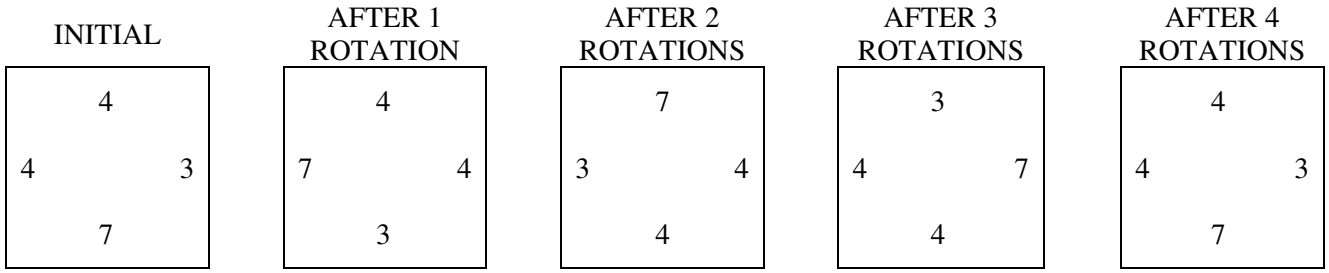
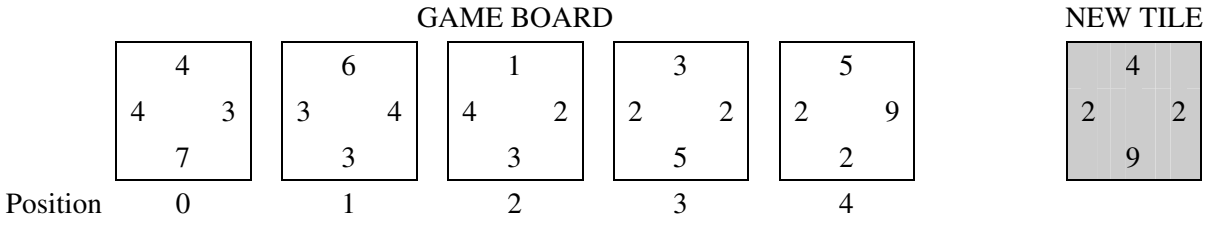
4. A game uses square tiles that have numbers on their sides. Each tile is labeled with a number on each of its four sides and may be rotated clockwise, as illustrated below.




The tiles are represented by the `NumberTile` class, as given below.

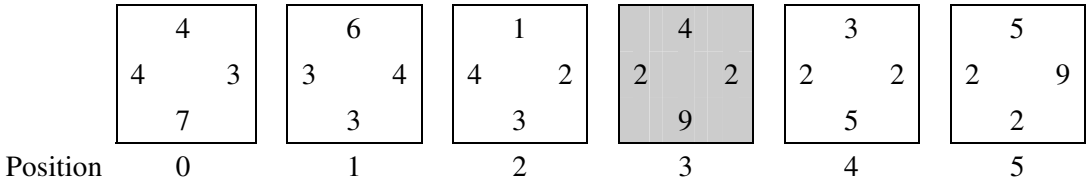
```
public class NumberTile
{
  /** Rotates the tile 90 degrees clockwise
   */
  public void rotate()
  {  /* implementation not shown */  }


  /** @return value at left edge of tile
   */
  public int getLeft()
  {  /* implementation not shown */  }


  /** @return value at right edge of tile
   */
  public int getRight()
  {  /* implementation not shown */  }

  // There may be instance variables, constructors, and methods that are not shown.
}
```

Tiles are placed on a game board so that the adjoining sides of adjacent tiles have the same number. The following figure illustrates an arrangement of tiles and shows a new tile that is to be placed on the game board.



In its original orientation, the new tile can be inserted between the tiles at positions 2 and 3 or between the tiles at positions 3 and 4. If the new tile is rotated once, it can be inserted before the tile at position 0 (the first tile) or after the tile at position 4 (the last tile). Assume that the new tile, in its original orientation, is inserted between the tiles at positions 2 and 3. As a result of the insertion, the tiles at positions 3 and 4 are moved one location to the right, and the new tile is inserted at position 3, as shown below.

GAME BOARD AFTER INSERTING TILE




A partial definition of the `TileGame` class is given below.

```
public class TileGame
{
  /** represents the game board; guaranteed never to be null */
  private ArrayList<NumberTile> board;

  public TileGame()
  { board = new ArrayList<NumberTile>(); }

  /** Determines where to insert tile, in its current orientation, into game board
   *  @param tile the tile to be placed on the game board
   *  @return the position of tile where tile is to be inserted:
   *            0 if the board is empty;
   *           -1 if tile does not fit in front, at end, or between any existing tiles;
   *           otherwise, 0 ≤ position returned ≤ board.size()
   */
  private int getIndexForFit(NumberTile tile)
  {  /* to be implemented in part (a) */  }

  /** Places tile on the game board if it fits (checking all possible tile orientations if necessary).
   *  If there are no tiles on the game board, the tile is placed at position 0.
   *  The tile should be placed at most 1 time.
   *  Precondition: board is not null
   *  @param tile the tile to be placed on the game board
   *  @return true if tile is placed successfully; false otherwise
   *  Postcondition: the orientations of the other tiles on the board are not changed
   *  Postcondition: the order of the other tiles on the board relative to each other is not changed
   */
  public boolean insertTile(NumberTile tile)
  {  /* to be implemented in part (b) */  }

  // There may be instance variables, constructors, and methods that are not shown.
}
```

# AP® COMPUTER SCIENCE A
# 2009 SCORING GUIDELINES

## Question 4: Tile Game

**Part (a)** `getIndexForFit` **6 points**

**+1** empty board
- **+1/2** checks for zero-sized board
- **+1/2** returns 0 if empty board detected

**+1** accesses tiles from board
- **+1/2** accesses any tile from board
- **+1/2** accesses all tiles of board (as appropriate) with no out-of-bounds access potential

**+1** uses tile values
- **+1/2** accesses left or right value of any tile
- **+1/2** compares left (right) value of parameter with right (left) value of any tile from board

**+2** determines tile fit
- **+1/2** only right value of parameter compared with left value of initial tile of board
- **+1/2** only left value of parameter compared with right value of final tile of board
- **+1** compares appropriate values of parameter and interior tiles of board

**+1** result
- **+1/2** returns located index if tile fits in board
- **+1/2** returns -1 if tile does not fit in board

**Part (b)** `insertTile` **3 points**

**+1/2** invokes `getIndexForFit` or replicates functionality with no errors

**+1 1/2** tile orientation
- **+1/2** invokes `rotate` on parameter
- **+1/2** performs **all** necessary rotations
- **+1/2** invokes `getIndexForFit` for each necessary orientation

**+1/2** adds tile correctly and only if `getIndexForFit` returns value other than -1

**+1/2** returns `true` if `getIndexForFit` returns value other than -1; `false` otherwise