

**Course Experiment Report**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Course:** | Java Language | | | | | | |
|  |  | | | | | | |
| **Semester:** | 1-18th | **week** | 2nd | **year** | | 1st | **term** |
|  |  |  |  |  | |  |  |
| **Major:** | Software Engineering | | | | | **Class:** | 2019 |
|  |  | | | | |  |  |
| **Student name:** | Fu Ruoxuan | | **Student No.:** | | 222019321062060 | | |
|  |  | |  | |  | | |
| **Teacher:** | Wang Xiaomeng | | | | | | |

College of Computer and Information Science

|  |  |  |  |
| --- | --- | --- | --- |
| Project | Exp4 Objects and Classes | | |
| Time | 2020.11.4 | Type | □Verification □Design ☑Synthetical |
| 1. Answer the questions  (1) When will the no-arg constructor be automatically added?  No-arg constructor will be automatically added when the object is created without arguments and there are not no-arg constructor designed by programmer in the class.  (2) What is the difference between static members and object members?  A static member, also known as a class member, is a variable or method that has been modified with a static modifier. Object members are variables or methods that have not been modified with a static modifier inside. The main difference is that there is no static modifier. The result is that static members can be called directly from the class name, while obejct members have to be instantiated with the new identifier. Object members can access static members while Static members can not access object members.  (3) What do you think is the difference between procedural-oriented programming and object-oriented programming?  a) The thinking is no the same  Procedural-oriented programming put attention on procedure. The program is designed with the primary goal of what is happening.  Object-oriented programming considers objects as primary elements.  b) The Characteristic is not the same  Process-oriented programming is to analyze the steps needed to solve the problem, and then use functions to implement these steps step by step, when used one by one can be called in turn.  Object-Oriented programming’s basic building blocks of a system can be recognized as a set of identifiable discrete objects with unique static types and multiple possible dynamic types that share data and operations in different classes of the basic hierarchy.  (4) Other experience.  When printing an array of two-dimensional objects, you cannot use the members of the object directly when the object has a constructor with arguments. You need to skip objects that are not constructed in order to compile rightly.  When printing a two-dimensional array and changing the number of rows and columns, care needs to be taken that the index is correct  2. All Codes  1. Geometry: n-sided regular polygon  1）RegularPolygon.java   1. **package** exp4; 3. **public** **class** RegularPolygon { 4. **private** **int** n; 5. **private** **double** side; 6. **private** **double** x; 7. **private** **double** y; 9. **public** RegularPolygon(){ 10. **this**.n = 3; 11. **this**.side = 1; 12. **this**.x = 0; 13. **this**.y = 0; 14. } 15. **public** RegularPolygon(**int** number,**double** length){ 16. **this**.n = number; 17. **this**.side = length; 18. **this**.x = 0; 19. **this**.y = 0; 20. } 21. **public** RegularPolygon(**int** number,**double** length,**double** x,**double** y){ 22. **this**.n = number; 23. **this**.side = length; 24. **this**.x = x; 25. **this**.y = y; 26. } 28. **public** **int** getN() { 29. **return** n; 30. } 32. **public** **void** setN(**int** n) { 33. **this**.n = n; 34. } 36. **public** **double** getSide(){ 37. **return** side; 38. } 40. **public** **void** setSide(**double** side) { 41. **this**.side = side; 42. } 44. **public** **double** getX(){ 45. **return** x; 46. } 48. **public** **void** setX(**double** x) { 49. **this**.x = x; 50. } 52. **public** **double** getY(){ 53. **return** y; 54. } 56. **public** **void** setY(**double** y) { 57. **this**.y = y; 58. } 60. **public** **double** getPerimeter(){ 61. **return** n\*side; 62. } 64. **public** **double** getArea() { 65. **return** (n\*side\*side)/(4\*Math.tan(Math.PI/n)); 66. } 67. }   