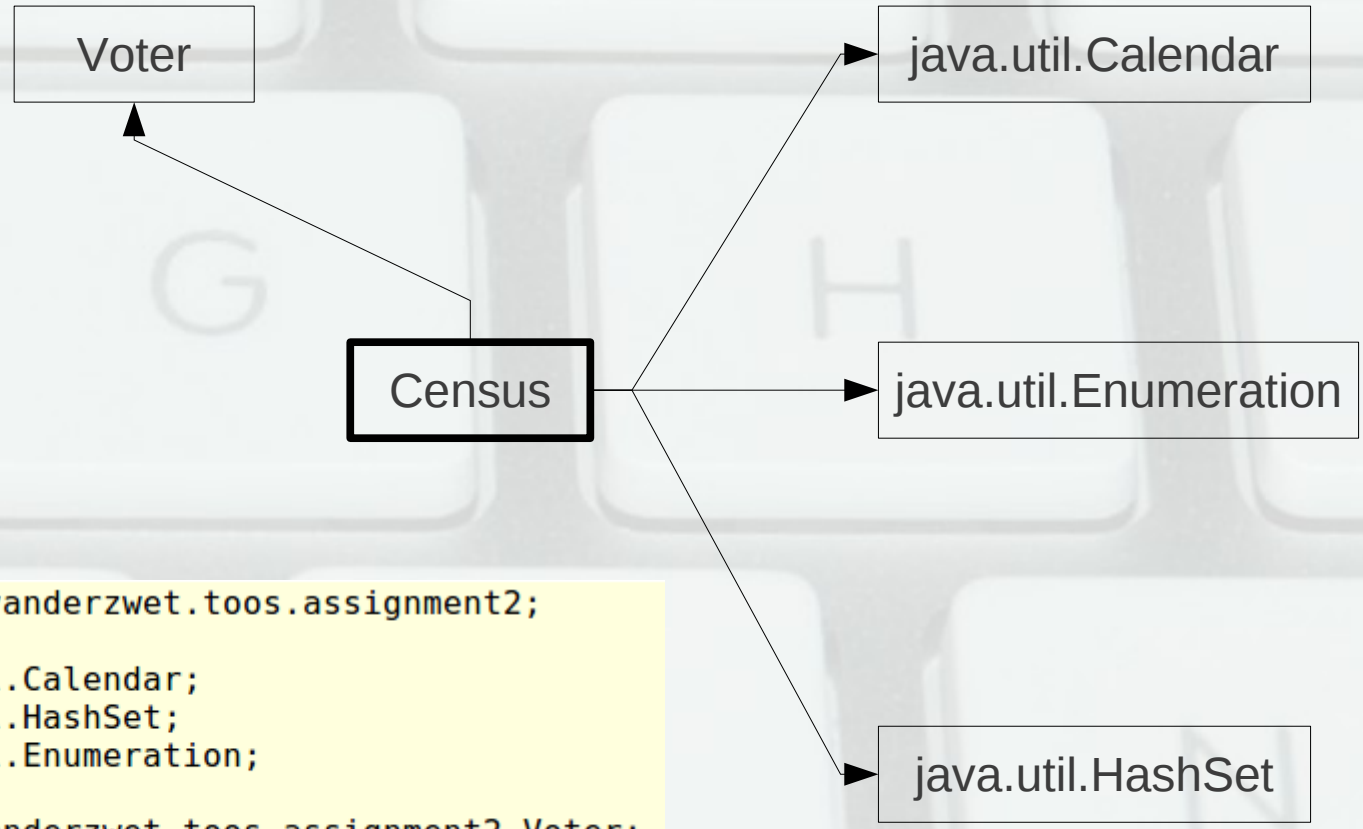
LIACS - Leiden University – TOOS2010 - Rick van der Zwet <hvdzwet@liacs.nl>

- Goal: Design method to automate of Object Oriented software given a Sequence Diagram (*SD*).
- Implementation: (Automatically) Write a wrapper around the CUT to test for properties being accessed. (WrapTest).
- Issues UML input needs to be consistent not left to human interpretation

The Idea

- On creation wrapper around all classes to test for function calls.
- Include special functions to test for variable access.
- Use input space partitioning to generate all possible inputs.

