

CS608

Software Testing

Dr. Stephen Brown

Room Eolas 116

stephen.brown@mu.ie

CS608

Testing Object-Oriented Software

(Essentials of Software Testing, Chapter 9)

“Object-Oriented Testing” or OOT

Introduction to OOT

- The black-box and white-box unit test techniques seen so far have been applied to a single static method
- This allows key concepts of the test design techniques to be introduced without the additional complexities introduced by classes and objects
- The same techniques can be applied to testing the instance methods of a class – this is unit testing where the unit is a class
- Today we will look at “**testing in class context**”
- Then (next week) look at more advanced OO testing techniques

“Testing in Class Context”

- Methods interact with each other and cannot be tested independently
- Methods interact with other methods via **class attributes**
- Testing in class context refers to testing methods not individually, but in the wider context of their class
- Test data may be derived using any of the previously covered black-box or white-box techniques
- Accessing attributes: getter and setter methods
- Note: we will consider testing methods in a class called by other methods in the same class when we look at integration testing

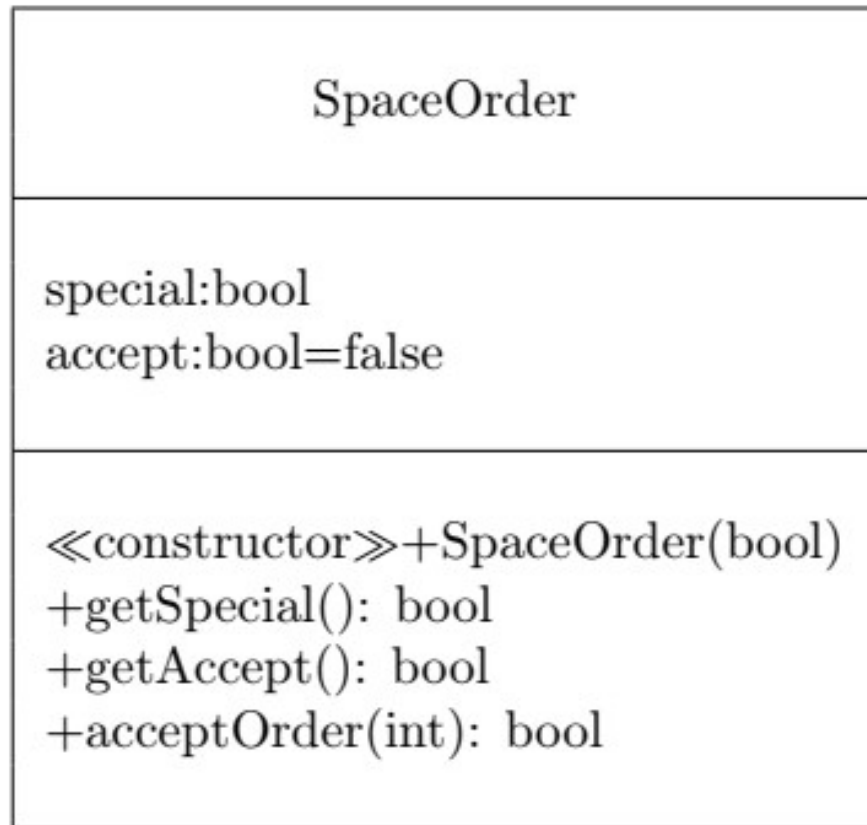
Use of Setter and Getter Methods in Testing

1. (After the constructor is called) typically, **setter** methods must be called first to initialise any attributes used by the method being tested
2. Then the **method** is called, passing required inputs as parameters
3. The method **return value** must be verified
4. And then any changes the method has made to the attributes must be verified by calling **getter** methods
5. The **interaction** of a number of methods must be tested, not the operation of a single method
6. Whether setters and getters is good object-oriented (OO) design is an open question – the software tester needs to access the attributes

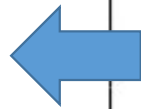
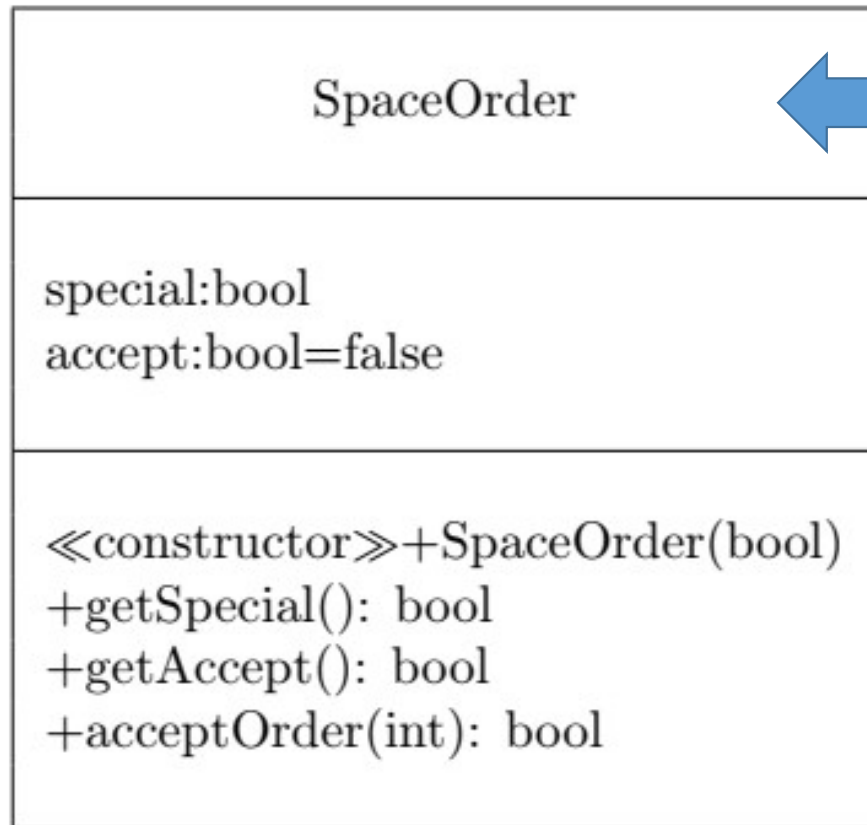
New Example: SpaceOrder

- Class SpaceOrder supports booking space in a warehouse
- Method acceptOrder() decides whether an order can be accepted
- In general, all orders must fall within a specified minimum and maximum
- For special customers, orders for less than the minimum space are accepted
- Demonstrate using equivalence partition testing
- Tests for other black-box and white-box techniques would be developed in exactly the same way (BVA, DT, SC, BC)

