OODP– Session 5b

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

Adapter

Adapter

1. Start Eclipse
2. Close any open project
3. Create new java project call it “Adapter”
4. Create three new packages “oldCode”,”newCode”, “tests”
5. Create a new Junit test “PersonTest” in package tests.

“PersonTest” Code in next slide

Select

Adapter

In new file “PersonTest”

```
@Test
public void test() {
        String expectedAnswer = “A person";
        String actualAnswer;

        Person classUnderTest = new Person ();

        actualAnswer = classUnderTest.whoAreYou();

        assertEquals("Wrong Answer!", expectedAnswer, actualAnswer);
}
```

1. Use IDE to create class ‘Person’ check that it is in the proper package
2. Use IDE to create method ‘`whoAreYou`’ in class ‘Person’
3. Run test. If it fails properly, then commit
4. Implement method ‘`whoAreYou`’ in class ‘Person’
5. Run test. If it passes like it should, then commit

Adapter

New Person
File you should get

PersonTest.java | Person.java

```
/**..
package oldCode;

/**
 * @author oded
 *
 */
public class Person {

    public String whoAreYou() {
        return "A person";
    }

}
```

Adapter

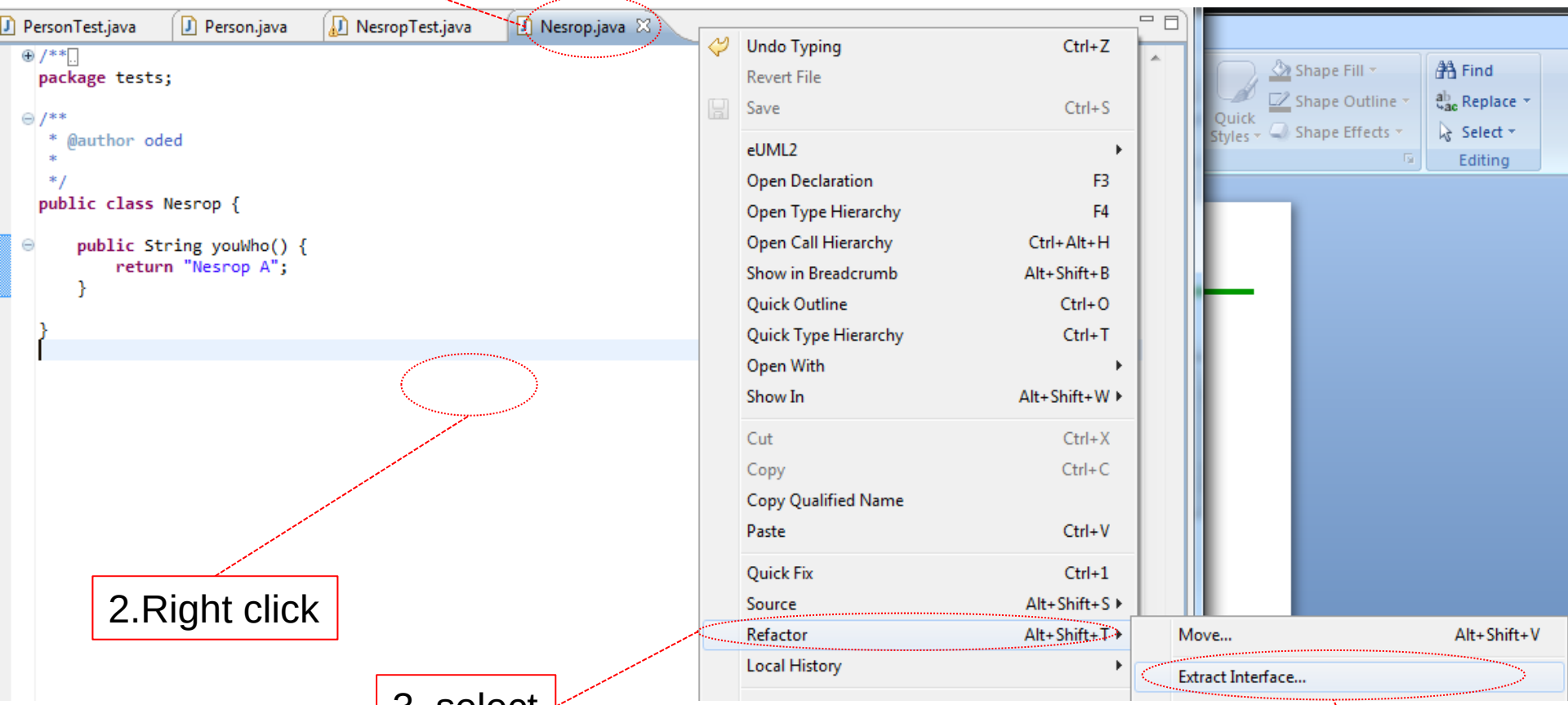
In new file
“NesropTest”

```
@Test
public void test() {
        String expectedAnswer = “Nesrop A";
        String actualAnswer;

        Nosrep classUnderTest = new nosreP ();

        actualAnswer = classUnderTest.youWho();

        assertEquals("Wrong Answer!", expectedAnswer, actualAnswer);
}
```

1. Repeat the steps used for Person

Adapter (using inheritance)

1.In this file



2.Right click

3. select

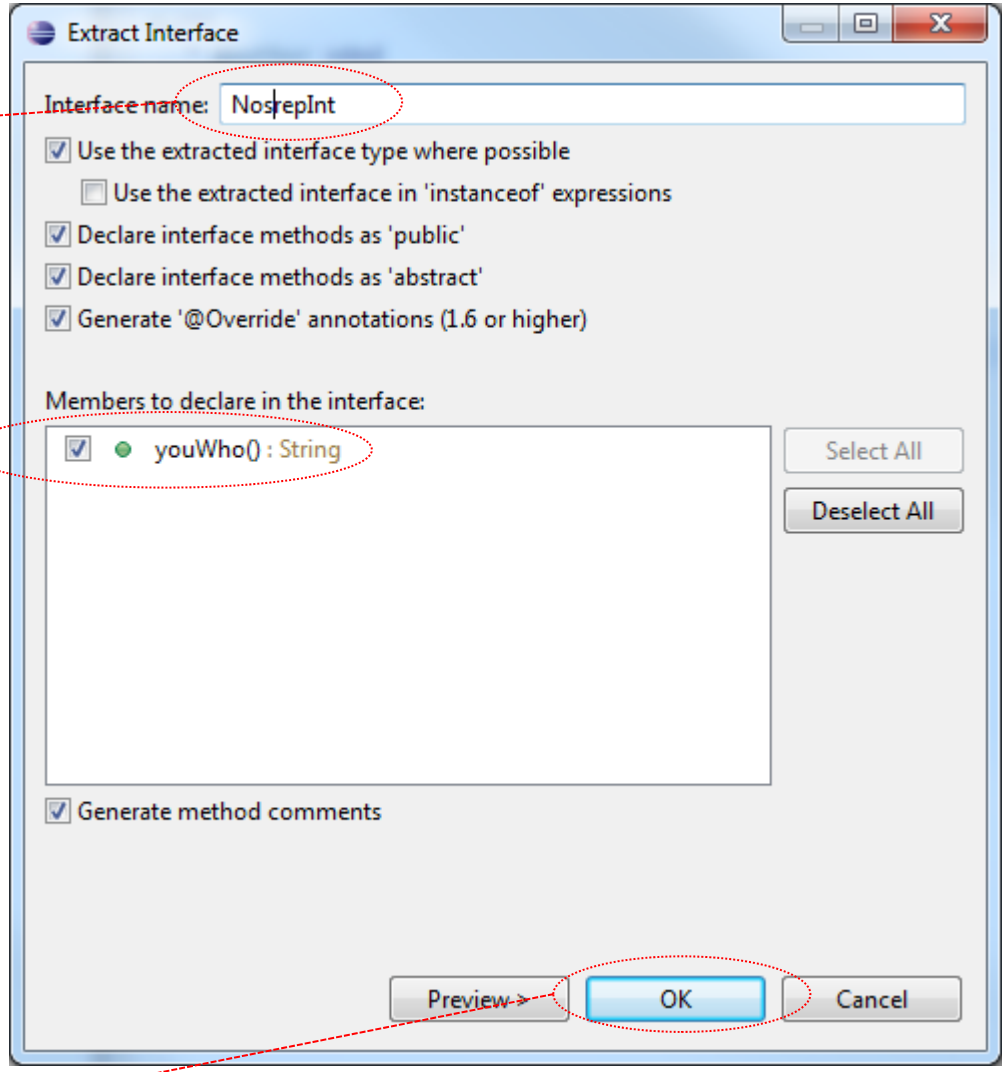
4. select

5. Run test. If it passes like it should, then commit

Adapter

1.Fill in

2. select



To see the file generated you need to select it from the Package Explorer

3. select

Adapter (using inheritance)

