OODP– Session 4c

Session times

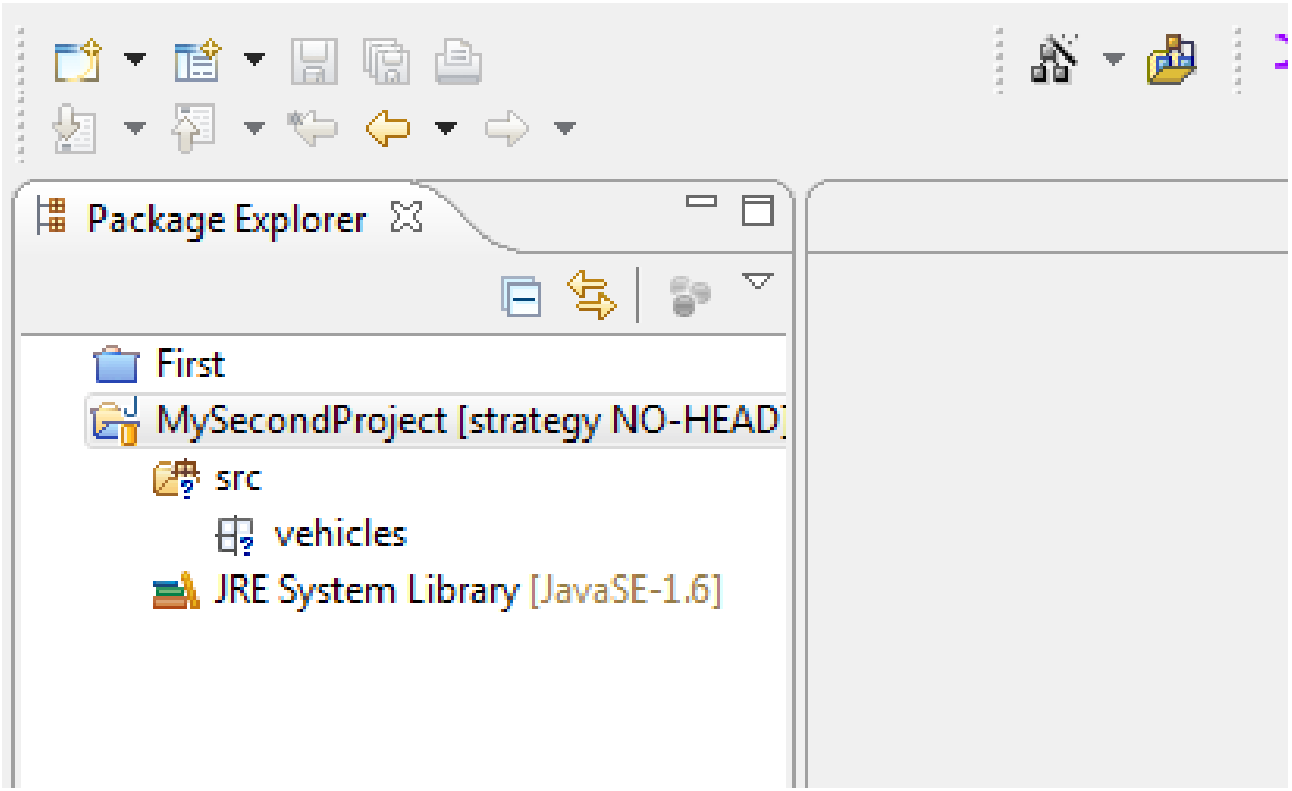
PT group 1 – Monday	18:00-21:00	room: Malet 403
PT group 2 – Thursday	18:00-21:00	room: Malet 407
FT - Tuesday	13:30-17:00	room: Malet 404