FunctionWrapper

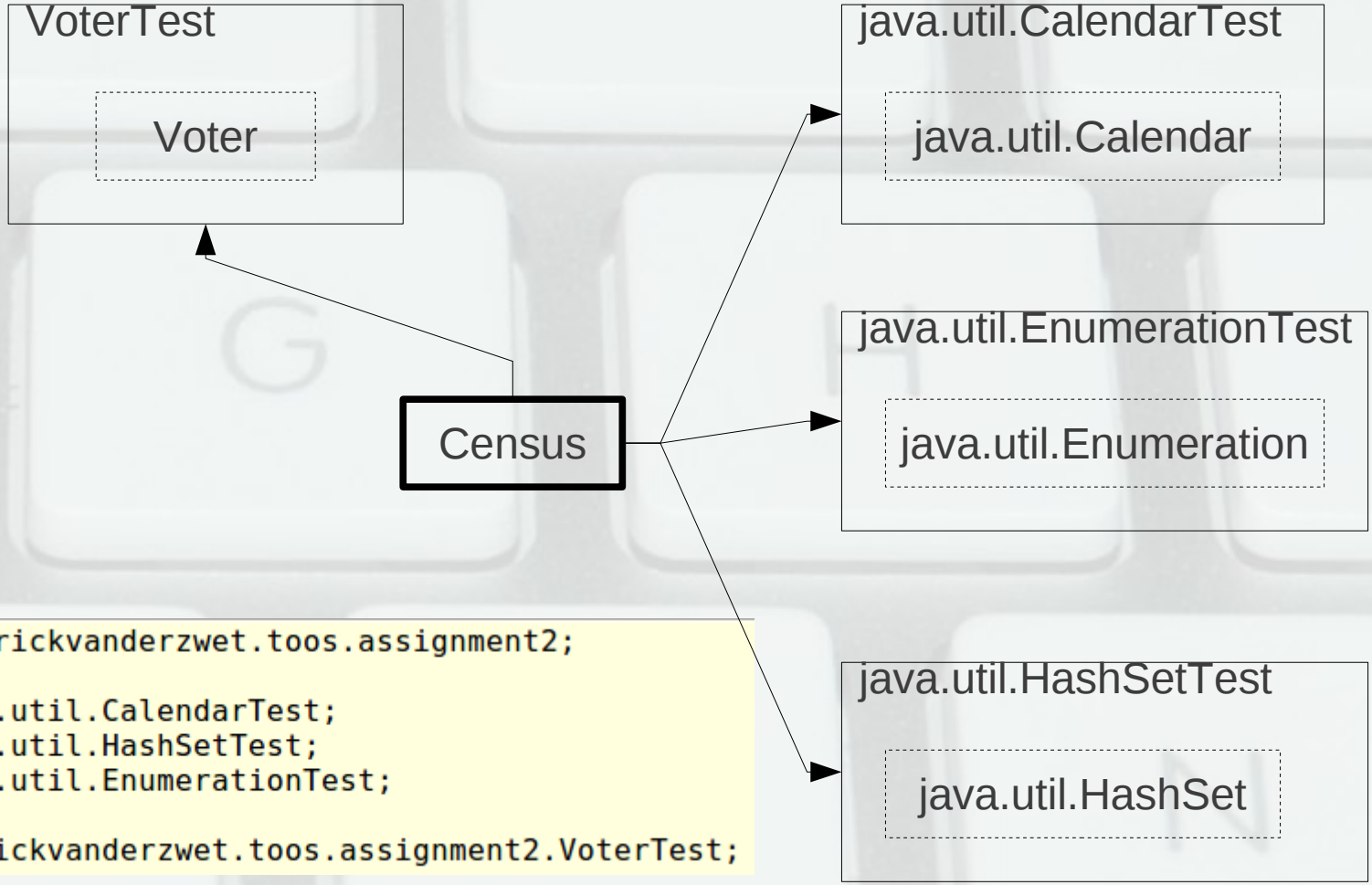



```
package nl.rickvanderzwet.toos.assignment2;

import java.util.Calendar;
import java.util.HashSet;
import java.util.Enumeration;

import nl.rickvanderzwet.toos.assignment2.Voter;
```

FunctionWrapper - continued




```
package nl.rickvanderzwet.toos.assignment2;

import java.util.CalendarTest;
import java.util.HashSetTest;
import java.util.EnumerationTest;

import nl.rickvanderzwet.toos.assignment2.VoterTest;
```

FunctionWrapper - example

```
package nl.rickvanderzwet.toos.assignment2;

public class Voter {
  public Boolean getVote() {
    return fvote;
  }
```

Extend and a public counter will do the trick

```
package nl.rickvanderzwet.toos.assignment2;

public class VoterWrapTest extends Voter {
  public getVote_counter = 0;

  public Boolean getVote() {
    getVote_counter++;
    return super.getVote();
  }
```

The Idea

- On creation wrapper around all classes to test for function calls.
- Include special functions to test for variable access.
- Use input space partitioning to generate all possible inputs.

