Started on	Friday, 22 March 2019, 3:36 PM
State	Finished
Completed on	Saturday, 23 March 2019, 11:39 PM
Time taken	1 day 8 hours
Marks	6.37/7.00
Grade	**90.95** out of 100.00

Information

General Lab Information

Testing is important to make sure code behaves the way we expect it to, and continues to do so when altering other code that may affect it. It's very easy to break code in unexpected ways, and testing helps us to find these issues. Spaceman doesn't want any surprises once he's out there floating around, he needs to know that everything works the way that it should.

Manually testing code is slow and inaccurate. Automated testing frameworks allow tests to be run frequently, and their results to be quickly checked. JUnit is an automated unit testing framework for Java and it integrates nicely into Eclipse.

We will also look at Java exceptions, a language feature to help you both debug your code, and to help your code fix problems unattended on the fly. You have probably seen an exception before like "NullPointerException" or "ArrayIndexOutOfBoundsException"

The goals for this lab are to:

- Learn the basics of JUnit
- Learn what good tests look like
- Learn what to test and what not to test
- How to use JUnit tags
- Learn about Java exceptions and stack traces
- Learn how to catch and process exceptions
- Learn how to throw your own exceptions
- Learn how to make your own exception

Response history

Step	Time	Action	State
1	22/03/19, 15:36	Started	
2	23/03/19, 22:10	Seen	
3	**23/03/19, 23:39**	**Attempt finished**	

Information

Writing Good Unit Tests

A unit test tests one distinct unit of code, such as a function or a class. Each unit test should only test one thing. This helps to keep your test classes simple and understandable. If one test is testing five different things, it's hard to know which is at fault when it fails. A good way to come up with tests is using **Equivalence Partitioning** and **Boundary Value Analysis**.

Response history

Step	Time	Action	State
1	22/03/19, 15:36	Started	
2	23/03/19, 22:10	Seen	
3	**23/03/19, 23:39**	**Attempt finished**	

Information

Baby's First JUnit Test

JUnit is an automated unit testing framework for Java. JUnit is built into Eclipse, allowing you to create, run, and see results of tests within Eclipse. A JUnit test class is a class that tests a single class within your project. We're going to create a JUnit test class for the RocketShip class.

```
public class RocketShip {

    int MAX_FUEL_LEVEL = 100;

    int fuelLevel;
    int currentHeight;

    public RocketShip(int fuelLevel) {
        this.fuelLevel = fuelLevel;
        this.currentHeight = 0;
    }

    public int getFuelLevel() {
        return fuelLevel;
    }

    public int getCurrentHeight() {
        return currentHeight;
    }

    public void fuelUp(int fuelAmount) {
        fuelLevel += fuelAmount;
    }

    public void takeOff() {
            fuelLevel -= 20;
            currentHeight += 20;
    }

    public void goHigher() {
        fuelLevel -= 10;
        currentHeight += 50;
    }

    public void goLower() {
        currentHeight -= 50;
    }

    public void land() {
        currentHeight = 0;
    }
}
```

Adding a JUnit test class:

1. Create a Java project with the RocketShip class above
2. Open source (src) > main to find the RocketShip class. If you do not have a package named main, create one in your source folder now and move the RocketShip class into it
3. Create a new package called test at the same level as main i.e. in the source folder. (any test classes should follow the same structure as the classes in main, but within the test package)
4. Right click on the test package and select 'New' > 'JUnit Test Case'
5. Name this class 'RocketShipTest' (as this class will test the methods within the RocketShip class)
6. If prompted to add JUnit to the class path or build path, click yes, and select JUnit 5 if given a choice (you should only need to do this once per project)

A test class will be created along with a sample test case. It should look something like this:

```
package test;

import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;

class RocketShipTest {

        @Test
        void test() {
                fail("Not yet implemented");
        }
}
```

JUnit Basics

The general idea of a unit test is to test one unit of your code. For example, in the Rocketship class (download above), there is a method fuelUp(int fuelAmount). This is the unit of code that we are going to write a unit test for.

Essentially, with a unit test, we want to run some of our code, and check that the outcome is what we expected. When we call the fuelUp() method, we expect that the rocketships fuel level increases by the amount that we give it, but will not go above the maximum fuel level of 100. Here's a unit test for this method:

```
@Test
public void fuelUpTest() {
    RocketShip testRocketShip = new RocketShip(50);
    testRocketShip.fuelUp(30);
    assertEquals(80, testRocketShip.getFuelLevel());
}
```

- We initialised a rocketship with fuel level 50
- Then we called the fuelUp() method, passing in 30 as the amount of fuel to add.
- Finally, we checked the fuel level of the rocketship using an assert method (we'll look at this next), to make sure that it is what we expected (80)

JUnit Assertions

In the test above, we tested that the fuel was what we expected using assertEquals(). JUnit provides a number of different assert statements that are used for different situations. Here are a few of the main ones:

- assertEquals() - check that two values passed in are equal
- assertArrayEquals() - check that two arrays passed in are equal
- assertTrue()/assertFalse() - ensure that the expression passed in evaluates to True or False

You can find more information on assert in the Java User Guide (assert methods)

JUnit Annotations:

You may have noticed that on the test we just created, there is a @Test annotation. JUnit has a number of annotations that affect how and when the methods they annotate are executed:

- @Test - signifies a test
- @BeforeAll - the method will be executed once before the entire test class
- @BeforeEach - the method is executed before each test in the class
- @AfterAll - the method is executed after all tests in the class have been run
- @AfterEach - the method is executed after each test in the class

These are the main tags that you will need, however there are a number of other handy tags available, which you can find in the JUnit User Guide (Annotations).

