SPECIMEN MATERIAL

AQA

AS
COMPUTER SCIENCE
(7516/1B)

Skeleton Program

```
/**
 * Skeleton Program code for the AQA AS Unit 1 SAM
 * This code should be used in conjunction with the Preliminary Material
 * written by the AQA Programmer Team developed in the Netbeans 6.9.1 IDE
 * Additional classes AQAConsole, AQAReadTextFile and AQAWriteTextFile may
 * be used.
 *
 * A package name may be chosen and private and public modifiers added.
 * Permission to make these changes to the Skeleton Program does not
 * need to be obtained from AQA/AQA Programmer
 *
 */

import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.util.Random;

public class Main {
  final int NO_OF_TRAPS = 2;
  final int N_S_DISTANCE = 5;
  final int W_E_DISTANCE = 7;
  AQAConsole console = new AQAConsole();
  Random random = new Random();

  /*
   * The Logical class is used to allow the value of a Boolean variable to be
returned from a subroutine.
   */
  class Logical {
    boolean is;
  }

  class CellReference {
    int noOfCellsSouth;
    int noOfCellsEast;
  }

  public class GameData {
    CellReference[] trapPositions = new CellReference[NO_OF_TRAPS];
    CellReference monsterPosition = new CellReference();
    CellReference playerPosition = new CellReference();
    CellReference flaskPosition = new CellReference();
    Logical monsterAwake = new Logical();
  }

  void setUpTrapPositions(CellReference[] trapPositions) {
    for (int count = 0; count < NO_OF_TRAPS; count++) {
      trapPositions[count] = new CellReference();
    }
```

```
  }
  void displayMenu() {
    console.println("MAIN MENU");
    console.println();
    console.println("1.  Start new game");
    console.println("2.  Load game");
    console.println("3.  Save game");
    console.println("4.  Play training game");
    console.println("9.  Quit");
    console.println();
    console.print("Please enter your choice: ");
  }

  int getMainMenuChoice() {
    int choice;
    choice = console.readInteger("");
    console.println();
    return choice;
  }

  void resetCavern(char[][] cavern) {
    int count1;
    int count2;
    for (count1 = 0; count1 < N_S_DISTANCE; count1++) {
      for (count2 = 0; count2 < W_E_DISTANCE; count2++) {
        cavern[count1][count2] = ' ';
      }
    }
  }

  CellReference getNewRandomPosition() {
    CellReference position;
    position = new CellReference();
    do {
      position.noOfCellsSouth = random.nextInt(N_S_DISTANCE);
      position.noOfCellsEast = random.nextInt(W_E_DISTANCE);
      } while ((position.noOfCellsSouth == 0) && (position.noOfCellsEast == 0));
    /*
     * a random coordinate of (0,0) needs to be rejected as this is the starting
position of the player
     */
    return position;
  }

  void setPositionOfItem(char[][] cavern, CellReference objectPosition, char item,
boolean newGame) {
    CellReference position;
    position = new CellReference();
    if (newGame && (item != '*')) {
      do {
        position = getNewRandomPosition();
      } while (cavern[position.noOfCellsSouth][position.noOfCellsEast] != ' ');
      objectPosition.noOfCellsSouth = position.noOfCellsSouth;
      objectPosition.noOfCellsEast = position.noOfCellsEast;
```

```
    }
    cavern[objectPosition.noOfCellsSouth][objectPosition.noOfCellsEast] = item;
  }

  void setUpGame(char[][] cavern, CellReference[] trapPositions, CellReference
monsterPosition, CellReference playerPosition, CellReference flaskPosition,
Logical monsterAwake, boolean newGame) {
    int count;
    resetCavern(cavern);
    if (newGame) {
      playerPosition.noOfCellsSouth = 0;
      playerPosition.noOfCellsEast = 0;
      monsterAwake.is = false;
    }
    for (count = 0; count < NO_OF_TRAPS; count++) {
      setPositionOfItem(cavern, trapPositions[count], 'T', newGame);
    }
    setPositionOfItem(cavern, monsterPosition, 'M', newGame);
    setPositionOfItem(cavern, flaskPosition, 'F', newGame);
    setPositionOfItem(cavern, playerPosition, '*', newGame);
  }

  void setUpTrainingGame(char[][] cavern, CellReference[] trapPositions,
CellReference monsterPosition, CellReference playerPosition, CellReference