2）test.java   1. **package** exp4; 3. **public** **class** test { 4. **public** **static** **void** main(String[] args) { 5. RegularPolygon rp1 = **new** RegularPolygon(); 6. RegularPolygon rp2 = **new** RegularPolygon(6, 2.0); 7. RegularPolygon rp3 = **new** RegularPolygon(4, 3.0, 1, 1); 8. System.out.printf("The perimeter and the area of the " 9. + "first polygon are %4.2f and %4.2f\n", rp1.getPerimeter(), 10. rp1.getArea()); 11. System.out.printf("The perimeter and the area of the " 12. + "first polygon are %4.2f and %4.2f\n", rp2.getPerimeter(), 13. rp2.getArea()); 14. System.out.printf("The perimeter and the area of the " 15. + "first polygon are %4.2f and %4.2f\n", rp3.getPerimeter(), 16. rp3.getArea()); 17. } 18. } | | | |
| 2. gomokuGame   1. Stone.java 2. **package** gomokuGame; 4. **public** **class** Stone { 5. **public** **final** **static** **int** WHITE = 0; 6. **public** **final** **static** **int** BLACK = 1; 7. **private** **int** color; 9. **public** Stone(**int** color){ 10. **this**.setColor(color); 11. } 13. **public** **int** getColor(){ 14. **return** color; 15. } 17. **public** **void** setColor(**int** color){ 18. **this**.color = color; 19. } 21. **public** **static** String colorToString(**int** color){ 22. **return** color == WHITE ? "white": "black"; 23. } 25. **public** **char** getShape() { 26. **return** color == WHITE ? '\u25CB': '\u25CF'; 27. } 29. }   2） Player.java   1. **package** gomokuGame; 2. **import** java.util.Scanner; 3. **public** **class** Player { 4. **private** String name; 5. **private** **int** color; 7. **public** Player(String name, **int** color){ 8. **this**.setName(name); 9. **this**.setColor(color); 10. } 12. **public** String getName() { 13. **return** name; 14. } 16. **public** **void** setName(String name){ 17. **this**.name = name; 18. } 20. **public** **int** getColor() { 21. **return** color; 22. } 24. **public** **void** setColor(**int** color){ 25. **this**.color = color; 26. } 28. **public** **void** play(Board board,Scanner input){ 29. System.out.printf("Player %s[%s] put a stone at:", **this**.getName(),Stone.colorToString(color)); 30. **int** row = input.nextInt() - 1; 31. **int** column  = input.nextInt() - 1; 33. **boolean** success = board.putAStone(row,column, **new** Stone(color)); 34. **if**(!success) { 35. System.out.println("Illegal Input"); 36. } 37. } 39. }   3）Board.java   1. **package** gomokuGame; 3. **public** **class** Board { 5. **public**  **final** **static** **int** SIZE = 15; 6. **private** Stone[][] board; 7. **private** **int** remain; 8. **private** **int** whichColorToPlay; 9. **private** **int** winColor = -1; 11. **public** Board(){ 12. board = **new** Stone[SIZE][SIZE]; 13. remain = SIZE \* SIZE; 14. whichColorToPlay = (**int**)(Math.random()\*2); 15. **for**(**int** i = 0;i < SIZE;i++){ 16. **for**(**int** j = 0;j < SIZE;j++){ 17. board[i][j] = **null**; 18. } 19. } 20. } 22. **public** **void** printBoard() { 23. **for** (**int** i = 0; i < SIZE + 1; i++) { 24. **for** (**int** j = 0; j < SIZE + 1; j++) { 25. **if** (i == 0 && j == 0){ 26. System.out.print("   "); 27. } **else** { 28. **if** (i == 0){ 29. **if** (j < 10) 30. System.out.printf("%d  ",j); 31. **else** 32. System.out.printf("%d ",j); 33. } **else** **if**(j == 0){ 34. **if** (i < 10) 35. System.out.printf("%d  ",i); 36. **else** 37. System.out.printf("%d ",i); 38. } **else** **if** (board[i-1][j-1] != **null**) { 39. **if**(j < 10) 40. System.out.printf("%c  ",board[i-1][j-1].getShape()); 41. **else** 42. System.out.printf("%c ",board[i-1][j-1].getShape()); 43. } **else** 44. **if**(j < 10) 45. System.out.print("+  "); 46. **else** 47. System.out.print("+  "); 48. } 49. } 50. System.out.println(); 51. } 52. }  55. **public**  **boolean** putAStone(**int** row,**int** column, Stone stone){ 56. **if**(board[row][column]!