AP® COMPUTER SCIENCE A
2007 SCORING GUIDELINES

Question 4: Game Design (Design)

Part A: RandomPlayer **4 points**

+1/2 `class RandomPlayer extends Player`

+1 constructor
- **+1/2** `public RandomPlayer(String aName)`
- **+1/2** `super(aName)`

+2 1/2 `getNextMove`
- **+1/2** `state.getCurrentMoves()`
- **+1** if no moves
  - **+1/2** test if size = 0
  - **+1/2** return "no move" only if 0 moves
- **+1** if moves
  - **+1/2** select random move index
  - **+1/2** return random move

Part B: play **5 points**

+1/2 print initial state (OK to print in loop)

+3 make repeated moves
- **+1** repeat until `state.isGameOver()`
- **+1/2** `state.getCurrentPlayer()`
- **+1/2** `player.getNextMove(state)`
- **+1/2** display player and move
- **+1/2** make move

+1 1/2 determine winner
- **+1/2** `state.getWinner()`
- **+1/2** display message if draw (if `getWinner` returns null)
- **+1/2** display message if winner

lose both if done before game ends

AP® COMPUTER SCIENCE A
2007 CANONICAL SOLUTIONS

Question 4: Game Design (Design)

PART A:

```
public class RandomPlayer extends Player
{
    public RandomPlayer(String aName)
    {
        super(aName);
    }

    public String getNextMove(GameState state)
    {
        ArrayList<String> possibleMoves = state.getCurrentMoves();
        if (possibleMoves.size() == 0) {
            return "no move";
        }
        else {
            int randomIndex = (int)(Math.random()*possibleMoves.size());
            return possibleMoves.get(randomIndex);
        }
    }
}
```

PART B:

```
public void play()
{
    System.out.println("Initial state:" + state);

    while (!state.isGameOver()) {
        Player currPlayer = state.getCurrentPlayer();
        String currMove = currPlayer.getNextMove(state);
        System.out.println(currPlayer.getName() + ": " + currMove);
        state.makeMove(currMove);
    }

    Player winner = state.getWinner();
    if (winner != null) {
        System.out.println(winner.getName() + " wins");
    }
    else {
        System.out.println("Game ends in a draw");
    }
}
```

4a,

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.

```
public class RandomPlayer extends Player {
    Random rand = RandNumGenerator.getInstance();
    public RandomPlayer(String aName) {
        super(aName);
    }
    public getNextMove(GameState state) {
        int numMoves = state.getCurrentMoves().size();
        if (numMoves == 0) {
            return "no move";
        }
        int choice = rand.nextInt(numMoves);
        return state.getCurrentMoves().get(choice);
    }
}
```

Part (b) begins on page 20.

A4a2

Complete method `play` below.

```
/** Plays an entire game, as described in the problem description
 */
public void play()
    System.out.println (state);
    while (!state.isGameOver()) {
        Player current = state.getCurrentPlayer();
        System.out.println (current.getName());
        String nextMove = current.getNextMove();
        System.out.println (nextMove);
        state.makeMove (nextMove);
    }
    if (state.getWinner() == null) {
        System.out.println ("Game ends in a draw");
    } else {
        System.out.println (state.getWinner().getName() + " wins");
    }
```

GO ON TO THE NEXT PAGE.

A4b,

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.

```
public class RandomPlayer extends Player
{

  private String name;
  public RandomPlayer( String aName )
  {
    name = aName;
  }
  public String getNextMove( GameState state )
  {
      ArrayList<String> validMoves = state.getCurrentMoves();
      if ( validMoves.size() = 0)
      {  String nM = "no move";
         return nM;
      }
      else
      {
        Random randNum = RandNumGenerator.getInstance();
        return ( validMoves.get( randNum.nextInt() ));
      }
  }
}
```

Part (b) begins on page 20.