Response history

Step	Time	Action	State
1	22/03/19, 15:36	Started	
2	23/03/19, 22:10	Seen	
3	**23/03/19, 23:39**	**Attempt finished**	

Question **1**
Correct
Mark 1.00 out of 1.00

Match the JUnit annotations below with their uses:

@BeforeEach	Executed before each test	✓
@DisplayName	Provide a description of the test case that is shown when it is run	✓
@Disabled	Used to stop a certain test from being executed	✓
@ParameterizedTest	The test is run multiple times with different arguments	✓

Your answer is correct.

The correct answer is: @BeforeEach → Executed before each test, @DisplayName → Provide a description of the test case that is shown when it is run, @Disabled → Used to stop a certain test from being executed, @ParameterizedTest → The test is run multiple times with different arguments

Correct
Marks for this submission: 1.00/1.00.

Response history

Step	Time	Action	State	Marks
1	22/03/19, 15:36	Started	Not complete	
2	23/03/19, 22:10	Submit: @BeforeEach -> Executed before each test; @DisplayName -> Provide a description of the test case that is shown when it is run; @Disabled -> Used to stop a certain test from being executed; @ParameterizedTest -> The test is run multiple times with different arguments	Correct	1.00
3	**23/03/19, 23:39**	**Attempt finished**	**Correct**	**1.00**

Question **2**

Correct

Mark 0.67 out of 1.00

Annotation Count

The following test class tests a class called Spaceman.

```
class SpacemanTest {
  private Spaceman testSpaceman;

  @BeforeEach
  public init() {
        testSpaceman = new Spaceman ("Spacedude", 25);
  }

  @Test
  public ageUpTest() {
    testSpaceman.ageUp();
    assertEqual(26, testSpaceman.getAge();
  }

  @Test
  public spacemanSaysTest() {
        String spaceText = testSpaceman.spacemanSays("hello world");
    assertEqual("Spacedude says: hello world", spaceText);
  }
}
```

How many times are each of the methods of the SpacemanTest class executed when the class is run?

ageUpTest()	1 ✓
init()	2 ✓
spacemanSaysTest()	1 ✓

Your answer is correct.

The correct answer is: ageUpTest() → 1, init() → 2, spacemanSaysTest() → 1

Correct

Marks for this submission: 1.00/1.00. Accounting for previous tries, this gives **0.67/1.00**.

Response history

Step	Time	Action	State	Marks
1	22/03/19, 15:36	Started	Not complete	
2	23/03/19, 22:20	Submit: ageUpTest() -> 1; init() -> 3; spacemanSaysTest() -> 1	Partially correct	0.67
3	23/03/19, 22:20	Submit: ageUpTest() -> 1; init() -> 2; spacemanSaysTest() -> 1	Correct	0.67
4	**23/03/19, 23:39**	**Attempt finished**	**Correct**	**0.67**

Question **3**

Correct

Mark 1.00 out of 1.00

Testing for takeoff

Using your RocketShipTest file, add another test method called takeOffTest which creates a RocketShip with 50 fuel and then calls the takeOff method.
After these calls, check (assert) that the fuelLevel is 30, and that the currentHeight is 20.
Try editing your RocketShip's actual takeOff method to make sure that your tests fail if the method doesn't match these expectations.

Note: You don't need to include any imports for this question.

Answer: (penalty regime: 0, 10, 20, ... %)

```
class RocketShipTest {
    private RocketShip testRocketShip = new RocketShip(50);

    @Test
    public void takeOffTest() {
        testRocketShip.takeOff();
        assertEquals(30, testRocketShip.getFuelLevel());
        assertEquals(20, testRocketShip.getCurrentHeight());
    }
}
```

	Test	Expected	Got	

✓	`/* Using the following (correct) implementation of the takeOff method in the RocketShip class`
	`public void takeOff() {`
	`    fuelLevel -= 20;`
	`    currentHeight = 20;`
	`}`
	`*/`
	
	`RocketShipTest test = new RocketShipTest();`
	`test.takeOffTest();`
	`System.out.println("Test passed");`
		`Test passed`	`Test passed`	✓

✓	`/* Using the following (incorrect) implementation of the takeOff method in the RocketShip class`
	`public void takeOff() {`
	`    fuelLevel -= 20;`
	`}`
	`*/`
	
	`RocketShipTest test = new RocketShipTest();`
	`try {`
	`    test.takeOffTest();`
	`    System.out.println("Test passed but it should have failed");`
	`} catch (AssertionFailedError e) {`
	`    System.out.println(e.getMessage());`
	`}`
		`Expected: <20> but was: <0>`	`Expected: <20> but was: <0>`	✓

	Test	Expected	Got	

✓	`/* Using the following (incorrect) implementation of the takeOff method in the RocketShip class`
	`public void takeOff() {`
	`    currentHeight = 20;`
	`}`
	`*/`
	
	`RocketShipTest test = new RocketShipTest();`
	`try {`
	`    test.takeOffTest();`
	`    System.out.println("Test passed but it should have failed");`
	`} catch (AssertionFailedError e) {`
	`    System.out.println(e.getMessage());`
	`}`
		`Expected: <30> but was: <50>`	`Expected: <30> but was: <50>`	✓

Passed all tests! ✓

Question author's solution:

```
import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.Test;

public class RocketShipTest {

    @Test
    public void takeOffTest() {
        RocketShip ship = new RocketShip(50);
        ship.takeOff();
        assertEquals(30, ship.getFuelLevel());
        assertEquals(20, ship.getCurrentHeight());
    }
}
```

Correct

Marks for this submission: 1.00/1.00.