UML Class Diagram

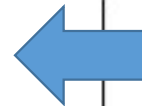
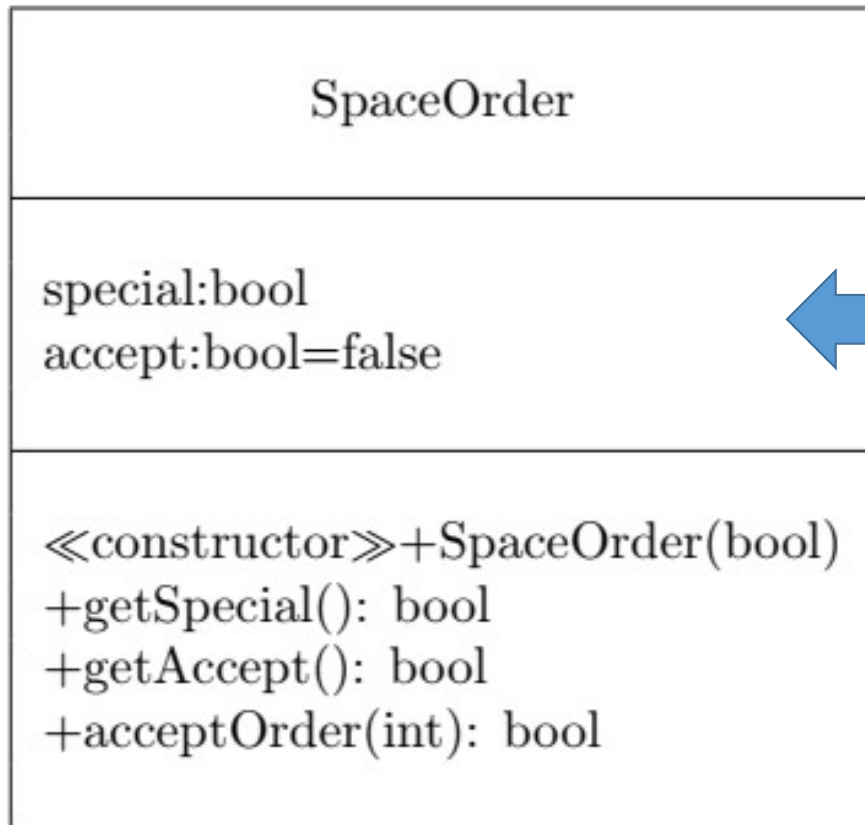


UML Class Diagram: Recap



The class name: SpaceOrder

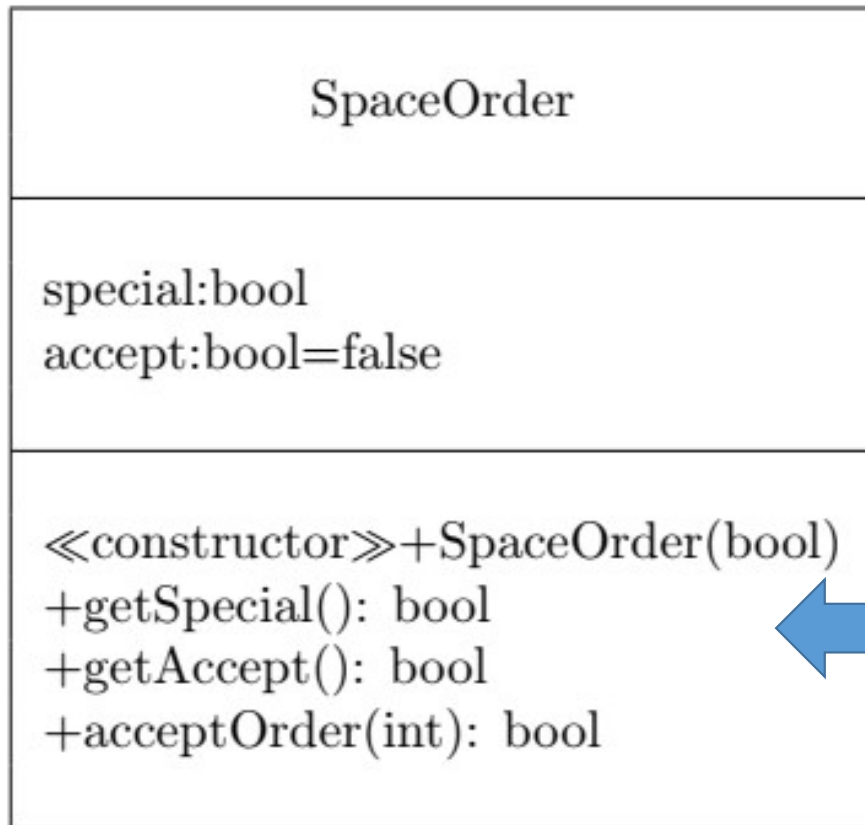
UML Class Diagram: Recap



Attributes:

- special is a boolean
- accept is a boolean, initialised to false

UML Class Diagram: Recap



Methods:

- public constructor `SpaceOrder()` accepts a single boolean parameter
- public method `getSpecial()` has no parameters and returns a boolean value
- public method `getAccept()` has no parameters and returns a boolean value
- public method `acceptOrder()` accepts a single integer parameter, and returns a boolean value

Source Code

SpaceOrder
special:bool accept:bool=false
«constructor»+SpaceOrder(bool) +getSpecial(): bool +getAccept(): bool +acceptOrder(int): bool

```
1 public class SpaceOrder {  
2  
3     protected boolean special;  
4     protected boolean accept=false;  
5  
6     public SpaceOrder(boolean isSpecial) {  
7         special = isSpecial;  
8     }  
9  
10    public boolean getSpecial() {  
11        return this.special;  
12    }  
13  
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<=1024 && (space>=16 || this.special))  
20            this.accept = true;  
21        return status;  
22    }  
23  
24    public boolean getAccept() {  
25        return this.accept;  
26    }  
27  
28 }
```

Source Code

SpaceOrder
special:bool accept:bool=false
«constructor»+SpaceOrder(bool) +getSpecial(): bool +getAccept(): bool +acceptOrder(int): bool

```
1 public class SpaceOrder {
2
3     protected boolean special;
4     protected boolean accept=false;
5
6     public SpaceOrder(boolean isSpecial) {
7         special = isSpecial;
8     }
9
10    public boolean getSpecial() {
11        return this.special;
12    }
13
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<=1024 && (space>=16 || this.special))
20            this.accept = true;
21        return status;
22    }
23
24    public boolean getAccept() {
25        return this.accept;
26    }
27
28 }
```

Source Code

SpaceOrder
special:bool accept:bool=false
«constructor»+SpaceOrder(bool) +getSpecial(): bool +getAccept(): bool +acceptOrder(int): bool

```
1 public class SpaceOrder {
2
3     protected boolean special;
4     protected boolean accept=false;
5
6     public SpaceOrder(boolean isSpecial) {
7         special = isSpecial;
8     }
9
10    public boolean getSpecial() {
11        return this.special;
12    }
13
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<=1024 && (space>=16 || this.special))
20            this.accept = true;
21        return status;
22    }
23
24    public boolean getAccept() {
25        return this.accept;
26    }
27
28 }
```

SpaceOrder API

- **void example.SpaceOrder(boolean isSpecial)**

Constructor

Parameters:

isSpecial – specifies whether this SpaceOrder should be special or not

- **boolean example.SpaceOrder.getSpecial()**

Returns:

the value of special

- **boolean example.SpaceOrder.acceptOrder(int space)**

Determine whether a warehouse space order can be accepted.

If valid input data, set accept true if order can be accepted, and false if not.

If invalid input data (value of space), set accept false.

An order can be accepted as follows. For all orders, the space must not be greater than the maximum space of 1024 m². For a standard order, the space must be at least the minimum space of 16m² (a special order has no lower limit).