VariableTest

```
package nl.rickvanderzwet.toos.assignment2;

public class Voter {
  public Boolean fvote;
```

```
package nl.rickvanderzwet.toos.assignment2;

public class Voter {
  public Boolean fvote;

  public Boolean fvoteWrapTest()  {
    return fvote;
  }
```

- Add special function to return the public variable.
- Most preferred you want to 'hide' all your public variables behind interfaces.

The Idea

- On creation wrapper around all classes to test for function calls.
- Include special functions to test for variable access.
- Use input space partitioning to generate all possible inputs.

Input Space Partitioning (ISP)

1) The FunctionWrapper will tell you which external classes are used.
2) Using the description (javadoc) you will find the scope of the variables returned.
3) Using this information you can (automatically) write test cases such that (most of) all is covered.

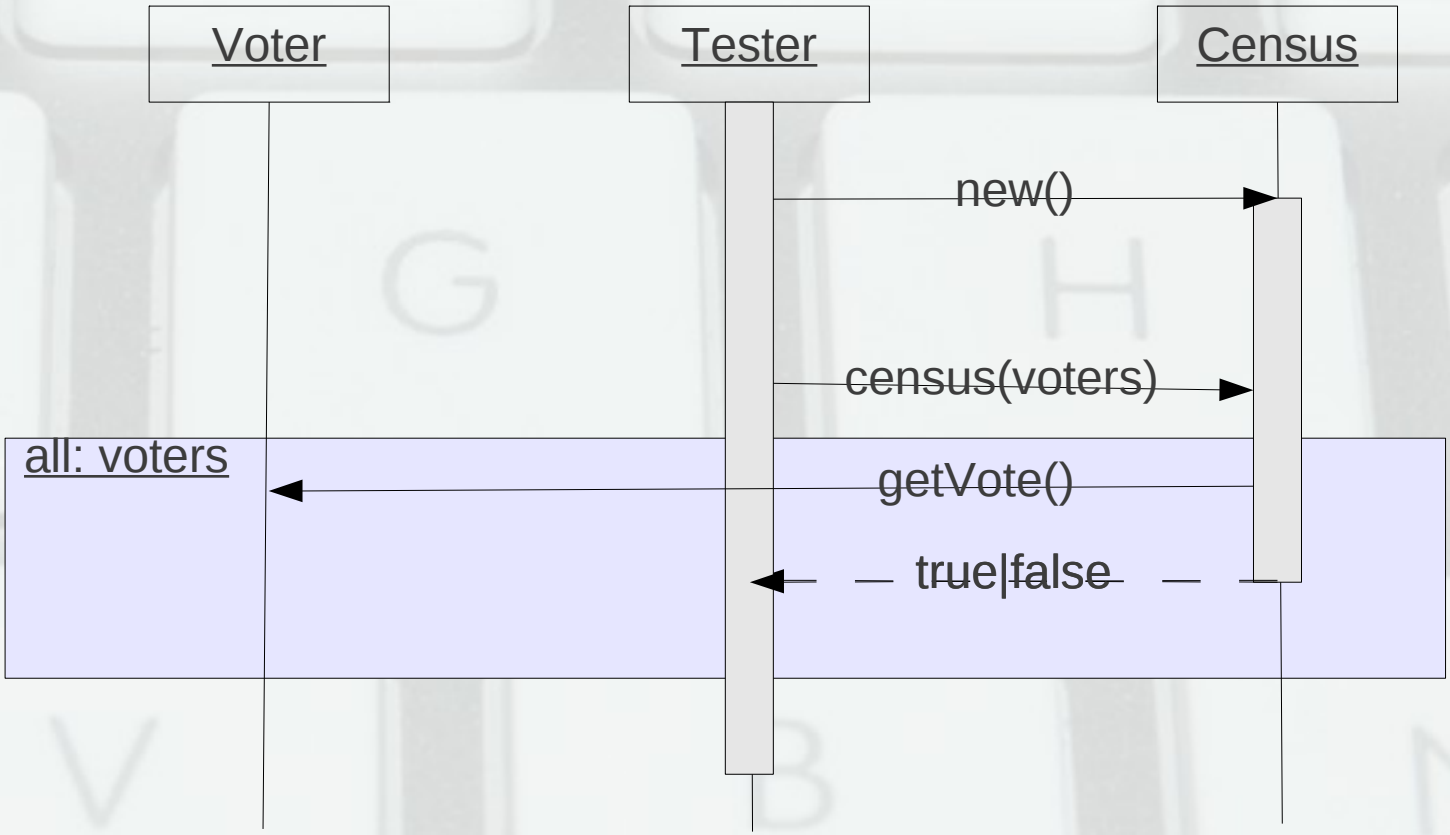
ISP - Example

```
/* Nobody likes voting on Tuesday don't they? */
Calendar rightNow = Calendar.getInstance();
if (rightNow.get(Calendar.DAY_OF_WEEK) == Calendar.WEDNESDAY) {
  logger.debug("Sorry not today");
  return false;
}
```