Email: oded@dcs.bbk.ac.uk

Web Page: http://www.dcs.bbk.ac.uk/~oded

Visiting Hours: Tuesday 17:00 to 19:00

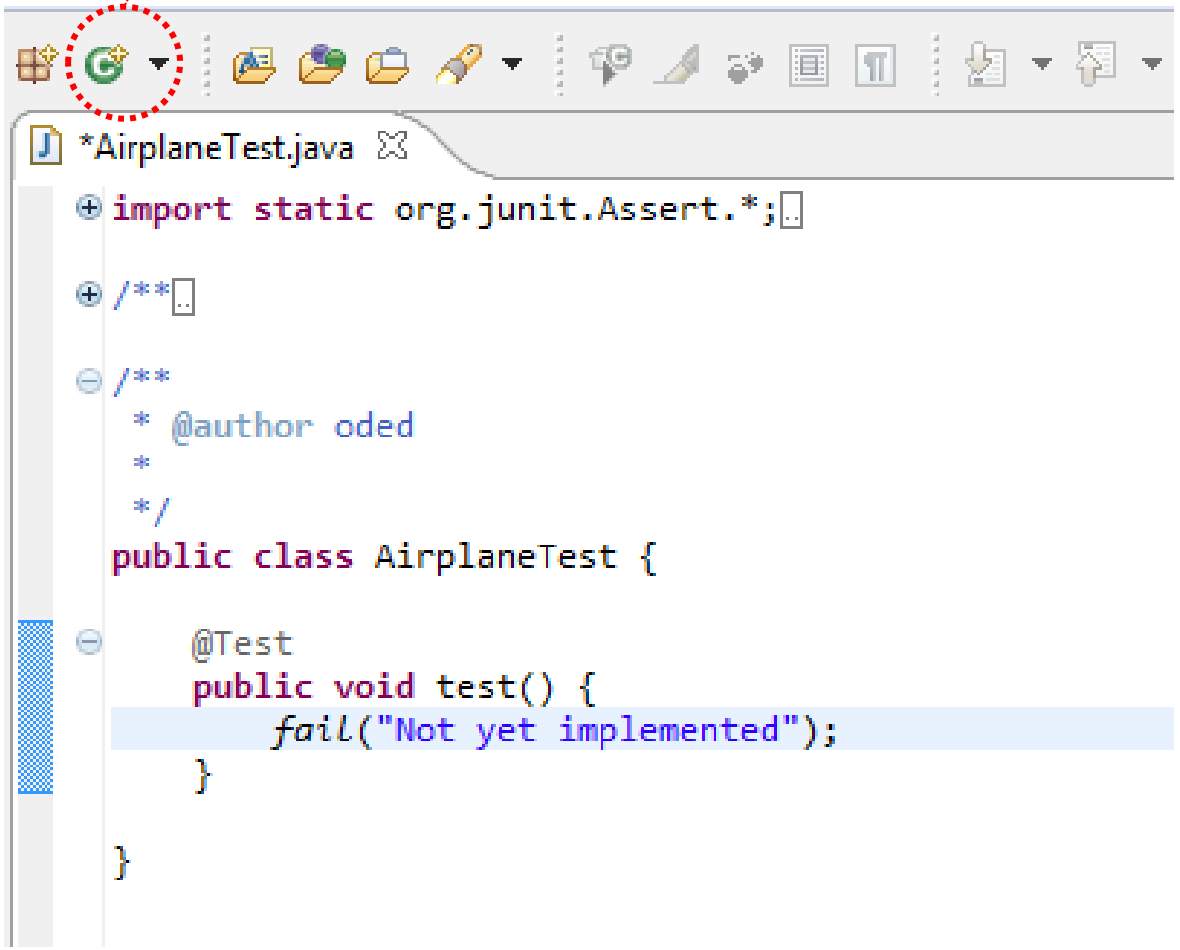
Start a new project



Make sure it is a **Java Project** since otherwise you won't get all the nice Java support Eclipse has to offer

