Fixed CS201 Database of questions for final exam

Here are the questions. They convey the things I will test. In general, to study for the test, you should get the code for TicTacToe and the unit tests in the jar file sent out a few nights ago and those in TicTacToeTester. You should take the TicTacToeTester files and try to build a working tic-tac-toe program (the way you did in the LabExam). This will help you understand the concepts behind the questions in the database.

A bad strategy for testing is to try to memorize the answers. I will change the question numbers, the choices (not the questions) and the order in which the choices are presented. None of these changes will matter if you create the LabExam code yourself from the tests.

You should understand the things being tested for:

a. Compile time errors
   i. does a class exist
   ii. do its constructors exist
   iii. do the methods exist
   iv. are certain tests overridden (?) like equals or toString
   v. does a method need to throw an exception
   vi. does instance data exist
   vii. is the instance data initialized correctly
   viii. is the for loop properly written
   ix. does inheritance come into play (extends vs. implements keyword)
   x. can they throw an exception properly
   xi. can they access an array properly
   xii. can they declare an array variable properly
   xiii. can they initialize an array properly
   xiv. do they understand final, protected, private, public?

b. Run-time errors
   i. does the method return the right thing
   ii. does the method throw an exception
   iii. do local variables overshadow instance data
   iv. are the for loops testing at the right bounds (like, do they go from 0 to maxrows, when they should go from 0 to maxrows-1)
   v. are they accessing an array before initializing it?
   vi. are they accessing an array out of bounds (less than 0, off by one errors)

Only circle one answer for each question. Select the BEST answer from the selection. Do not circle more than one answer for each question or points will be deducted. If you do not know an answer, leave that question blank. Points will be deducted for wrong answers.
BoardTester0.java

1. // BoardTester0.java
2. import java.io.PrintWriter;
3. import java.io.StringWriter;
4. import junit.framework.*;
5. public class BoardTester0 extends TestCase {
6. private Board tttB;
7. private final int MAXROWS = 3;
8. private final int MAXCOLS = 3;
9. protected void setUp() throws Exception {
10. }
11. public void testConstructor() {
12. tttB = new Board(MAXROWS, MAXCOLS);
13. }
14. }

For the following questions, use the code given above.

1. Choose the best answer.

   a. BoardTester0 is a subclass of TestCase
   b. BoardTester0 is a superclass of TestCase
   c. BoardTester0 implements TestCase
   d. BoardTester0 inherits from TestCase
   e. 2 or more of above

2. Assume I add the following statement to testConstructor() after line 12
   int MAXROWS = 4;

   a. This will cause a compile-time error
   b. This will cause a run-time error
   c. This will cause later uses of MAXROW to possibly be incorrect
   d. Both a and b
   e. none of the above

3. When the programmer works with the above code "the assignment statement to tttB in testConstructor() will ensure that a Board constructor exists." This will happen at:

   a. run-time
   b. debug-time
   c. thread-time
   d. compile-time
   e. neither a nor b

4. The declaration
   private Board tttB;
   in the Board class above causes what to happen:

   a. a Board object is created whenever a BoardTester0 object is created
   b. a Board reference is created whenever a BoardTester0 object is created
c. nothing, because the data is private and therefore no one can change it
d. none of the above

5. The keyword **protected** in line 9 means:
   a. all subclasses of TestCase may call the method
   b. **only subclasses of BoardTester0 may call the method**
   c. anyone may use the method, but only subclasses of BoardTester0 may change it
d. only methods inside BoardTester0 may call the method
e. none of the above

6. Which keyword in the class above makes sure that MAXROW will not be changed after the initial assignment
   a. private
   b. public
   c. protected
   d. final
   e. int

7. Choose the best answer. What does the test in testConstructor() do?
   a. makes sure tttB is not uninitialized
   b. makes sure there is a Board constructor
   c. **makes sure there is a 2-argument (int, int) Board constructor**
   d. makes sure the 2-argument (int, int) Board constructor creates an NxN board
e. none of the above

8. When does the test in testConstructor() happen:
   a. compile-time
   b. run-time
   c. on the heap
   d. on the stack
e. none of the above