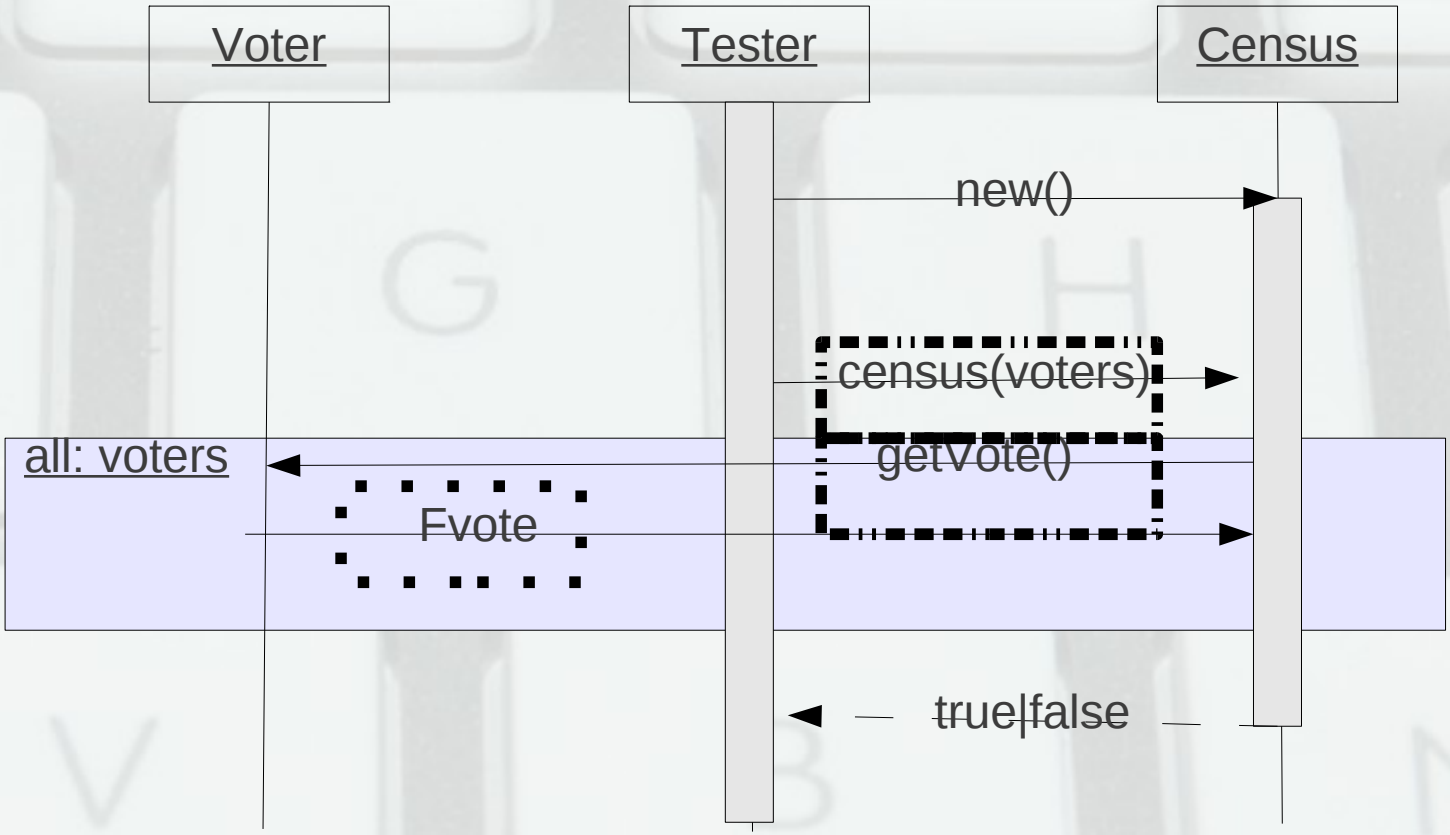
```
/* Nobody likes voting on Tuesday don't they? */
Calendar rightNow = CalendarWrapTest.getInstance();
if (rightNow.get(CalendarWrapTest.DAY_OF_WEEK) == CalendarWrapTest.WEDNESDAY) {
  logger.debug("Sorry not today");
  return false;
}
```

