

CSC7203: Advanced OO

J Paul Gibson, D311

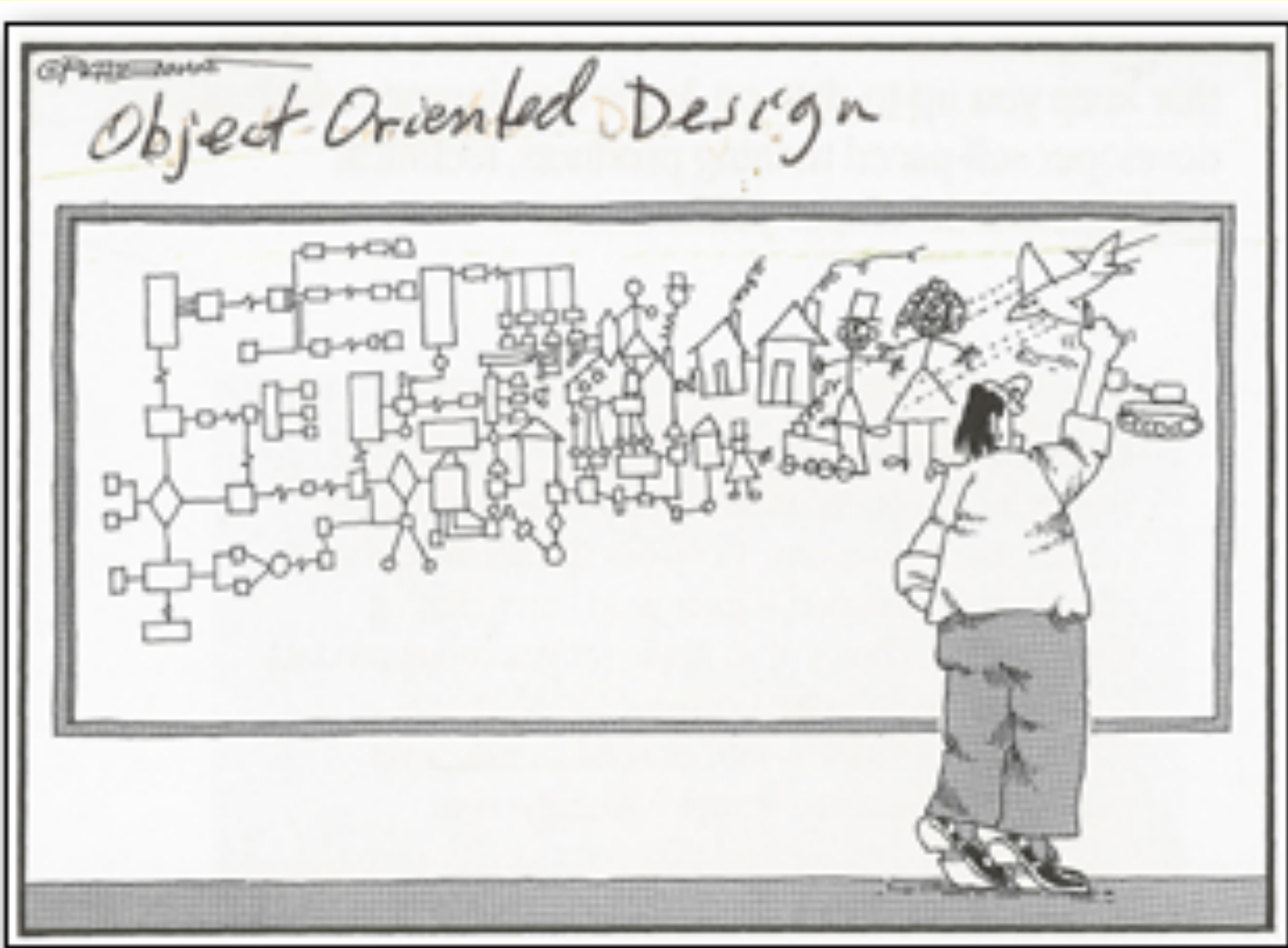
`paul.gibson@telecom-sudparis.eu`

`http://www-public.tem-tsp.eu/~gibson/Teaching/CSC7203/`

Introduction - Domino Revisited

`/~gibson/Teaching/CSC7203/CSC7203-AdvancedOO-L1.pdf`

Objects: from real world to code?