9. Write the **minimum** piece of code that will allow the above test to compile and pass
   ```java
   public class Board {
       public Board(int r, int c) {
   ```

10. If no Board class exists, what is the first line that will not compile in the above class:
    a. 0
    b. 4
    c. 12
    d. 6
    e. none of the above

11. If the Board class exists, but does not include any constructors, what is the first line that will not compile:
    a. 0
    b. 4
    c. 12
    d. 6
    e. none of the above
1. // BoardTester1.java
2. import java.io.PrintWriter;
3. import java.io.StringWriter;
4. import junit.framework.*;

5. public class BoardTester1 extends TestCase {
6.     private Board tttB;
7.     private final int MAXROWS = 3;
8.     private final int MAXCOLS = 3;
9.     protected void setUp() throws Exception {
10.         tttB = new Board(MAXROWS, MAXCOLS);
11.         tttB.init();
12.     }
13.     /*
14.     a board will be constructed but not populated,
15.     we should get nulls
16.     */
17.     public void testGetRowColempty() {
18.         tttB = new Board(MAXROWS, MAXCOLS);
19.         String s = null;
20.         try {
21.             s = tttB.getSymbolAtMove(new Move(0, 0));
22.         } catch (MoveException rce) {
23.             rce.printStackTrace(System.out);
24.             assertEquals(true, false);
25.         } finally {
26.             assertEquals(null, s);
27.         }
28.     }
29. }

The following questions relate to just the code in BoardTester1.java and not the code in BoardTester0.java.

12. As written, upon how many user-written classes does BoardTester1 depend to compile (don't include BoardTester1)?
   a. 0
   b. 1
   c. 2
   d. 3 (Board, MoveException, Move)
   e. 4

13. To which class does the init() method belong?
   a. Board
   b. Move
   c. MoveException
   d. Player
   e. BoardTester1
14. Can we tell if the init() method does anything to the instance data of its class?
   a. yes
   b. no
   c. 

15. The 2-arg Board constructor populates its internal representation of a tic-tac-toe board with the following kind of data:
   a. references to String objects (not null)
   b. int
   c. char
   d. float
   e. none of the above

16. The assignment on line 21 should result in:
   a. an exception being thrown
   b. the assignment being skipped
   c. the assignment of a blank String to s
   d. the assignment of null to s
   e. none of the above

17. Should the call to getSymbolAtMove() on line 21 result in an exception being thrown?
   a. yes
   b. no
   c. cannot tell until the code runs

18. What type of exception should getSymbolAtMove() throw?
   a. Exception
   b. MoveException
   c. exception
   d. moveexception
   e. none of the above

19. If getSymbolAtMove() throws an exception that is a subclass of MoveException, will the catch-block on lines 21 and 22 be executed?
   a. yes
   b. no
   c. can't tell from the code
   d. 

20. If getSymbolAtMove() throws an exception that is not a subclass of MoveException, will the finally block (line 26) be executed?
   a. yes
   b. no
   c. can't tell from the code
21. If the compiler indicates an error on line 22 (and nowhere else), which of the following may be a cause of the error?

a. the catch statement is syntactically wrong
b. the MoveException class does not exist
c. the getSymbolAtMove() method is not declared correctly
d. two of the above
e. only one of the above

22. If the compiler indicates an error on line 22 (and nowhere else), and it indicates that the catch statement is never reached, which of the following is the best cause of the error?

a. the catch statement is syntactically wrong
b. the MoveException class does not exist
c. the getSymbolAtMove() method is not declared correctly
d. two of the above
e. only one of the above (probably the getSymbolAtMove() method is not declared correctly, i.e., it should throw a MoveException and probably does not)

23. For the exception handling code in lines 20-27, which of the following code segments, if inserted into the Board class, will help BoardTester1.java compile?

