

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

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

    Person classUnderTest = new Person ();

    actualAnswer = classUnderTest.whoAreYou();

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



In new file
"PersonTest"

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 X
+ /**
  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

The screenshot shows an IDE with several tabs: PersonTest.java, Person.java, NesropTest.java, and Nesrop.java. The Nesrop.java tab is active and contains the following code:

```
package tests;

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

    public String youWho() {
        return "Nesrop A";
    }

}
```

A context menu is open over the code, listing various actions such as Undo Typing, Save, eUML2, Cut, Copy, Paste, and Refactor. The Refactor option is highlighted, and a submenu is open below it, showing options like Move... and Extract Interface... The Extract Interface... option is highlighted in the submenu.

2. Right click

3. select

Move... Alt+Shift+V

Extract Interface...

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

4. select

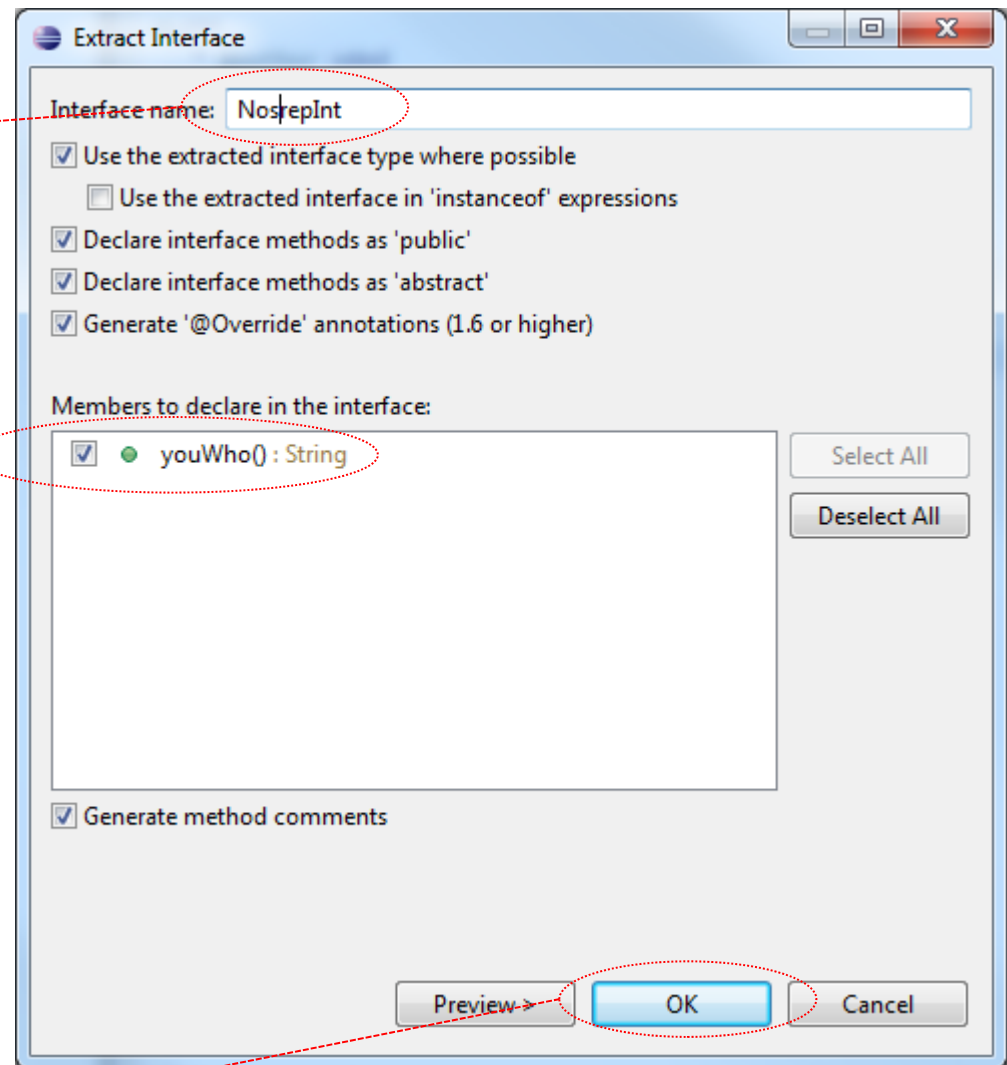
Adapter

1. Fill in

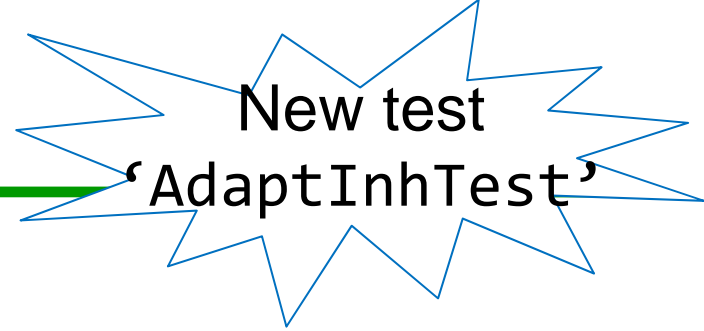
2. select

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

3. select



Adapter (using inheritance)



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

Java Class
Create a new Java class.

Source folder: Adapater/src

Package: newCode

Enclosing type: tests.AdaptInhTest

Name: AdaptInh

Modifiers: public default private protected
 abstract final static

Superclass: java.lang.Object

Interfaces: oldCode.NosrepInt

Which method stubs would you like to create?
 public static void main(String[] args)
 Constructors from superclass
 Inherited abstract methods

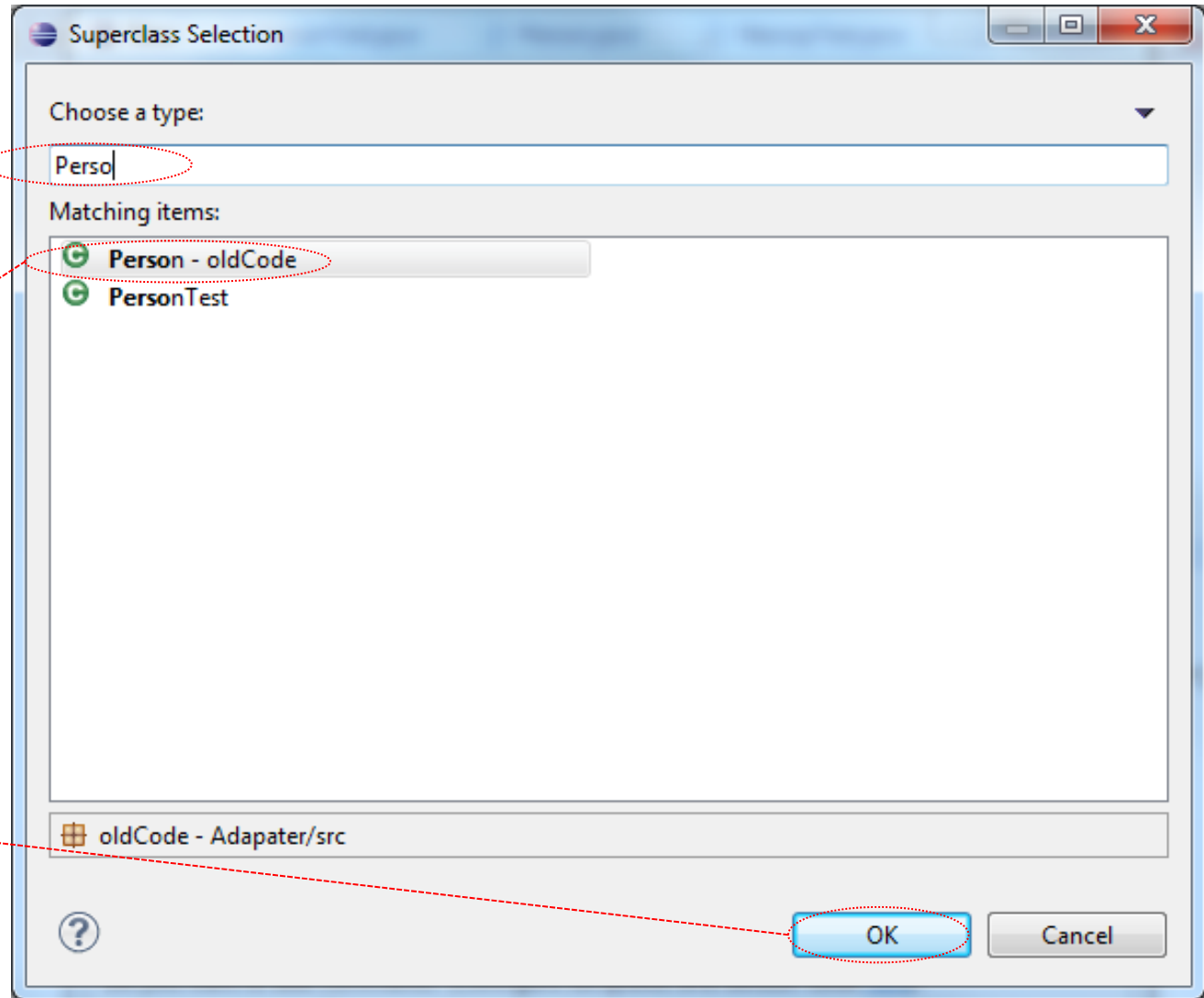
Do you want to add comments? (Configure templates and default value [here](#))
 Generate comments