Response history

Step	Time	Action	State	Marks
1	22/03/19, 15:36	Started	Not complete	
2	23/03/19, 22:44	Submit: class RocketShipTest { private RocketShip testRocketShip; @BeforeEach public void init() { testRocketShip = new RocketShip(50); } @Test public void test() { fail("Not yet implemented"); } @Test public void fuelUpTest() { testRocketShip.fuelUp(30); assertEquals(80, testRocketShip.getFuelLevel()); } @Test public void takeOffTest() { testRocketShip.takeOff(); assertEquals(30, testRocketShip.getFuelLevel()); assertEquals(20, testRocketShip.getCurrentHeight()); } }	Incorrect	0.00
3	23/03/19, 22:45	Submit: class RocketShipTest { private RocketShip testRocketShip = new RocketShip(50); @Test public void takeOffTest() { testRocketShip.takeOff(); assertEquals(30, testRocketShip.getFuelLevel()); assertEquals(20, testRocketShip.getCurrentHeight()); } }	Correct	1.00
4	**23/03/19, 23:39**	**Attempt finished submitting:**	**Correct**	**1.00**

Information

Testing the RocketShip class

Have a go at testing the rest of the methods in the RocketShip class. Remember to test more than just the blue sky scenario. What happens if we try to add negative fuel with fuelUp(), is this correct behavior? Generally it is unnecessary to test getters and setters, unless they do some kind of validation or alter the data in some way.

Response history

Step	Time	Action	State
1	22/03/19, 15:36	Started	
2	23/03/19, 22:10	Seen	
3	**23/03/19, 23:39**	**Attempt finished**	

Information

Java Exceptions

An exception is "an event that occurs during the execution of a program that disrupts the normal flow of instructions." Here are some situations that cause exceptions:

- referencing a null pointer
- out of bounds in an array
- division by zero

When an error like above occurs, an exception is thrown, the code that caused the exception is stopped, and control moves to another part of the code that deals with the exception. Java has a number of built in exceptions (though you can also make your own custom exceptions). Here are a few:

- IllegalArgumentException
- NullPointerException
- ArrayIndexOutOfBoundsException

Response history

Step	Time	Action	State
1	22/03/19, 15:36	Started	
2	23/03/19, 22:57	Seen	
3	**23/03/19, 23:39**	**Attempt finished**	

Information

Throwing Exceptions

Let's have another look at the RocketShip class, in particular the fuelUp() method. What happens if a negative integer is passed in as fuelAmount?

```
public void fuelUp(int fuelAmount) {
    fuelLevel = fuelLevel + fuelAmount;
}
```

If you haven't guessed, what happens is that the fuelLevel goes down. That's not right! We want to make sure that the value passed in for fuelAmount is positive, and if it's not, then we can throw an exception. The built in exception that makes the most sense for this situation is IllegalArgumentException.

So how do I throw an exception, you ask? Like this:

```
throw new IllegalArgumentException("Fuel amount must be positive for fueling up");
```

This creates a new exception, with the error message that we passed in. So now the fuelUp() method looks like this:

```
public void fuelUp(int fuelAmount) {
    if(fuelAmount > 0) {
        fuelLevel += fuelAmount;
    } else {
        throw new IllegalArgumentException("Fuel amount must be positive for fueling up");
    }
}
```

Response history

Step	Time	Action	State
1	22/03/19, 15:36	Started	
2	23/03/19, 22:57	Seen	
3	**23/03/19, 23:39**	**Attempt finished**	

Question **4**

Correct

Mark 1.00 out of 1.00

Starman takes off! (throwing exceptions)

Starman is ready for his big adventure in his shiny new rocketship. He's got his towel, some snacks, and a handy guide to help him on his way. He pushes the big green button labelled "Take Off!" and his rocketship makes a loud screech and ... nothing. Huh? There's no fuel! The rocketship shouldn't have tried to take off with no fuel.

Update the takeOff() method of the Rocketship class to only take off if there is at least 20 units of fuel. If not, throw an IllegalStateException with the error message "Not enough fuel!"

Remember to include the whole Rocketship class in your answer.