New test

Use this to create test

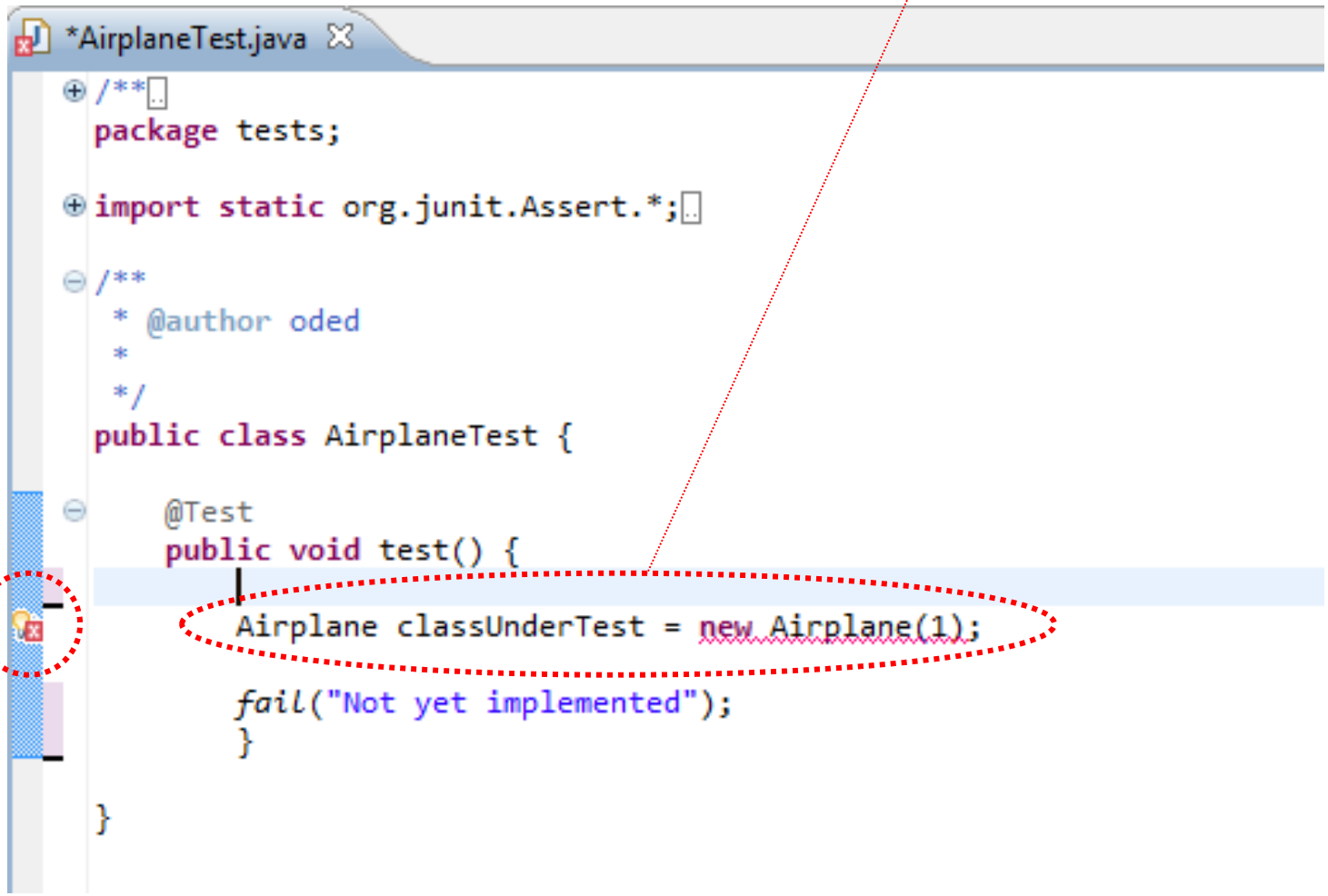


1. check that the test fails if it does, then commit

Preparing for a new class

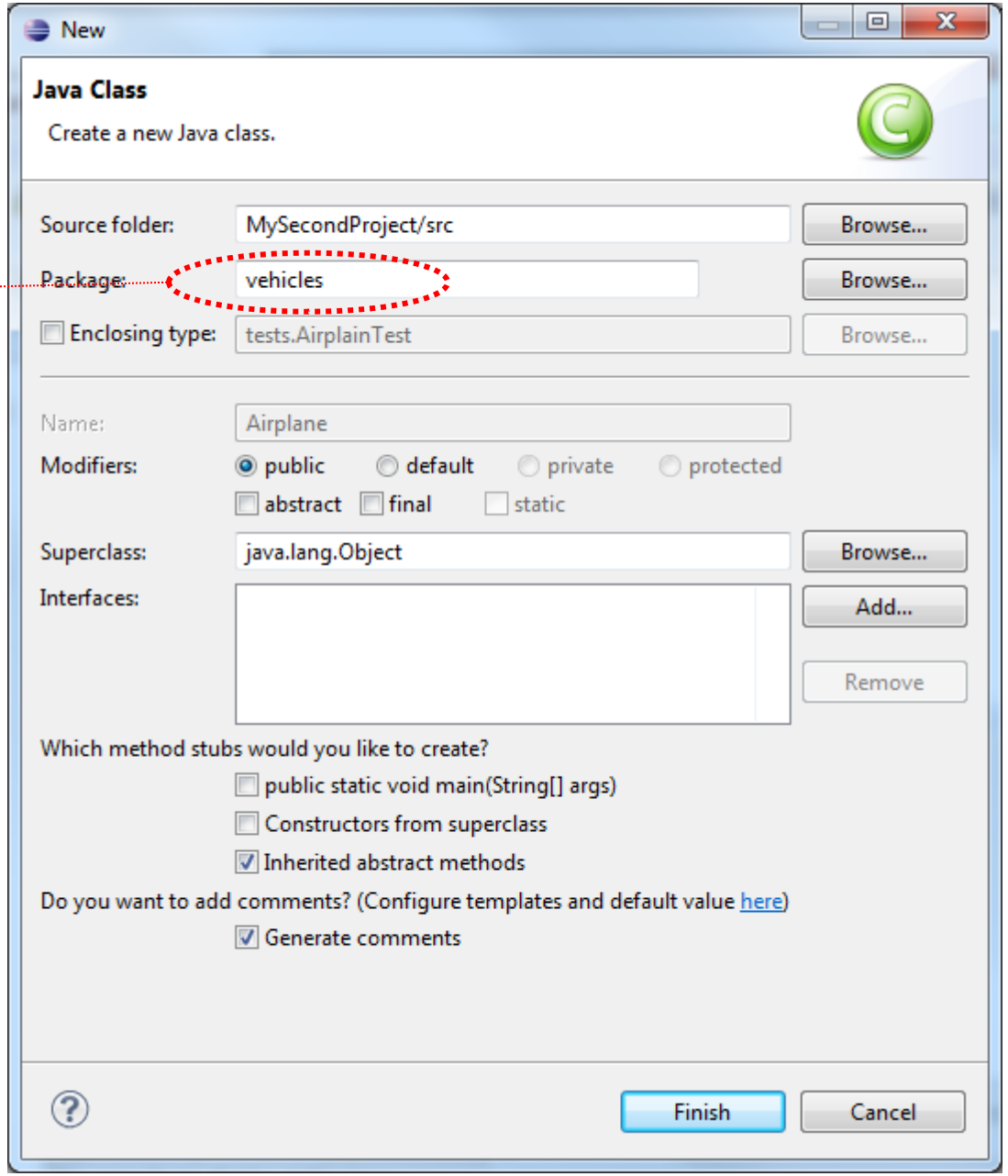
New Code

Use this to create *Airplane* class



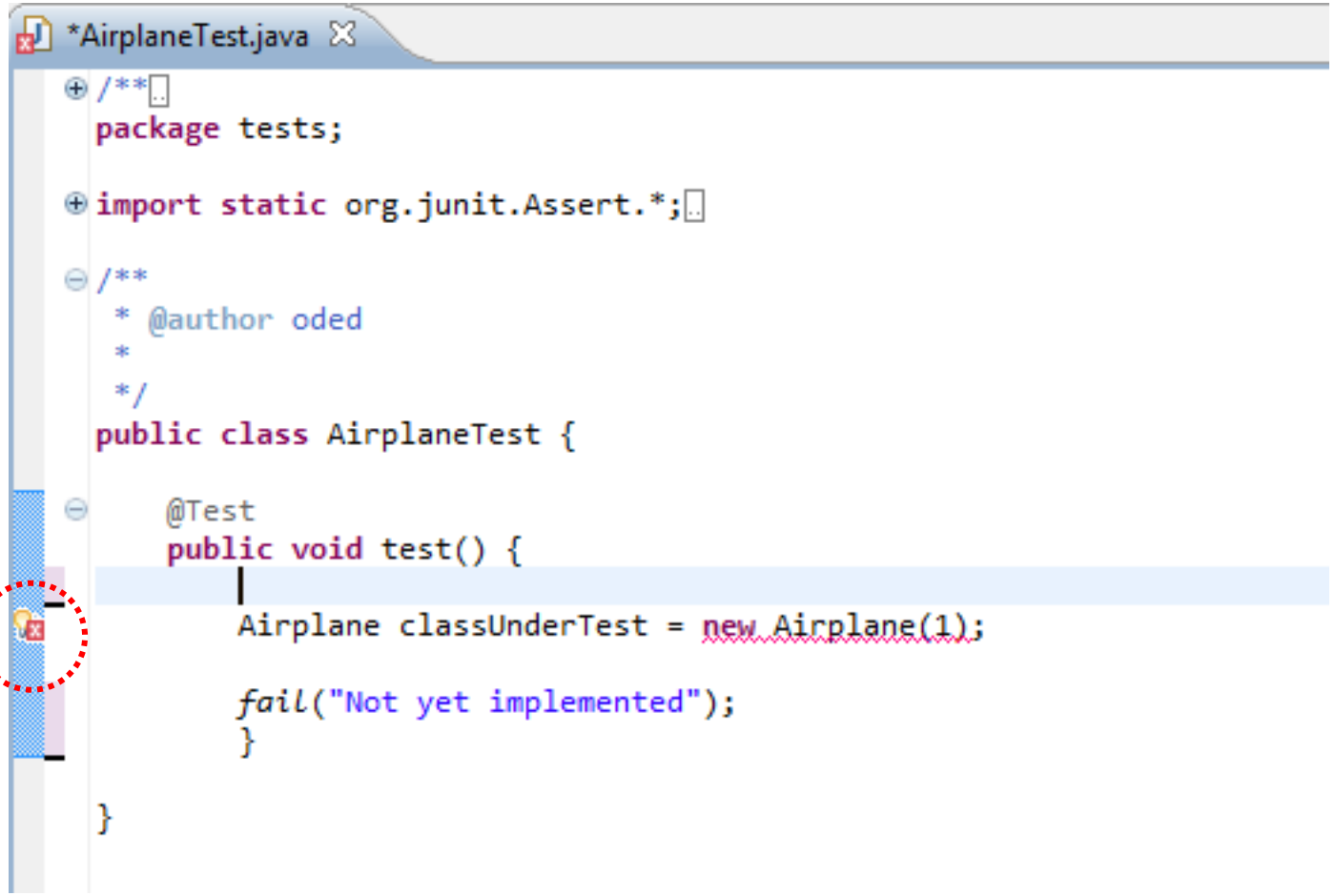
New class

Make in proper Package



Creating the constructor

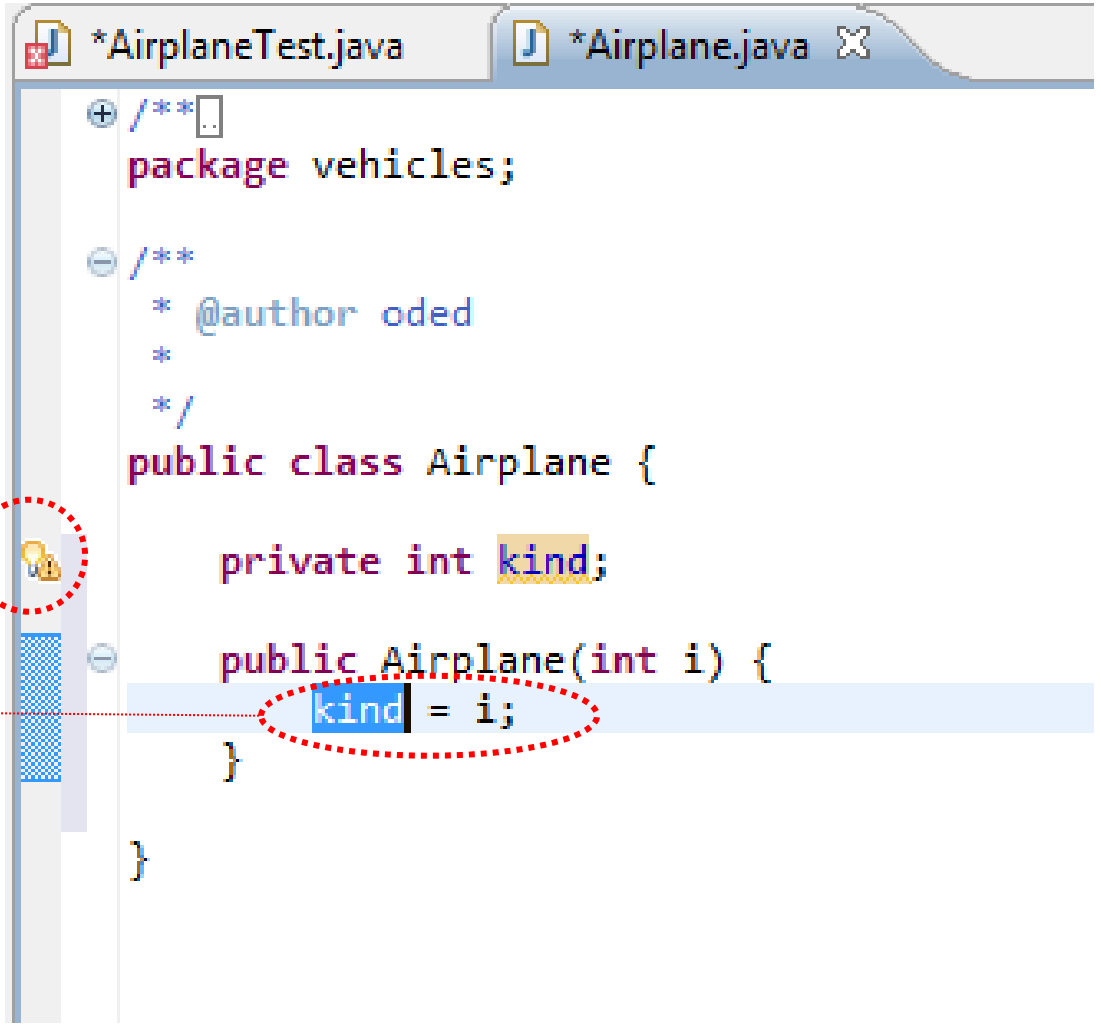
Use this to create the constructor for *Airplane* class



New Airplane code (with addition of a field)

What is this?

Start by adding this and then use IDE to create the field



Run test and if no problems commit

New test code

```
public void test() {

        String expectedOutput = "Like a fighter jet";
        String stringReturned = null;

        Airplane classUnderTest = new Airplane(1);

        stringReturned = classUnderTest.howDoYouFly();

        assertEquals("Wrong Answer !",  stringReturned, expectedOutput);
}
```

1. Use IDE to create new method in airplane class

2. Run test – if fails commit

Code may require adjusting if copy pasted

New how do you fly method (in airplane class)

```
public String howDoYouFly() {
        switch(kind){
        case 1: return "Like a fighter jet";
        case 2: return "I don't fly";
        case 3: return "Like a passenger plane";
        default: return null;
        }
}
```

1. Run test – if passes commit

New Liftoff test (in airplane class)

```
@Test
public void test2() {

        String expectedOutput = "Vertically";
        String stringReturned = null;

        Airplane classUnderTest = new Airplane(1);

        stringReturned = classUnderTest.howDoYouLiftOff();

        assertEquals("Wrong Answer !", stringReturned, expectedOutput);

}
```

1. Use IDE to create new method in airplane class

2. Run test – if fails commit

howDoYouLiftOff Code (for airplane class)

```
public String howDoYouLiftOff() {
      switch(kind){
      case 1: return "Vertically";
      case 2: return "I Liftoff";
      case 3: return "Horizontally";
      default: return null;
      }
}
```