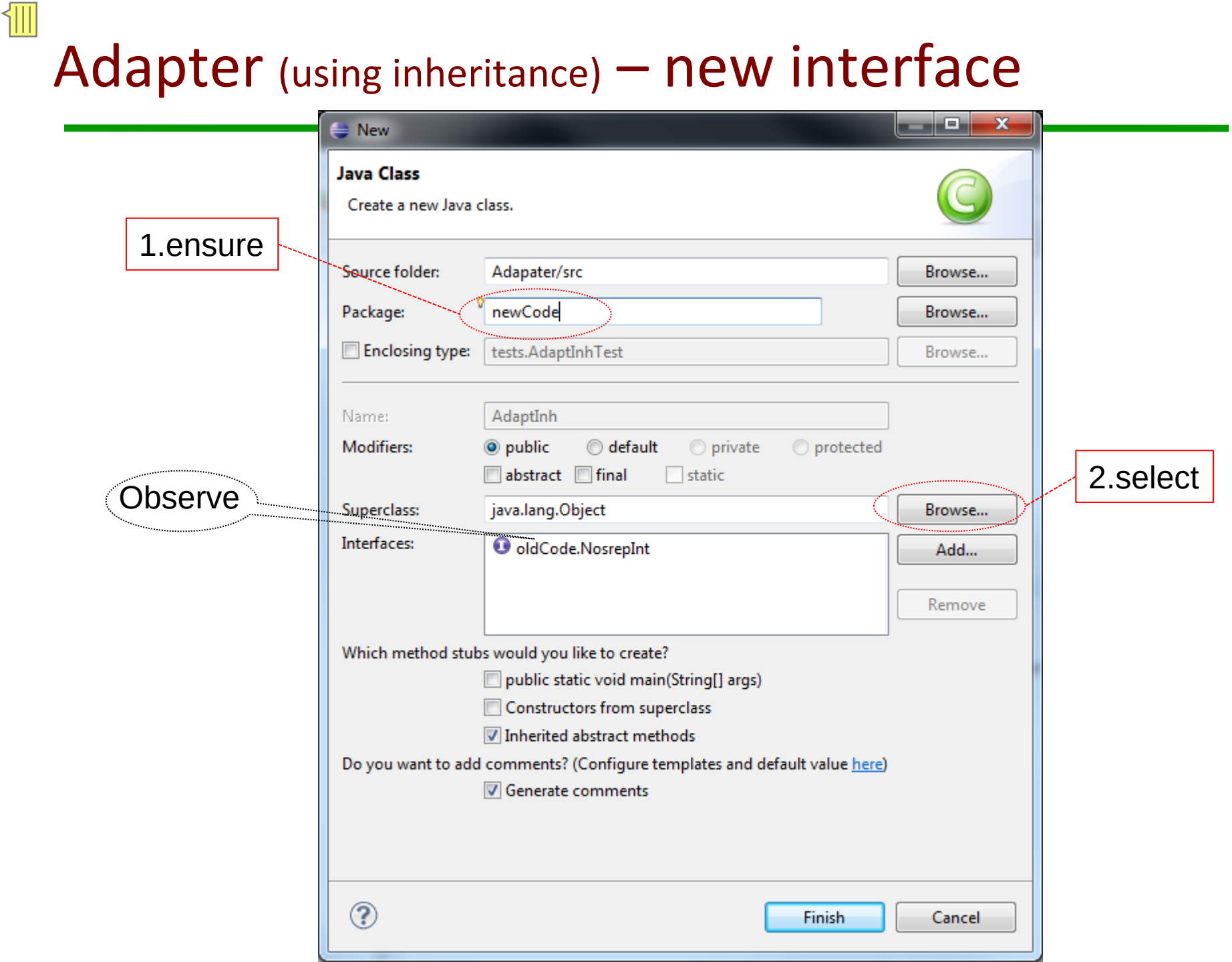
New test ‘AdaptInhTest’

1. We want an inheritance based adapter
2. It should have the ‘NesRop’ interface and the ‘Person’ implementation
3. We start generating it by using a new test ‘AdaptInhTest’

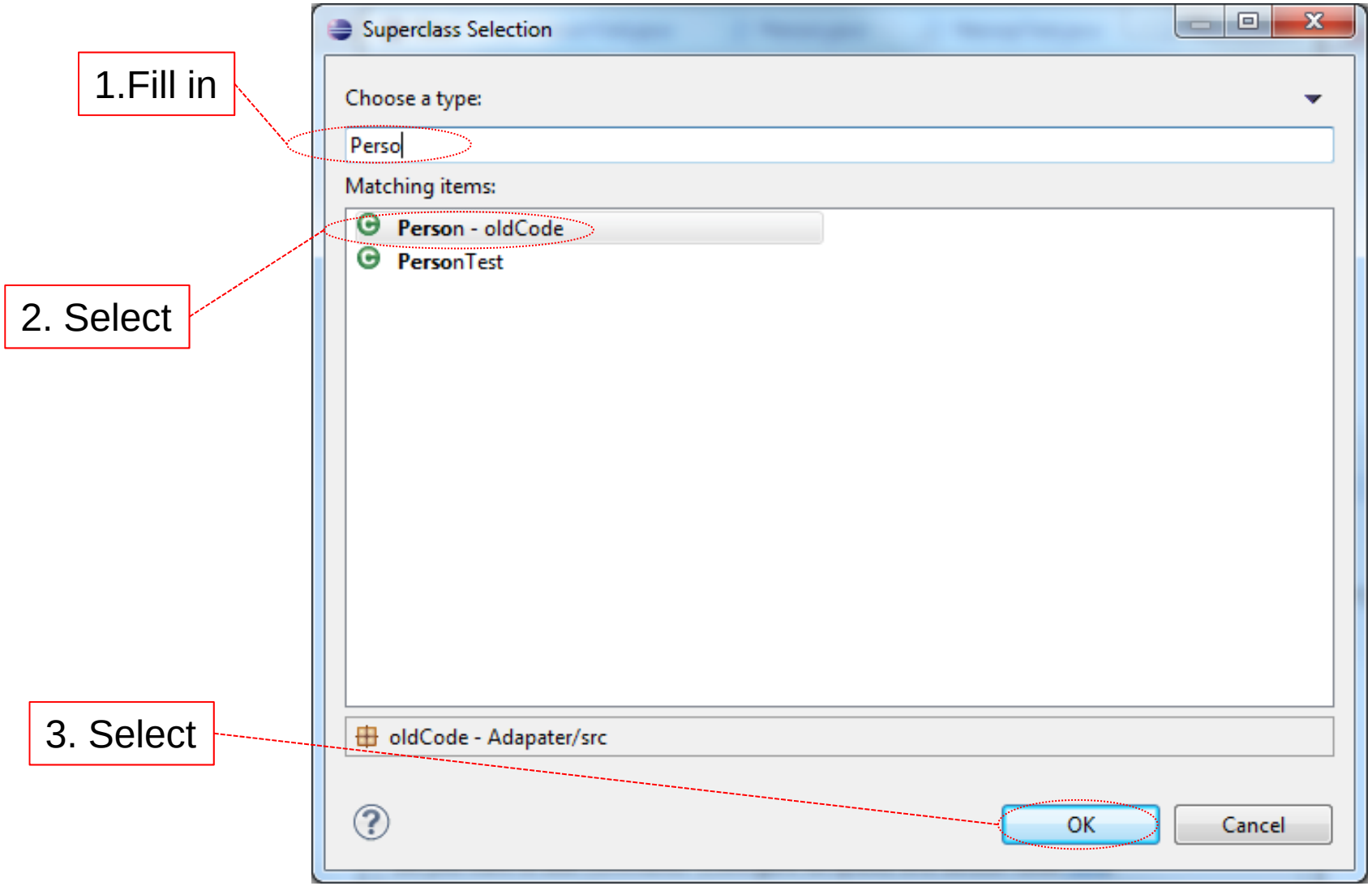
```
@Test
public void test() {
        String expectedAnswer = “A person";
        String actualAnswer;

        Nosrep classUnderTest = new AdaptInh();

        actualAnswer = classUnderTest.youWho();

        assertEquals("Wrong Answer!", expectedAnswer, actualAnswer);
}
```

1. Use test to get to the window in the next slide

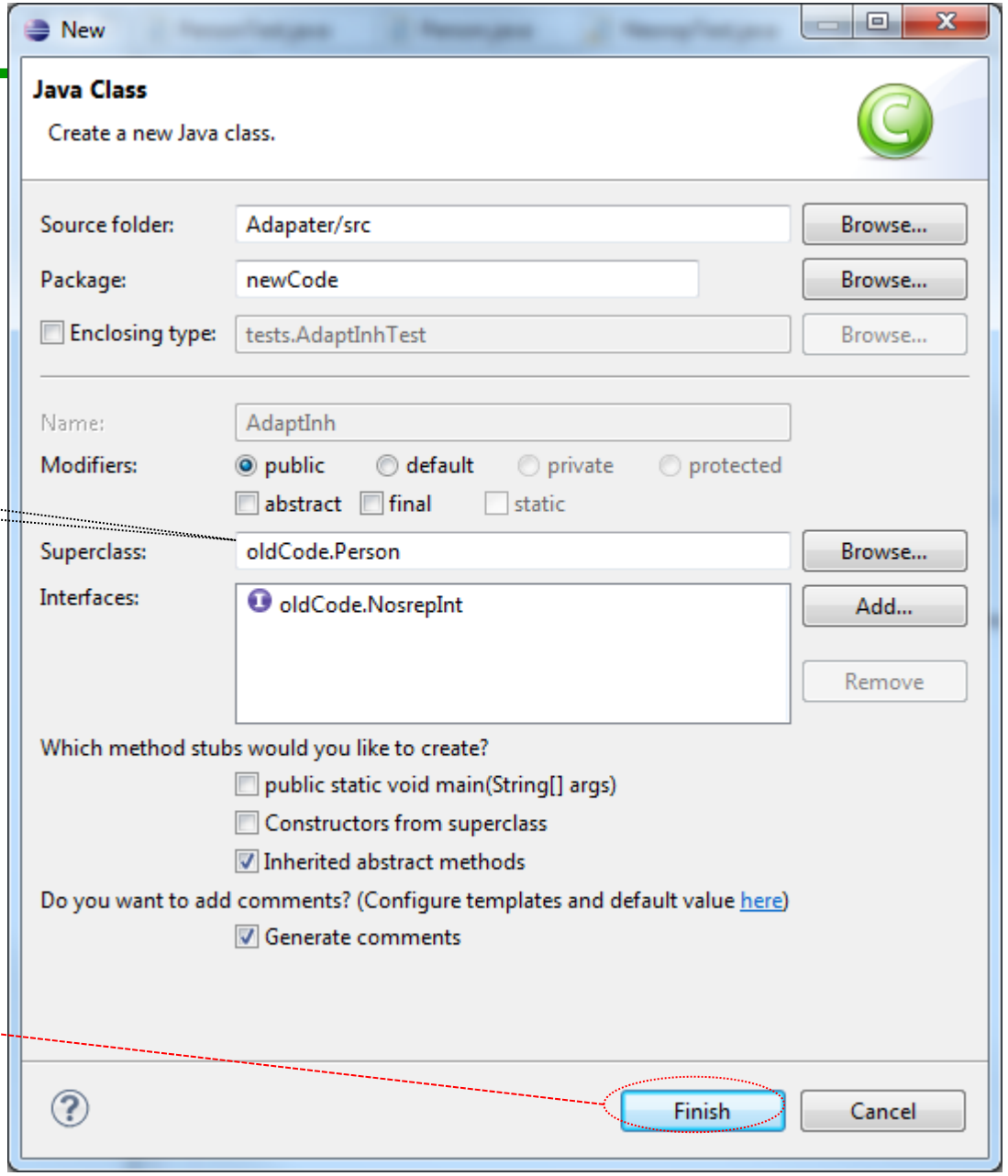
Adapter (using inheritance) – new interface