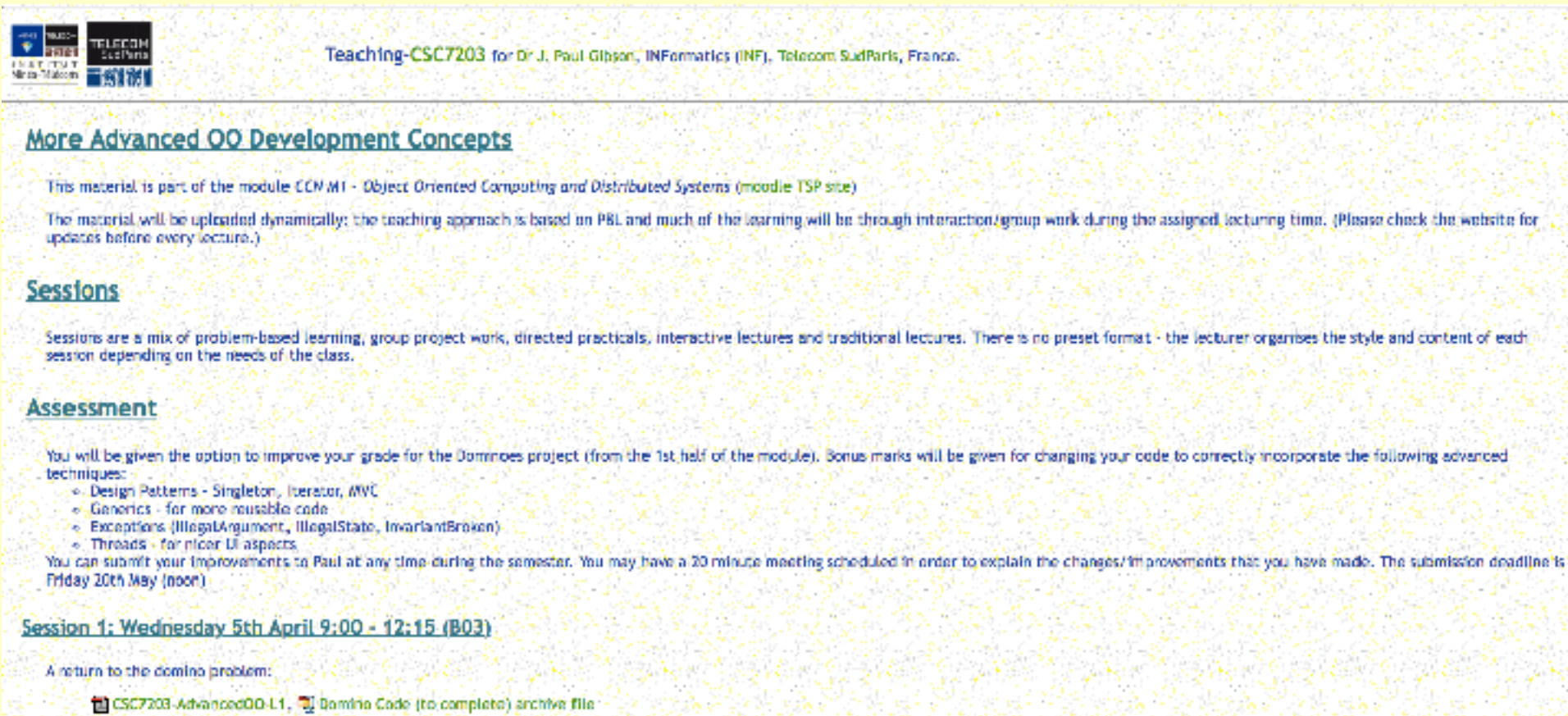


From module description – more advanced OO topics

- Design (formal versus informal) and **patterns**
- **Testing** OO systems with **JUnit**
- **Documentation with Javadocs**
- **Reuse and generics**
- **Reflection**
- **Exceptions**
- **Threads and Events**

Web Site: continually updated

<http://www-public.tem-tsp.eu/~gibson/Teaching/CSC7203/>



The screenshot shows a web page for the course CSC7203. At the top left, there are logos for INRIA, TELECOM SUD PARIS, and INFORMATIQUE. The main header text reads "Teaching-CSC7203 for Dr J. Paul Gibson, INFORMATIQUE (INF), Telecom SudParis, France." Below this is a section titled "More Advanced OO Development Concepts" with a sub-header "More Advanced OO Development Concepts". The text explains that the material is part of the module CCM.M1 - Object Oriented Computing and Distributed Systems (moodle TSP site) and will be updated dynamically. A section titled "Sessions" describes the mix of learning activities. An "Assessment" section details the Dominoes project and lists advanced techniques like Design Patterns, Generics, Exceptions, and Threads. A "Session 1" section is also visible.