Answer: (penalty regime: 0, 10, 20, ... %)

```
public class RocketShip {

    int MAX_FUEL_LEVEL = 100;

    int fuelLevel;
    int currentHeight;

    public RocketShip(int fuelLevel) {
        this.fuelLevel = fuelLevel;
        this.currentHeight = 0;
    }

    public int getFuelLevel() {
        return fuelLevel;
    }

    public int getCurrentHeight() {
        return currentHeight;
    }

    public void fuelUp(int fuelAmount) {
```

	Test	Expected	Got	

✓	`RocketShip testRocketShip = new RocketShip(19);`
	`try {`
	`    testRocketShip.takeOff();`
	`    System.out.println("Uh oh! Exception not thrown");`
	`} catch (IllegalStateException e) {`
	`    System.out.println(e.getMessage());`
	`}`
		Not enough fuel!	Not enough fuel!	✓

✓	`RocketShip testRocketShip = new RocketShip(20);`
	`try {`
	`    testRocketShip.takeOff();`
	`    System.out.println("Take off!");`
	`} catch (IllegalStateException e) {`
	`    System.out.println(e.getMessage());`
	`}`
		Take off!	Take off!	✓

Passed all tests! ✓

Question author's solution:

```
public class RocketShip {

    int MAX_FUEL_LEVEL = 100;

    int fuelLevel;
    int currentHeight;

    RocketShip(int fuelLevel) {
        this.fuelLevel = fuelLevel;
        currentHeight = 0;
    }

    public int getFuelLevel() {
        return fuelLevel;
    }

    public void setFuelLevel(int fuelLevel) {
        this.fuelLevel = fuelLevel;
    }

    public int getCurrentHeight() {
        return currentHeight;
    }

    public void setCurrentHeight(int currentHeight) {
        this.currentHeight = currentHeight;
    }

    public void fuelUp(int fuelAmount) {
        fuelLevel += fuelAmount;
    }

    public void takeOff() {
        if (fuelLevel < 20) {
            throw new IllegalStateException("Not enough fuel!");
        }
        fuelLevel -= 20;
        currentHeight += 20;
    }

    public void goHigher() {
        fuelLevel -= 10;
        currentHeight += 50;
    }

    public void land() {
        currentHeight = 0;
    }
}
```

Correct

Marks for this submission: 1.00/1.00.

Response history

Step	Time	Action	State	Marks
1	22/03/19, 15:36	Started	Not complete	
2	23/03/19, 22:57	Submit: public class RocketShip { int MAX_FUEL_LEVEL = 100; int fuelLevel; int currentHeight; public RocketShip(int fuelLevel) { this.fuelLevel = fuelLevel; this.currentHeight = 0; } public int getFuelLevel() { return fuelLevel; } public int getCurrentHeight() { return currentHeight; } public void fuelUp(int fuelAmount) { fuelLevel += fuelAmount; } public void takeOff() { if (fuelLevel >= 20) { fuelLevel -= 20; currentHeight += 20; } else throw new IllegalStateException("Not enough fuel!"); } public void goHigher() { fuelLevel -= 10; currentHeight += 50; } public void goLower() { currentHeight -= 50; } public void land() { currentHeight = 0; } }	Correct	1.00
3	**23/03/19, 23:39**	**Attempt finished submitting:**	**Correct**	**1.00**

Information

Custom Exceptions

What if we think IllegalStateException doesn't quite describe the situation? Maybe IllegalStateException is used a lot throughout our code and we want to differentiate different kinds of errors. This is where custom exceptions come in. We can create our own exception classes by subclassing existing any Throwable class.

```
public class InsufficientFuelException extends IllegalStateException {

    public InsufficientFuelException() {}

    public InsufficientFuelException(String message) {
        super(message);
    }
}
```

Here we chose to make the new exception class, InsufficientFuelException, a subclass of IllegalStateException. There are two constructors, so that the exception can be instantiated with or without an error message.

Add this class to your code and modify the takeOff method to throw this instead of IllegalStateException, as we'll want to check for this specific one in later questions!

Response history

Step	Time	Action	State
1	22/03/19, 15:36	Started	
2	23/03/19, 22:57	Seen	
3	**23/03/19, 23:39**	**Attempt finished**	

Question **5**

Correct

Mark 0.90 out of 1.00

Landing safely (making your own exceptions)

Starman wants to make sure his rocketship always lands safely, and make sure that it doesn't try to land from too high up. Modify the land() method to use a new LandingException for if the conditions are not right to land.

- Create a LandingException class that extends IllegalStateException.
- Alter the land() method in the Rocketship class, throw a LandingException with the message "Too high to land!" if the currentHeight is above 20.

Remember to include the entire Rocketship class in your answer, as well as the class for your new custom exception.

Answer: (penalty regime: 0, 10, 20, ... %)

```java
public class RocketShip {

    int MAX_FUEL_LEVEL = 100;

    int fuelLevel;
    int currentHeight;

    public RocketShip(int fuelLevel) {
        this.fuelLevel = fuelLevel;
        this.currentHeight = 0;
    }

    public int getFuelLevel() {
        return fuelLevel;
    }

    public int getCurrentHeight() {
        return currentHeight;
    }

    public void fuelUp(int fuelAmount) {
```

	Test	Expected	Got	

✓	`RocketShip testRocketShip = new RocketShip(50);`
	`try {`
	`    testRocketShip.takeOff();`
	`    testRocketShip.goHigher();`
	`    testRocketShip.goHigher();`
	`    testRocketShip.goHigher();`
	`    testRocketShip.land();`
	`} catch (LandingException e) {`
	`    System.out.println(e.getMessage());`
	`}`
		`Too high to land!`	`Too high to land!`	✓