Adapter (using inheritance) – new interface



Adapter (using inheritance)



Observe

3. Select

Adapter (using inheritance) – new interface

```java
/**
package newCode;

import oldCode.NosrepInt;

/**
 * @author oded
 *
 */
public class AdaptInh extends Person implements NosrepInt {

    /* (non-Javadoc)
     * @see oldCode.NosrepInt#youWho()
     */
    @Override
    public String youWho() {
        // TODO Auto-generated method stub
        return null;
    }

}
```

Code You should get

1. Run test if it fails properly, then commit

2. Update code

```java
 */
public class AdaptInh extends Person implements NosrepInt {

    /* (non-Javadoc)
     * @see oldCode.NosrepInt#youWho()
     */
    @Override
    public String youWho() {
        return whoAreYou();
    }

}
```

3. Run test. If it passes, then commit

Adapter (using delegation)

New test ‘AdaptInhTest’

1. We want a delegation based adapter
2. It should have the ‘NesRop’ interface and the ‘Person’ implementation
3. We start generating it by using a new test ‘AdaptInhTest’

```
@Test
public void test() {
        String expectedAnswer = “A person";
        String actualAnswer;

        Nosrep classUnderTest = new AdaptDel();

        actualAnswer = classUnderTest.youWho();

        assertEquals("Wrong Answer!", expectedAnswer, actualAnswer);
}
```

1. Use test to get to generate ‘AdaptDel’ class (no need to inherit person)

2. Run test if it fails properly, then commit

Adapter (using delegation)

```java
/**
package newCode;

import oldCode.NosrepInt;

/**
 * @author oded
 *
 */
public class AdaptDel implements NosrepInt {

    /* (non-Javadoc)
     * @see oldCode.NosrepInt#youWho()
     */
    @Override
    public String youWho() {
        // TODO Auto-generated method stub
        return null;
    }

}
```

Code you should get

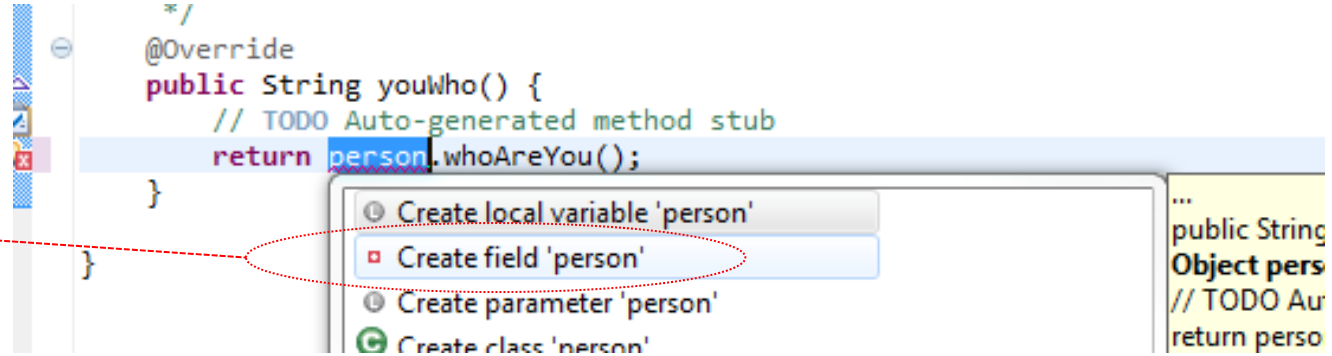
1. Update code

2. Select

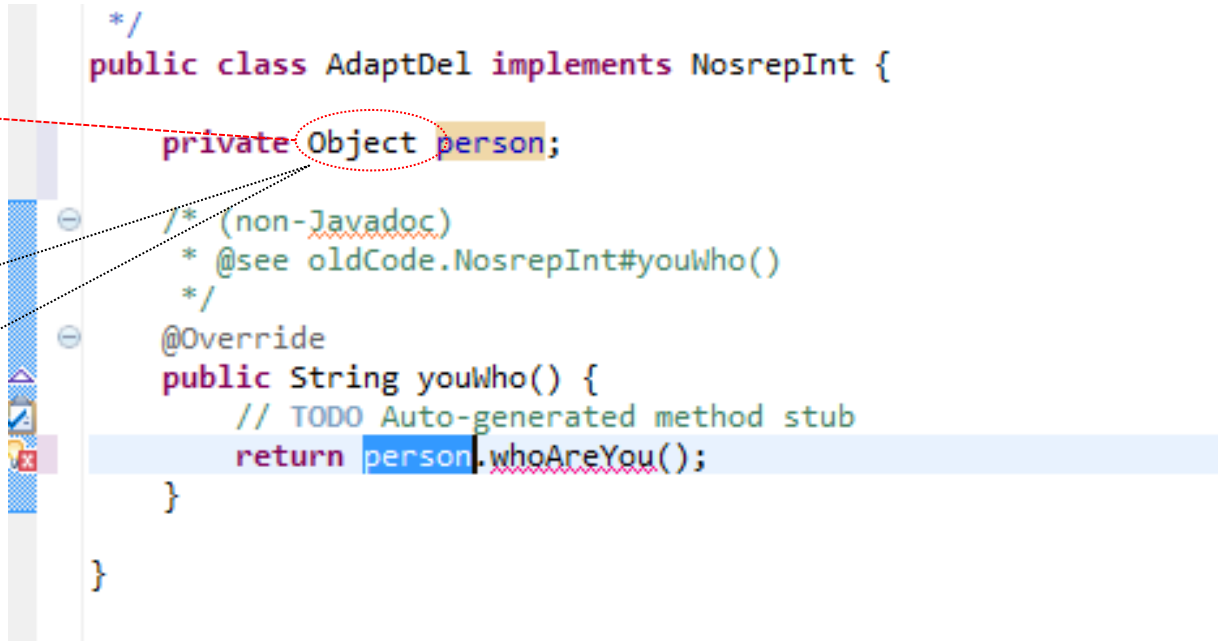
```java
 */
public class AdaptDel implements NosrepInt {

    /* (non-Javadoc)
     * @see oldCode.NosrepInt#youWho()
     */
    @Override
    public String youWho() {
        // TODO Auto-generated method stub
        return person.whoAreYou();
    }

}
```

Adapter (using delegation)

1. Select



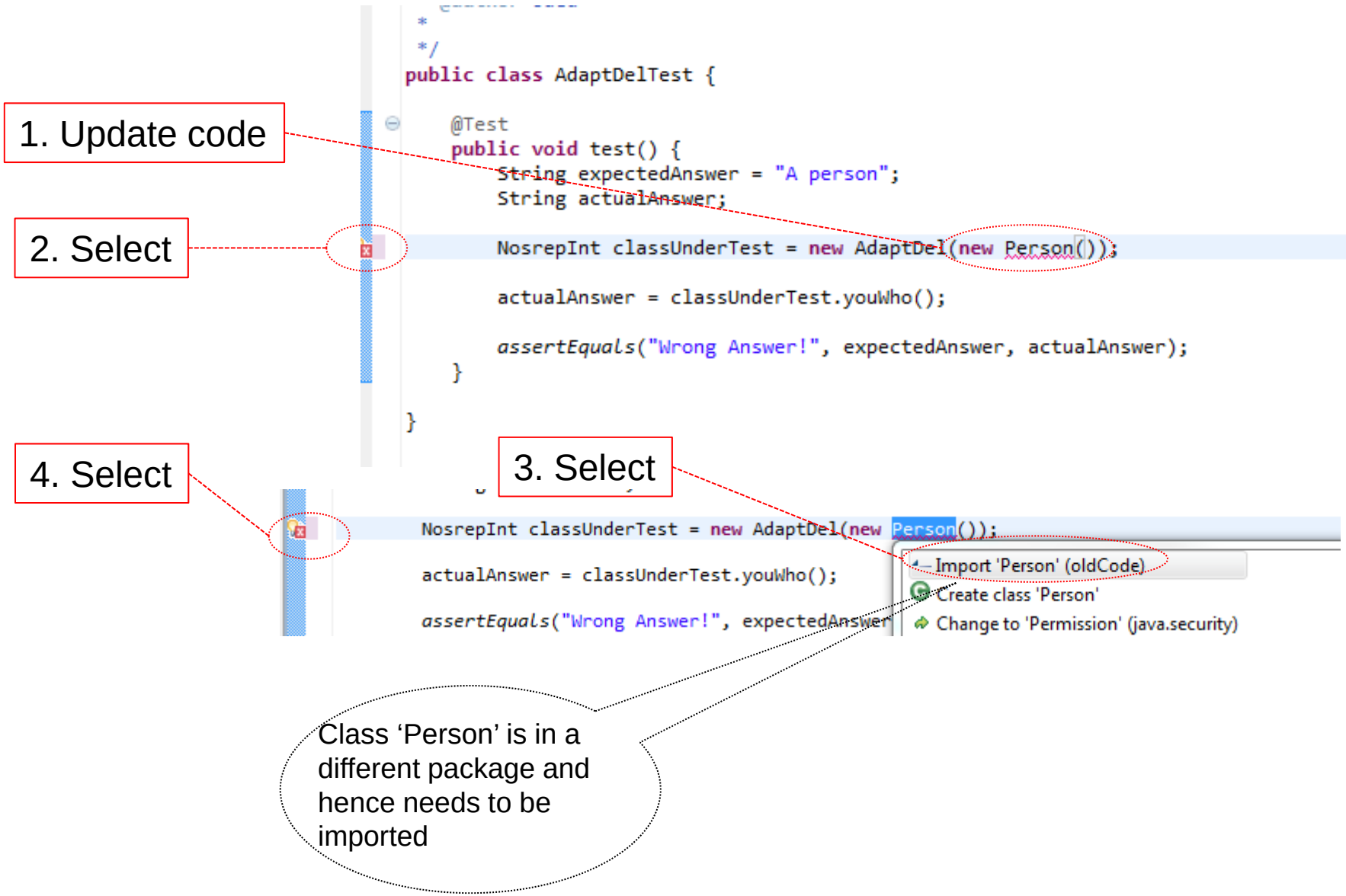
2. Change to 'Person'