Parameters:

space – space in m² requested in the order (must be >0)

Returns:

true if valid input data, and the attribute accept has been set successfully.
Otherwise false

- **boolean example.SpaceOrder.getAccept()**

Returns:

the value of accept (whether the SpaceOrder has been accepted or not)

constructor

- **void example.SpaceOrder(boolean isSpecial)**

Constructor

Parameters:

isSpecial – specifies whether this SpaceOrder should be special or not

- **boolean example.SpaceOrder.getSpecial()**

Returns:

the value of special

- **boolean example.SpaceOrder.acceptOrder(int space)**

- **void example.SpaceOrder(boolean isSpecial)**

Constructor

Parameters:

isSpecial – specifies whether this SpaceOrder should be special or not

space – space in m² requested in the order (must be >0)

Returns:

true if valid input data, and the attribute accept has been set successfully.
Otherwise false

- **boolean example.SpaceOrder.getAccept()**

Returns:

the value of accept (whether the SpaceOrder has been accepted or not)

getSpecial()

- `void example.SpaceOrder(boolean isSpecial)`

Constructor

Parameters:

isSpecial – specifies whether this SpaceOrder should be special or not

- `boolean example.SpaceOrder.getSpecial()`

Returns:

the value of special

- `boolean example.SpaceOrder.acceptOrder(int space)`

- `boolean example.SpaceOrder.getSpecial()`

Returns:

the value of special

limit).

Parameters:

space – space in m² requested in the order (must be >0)

Returns:

true if valid input data, and the attribute accept has been set successfully.
Otherwise false

- `boolean example.SpaceOrder.getAccept()`

Returns:

the value of accept (whether the SpaceOrder has been accepted or not)

acceptOrder()

- `void example.SpaceOrder(boolean isSpecial)`

Constructor

Parameters:

`isSpecial` – specifies whether this `SpaceOrder` should be special or not

- `boolean example.SpaceOrder.acceptOrder(int space)`

Determine whether a warehouse space order can be accepted.

If valid input data, set `accept` true if order can be accepted, and false if not.

If invalid input data (value of `space`), set `accept` false.

An order can be accepted as follows. For all orders, the space must not be greater than the maximum space of 1024 m². For a standard order, the space must be at least the minimum space of 16m² (a special order has no lower limit).

Parameters:

`space` – space in m² requested in the order (must be >0)

Returns:

true if valid input data, and the attribute `accept` has been set successfully.
Otherwise false

getAccept()

- **void example.SpaceOrder(boolean isSpecial)**

Constructor

Parameters:

isSpecial – specifies whether this SpaceOrder should be special or not

- **boolean example.SpaceOrder.getSpecial()**

Returns:

the value of special

- **boolean example.SpaceOrder.acceptOrder(int space)**

- **boolean example.SpaceOrder.getAccept()**

Returns:

the value of accept (whether the SpaceOrder has been accepted or not)

Parameters:

space – space in m² requested in the order (must be >0)

Returns:

true if valid input data, and the attribute accept has been set successfully.
Otherwise false

- **boolean example.SpaceOrder.getAccept()**

Returns:

the value of accept (whether the SpaceOrder has been accepted or not)

Key Steps for Developing Tests in Class Context

1. Decide which methods to test
2. Select a test technique
3. Analyse interactions between methods and attributes
4. Develop TCI's
5. Develop Test Cases

Analysis

- During unit testing in the previous chapters, there was a single static method to test (the method `giveDiscount()`)
- In contrast, a class may contain many methods, and a decision must be made which should be tested

1. Deciding on Which Methods to Test

- Five categories of method from a software testing viewpoint:
 1. Static methods
 2. Constructors
 3. Accessor methods (getters and setters)
 - Other Methods:
 4. Methods with no class interaction
 5. Methods with class interaction

Methods to Test in Class SpaceOrder

1. **Static methods** – none in this class
2. Constructors
3. Accessor methods (getters and setters)
4. Methods with no class interaction
5. Methods with class interaction

Methods to Test in Class SpaceOrder

1. Static methods
2. **Constructors** – test to ensure attributes are correctly initialised:
 - SpaceOrder(boolean)
3. Accessor methods (getters and setters)
4. Methods with no class interaction
5. Methods with class interaction

Methods to Test in Class SpaceOrder

1. Static methods
2. Constructors
3. **Accessor methods (getters and setters)** – the general rule is to only test these if they are manually written or contain more than a single assignment (setter) or return statement (getter). However, in this example the class is small and the getters and setters can easily be verified through a code review. Therefore we will not test these:
 - getAccept()
 - getSpecial()
4. Methods with no class interaction
5. Methods with class interaction

Testing Setters and Getters

- If you did test them, you would write simple EP tests to make sure that if a value is written to an attribute (using the setter) then the same value is read back (using the getter)
- Probably place all in a single test method for the class

```
@test testSettersAndGetters() {  
    object.setX(100);  
    assertEquals(object.getX(), 100);  
}
```

Methods to Test in Class SpaceOrder

1. Static methods
2. Constructors
3. Accessor methods (getters and setters)
4. **Methods with no class interaction (do not read or write any of the class attributes) – none**
 - if present, these can be tested individually using non-OO techniques approaches as shown previously
5. Methods with class interaction

Methods to Test in Class SpaceOrder

1. Static methods – none in this class
2. Constructors
3. Accessor methods (getters and setters)
4. Methods with no class interaction
5. **Methods with class interaction (read and/or write the class attributes)** – we will test all of these:
 - `acceptOrder(int)`

2. Selecting a Test Technique

- Decide which test technique to use to select test data
- For simplicity, we will use **EP testing**
- Would use essentially the same approach for other techniques: BVA, DT, SC, BC
- Only the results of the analysis will be shown

3. Now Analyse the Methods and Attributes

- Constructors and initialisation
- Accessor methods: getters and setters
- Methods with class interaction

Constructors and Initialisation

- SpaceOrder has a single constructor, which sets the value for special

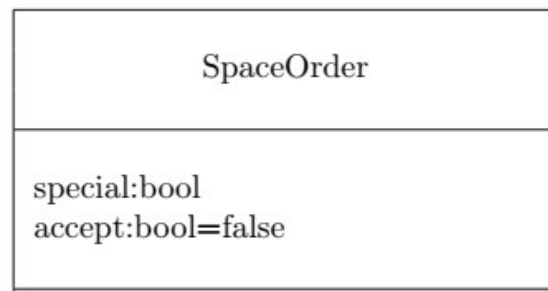
- `void example.SpaceOrder(boolean isSpecial)`

Constructor

Parameters:

`isSpecial` – specifies whether this SpaceOrder should be special or not

- Class Diagram specifies that `accept` is initialised to `false`



Constructors and Initialisation

- SpaceOrder has a single constructor, which sets the value for special
- Class Diagram specifies that accept is initialised to false

Constructor	Attribute	Value
SpaceOrder(x:bool)	special	true or false (as passed to the constructor)
	accept	false (specified in the class diagram)

Accessor Methods

- We have decided not to test the getter and setter methods
- However some of the attributes can only be accessed via their setters and getters, so it is important to identify these for each attribute

Attribute	Getters	Setters
special	getSpecial()	–
accept	getAccept()	–

- The method `acceptOrder()` reads and writes to attributes (`special` and `accept`) but is not regarded as a pure getter or a setter method, as it does other processing