=**null**){ 57. **return** **false**; 58. } **else** { 59. board[row][column] = stone; 60. remain--; 61. whichColorToPlay = stone.getColor() == Stone.WHITE ? Stone.BLACK:Stone.WHITE; 62. winColor = judge(row,column); 63. **return** **true**; 64. } 65. } 67. **public** **int** judge(**int** row,**int** column){ 68. String kernel = **new** String(**new** **char**[5]).replace("\0", 69. String.valueOf(board[row][column].getColor())); 70. // Determine whether there are five consecutive colored stones in the horizontal direction 71. StringBuffer lineX = **new** StringBuffer(); 72. **for**(**int** i = 0; i < SIZE; i++) { 73. **if**(board[row][i] != **null**) { 74. lineX.append(board[row][i].getColor()); 75. }**else** { 76. lineX.append("N"); 77. } 78. } 79. **if**(lineX.indexOf(kernel) >= 0) 80. **return** board[row][column].getColor(); 82. StringBuffer lineY = **new** StringBuffer(); 83. **for** (**int** i = 0;i < SIZE;i++){ 84. **if**(board[i][column]!=**null**) { 85. lineY.append(board[i][column].getColor()); 86. } **else**{ 87. lineY.append("N"); 88. } 89. } 90. **if** (lineY.indexOf(kernel) >= 0) 91. **return** board[row][column].getColor(); 93. StringBuffer lineDiagA = **new** StringBuffer(); 94. **for**(**int** i = 0;i < SIZE;i++){ 95. **for** (**int** j = 0;j < SIZE;j++) { 96. **if** (Math.abs(i - j) == Math.abs(row - column)) { 97. **if** (board[i][j] != **null**) { 98. { 99. lineDiagA.append(board[i][j].getColor()); 100. } 101. } **else** { 102. lineDiagA.append("N"); 103. } 104. } 105. } 106. } 107. **if**(lineDiagA.indexOf(kernel) >= 0) 108. **return** board[row][column].getColor(); 110. StringBuffer lineDiagB = **new** StringBuffer(); 111. **for**(**int** i = 0;i < SIZE;i++){ 112. **for** (**int** j = 0;j < SIZE;j++) 113. **if** ((i + j) == (row + column)) { 114. **if** (board[i][j] != **null**) { 115. { 116. lineDiagB.append(board[i][j].getColor()); 117. } 118. } **else** { 119. lineDiagB.append("N"); 120. } 121. } 122. } 123. **if**(lineDiagB.indexOf(kernel) >= 0) 124. **return** board[row][column].getColor();  127. **return** -1; 128. } 130. **public** **int** getRemain() { 131. **return** remain; 132. } 134. **public** **void** setRemain(**int** remain) { 135. **this**.remain = remain; 136. } 138. **public** **int** getWhichColorToPlay() { 139. **return** whichColorToPlay; 140. } 142. **public** **void** setWhichColorToPlay(**int** whichColorToPlay) { 143. **this**.whichColorToPlay = whichColorToPlay; 144. } 146. **public** **int** getWinColor() { 147. **return** winColor; 148. } 150. **public** **void** setWinColor(**int** winColor) { 151. **this**.winColor = winColor; 152. } 153. }   4）Gomoku.java   1. **package** gomokuGame; 3. **import** java.util.Scanner; 5. **public** **class** Gomoku { 6. **public**  **static** **void** main(String[] args) { 7. Scanner input = **new** Scanner(System.in); 8. System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Welcome to Gomoku Game\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"); 9. Board board = **new** Board(); 11. System.out.print("Enter player1 name:"); 12. String playerName1 = input.next(); 13. System.out.print("Enter player2 name:"); 14. String playerName2 = input.next(); 15. **int** playerColor1 = (**int**)(Math.random() \* 2); 16. **int** playerColor2 = Math.abs(playerColor1-1); 17. Player player1 = **new** Player(playerName1,playerColor1); 18. Player player2 = **new** Player(playerName2,playerColor2); 19. System.out.printf("Player1[%s, %s] VS Player2[%s, %s]\n", player1.getName(), Stone.colorToString(player1.getColor()), player2.getName(), 20. Stone.colorToString(player2.getColor())); 22. **while**(board.getWinColor()==-1&&board.getRemain()!=0) { 23. **if**(board.getWhichColorToPlay() == player1.getColor()) { 24. player1.play(board, input); 25. }**else** { 26. player2.play(board, input); 27. } 28. board.printBoard(); 29. } 31. **if**(board.getRemain() == 0) { 32. System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Game Draw\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"); 33. } **else** **if**(board.getWinColor()==playerColor1){ 34. System.out.printf("Player1 %s[%s] won the game", player1.getName(), 35. Stone.colorToString(player1.getColor())); 36. } **else** { 37. System.out.printf("Player2 %s[%s] won the game", player2.getName(), 38. Stone.colorToString(player2.getColor())); 39. } 40. input.close(); 41. } 43. } | | | |
|  | | | |

|  |  |  |
| --- | --- | --- |
| Evaluation | Code Correctness (60%): |  |
| Experience (40%): |  |
| Score： | |