|  |  |
| --- | --- |
| 1. Refer to the code below,   public interface Sports {  void method1( );  void method2( );  int method3(double d);  }  public class Baseball implements Sports {  public Baseball( ) { . . . }  public void method1( ) { //some code…}  public void method2( ) { //some code…}  public int method3(double c ) { //some code…}  public int statevar1;  }    public class Football implements Sports {  public Football( ) { . . . }  public void method1( ) { //some code…}  public void method2( ) { //some code…}  public int method3(double c ) { //some code…}  public int statevar1;  }  public class Tester {  public static void main(String[] args) {  Sports x = new Baseball( );  Sports y = new Football( );  x.method2( );  y.method2( );  . . . more code . . .  }  } | |
| 1. Which methods, if any, in the Sports interface are abstract? | |
|  | /1 |
| 1. public class Hockey implements Sports {   //What methods, if any, must we implement here?  } | |
|  | /1 |
| 1. Look at the classes Baseball and Football. Both implement method1. Do both implementations have to have identical code? If so, why? | |
|  | /1 |
| 1. In the “more code” section of Tester what would the following return?   (x instanceof Sports) | |
|  | /1 |
| 1. In the “more code” section of Tester what would the following return?   (y instanceof Football) | |
|  | /1 |
| 1. The property of two classes being able to have methods of the same name (but with possibly different implementations) is known as | |
|  | /1 |
| 1. Modify the following class so that it will simultaneously inherit the Red class and implement both the Eagle and Bobcat interfaces.   public class Austria { . . . } | |
|  | /1 |

|  |
| --- |
| In this question, you will complete methods in classes that can be used to represent a multi-player game. You will be able to implement these methods without knowing the specific game or the players' strategies.  The GameState interface describes the current state of the game. Different implementations of the interface can be used to play different games. For example, the state of a checkers game would include the positions of all the pieces on the board and which player should make the next move.  The GameState interface specifies these methods. The Player class will be described in part (a). |
| A 37-line code segment reads as follows. Line 1: public interface Game State. Line 2: open brace. Line 3: forward slash, asterisk, asterisk, at, return true if the game is in an ending state, semicolon. Line 4: asterisk, false otherwise. Line 5: asterisk, forward slash. Line 6, highlighted: boolean is Game Over, open parenthesis, close parenthesis, semicolon. Line 7: blank. Line 8: blank. Line 9: forward slash, asterisk, asterisk, Precondition, colon, is Game Over, open parenthesis, close parenthesis, returns true. Line 10: asterisk, at, return the player that won the game or null if there was no winner. Line 11: asterisk, forward slash. Line 12, highlighted: Player get Winner, open parenthesis, close parenthesis, semicolon. Line 13: blank. Line 14: blank. Line 15: forward slash, asterisk, asterisk, Precondition, colon, is Game Over, open parenthesis, close parenthesis, returns false. Line 16: asterisk, at, return the player who is to make the next move. Line 17: asterisk, forward slash. Line 18, highlighted: Player get Current Player, open parenthesis, close parenthesis, semicolon. Line 19: blank. Line 20: blank. Line 21: forward slash, asterisk, asterisk, at, return a list of valid moves for the current player, semicolon. Line 22: asterisk, the size of the returned list is 0 if there are no valid moves. Line 23: asterisk, forward slash. Line 24, highlighted: Array List, open angular bracket, String, close angular bracket, get Current Moves, open parenthesis, close parenthesis, semicolon. Line 25: blank. Line 26: blank. Line 27: forward slash, asterisk, asterisk, Updates game state to reflect the effect of the specified move. Line 28: asterisk, at, param move a description of the move to be made. Line 29: asterisk, forward slash. Line 30, highlighted: void make Move, open parenthesis, String move, close parenthesis, semicolon. Line 31: blank. Line 32: blank. Line 33: forward slash, asterisk, asterisk, at, return a string representing the current Game State. Line 34: asterisk, forward slash. Line 35, highlighted: String to String, open parenthesis, close parenthesis, semicolon. Line 36: blank. Line 37: close brace. |
| The makeMove method makes the move specified, updating the state of the game being played. Its parameter is a String that describes the move. The format of the string depends on the game. In tic-tac-toe, for example, the move might be something like "X-1-1", indicating an X is put in the position (1, 1).  The Player class provides a method for selecting the next move. By extending this class, different playing strategies can be modeled.  A 22-line code segment reads as follows. Line 1: public class Player. Line 2: open brace. Line 3, highlighted: private String name, semicolon, end highlight, forward slash, forward slash, name of this player. Line 4: blank. Line 5: blank. Line 6, highlighted: public Player, open parenthesis, String a Name, close parenthesis. Line 7: open brace, name equals a Name, semicolon, close brace. Line 8: blank. Line 9: blank. Line 10, highlighted: public String get Name, open parenthesis, close parenthesis. Line 11: open brace, return name, semicolon, close brace. Line 12: blank. Line 13: blank. Line 14: forward slash, asterisk, asterisk, This implementation chooses the first valid move. Line 15: asterisk, Override this method in subclasses to define players with other strategies. Line 16: asterisk, at, param state the current state of the game, semicolon, its current player is this player. Line 17: asterisk, at, return a string representing the move chosen, semicolon. Line 18: asterisk, open double quote, no move, close double quote, if no valid moves for the current player. Line 19: asterisk, forward slash. Line 20, highlighted: public String get Next Move, open parenthesis, Game State, state, close parenthesis. Line 21: open brace, forward slash, asterisk, implementation not shown, asterisk, forward slash, close brace. Line 22: close brace. |
| 1. The method getNextMove returns the next move to be made as a string, using the same format as that used by makeMove in GameState. Depending on how the getNextMove method is implemented, a player can exhibit different game-playing strategies.   Write the complete class declaration for a RandomPlayer class that is a subclass of Player. The class should have a constructor whose String parameter is the player's name. It should override the getNextMove method to randomly select one of the valid moves in the given game state. If there are no valid moves available for the player, the string "no move" should be returned. |
| 1. The GameDriver class is used to manage the state of the game during game play. The GameDriver class can be written without knowing details about the game being played   A 15-line code segment reads as follows. Line 1: public class Game Driver. Line 2: open brace. Line 3, highlighted: private Game State state, semicolon, forward slash, forward slash, the current state of the game. Line 4: blank. Line 5, highlighted: public Game Driver, open parenthesis, Game State initial, close parenthesis. Line 6: open brace, state equals initial, semicolon, close brace. Line 7: blank. Line 8: blank. Line 9: forward slash, asterisk, asterisk, Plays an entire game comma as described in the problem description. Line 10: asterisk, forward slash. Line 11, highlighted: public void play, open parenthesis, close parenthesis. Line 12: open brace, forward slash, asterisk, to be implemented in part, (b), asterisk, forward slash, close brace. Line 13: blank. Line 14: forward slash, forward slash, There may be fields comma constructors comma and methods that are not shown. Line 15: close brace. |
| Write the GameDriver method play. This method should first print the initial state of the game. It should then repeatedly determine the current player and that player's next move, print both the player's name and the chosen move, and make the move. When the game is over, it should stop making moves and print either the name of the winner and the word "wins" or the message "Game ends in a draw" if there is no winner. You may assume that the GameState makeMove method has been implemented so that it will properly handle any move description returned by the Player getNextMove method, including the string "no move".  Complete method play below  A 3-line code segment reads as follows. Line 1: forward slash, asterisk, asterisk, Plays an entire game comma, as described in the problem description. Line 2: asterisk, forward slash. Line 3: public void play, open parenthesis, close parenthesis. |