Methods with Class Interaction

- There are no methods with no class interaction
- There is a single method with class interaction: acceptOrder(int)
- The table below details the attributes read and written

Method	Read	Written
acceptOrder(int)	special	accept

4. Equivalence Partition TCIs

- Using the technique shown for EP, equivalence partition tests are developed for the inputs and outputs for each method
- The full working of each step will not be shown, just the results of each
- For each method, the inputs and outputs may be explicit or implicit:
 - Explicit inputs are parameters passed to the method
 - Implicit inputs are attributes read by the method
 - An explicit output is the return value from a method, or an exception raised
 - Implicit outputs are attributes written by the method
- Rule of thumb: implicit means using attributes

Parameter “space” to acceptOrder()

- The only parameter with interesting value lines (that require analysis) is the input parameter *space* passed to acceptOrder()
- All the other parameters are boolean and do not require further analysis to identify their equivalence partitions

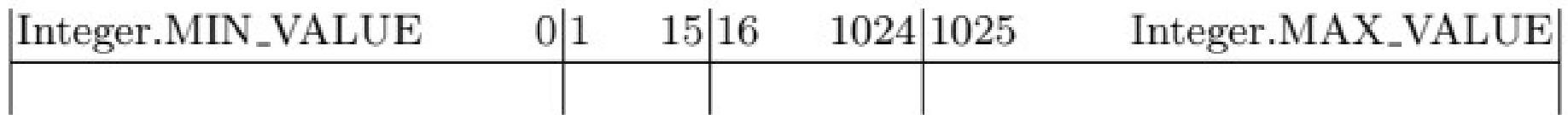


Figure 9.3: Value Line for space

OO EPs for SpaceOrder

- * indicates an input error partition
- “accept” only has the possible EP false as output from SpaceOrder()
- Use simple method name if unique as shown
 - Otherwise use full method signature, with parameter types
- If an attribute is both an input and an output for a method, then the input and output equivalence partitions must be separately listed – not needed here

Method	Name	Equivalence Partition
SpaceOrder()	isSpecial	true false
	special	true false
	accept	false
acceptOrder()	space	(*) Integer.MIN_VALUE..0 1..15 16..1024 1025..Integer.MAX_VALUE
	return value	true false
	special	true false
	accept	true false

EP Test Coverage Items for SpaceOrder

	TCI	Method	Name	Equivalence Partition	Test Case
inputs {	EP1	SpaceOrder()	isSpecial	true	To Be Completed Later
	EP2			false	
outputs {	EP3		special	true	
	EP4			false	
	EP5		accept	false	
	EP6*	acceptOrder()	space	Integer.MIN_VALUE..0	
	EP7			1..15	
	EP8			16..1024	
	EP9			1025..Integer.MAX_VALUE	
inputs {	EP10		special	true	
	EP11			false	
	EP12		accept	true	
	EP13			false	
outputs {	EP14		return value	true	
	EP15			false	

5. Developing the Test Cases

- The significant difference in developing OO test cases is that, unlike where a single method was called in our previous EP example, multiple methods must be called for OO testing
- And they must be called in the correct order for the test to work
- Not all books show this level of detail in designing OO tests:
 - recommend this as good practice
 - Otherwise, when implementing the tests, the tester has to redo the analysis to work out the methods and the ordering required.

Selected OO EP Data Values

Method	Name	Equivalence Partition	Value
acceptOrder()	space	Integer.MIN_VALUE..0	-5000
		1..15	7
		16..1024	504
		1025..Integer.MAX_VALUE	5000

EP Test Case T1 (focus: constructor)

ID	TCI Covered	Inputs	Expected Return Value
----	-------------	--------	-----------------------

- As always, develop the test cases using uncovered TCI's in order

EP1	SpaceOrder()	isSpecial	true
EP2			false
EP3		special	true
EP4			false
EP5		accept	false
EP6*	acceptOrder()	space	Integer.MIN_VALUE..0
EP7			1..15
EP8			16..1024
EP9			1025..Integer.MAX_VALUE
EP10		special	true
EP11			false
EP12		accept	true
EP13			false
EP14		return value	true
EP15			false

EP Test Case T1 (focus: constructor)

ID	TCI Covered	Inputs	Expected Return Value
T1	EP1	new SpaceOrder(true)	

- First, call the constructor
- Here, we pass the value for special to the constructor (parameter name isSpecial)
 - EP1

EP1	SpaceOrder()	isSpecial	true
EP2			false
EP3		special	true
EP4			false
EP5		accept	false
EP6*	acceptOrder()	space	Integer.MIN_VALUE..0
EP7			1..15
EP8			16..1024
EP9			1025..Integer.MAX_VALUE
EP10		special	true
EP11			false
EP12		accept	true
EP13			false
EP14		return value	true
EP15			false

EP Test Case T1 (focus: constructor)

ID	TCI Covered	Inputs	Expected Return Value
T1	EP1 EP3	new SpaceOrder(true) getSpecial()	true

- Next, call getSpecial() to ensure special has been set correctly
 - Output TCI, EP3

EP1	SpaceOrder()	isSpecial	true
EP2			false
EP3		special	true
EP4			false
EP5		accept	false
EP6*	acceptOrder()	space	Integer.MIN_VALUE..0
EP7			1..15
EP8			16..1024
EP9			1025..Integer.MAX_VALUE
EP10		special	true
EP11			false
EP12		accept	true
EP13			false
EP14		return value	true
EP15			false

EP Test Case T1 (focus: constructor)

ID	TCI Covered	Inputs	Expected Return Value
T1	EP1 EP3 EP5	new SpaceOrder(true) getSpecial() getAccept()	true false

- Next, call getAccept() to ensure "accept" has been set correctly
 - Output TCI, EP5

EP1	SpaceOrder()	isSpecial	true
EP2			false
EP3		special	true
EP4			false
EP5		accept	false
EP6*	acceptOrder()	space	Integer.MIN_VALUE..0
EP7			1..15
EP8			16..1024
EP9			1025..Integer.MAX_VALUE
EP10		special	true
EP11			false
EP12		accept	true
EP13			false
EP14		return value	true
EP15			false

EP Test Case T1 (focus: constructor)

ID	TCI Covered	Inputs	Expected Return Value
T1	EP1 EP3 EP5	new SpaceOrder(true) getSpecial() getAccept()	true false

ORDERING ↓

- The required **sequence** of method calls is shown (in order)

EP Test Case T1 (focus: constructor)

ID	TCI Covered	Inputs	Expected Return Value
T1	EP1 EP3 EP5	new SpaceOrder(<u>true</u>) getSpecial() getAccept()	<u>true</u> <u>false</u>

- The required **sequence** of method calls is shown (in order)
- Each call includes the test data consisting of the input parameters and the expected return values

EP Test Case T1 (focus: constructor)

ID	TCI Covered	Inputs	Expected Return Value
T1	EP1 EP3 EP5	new <u>SpaceOrder</u> (true) getSpecial() getAccept()	true false

- The required **sequence** of method calls is shown (in order)
- Each call includes the test data consisting of the input parameters and the expected return values
- Normal practice to develop test cases for the constructor(s) first

EP Test Case T1 (focus: constructor)