flaskPosition, Logical monsterAwake) {
    resetCavern(cavern);
    playerPosition.noOfCellsSouth = 2;
    playerPosition.noOfCellsEast = 4;
    monsterAwake.is = false;
    trapPositions[0].noOfCellsSouth = 1;
    trapPositions[0].noOfCellsEast = 6;
    trapPositions[1].noOfCellsSouth = 3;
    trapPositions[1].noOfCellsEast = 4;
    monsterPosition.noOfCellsSouth = 0;
    monsterPosition.noOfCellsEast = 3;
    flaskPosition.noOfCellsSouth = 4;
    flaskPosition.noOfCellsEast = 5;
    setUpGame(cavern, trapPositions, monsterPosition, playerPosition,
flaskPosition, monsterAwake, false);
  }

  void loadGame(CellReference[] trapPositions, CellReference
monsterPosition, CellReference playerPosition, CellReference
flaskPosition, Logical monsterAwake) {
    GameData loadedGameData;
    loadedGameData = new GameData();
    String fileName;
    console.print("Enter the name of the file to load: ");
    fileName = console.readLine();
    console.println();
    readGame(fileName, loadedGameData);
    trapPositions[0] = loadedGameData.trapPositions[0];
    trapPositions[1] = loadedGameData.trapPositions[1];
```

```
    monsterPosition.noOfCellsSouth =
loadedGameData.monsterPosition.noOfCellsSouth;
    monsterPosition.noOfCellsEast = loadedGameData.monsterPosition.noOfCellsEast;
    playerPosition.noOfCellsSouth = loadedGameData.playerPosition.noOfCellsSouth;
    playerPosition.noOfCellsEast = loadedGameData.playerPosition.noOfCellsEast;
    flaskPosition.noOfCellsSouth = loadedGameData.flaskPosition.noOfCellsSouth;
    flaskPosition.noOfCellsEast = loadedGameData.flaskPosition.noOfCellsEast;
    monsterAwake.is = loadedGameData.monsterAwake.is;
  }

  void readGame(String filename, GameData loadedGameData) {
    try {
      DataInputStream in = new DataInputStream(new BufferedInputStream(new
FileInputStream(filename)));
      for (int count = 0; count < NO_OF_TRAPS; count++) {
        loadedGameData.trapPositions[count] = new CellReference();
        loadedGameData.trapPositions[count].noOfCellsSouth = in.readInt();
        loadedGameData.trapPositions[count].noOfCellsEast = in.readInt();
      }
      loadedGameData.monsterPosition.noOfCellsSouth = in.readInt();
      loadedGameData.monsterPosition.noOfCellsEast = in.readInt();
      loadedGameData.playerPosition.noOfCellsSouth = in.readInt();
      loadedGameData.playerPosition.noOfCellsEast = in.readInt();
      loadedGameData.flaskPosition.noOfCellsSouth = in.readInt();
      loadedGameData.flaskPosition.noOfCellsEast = in.readInt();
      loadedGameData.monsterAwake.is = (boolean) in.readBoolean();
      in.close();
    } catch (Exception e) {
      console.println(e.toString());
    }
  }

  void saveGame(CellReference[] trapPositions, CellReference monsterPosition,
CellReference playerPosition, CellReference flaskPosition, Logical monsterAwake) {
    GameData currentGameData;
    currentGameData = new GameData();
    String filename;
    currentGameData.trapPositions = trapPositions;
    currentGameData.monsterPosition = monsterPosition;
    currentGameData.playerPosition = playerPosition;
    currentGameData.flaskPosition = flaskPosition;
    currentGameData.monsterAwake = monsterAwake;
    console.print("Enter new file name: ");
    filename = console.readLine();
    console.println();
    writeGame(filename, currentGameData);
  }

  void writeGame(String filename, GameData currentGameData) {
    try {
      DataOutputStream out = new DataOutputStream(new BufferedOutputStream(new
FileOutputStream(filename)));
      for (int count = 0; count < NO_OF_TRAPS; count++) {
        out.writeInt(currentGameData.trapPositions[count].noOfCellsSouth);
```

```
        out.writeInt(currentGameData.trapPositions[count].noOfCellsEast);
      }
      out.writeInt(currentGameData.monsterPosition.noOfCellsSouth);
      out.writeInt(currentGameData.monsterPosition.noOfCellsEast);
      out.writeInt(currentGameData.playerPosition.noOfCellsSouth);
      out.writeInt(currentGameData.playerPosition.noOfCellsEast);
      out.writeInt(currentGameData.flaskPosition.noOfCellsSouth);
