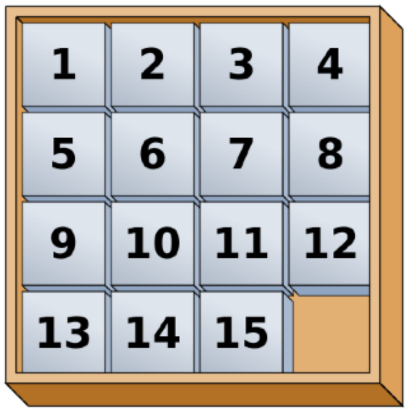
CS 301 Fall 2017 **Project Assignment 1** Friday, September 1, 2017

To Deliver: Release 1.0 in your SVN repository under a subdirectory tags

To Download: P1.zip from blackboard site
Hard & final submission deadline: Monday Sep 18, at midnight, 11.59 pm

Objectives:
- Programming in Java: To implement classes for a given interface
- SW Development tools: To learn unit testing with Junit, working with a version control system
- SW Development: To get experience with writing comments before coding

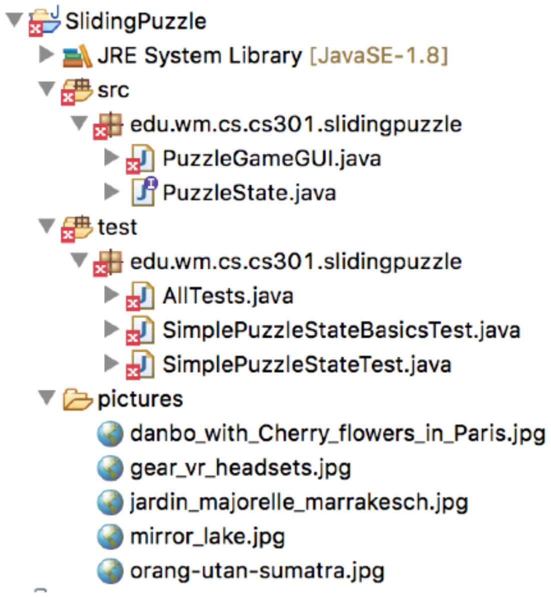
Context: You are a freelance software developer hired by a company to empower their development team on a running project. Your job is to implement some missing code for a Sliding Puzzle game. The head designer determined the overall design, fixed interfaces and charges you to fill in the blanks to make the overall product work. You get paid for an overall working solution (not by the hour). Your code is evaluated with the help of a given set of test cases known to you.



Case study: Sliding Puzzle:

You will need to implement at least 1 class.
1. SimplePuzzleState: a representation of the state of a puzzle
You can add other classes as you deem necessary but you are not allowed to change the given interfaces. See also http://en.wikipedia.org/wiki/Fifteen_puzzle for further information on the Sliding Puzzle problem itself.

Task list:
1) **Prerequisites:**
    1) Create new Project "SlidingPuzzle" in Eclipse
    2) Create a 2nd source folder named "test"
    3) Create a new package "edu.wm.cs.cs301.slidingpuzzle" in both folders (src, test)
    4) Extract files from P1.zip. and import the files with suffix .java into the Eclipse project under the src folder into the package edu.wm.cs.cs301.slidingpuzzle.
    5) Move files AllTests.java, SimplePuzzleStateTest.java and SimplePuzzleStateBasicsTest.java from src to test into package edu.wm.cs.cs301.slidingpuzzle.
    6) Create a 3rd folder (a plain folder not a source folder) named "pictures" and add the files with suffix .jpg to the pictures folder
    7) The resulting directory structure looks like the one in the figure to right. The little red boxes with white crosses indicate errors.
    8) Check for error messages in "Problem" window, you