ID	TCI Covered	Inputs	Expected Return Value
T1	EP1 EP3 EP5	new SpaceOrder(true) getSpecial() getAccept()	true false

- The required **sequence** of method calls is shown (in order)
- Each call includes the test data consisting of the input parameters and the expected return values
- Normal practice to develop test cases for the constructor(s) first
- Then for any accessor methods being tested

EP Test Case T1 (focus: constructor)

ID	TCI Covered	Inputs	Expected Return Value
T1	EP1 EP3 EP5	new SpaceOrder(true) getSpecial() getAccept()	true false

- The required **sequence** of method calls is shown (in order)
- Each call includes the test data consisting of the input parameters and the expected return values
- Normal practice to develop test cases for the constructor(s) first
- Then for any accessor methods being tested
- Finally for the other methods

EP Test Case T1 (focus: constructor)

ID	TCI Covered	Inputs	Expected Return Value
T1	EP1 EP3 EP5	new SpaceOrder(true) getSpecial() getAccept()	true false

- The required **sequence** of method calls is shown (in order)
- Each call includes test data: input parameters and expected return values
- Normal practice to develop test cases for the constructor(s) first
- Then for any accessor methods being tested
- Finally for the other methods
- Showing the test case identifier(s) covered **next to the relevant method call** makes reviewing the test cases easier

Test Case T1 Explained

ID	TCI Covered	Inputs	Expected Return Value
T1	EP1	new SpaceOrder(true)	
	EP3	getSpecial()	true
	EP5	getAccept()	false

- Line 1 calls the constructor
- This generates an instance of class SpaceOrder, providing the value true (EP1) as an input
- There is no (testable) return value from a constructor

Test Case T1 Explained

ID	TCI Covered	Inputs	Expected Return Value
T1	EP1	new SpaceOrder(true)	
	EP3	getSpecial()	true
	EP5	getAccept()	false

- Line 2 calls getSpecial()
- This returns the value of the attribute special
- This should have been set to true by the constructor if it is working correctly (which is EP3), so the expected return value is true


Test Case T1 Explained

ID	TCI Covered	Inputs	Expected Return Value
T1	EP1	new SpaceOrder(true)	
	EP3	getSpecial()	true
	EP5	getAccept()	false

- Line 3 calls getAccept()
- This returns the value of the attribute accept
- According to the specification, its default value at initialisation is false, so this is the return value for getAccept() that is expected (EP5)

Test Case T2

- Next we Consider EP2, which should produce EP4



ID	TCI Covered	Inputs	Expected Results Return Value
T1	EP1 EP3 EP5	new SpaceOrder(true) getSpecial() getAccept()	true false
T2	EP2 EP4	new SpaceOrder(false) getSpecial()	false

- We don't need to repeat the test for EP5

EP1	SpaceOrder()	isSpecial	true
EP2			false
EP3		special	true
EP4			false
EP5		accept	false
EP6*	acceptOrder()	space	Integer.MIN_VALUE..0
EP7			1..15
EP8			16..1024
EP9			1025..Integer.MAX_VALUE
EP10		special	true
EP11			false
EP12		accept	true
EP13			false
EP14		return value	true
EP15			false

Test Case T3 (focus: acceptOrder)

- Next we consider acceptOrder()
- Starting with EP7 and EP10 which should produce EP14 and EP12

ID	TCI Covered	Inputs	Expected Results Return Value
T1	EP1 EP3 EP5	new SpaceOrder(true) getSpecial() getAccept()	true false
T2	EP2 EP4	new SpaceOrder(false) getSpecial()	false
T3	[EP1]	new SpaceOrder(true)	



- Note: use isSpecial==true as testing for EP10 here (not EP3)

EP1	SpaceOrder()	isSpecial	true
EP2			false
EP3		special	true
EP4			false
EP5		accept	false
EP6*	acceptOrder()	space	Integer.MIN_VALUE..0
EP7			1..15
EP8			16..1024
EP9			1025..Integer.MAX_VALUE
EP10		special	true
EP11			false
EP12		accept	true
EP13			false
EP14		return value	true
EP15			false

Test Case T3 (focus: acceptOrder)

- Next we consider acceptOrder()
- Starting with EP7 and EP10 which should produce EP14 and EP12

ID	TCI Covered	Inputs	Expected Results Return Value
T1	EP1 EP3 EP5	new SpaceOrder(true) getSpecial() getAccept()	true false
T2	EP2 EP4	new SpaceOrder(false) getSpecial()	false
T3	[EP1] EP7,10,14	new SpaceOrder(true) acceptOrder(7)	true



EP1	SpaceOrder()	isSpecial	true
EP2			false
EP3		special	true
EP4			false
EP5		accept	false
EP6*	acceptOrder()	space	Integer.MIN_VALUE..0
EP7			1..15
EP8			16..1024
EP9			1025..Integer.MAX_VALUE
EP10		special	true
EP11			false
EP12		accept	true
EP13			false
EP14		return value	true
EP15			false

Test Case T3 (focus: acceptOrder)

- Next we consider acceptOrder()
- Starting with EP7 and EP10 which should produce EP14 and EP12

ID	TCI Covered	Inputs	Expected Results Return Value
T1	EP1 EP3 EP5	new SpaceOrder(true) getSpecial() getAccept()	true false
T2	EP2 EP4	new SpaceOrder(false) getSpecial()	false
T3	[EP1] EP7,10,14 EP12	new SpaceOrder(true) acceptOrder(7) getAccept()	true true



EP1	SpaceOrder()	isSpecial	true
EP2			false
EP3		special	true
EP4			false
EP5		accept	false
EP6*	acceptOrder()	space	Integer.MIN_VALUE..0
EP7			1..15
EP8			16..1024
EP9			1025..Integer.MAX_VALUE
EP10		special	true
EP11			false
EP12		accept	true
EP13			false
EP14		return value	true
EP15			false

Test Case T4

- EP8 and EP11 which should produce EP14 and EP12

ID	TCI Covered	Inputs	Expected Results Return Value
T1	EP1 EP3 EP5	new SpaceOrder(true) getSpecial() getAccept()	true false
T2	EP2 EP4	new SpaceOrder(false) getSpecial()	false
T3	[EP1] EP7,10,14 EP12	new SpaceOrder(true) acceptOrder(7) getAccept()	true true
T4	[EP2]	new SpaceOrder(false)	



EP1	SpaceOrder()	isSpecial	true
EP2			false
EP3		special	true
EP4			false
EP5		accept	false
EP6*	acceptOrder()	space	Integer.MIN_VALUE..0
EP7			1..15
EP8			16..1024
EP9			1025..Integer.MAX_VALUE
EP10		special	true
EP11			false
EP12		accept	true
EP13			false
EP14		return value	true
EP15			false

Test Case T4

- EP8 and EP11 which should produce EP14 and EP12

ID	TCI Covered	Inputs	Expected Results Return Value
T1	EP1 EP3 EP5	new SpaceOrder(true) getSpecial() getAccept()	true false
T2	EP2 EP4	new SpaceOrder(false) getSpecial()	false
T3	[EP1] EP7,10,14 EP12	new SpaceOrder(true) acceptOrder(7) getAccept()	true true
T4	[EP2] EP8,11,[14]	new SpaceOrder(false) acceptOrder(504)	true



