

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 “example”, “tests”
5. Create a new Junit test “PersonTest” in package tests.

“PersonTest” Code in next slide

Select

Adapter – first classes for example

```
@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'

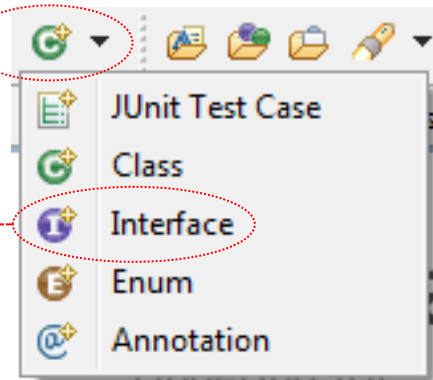
3. Run test. If it fails properly, then commit

4. Implement method 'whoAreYou' in class 'Person'

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

Adapter (using inheritance) – new interface

1. Create a new interface with the following code



```
public interface AnotherPerson {  
    public abstract String youWho();  
}
```

Adapter – new test in new file

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

    AnotherPerson classUnderTest = new Nosrep();

    actualAnswer = classUnderTest.youAreWho();

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

1. Use IDE to create class 'Nosrep' check that it is in the proper package

2. Use IDE to create method 'who' in class 'Nosrep'

3. Run test if fails properly, then commit

Adapter – code for class 'Nosrep'

```
public class Nosrep implements AnotherPerson {  
  
    private Person person;  
  
    public Nosrep(){  
        person = new Person();  
    }  
  
    public String youWho() {  
        return person.whoAreYou();  
    }  
}
```

1. Update code

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

This is an example of adapter using delegation



Adapter – new test in new file 'PAPTest'

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

    PAP classUnderTest = new PAP();

    actualAnswer = classUnderTest.youWho();

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

1. Use IDE to create class 'PAP' as seen in the next slide

Adapter

The image shows a 'New Java Class' dialog box with the following fields and options:

- Source folder: adapter/src
- Package: example
- Enclosing type: tests.PAPTest
- Name: PAP
- Modifiers: public (selected), default, private, protected, abstract, final, static
- Superclass: example.Person
- Interfaces: example.AnotherPerson
- Which method stubs would you like to create?
 - public static void main(String[] args)
 - Constructors from superclass
 - Inherited abstract methods
- Do you want to add comments? (Configure templates and default value [here](#))
 - Generate comments
- Buttons: Finish, Cancel

Annotations on the left side of the dialog:

1. ensure (points to the Package field)
2. select (points to the Superclass field)
3. select (points to the Interfaces field)
4. select (points to the 'Generate comments' checkbox)
5. select (points to the Finish button)

Adapter – PAP class generated code

```
-  
*/  
public class PAP extends Person implements AnotherPerson {  
  
    /* (non-Javadoc)  
    * @see example.AnotherPerson#youWho()  
    */  
    @Override  
    public String youWho() {  
        // TODO Auto-generated method stub  
        return null;  
    }  
}
```

1. Run test. If test fails properly, then commit

Adapter – PAP class implemented

```
-  
*/  
public class PAP extends Person implements AnotherPerson {  
  
    /* (non-Javadoc)  
    * @see example.AnotherPerson#youWho()  
    */  
    @Override  
    public String youWho() {  
        // TODO Auto-generated method stub  
        return null;  
    }  
}
```

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

Select

Adapter

