Test Case Selection

December 11, 2015

Kurt E. Clothier
Lovedeep Gondara
Kyle Kampfen
[Group 5]

CSC 578 B — Software Engineering
University of Illinois Springfield
Instructor: West, Roger

PlayingCard related Tests

Tested Code:

```
public PlayingCardFace createFace(final String face) throws IllegalArgumentException {}
```

Test Case Selection Method		Reasoning		
Black-Box, Legal Inputs		Playing Card attributes should not be null, but are a valid String		
Inputs	**Rationale**	**Expected Output**	**Observed Output**	**Pass/Fail**
null	null value	IllegalArguementException	IllegalArguementException	Pass
“2”	valid input	none, successful object creation	none	Pass
2	not a string	Complier Error	Compiler Error	Pass

Tested Code:

```
public PlayingCard createPlayingCard(final PlayingCardFace face,
                                     final PlayingCardGroup group)
                                     throws IllegalArgumentException {}
```

Test Case Selection Method		Reasoning		
Black-Box, Legal Inputs		Playing Card cannot be created with all null attributes		
Inputs	**Rationale**	**Expected Output**	**Observed Output**	**Pass/Fail**
null, null	null	Exception	Exception	Pass
createFace(“King”), null	null group	none, successful object creation	none	Pass
null, createGroup(“Red”)	null face	none, successful object creation	none	Pass
createFace(“King”), createGroup(“Red”)	non null	none, successful object creation	none	Pass

Tested Code:

```
        /**
         * Compares the specified object with this <tt>PlayingCard</tt> for equality.
         * Returns <tt>true</tt> if the given object is non-null and is this <tt>PlayingCard</tt>.
         * The copmareTo() method should be used for value comparisons.
         */
        @Override public boolean equals(final Object that) {}
```

Test Case Selection Method	Reasoning			
Black-Box, Equivalence Partitioning	Playing Card should equal another playing card, and only if they have the same (equal) attributes. Test performed on: PlayingCard(Queen, Hearts)			
Inputs	**Rationale**	**Expected Output**	**Observed Output**	**Pass/Fail**
PlayingCard(Queen, Hearts)	Same Card	true	true	Pass
PlayingCard(Queen, null)	Same Face w/null	false	false	Pass
PlayingCard(null, Hearts)	Same Group w/null	false	false	Pass
PlayingCard(Queen, Clubs)	Same Face	false	false	Pass
PlayingCard(King, Hearts)	Same Group	false	false	Pass
null	null	false	false	Pass
PlayingCardFace(“Queen”)	not a card	false	false	Pass

Tested Code:

```
/** Compare this <tt>PlayingCard</tt> to that <tt>PlayingCard</tt>, lexicographically.
  * Ex: 3 of Clubs = 3 of clubs < 4 of clubs < 4 of Hearts < Two of Diamonds
  * @return N, where N = {-n,0,n if this <,==,> that}
  */
   @Override public int compareTo(final PlayingCard that) {}
```

Test Case Selection Method	Reasoning			
Black-Box, Equivalence Partitioning	Playing Card is compared lexicographically to other Playing Cards. Test cards that should be <, =, & > the test card: PlayingCard(King, Clubs)			
Inputs	**Rationale**	**Expected Output**	**Observed Output**	**Pass/Fail**
PlayingCard(King, Clubs)	same card	0 (=)	0	Pass
PlayingCard(Queen, Clubs)	> face, same group,	- value (<)	-6	Pass
PlayingCard(King, Hearts)	same face, > group	- value (<)	-5	Pass
PlayingCard(Queen, Hearts)	> face, > group	- value (<)	-6	Pass
null	null check	+ value (>)	1	Pass
PlayingCard(null, Clubs)	null face, same group	+ value (>)	1	Pass
PlayingCard(King, null)	same face, null group	+ value (>)	1	Pass
PlayingCard(null, Hearts)	null face, > group	- value (<)	-6	Pass
PlayingCard(Queen, null)	> face, null group	+ value (>)	1	Pass

Deck related Tests

Tested Code:

```
/** Returns a number of Playing Cards from the top of this deck
 * Returns an empty array if all cards have been dealt.
 * If the specified number is larger than the number of remaining cards,
 * only those remaining will be dealt.
 */
public PlayingCard[] deal(final int number) {}
```

Test Case Selection Method	Reasoning			
Black-Box, Partitioning	Deck must correctly deal a number of playing cards, if any are left (52 total).			
Inputs	**Rationale**	**Expected Output**	**Observed Output**	**Pass/Fail**
-1	negative input	PlayingCard[0]	PlayingCard[0]	Pass
0	zero input	PlayingCard[0]	PlayingCard[0]	Pass
1	positive input	PlayingCard[1]	PlayingCard[1]	Pass
30	> 1	PlayingCard[30]	PlayingCard[30]	Pass
30	> remaining cards	PlayingCard[21] (52-1-30 = 21)	PlayingCard[21]	Pass