root=http://download.oracle.com/javase/1.4.2/docs/api/
$ROOT/java/util/Calendar.html#getInstance()
$ROOT/java/util/Calendar.html

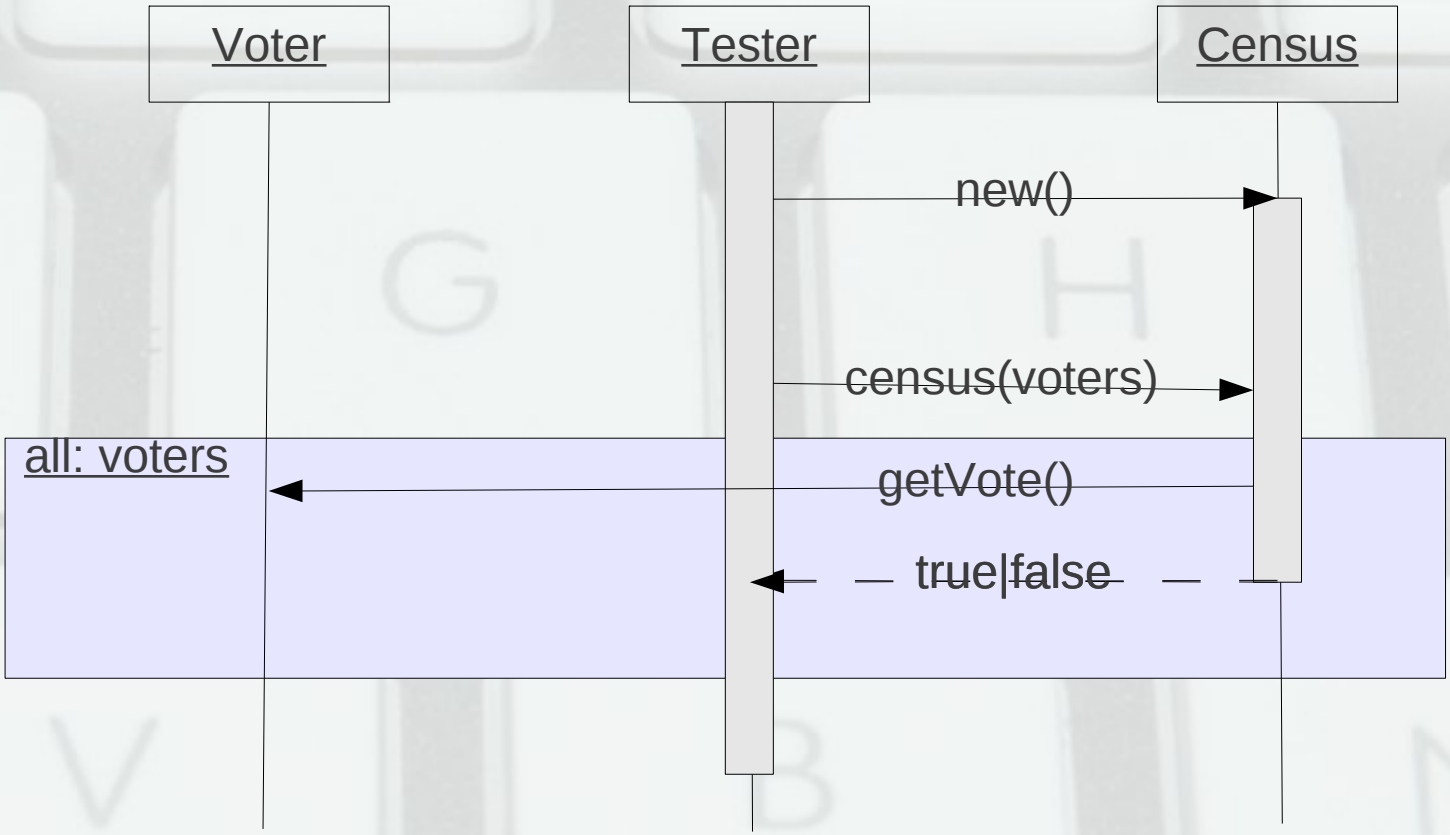
The Big Picture