Teaching-CSC7203 for Dr J. Paul Gibson, INFORMATIQUE (INF), Telecom SudParis, France.

More Advanced OO Development Concepts

This material is part of the module CCM.M1 - Object Oriented Computing and Distributed Systems (moodle TSP site)

The material will be updated dynamically; the teaching approach is based on PBL and much of the learning will be through interaction/group work during the assigned lecturing time. (Please check the website for updates before every lecture.)

Sessions

Sessions are a mix of problem-based learning, group project work, directed practicals, interactive lectures and traditional lectures. There is no preset format - the lecturer organises the style and content of each session depending on the needs of the class.

Assessment

You will be given the option to improve your grade for the Dominoes project (from the 1st half of the module). Bonus marks will be given for changing your code to correctly incorporate the following advanced techniques:

- Design Patterns - Singleton, Iterator, MVC
- Generics - for more reusable code
- Exceptions (IllegalArgument, IllegalState, InvariantBroken)
- Threads - for nicer UI aspects

You can submit your Improvements to Paul at any time during the semester. You may have a 20 minute meeting scheduled in order to explain the changes/improvements that you have made. The submission deadline is Friday 20th May (noon)

Session 1: Wednesday 5th April 9:00 - 12:15 (B03)

A return to the domino problem:

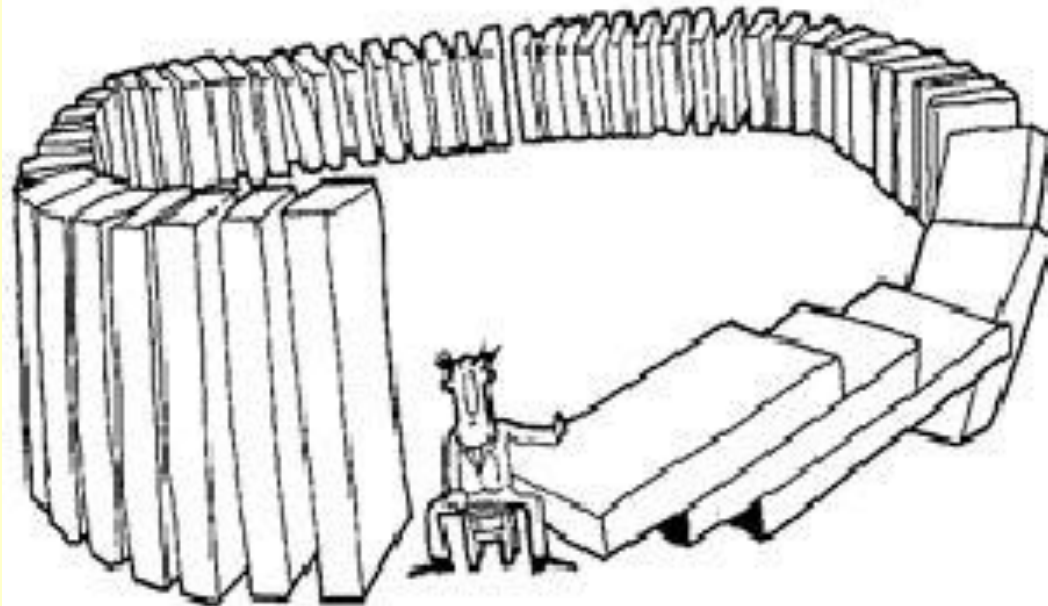
[CSC7203-AdvancedOO-L1](#), [Domino Code \(to complete\) archive file](#)



Dominoes not Domino's



**In complex systems, cause and effect
are often distant in time and space**



Review of the Dominoes Case Study

CONTEXT :

Simulation of a domino game between the computer and a user.

THE GAME.

The domino set contains **28** dominos.

A round begins by distributing 6 dominos to the 2 players.