1.ensure

2.select

Observe

Adapter (using inheritance) – new interface



Adapter (using inheritance)

The screenshot shows the 'New Class' dialog box in an IDE. The dialog is titled 'New Class' and contains the following fields and options:

- Source folder:** Adapater/src
- Package:** newCode
- Enclosing type:** tests.AdaptInhTest
- Name:** AdaptInh
- Modifiers:** public, default, private, protected, abstract, final, static
- Superclass:** oldCode.Person
- Interfaces:** oldCode.NosrepInt
- 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 of the dialog, there are three buttons: a help button (question mark), a 'Finish' button, and a 'Cancel' button. The 'Finish' button is highlighted with a blue border and a dotted line from the '3. Select' text box.

Observe

3. Select

Adapter (using inheritance) – new interface

```
/**\n package newCode;\n\n import oldCode.NosrepInt;\n\n /**\n  * @author oded\n  *\n  */\n public class AdaptInh extends Person implements NosrepInt {\n\n     /* (non-Javadoc)\n     * @see oldCode.NosrepInt#youWho()\n     */\n     @Override\n     public String youWho() {\n         // TODO Auto-generated method stub\n         return null;\n     }\n\n }
```

Code You should get

1. Run test if it fails properly, then commit

2. Update code

```
/**\n public class AdaptInh extends Person implements NosrepInt {\n\n     /* (non-Javadoc)\n     * @see oldCode.NosrepInt#youWho()\n     */\n     @Override\n     public String youWho() {\n         return whoAreYou();\n     }\n\n }
```

3. Run test. If it passes, then commit

Adapter (using delegation)



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)

```
⊕ /**  
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

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

2. Select



Adapter (using delegation)

1. Select

```
*/
@Override
public String youWho() {
    // TODO Auto-generated method stub
    return person.whoAreYou();
}
}
```

... public String
Object pers
// TODO Au
return perso

- Create local variable 'person'
- Create field 'person'
- Create parameter 'person'
- Create class 'person'

2. Change to 'Person'

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

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)

1. Update code

2. Select

4. Select

3. Select

```
public class AdaptDelTest {  
  
    @Test  
    public void test() {  
        String expectedAnswer = "A person";  
        String actualAnswer;  
  
        NosrepInt classUnderTest = new AdaptDel(new Person());  
  
        actualAnswer = classUnderTest.youWho();  
  
        assertEquals("Wrong Answer!", expectedAnswer, actualAnswer);  
    }  
}
```

```
NosrepInt classUnderTest = new AdaptDel(new Person());  
actualAnswer = classUnderTest.youWho();  
assertEquals("Wrong Answer!", expectedAnswer, actualAnswer);
```

- Import 'Person' (oldCode)
- Create class 'Person'
- Change to 'Permission' (java.security)

Class 'Person' is in a different package and hence needs to be imported

Adapter (using delegation)

```
String expectedAnswer = "A person";  
String actualAnswer;  
  