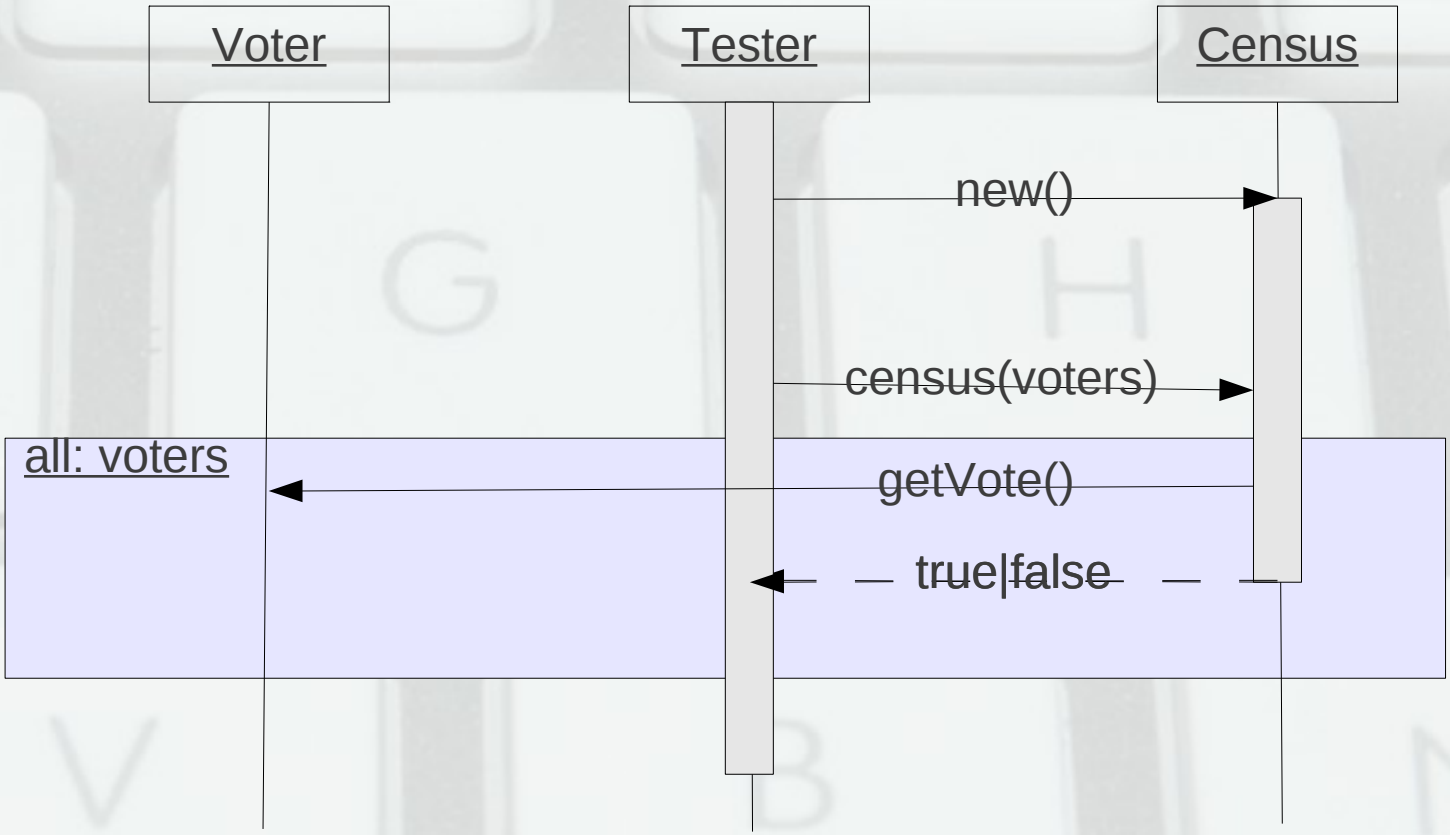
The Big Picture



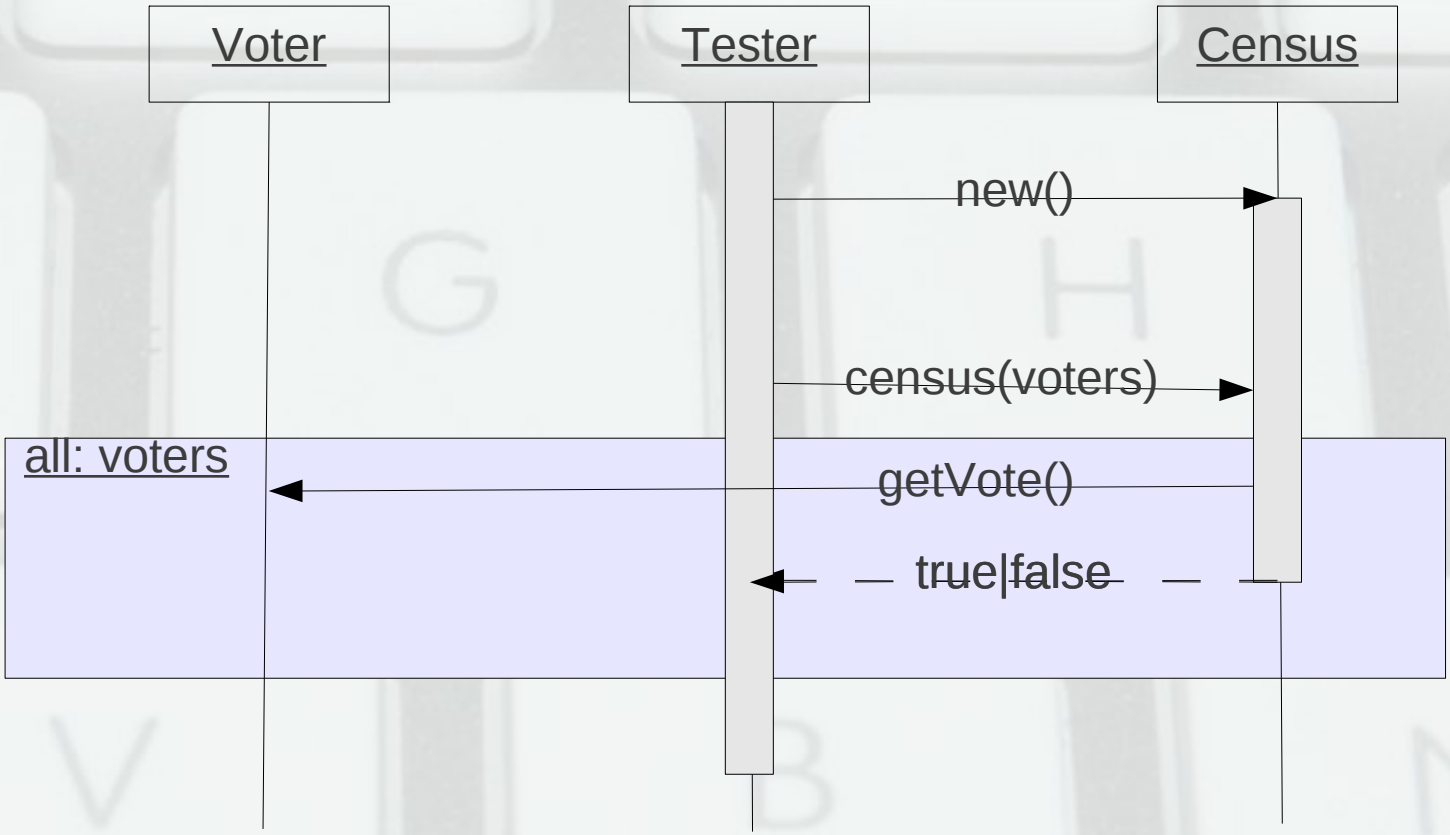
The Big Picture



The Big Picture



The Big Picture



Conclusions of 'WrapTest'

a) Allows us to implement a test case from a *SD.*

b) No proof of correctness can be given.

c) Some details are left for 'human' implementation.

d) Hard to 'wrap' libraries.