EP1	SpaceOrder()	isSpecial	true
EP2			false
EP3		special	true
EP4			false
EP5		accept	false
EP6*	acceptOrder()	space	Integer.MIN_VALUE..0
EP7			1..15
EP8			16..1024
EP9			1025..Integer.MAX_VALUE
EP10		special	true
EP11			false
EP12		accept	true
EP13			false
EP14		return value	true
EP15			false

Test Case T4

- EP8 and EP11 which should produce EP14 and EP12

ID	TCI Covered	Inputs	Expected Results Return Value
T1	EP1 EP3 EP5	new SpaceOrder(true) getSpecial() getAccept()	true false
T2	EP2 EP4	new SpaceOrder(false) getSpecial()	false
T3	[EP1] EP7,10,14 EP12	new SpaceOrder(true) acceptOrder(7) getAccept()	true true
T4	[EP2] EP8,11,[14] [EP12]	new SpaceOrder(false) acceptOrder(504) getAccept()	true true



EP1	SpaceOrder()	isSpecial	true
EP2			false
EP3		special	true
EP4			false
EP5		accept	false
EP6*	acceptOrder()	space	Integer.MIN_VALUE..0
EP7			1..15
EP8			16..1024
EP9			1025..Integer.MAX_VALUE
EP10		special	true
EP11			false
EP12		accept	true
EP13			false
EP14		return value	true
EP15			false

Test Case T5

- EP9 and EP10 which should produce EP14 and EP13

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



EP1	SpaceOrder()	isSpecial	true
EP2			false
EP3		special	true
EP4			false
EP5		accept	false
EP6*	acceptOrder()	space	Integer.MIN_VALUE..0
EP7			1..15
EP8			16..1024
EP9			1025..Integer.MAX_VALUE
EP10		special	true
EP11			false
EP12		accept	true
EP13			false
EP14		return value	true
EP15			false

Test Case T5

- EP9 and EP10 which should produce EP14 and EP13

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



EP1	SpaceOrder()	isSpecial	true
EP2			false
EP3		special	true
EP4			false
EP5		accept	false
EP6*	acceptOrder()	space	Integer.MIN_VALUE..0
EP7			1..15
EP8			16..1024
EP9			1025..Integer.MAX_VALUE
EP10		special	true
EP11			false
EP12		accept	true
EP13			false
EP14		return value	true
EP15			false

Test Case T5

- EP9 and EP10 which should produce EP14 and EP13

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



EP1	SpaceOrder()	isSpecial	true
EP2			false
EP3		special	true
EP4			false
EP5		accept	false
EP6*	acceptOrder()	space	Integer.MIN_VALUE..0
EP7			1..15
EP8			16..1024
EP9			1025..Integer.MAX_VALUE
EP10		special	true
EP11			false
EP12		accept	true
EP13			false
EP14		return value	true
EP15			false

Test Case T6 (error TCIs)

- Next we consider the error partitions for acceptOrder()
- EP6* and EP2 which should produce EP15 and EP13

EP1	SpaceOrder()	isSpecial	true
EP2			false
EP3		special	true
EP4			false
EP5		accept	false
EP6*	acceptOrder()	space	Integer.MIN_VALUE..0
EP7			1..15
EP8			16..1024
EP9			1025..Integer.MAX_VALUE
EP10		special	true
EP11			false
EP12		accept	true
EP13			false
EP14		return value	true
EP15			false



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

Test Case T6 (error TCIs)

- Next we consider the error partitions for acceptOrder()
- EP6* and EP2 which should produce EP15 and EP13

EP1	SpaceOrder()	isSpecial	true
EP2			false
EP3		special	true
EP4			false
EP5		accept	false
EP6*	acceptOrder()	space	Integer.MIN_VALUE..0
EP7			1..15
EP8			16..1024
EP9			1025..Integer.MAX_VALUE
EP10		special	true
EP11			false
EP12		accept	true
EP13			false
EP14		return value	true
EP15			false



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

Test Case T6 (error TCIs)

- Next we consider the error partitions for acceptOrder()
- EP6* and EP2 which should produce EP15 and EP13

EP1	SpaceOrder()	isSpecial	true
EP2			false
EP3		special	true
EP4			false
EP5		accept	false
EP6*	acceptOrder()	space	Integer.MIN_VALUE..0
EP7			1..15
EP8			16..1024
EP9			1025..Integer.MAX_VALUE
EP10		special	true
EP11			false
EP12		accept	true
EP13			false
EP14		return value	true
EP15			false

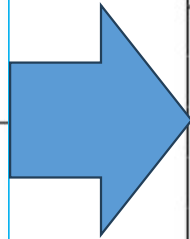


ID	Test Cases Covered	Inputs	Expected Results Return Value
T1	EP1 EP3 EP5	new SpaceOrder(true) getSpecial() getAccept()	true false
T2	EP2 EP4	new SpaceOrder(false) getSpecial()	false
T3	[EP1] EP7,10,14 EP12	new SpaceOrder(true) acceptOrder(7) getAccept()	true true
T4	[EP2] EP8,11,[14] [EP12]	new SpaceOrder(false)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

Developing All the Test Cases

- Approach systematically – cover the test coverage items in order
- Avoid unnecessary duplication
- Cover the error test coverage items last
- Counting the maximum number of test coverage items for each method being tested gives an indication of the minimum number of test cases expected
- A minimum of six test cases is expected
 - The constructor has a maximum of two (for isSpecial)
 - acceptOrder() has a maximum of four (for space)

Complete the TCI Table



ID	Test Cases Covered
T1	EP1 EP3 EP5
T2	EP2 EP4
T3	[EP1] EP7,10,14 EP12
T4	[EP2] EP8,11,[14] [EP12]
T5	[EP2] EP9,[10,14] EP13
T6*	[EP2] EP6,15 [EP13]

TCI	Method	Name	Equivalence Partition	Test Case
EP1	SpaceOrder()	isSpecial	true	T1
EP2			false	T2
EP3		special	true	T1
EP4			false	T2
EP5		accept	false	T1
EP6*	acceptOrder()	space	Integer.MIN_VALUE..0	T6
EP7			1..15	T3
EP8			16..1024	T4
EP9			1025..Integer.MAX_VALUE	T5
EP10		special	true	T3
EP11			false	T4
EP12		accept	true	T3
EP13			false	T5
EP14		return value	true	T3
EP15			false	T6

Reviewing Your Work

- All TCIs covered
- No (unnecessary) duplication

TCI	Method	Name	Equivalence Partition	Test Case
EP1	SpaceOrder()	isSpecial	true	T1
EP2			false	T2
EP3		special	true	T1
EP4			false	T2
EP5		accept	false	T1
EP6*	acceptOrder()	space	Integer.MIN_VALUE..0	T6
EP7			1..15	T3
EP8			16..1024	T4
EP9			1025..Integer.MAX_VALUE	T5
EP10		special	true	T3
EP11			false	T4
EP12		accept	true	T3
EP13			false	T5
EP14		return value	true	T3
EP15			false	T6

