cphbusiness

COPENHAGEN BUSINESS ACADEMY

Automated test

Car diagnose computer

cphbusiness

Car diagnose computer

- Car has to be build to be tested
- Each test will tell of a potential error
- Only errors that matters will be reported
- If all tests succeed, it is safe to drive the car

Class diagnosis

- Class~~Car~~ has to be build to be tested
- Each test will tell of a potential error
- Only errors that matters will be reported
- If all tests succeed, commit ~~drive~~ the Class ~~car~~

- Which aspects of the ~~Car~~Class should we test?
- How to make it easy to test a ~~Car~~Class

- How to automate the test
  - To make it easy to test after you have made changes

Test concepts – expected results

- To test a method, we need to specify what is the expected outcome of method execution:
  - Return value
  - Thrown exception(s)
  - Call(s) to other objects (behavior)
  - Changes to the object of the method
  - Changes to external components (for example database)

Test *levels*

- **Unit tests**
  - At the level of individual methods and classes
  - Tool: JUnit
- **Integration tests**
  - Across layers
  - Tool: JUnit + Stubs + Mock objects
- **System tests**
  - Whole functionalities; end-to-end
  - Tool: Sometimes automated: JUnit + Selenium, Performance, usability etc.
- **Acceptance tests**
  - A contract betwn. the developers and the product owner on when a user story is implemented as expected
  - Tool: Sometimes automated + Sprint Review meeting

Testing in the FOG project

- **Unit tests**
  - **Must have some**
- Integration tests
  - Should have one
- ~~System tests~~
  - Cool if you do
- Acceptance tests
  - Happens at sprint review meetings (based on acceptance criteria defined for each story)

Test Discussion

- What is the purpose of testing?
- What is good testing?

White-box & black-box testing

- White-box testing focuses on the **source code (text)** of the program.
  - The tester constructs a test suite that demonstrates that all branches of the program can be executed.
  - The test suite is said to cover the statements of the program.

- Black-box testing focuses on the **problem** that the program is supposed to solve
  - More precisely, the problem statement or specification for the program.

Black-Box Techniques

Equivalence Partitioning

Purpose (rationale)

- Aims at minimizing the number of test cases

How

- Input values are partitioned into equivalence classes if they result in the same program behavior, i.e. will find the same errors
- A test case is written for each partition
- OBS! Invalid input are separate equivalence classes

Equivalence Partitioning - Example

```
// pre: 0 < age
// post: returns true if age >= 18, otherwise false

public boolean legalAge(int age) { ... }
```

Equivalence classes	Test case (legalAge)	
age <= 0	invalid	**-1**
0 < age <18	not legal age	**10**
18 <= age	legal age	**20**

Boundary Value Analysis

Purpose (rationale)

- Errors often show at the boundaries between equivalence classes

How

- Choose minimum and maximum values from an equivalence class together with first or last value respectively in adjacent equivalence classes

Boundary Values – Same Ex.

```
// pre: 0 < age
// post: returns true if age >= 18, otherwise false

public boolean legalAge(int age) { ... }
```

Test cases (not legal age equivalence class)

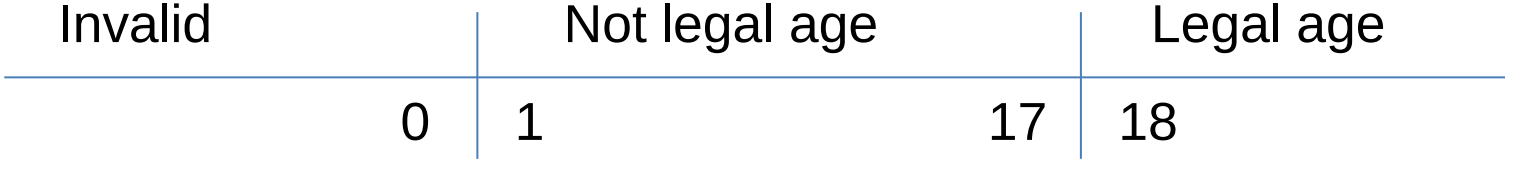
0 (adjacent partition)

1 (min. value - close to boundary)

10 (normal input – found during EP)

17 (max. value - close to boundary)

18 (adjacent partition)

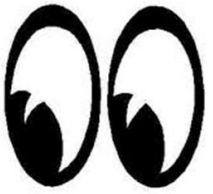


What to do with open boundaries?

- Open boundaries are more difficult to test, but there are ways to approach them.