The one who starts is the one who has a double six. If no one has a double six, the double five is looked for, then the double four etc. In the absence of a double, the human user will start.

When it is his/her/its turn to play:

- the player puts down a domino matching with one of the extremities of the dominos that are already on the "table".
- if the player does not have any matching domino, he/she/it takes from the stock a domino until he/she/it gets a valid one or that the stock is empty. In that latter case, the player passes and it is the other player's turn.

A round stops when one of the players does not have any domino left or that no player can either puts a domino down or take one from the stock.

The dominos are the following:

(6,6),(6,5),(6,4),(6,3),(6,2),(6,1),(6,0),
(5,5),(5,4),(5,3),(5,2),(5,1),(5,0),
(4,4),(4,3),(4,2),(4,1),(4,0),
(3,3),(3,2),(3,1),(3,0),
(2,2),(2,1),(2,0),
(1,1),(1,0),
(0,0)

Analysis of the Dominoes Case Study

1) Preparing for future extensions/variations and re-usability

Can the domino set have more/less elements?

Do we always start with 6 dominoes for each player?

Are there always 2 players?

Is the starting rule always the same?

Do the dominoes always have numbers on their faces?

Can we simulate different types of computer AI? ...

2) Clarification

Can a player take from the stock even when they are able to play from hand?

Do the players have to keep taking from the stock when they cannot play?

Can a player look in the stock when they take a domino?

Can a player see the stock elements?

Can a player see how many elements are in stock?

Can a player see the other players' dominoes (or how many they have)? ...

Analysis/Discussion of the Dominoes Design Decisions

Each team can present their designs to the other teams

What different design decisions were made?

Which decisions do we think are ‘good’ and why?

Which decisions do we think are ‘bad’ and why?

What makes a good design? (Criteria to be used?)

Do we see any good decisions that were evident in different designs (that could be candidate ‘design patterns’)?

(We can ask the same sort of questions with respect to the code ... leading to ‘implementation patterns’)

Some Software Engineering Tips/Good Habits

Should specify the required behaviour in a Java **interface**, with good **documentation** and **tests**

Good habit to use (**static**) constants (**final**) where appropriate

Good habit to encapsulate state (using **private**)

Good habit to implement methods for:

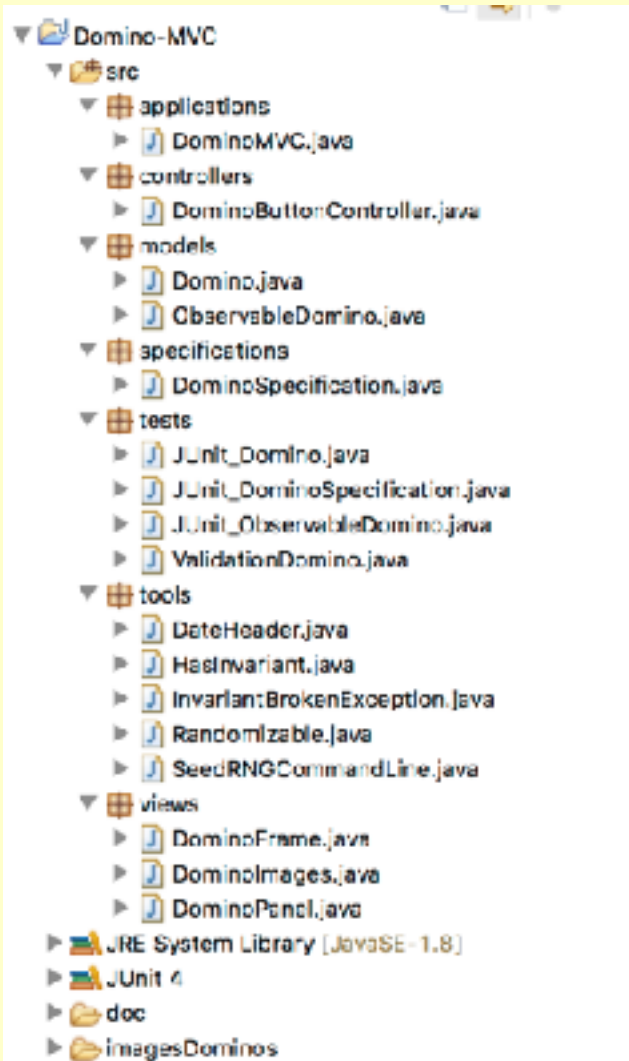
invariant

toString

equals (and **hashCode**)

randomize

A Domino Model View Controller System



Download the Domino MVC from the website :