1. Run test – if passes commit

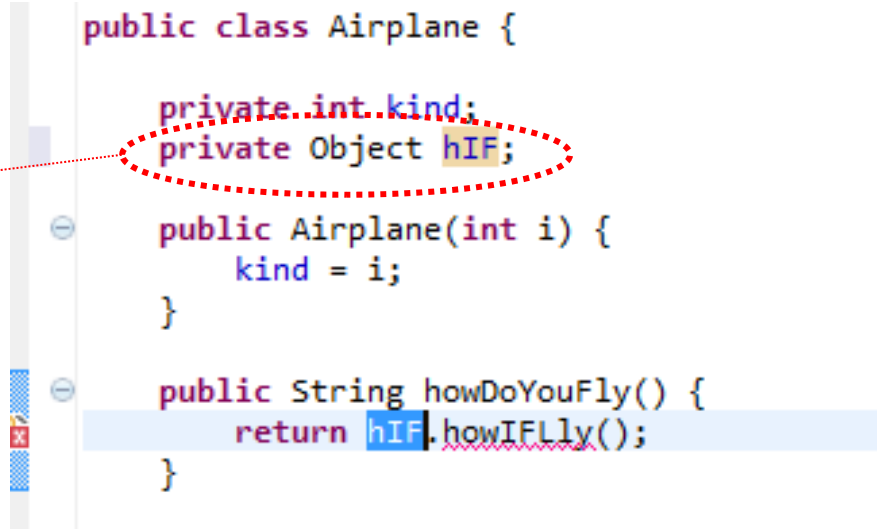
Now Apply the Strategy Pattern

Refactoring Flying

```
public String howDoYouFly() {
      return hIF.howIFLly();
}
```

1. New Code for **howDoYouFly()** in airplane class use IDE to create hIF field

Type object was created, because the type was not explicit (in Java all classes inherit Object by default)



Refactoring Flying (delegating flying to a class)

Fly like a fighter jet

1.Change Object to '**FFJ**'

Before

```
public class Airplane {

    private int kind;
    private Object hIF;

    public Airplane(int i) {
        kind = i;
    }

    public String howDoYouFly() {
        return hIF.howIFLly();
    }
```

After

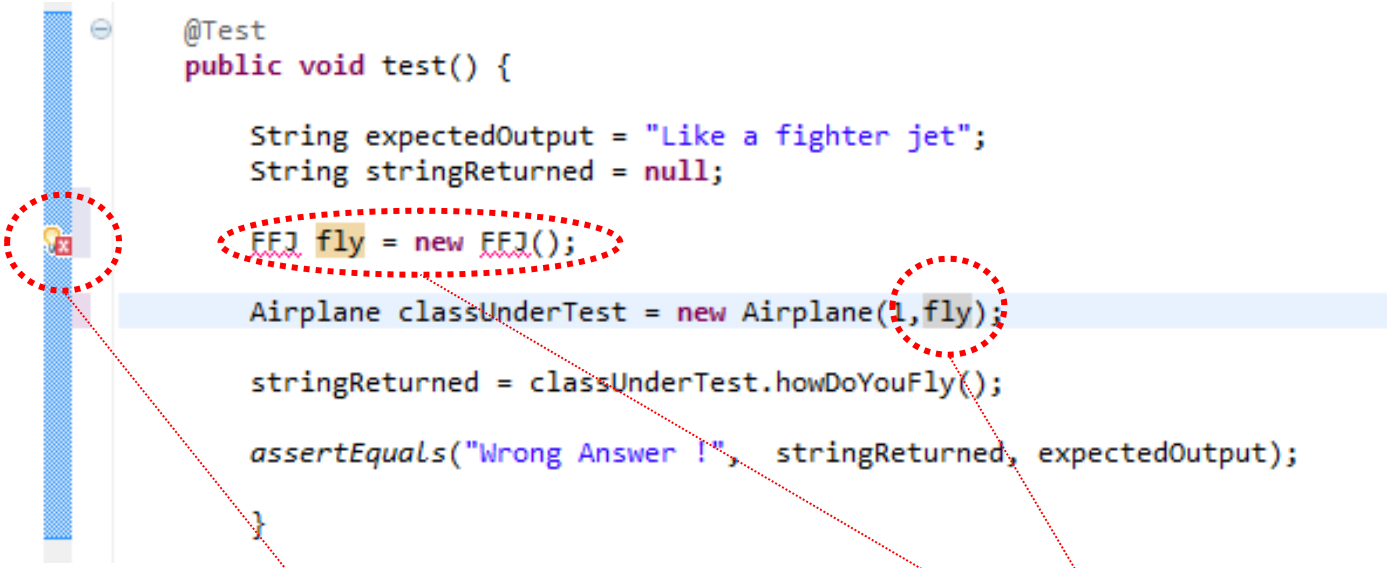
```
public class Airplane {

    private int kind;
    private FFJ hIF;

    public Airplane(int i) {
        kind = i;
    }

    public String howDoYouFly() {
        return hIF.howIFly();
    }
```

2. Use this to create 'FFJ' class

3. Use this to create 'howDoYouFly' method

If you run your test now. It should fail because no one sets

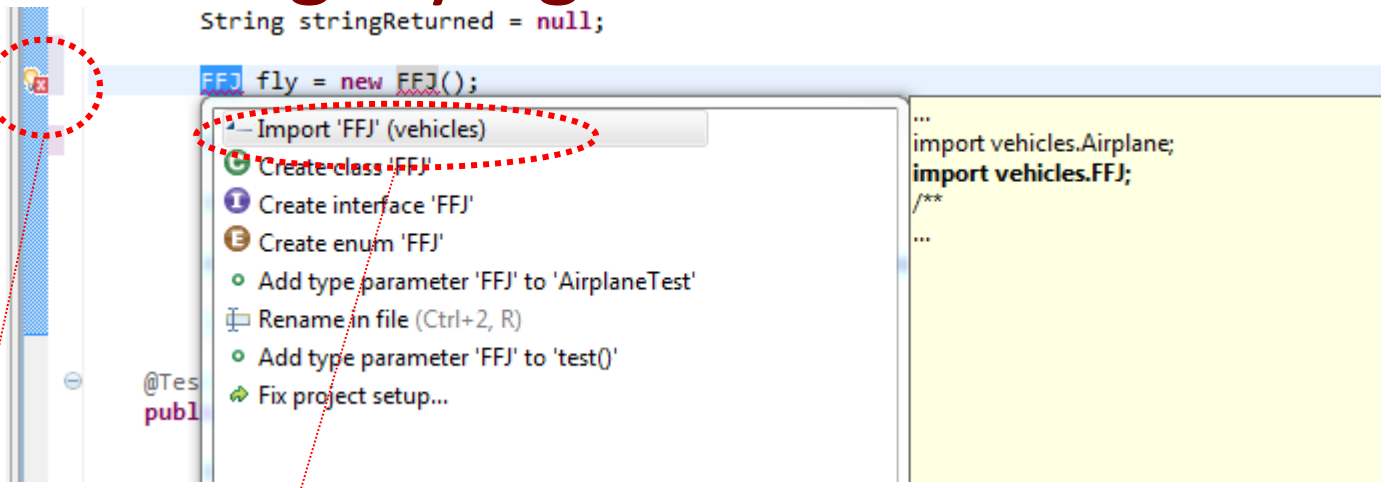
Refactoring Flying – setting the flying object by changing the test