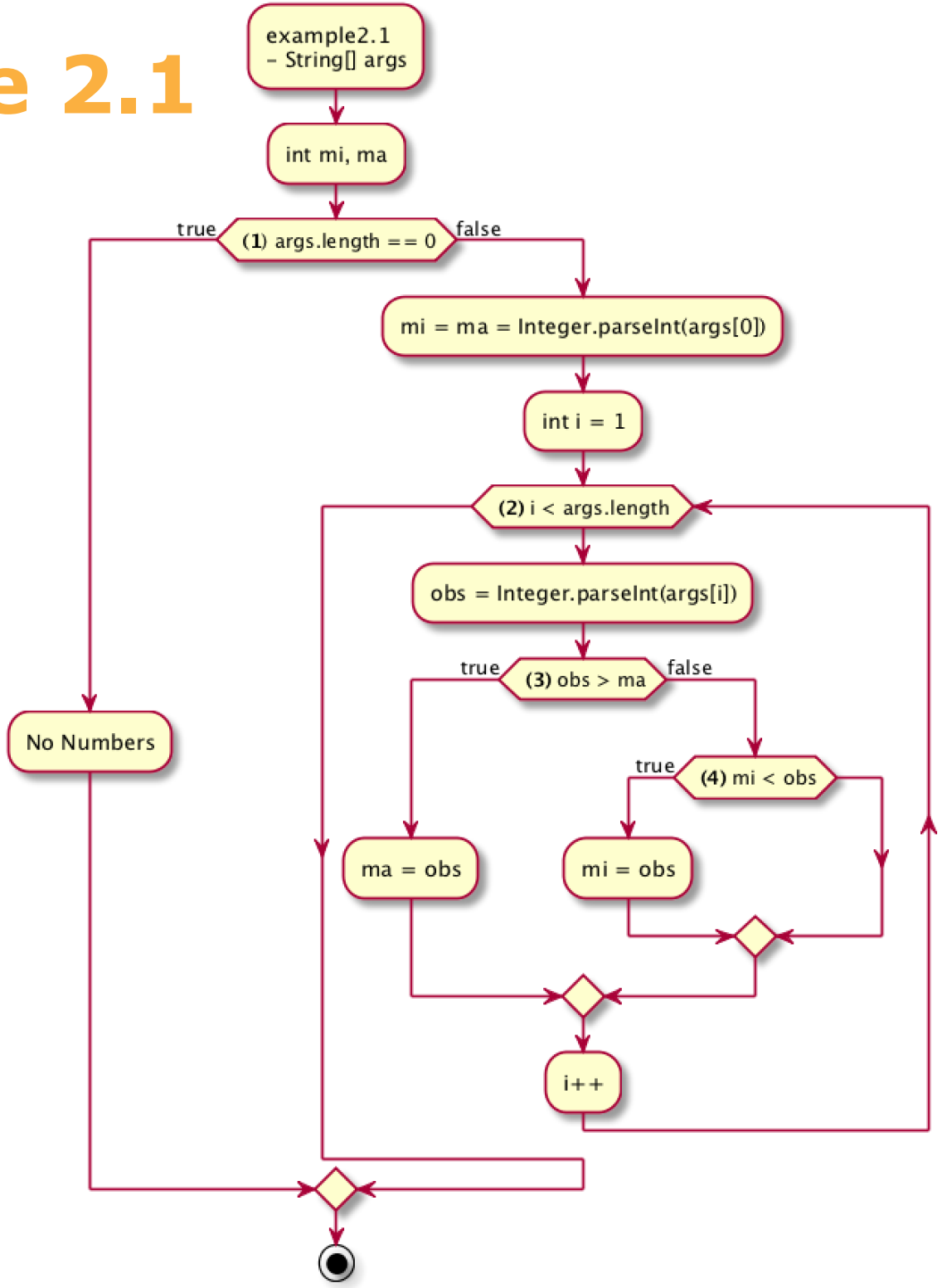
- The best solution is to find out what the boundary should be specified as:
  - Ask PO
  - investigate other related areas of the system. Ex.:
    - The input field that holds the account balance figure may be only six figures plus two decimal figures. This would give a maximum account balance of $999 999.99 so we could use that as our maximum boundary value.

Whitebox (Peter Sestoft Example 2.1



```
public static void main ( String[] args )
{
  int mi, ma;
  if (args.length == 0)                                    /* 1 */
    System.out.println("No numbers");
  else
    {
      mi = ma = Integer.parseInt(args[0]);
      for (int i = 1; i < args.length; i++)                /* 2 */
        {
          int obs = Integer.parseInt(args[i]);
          if (obs > ma) ma = obs;                          /* 3 */
          else if (mi < obs) mi = obs;                     /* 4 */
        }
      System.out.println("Minimum = " + mi + "; maximum = " + ma);
    }
}
```

Example 2.1



Whitebox testing: constructing test cases

The resulting test suite includes enough input data sets to make sure that:

- all methods have been called,
- that both the true and false branches have been executed in if statements,
- that every loop has been executed zero, one, and more times – *why is "more times" important?*
- that all branches of every switch statement have been executed.

For every input data set, the expected output must be specified also.

Test coverage

- In practice white box testing is done on
  - Algorithms
  - To analyze for security holes

- Normally one uses **automated test coverage** tools:
  - Measures to which extend our code has been tested
  - Integrated into Netbeans – colors your code red, green, yellow
  - It is pragmatic and "good enough".