NosrepInt classUnderTest = new AdaptDel(new Person());  
  
actualAnswer = classUnderTest.  
assertEquals("Wrong Answer!",  
}  
...  
imp  
imp  
/**  
...  
put
```

1. Select

Adapter (using delegation)

```
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

```
    /*  
    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();

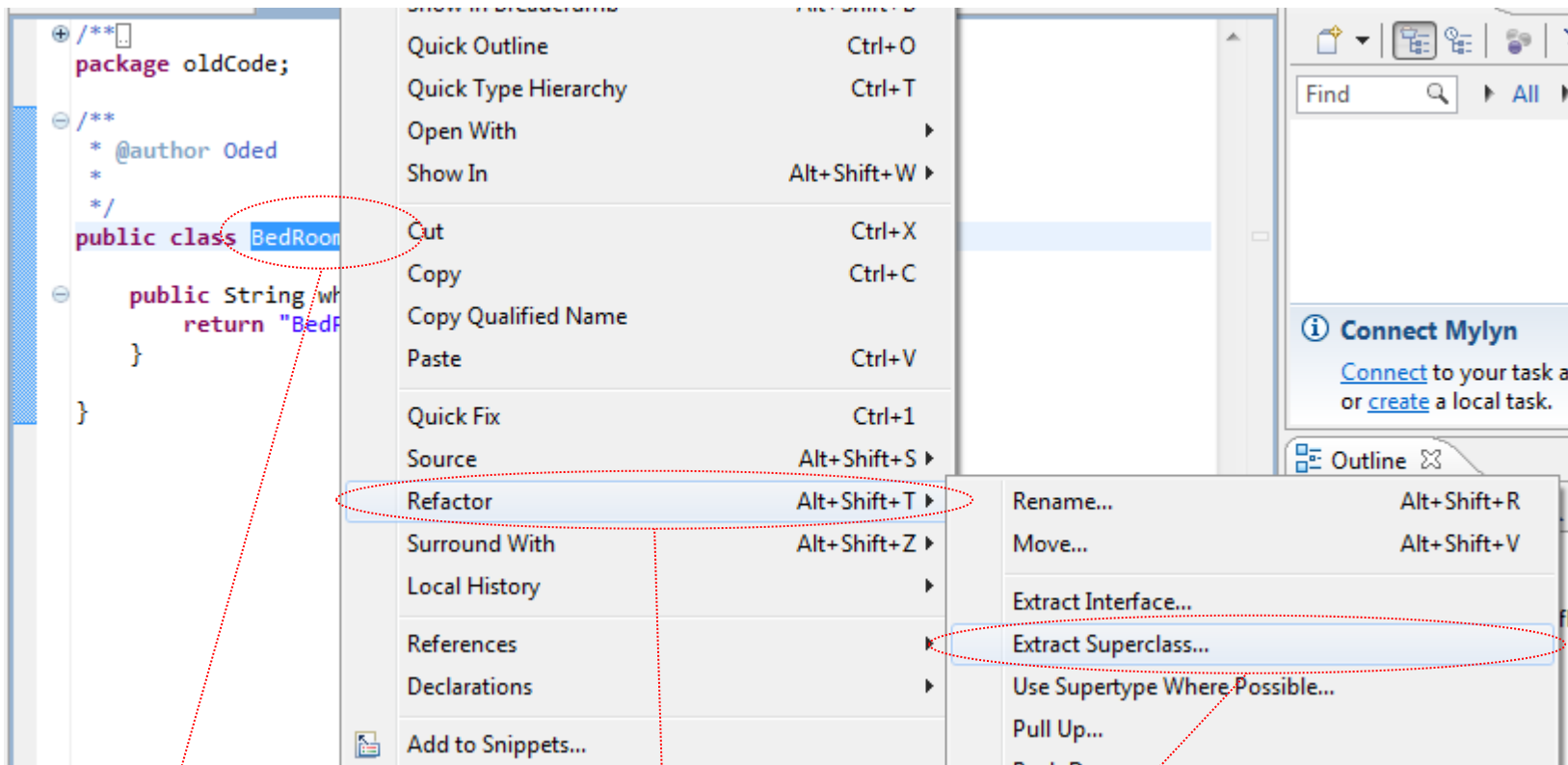
        actualResult = classUnderTest.whatKindOfRoomAreYou();