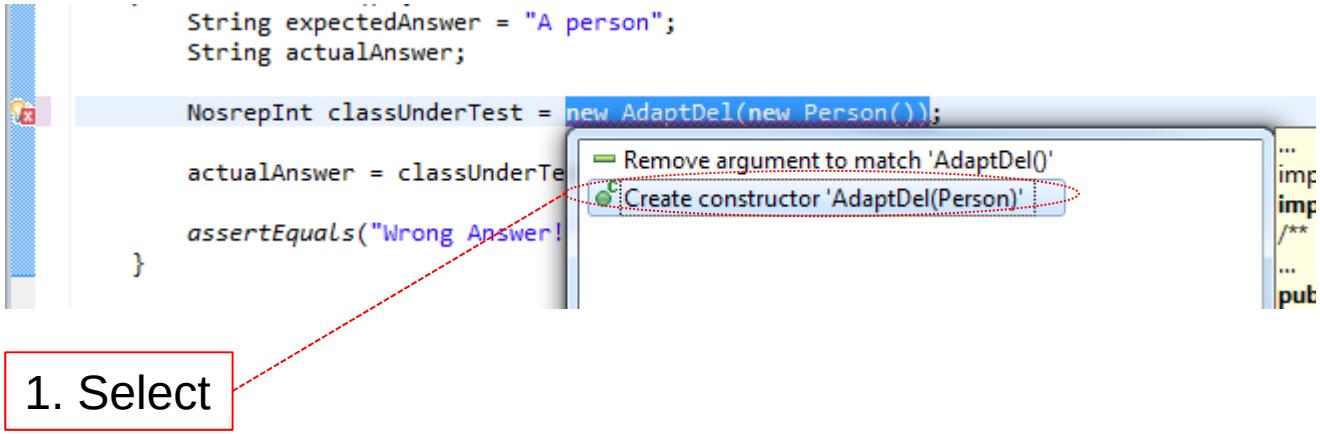
This happened because person field was not specified and hence it is assumed class is of type 'Object', since all classes are

The variable 'person' must be assigned a value, lets do that by changing 'AdapterDelTest'

Adapter (using delegation)



Adapter (using delegation)



Adapter (using delegation)

```java
public class AdaptDel implements NosrepInt {

    private Person person;

    public AdaptDel(Person person2) {
        // TODO Auto-generated constructor stub
    }

    /* (non-Javadoc)
     * @see oldCode.NosrepInt#youWho()
     */
    @Override
    public String youWho() {
        return person.whoAreYou();
    }

}
```

Code you should get

1. Update code

```java
 */
public class AdaptDel implements NosrepInt {

    private Person person;

    public AdaptDel(Person person2) {
        person = person2;
    }

    /* (non-Javadoc)
     * @see oldCode.NosrepInt#youWho()
     */
    @Override
```

2. Run test. If it passes, then commit

Decorator

Decorator

1. Start Eclipse
2. Close any open project
3. Create new java project call it “Decorator”
4. Create three new packages “oldCode”, “newCode”, “tests”
5. Create a new Junit test “RoomTest” in package tests.

“RoomTest” Code in next slide

Select

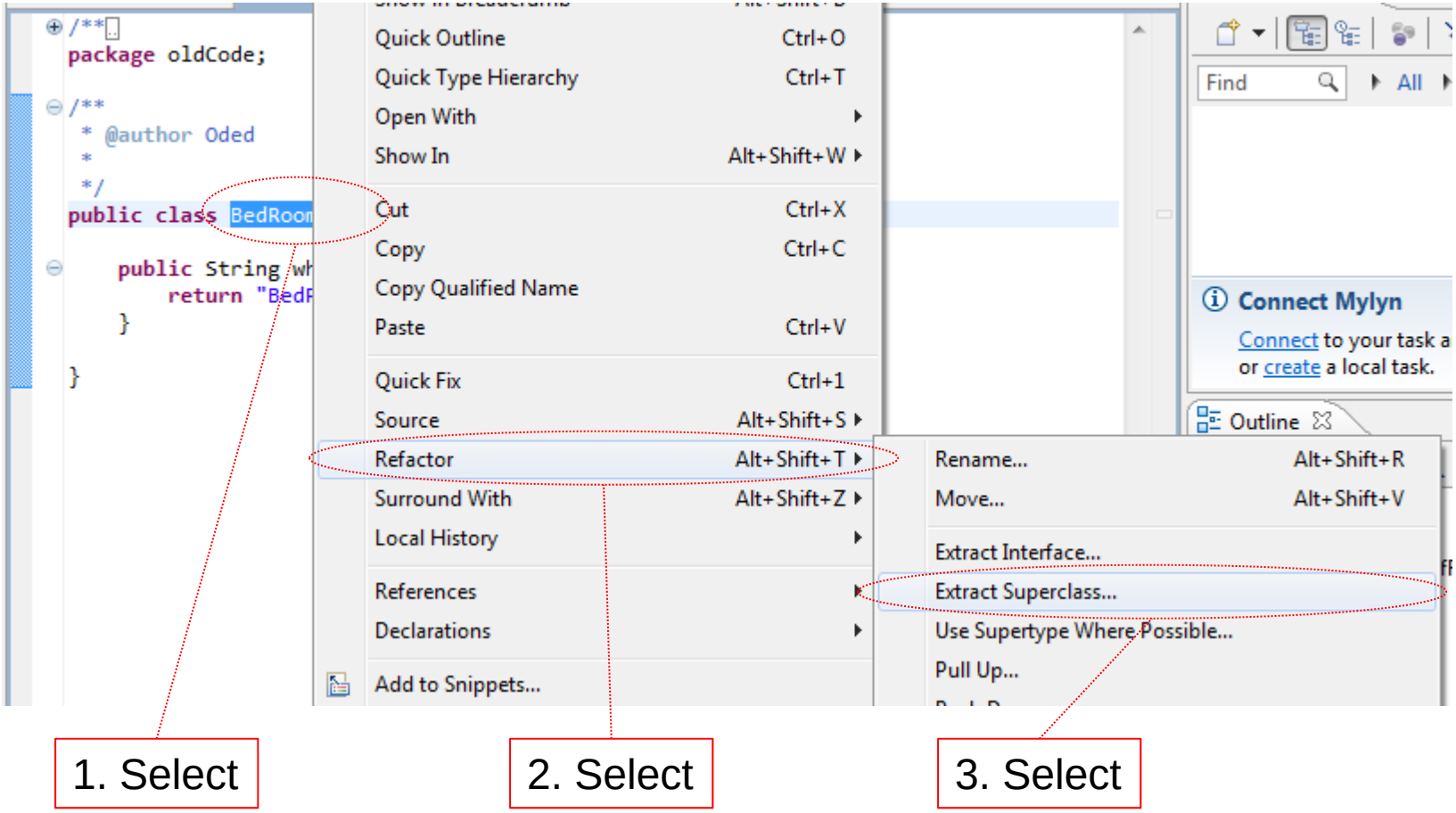
Decorator


```
public class RoomTest {
        @Test
        public void test() {
                String expectedResult = "BedRoom";
                String actualResult = null;

                BedRoom classUnderTest = new BedRoom();

                actualResult = classUnderTest.whatKindOfRoomAreYou();

                assertEquals("Wrong answer! ", expectedResult, actualResult);
        }
}
```

1. Use IDE to generate class “BedRoom” (make sure it is in package “oldCode”)
2. Use IDE to generate method “`whatKindOfRoomAreYou`” in class “BedRoom”
3. Run test, if fails in the proper way commit code.
4. Fix “`whatKindOfRoomAreYou`” so that test passes.
5. Commit

