# CS608
# Software Testing

Dr. Stephen Brown

Room Eolas 116

stephen.brown@mu.ie

# CS608

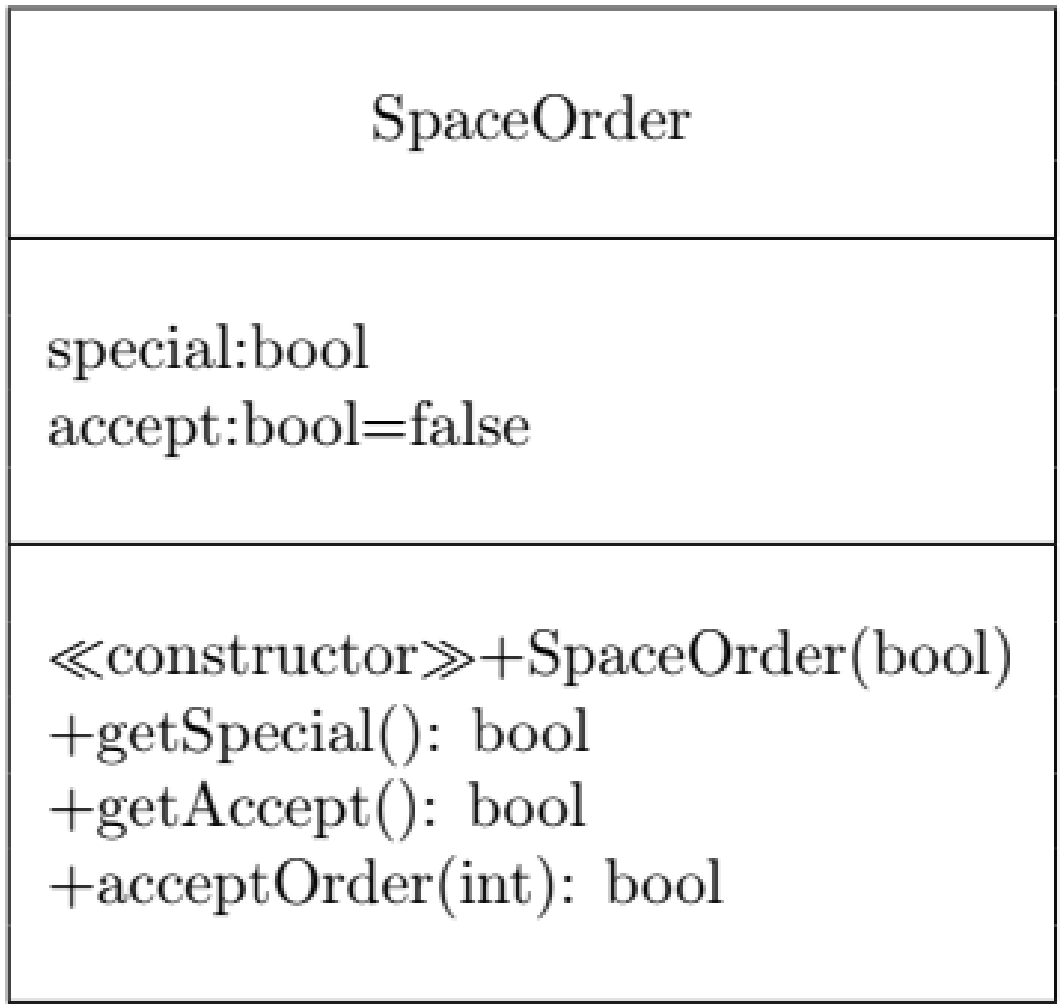
Testing Object-Oriented Software in More Detail

# EVALUATION
# EP TESTING IN CLASS CONTEXT

# EVALUATION

- Now we have seen inheritance and state-based testing
- We can evaluate EP testing in class context, using:
  - Simple typo faults
  - Inheritance faults
  - State-based faults
- Strengths and Weaknesses
- Key Points
- Notes for Experienced Testers