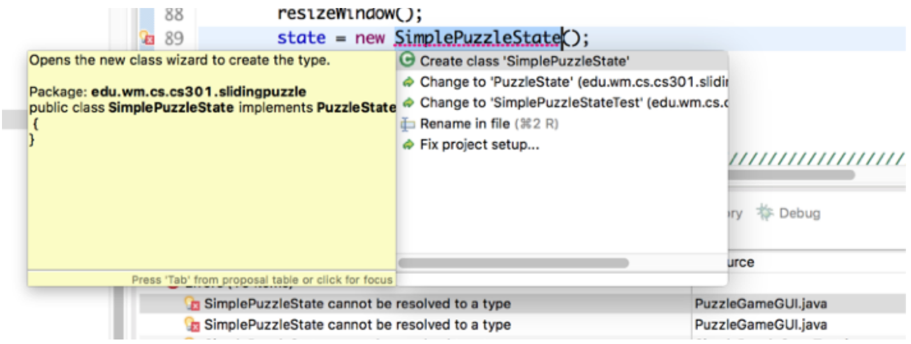
```
SlidingPuzzle
  JRE System Library [JavaSE-1.8]
  src
    edu.wm.cs.cs301.slidingpuzzle
      PuzzleGameGUI.java
      PuzzleState.java
  test
    edu.wm.cs.cs301.slidingpuzzle
      AllTests.java
      SimplePuzzleStateBasicsTest.java
      SimplePuzzleStateTest.java
  pictures
    danbo_with_Cherry_flowers_in_Paris.jpg
    gear_vr_headsets.jpg
    jardin_majorelle_marrakesch.jpg
    mirror_lake.jpg
    orang-utan-sumatra.jpg
```

may need to add Junit to the build path of the project

1) Hint: Package Explorer Window, right click SlidingPuzzle->Build Path->Configure Build Path-> add Library: select Junit4 to add

9) Check for remaining error messages in “Problem” window: click on an error message to get to the line of code it refers to. In the figure below this leads for example to line 105 where “SimplePuzzleState()” is underlined in red. If you click on the little error marker next to the line number 105. The popup message you see in the figure comes up with suggestions on how to fix the problem. We need to create the missing class “SimplePuzzleState.java”. Pick the corresponding choice form the “quick fix” pop up menu, which will create a dummy class for you that at least makes the compiler happy (you can create that class with its necessary methods manually but that is more work).

    1) Hint: Eclipse provides you with quick fix suggestions, use them to produce a dummy default implementation for the SimplePuzzleState class.



```
        resizeWindow();
        state = new SimplePuzzleState();
```

Opens the new class wizard to create the type.

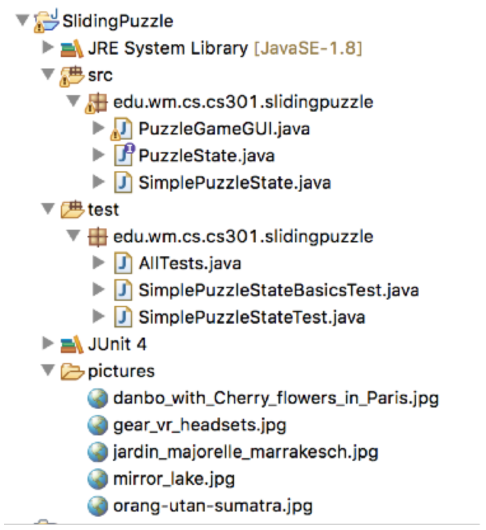
```
Package: edu.wm.cs.cs301.slidingpuzzle
public class SimplePuzzleState implements PuzzleState
{
}
```

Press 'Tab' from proposal table or click for focus

Create class 'SimplePuzzleState'
Change to 'PuzzleState' (edu.wm.cs.cs301.slidi
Change to 'SimplePuzzleStateTest' (edu.wm.cs.
Rename in file (⌘2 R)
Fix project setup...
ry
Debug
urce