      out.writeInt(currentGameData.flaskPosition.noOfCellsEast);
      out.writeBoolean(currentGameData.monsterAwake.is);
      out.close();
    } catch (Exception e) {
      console.println(e.toString());
    }
  }

  void displayCavern(char[][] cavern, Logical monsterAwake) {
    int count1;
    int count2;
    for (count1 = 0; count1 < N_S_DISTANCE; count1++) {
      console.println(" ------------- ");
      for (count2 = 0; count2 < W_E_DISTANCE; count2++) {
        if (cavern[count1][count2] == ' ' || cavern[count1][count2] == '*' ||
(cavern[count1][count2] == 'M' && monsterAwake.is)) {
          console.print("|" + cavern[count1][count2]);
        } else {
          console.print("| ");
        }
      }
      console.println("|");
    }
    console.println(" ------------- ");
    console.println();
  }

  void displayMoveOptions() {
    console.println();
    console.println("Enter N to move NORTH");
    console.println("Enter E to move EAST");
    console.println("Enter S to move SOUTH");
    console.println("Enter W to move WEST");
    console.println("Enter M to return to the Main Menu");
    console.println();
  }

  char getMove() {
    char move;
    move = console.readChar();
    console.println();
    return move;
  }

  void makeMove(char[][] cavern, char direction, CellReference playerPosition) {
    cavern[playerPosition.noOfCellsSouth][playerPosition.noOfCellsEast] = ' ';
```

```
    switch (direction) {
      case 'N':
        playerPosition.noOfCellsSouth--;
        break;
      case 'S':
        playerPosition.noOfCellsSouth++;
        break;
      case 'W':
        playerPosition.noOfCellsEast--;
        break;
      case 'E':
        playerPosition.noOfCellsEast++;
        break;
    }
    cavern[playerPosition.noOfCellsSouth][playerPosition.noOfCellsEast] = '*';
  }

  boolean checkValidMove(CellReference playerPosition, char direction) {
    boolean validMove;
    validMove = true;
    if (!(direction == 'N' || direction == 'S' || direction == 'W'
            || direction == 'E' || direction == 'M')) {
      validMove = false;
    }
    return validMove;
  }

  boolean checkIfSameCell(CellReference firstCellPosition, CellReference
secondCellPosition) {
    boolean inSameCell;
    inSameCell = false;
    if ((firstCellPosition.noOfCellsSouth == secondCellPosition.noOfCellsSouth)
            && (firstCellPosition.noOfCellsEast ==
secondCellPosition.noOfCellsEast)) {
      {
        inSameCell = true;
      }
    }
    return inSameCell;
  }

  void displayWonGameMessage() {
    console.println("Well done!  You have found the flask containing the Styxian
potion.");
    console.println("You have won the game of MONSTER!");
    console.println();
  }

  void displayTrapMessage() {
    console.println("Oh no!  You have set off a trap.  Watch out, the monster is
now awake!");
    console.println();
  }
```

```
  void moveFlask(char[][] cavern, CellReference newCellForFlask, CellReference
flaskPosition) {
    cavern[newCellForFlask.noOfCellsSouth][newCellForFlask.noOfCellsEast] = 'F';
    cavern[flaskPosition.noOfCellsSouth][flaskPosition.noOfCellsEast] = ' ';
    flaskPosition = newCellForFlask;
  }

  void makeMonsterMove(char[][] cavern, CellReference monsterPosition,
CellReference flaskPosition, CellReference playerPosition) {
    CellReference originalMonsterPosition;
    boolean monsterMovedToSameCellAsFlask;
    originalMonsterPosition = new CellReference();
    originalMonsterPosition.noOfCellsEast = monsterPosition.noOfCellsEast;
    originalMonsterPosition.noOfCellsSouth = monsterPosition.noOfCellsSouth;
    cavern[monsterPosition.noOfCellsSouth][monsterPosition.noOfCellsEast] = ' ';
    if (monsterPosition.noOfCellsSouth < playerPosition.noOfCellsSouth) {
      monsterPosition.noOfCellsSouth++;
    } else if (monsterPosition.noOfCellsSouth > playerPosition.noOfCellsSouth) {
      monsterPosition.noOfCellsSouth--;
    } else if (monsterPosition.noOfCellsEast < playerPosition.noOfCellsEast) {
      monsterPosition.noOfCellsEast++;
    } else {
      monsterPosition.noOfCellsEast--;