This happened because the 'FFJ' is in a different package. We shall deal with this in the next slide

New code

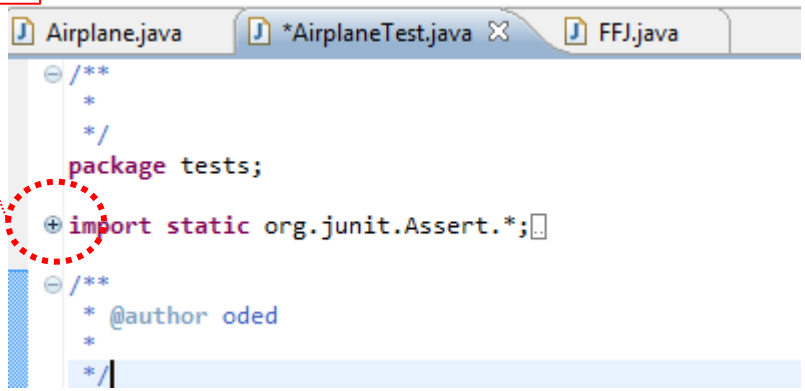
Refactoring Flying – importing



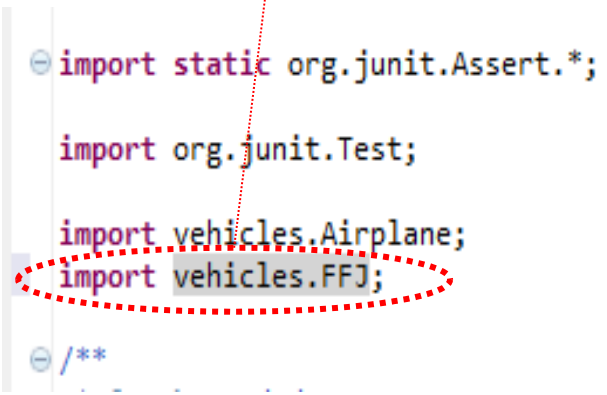
1. Select

2. Select

3. Select



This was added



Refactoring Flying –new constructor code

1. Use to Create new Constructor for 'AirPlane'

2. New code for constructor

```
public class AirplaneTest {

    @Test
    public void test() {

        String expectedOutput = "Like a fighter jet";
        String stringReturned = null;

        FFJ fly = new FFJ();

        Airplane classUnderTest = new Airplane(1,fly);

        stringReturned = classUnderTest.howDoYouFly();

        assertEquals("Wrong Answer !",  stringReturned, expectedOutput);

        }
}

public Airplane(int i, FFJ fly) {
    kind = i;
    hIF = fly;
}

public String howDoYouFly() {
```

Run your test now it should fail because 'Airplane' returned the wrong answer

Refactoring Flying – Fixing 'FFJ'

1. New code

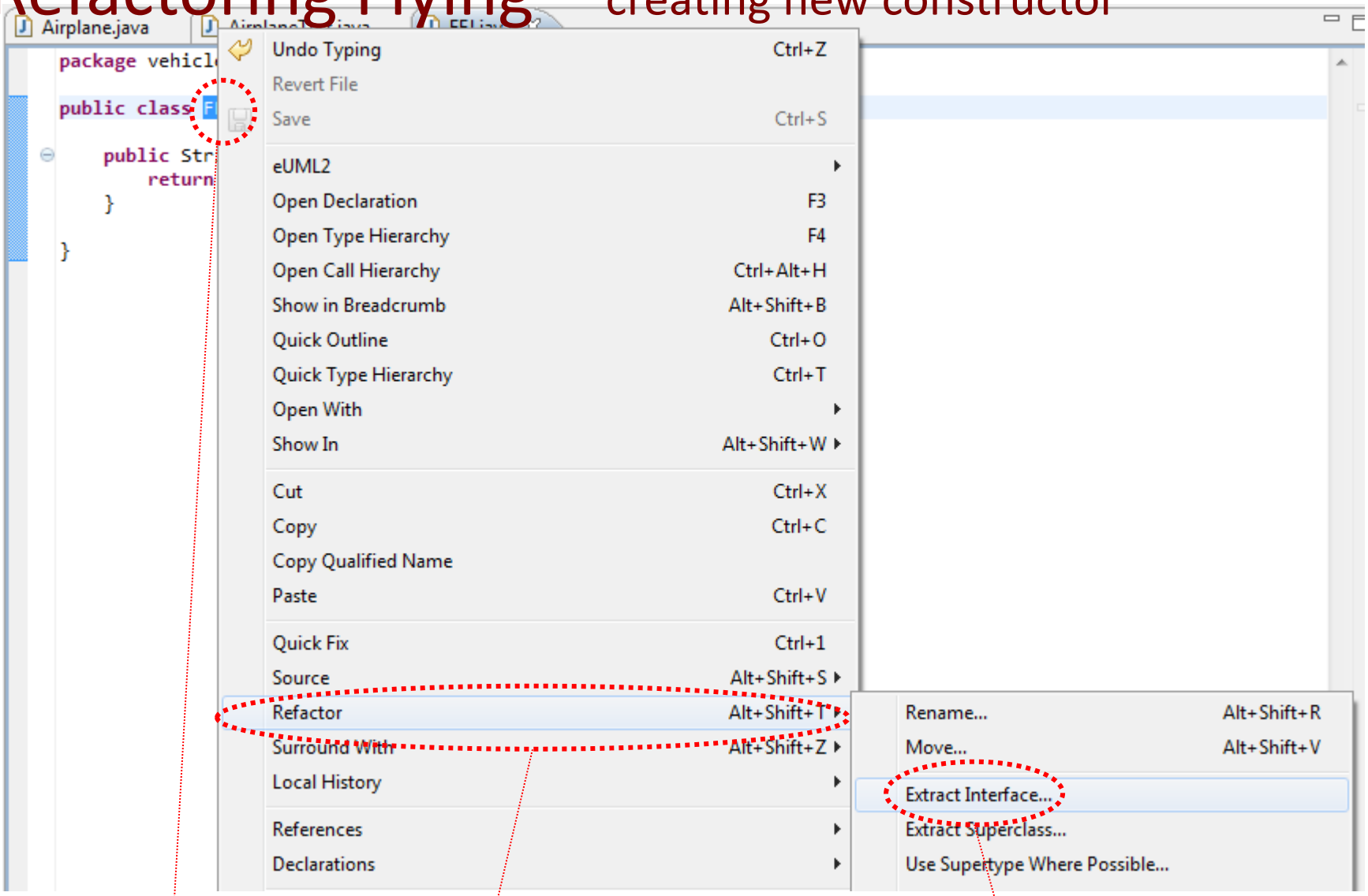
Airplane.java | AirplaneTest.java | FFJ.java

```
package vehicles;

public class FFJ {

    public String howIFly() {
        return "Like a fighter jet";
    }

}
```