        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"



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

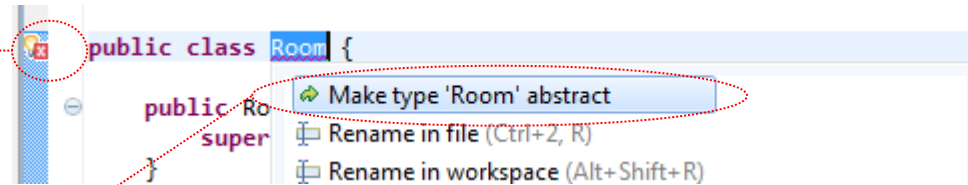
Decorator

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

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();

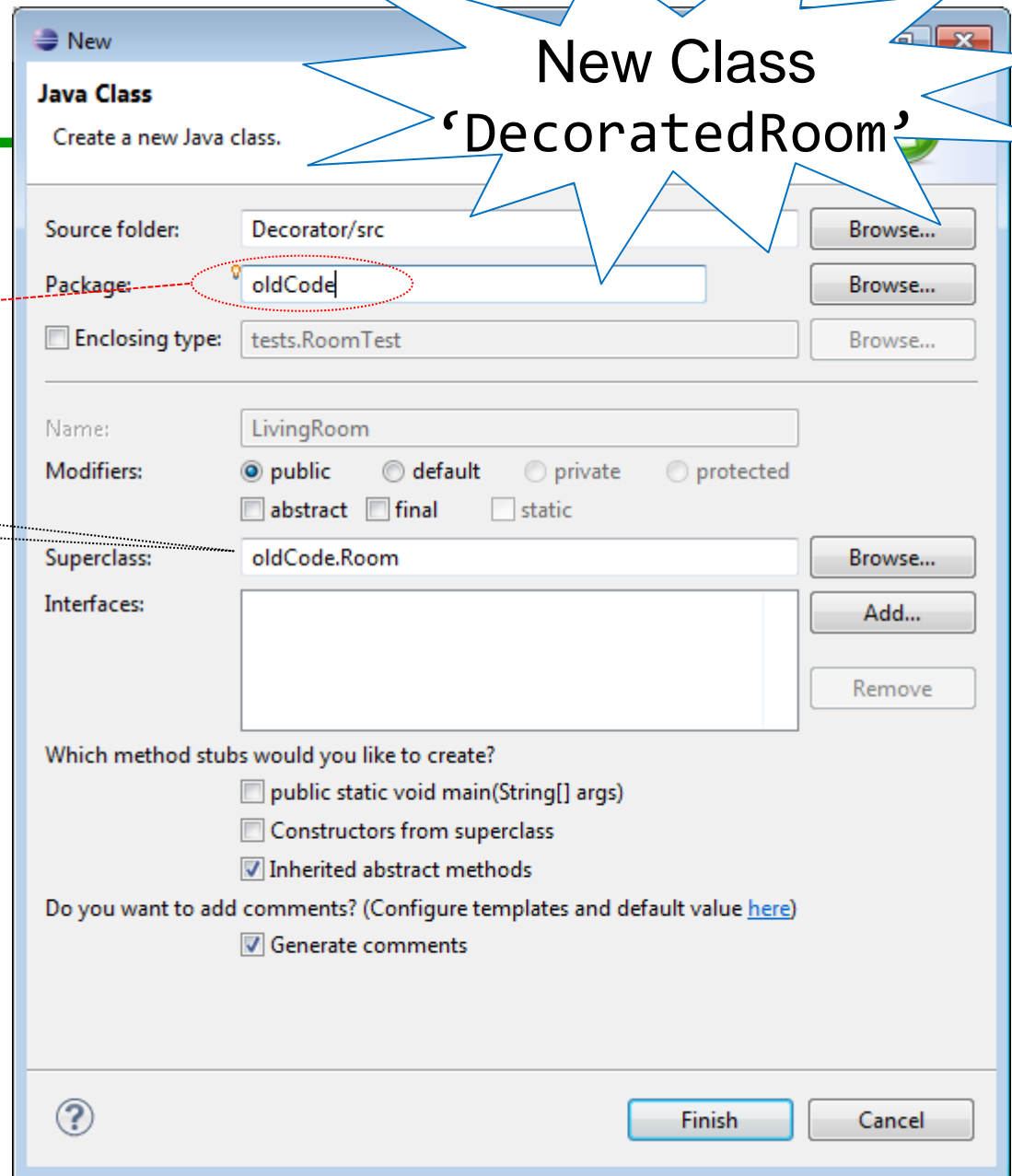
    assertEquals("Wrong answer! ", expectedResult, actualResult);
}
```