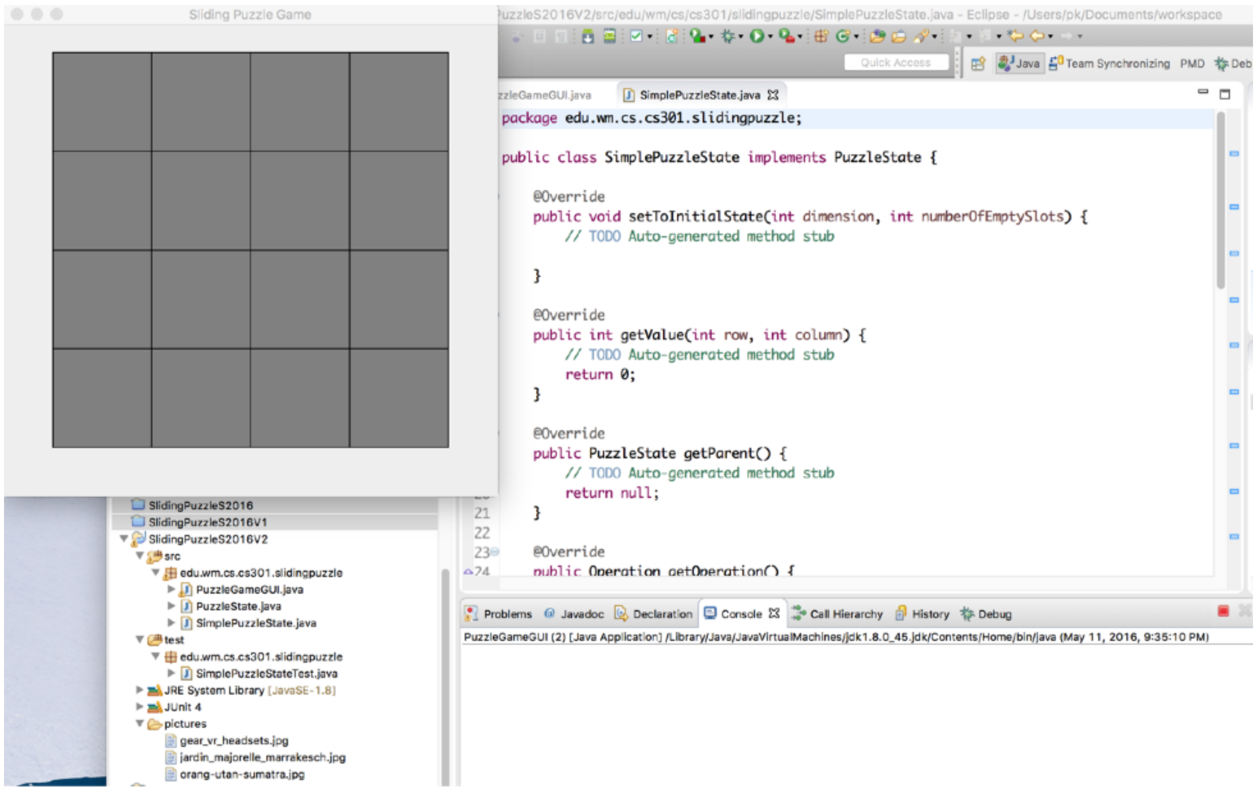
```
SimplePuzzleState cannot be resolved to a type    PuzzleGameGUI.java
SimplePuzzleState cannot be resolved to a type    PuzzleGameGUI.java
```

10) The problem window should be empty after fixing the errors (but for some Warnings maybe) and the Package Explorer window should show this directory structure for your SlidingPuzzle project.



```
▼ SlidingPuzzle
  ▶ JRE System Library [JavaSE-1.8]
  ▼ src
    ▼ edu.wm.cs.cs301.slidingpuzzle
      ▶ PuzzleGameGUI.java
      ▶ PuzzleState.java
      ▶ SimplePuzzleState.java
  ▼ test
    ▼ edu.wm.cs.cs301.slidingpuzzle
      ▶ AllTests.java
      ▶ SimplePuzzleStateBasicsTest.java
      ▶ SimplePuzzleStateTest.java
  ▶ JUnit 4
  ▼ pictures
      danbo_with_Cherry_flowers_in_Paris.jpg
      gear_vr_headsets.jpg
      jardin_majorelle_marrakesch.jpg
      mirror_lake.jpg
      orang-utan-sumatra.jpg
```

11) Run the project in Eclipse. If it works, you’ll see a window with 1 grey square and a grid. If you see this, congratulations! You are good to go to start programming for Project 1. The figure below shows this plus the generated code for SimplePuzzleState.java.