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

Test Implementation

(More detail in
following slides)

```
1 public class SpaceOrderTest {
2
3     @DataProvider(name="constructorData")
4     public Object[][] getConstructorData() {
5         return new Object[][] {
6             // TID isSpecial, e_special, e_accept
7             {"SpaceOrderTest T1", true, true, false},
8             {"SpaceOrderTest T2", false, false, false},
9         };
10    }
11
12    @Test(dataProvider="constructorData")
13    public void testConstructor(String tid, boolean isSpecial,
14                               boolean expectedSpecial, boolean expectedAccept) {
15        SpaceOrder o = new SpaceOrder(isSpecial);
16        assertEquals( o.getSpecial(), expectedSpecial );
17        assertEquals( o.getAccept(), expectedAccept );
18    }
19
20    @DataProvider(name="acceptOrderData")
21    public Object[][] getAcceptOrderData() {
22        return new Object[][] {
23            // TID special, space, e_rv e_accept
24            { "SpaceOrderTest T3", true, 7, true, true},
25            { "SpaceOrderTest T4", false, 504, true, true},
26            { "SpaceOrderTest T5", false, 5000, true, false},
27            { "SpaceOrderTest T6", false, -5000, false, false},
28        };
29    }
30
31    @Test(dataProvider="acceptOrderData",
32          dependsOnMethods={"testConstructor"})
33    public void testAcceptOrder(String tid, boolean special,
34                                int sqm, boolean expectedReturn, boolean
35                                expectedAccept) {
36        SpaceOrder o = new SpaceOrder(special);
37        assertEquals( o.acceptOrder(sqm), expectedReturn );
38        assertEquals( o.getAccept(), expectedAccept );
39    }
40 }
```

Notes

- The method testConstructor, on lines 12-17, uses parameterised data to implement test cases T1 and T2

```

1 public class SpaceOrderTest {
2
3     @DataProvider(name="constructorData")
4     public Object[][] getConstructorData() {

```

ID	Test Cases 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

```

16         assertEquals( o.getAccept(), expectedAccept );
17     }

```

```

3     @DataProvider(name="constructorData")
4     public Object[][] getConstructorData() {
5         return new Object[][] {
6             // TID isSpecial, e_special, e_accept
7             {"SpaceOrderTest T1", true, true, false},
8             {"SpaceOrderTest T2", false, false, false},
9         };
10    }
11
12    @Test(dataProvider="constructorData")
13    public void testConstructor(String tid, boolean isSpecial,
14                               boolean expectedSpecial, boolean expectedAccept) {
14        SpaceOrder o = new SpaceOrder(isSpecial);
15        assertEquals( o.getSpecial(), expectedSpecial );
16        assertEquals( o.getAccept(), expectedAccept );
17    }

```

```

37
38 }

```

Notes

- The method testConstructor, on lines 12-17, uses parameterised data to implement test cases T1 and T2
- The data is provided by the data provider named constructorData defined on lines 3-10

```

1 public class SpaceOrderTest {
2
3     @DataProvider(name="constructorData")
4     public Object[][] getConstructorData() {

```

ID	Test Cases 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

```

16         assertEquals( o.getAccept(), expectedAccept );
17     }

```

```

3     @DataProvider(name="constructorData")
4     public Object[][] getConstructorData() {
5         return new Object[][] {
6             // TID isSpecial, e_special, e_accept
7             {"SpaceOrderTest T1", true, true, false},
8             {"SpaceOrderTest T2", false, false, false},
9         };
10    }
11
12    @Test(dataProvider="constructorData")
13    public void testConstructor(String tid, boolean isSpecial,
14                               boolean expectedSpecial, boolean expectedAccept) {
14        SpaceOrder o = new SpaceOrder(isSpecial);
15        assertEquals( o.getSpecial(), expectedSpecial );
16        assertEquals( o.getAccept(), expectedAccept );
17    }

```

```

37
38 }

```

Notes

- The method `testAcceptOrder()`, on lines 30-36, uses parameterised data to implement test cases T3 to T6

T3	[EP1] EP7,10,14 EP12	new SpaceOrder(true) acceptOrder(7) getAccept()	true true	
T4	[EP2] EP8,11,[14] [EP12]	new SpaceOrder(false) acceptOrder(504) getAccept()	true true	special, e_accept true, false}, false, false},
T5	[EP2] EP9,[10,14] EP13	new SpaceOrder(false) acceptOrder(5000) getAccept()	true false	
T6*	[EP2] EP6,15 [EP13]	new SpaceOrder(false) acceptOrder(-5000) getAccept()	false false	olean isSpecial, ectedAccept) {

```

19      @DataProvider(name="acceptOrderData")
20      public Object[][] getAcceptOrderData() {
21          return new Object[][] {
22              //          TID    special, space, e_rv e_accept
23              { "SpaceOrderTest T3", true, 7, true, true},
24              { "SpaceOrderTest T4", false, 504, true, true},
25              { "SpaceOrderTest T5", false, 5000, true, false},
26              { "SpaceOrderTest T6", false, -5000, false, false},
27          };
28      }
29
30      @Test(dataProvider="acceptOrderData",
31            dependsOnMethods={"testConstructor"})
32      public void testAcceptOrder(String tid, boolean special,
33                                int sqm, boolean expectedReturn, boolean
34                                expectedAccept) {
35          SpaceOrder o = new SpaceOrder(special);
36          assertEquals( o.acceptOrder(sqm), expectedReturn );
37          assertEquals( o.getAccept(), expectedAccept );
38      }

```


Notes

- The method `testAcceptOrder()`, on lines 30-36, uses parameterised data to implement test cases T3 to T6
- The data provided by “acceptOrderData” defined on lines 19-28

T3	[EP1] EP7,10,14 EP12	new SpaceOrder(true) acceptOrder(7) getAccept()	true true	
T4	[EP2] EP8,11,[14] [EP12]	new SpaceOrder(false) acceptOrder(504) getAccept()	true true	special, e_accept true, false}, false, false},
T5	[EP2] EP9,[10,14] EP13	new SpaceOrder(false) acceptOrder(5000) getAccept()	true false	
T6*	[EP2] EP6,15 [EP13]	new SpaceOrder(false) acceptOrder(-5000) getAccept()	false false	olean isSpecial, ectedAccept) {

```

19      @DataProvider(name="acceptOrderData")
20      public Object[][] getAcceptOrderData() {
21          return new Object[][] {
22              //          TID    special, space, e_rv e_accept
23              { "SpaceOrderTest T3", true, 7, true, true},
24              { "SpaceOrderTest T4", false, 504, true, true},
25              { "SpaceOrderTest T5", false, 5000, true, false},
26              { "SpaceOrderTest T6", false, -5000, false, false},
27          };
28      }
29
30      @Test(dataProvider="acceptOrderData",
31             dependsOnMethods={"testConstructor"})
32      public void testAcceptOrder(String tid, boolean special,
33                                 int sqm, boolean expectedReturn, boolean
34                                 expectedAccept) {
35          SpaceOrder o = new SpaceOrder(special);
36          assertEquals( o.acceptOrder(sqm), expectedReturn );
37          assertEquals( o.getAccept(), expectedAccept );
38      }

```

Notes

- “Depends on”
testConstructor forces
the constructor tests
to run first – not strictly
necessary...