a. public String getSymbolAtMove(Move move) throws MoveException {
   int row = move.getRow();
   int col = move.getCol();
   try {
      if(goodMove(move) == false) {
         throw new MoveException();
      }
   } catch (MoveException me) {
      return tttBoard[0][0];
   }
   return tttBoard[row][col];
}

b. public String getSymbolAtMove(Move move) throws MoveException {
   int row = move.getRow();
   int col = move.getCol();
   if(goodMove(move) == false) {
      throw new MoveException();
   }
   return tttBoard[row][col];
}

c. public String getSymbolAtMove(Move move) {
   int row = move.getRow();
   int col = move.getCol();
   try {
      if(goodMove(move) == false) {
         throw new MoveException();
      }
   } catch (MoveException me) {
      return tttBoard[0][0];
   }
   return tttBoard[row][col];
}
d. all three of a, b and c
   e. only two of a, b and c (a will never throw an exception, but it will allow the compiler
to pass the test, b will throw an exception if goodMove() detects a bad move, c has
an exception being thrown, but it is caught in the method, so the method will never
throw it and the compile can tell this from the signature.)

24. Given that setUp() is called before testGetRowColempty(), is the Board object
   referred to by tttB on line 21 initialized?
   a. true
   b. false

25. Given that setUp() is called before testGetRowColempty(), but a new Board object
   is created and not initialized, is the reference variable tttB on line 21 null (true) or
   not (false)?
   a. true
   b. false

26. Suppose we successfully construct a Board object as in line 18. Given the code in
   lines 19-27, are we expecting that a method call to getSymbolAtMove will return
   references to empty Strings ("") or null reference when passed a legal Move object?
   a. empty Strings
   b. null reference
   c. can't tell from code

27. Suppose we have the getSymbolAtMove(Move m) method as tested in lines 19-27.
   Further, suppose that a call to a method goodMove(new Move(a,b)) returns true
   when neither player has a piece or symbol at location a,b on the board. Which of
   the following lines makes this test the best (assume that code above this line has
   checked to make sure the move is a legal one for the board)
   a. if(board[move.getRow()][move.getCol()] != ") return false;
   b. if(board[move.getRow()][move.getCol()] != null) return false;
   c. tmp=board[move.getRow()][move.getCol()];
      if(tmp= = ""
   d. tmp=board[move.getRow()][move.getCol()];
      if(tmp! = "" || tmp != null) return false;
   e. none of the above (a is wrong if we have a null on the board, b is wrong if we have
      an empty string on the board, c is wrong since it requires both null and empty string,
      d is wrong since it falsely catches a null on the first part and falsely catches an empty
      string on the second)

28. Which of the following statements most accurately describes
   testGetRowColempty()?
   a. the test is a good test to make sure an uninitialized board will return null references
      when each location is accessed.
   b. the test is a poor test since all we need to do is write a method which always returns
      null references.
   c. the test is a poor test since all we need to do is write a method which returns a null
      reference when Move(0,0) is passed in.
   d. the test is a poor test but at least it ensures the programmer must add the throws
      keyword to the signature of the method.
   e. two or more of the above (a is wrong, but b, c and d are correct)
29. If one wanted to improve the test coverage in testGetRowColempty() how many tests should it include?
   a. a number of cases in which the column portion of a Move is less than zero and the row is on the board.
   b. MAXROWS x MAXCOLS tests of getSymbolAtMove()
   c. a number of cases in which the row portion of a move is greater than MAXROWS and the column is on the board
   d. all of a, b and c
   e. two of a, b and c (a and b are right, c is not a good test since it misses the case where a row is equal to MAXROWS and this is an illegal row)

30. The "test" of the method init() in testInit() happens at
   a. package-time
   b. run-time
   c. compile-time (at run time, we aren't really testing anything)
   d. can't tell from the code
   e. two from a, b and c

31. Suppose I want the board, tttB, to be initialized for a series of seven (7) tests in one tester class. The methods in the class would be called testInit(), testConstructor1(), testSet(), testGet(), setUp(), testConstructor2(), testConstructor3() and testBoardToString(). The best way to make sure the board is initialized is to:
   a. put a call to init() in testInit() and make sure it is the first test run
   b. put a call to init() at the beginning of each test
   c. put a call to init() in setUp()
   d. put a call to init() in setUp() and then call each test from setUp, one at a time.
   e. use a for loop to call init() seven times.
32. Suppose I have a series of seven (7) tests in one tester class. The methods in the class would be called testInit(), testConstructor1(), testSet(), testGet(), setUp(), testConstructor2(), testConstructor3() and testBoardToString(). A reasonable set of calls to the methods by the unit tester would be:

a. setUp(), testBoardToString(), testConstructor1(), testConstructor2(), testConstructor3(), testGet(), testInit(), testSet()
b. setUp(), testInit(), testConstructor3(), testConstructor2(), testConstructor1(), testSet(), testGet(), testBoardToString()
c. none of the above  (a reasonable sequence would be setUp(), testBoard(), setUp(), testConstructor1(), setUp(), testConstructor2(), setUp(), etc.)