Decorator

1. Use Refactor to extract Super Class from class “BedRoom”

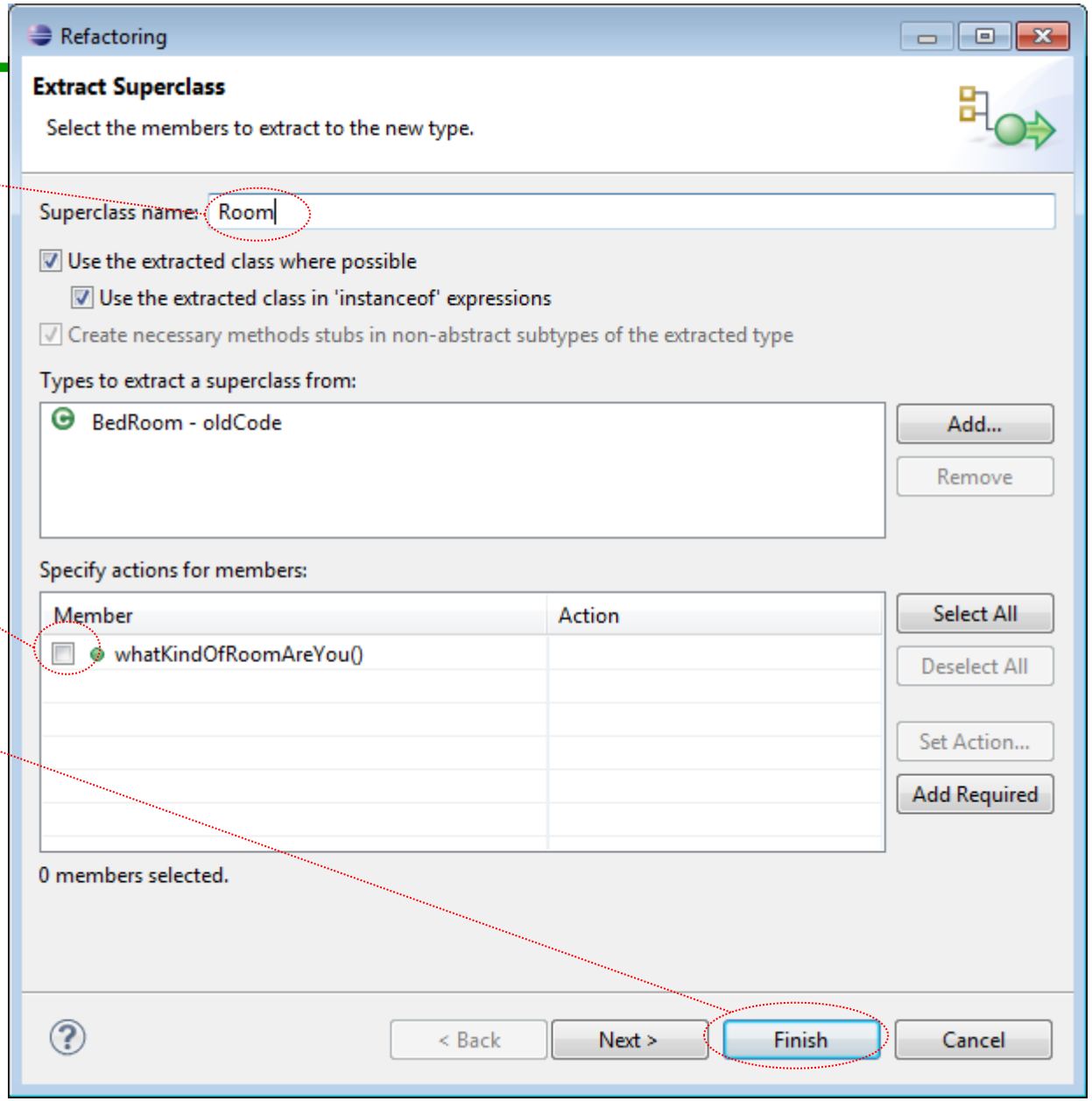


Decorator

1. Fill in

2. Don’t Select!

3. Select



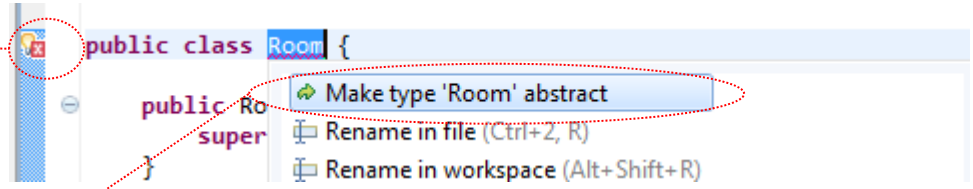
Decorator

This is a call to the constructor of class"Room"'s superclass

```
public class Room {

    public Room() {
    super();
    }

    public abstract String whatKindOfRoomAreYou();

}
```

1. Add this line of code

2. Select



3. Select

Decorator

This is a call to the constructor of class"Room"'s superclass

```
package oldCode;

public abstract class Room {

    public Room() {
        super();
    }

    public abstract String whatKindOfRoomAreYou();

}
```

1. Change

```
public class RoomTest {

    @Test
    public void test() {
        String expectedResult = "BedRoom";
        String actualResult = null;

        Room classUnderTest = new BedRoom();

        actualResult = classUnderTest.whatKindOfRoomAreYou();

        assertEquals("Wrong answer! ", expectedResult, actualResult);
    }
```

2. Run test if passes like it should then commit

Decorator

New test ‘LivingRoomTest’

1. Create new test by copy past from the existing one
2. Use test to generate “LivingRoom” class (remember it should implement Interface “Room” and be in package “oldCode”)
3. Run tests. If it fails properly, then commit. Fix problem and when test passes commit again.

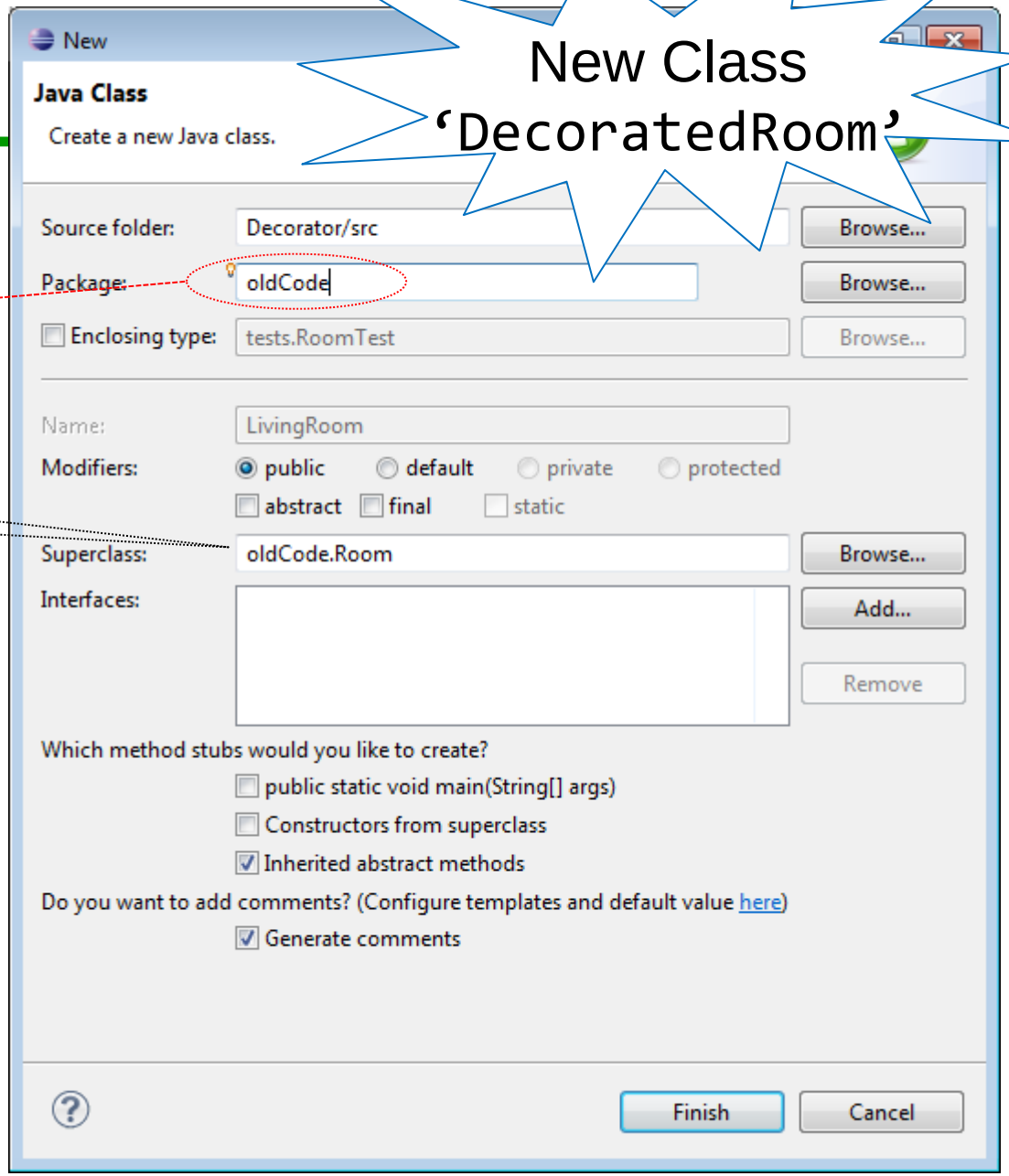
```
@Test
public void test2() {
    String expectedResult = "LivingRoom";
    String actualResult = null;

    Room classUnderTest = new LivingRoom();

    actualResult = classUnderTest.whatKindOfRoomAreYou();

    assertEquals("Wrong answer! ", expectedResult, actualResult);
}
```