✓	`RocketShip testRocketShip = new RocketShip(50);`
	`testRocketShip.takeOff();`
	`testRocketShip.land();`
	`System.out.println("Landed!");`
		`Landed!`	`Landed!`	✓

✓	`RocketShip testRocketShip = new RocketShip(50);`
	`try {`
	`    testRocketShip.takeOff();`
	`    testRocketShip.goHigher();`
	`    testRocketShip.goHigher();`
	`    testRocketShip.goHigher();`
	`    testRocketShip.land();`
	`} catch (IllegalStateException e) {`
	`    System.out.println(e.getMessage());`
	`}`
		`Too high to land!`	`Too high to land!`	✓

Passed all tests! ✓

Question author's solution:

```
public class LandingException extends IllegalStateException {
    public LandingException() {
        super();
    }

    public LandingException(String message) {
        super(message);
    }

}

public class RocketShip {

    int MAX_FUEL_LEVEL = 100;

    int fuelLevel;
    int currentHeight;

    RocketShip(int fuelLevel) {
        this.fuelLevel = fuelLevel;
        currentHeight = 0;
    }

    public int getFuelLevel() {
        return fuelLevel;
    }

    public void setFuelLevel(int fuelLevel) {
        this.fuelLevel = fuelLevel;
    }

    public int getCurrentHeight() {
        return currentHeight;
    }

    public void setCurrentHeight(int currentHeight) {
        this.currentHeight = currentHeight;
    }

    public void fuelUp(int fuelAmount) {
        fuelLevel += fuelAmount;
    }

    public void takeOff() {
        fuelLevel -= 20;
        currentHeight = 20;
    }

    public void goHigher() {
        fuelLevel -= 10;
        currentHeight += 50;
    }

    public void land() {
        if (currentHeight > 20) {
            throw new LandingException("Too high to land!");
        }
        currentHeight = 0;
    }
}
```

Correct

Marks for this submission: 1.00/1.00. Accounting for previous tries, this gives **0.90/1.00**.

Response history

Step	Time	Action	State	Marks
1	22/03/19, 15:36	Started	Not complete	
2	23/03/19, 23:07	Submit: public class RocketShip { int MAX_FUEL_LEVEL = 100; int fuelLevel; int currentHeight; public RocketShip(int fuelLevel) { this.fuelLevel = fuelLevel; this.currentHeight = 0; } public int getFuelLevel() { return fuelLevel; } public int getCurrentHeight() { return currentHeight; } public void fuelUp(int fuelAmount) { fuelLevel += fuelAmount; } public void takeOff() { if (fuelLevel >= 20) { fuelLevel -= 20; currentHeight += 20; } else throw new IllegalStateException("Not enough fuel!"); } public void goHigher() { fuelLevel -= 10; currentHeight += 50; } public void goLower() { currentHeight -= 50; } public void land() { if (currentHeight > 20) currentHeight = 0; else throw new LandingException("Too high to land!"); } } class LandingException extends IllegalStateException { public LandingException() {} public LandingException(String message) { super(message); } }	Incorrect	0.00
3	23/03/19, 23:10	Submit: public class RocketShip { int MAX_FUEL_LEVEL = 100; int fuelLevel; int currentHeight; public RocketShip(int fuelLevel) { this.fuelLevel = fuelLevel; this.currentHeight = 0; } public int getFuelLevel() { return fuelLevel; } public int getCurrentHeight() { return currentHeight; } public void fuelUp(int fuelAmount) { fuelLevel += fuelAmount; } public void takeOff() { if (fuelLevel >= 20) { fuelLevel -= 20; currentHeight += 20; } else throw new IllegalStateException("Not enough fuel!"); } public void goHigher() { fuelLevel -= 10; currentHeight += 50; } public void goLower() { currentHeight -= 50; } public void land() { if (currentHeight > 20) throw new LandingException("Too high to land!"); else currentHeight = 0; } } class LandingException extends IllegalStateException { public LandingException() {} public LandingException(String message) { System.out.print(message); } }	Incorrect	0.00
4	23/03/19, 23:17	Submit: public class RocketShip { int MAX_FUEL_LEVEL = 100; int fuelLevel; int currentHeight; public RocketShip(int fuelLevel) { this.fuelLevel = fuelLevel; this.currentHeight = 0; } public int getFuelLevel() { return fuelLevel; } public int getCurrentHeight() { return currentHeight; } public void fuelUp(int fuelAmount) { fuelLevel += fuelAmount; } public void takeOff() { if (fuelLevel >= 20) { fuelLevel -= 20; currentHeight += 20; } else throw new IllegalStateException("Not enough fuel!"); } public void goHigher() { fuelLevel -= 10; currentHeight += 50; } public void goLower() { currentHeight -= 50; } public void land() { if (currentHeight > 20) throw new LandingException("Too high to land!"); else currentHeight = 0; } } class LandingException extends IllegalStateException { public LandingException() {} public LandingException(String message) { super(message); } }	Correct	0.90
5	**23/03/19, 23:39**	**Attempt finished submitting:**	**Correct**	**0.90**

Information

Catching Exceptions

So now we can throw our own exceptions, but what then?

When an exception is thrown, we want to catch it, and then decide what to do. Generally it is a good idea to print the error message for caught exceptions so you can see what has gone wrong. You may also take other actions dependent on the situation.