Complete method `play` below.

```
/** Plays an entire game, as described in the problem description
 */
public void play()
{
  System.out.println(state.toString());
  while (!isGameOver)
  {
    Player p = state.getCurrentPlayer();
    ArrayList<String> validMoves = state.getCurrentMoves();
    Random randNum = RandNumGenerator.getInstance();
    state.makeMove(validMoves.get(randNum.nextInt()));
  }
  Player w = state.getWinner();
  if (w == null)
    System.out.println("Game ends in a draw");
  else
    System.out.println(w.getName() + " wins");
}
```

Attc 1

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.

```
Public /class  RandomPlayer extends Player

{
  Private String name;
    public Player(String aNone)
        name = aName;

    Public String getname();
        return name;
    }

  ArrayList<String> getcurrentmoves();

    Public String getNextMove(GameState State = null;

    if(getcurrentmoves = false)
        return " no more "

      else
    Void MakeMove(string move)

    }
```

Part (b) begins on page 20.

Complete method `play` below.

```
/** Plays an entire game, as described in the problem description
 */
public void play()
 {
    super (Game State)
      System.out.println (State)

    public String iget name()
        return name;

   public String getNext Move (Gamestate state)
        return Nextmove;

    void make move (String move)
       if ( is Game over = true)
System.out. println (name + "wins");
    else if ( get current moves = false)
      system .out . println ( "Game ends in a draw")
        }
```

AP® COMPUTER SCIENCE A
2007 SCORING COMMENTARY

Question 4

Overview

This question centered on abstraction, class design, and inheritance. Students were provided with an abstract framework for representing different types of games, including a `GameState` interface for capturing the state of a particular game and a `Player` class for representing a game player. In part (a) students were required to extend `Player` by designing and implementing a `RandomPlayer` class that always selects its move at random. This involved knowing the syntax of inheritance and also recognizing which methods needed to be overridden. Overriding the `getNextMove` method required calling the `getCurrentMoves` method defined by the `GameState` interface, randomly selecting a move (if one exists), and returning that move. In part (b) students were required to implement the `play` method of a `GameDriver` class, which calls the appropriate `GameState` and `Player` methods to alternate player moves until the game is over.

Sample: A4a
Score: 8½

For part (a) of this solution, the code includes the proper class header as well as the correct constructor header. The constructor includes a correct call to `super`. The `getNextMove` method properly locates the current moves that are possible and checks to see if there are any possible moves. It then correctly returns either "no move" or a randomly selected move. The solution earned all 4 points available for this portion of the question.

In part (b) the solution properly prints the state of the game. Both the loop and the call to `getCurrentPlayer` are correct. The call to `getNextMove` is incorrect because it is missing the parameter (`state`), so this part of the solution lost ½ point. The display of the player's name and move works as requested, and the move is made correctly. The `getWinner` method is properly called and checked and the correct message is printed, so all the remaining points were awarded for this solution.

Sample: A4b
Score: 5½

The solution for part (a) includes the proper class header and constructor header but is missing the call to `super()` so it lost that ½ point. The `getNextMove` method properly locates the current moves that are possible and checks to see if there are any possible moves. It then returns either "no move" or a randomly selected move. The ½ point for random was lost because of the missing parameter on the call to `nextInt`. The student was awarded 3 of the 4 possible points for this part of the question.

In part (b) the code properly prints the state of the game. Because the call to `isGameOver` is incorrectly implemented (it must be called on the `state` object) the solution lost the 1 point awarded for this check. The current player is properly accessed and earned that ½ point. The student attempts to reimplement the `getNextMove` method, so the `getNextMove`, display, and `makeMove` credit was lost. The `getWinner` method is properly called and checked, and the correct message is printed, so the last three ½ points were earned. For this part the student earned 2½ out of 5 possible points.

Question 4 (continued)

Sample: A4c
Score: 1½

Part (a) of this student's solution includes the proper class header and earned that ½ point, but the constructor header is incorrect and there is no call to `super()`, which lost the credit given for those actions. The method `getNextMove` does not properly locate the current moves that are possible. The check for `false` is not the correct way to determine whether the number of moves is 0, but the correct message is returned if no moves are available earning that ½ point. The remaining code does not fit the question requirements, so the solution earned no additional credit. Part (a) earned 1 out of 4 possible points.

Part (b) properly prints the state of the game and earned that ½ point but earned no further credit for this part of the question. There is no loop, no call to `getCurrentPlayer`, no call to `getNextMove` (instead the method is reimplemented), no call to `makeMove` (again the method is reimplemented), and the test for a win is incorrect. For this part the student earned ½ point out of 5 possible points.