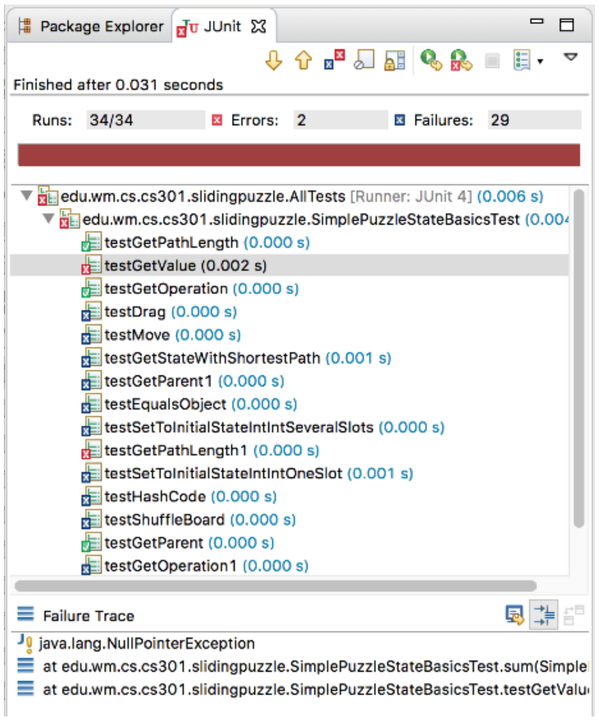


```
package edu.wm.cs.cs301.slidingpuzzle;

public class SimplePuzzleState implements PuzzleState {

    @Override
    public void setToInitialState(int dimension, int numberOfEmptySlots) {
        // TODO Auto-generated method stub

    }

    @Override
    public int getValue(int row, int column) {
        // TODO Auto-generated method stub
        return 0;
    }

    @Override
    public PuzzleState getParent() {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public Operation getOperation() {
```

```
SlidingPuzzleS2016
SlidingPuzzleS2016V1
SlidingPuzzleS2016V2
  src
    edu.wm.cs.cs301.slidingpuzzle
      PuzzleGameGUI.java
      PuzzleState.java
      SimplePuzzleState.java
  test
    edu.wm.cs.cs301.slidingpuzzle
      SimplePuzzleStateTest.java
  JRE System Library [JavaSE-1.8]
  JUnit 4
  pictures
    gear_vr_headsets.jpg
    jardin_majorelle_marrakesch.jpg
    orang-utan-sumatra.jpg
```

```
PuzzleGameGUI (2) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_45.jdk/Contents/Home/bin/java (May 11, 2016, 9:35:10 PM)
```

12) The SimplePuzzleStateBasicsTest.java, and SimplePuzzleStateTest.java classes are Junit test. Run AllTests.java as a Junit test case to execute all test cases. The current default implementation for the SimplePuzzleState fails miserably as you can see from the figure to the right. Your Junit test results should look the same at this stage. (When you hand in your solution, the top bar should be all green and your code should pass all tests.)