To catch exceptions, we wrap the code that may cause an exception in a try/catch block, like this:

```
try {
    // code that may cause an IllegalArgumentException
} catch (IllegalArgumentException e) {
     // do something
}
```

In the catch block, you can get the error message for the exception thrown using e.getMessage(), which ideally will give you more information about what caused the error.

This try/catch block will only catch IllegalArgumentException exceptions, as this is the type that is specified by the catch block. This means that if a different exception is thrown by the code in the try block, this will not be caught here.

Response history

Step	Time	Action	State
1	22/03/19, 15:36	Started	
2	23/03/19, 23:27	Seen	
3	**23/03/19, 23:39**	**Attempt finished**	

Question **6**

Correct

Mark 0.80 out of 1.00

Test Flight (catching exceptions)

Starman wants to be able to check that his rocketship is in good shape before going on long adventures.

Add a testFlight method to the RocketShip class. The method should:

- Take off using the takeOff() method
- Increase altitude once using the goHigher() method
- Decrease altitude using the goLower() method
- Land back safely using the land() method

So what happens if the rocketship doesn't have enough fuel to take off when the testFlight() method is called? An exception will be thrown! So now you need to be ready to catch the exception if it is thrown, by using a try/catch block around the code inside the method. If an InsufficientFuelException is caught, then print out it's error message which you can get with exception.getMessage().

Remember to include the RocketShip class, and the classes of any custom exceptions used by your RocketShip class.

For example:

Test	Result

`RocketShip testRocketShip = new RocketShip(70);`
`testRocketShip.testFlight();`
 	
`System.out.println(testRocketShip.getCurrentHeight());`
`System.out.println(testRocketShip.getFuelLevel());`
	`0`
	`40`

Answer: (penalty regime: 0, 10, 20, ... %)

```java
public class RocketShip {

    int MAX_FUEL_LEVEL = 100;

    int fuelLevel;
    int currentHeight;

    public RocketShip(int fuelLevel) {
        this.fuelLevel = fuelLevel;
        this.currentHeight = 0;
    }

    public int getFuelLevel() {
        return fuelLevel;
    }

    public int getCurrentHeight() {
        return currentHeight;
    }

    public void fuelUp(int fuelAmount) {
        fuelLevel += fuelAmount;
```

	Test	Expected	Got	

✓	`RocketShip testRocketShip = new RocketShip(0);`
	`try {`
	`    testRocketShip.testFlight();`
	`    System.out.println(testRocketShip.getCurrentHeight());`
	`} catch (InsufficientFuelException e) {`
	`    System.out.println("Oops! You didn't catch the exception");`
	`}`
		`Not enough fuel!`
		`0`
			`Not enough fuel!`
			`0`
				✓

✓	`RocketShip testRocketShip = new RocketShip(70);`
	`testRocketShip.testFlight();`
	
	`System.out.println(testRocketShip.getCurrentHeight());`
	`System.out.println(testRocketShip.getFuelLevel());`
		`0`
		`40`
			`0`
			`40`
				✓