DominoMVC.zip

Let's quickly check the *quality of the design/code*

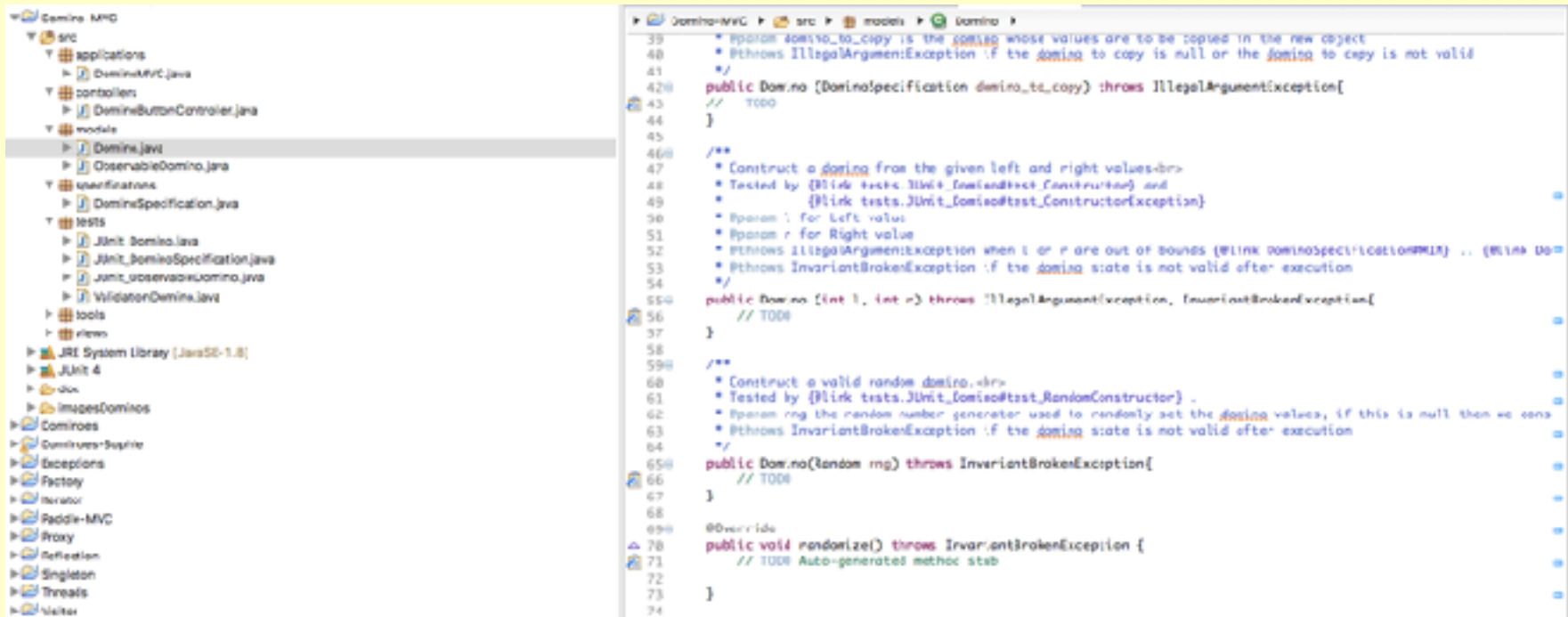
You are to:

1. read the Domino specification
2. look at the Domino JUnit tests
3. implement the Domino class methods
4. run the unit tests
5. run the validation tests
6. if all tests pass, then run the application