# Reminder: Class SpaceOrder



# Reminder: EP Tests in Class Context

| ID | TCI Covered | Inputs | Expected Results Return Value |
|---|---|---|---|
| T1 | EP1 | new SpaceOrder(true) | |
| | EP3 | getSpecial() | true |
| | EP5 | getAccept() | false |
| T2 | EP2 | new SpaceOrder(false) | |
| | EP4 | getSpecial() | false |
| T3 | [EP1] | new SpaceOrder(true) | |
| | EP7,10,14 | acceptOrder(7) | true |
| | EP12 | getAccept() | true |
| T4 | [EP2] | new SpaceOrder(false) | |
| | EP8,11,[14] | acceptOrder(504) | true |
| | [EP12] | getAccept() | true |
| T5 | [EP2] | new SpaceOrder(false) | |
| | EP9,[10,14] | acceptOrder(5000) | true |
| | EP13 | getAccept() | false |
| T6* | [EP2] | new SpaceOrder(false) | |
| | EP6,15 | acceptOrder(-5000) | false |
| | [EP13] | getAccept() | false |

# Reminder: EP Test Results

```
PASSED: testConstructor("SpaceOrderTest T1", true, true, false)
PASSED: testConstructor("SpaceOrderTest T2", false, false, false)
PASSED: testAcceptOrder("SpaceOrderTest T3", true, 7, true, true)
PASSED: testAcceptOrder("SpaceOrderTest T4", false, 504, true, true)
PASSED: testAcceptOrder("SpaceOrderTest T5", false, 5000, true, false)
PASSED: testAcceptOrder("SpaceOrderTest T6", false, -5000, false, false)
=============================================
Command line suite
Total tests run: 6, Passes: 6, Failures: 0, Skips: 0
=============================================
```

# Evaluation of EP Tests in Class Context

- The tests developed for conventional testing in class context are evaluated in by introducing faults and examining the test results
- Limitations: Three types of fault are demonstrated:
    1. A simple typo fault which you would expect to find with equivalence partition testing
    2. A more complex, state-based fault
    3. A more complex, inheritance-based fault (in a new subclass)

# Simple Typo Fault

```
14    public boolean acceptOrder(int space) {
15      boolean status=true;
16      this.accept = false;
17      if (space<=0)
18        status=false;
19      else if (space<=10240 && (space>=16 || this.special)) // typo
              fault
20        this.accept = true;
21      return status;
22    }
```

- 10240 is a typo: should be 1024

# Test Results (Typo Fault)

```
PASSED: testConstructor("SpaceOrderTest T1", true, true, false)
PASSED: testConstructor("SpaceOrderTest T2", false, false, false)
PASSED: testAcceptOrder("SpaceOrderTest T3", true, 7, true, true)
PASSED: testAcceptOrder("SpaceOrderTest T4", false, 504, true, true)
PASSED: testAcceptOrder("SpaceOrderTest T6", false, -5000, false, false)
FAILED: testAcceptOrder("SpaceOrderTest T5", false, 5000, true, false)
java.lang.AssertionError: expected [false] but found [true]
        at example.SpaceOrderTest.testAcceptOrder(SpaceOrderTest.java:40)
===============================================
Command line suite
Total tests run: 6, Passes: 5, Failures: 1, Skips: 0
===============================================
```

- The simple fault has been detected
- As for any EP test, there is no guarantee that any particular fault will be found

# State-Based Fault

- Support for a new attribute locked has been incorrectly added to the class; the goal was to lock "accept" until it has been read
- This fault prevents acceptOrder() from working correctly if it is called more than once
- getAccept() corrupts the object (by incorrectly locking it)

```
1  public class SpaceOrder {
2
3     boolean special;
4     boolean accept=false;
5     boolean locked=false;
6
7     public SpaceOrder(boolean isSpecial) {
8        special = isSpecial;
9     }
10
11    public boolean getSpecial() {
12       return this.special;
13    }
14
15    public boolean acceptOrder(int space) {
16       if (locked)
17          return false;
18       boolean status=true;
19       this.accept = false;
20       if (space<=0)
21          status=false;
22       else if (space<=1024 && (space>=16 || this.special))
23          this.accept = true;
24       return status;
25    }
26
27    public boolean getAccept() {
28       locked = true;
29       return this.accept;
30    }
31
32 }
```