Run your test now it should pass – commit (hopefully you have already done once or twice)

- We have only delegated one of the possibilities for Flying.
- Next we will add another way

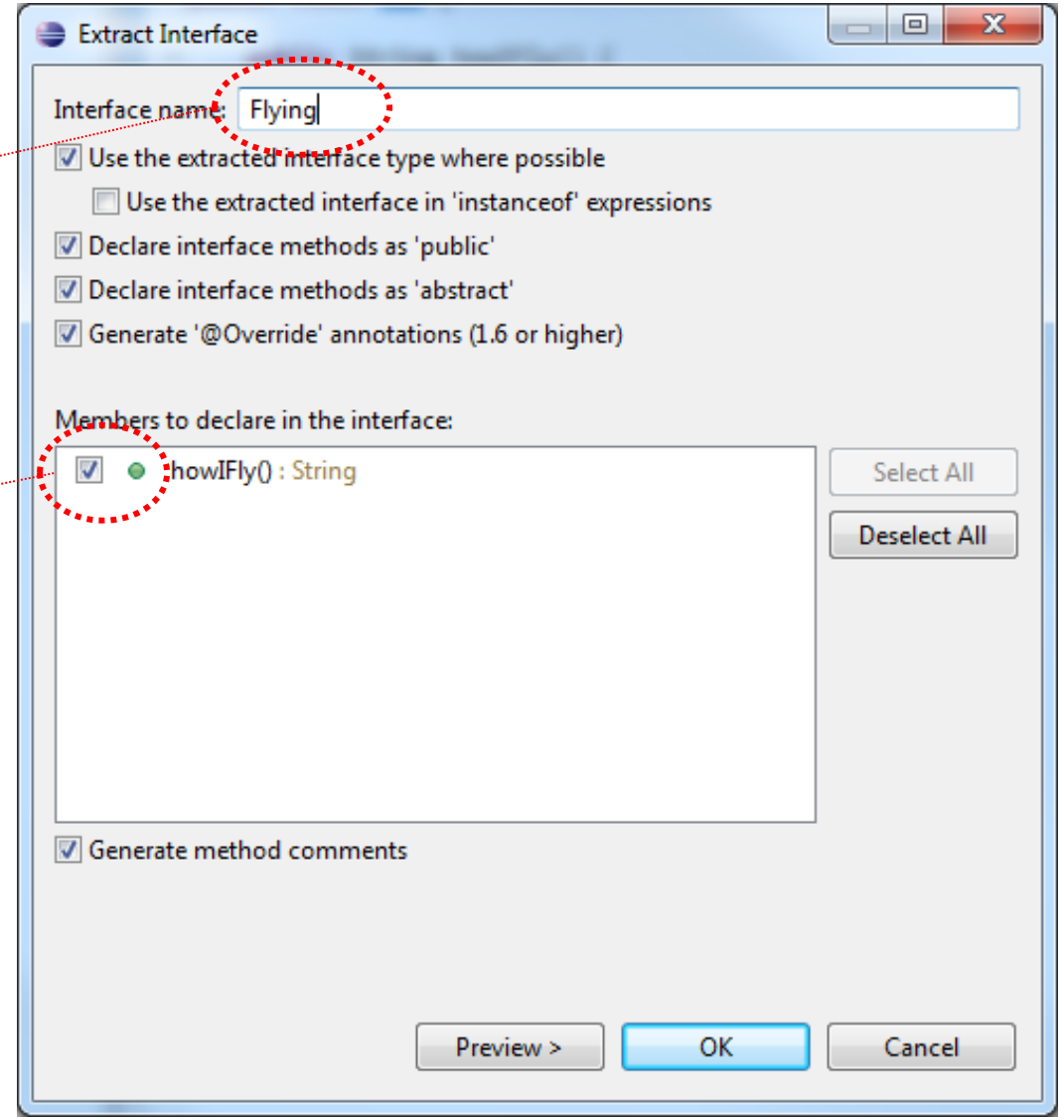
Refactoring Flying – creating new constructor



