Carleton University
Department of Systems and Computer Engineering
SYSC 2004 A - Object-Oriented Software Development – Fall 2012
Midterm Exam – October 11th, 2012
Sample Solutions

Please read these instructions before you answer any of the exam questions:

1. You have 75min (1hr15min) for four questions worth 40 marks on 10 pages. (Question 4 is a bonus question, so the exam will be marked out of 30.)
2. The midterm exam is closed book. **Calculators are not permitted**.
3. Do not talk to any other students from the time the exam begins until your completed exam has been submitted and you have left the exam room.
4. **Ask a question only if you believe there is a mistake on the exam**. Otherwise, make a reasonable assumption and proceed.
5. The questions on the midterm are based on a `Pick-A-Number` game. In each round of this game, first the game picks a number between 1 and "max", and then each player tries to guess the number. The winner of the round is the player that is closest to the original number chosen for that round by the game (don't worry about how we calculate the round winner if two or more players are equally close). The player that wins the most rounds wins the game (there may be a tie if several players win the same number of rounds).
6. API documentation for `ArrayList` and `Random` are provided at the end of this question paper, along with the BlueJ class diagram for `Pick-A-Number`. **You may carefully remove the reference page from the exam** (page 10).

Question 1 (7 marks)

In this question you are to develop a class called "`Pick`" that models a random number generator used to generate integers between 1 and "max".

a) What class from the Java libraries will be helpful in writing class `Pick`? (**1 mark**)

```
Random
```

b) How do we include this Java library class in our code? (**1 mark**)

```
import java.util.Random;
```
or
```
import java.util.*;
```

c) Given the information below, complete the fields, constructor and method `guess`:

```
public class Pick
{
    /** our random number generator */  // 0.5 marks

    private Random r; // insert your answer to a) here

    /** we will generate random numbers between 1 and max */
    private int max;

    /** default maximum, if invalid value specified */
    public final static int DEFAULT_MAX = 5;

    /**
     * Constructor for objects of class Pick.  The parameter
     * given must be at least 1, otherwise we use DEFAULT_MAX.
     *
     * @param int The largest number we will generate
     */
    public Pick(int max) // 2.5 marks
    {
        r = new Random(); // 1 mark
        if (max < 1) // 0.5 marks
        {
            this.max = DEFAULT_MAX; // 0.5 marks
        }
        else
        {
            this.max = max; // 0.5 marks
        }
    }

    /**
     * Returns a random number between 1 and max.
     *
     * @return int A number between 1 and max
     */
    public int guess() // 2 marks
    {
        return r.nextInt(max) + 1; // 1 mark for nextInt and
                                   // 0.5 marks for "+1" and
                                   // 0.5 marks for "return"
        // can be done in two or three steps, such as:
        // int temp;
        // temp = r.nextInt(max);
        // return temp+1;
    }
}
```

Question 2 (4 marks)

In this question you are to complete a class called "`Player`" that models a player in a game involving guessing numbers. Given the fields and method signatures below, complete the constructor and the missing method:

```
public class Player
{
    /** The player's name. */
    private String name;

    /** The Pick object shared by all players. */
    private Pick choose;

    /** The random value most recently picked by this player. */
    private int currentGuess;

    /** The player's score (number of wins). */
    private int totalScore;

    /**
     * Constructs a new player using the specified generator.
     *
     * @param name The player's name
     * @param choose The Pick object shared by all players
     */
    public Player(String name, Pick choose)// 2 marks as follows:
    {
        this.name = name; // 0.5 marks
        this.choose = choose; // 0.5 marks
        currentGuess = 0; // optional; can set to anything
        totalScore = 0;  // 1 mark
    }

    /** Resets this player's total score to 0. */
    public void resetTotalScore()
    {
       totalScore = 0;
    }

    /**
     * Returns this player's cumulative score.
     *
     * @return The player's number of wins so far
     */
    public int score()
    {
       return totalScore;
    }
```

```
    /**
     * Returns this player's name.
     *
     * @return The name of this player.
     */
    public String name()
    {
       return name;
    }

    /**
     * Returns this player's guess.
     *
     * @return The guess by this player.
     */
    public int number()
    {
       return currentGuess;
    }

    /**
     * The player takes a turn, i.e. the player makes a guess
     * and the guess is stored in currentGuess.
     */
    public void takeTurn() // 2 marks
    {
        currentGuess = choose.guess();
        // 0.5 marks for left hand side; 1.5 for right
    }
}
```

Question 3 (19 marks)

In this question you are to complete a class called "`PickANumber`" that models a game where players (a list of Player objects) try to guess integers between 1 and "max".