# Impact

```
15    public boolean acceptOrder(int space) {
16      if (locked)
17        return false;
```

- In essence, this introduces a new state Locked which is not in the state diagram
- acceptOrder() does not work in this state

# Test Results (State Fault)

```
PASSED: testConstructor("SpaceOrderTest T1", true, true, false)
PASSED: testConstructor("SpaceOrderTest T2", false, false, false)
PASSED: testAcceptOrder("SpaceOrderTest T3", true, 7, true, true)
PASSED: testAcceptOrder("SpaceOrderTest T4", false, 504, true, true)
PASSED: testAcceptOrder("SpaceOrderTest T5", false, 5000, true, false)
PASSED: testAcceptOrder("SpaceOrderTest T6", false, -5000, false, false)
===============================================
Command line suite
Total tests run: 6, Passes: 6, Failures: 0, Skips: 0
===============================================
```

- The state fault has not been detected
  - Why? The EP tests do not check state-based behaviour
- A more systematic exploration of state-based behaviour is required to reliably detect such faults (state-based tests)

# Demonstrating the Fault

| Number | Start State | Event | End State |
|---|---|---|---|
| 1 | Start | Constructor | Unready |
| 2 | Unready | getSpecial() | Unready |
| 3 | Unready | getAccept() | Unready |
| 4 | Unready | acceptOrder() | Ready |
| 5 | Ready | getSpecial() | Ready |
| 6 | Ready | getAccept() | Ready |
| 7 | Ready | acceptOrder() | Ready |

- The results of running the state-based tests (from last weeks' lecture/book Section 9.4.11) against the faulty implementation of SpaceOrder are:

```
FAILED: allTransitionsTest
java.lang.AssertionError: expected [true] but found [false]
        at example.SpaceOrderStateTest.allTransitionsTest(SpaceOrderStateTest.
            java:22)
===============================================
Command line suite
Total tests run: 1, Passes: 0, Failures: 1, Skips: 0
===============================================
```

# Demonstrating the Fault

- The results of running the state-based tests (from last weeks' lecture/book Section 9.4.11) against the faulty implementation of SpaceOrder are:

```
FAILED: allTransitionsTest
java.lang.AssertionError: expected [true] but found [false]
        at example.SpaceOrderStateTest.allTransitionsTest(SpaceOrderStateTest.
            java:22)
===============================================
Command line suite
Total tests run: 1, Passes: 0, Failures: 1, Skips: 0
===============================================
```

- The fault has been found
- Systematic testing of the transitions is more likely to find state-based faults than conventional testing in class context

# Inheritance Fault

- TrackableSpaceOrder inherits from SpaceOrder
- The method acceptOrder() has been overridden on lines 17-20
- The implementation on line 19 is incomplete
- Often seen where a empty of skeleton method is coded initially, and the developer has forgotten to complete it

```
1  public class TrackableSpaceOrder extends SpaceOrder {
2
3      private long code=0;
4
5      public TrackableSpaceOrder(boolean isSpecial) {
6              super(isSpecial);
7          }
8
9      public void setTrackCode( int newValue ) {
10         code = newValue;
11     }
12
13     public int getTrackCode() {
14         return (int)code;
15     }
16
17     @Override
18         public boolean acceptOrder(int space) {
19         return true;
20     }
21
22 }
```