Tested Code:

```
/** Compares the specified object with this <tt>Deck</tt> for equality.
 * Returns <tt>true</tt> if the given object is non-null and is a <tt>Deck</tt>
 * containing the same <tt>PlayingCards</tt> as this deck.
 */
@Override public boolean equals(final Object that) {}
```

Test Case Selection Method	Reasoning			
Black-Box, Dynamic Testing	A cloned deck should not be equal after shuffling.			
Inputs	**Rationale**	**Expected Output**	**Observed Output**	**Pass/Fail**
deck.equals(deck2);	Identical Deck	true	true	Pass
deck.shuffle; deck.equals(deck2)	Identical Deck that has been shuffled	false	false	Pass

Plugin related Tests
Tested Code:

```
public String checkParamsFor(final PluginKeyword keyword) throws PluginException {}
```

Test Case Selection Method	**Reasoning**			
Black-Box, Dynamic Testing	Testing a Plugin file, created using a text file containing with only the text “nam” present. Keywords should not be found until after they are added.			
Inputs	**Rationale**	**Expected Output**	**Observed Output**	**Pass/Fail**
PluginKeyword.NAME	Keyword not present	PluginException	PluginException	Pass
update plugin to “name”, PluginKeyword.NAME	Keyword present, but without parameter	PluginException (no parameters)	PluginException	Pass
update plugin to “name test3”, PluginKeyword.NAME	Keyword and parameter are present	“test3”	“test3”	Pass

Tested Code:

```
public String[] checkCSVParamsFor(final PluginKeyword keyword) throws PluginException {}
```

Test Case Selection Method	**Reasoning**			
Black-Box, Dynamic Testing	Some keywords have comma separated value parameters. Here, we test the effectiveness of retrieving them, again with only the text “nam” present.			
Inputs	**Rationale**	**Expected Output**	**Observed Output**	**Pass/Fail**
PluginKeyword.NAME	Keyword not present	PluginException	PluginException	Pass
update plugin to “name”, PluginKeyword.NAME	Keyword present, but without parameters	PluginException (no parameters)	PluginException	Pass
update plugin to “name test3”, PluginKeyword.NAME	Keyword and single parameter are present	{“test3”}	{“test3”}	Pass
update plugin to “name test1, test2, test3”, PluginKeyword.NAME	Keyword and CSV parameters are present	{“test1”, “test2”, “test3”}	{“test1”, “test2”, “test3”}	Pass

Tested Code:

```
        public int checkIndexOf(final PluginKeyword keyword) throws PluginException {}

        /** Returns the numeric parameter found after the the keyword on the specified line.
         * Parameter must be a whole number.
         */
        public int checkNumericParams(final PluginKeyword keyword, final int lineNdx)
                                      throws PluginException, IndexOutOfBoundsException {}
```

Test Case Selection Method	**Reasoning**			
Black-Box, Dynamic Integration testing	Come keywords use numeric parameters. Here, we use checkIndexOf() to locate the index of a keyword; checkNumericParameter gets the parameter.			
Inputs	**Rationale**	**Expected Output**	**Observed Output**	**Pass/Fail**
PluginKeyword.NAME	Keyword not present	PluginException	PluginException	Pass
update plugin to “name”	Missing parameters	PluginException	PluginException	Pass
update plugin to “name 2”	single numeric param	2	2	Pass
update plugin to “name -2”	negative parameter	-2	-2	Pass
update plugin to “name 2.0”	decimal point	PluginException	PluginException	Pass

GUI related Tests

Tested Code:

```
{
    offscreen = new BufferedImage(450, 550, BufferedImage.TYPE_3BYTE_BGR);
}
```

Test Case Selection Method	Rationale	Inputs	Expected Output	Observed Output	Pass/Fail
Dynamic testing	Negative image size, x axis	-1,550	Exception	Exception	Pass
Dynamic testing	Negative image size, y axis	450,-1	Exception	Exception	Pass
Dynamic testing	Negative image size, both axis	-1,-1	Exception	Exception	Pass
Dynamic testing	Limit image buffer	450,5500	Exception	Run	Fail

Tested Code:

```
{
        msg = new JTextArea("Welcome to the card game\n", 4, 20);
}
```

Test Case Selection Method	Rationale	Inputs	Expected Output	Observed Output	Pass/Fail
Dynamic testing	Negative text area, rows	-1,20	Exception	Exception	Pass
Dynamic Testing	Negative text area, columns	4,-1	Exception	Exception	Pass
Dynamic testing	Negative text area, rows and columns	-1,-1	Exception	Exception	Pass
Dynamic testing	Column size non proportional to game screen	4, 60	Exception	Run	Fail