Nyagah kelvin Mbugua

Scm211-0257/2017

Java cat

Question one

a. `ButtonCanvas` could be built using multiple inheritance. What does this mean in this context? [2 marks]

In this context, it implies that for the buttonCanvas to be able to implement both the Button and the Canvas classes simultaneously ,it will need to inherit both the Canvas and the Button classes i.e. Canvas class inherits the Button class then the buttonClass inherits the Canvas class or the other way around.

b. Give two reasons why this might be desirable. [2 marks]

i. This could eliminate coding redundancy because the need to restate all the Button and the Canvas class members in the buttonCanvas class is eliminated through multiple inheritance.

ii. It would ensure consistency in classes' members hence accuracy in intended implementations because the members of the buttonCanvas will have same members to its parent classes.

c. Give two complexities that arise in this case. [2 marks]

i. The subclass (buttonCanvas) may inherit even the undesired parts of the parent classes (Canvas and the Button classes).

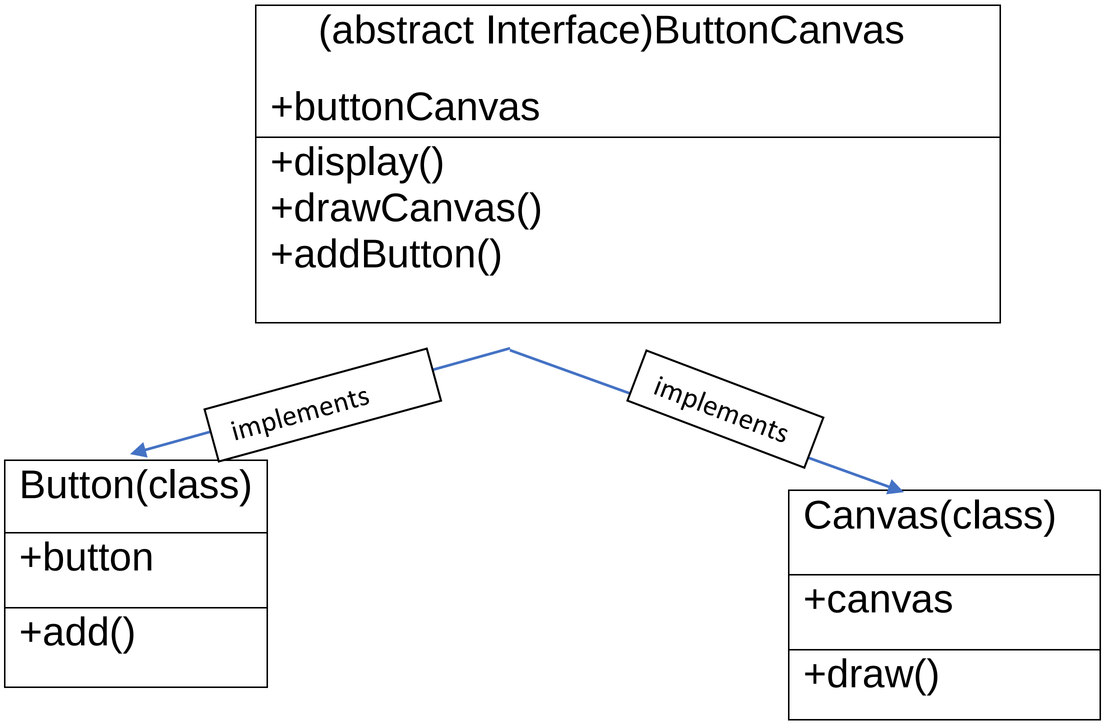
ii. The subclass (buttonCanvas) is fully dependent on the parent classes (Button and the Canvas classes).

d. Java interfaces originally contained only abstract methods and static final fields. How did this restriction avoid the complexities of extending multiple classes? [3 marks]

By abstraction methods style, the superclass only declares the general structure of the methods, without providing complete implementation of the methods; leaving it to each subclass to fill in its customized implementation. This ensure intended method implementation is achieved.

Methods from superclass declared as final cannot be overridden. This ensures the desired superclass methods are implemented throughout the entire subclasses set.

e. UML diagram for building ButtonCanvas Using abstract interfaces. [9 marks]



f. Recent versions of Java added default methods to interfaces. What is the impact of this with respect to multiple inheritance? [2 marks]

Default interfaces' methods are implicitly public. Regarding multiple inheritance, these methods ensures that subclasses include customized forms of the interfaces' methods which helps in maintaining interface-defined structure