```
~/  
public class PAP extends Person implements AnotherPerson {  
  
    /* (non-Javadoc)  
    * @see example.AnotherPerson#youWho()  
    */  
    @Override  
    public String youWho() {  
        // TODO Auto-generated method stub  
        return whoAreYou();  
    }  
}
```

1. Change code to thist

2. Run test. If it passes like it should, 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();

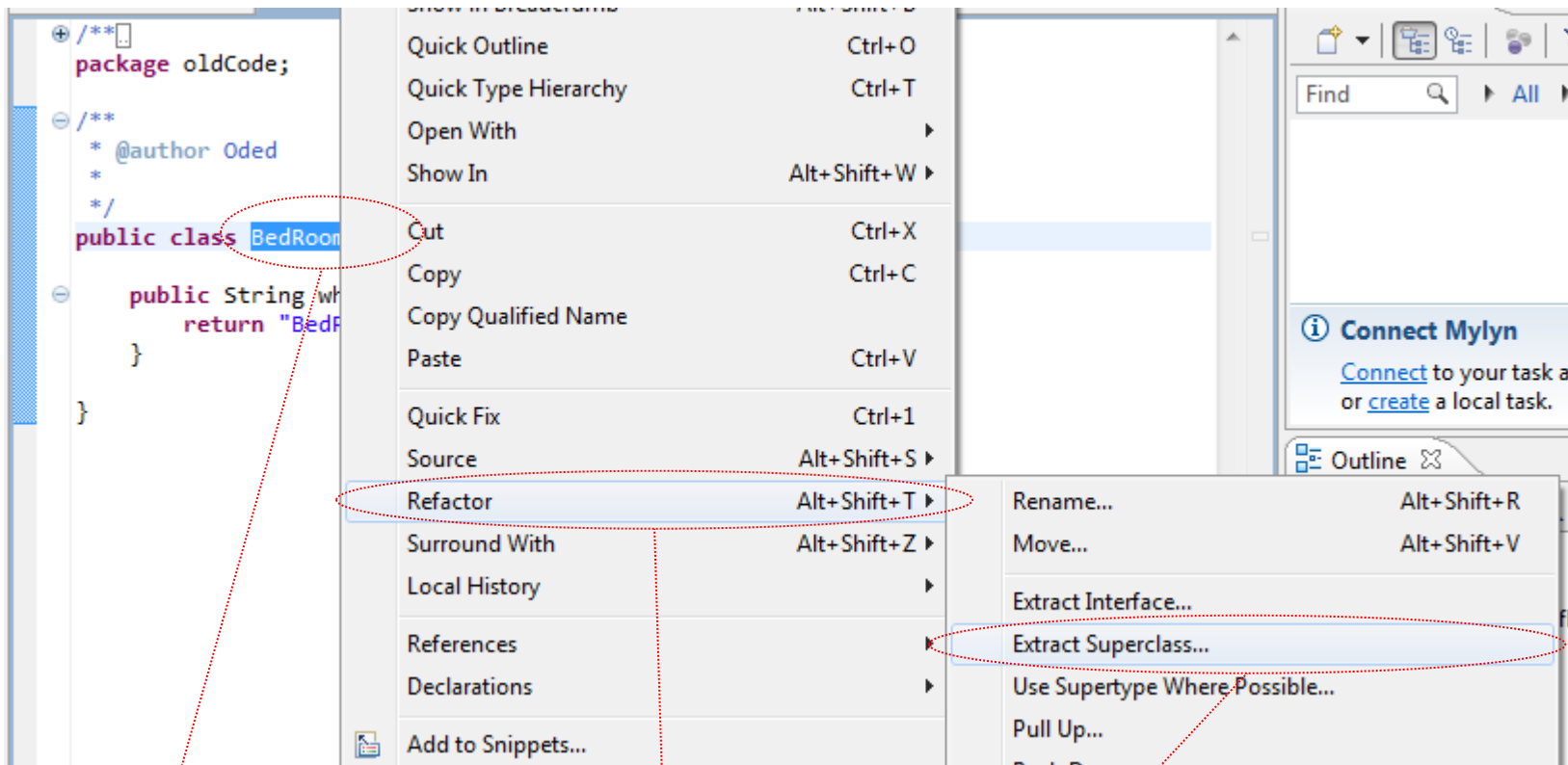
        actualResult = classUnderTest.whatKindOfRoomAreYou();

        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"



1. Select

2. Select

3. Select

Decorator

1. Fill in

2. Don't Select!

3. Select

Refactoring

Extract Superclass

Select the members to extract to the new type.

Superclass name:

Use the extracted class where possible

- Use the extracted class in 'instanceof' expressions

Create necessary methods stubs in non-abstract subtypes of the extracted type

Types to extract a superclass from:

- BedRoom - oldCode

Buttons: Add..., Remove

Specify actions for members:

Member	Action
<input type="checkbox"/> whatKindOfRoomAreYou()	

Buttons: Select All, Deselect All, Set Action..., Add Required

0 members selected.

Buttons: ? < Back Next > **Finish** Cancel

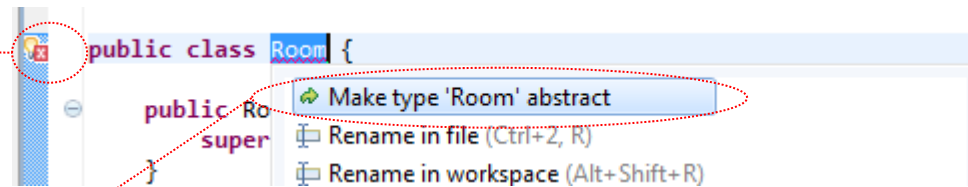