Changed

Decorator

1. Make sure

Can be selected manually



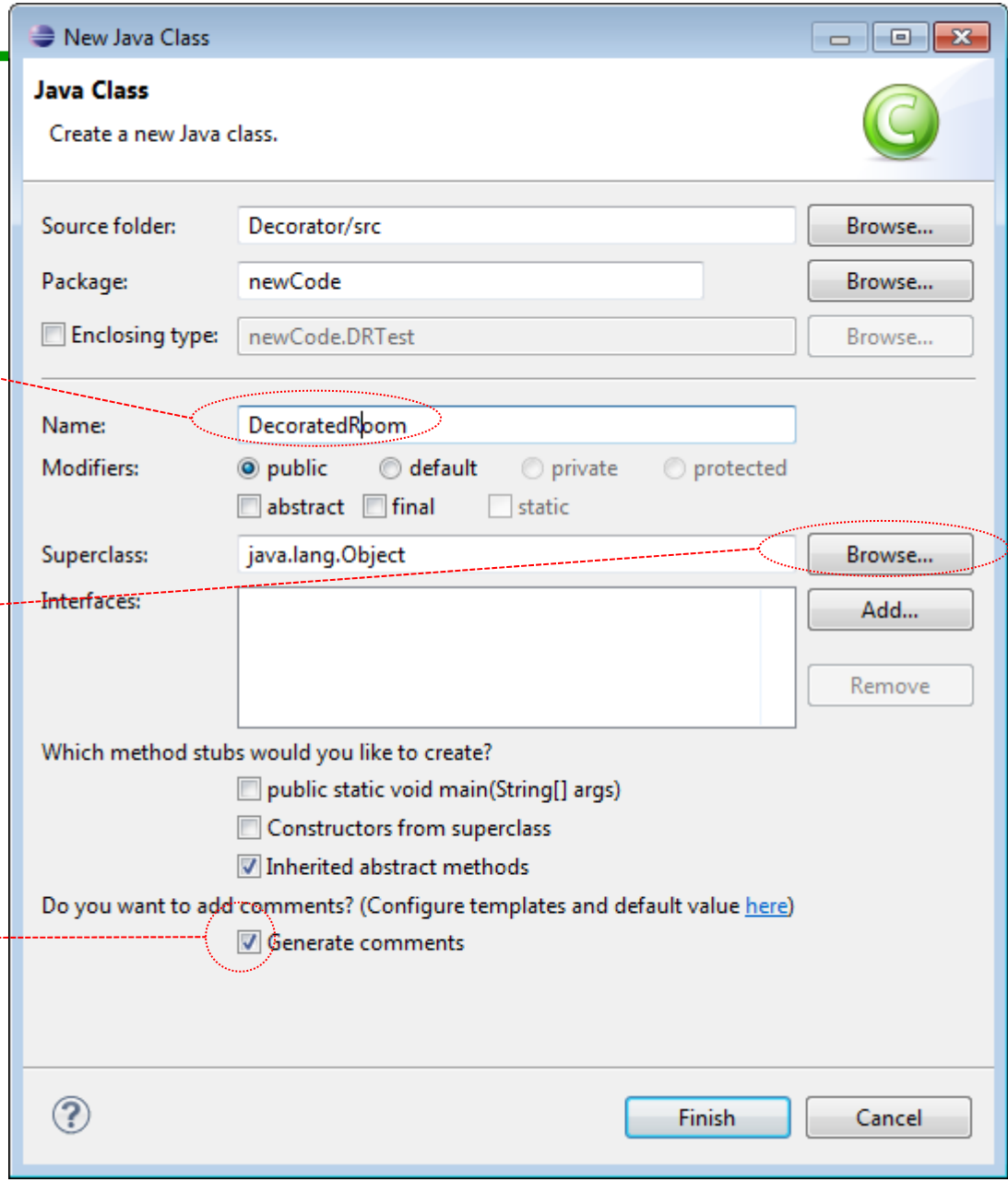
Run tests if expected failure then commit. Fix problem and when test passes commit again.