The only file you should edit is ***models.Domino.java***

A Domino Model View Controller System

The only file you should edit is *models.Domino.java*



The screenshot shows an IDE with a project explorer on the left and a code editor on the right. The project explorer shows a package structure for 'Domino MVC' with sub-packages like 'applications', 'controller', 'module', 'specifications', 'tests', and 'tools'. The 'module' package contains 'Domino.java'. The code editor shows the 'Domino.java' file with several TODO comments and code blocks. The code includes a constructor for 'Domino' that takes a 'DominoSpecification' and throws an 'IllegalArgumentException' if the specification is null or invalid. It also includes a constructor for 'Domino' that takes two integers and throws an 'IllegalArgumentException' and an 'InvariantBrokenException' if the integers are out of bounds or the state is invalid. Finally, it includes a constructor for 'Domino' that takes a 'Random' object and throws an 'InvariantBrokenException' if the state is invalid. The code also includes a 'randomize()' method that throws an 'InvariantBrokenException' if the state is invalid.

```
39  * @param domino_to_copy is the domino whose values are to be copied in the new object
40  * @throws IllegalArgumentException if the domino to copy is null or the domino to copy is not valid
41  */
42  public Domino (DominoSpecification domino_to_copy) throws IllegalArgumentException{
43  // TODO
44  }
45
46  /**
47  * Construct a domino from the given left and right values.-br>
48  * Tested by {link tests.JUnit_Domino@test_Constructor} and
49  * {link tests.JUnit_Domino@test_ConstructorException}
50  * @param l for Left value
51  * @param r for Right value
52  * @throws IllegalArgumentException when l or r are out of bounds {link DominoSpecification@MIN} .. {link DominoSpecification@MAX}
53  * @throws InvariantBrokenException if the domino state is not valid after execution
54  */
55  public Domino (int l, int r) throws IllegalArgumentException, InvariantBrokenException{
56  // TODO
57  }
58
59  /**
60  * Construct a valid random domino.-br>
61  * Tested by {link tests.JUnit_Domino@test_RandomConstructor} .
62  * @param rng the random number generator used to randomly set the domino values, if this is null then we cons-
63  * @throws InvariantBrokenException if the domino state is not valid after execution
64  */
65  public Domino(Random rng) throws InvariantBrokenException{
66  // TODO
67  }
68
69  @Override
70  public void randomize() throws InvariantBrokenException {
71  // TODO Auto-generated method stub
72  }
73
74
```

Check out the **TODOs**

A Domino Model View Controller System

Finished after 0.073 seconds

Runs: 14/14 Errors: 0 Failures: 14

- tests.JUnit_Domino [Runner: JUnit 4] (0.002 s)
 - test_CopyConstructorKO (0.000 s)
 - test_CopyConstructorOK (0.000 s)
 - test_DefaultConstructor (0.000 s)
 - test_RandomConstructor (0.000 s)
 - test_Constructor (0.000 s)
 - test_ConstructorException (0.000 s)
 - test_switchSides (0.000 s)
 - test_getLeft (0.000 s)
 - test_copy (0.000 s)
 - test_hashCode (0.000 s)
 - test_equals (0.000 s)
 - test_getRight (0.000 s)
 - test_toString (0.000 s)
 - test_setup (0.000 s)

Failure Trace

java.lang.AssertionError: Expected exception: java.lang.IllegalArgumentException

Unit tests



Package Explorer JUnit

Finished after 0.029 seconds

Runs: 14/14 Errors: 0 Failures: 0

- tests.JUnit_Domino [Runner: JUnit 4] (0.000 s)
 - test_CopyConstructorKO (0.000 s)
 - test_CopyConstructorOK (0.000 s)
 - test_DefaultConstructor (0.000 s)
 - test_RandomConstructor (0.000 s)
 - test_Constructor (0.000 s)
 - test_ConstructorException (0.000 s)
 - test_switchSides (0.000 s)
 - test_getLeft (0.000 s)
 - test_copy (0.000 s)
 - test_hashCode (0.000 s)
 - test_equals (0.000 s)
 - test_getRight (0.000 s)
 - test_toString (0.000 s)
 - test_setup (0.000 s)

A Domino Model View Controller System

The seed used for the random number generator in the test is 0.
You can override this value by passing an integer value as a main argument parameter, if you so wish.

```
*****  
Execution Date/Time 2016/03/16 18:38:31  
*****
```

```
Constructing all possible dominoes  
0 : 0, hashCode = 9  
...  
6 : 6, hashCode = 81
```

```
Randomly creating a domino  
5 : 2  
left = 5  
right = 2  
hash = 40
```

```
Switching sides  
2 : 5  
left = 2  
right = 5  
hash = 40
```

```
Making a copy  
2 : 5
```