Question two

a. Show how to set up a simple Java 2-dimensional array of `Boolean` values to represent a Life Board, with all cells initially “dead”. [2 marks]

```
boolean [ ][ ] cells = new boolean [ row ] [ col ];
```

b. For a location (i, j) on the board, give code that will decide whether the next state of that cell should be alive or dead. Make it clear how your code copes if the cell is at the boundary of the board. [4 marks]

```
public class NewClass {
private boolean[][] cells; // Will be set at construction
private int row;
private int col;
private void doGeneration() {
  boolean[][] nextGeneration = new boolean[row][col];

  for (int i = 1; i < row - 1; i++) {
    for (int j = 1; j < col - 1; j++) {
      int neighbors = 0;

      // Check surrounding cells.
      for (int neighborRow = i - 1; neighborRow <= i + 1; neighborRow++) {
        for (int neighborCol = col - 1; neighborCol <= col  + 1; neighborCol++) {
          if (neighborRow  != i || neighborCol != j) {
            if (cells[neighborRow  * row + neighborCol]) {
              neighbors++;
            }
          }
        }
      }

      Int new = i * row  + j;

      switch (neighbors) {
        case 0:
        case 1:
          nextGeneration[new][j] = false;
          break;
```

```
                case 2:
                    nextGeneration[new][j ]= cells[new][j];
                    break;

                case 3:
                    nextGeneration[new][j] = true;
                    break;

                default:
                    nextGeneration[new][j]= false;
                    break;
            }
        }
    }

    cells = nextGeneration;
}
```

c. Referring to part (b), write code that takes one board representing the current state of the game and fills in a second board-array with the state arrived at after one time step. What would happen if instead of using two arrays you wrote the new cell state directly back, using just a single copy of the board?
   [3 marks]

```
public class Life {

        private static final int ROW = 1000;
        private static final int COL = 1000;

    public static void main(String[] args) {
        Random randGen = new Random();
        boolean[][] nextBoard = new boolean[ROW+2][COL+2];
        boolean[][] currBoard = new boolean[ROW+2][COL+2];

        for(int i = 0; i <= currBoard.length; i++) {
            for (int j = 0; i <= nextBoard.length; j++) {

currBoard[i][j] = false;

  nextBoard[i][j] = false;

            }
        }
        for (int k = 1; k < currBoard.length - 1; k++) {
            for (int l = 1; l < currBoard.length - 1; l++) {
                if (randGen.nextInt(10)==2) {
                    currBoard[k][l] = true;
                }
            }
        }
    }

    public static int countLiveNeighbors(int row, int col, boolean[][] board){
        int count = 0;
        if (row-1 >= 0 && board[row-1][col] == true) {
            count++;
        }
        if (row+1 < COL && board[row+1][col] == true) {
            count++;
        }
        return count;
    }

  }
```

d. Re-work your solution to part (c) so that you can perform a time step using just one board. You may need to use a 1000-element vector to store information in a way that makes the update safe. [7 marks]

```
public class Life {

    private static final int ROW = 1000;

    private static final int COL = 1000;


  public static void main(String[] args) {

    Random randGen = new Random();

    boolean[][] nextBoard = new boolean[ROW+2][COL+2];

    boolean[][] currBoard = new boolean[ROW+2][COL+2];


    for(int i = 0; i <= currBoard.length; i++) {

      for (int j = 0; i <= nextBoard.length; j++) {

     currBoard[i][j] = false;
```

```
                nextBoard[i][j] = false;
            }
        }
        for (int k = 1; k < currBoard.length - 1; k++) {
            for (int l = 1; l < currBoard.length - 1; l++) {
                if (randGen.nextInt(10)==2) {
                    currBoard[k][l] = true;
                }
            }
        }
    }

    public static int countLiveNeighbors(int row, int col, boolean[][] board){
        int count = 0;
        if (row-1 >= 0 && board[row-1][col] == true) {
            count++;
        }
        if (row+1 < COL && board[row+1][col] == true) {
            count++;
        }
        return count;
    }

}
```

e. All the code you have written so far uses an array of `Boolean` values. Some programmers would instead use an array of `int` values and treat each of the 32 bits in each `int` as giving the status of a cell. Suppose you have a2-dimensional array of integers of size 1024 by 32 (that size is chosen so the array of integers may be viewed as a 1024 by 1024 array of bits): give code to retrieve a bit from a given position (i, j). [4 marks]

```
    //Using array of integers instead of booleans
package game_of_life;
public class GAME_OF_LIFE {
 public static final int ROW = 1000;
 public static final  int COL= 1000;
public static int[][] board = new int[ROW][COL];
   public static void main(String[] args) {
    RetrieveBit();
   }
   public static void CreateBoard()
     for(int i =0;i<ROW;i++){
         for(int j = 0;j<COL;j++){
             if(ROW%(i+2)==0&&ROW%(j+3)==0){
           board[i][j] = 1;
             }
             }


         }
   }


 public static void RetrieveBit(){
    for(int i =0;i<ROW;i++){
          for(int j = 0;j<COL;j++){
            System.out.print(board[i][j]);
              }
       }
 }
}
```