Passed all tests! ✓

Question author's solution:

```
public class InsufficientFuelException extends IllegalStateException {
    public InsufficientFuelException() {}

    public InsufficientFuelException(String message) {
        super(message);
    }
}

class RocketShip {

    int MAX_FUEL_LEVEL = 100;

    int fuelLevel;
    int currentHeight;

    RocketShip(int fuelLevel) {
        this.fuelLevel = fuelLevel;
        this.currentHeight = 0;
    }

    public int getFuelLevel() {
        return fuelLevel;
    }

    public void setFuelLevel(int fuelLevel) {
        this.fuelLevel = fuelLevel;
    }

    public int getCurrentHeight() {
        return currentHeight;
    }

    public void setCurrentHeight(int currentHeight) {
        this.currentHeight = currentHeight;
    }

    public void fuelUp(int fuelAmount) {
        this.fuelLevel += fuelAmount;
    }

    public void takeOff() {
        if (this.fuelLevel >= 20) {
            this.fuelLevel -= 20;
            this.currentHeight += 20;
        } else {
            throw new InsufficientFuelException("Not enough fuel!");
        }

    }

    public void goHigher() {
        this.fuelLevel -= 10;
        this.currentHeight += 50;
    }

    public void goLower() {
        this.currentHeight -= 50;
    }

    public void land() {
        this.currentHeight = 0;
    }

    public void testFlight() {
        try {
            this.takeOff();
            this.goHigher();
            this.goLower();
            this.land();
        } catch (InsufficientFuelException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Correct

Marks for this submission: 1.00/1.00. Accounting for previous tries, this gives **0.80/1.00**.

Response history

Step	Time	Action	State	Marks
1	22/03/19, 15:36	Started	Not complete	
2	23/03/19, 23:27	Submit: public class RocketShip { int MAX_FUEL_LEVEL = 100; int fuelLevel; int currentHeight; public RocketShip(int fuelLevel) { this.fuelLevel = fuelLevel; this.currentHeight = 0; } public int getFuelLevel() { return fuelLevel; } public int getCurrentHeight() { return currentHeight; } public void fuelUp(int fuelAmount) { fuelLevel += fuelAmount; } public void takeOff() { if (fuelLevel >= 20) { fuelLevel -= 20; currentHeight += 20; } else throw new IllegalStateException("Not enough fuel!"); } public void goHigher() { fuelLevel -= 10; currentHeight += 50; } public void goLower() { currentHeight -= 50; } public void land() { if (currentHeight > 20) throw new LandingException("Too high to land!"); else currentHeight = 0; } public void testFlight() { try { takeOff(); goHigher(); goLower(); land(); } catch (LandingException ex) { System.out.print(ex.getMessage()); } } } class LandingException extends IllegalStateException { public LandingException() {} public LandingException(String message) { super(message); } }	Incorrect	0.00
3	23/03/19, 23:31	Submit: public class RocketShip { int MAX_FUEL_LEVEL = 100; int fuelLevel; int currentHeight; public RocketShip(int fuelLevel) { this.fuelLevel = fuelLevel; this.currentHeight = 0; } public int getFuelLevel() { return fuelLevel; } public int getCurrentHeight() { return currentHeight; } public void fuelUp(int fuelAmount) { fuelLevel += fuelAmount; } public void takeOff() { if (fuelLevel >= 20) { fuelLevel -= 20; currentHeight += 20; } else throw new IllegalStateException("Not enough fuel!"); } public void goHigher() { fuelLevel -= 10; currentHeight += 50; } public void goLower() { currentHeight -= 50; } public void land() { if (currentHeight > 20) throw new LandingException("Too high to land!"); else currentHeight = 0; } public void testFlight() { try { takeOff(); goHigher(); goLower(); land(); } catch (LandingException ex) { System.out.print(ex.getMessage()); } } } class LandingException extends IllegalStateException { public LandingException() {} public LandingException(String message) { super(message); } } class InsufficientFuelException extends IllegalStateException { public InsufficientFuelException() {} public InsufficientFuelException(String message) { super(message); } }	Incorrect	0.00
4	23/03/19, 23:34	Submit: public class RocketShip { int MAX_FUEL_LEVEL = 100; int fuelLevel; int currentHeight; public RocketShip(int fuelLevel) { this.fuelLevel = fuelLevel; this.currentHeight = 0; } public int getFuelLevel() { return fuelLevel; } public int getCurrentHeight() { return currentHeight; } public void fuelUp(int fuelAmount) { fuelLevel += fuelAmount; } public void takeOff() { if (fuelLevel >= 20) { fuelLevel -= 20; currentHeight += 20; } else throw new IllegalStateException("Not enough fuel!"); } public void goHigher() { fuelLevel -= 10; currentHeight += 50; } public void goLower() { currentHeight -= 50; } public void land() { if (currentHeight > 20) throw new LandingException("Too high to land!"); else currentHeight = 0; } public void testFlight() { try { takeOff(); goHigher(); goLower(); land(); } catch (IllegalStateException ex) { System.out.print(ex.getMessage()); } } } class LandingException extends IllegalStateException { public LandingException() {} public LandingException(String message) { super(message); } } class InsufficientFuelException extends IllegalStateException { public InsufficientFuelException() {} public InsufficientFuelException(String message) { super(message); } }	Incorrect	0.00

Step	Time	Action	State	Marks
5	23/03/19, 23:35	Submit: public class RocketShip { int MAX_FUEL_LEVEL = 100; int fuelLevel; int currentHeight; public RocketShip(int fuelLevel) { this.fuelLevel = fuelLevel; this.currentHeight = 0; } public int getFuelLevel() { return fuelLevel; } public int getCurrentHeight() { return currentHeight; } public void fuelUp(int fuelAmount) { fuelLevel += fuelAmount; } public void takeOff() { if (fuelLevel >= 20) { fuelLevel -= 20; currentHeight += 20; } else throw new IllegalStateException("Not enough fuel!"); } public void goHigher() { fuelLevel -= 10; currentHeight += 50; } public void goLower() { currentHeight -= 50; } public void land() { if (currentHeight > 20) throw new LandingException("Too high to land!"); else currentHeight = 0; } public void testFlight() { try { takeOff(); goHigher(); goLower(); land(); } catch (IllegalStateException ex) { System.out.println(ex.getMessage()); } } } class LandingException extends IllegalStateException { public LandingException() {} public LandingException(String message) { super(message); } } class InsufficientFuelException extends IllegalStateException { public InsufficientFuelException() {} public InsufficientFuelException(String message) { super(message); } }	Correct	0.80
6	**23/03/19, 23:39**	**Attempt finished submitting:**	**Correct**	**0.80**

Information

Finally ...

Exceptions halt the code currently being executed, but what if we have unfinished business? For example, when working with files, it is best practice to close them after we have finished with them. But what if an exception occurs before we can do this?

This is where finally comes in. Finally can be added to a try/catch block, which looks like this:

```
try {
    // code that may cause an IllegalArgumentException
} catch (IllegalArgumentException e) {
    // do something with the exception
} finally {
    // clean up
}
```

The code inside the finally block will be executed regardless of whether an exception is thrown.

Response history

Step	Time	Action	State
1	22/03/19, 15:36	Started	
2	23/03/19, 23:27	Seen	
3	**23/03/19, 23:39**	**Attempt finished**	

Question **7**

Correct

Mark 1.00 out of 1.00

Test flight cleanup (finally)

Test flights can make quite a mess, regardless of whether everything goes according to plan.

Update your testFlight() method to include a finally block with the try/catch block, so that regardless of whether an exception is thrown, "Cleaning up launch pad" is printed.

Remember to include the code for any exceptions you use in your answer.

For example:

Test	Result

`RocketShip testRocketShip = new RocketShip(0);`
`try {`
`    testRocketShip.testFlight();`
`    System.out.println(testRocketShip.getCurrentHeight());`
`} catch (InsufficientFuelException e) {`
`    System.out.println("Oops! You didn't catch the exception");`
`}`
	`Not enough fuel!`
	`Cleaning up launch pad`
	`0`

Answer: (penalty regime: 0, 10, 20, ... %)

```java
public class RocketShip {

    int MAX_FUEL_LEVEL = 100;

    int fuelLevel;
    int currentHeight;

    public RocketShip(int fuelLevel) {
        this.fuelLevel = fuelLevel;
        this.currentHeight = 0;
    }

    public int getFuelLevel() {
        return fuelLevel;
    }

    public int getCurrentHeight() {
        return currentHeight;
    }

    public void fuelUp(int fuelAmount) {
        fuelLevel += fuelAmount;
```

	Test	Expected	Got	

✓	`RocketShip testRocketShip = new RocketShip(0);`
	`try {`
	`    testRocketShip.testFlight();`
	`System.out.println(testRocketShip.getCurrentHeight());`
	`} catch (InsufficientFuelException e) {`
	`    System.out.println("Oops! You didn't catch the exception");`
	`}`
		`Not enough fuel!`
		`Cleaning up launch pad`
		`0`
			`Not enough fuel!`
			`Cleaning up launch pad`
			`0`
				✓

✓	`RocketShip testRocketShip = new RocketShip(100);`
	`testRocketShip.testFlight();`
	`System.out.println("Completed test flight!");`
		`Cleaning up launch pad`
		`Completed test flight!`
			`Cleaning up launch pad`
			`Completed test flight!`
				✓

Passed all tests! ✓

Question author's solution:

```
public class InsufficientFuelException extends IllegalStateException {
    public InsufficientFuelException() {}

    public InsufficientFuelException(String message) {
        super(message);
    }
}

class RocketShip {

    int MAX_FUEL_LEVEL = 100;

    int fuelLevel;
    int currentHeight;

    RocketShip(int fuelLevel) {
        this.fuelLevel = fuelLevel;
        this.currentHeight = 0;
    }

    public int getFuelLevel() {
        return fuelLevel;
    }

    public void setFuelLevel(int fuelLevel) {
        this.fuelLevel = fuelLevel;
    }

    public int getCurrentHeight() {
        return currentHeight;
    }

    public void setCurrentHeight(int currentHeight) {
        this.currentHeight = currentHeight;
    }

    public void fuelUp(int fuelAmount) {
        this.fuelLevel += fuelAmount;
    }

    public void takeOff() {
        if (this.fuelLevel >= 20) {
            this.fuelLevel -= 20;
            this.currentHeight += 20;
        } else {
            throw new InsufficientFuelException("Not enough fuel!");
        }

    }

    public void goHigher() {
        this.fuelLevel -= 10;
        this.currentHeight += 50;
    }

    public void goLower() {
        this.currentHeight -= 50;
    }

    public void land() {
        this.currentHeight = 0;
    }

    public void testFlight() {
        try {
            this.takeOff();
            this.goHigher();
            this.goLower();
            this.land();
        } catch (InsufficientFuelException e) {
            System.out.println(e.getMessage());
        } finally {
            System.out.println("Cleaning up launch pad");
        }
    }
}
```

Correct

Marks for this submission: 1.00/1.00.

Response history

Step	Time	Action	State	Marks
1	22/03/19, 15:36	Started	Not complete	
2	23/03/19, 23:37	Submit: public class RocketShip { int MAX_FUEL_LEVEL = 100; int fuelLevel; int currentHeight; public RocketShip(int fuelLevel) { this.fuelLevel = fuelLevel; this.currentHeight = 0; } public int getFuelLevel() { return fuelLevel; } public int getCurrentHeight() { return currentHeight; } public void fuelUp(int fuelAmount) { fuelLevel += fuelAmount; } public void takeOff() { if (fuelLevel >= 20) { fuelLevel -= 20; currentHeight += 20; } else throw new IllegalStateException("Not enough fuel!"); } public void goHigher() { fuelLevel -= 10; currentHeight += 50; } public void goLower() { currentHeight -= 50; } public void land() { if (currentHeight > 20) throw new LandingException("Too high to land!"); else currentHeight = 0; } public void testFlight() { try { takeOff(); goHigher(); goLower(); land(); } catch (IllegalStateException ex) { System.out.println(ex.getMessage()); } finally { System.out.println("Cleaning up launch pad"); } } } class LandingException extends IllegalStateException { public LandingException() {} public LandingException(String message) { super(message); } } class InsufficientFuelException extends IllegalStateException { public InsufficientFuelException() {} public InsufficientFuelException(String message) { super(message); } }	Correct	1.00
3	**23/03/19, 23:39**	**Attempt finished submitting:**	**Correct**	**1.00**

◄ Quiz 4 - Inheritance, Interfaces, and Collection Jump to... Tutorial 1 - GUI Basics ►