Decorator

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

1. Use Refactor to extract Interface from class "BedRoom" call the interface "Room"
2. Run test, if passes then commit
3. Create new test by copy past from the existing one
4. Use test to generate "LivingRoom" class (remember it should implement Interface "Room" and be in package "oldCode")
5. 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

New

Java Class

Create a new Java class.

Source folder: Decorator/src

Package: **oldCode**

Enclosing type: tests.RoomTest

Name: LivingRoom

Modifiers: public default private protected
 abstract final static

Superclass: oldCode.Room

Interfaces:

Which method stubs would you like to create?

public static void main(String[] args)
 Constructors from superclass
 Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))

Generate comments

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

Decorator – new class 'DecoratedRoom' in newCode

1. Fill in

New Java Class

Java Class

Create a new Java class.

Source folder: Decorator/src

Browse...

Package: newCode

Browse...

Enclosing type: newCode.DRTTest

Browse...

Name: DecoratedRoom

Modifiers: public default private protected

abstract final static

Superclass: java.lang.Object

Browse...

Interfaces:

Add...

Remove

Which method stubs would you like to create?

public static void main(String[] args)

Constructors from superclass

Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))

Generate comments

3. select

2. select

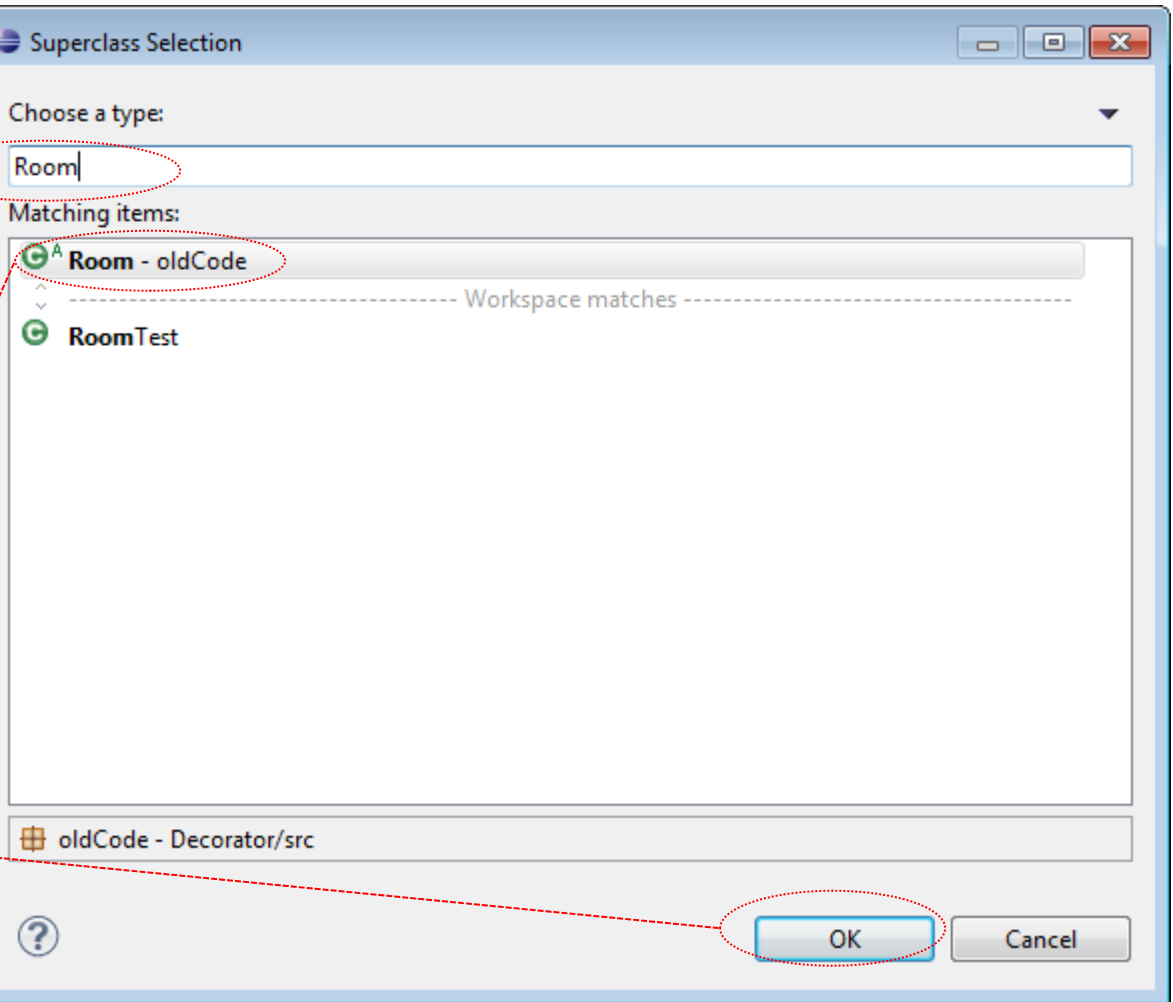


Finish

Cancel

Decorator – new class in newCode

1. Fill in



2. select

3. select

Decorator

New Java Class

Java Class

Create a new Java class.

Source folder: Decorator/src Browse...

Package: newCode Browse...

Enclosing type: newCode.DRTTest Browse...

Name: DecoratedRoom

Modifiers: public default private protected
 abstract final static

Superclass: oldCode.Room Browse...

Interfaces: Add...
Remove

Which method stubs would you like to create?

- public static void main(String[] args)
- Constructors from superclass
- Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))

- Generate comments

Finish Cancel

Was added

Select

Decorator

What you got

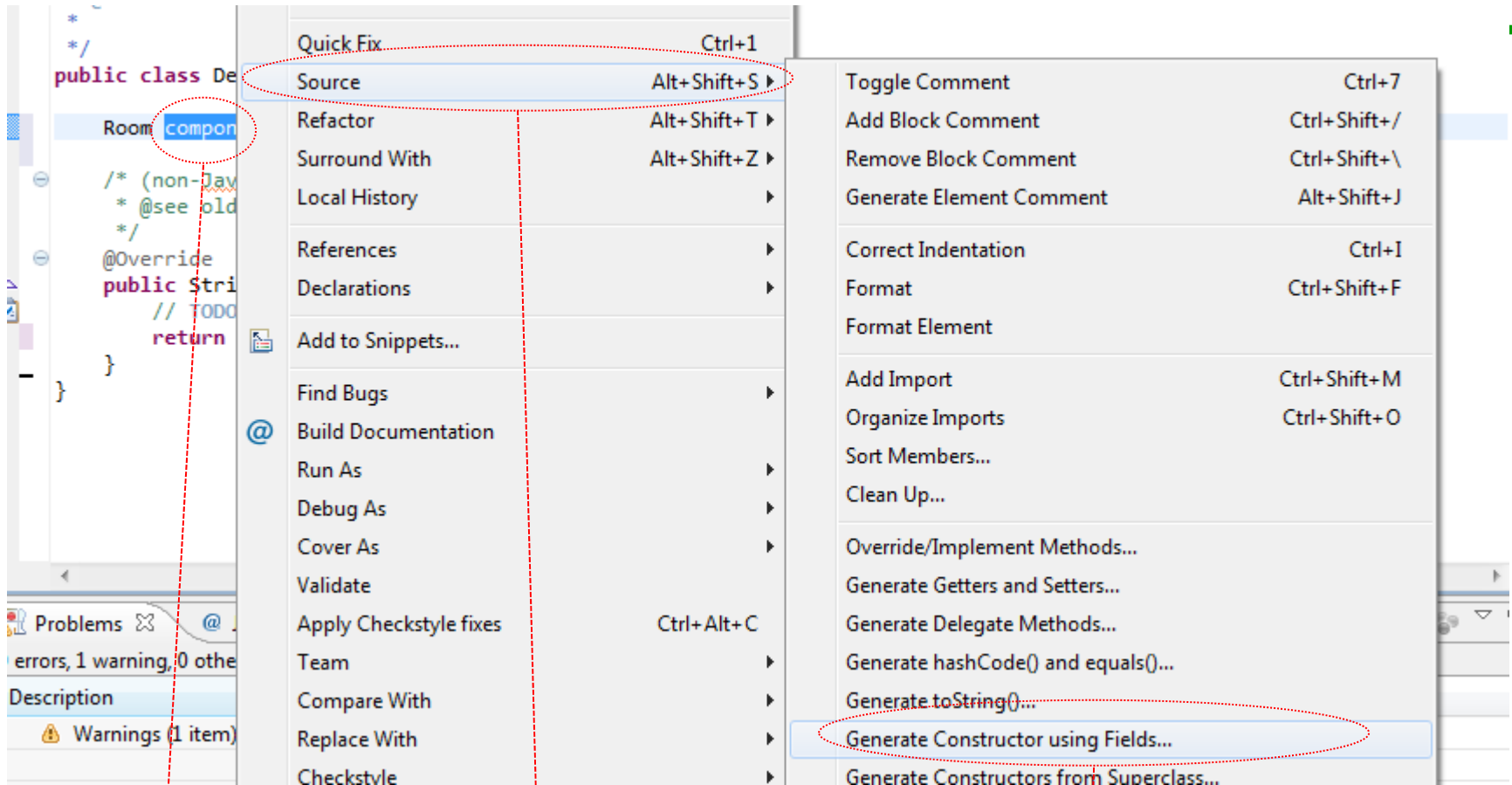
```
public class DecoratedRoom extends Room {  
    /* (non-Javadoc)  
    * @see oldCode.Room#whatKindOfRoomAreYou()  
    */  
    @Override  
    public String whatKindOfRoomAreYou() {  
        // TODO Auto-generated method stub  
        return null;  
    }  
}
```

1. Add