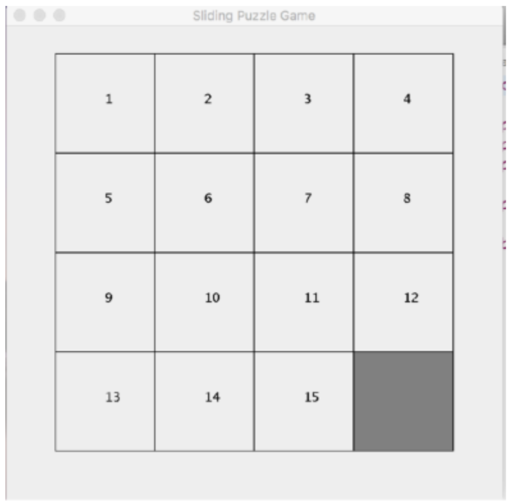


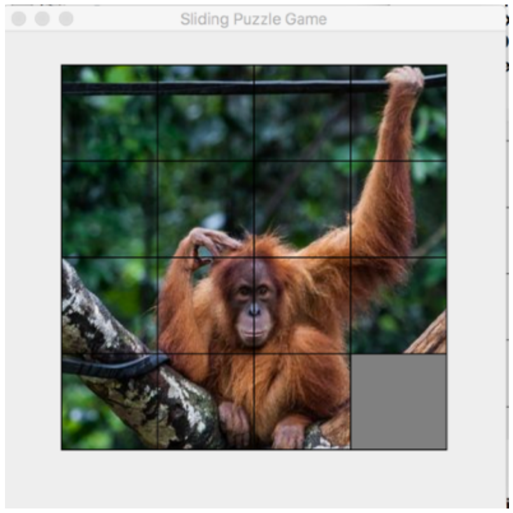
```
Finished after 0.031 seconds

Runs: 34/34      Errors: 2      Failures: 29

edu.wm.cs.cs301.slidingpuzzle.AllTests [Runner: JUnit 4] (0.006 s)
  edu.wm.cs.cs301.slidingpuzzle.SimplePuzzleStateBasicsTest (0.004
    testGetPathLength (0.000 s)
    testGetValue (0.002 s)
    testGetOperation (0.000 s)
    testDrag (0.000 s)
    testMove (0.000 s)
    testGetStateWithShortestPath (0.001 s)
    testGetParent1 (0.000 s)
    testEqualsObject (0.000 s)
    testSetToInitialStateIntIntSeveralSlots (0.000 s)
    testGetPathLength1 (0.000 s)
    testSetToInitialStateIntIntOneSlot (0.001 s)
    testHashCode (0.000 s)
    testShuffleBoard (0.000 s)
    testGetParent (0.000 s)
    testGetOperation1 (0.000 s)

Failure Trace
java.lang.NullPointerException
at edu.wm.cs.cs301.slidingpuzzle.SimplePuzzleStateBasicsTest.sum(Simple
at edu.wm.cs.cs301.slidingpuzzle.SimplePuzzleStateBasicsTest.testGetValu
```

2) **Optional**: Create a documentation in html format with javadoc for your Eclipse project for your own productivity
   1) In Eclipse: call menu-Project->Generate Javadoc, let generated files reside in directory SlidingPuzzle/doc
3) **Required:** Implement class SimplePuzzleState.java such that it implements the PuzzleState.java interface and such that the overall Puzzle Game works and your code passes junit tests in AllTests.
   1) You can NOT change the interface PuzzleState!
   2) You can have other classes that support your SimplePuzzleState as necessary.
   3) Your SimplePuzzleState classes can have additional private methods as you deem necessary.

4) How to get started?

1) Read the PuzzleState.java interface to understand what your SimplePuzzleState.java class is supposed to do. Ask yourself what information a SimplePuzzleState object needs to store such that it can provide the required functionality. Decide what data structure may be good for this.
2) Read the PuzzleGameGUI.java code to learn more about Java programming and to see how the SimplePuzzleState objects are used.
3) General rule: think before you code, don't start just hacking your way to a solution.
4) Begin the implementation with simple methods that are straightforward to implement to build some experience with Java. Get/set methods are prime suspects for this.
5) For non-trivial methods, start writing an outline of the individual steps for your algorithm as comments in plain English into the corresponding method in your java class files first. Once you have it all nailed down at a conceptual level (sequence of high-level steps towards as solution) and fully understand what you want to do, encode one step after another in Java. This trick does a neat separation of concerns: deciding how to solve a problem from deciding how code a solution in Java. Leave the comments as documentation to express what the code should do. This will allow for a comparison (code review) with your actual Java code to find errors as a mismatch between what code should do and what it is really doing. The devil is in the details and this process helps catching them.
6) From the two JUnit Test classes, begin with the SimplePuzzleStateBasicsTest because the other JUnit test class is more rigid and needs more methods in SimplePuzzleState to work. (It requires setToInitialState and move() methods to work in its setUp() method, which is executed before any of the actual test methods.)
7) Clarify the notion of equality for 2 SimplePuzzleStates and implement the equals() and hashcode() methods. (Warning: equality is tricky, check the Java docs!)
8) Note: PuzzleState does not work with exceptions, its methods return null if some operation can not be performed. For this assignment, we do not(!) want to deal with the extra complexity of error handling mechanisms in Java.
9) Optional but useful: Feel free to write more Junit tests in a separate test class in your test folder to check more cases.