    }
    monsterMovedToSameCellAsFlask = checkIfSameCell(monsterPosition,
flaskPosition);
    if (monsterMovedToSameCellAsFlask) {
      moveFlask(cavern, originalMonsterPosition, flaskPosition);
    }
    cavern[monsterPosition.noOfCellsSouth][monsterPosition.noOfCellsEast] = 'M';
  }

  void displayLostGameMessage() {
    console.println("ARGHHHHHH!  The monster has eaten you.  GAME OVER.");
    console.println("Maybe you will have better luck next time you play
MONSTER!");
    console.println();
  }

  void playGame(char[][] cavern, CellReference[] trapPositions, CellReference
monsterPosition, CellReference playerPosition, CellReference flaskPosition,
Logical monsterAwake) {
    int count;
    boolean eaten;
    boolean flaskFound;
    char moveDirection;
    boolean validMove;
    eaten = false;
    flaskFound = false;
    displayCavern(cavern, monsterAwake);
    do {
      do {
        displayMoveOptions();
```

```
        moveDirection = getMove();
        validMove = checkValidMove(playerPosition, moveDirection);
      } while (!validMove);
      if (moveDirection != 'M') {
        makeMove(cavern, moveDirection, playerPosition);
        displayCavern(cavern, monsterAwake);
        flaskFound = checkIfSameCell(playerPosition, flaskPosition);
        if (flaskFound) {
          displayWonGameMessage();
        }
        eaten = checkIfSameCell(monsterPosition, playerPosition);
        if (!monsterAwake.is && !flaskFound && !eaten) {
          monsterAwake.is = checkIfSameCell(playerPosition, trapPositions[0]);
          if (!monsterAwake.is) {
            monsterAwake.is = checkIfSameCell(playerPosition, trapPositions[1]);
          }
          if (monsterAwake.is) {
            displayTrapMessage();
            displayCavern(cavern, monsterAwake);
          }
        }
        if (monsterAwake.is && !eaten && !flaskFound) {
          count = 0;
          do {
            makeMonsterMove(cavern, monsterPosition, flaskPosition,
playerPosition);
            eaten = checkIfSameCell(monsterPosition, playerPosition);
            console.println();
            console.println("Press Enter key to continue");
            console.readLine();
            displayCavern(cavern, monsterAwake);
            count++;
          } while (!((count == 2) || eaten));
        }
        if (eaten) {
          displayLostGameMessage();
        }
      }
    } while (!(eaten || flaskFound || moveDirection == 'M'));
  }

  public Main() {
    char[][] cavern = new char[N_S_DISTANCE][W_E_DISTANCE];
    int choice;
    CellReference flaskPosition = new CellReference();
    Logical monsterAwake = new Logical();
    CellReference monsterPosition = new CellReference();
    CellReference playerPosition = new CellReference();
    CellReference[] trapPositions = new CellReference[NO_OF_TRAPS];
    setUpTrapPositions(trapPositions);
    do {
      displayMenu();
      choice = getMainMenuChoice();
      switch (choice) {
```

```
        case 1:
          setUpGame(cavern, trapPositions, monsterPosition, playerPosition,
flaskPosition, monsterAwake, true);
          playGame(cavern, trapPositions, monsterPosition, playerPosition,
flaskPosition, monsterAwake);
          break;
        case 2:
          loadGame(trapPositions, monsterPosition, playerPosition, flaskPosition,
monsterAwake);
          setUpGame(cavern, trapPositions, monsterPosition, playerPosition,
flaskPosition, monsterAwake, false);
          playGame(cavern, trapPositions, monsterPosition, playerPosition,
flaskPosition, monsterAwake);
          break;
        case 3:
          saveGame(trapPositions, monsterPosition, playerPosition, flaskPosition,
monsterAwake);
          break;
        case 4:
          setUpTrainingGame(cavern, trapPositions, monsterPosition,
playerPosition, flaskPosition, monsterAwake);
          playGame(cavern, trapPositions, monsterPosition, playerPosition,
flaskPosition, monsterAwake);
          break;
      }
    } while (!(choice == 9));
  }

  /**
   * @param args the command line arguments
   */
  public static void main(String[] args) {
    new Main();
  }
}
```