33. Given lines 10 and 13, how sure can we be that the Board class has any instance data?

a. absolutely, since the Board constructor takes 2 parameters.
b. reasonably sure, since the Board constructor takes 2 parameters and there is an init() method with no return value
c. not sure at all, since the test is not based on any return value from a member method.
d. absolutely sure it does not, since the test is not based on any return value from a member method.
e. none of the above
1. // BoardTester3 - tests column initialization
2. import java.io.PrintWriter;
3. import java.io.StringWriter;
4. import junit.framework.*;
5. public class BoardTester3 extends TestCase {
6.     private Board tttB;
7.     private final int MAXROWS = 3;
8.     private final int MAXCOLS = 3;
9.     protected void setUp() throws Exception {
10.         tttB = new Board(MAXROWS, MAXCOLS);
11.         tttB.init();
12.     }
13. }
14. // board has been initialized, so we should get "" strings
15. public String void testGetRowColinits() {
16.     try {
17.         s = tttB.getSymbolAtMove(new Move(0, 0));
18.         assertEquals(true, true);
19.     } catch (MoveException e) {
20.         //should not have an exception
21.         e.printStackTrace();
22.         assertEquals(false, true);
23.     } finally {
24.         assertEquals("", s);
25.     }
26. }
27.
28. If one is using test-first methodology, what does the code in the
testGetRowColinits() method force the programmer to create?
29.     a. a 2-argument method called getSymbolAtMove()
30.     b. a method called getSymbolAtMove which may throw a MoveException
31.     c. a method which should not throw an exception when passed a Move with 0,0
32.     components
33.     d. a method which should return an empty String
34.     e. all of the above.
35. If we added more boundary test cases to testGetRowColinits() could we then be
36.     sure of the absence of bugs in the method?
37.     a. true
38.     b. false (no, one can only show the presence of bugs, not the absence. However,
39.     we can show there is a high-likelihood there are no bugs in a program
40.     segment, but not by test cases)
41. Several people were able to pass complicated Moderator-based tests by simply
42.     returning true. Would you write code just to "pass the test" if you were writing
43.     production code?
44.     a. yes, test-first says write just enough code to pass the test.
45.     b. no, I would add more related tests to make sure the method under test would do
46.     what the tester intended.
1. // BoardTester6 - tests boundaries of board
2. import java.io.PrintWriter;
3. import java.io.StringWriter;
4. import junit.framework.*;
5. 
6. public class BoardTester6 extends TestCase {
7.     private Board tttB;
8.     private final String Player1 = "X";
9.     private final String Player2 = "0";
10.    private final int MAXROWS = 3;
11.    private final int MAXCOLS = 3;
12.    protected void setUp() throws Exception {
13.        tttB = new Board(MAXROWS, MAXCOLS);
14.        tttB.init();
15.    }
16. 
17.    public void testSetMove() {
18.        try {
19.            tttB.setMove(new Move(0, 0), Player1);
20.            assertEquals(Player1, tttB.getSymbolAtMove(new Move(0, 0)));
21.            tttB.setMove(new Move(-1, 0), Player1);
22.            assertEquals(Player1, tttB.getSymbolAtMove(new Move(0, 0)));
23.        } catch (MoveException rce) {
24.            rce.printStackTrace(System.err);
25.            // shouldn't get here
26.            assertEquals(false, true);
27.        }
28.        try {
29.            tttB.setMove(new Move(0, 0), Player1);
30.            tttB.setMove(new Move(0, 0), Player2);
31.            tttB.setMove(new Move(0, 0), Player2);
32.            tttB.setMove(new Move(0, 0), Player2);
33.        } catch (MoveException rce) {
34.            rce.printStackTrace(System.err);
35.            assertEquals(true, true);
36.        }
37.    }
38. 
39.    The code in testSetMove() could be improved by:
40.        a. making sure that for every position on the board, setting that position to a
41.            symbol 'X' would ensure that fetching from that position would return the
42.            same symbol 'X'.
43.        b. moving the setMove() calls out from the same try-catch blocks as the
44.            getSymbolAtMove() calls
45.        c. one of a or b
46.        d. both a and b
47.        e. none of a or b
48. 
49.    Using test-first methodology, the code ensures that the getSymbolAtMove() method will include the throws keyword in its signature:
50.        a. true
51.        b. false (it might be that setMove() will throw an exception)
52. 
53.    Using test-first methodology, the code ensures that the setMove() method will include the throws keyword in its signature:
54.        a. true
55.        b. false (it might be that getSymbolAtMove() will throw an exception)
40. Using test-first methodology, the code ensures that at least one of
getSymbolAtMove() or setMove() will include the `throws` keyword in its
signature:
   a. **true** (one of the two must indicate that it will throw and exception or the
code won't compile. We are just not sure which)
   b. false