T3	[EP1] EP7,10,14 EP12	new SpaceOrder(true) acceptOrder(7) getAccept()	true true	
T4	[EP2] EP8,11,[14] [EP12]	new SpaceOrder(false) acceptOrder(504) getAccept()	true true	special, e_accept true, false}, false, false},
T5	[EP2] EP9,[10,14] EP13	new SpaceOrder(false) acceptOrder(5000) getAccept()	true false	
T6*	[EP2] EP6,15 [EP13]	new SpaceOrder(false) acceptOrder(-5000) getAccept()	false false	olean isSpecial, ectedAccept() {

```

19      @DataProvider(name="acceptOrderData")
20      public Object[][] getAcceptOrderData() {
21          return new Object[][] {
22              //          TID    special, space, e_rv e_accept
23              { "SpaceOrderTest T3", true, 7, true, true},
24              { "SpaceOrderTest T4", false, 504, true, true},
25              { "SpaceOrderTest T5", false, 5000, true, false},
26              { "SpaceOrderTest T6", false, -5000, false, false},
27          };
28      }
29
30      @Test(dataProvider="acceptOrderData",
31             dependsOnMethods={"testConstructor"})
32      public void testAcceptOrder(String tid, boolean special,
33                                  int sqm, boolean expectedReturn, boolean
34                                  expectedAccept) {
35          SpaceOrder o = new SpaceOrder(special);
36          assertEquals( o.acceptOrder(sqm), expectedReturn );
37          assertEquals( o.getAccept(), expectedAccept );
38      }

```

Call the Method Being Tested Once per Case

- The test code only makes a single call to the method under test, matching the test cases...
- ...unless absolutely necessary to call it more than once to perform the test
- This makes reviewing the test code easier,
- Also makes debugging easier if a test fails
- You could, in theory, put all the test cases into a single test method
- But this would be bad practice:
 - Harder to understand
 - If an early test case failed, then the others would not be executed

EP Test Results for Class SpaceOrder

```
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
=====
```

- All the tests have passed

Note on Default Constructor

- If the default constructor is used
(i.e. no constructor defined in the class)
- Then all (primitive data type) attributes
are set to their default values

Note on Default Constructor

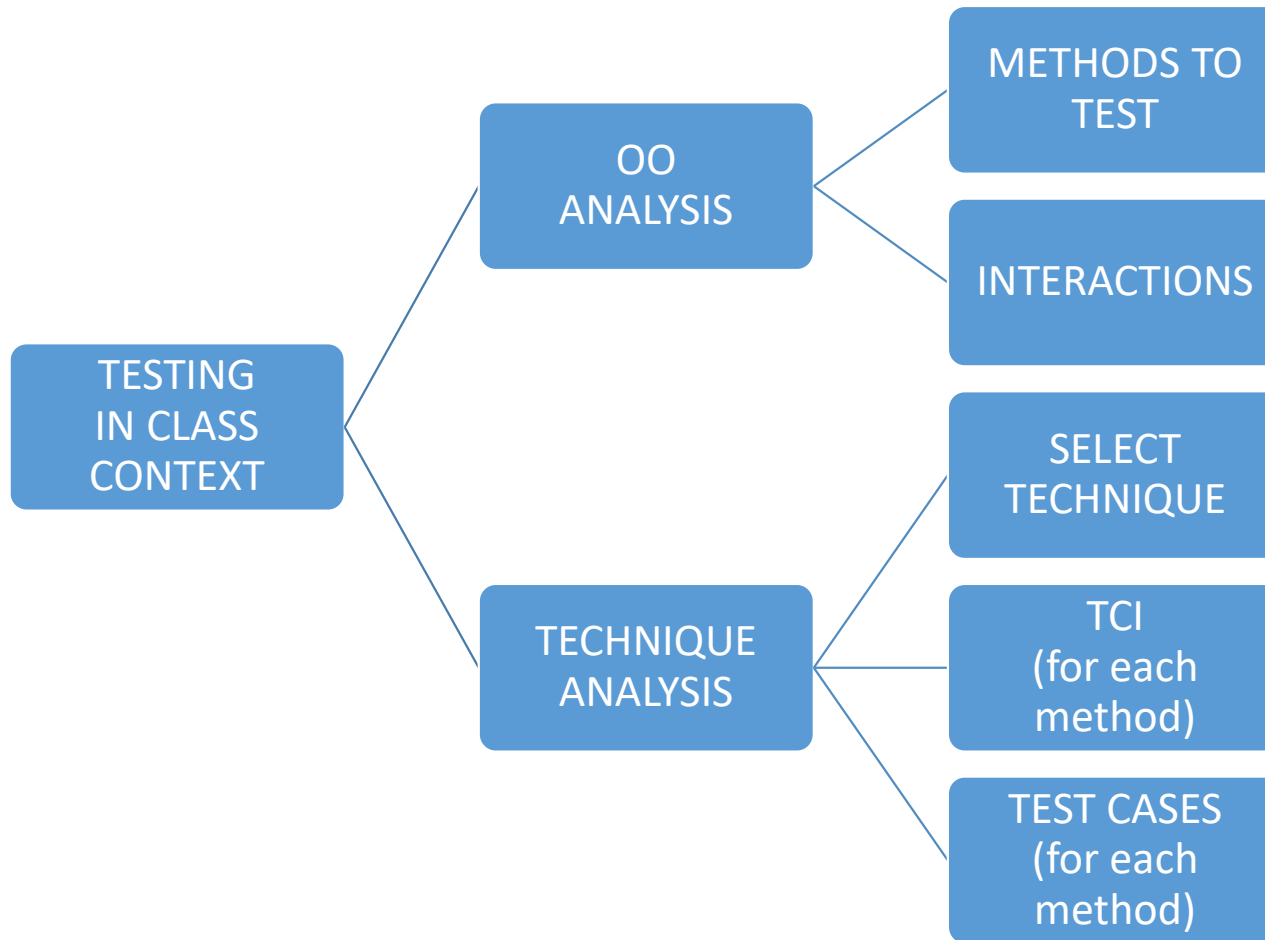
- If the default constructor is used (i.e. no constructor defined in the class)
- Then all (primitive data type) attributes are set to their default values
- Java defines these as follows (see [docs.oracle.com](https://docs.oracle.com/javase/7/docs/api/)):

Data Type	Default Value (for fields)
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	'\u0000'
String (or any object)	null
boolean	false

EP Testing “in Class Context”

- We have done class-wide testing
- In the context of a class
- Test Coverage Items (EPs) defined on a class-wide basis
 - For every method we are testing
- And the Test Cases call multiple methods
 - But call the method being tested just once

Testing in Class Context



Next Week

- Testing OO Software in More Detail
- Testing inheritance
- Testing sequences and state

This Afternoon

- Lab 6:
 - OOT – test method `SpaceOrder.acceptORder()` **in class context** with DT (combinations)
 - Deadline: next Monday lunchtime (but try and get it done today)
 - Follow the hints provided
- Work the problems on paper first
- Then implement, execute, and collect test results
- Assessment on Moodle
 - Quiz

Independent Study

- Read Chapters 5&6 (SC & BC in more detail)
- Read Chapter 9 (OOT), Sections 9.1-9.3
- Refresh your knowledge of UML
https://www.tutorialspoint.com/uml/uml_quick_guide.htm