Decorator –new class ‘DecoratedRoom’ in newCode

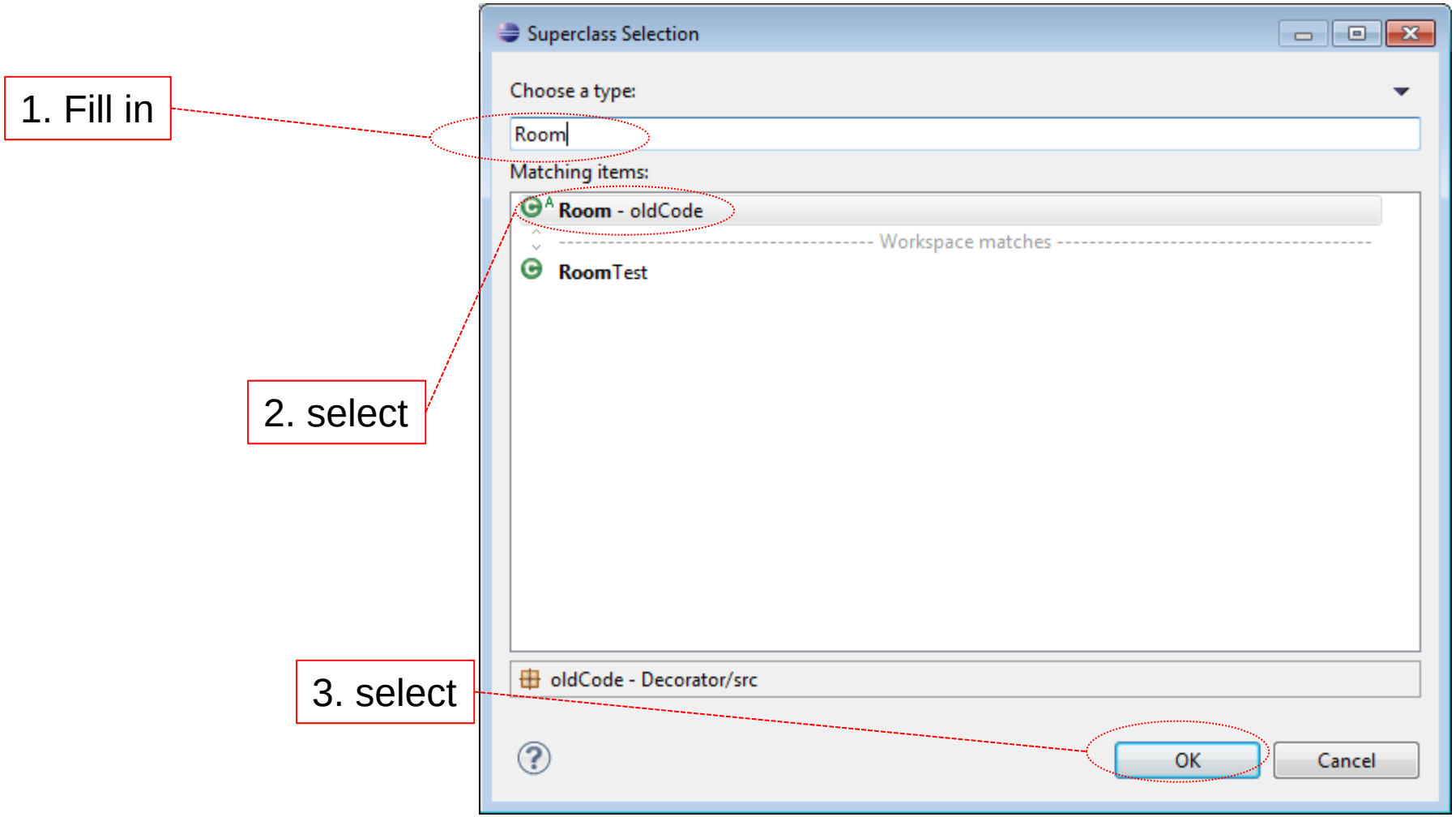
1. Fill in

3. select

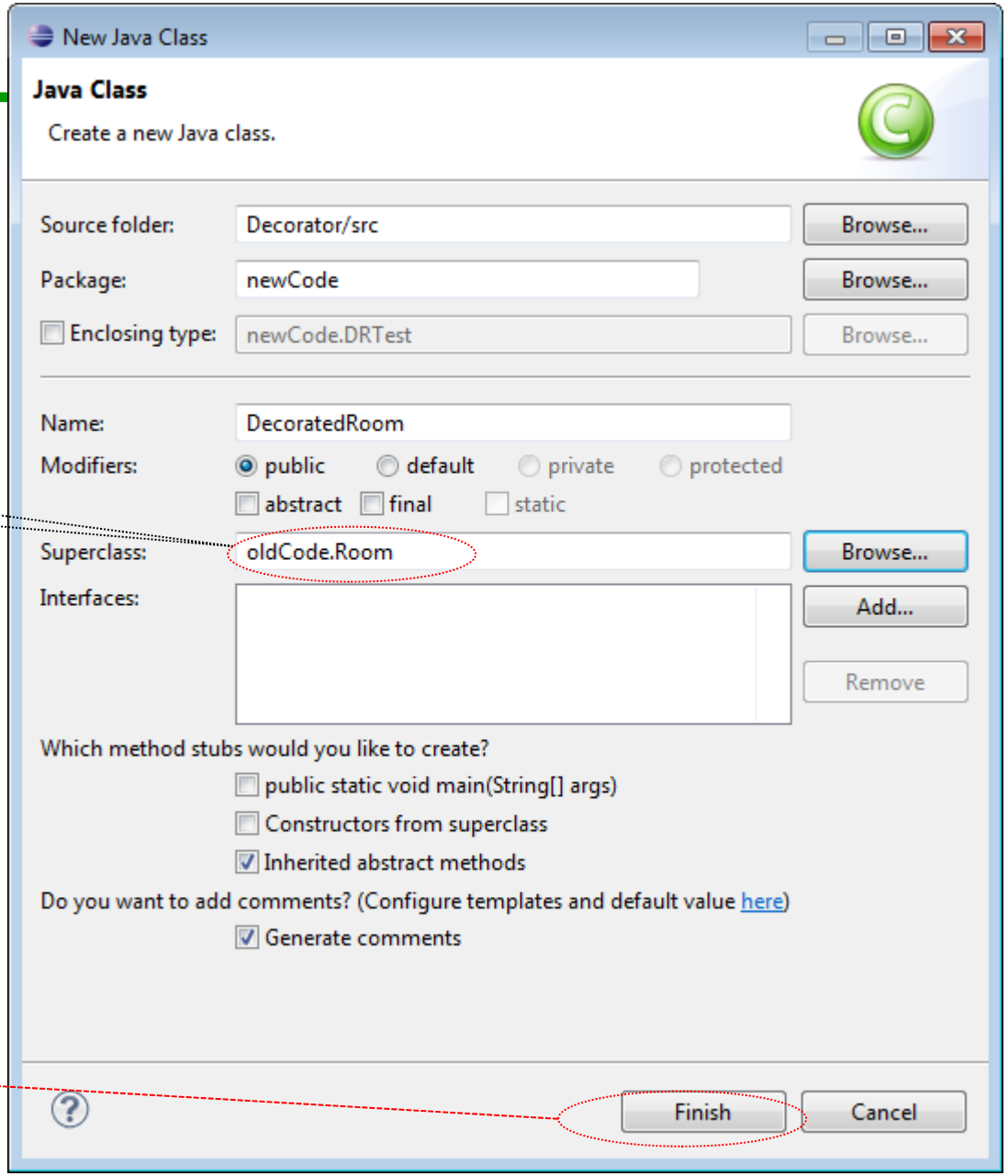
2. select



Decorator –new class in newCode



Decorator



Was added

Select

Decorator

What you got

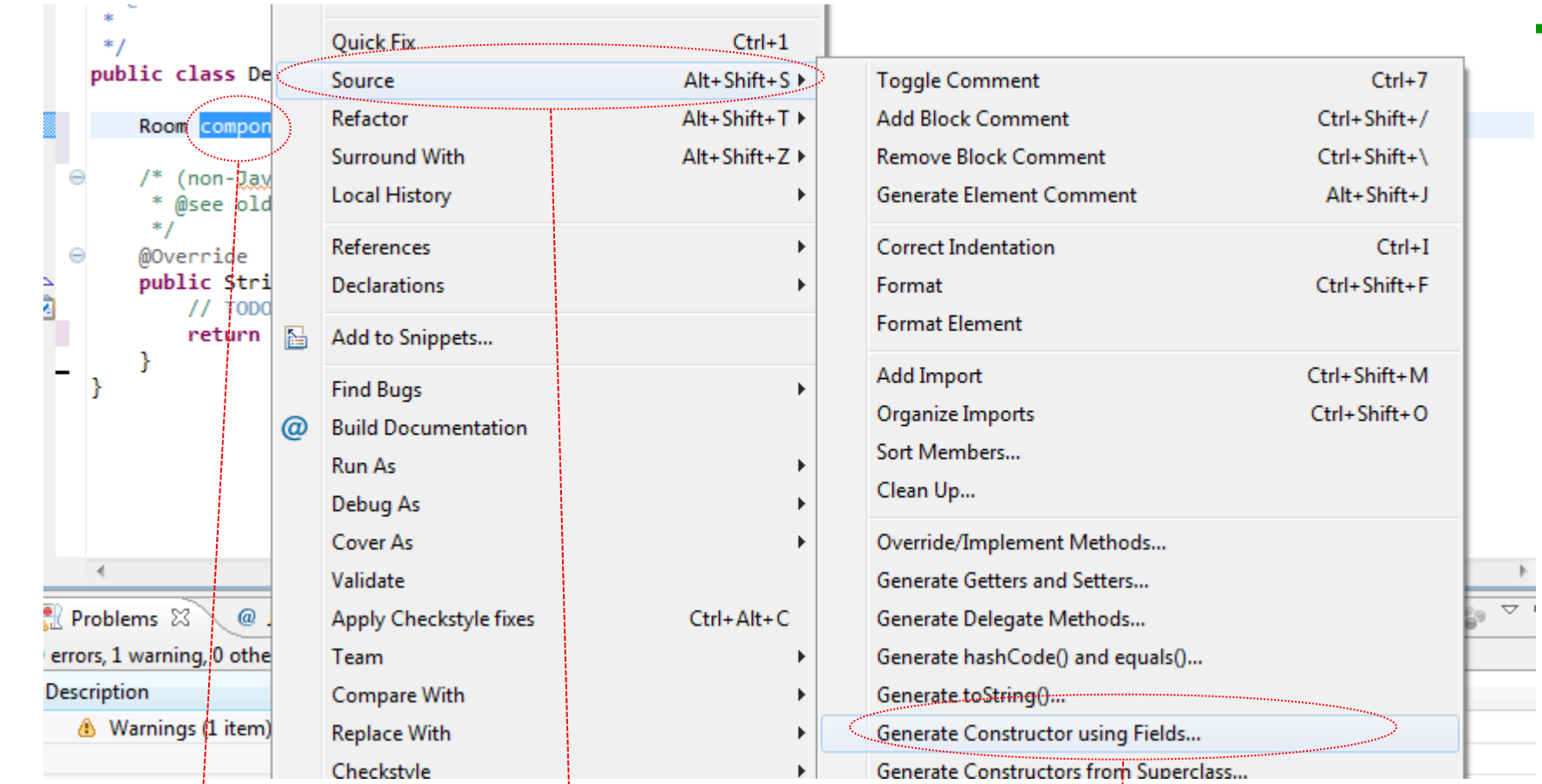
```
public class DecoratedRoom extends Room {

    /* (non-Javadoc)
     * @see oldCode.Room#whatKindOfRoomAreYou()
     */
    @Override
    public String whatKindOfRoomAreYou() {
        // TODO Auto-generated method stub
        return null;
    }
}
```

1. Add

1. Change to

```
 */
public class DecoratedRoom extends Room {

    Room component;

    /* (non-Javadoc)
     * @see oldCode.Room#whatKindOfRoomAreYou()
     */
    @Override
    public String whatKindOfRoomAreYou() {
        // TODO Auto-generated method stub
        return component.whatKindOfRoomAreYou();
    }
}
```

Decorator – still working on ‘DecoratedRoom’