Complete the missing portions of this class as indicated on the next four pages:

```
import java.util.ArrayList; // purposely omitted (no marks)

public class PickANumber
{
    /** The number of rounds in a game. */
    private int numberOfRounds;

    /** The Pick object used in this game. */
    private Pick choose;

    /** The highest number that can be guessed. */
    private int max;

    /** The list of players in the game. */ // 1 mark
```

```
private ArrayList<Player> players; // complete this line
// 0.5 marks for "ArrayList" and 0.5 for "<Player>"

/** The current (random) number that players try to guess. */
private int n; // between 1 and max

/**
 * An array containing the players that won each round.
 * (i.e. the player that won the first round is stored in
 * winners[0], the 2nd round winner in winners[1], etc.)
 */                                          // 1 mark
private Player[] winners; // complete this line
// 0.5 marks for "Player" and 0.5 for "[]"
/**
 * Constructs a new PickANumber game with the specified
 * number of rounds and the given maximum guess.  The
 * constructor must do something sensible if either parameter
 * is unreasonable.  Ensure that you initialize all fields.
 *
 * @param rounds The number of rounds in a game
 * @param max The largest number we can guess
 */
public PickANumber(int rounds, int max) // 4.5 marks as:
{
    if (rounds < 1) // invalid value
    {
        numberOfRounds = 5; // any reasonable value
    }
    else
    {
        numberOfRounds = rounds;
    }

    if (max < 1) // invalid value
    {
        this.max = Pick.DEFAULT_MAX; // this is public
    }  // no deduction if used another reasonable value
    else
    {
        this.max = max;
    }

    n = 0; // optional and can initialize to anything

    // must use this.max and not max
    choose = new Pick(this.max); // 1 mark

    // this uses the ArrayList default constructor
    players = new ArrayList<Player>(); // 1 mark

    // must use numberOfRounds, not rounds
    winners = new Player[numberOfRounds]; // 1 mark
}
```

```
/**
 * Adds a player to the game
 *
 * @param String Player's name
 */
public void addPlayer(String name) // 1.5 marks
{
    players.add(new Player(name,choose));
    // 0.5 marks for players.add(...),
    // 0.5 marks for new Player(...) and
    // 0.5 marks for Player(name,choose)
    // could be done in two steps:
    //   Player p = new Player(name,choose);
    //   players.add(p);
}

/**
 * Plays one game of Pick-A-Number.
 */
public void playGame() // 3 marks as follows:
{
   // if we have at least two players we proceed,
   // otherwise this method does nothing

   if (players.size()>=2) // complete the "if" (1 mark)
   {
        // reset all player scores to 0

        for (Player p : players) // 1 mark
        {
           p.resetTotalScore(); // 1 mark
        }
        // can also have something like:
        // for (int j=0; j<players.size(); j++)
        // {
        //    players.get(j).resetTotalScore();
        // }
        for (int round = 1; round <= numberOfRounds; round++)
        {
            playOneRound(round);
        }

        // use method announceResults() to get the winner(s)
        // and print this information
        . . . // you don't have to write this code

    } // end if
} // end playGame
```

```
/**
 * Play one round of the game.
 *
 * @param round the round number.
 */
private void playOneRound(int round) // 3 marks as:
{
   System.out.println("Round " + round + ":");

   // game picks a number (from 1 to max) and stores it in n
    n = choose.guess(); // 1 mark

   // each player takes a turn (i.e. invokes takeTurn())
    for (Player p : players)
    {
       p.takeTurn();
    }
    // 1 mark: could use a regular for loop as above

   Player roundWinner = getRoundWinner();

   // store the round winner (see line above) in the
   // appropriate element of the winners array
    winners[round-1] = roundWinner; // 1 mark
}


/**
 * Determines the round winner.  Updates that player's score
 * and returns the player that won the round.
 * @return Player The player that won the round.
 */
private Player getRoundWinner()
{
   . . . // you don't have to write this code
}
```

```
    /**
     * Returns a list of a player or players that won the game.
     * (Note that you need only use the list of players here --
     * you do not need to use the winners array.)
     *
     * @return The list of player(s) that won.  (5 marks)
     */
    private ArrayList<Player> announceWinners() // 0.5 marks
    {
       int highScore = 0; // local variable for highest score

       // find the highest score among all players and store
       // the highest score in highScore (declaration above)
       for (Player p : players) // 0.5 marks for loop
       {
          if (p.score()>highScore) // 0.5 marks for if
          {
             highScore = p.score(); // 0.5 marks for =
          }
       }
       // again, both here and below, regular for loops
       // could be used

       // put all players with the highest score (highScore) in
       // a (new) list and return the list. (If there is a tie,
       // there will be more than one player with highScore.)
       ArrayList<Player> winlist = new ArrayList<Player>();
              // 1 mark for declaration above

       for (Player p : players) // 0.5 marks for loop
       {
          if (p.score()==highScore) // 0.5 marks for if
          {
             winlist.add(p); // 0.5 marks for add
          }
       }
       return winlist; // 0.5 marks for returning list
    }

}
```