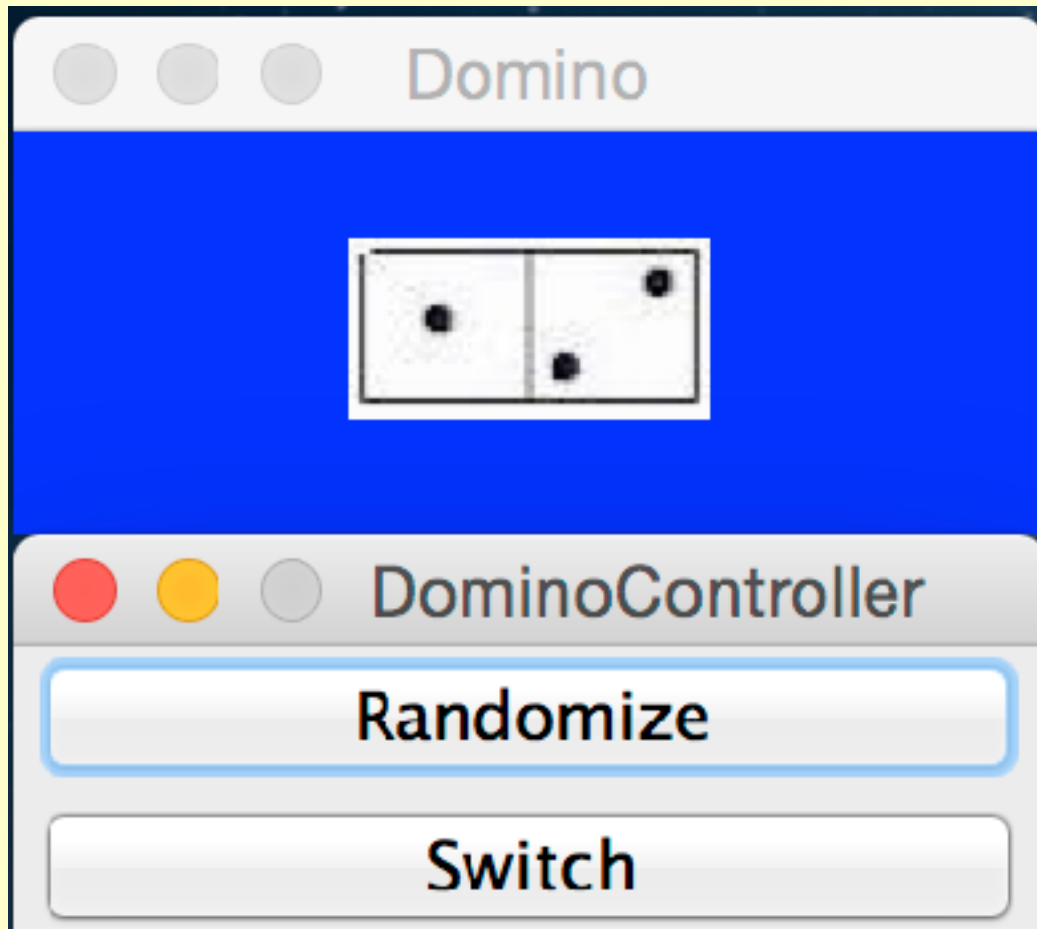
```
Checking exceptions for constructor  
Properly caught exceptionjava.lang.IllegalArgumentException: left value of -1 is smaller than MIN = 0  
Properly caught exceptionjava.lang.IllegalArgumentException: right value of -1 is smaller than MIN = 0  
Properly caught exceptionjava.lang.IllegalArgumentException: left value of 7 is bigger than MAX = 6  
Properly caught exceptionjava.lang.IllegalArgumentException: right value of 7 is bigger than MAX = 6
```

```
Looping until 2 randomly created dominoes are the same:  
4 : 2 --- 4 : 0  
...  
4 : 3 --- 3 : 4
```

```
Checking that the random construction appears 'reasonably' random so that:  
the frequency of each value chosen at random should be approx. equal to 100  
left [0] = 101  
right [0] = 95  
...  
right [5] = 89  
left [6] = 114  
right [6] = 103
```

Validation tests ... shortened

A Domino Model View Controller System



QUESTION:
What do you notice about the behaviour of the graphical application?

TODO: Draw the class diagram for the Domino-MVC code