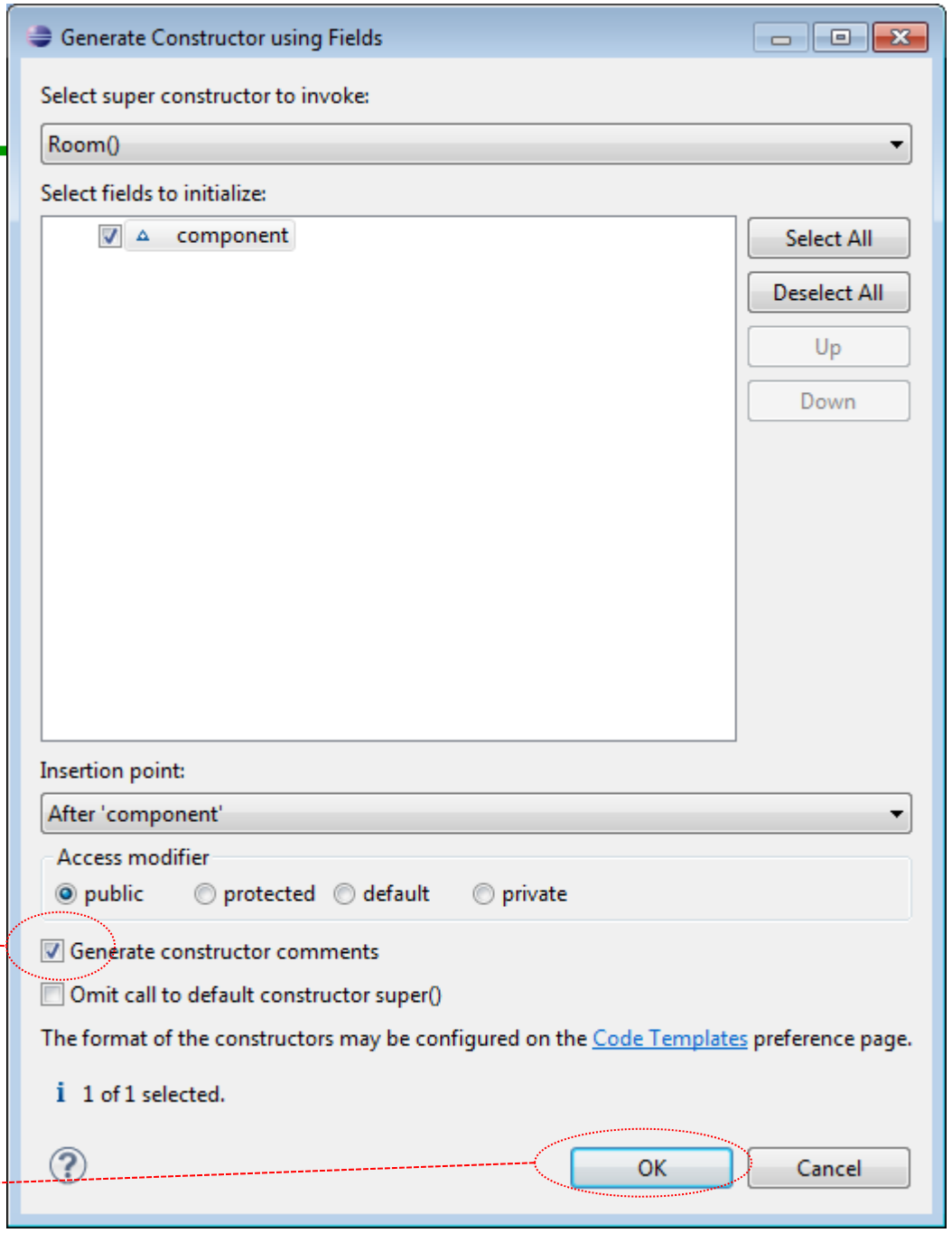


1. Select

2. Select

3. Select

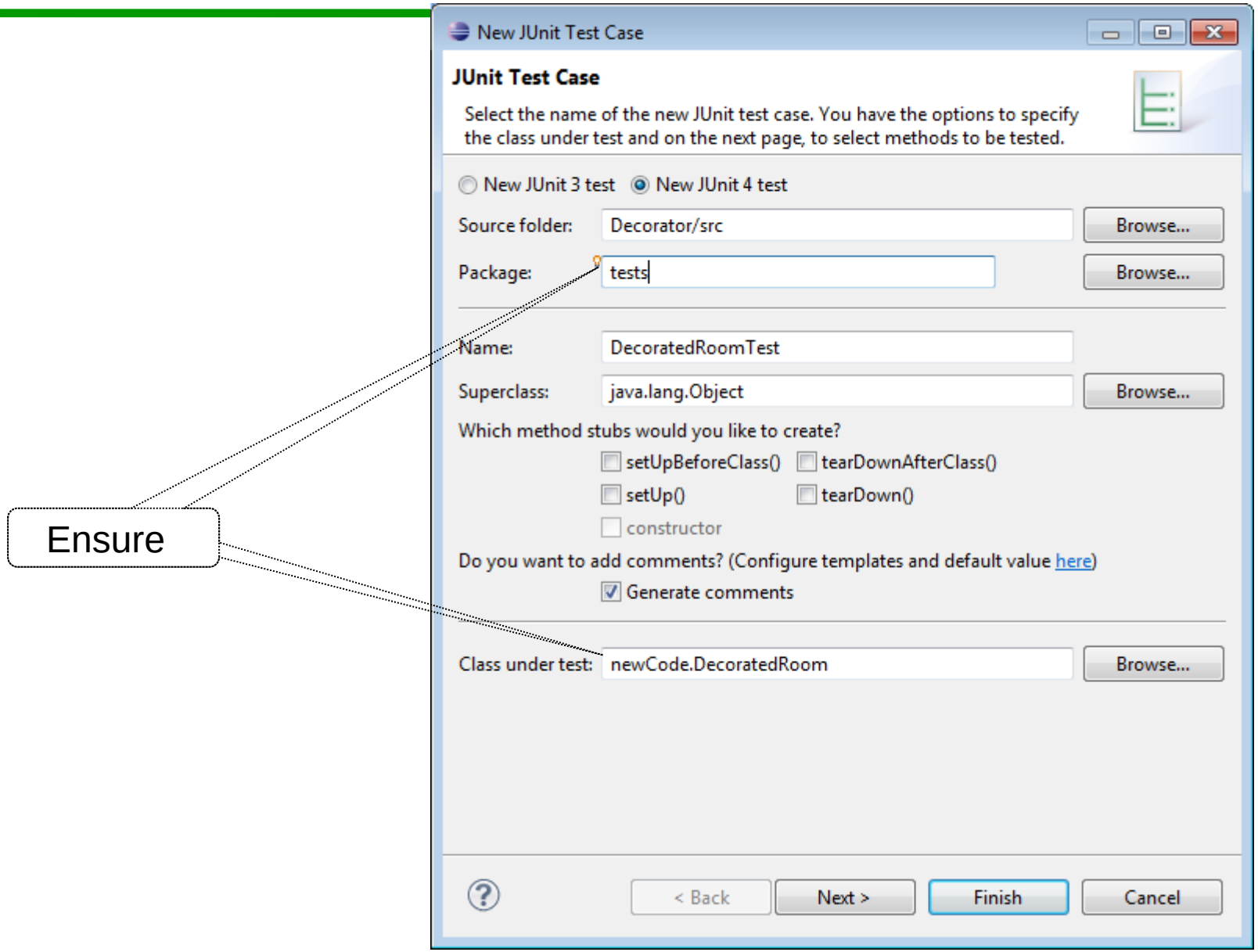
Decorator



1. Select

2. Select

Decorator – create test for ‘DecoratedRoom’



Decorator –

New test
'DecoratedRoomTest'

```
@Test
public void test() {
        String expectedResult = "BedRoom";
        String actualResult = null;

        DecoratedRoom classUnderTest = new DecoratedRoom(new BedRoom());

        actualResult = classUnderTest.whatKindOfRoomAreYou();

        assertEquals("Wrong answer! ", expectedResult, actualResult);
}
```

1. Change test code to this

2. Run test. If passes like it should, then commit

Decorator

New test
‘DecoratedRoomWithSwitchTest’

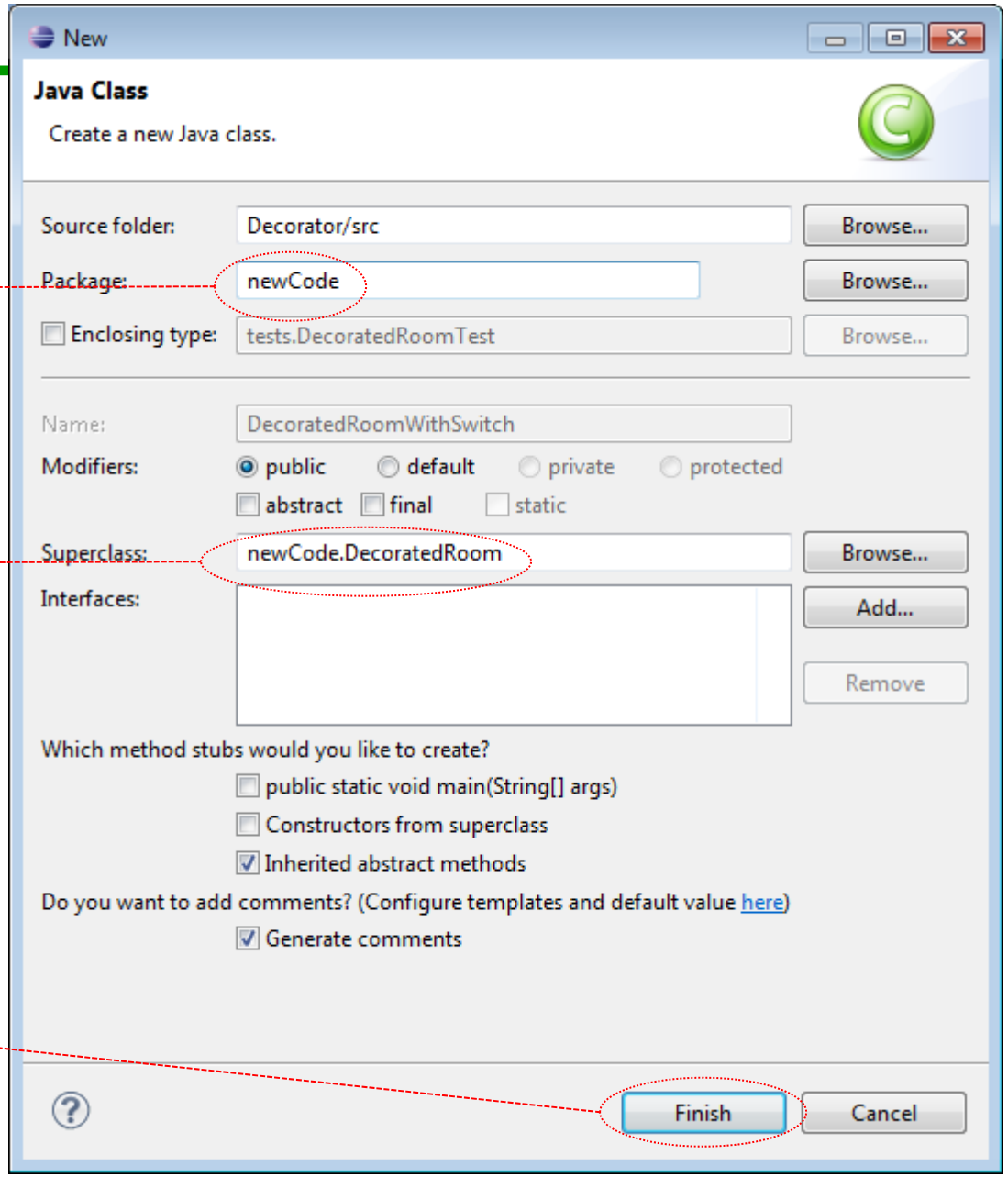
```
@Test
public void test() {
        String expectedResult = “yes";
        String actualResult = null;

        DecoratedRoomWithSwitch classUnderTest = new DecoratedRoomWithSwitch(new BedRoom());

        actualResult = classUnderTest.isLightOn();

        assertEquals("Wrong answer! ", expectedResult, actualResult);
}
```

Decorator – new sub - class

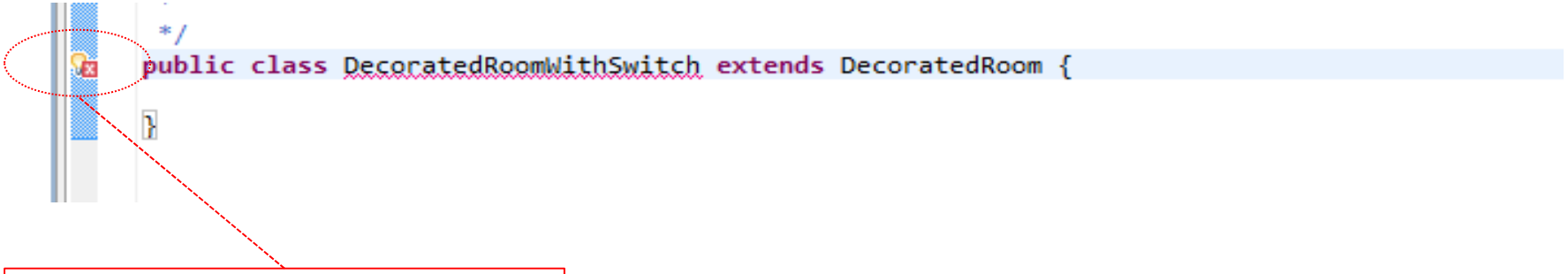
1.Ensure

2.Ensure

3.Ensure



Decorator – fixing code



1. Use to create constructor

2. Use Test code to create missing method

3. Run test if it fails properly, then commit

Decorator – code for class with switch

```
public class DecoratedRoomWithSwitch extends DecoratedRoom {

private String isLO;

public DecoratedRoomWithSwitch(Room component) {
          super(component);
           isLO = "yes";
          }

public String isLightOn() {
          return isLO;
          }
}
```

1. Run test. If passes like it should, then commit