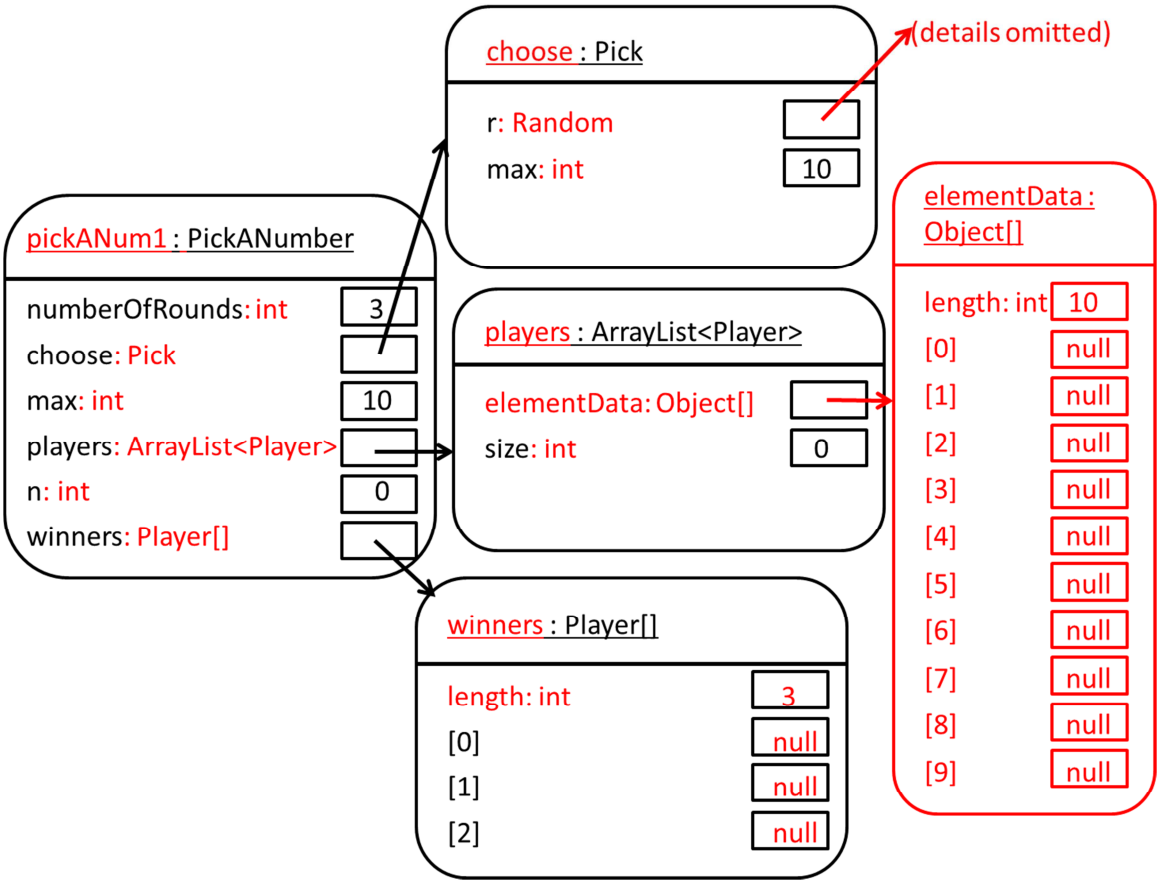
Question 4 (10 bonus marks)

Assume that we execute the line of code:

```
PickANumber pickANum1 = new PickANumber(3,10);
```

Draw an Object diagram below showing all objects and all their fields once this line of code has finished executing.

Note: You do not have to show the details of any Random objects, and you do not have to show the details of the array associated with any ArrayList objects.



Notes:
For full marks everything in black must be present. Things in red are optional.
4 marks for the PickANumber object and 2 marks each for the Pick, ArrayList<Player>, and Player[] objects.

Reference Material: (You may remove this page from the exam.)

API Summary - Class ArrayList<E>

```
// Constructs an empty list with an initial capacity of ten.
ArrayList();

// Appends o (of type E) to the end of this list. Returns true if
// successful, otherwise returns false.
boolean add(E o);

// Returns the number of objects stored in this list.
int size();

// Returns the object (of type E) at the specified position (index)
// in the list. index must be >= 0 and < size().
// The object is not removed from the list.
E get(int index);

// Returns true if this list has no elements, otherwise returns false.
boolean isEmpty();

// Removes the object (of type E) at the specified position (index)
// in the list. index must be >= 0 and < size(). Shifts any subsequent
// elements to the left (subtracts one from their indices).
// Returns the object that was removed from the list.
E remove(int index);
```

API Summary - Class Random

```
// Constructs a new random number generator.
Random();

// Returns a random integer between -2147483648 and 2147483647,
// inclusive.
int nextInt();


// Returns a random integer between 0 and n-1, inclusive.
int nextInt(int n);
```

Pick-A-Number Game Class Diagram (Note: arrows indicate "uses" relationships.)