```
    /*  
    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();  
        }  
    }  
}
```

1. Change to

Decorator – still working on ‘DecoratedRoom’

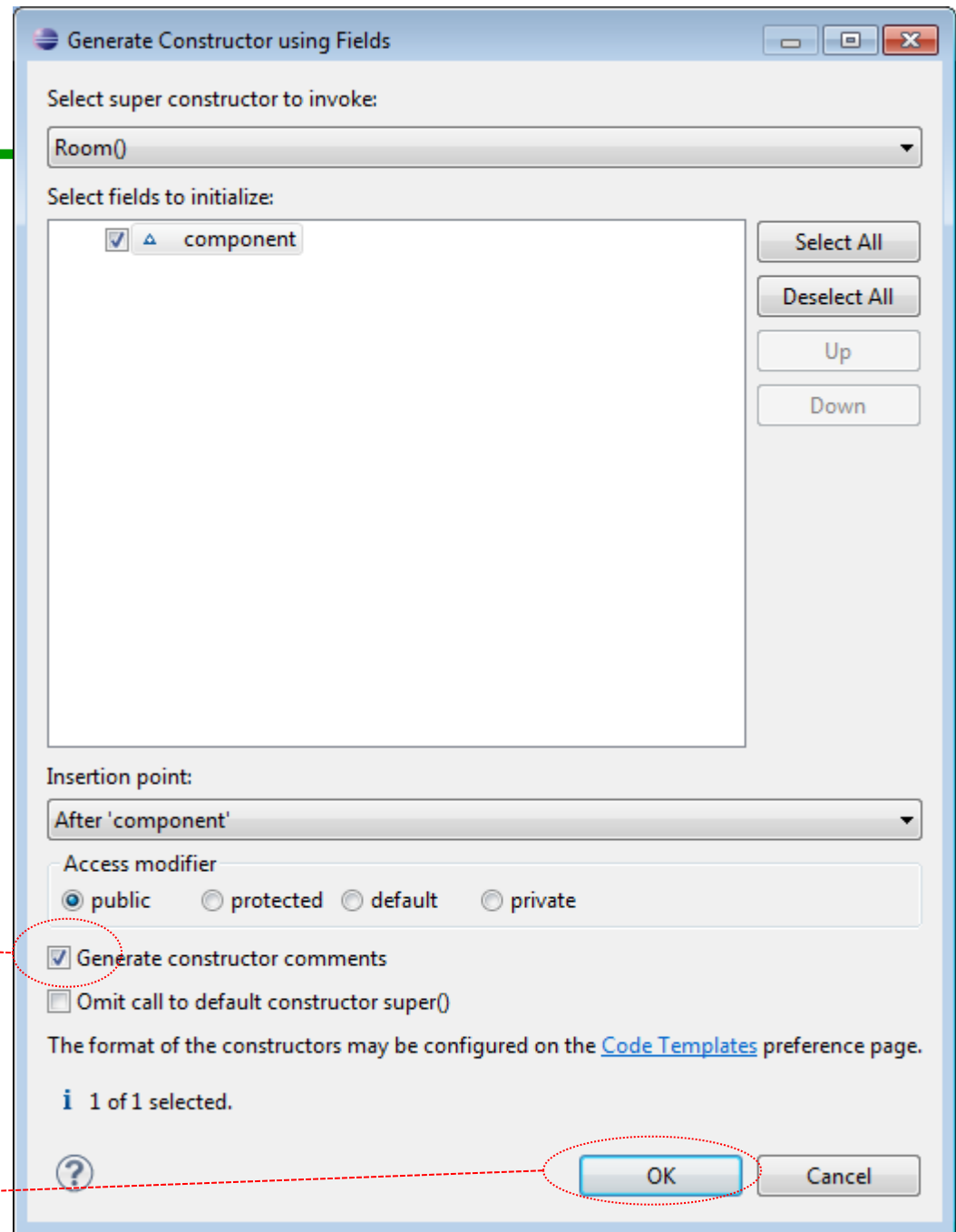


1. Select

2. Select

3. Select

Decorator



1. Select

2. Select

Decorator – create test for ‘DecoratedRoom’

New JUnit Test Case

Select the name of the new JUnit test case. You have the options to specify the class under test and on the next page, to select methods to be tested.

New JUnit 3 test New JUnit 4 test

Source folder: Decorator/src

Package: tests

Name: DecoratedRoomTest

Superclass: java.lang.Object

Which method stubs would you like to create?

setUpBeforeClass() tearDownAfterClass()
 setUp() tearDown()
 constructor

Do you want to add comments? (Configure templates and default value [here](#))

Generate comments

Class under test: newCode.DecoratedRoom

Ensure

Decorator – code for '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 ‘DecoratedRoomWithSwitch’