# Test Results (Inheritance Fault)

- Results of running the EP tests for SpaceOrder against an instance of class TrackableSpaceOrder

```
PASSED: testAcceptOrder("SpaceOrderTest T5", false, 5000, true, false)
FAILED: testAcceptOrder("SpaceOrderTest T3", true, 7, true, true)
java.lang.AssertionError: expected [true] but found [false]
        at example.TrackableSpaceOrderInhTest.testAcceptOrder(
            TrackableSpaceOrderInhTest.java:26)

FAILED: testAcceptOrder("SpaceOrderTest T4", false, 504, true, true)
java.lang.AssertionError: expected [true] but found [false]
        at example.TrackableSpaceOrderInhTest.testAcceptOrder(
            TrackableSpaceOrderInhTest.java:26)

FAILED: testAcceptOrder("SpaceOrderTest T6", false, -5000, false, false)
java.lang.AssertionError: expected [false] but found [true]
        at example.TrackableSpaceOrderInhTest.testAcceptOrder(
            TrackableSpaceOrderInhTest.java:25)
===============================================
Command line suite
Total tests run: 4, Passes: 1, Failures: 3, Skips: 0
===============================================
```

# Test Results (Inheritance Fault)

- Results of running the EP tests for SpaceOrder against an instance of class TrackableSpaceOrder
- We'll discuss how to do this in a few weeks time

```
PASSED: testAcceptOrder("SpaceOrderTest T5", false, 5000, true, false)
FAILED: testAcceptOrder("SpaceOrderTest T3", true, 7, true, true)
java.lang.AssertionError: expected [true] but found [false]
        at example.TrackableSpaceOrderInhTest.testAcceptOrder(
            TrackableSpaceOrderInhTest.java:26)

FAILED: testAcceptOrder("SpaceOrderTest T4", false, 504, true, true)
java.lang.AssertionError: expected [true] but found [false]
        at example.TrackableSpaceOrderInhTest.testAcceptOrder(
            TrackableSpaceOrderInhTest.java:26)

FAILED: testAcceptOrder("SpaceOrderTest T6", false, -5000, false, false)
java.lang.AssertionError: expected [false] but found [true]
        at example.TrackableSpaceOrderInhTest.testAcceptOrder(
            TrackableSpaceOrderInhTest.java:25)
===============================================
Command line suite
Total tests run: 4, Passes: 1, Failures: 3, Skips: 0
===============================================
```

- Three of the SpaceOrder EP tests have failed when run against a TrackableSpaceOrder instance
- Tests T1 and T2 can not be used as they test the constructor in SpaceOrder which is not a method of the class, and is not inheritable!

# Strengths and Weaknesses

- The limitations of conventional testing in state context are related both to the weaknesses of the underlying test technique (EP, BVA, DT, SC, BC) and to the additional complications introduced by testing in class context
- Conventional tests **may** find state-based and inheritance faults
- But they are **more likely** to be found by systematic testing against those fault models, using:
  - Inheritance Testing
  - State-Based Testing

# Key Points

- Methods need to be tested in class context, using setter and getter methods to assist in the tests
- All black-box and white-box test techniques can be used to perform conventional testing in class context
- Object-oriented programs have special features that require testing: two key ones are State-Based behaviour and Inheritance
- As the test code does not have access to private attributes, Built-In Testing can be a useful technique. It is particularly useful for verifying state invariants

# Key Points (Continued)

- There are many other aspects of testing object-oriented programs not shown here
- **Every UML diagram** is a specification, and therefore a potential source of test cases and test data
- UML 2.5
  - Effectively 14 diagrams
  - Plus 8 unofficial diagrams
  - And 4 “meta-diagrams”

# Notes for Experienced Testers

- Work usually done in their mind:
  - Selecting the methods to test, especially for getters and setters
  - For conventional testing in class context, deciding on the test design technique (EP, BVA, DT, SC, BC)
  - Identifying the test coverage items based on the test design technique
  - Designing the test cases
  - Writing the test code

# More Notes for Experienced Testers

- Especially in an agile development environment, an experienced tester may use the User Stories and acceptance criteria as the basis of OO tests:
  - This is relatively straightforward when the model that the classes implement matches the user problem domain closely
  - In this case particular user actions can be easily matched to method calls, and the sequence of interactions listed in an acceptance criteria for a user story can be easily mapped to a sequence of method calls
- The experienced tester will also probably code tests directly from the state diagram or inheritance tree, having made sure that the tests are designed to allow inheritance testing