Changed

Decorator

1. Make sure

Can be selected manually



New Class

'DecoratedRoom'

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



Finish

Cancel

3. select

2. select

Decorator – new class in newCode

1. Fill in

Choose a type:
Room

2. select

Matching items:

- Room - oldCode
- RoomTest

----- Workspace matches -----

3. select

oldCode - Decorator/src

OK

Cancel

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

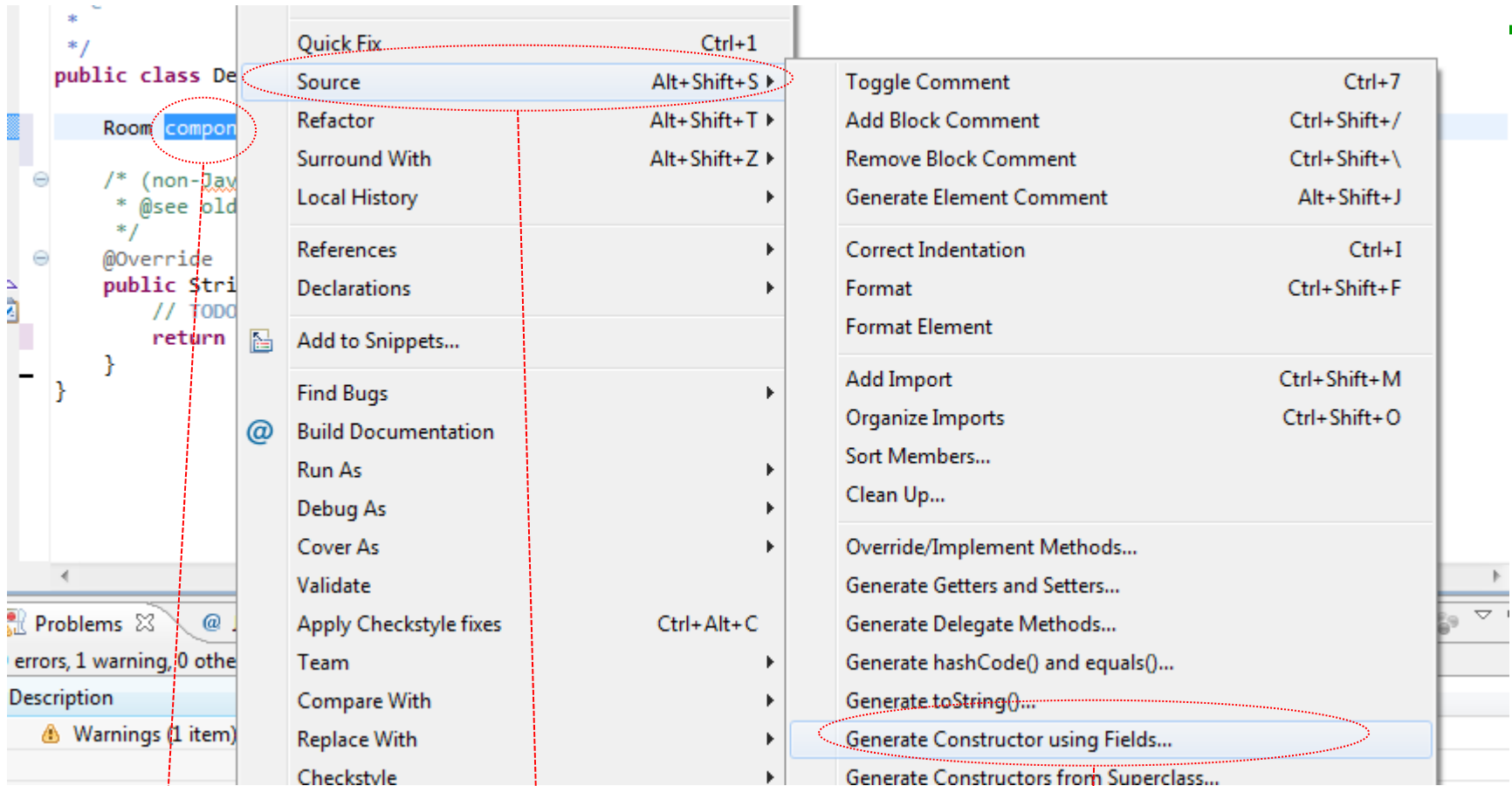
```
public class DecoratedRoom extends Room {  
    /* (non-Javadoc)  
    * @see oldCode.Room#whatKindOfRoomAreYou()  
    */  
    @Override  
    public String whatKindOfRoomAreYou() {  
        // TODO Auto-generated method stub  
        return null;  
    }  
}
```