```
@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

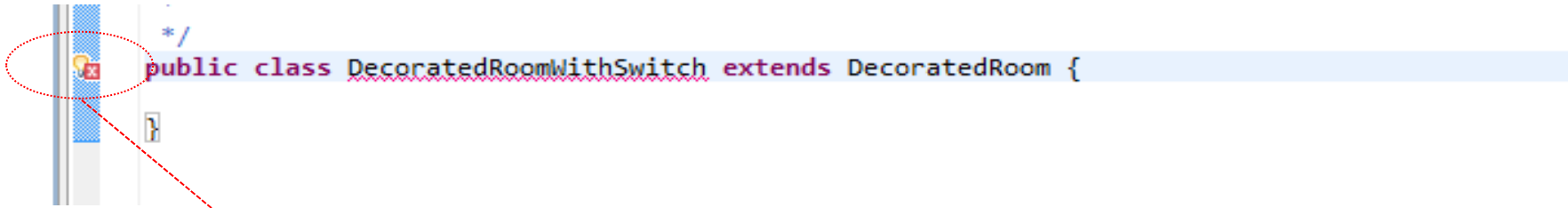
3.Ensure

The screenshot shows the 'New' dialog box in an IDE, titled 'New'. The main heading is 'Java Class' with the instruction 'Create a new Java class.' and a green 'C' icon. The dialog is divided into several sections:

- Source folder:** Decorator/src (with a 'Browse...' button)
- Package:** newCode (circled in red, with a 'Browse...' button)
- Enclosing type:** tests.DecoratedRoomTest (with a 'Browse...' button)
- Name:** DecoratedRoomWithSwitch
- Modifiers:** public (selected), default, private, protected, abstract, final, static
- Superclass:** newCode.DecoratedRoom (circled in red, with a 'Browse...' button)
- Interfaces:** (empty list, with 'Add...' and 'Remove' buttons)
- Which method stubs would you like to create?**
 - public static void main(String[] args)
 - Constructors from superclass
 - Inherited abstract methods
- Do you want to add comments? (Configure templates and default value [here](#))**
 - Generate comments

At the bottom, there is a 'Finish' button (circled in red) and a 'Cancel' button.

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