The figures show the puzzle game as you will see it when you make some progress with your implementation. Left figure: default setting with numbers. Right figure: after loading an image. Feel free to add your own favorite images to the pictures folder.

5) **What is required?**
    1) Make your code pass the given Junit tests in AllTests, i.e., SimplePuzzleStateTest and SimplePuzzleStateBasicsTest.
    2) Make the game work such that one can run the program and play the game.
        1) For the “Use shortest path”, a simple implementation for getStateWithShortestPath() that has 1 line of code and is only correct in some cases, but it will at least make the game playable for a user.
    3) Working with the version control system SVN:
        1) We will get started with SVN during the first project as well.
        2) Once your SVN repository works, perform regular updates on your progress. The recommended granularity is to update the repository whenever you made some progress, e.g. your code passes one more test. It is common practice to only upload code that at least compiles with no errors but you can decide on this for yourself.
        3) You are required to have at least 10 updates (versions) in your SVN repository as one can see in the version history. A version is the result of a “commit” operation.
    4) Final delivery: Create a tag with a Release 1.0 before the final deadline. Do a full release including your junit test cases.

6) What is optional?
    1) The feature “Use shortest path” is optional. If you do NOT want to work on this, there is a trivial implementation for getStateWithShortestPath() that has 1 line of code which is expected anyway. The optional part of the assignment is intended for students who know Java and look for an additional challenge. Calculating the shortest path to the solution can be computed with the help of a search procedure, e.g. breadth-first-search on a game tree that results from possible moves. Note: also adjust the Junit test class by settings its “checkShortestPathCalculation” flag to true.

Grading:

You will receive points if:

1) **Your SVN Repository** works:
    1) your subversion repository is set up correctly and the grader can download your release 1.0 from it. (Note: the grader will always pick the release with the highest number, so if there is a Release 1.1 and a Release 1.0, the grader will grade Release 1.1).
    2) your project history shows at least 10 revisions (versions) of your code together with meaningful comments on what you changed from one revision to the next. Note: if you use SVN like a professional developer, you will have 10 revisions easily and as a natural by product of your implementation work. Just for clarity: at least 10 revisions, but just 1 release and the release is most likely matching with the latest revision.
2) **Your implementation** of the SimplePuzzleState passes the given test cases in SimplePuzzleStateTest and SimplePuzzleStateBasicsTest and one can also run the application and play the game.
    1) You receive points for every test that your implementation passes. The total amounts of points for passing tests is about the same as for having a working game! Do not underestimate how important it is that your code passes the tests!

   2) The menu entry “Auto Mode” works for settings of 1, 2, and 3 empty slots if one shuffles the game, performs a few extra click and drag operations to extend the series and the Auto mode is able to get back to the initial state.

3) **Bonus Points (optional):** You get bonus points if the menu entry “Use shortest path” works such that the Auto mode is able to get back to the initial state on the shortest possible path. The code needs to pass all Junit tests with the flag “checkShortestPathCalculation” being set to true AND the game works with this feature. Please leave the print-statements in the GUI java class in tact which will help the grader to recognize the calculated path lengths.