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’

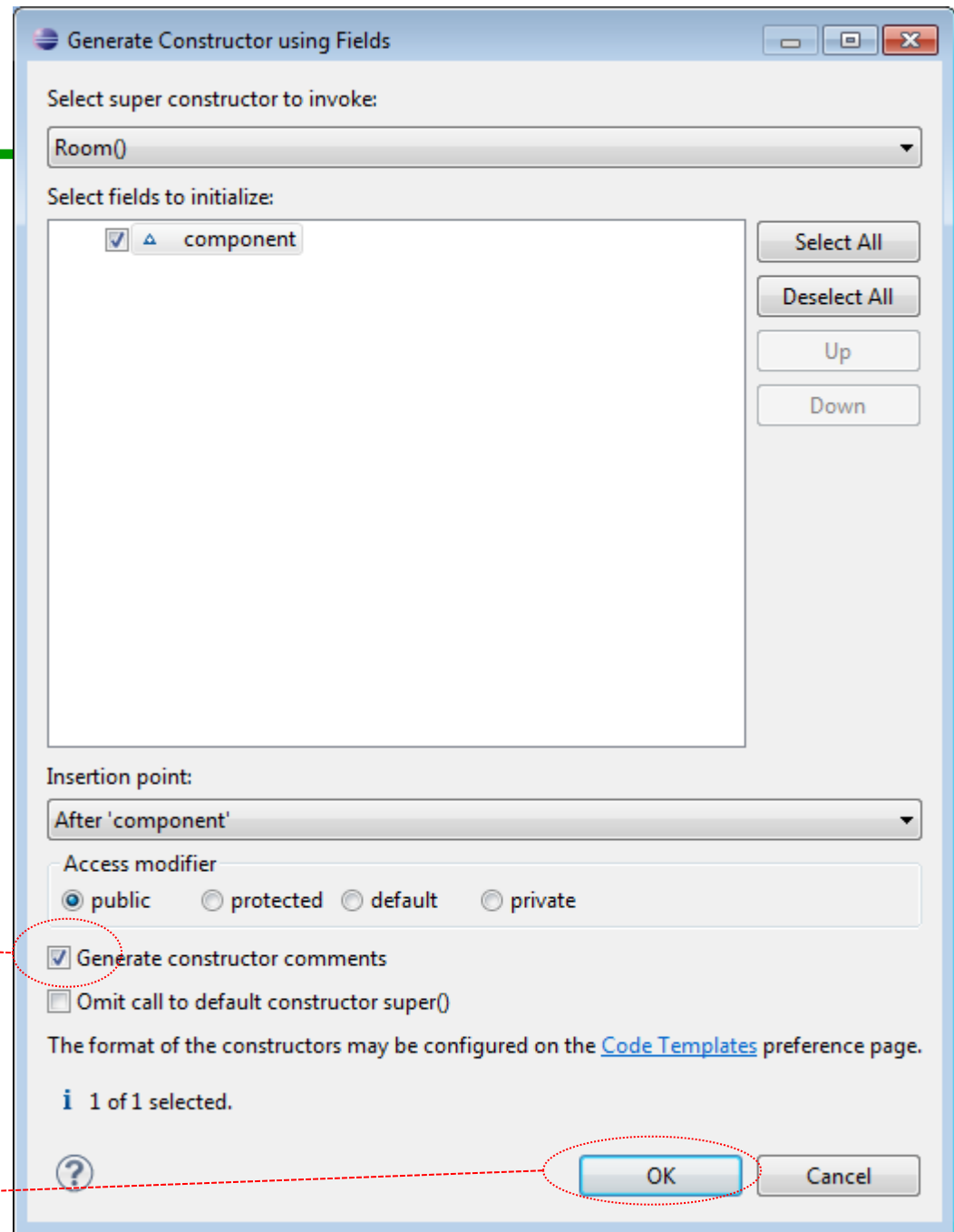


1. Select

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:

Package:

Name:

Superclass:

Which method stubs would you like to create?

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

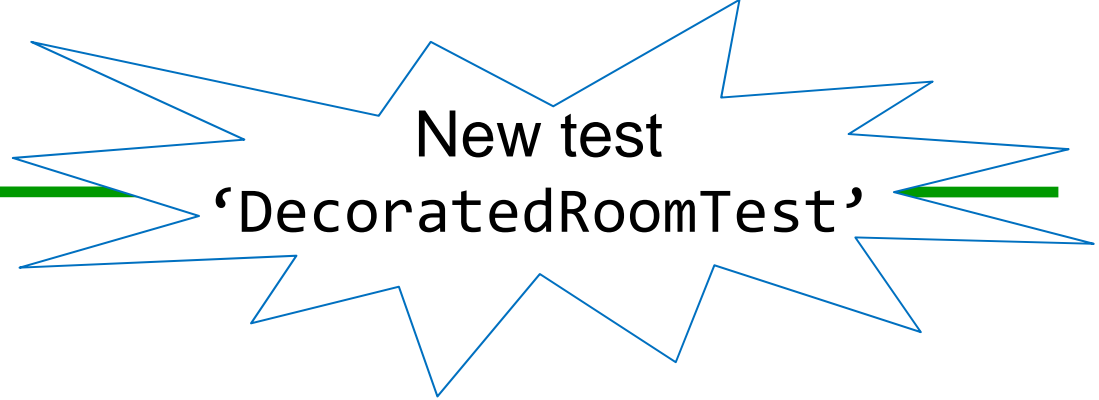
Do you want to add comments? (Configure templates and default value [here](#))

Generate comments

Class under test:

Ensure

Decorator –