41. By running the code, the `assertEquals` statements will ensure that the `setMove()`
method will include the `throws` keyword in its signature:
   a. **true** (assume `setMove()` cannot throw a MoveException: then
getSymbolAtMove(new Move(0,0)) on line 21, must be *capable* of
throwing a MoveException or the catch portion would be unreachable.
The `assertEqual` statement on line 25, inside the catch block implies that
getSymbolAtMove(new Move(0,0)) does not throw an exception. Moving
to line 28, we assume `setMove()` cannot throw a MoveException.
Therefore, the `getSymbolAtMove(new Move(0,0))` must throw the
MoveException expected on line 32. `getSymbolAtMove(new Move(0,0))`
both throws and does not throw an exception. This is a contradiction, so
the assumption that `setMove()` cannot throw a MoveException is false.
`setMove()` must be able to throw a MoveException.
   b. false
1. // BoardTester8 - board to string
2. import java.io.PrintWriter;
3. import java.io.StringWriter;
4. import junit.framework.*;
5.
6. public class BoardTester8 extends TestCase {
7.     private Board tttB;
8.     private final int MAXROWS = 3;
9.     private final int MAXCOLS = 3;
10. 
11.     private Player tttP1;
12.     private Player tttP2;
13. 
14.     private final String Player1 = "X";
15.     private final String Player2 = "0";
16. 
17.     StringWriter sw;
18.     PrintWriter p_err;
19.     PrintWriter p_out;
20. 
21.     protected void setUp() throws Exception {
22.         tttB = new Board(MAXROWS, MAXCOLS);
23.         tttB.init();
24.         p_err = new PrintWriter(System.err);
25.         p_out = new PrintWriter(System.out);
26.         sw = new StringWriter();
27.     }
28. 
29.     public void testBoardtoString() {
30.         tttB.init(Player1);
31.         sw = new StringWriter();
32.         p_out = new PrintWriter(sw);
33.         p_out.print(tttB.toString());
34.         p_out.flush();
35.         assertEquals("XXXXXXXXX", sw.toString());
36.     }
37. }

42. Using test-first methodology, the code in testBoardtoString ensures that a
    toString() will be written by the programmer to ensure that the code compiles:
    a. true
    b. false (objects inherit toString() from the superclass Object, so the code will
       compile even if the programmer does not provide a toString() method for
       the Board class.)

