CSC7203: Advanced OO

J Paul Gibson, A207

paul.gibson@telecom-sudparis.eu

http://www-public.telecom-sudparis.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
- Reuse and generics
- Reflection
- Threads and Events
- Comparing different OO programming languages
More Advanced OO Development Concepts

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

The material will be uploaded 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.

Session 1: Tue 17 Mar 13:45 - 17:00 (B04)
A return to the domino problem:

- CSC7203-AdvancedOO-L1
- Domino Code (to complete) archive file

Session 2: Wed 18 Mar 9:00 - 12:15 (B06)
Session 3: Thur 19 Mar 9:00 - 12:15 (B111)
Session 4: Mon 23 Mar 13:45 - 17:00 (B03)
Session 5: Wed 25 Mar 9:00 - 12:15 (B06)
Session 6: Thur 26 Mar 9:00 - 12:15 (B02)

URL: http://www-public.telecom-sudparis.eu/~gibson/Teaching/CSC7203/
Last Revision: 17th March 2015
Contact: paul.gibson@telecom-sudparis.eu
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 for formal modelling (using class invariants)
   Can a player take from the stock even when they are able to play from hand?
   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)?
Some Software Engineering Tips/Good Habits

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

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*

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

Unit tests
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 2015/03/16 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 exception java.lang.IllegalArgumentException: left value of -1 is smaller than MIN = 0
Properly caught exception java.lang.IllegalArgumentException: right value of -1 is smaller than MIN = 0
Properly caught exception java.lang.IllegalArgumentException: left value of 7 is bigger than MAX = 6
Properly caught exception java.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?