```
@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

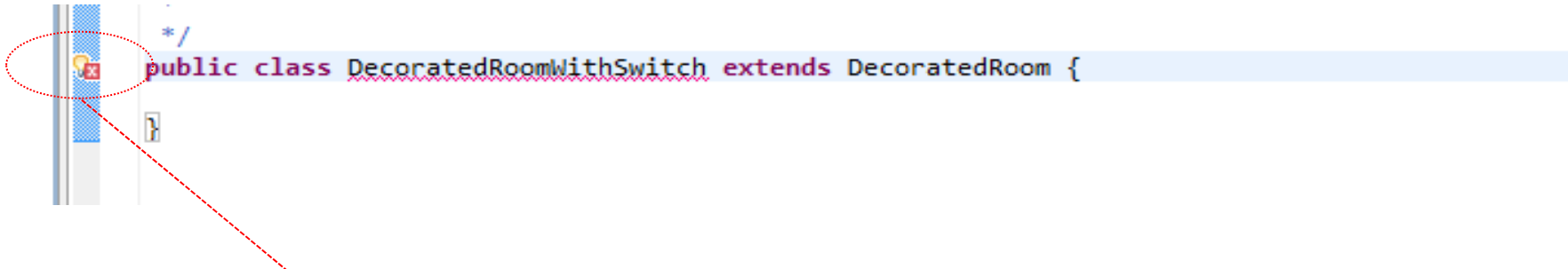
3.Ensure

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



```
public class DecoratedRoomWithSwitch extends DecoratedRoom {
```

The screenshot shows a code editor with a lightbulb icon (representing an IDE suggestion) next to the code. A red dashed line connects the lightbulb icon to the first step in the list below.

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