Refactoring Flying – creating new interface

1. Fill in

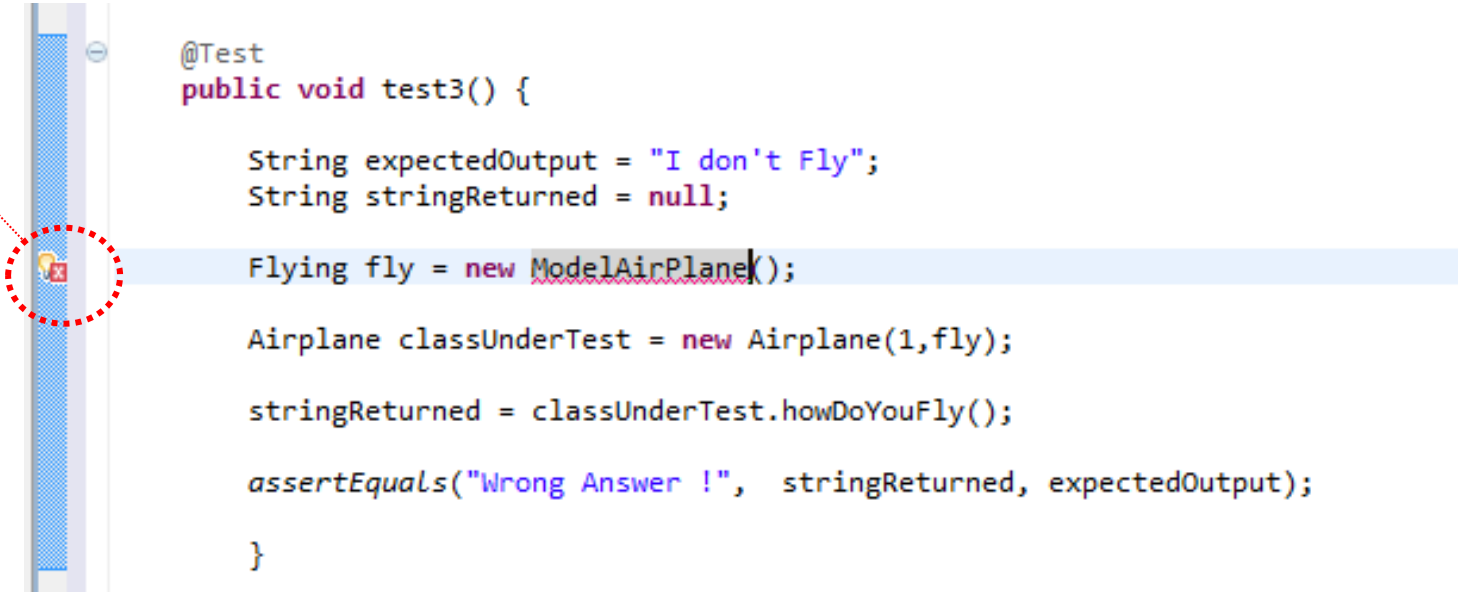
2. select



Run your test now it should pass – commit (note that you should have a new interface file)

Refactoring Flying – more flying using another test

1. Use to Create new class

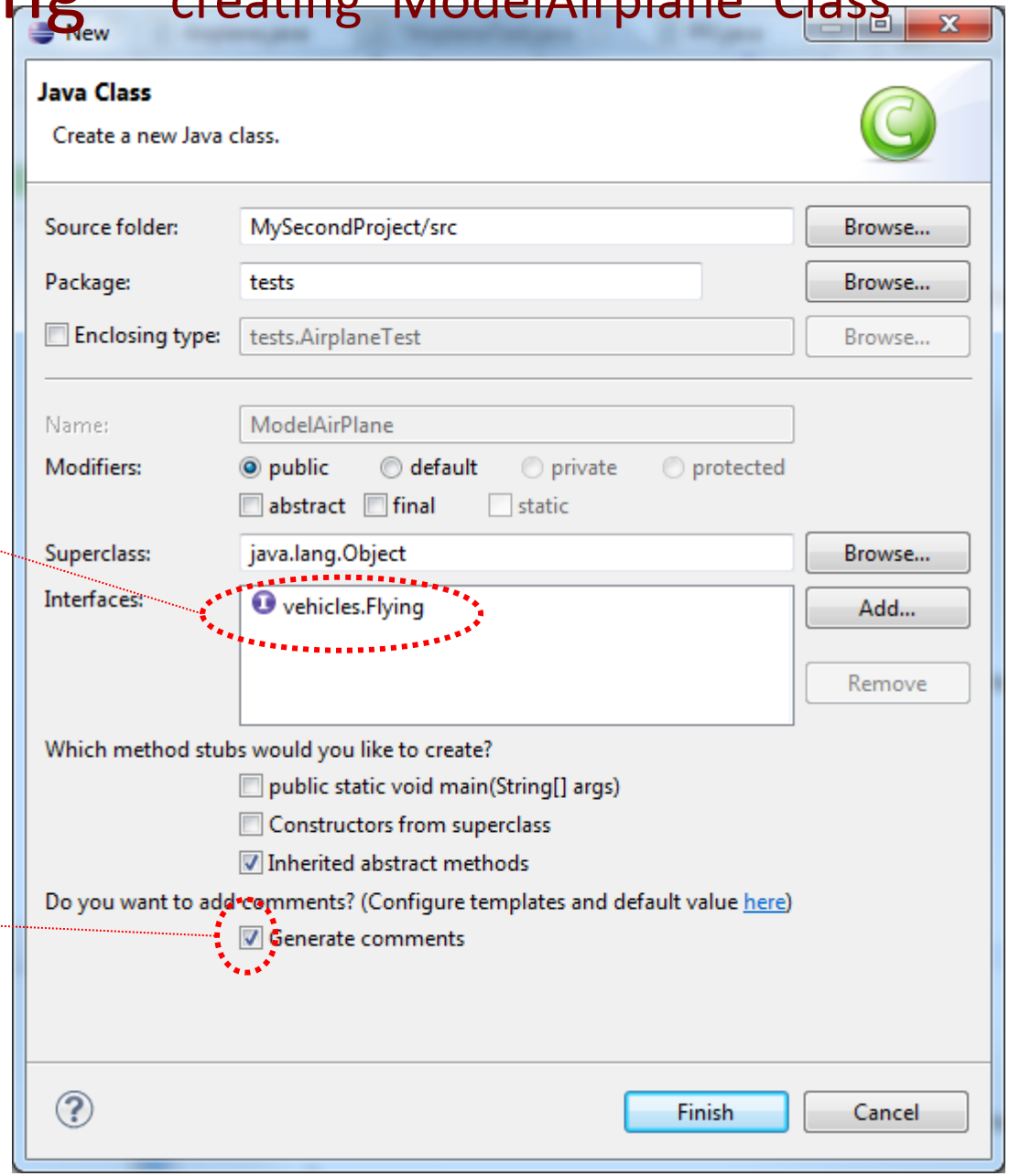


```
@Test
public void test3() {

    String expectedOutput = "I don't Fly";
    String stringReturned = null;

    Flying fly = new ModelAirPlane();

    Airplane classUnderTest = new Airplane(1,fly);

    stringReturned = classUnderTest.howDoYouFly();

    assertEquals("Wrong Answer !",  stringReturned, expectedOutput);

    }
```

Refactoring Flying – creating 'ModelAirplane' Class

This was done automatically (in general you can do this manually)

1. Select



Refactoring Flying – creating new constructor

1. Run your test now it should fail because ‘Airplane’ returned the wrong answer

2. Fix Model Airplane Class so that test passes

3. Commit

On your own

1. Add more types of flight
2. Apply Strategy to ‘takeoff’
3. Remember to get rid of constructors and variables that you don’t use anymore (some should remain)

Refactoring Flying – Snippet of automatically generated UML