43. By running the code, the assertEquals statements will ensure that the toString()
    method will be written by the programmer. Rewriting a method already provided
    by a superclass is called:
    a. rewriting
    b. overriding
    c. overloading
    d. overdrafting
    e. inheriting
44. setMove() method will include the throws keyword in its signature:
   a. true (assume setMove() cannot throw a MoveException: then getSymbolAtMove(new Move(0,0)) on line 21, must be capable of throwing a MoveException or the catch portion would be unreachable. The assertEqual statement on line 25, inside the catch block implies that getSymbolAtMove(new Move(0,0)) does not throw an exception. Moving to line 28, we assume setMove() cannot throw a MoveException. Therefore, the getSymbolAtMove(new Move(0,0)) must throw the MoveException expected on line 32. getSymbolAtMove(new Move(0,0)) both throws and does not throw an exception. This is a contradiction, so the assumption that setMove() cannot throw a MoveException is false. setMove() must be able to throw a MoveException.
   b. false

45. the testMove() method ensures:
   a. a Move class
   b. a Move constructor
   c. a 2-argument Move constructor
   d. a, b and c.
   e. only one of a, b or c.
46. At compile-time, testMove() ensures the programmer must write which of the following:
   a. a Move class
   b. a two-argument Move constructor
   c. an equals() method for the Move class
   d. a, b and c
   e. only two (2) of a, b and c. (equals() is inherited from the superclass, so this code does not require the programmer to write a new version of equals

47. Assume the code compiles and loads. At run-time, testMove() ensures the programmer must have written which of the following:
   a. a Move class
   b. a two-argument Move constructor
   c. an equals() method for the Move class (this test will fail if equals is not written by the programmer.)
   d. a, b and c
   e. none of the above

1. //Board
2. public class Board {
3.     int _row;
4.     int _col;
5.     Player [][][]_board;
6.  
7.     public Board(int maxrow, int maxcol) {
8.         maxrow = _row;
9.         maxcol = _col;
10.    }
11.  
12.     public void setPlayer(int row, int col, Player p) {
13.         _board[row][col] = p;
14.     }
15.  
16.     public static void main(String[] args) {
17.         int MAXROW=3;
18.         int MAXCOL=3;
19.         Player p = new Player("X");
20.         Board b = new Board(MAXROW,MIXCOL);
21.         b.setPlayer(MAXROW,MIXCOL,p);
22.     }
23. }

48. When run, the code in main() causes a runtime error traceable to line 20. The error occurs because we are trying to access the array at location MAXROW, MAXCOL, when the array holds places for 0 to MAXROW-1 rows and 0 to MAXCOLS-1 columns.
   a. true
   b. false (the error is attributable to the _board array never being created)
49. Assume line 10 is replaced with the following statement:
   \[
   \text{board} = \text{new Player[maxrow][maxcol]};
   \]
   Will the code in main() still cause a runtime error
   a. true
   b. false

50. Suppose line 20 is replace with the
   \[
   \text{b.setPlayer(MAXROW-1,MAXCOL-1,p)};
   \]
   51. Will the code in main() still cause a runtime error?
   a. true
   b. false

52. The root of the runtime error is found in code on lines:
   a. 1-5
   b. 7-11
   c. 12-14

1. // PlayerTester8 -
2. import junit.framework.*;
3. 
4. public class PlayerTester8 extends TestCase {
5.   private final int MAXROWS = 3;
6.   private final int MAXCOLS = 3;
7.   private Player tttP1;
8.   private Player tttP2;
9. 
10.  private final String Player1 = "X";
11.  private final String Player2 = "0";
12. 
13.  private PlayStrategy strategy;
14.  private Board tttB;
15. 
16.  protected void setUp() throws Exception {
17.    strategy = null;
18.    tttB = new Board(MAXROWS,MAXCOLS);
19.    tttP1 = new Player(Player1, strategy);
20.  }
21. 
22.  public void testGetMove() {
23.    try {
24.      tttP1.getMove(tttB);
25.      //assert statement here
26.    } catch( Exception ex) {
27.      // assert statement here
28.    }
29.  }
30. }
53. Assume that the user story states that `PlayStrategy` is an abstract class with an abstract method `getMove()`. When a `Player` object receives a method call for `getMove`, as in line 25, the `Player` object calls the `getMove()` method of the `PlayStrategy` subclass object that was passed to it on construction (line 20). Which assert statement should execute in `testGetMove()`?
   a. line 28
   b. line 26
   c. both.
   d. can't tell from the code
   e. neither

54. Assume we replace line 17 with the following statement.
   ```java
   strategy = new PlayStrategy()
   ```
   Will the code produce a compiler error?
   a. true (because `PlayStrategy` is abstract and we can't instantiate one).
   b. false