ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN

CCPG1009 - DISEÑO DE SOFTWARE

TERCERA EVALUACIÓN - II TÉRMINO 2019

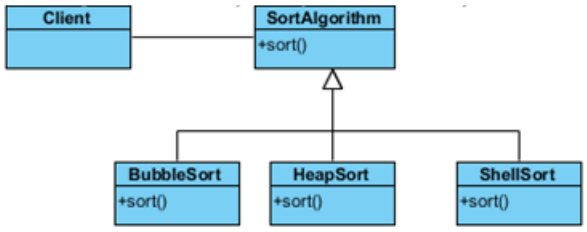
Nombre: _____ ***Matrícula:*** _____

COMPROMISO DE HONOR: Al firmar este compromiso, reconozco que el presente examen está diseñado para ser resuelto de manera individual, que puedo usar un lápiz o esferográfico; que sólo puedo comunicarme con la persona responsable de la recepción del examen; y, cualquier instrumento de comunicación que hubiere traído, debo apagarlo y depositarlo en la parte anterior del aula, junto con algún otro material que se encuentre acompañándolo. Además, no debo usar calculadora alguna, consultar libros, notas, ni apuntes adicionales a los que se entreguen en esta evaluación. Los temas debo desarrollarlos de manera ordenada. Firmo el presente compromiso, como constancia de haber leído y aceptado la declaración anterior. "Como estudiante de ESPOL me comprometo a combatir la mediocridad y actuar con honestidad, por eso no copio ni dejo copiar". _____ Firma	_____ 100 _____ Firma

Sección A

1. Defina cohesión y acoplamiento. Explique las razones por las cuales maximizar cohesión y minimizar acoplamiento produce sistemas más fáciles de mantener. **[8%]**

2. ¿A qué patrón de diseño corresponde el siguiente diagrama? **[4%]**

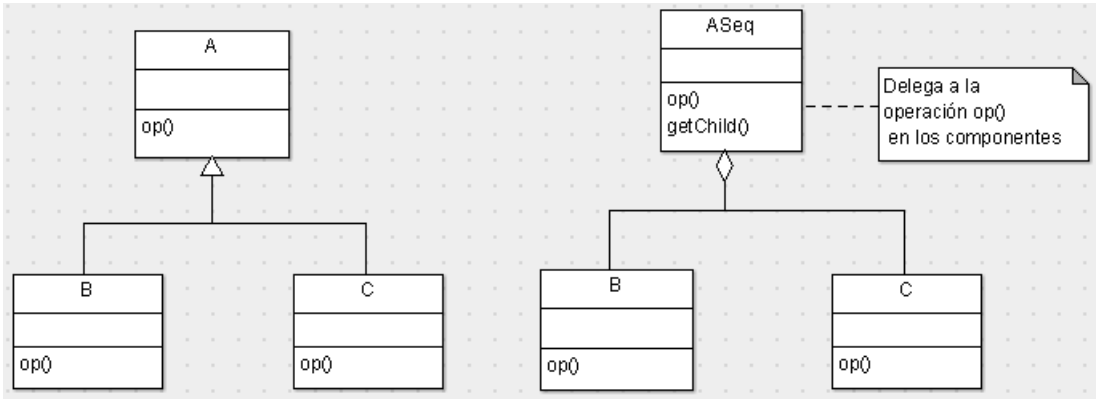


3. ¿Cuál(es) de las siguientes afirmaciones son correctas para el patrón *Singleton*? **[4%]**
   a. Se permite exactamente una instancia de una clase, el *Singleton*.
   b. Los objetos necesitan un único punto global de acceso.
   c. Define un método protegido "*getInstance*()" en la clase que retorna el *Singleton*.
   d. Todas son correctas.
   e. Ninguna de las anteriores.

4. ¿Cuál(es) de las siguientes afirmaciones son correctas para el Paradigma de Orientación a Aspectos (OAP)? **[4%]**
   a. La instancia de un aspecto es reutilizable.
   b. Los atributos de un aspecto se manejan como variables estáticas.
   c. La OAP intenta solucionar el "*weaving*" producido en la OOP.
   d. Un "*advice*" puede ser aplicado a varios métodos utilizando un solo "*pointcut*".
   e. Ninguna de las anteriores.

MSc. David Jurado Mosquera, Dr. Carlos Mera Gómez

5. Considere los siguientes dos diagramas de clases:



5.1 ¿Qué patrón de diseño puede ser útil en este caso y por qué? **[4%]**

5.2 Proponga una solución (diagramas de clase UML) que combine ambos diagramas. **[6%]**

Sección B

6. Se requiere un sistema de reservación de aulas para una Universidad. La descripción del proyecto es la siguiente:

   La Universidad mantiene un conjunto de aulas que son reservables, esencialmente para clases y pequeños grupos de enseñanza asociados con los módulos (como seminarios, talleres, etc.), es decir, eventos recurrentes, pero también para eventos y reuniones de una sola ocurrencia. Aulas individuales tienen una capacidad y equipamiento asociado con ellas, por ejemplo: retroproyectores, proyectores láser, pizarras, equipamiento para videoconferencia, etc.; el número y tipo de cada equipo debe ser almacenado y estar disponible de tal manera que las búsquedas de aulas apropiadas puedan incluir condiciones de que el aula tenga el equipamiento requerido. Únicamente empleados pueden solicitar la reserva de un aula y el sistema debe ser capaz de contactar a quien reservó el aula en caso de que una reprogramación sea necesaria. Una reserva de aula siempre especifica el día, hora y duración para la cual se hace la reserva y podría incluir un módulo asociado con ella. Las reservas de aulas asociadas con módulos son permitidas únicamente si ningún estudiante registrado en el módulo asociado tiene un cruce de horarios, lo cual se refiere a que un estudiante esté registrado en dos módulos diferentes que tengan reservas para aulas distintas en el mismo día y hora.

   a. Elabore un **diagrama de clases** para describir su diseño. Especifique multiplicidades, relaciones, visibilidad de métodos y atributos. Indique cualquier asunción que realice. (La puntuación será dividida igualmente entre sus clases, las relaciones entre clases y la multiplicidad en esas relaciones) **[30%]**
   b. Una de las operaciones que el sistema debe soportar es que los empleados puedan reservar aulas para un módulo. Un empleado especifica un conjunto de equipamiento requerido; un número de clases requeridas por semana; y fechas de inicio y fin para el módulo. Escriba la **especificación de un caso de uso** para soportar este tipo de reservas. **[10%]**
   c. Escriba **tres pruebas unitarias** para el caso de uso que usted especificó como parte de su solución en el literal b). **[10%]**

Sección C

7. Dado el siguiente código, identifique los **principios SOLID que se está violando, luego indique cuales son los malos olores de programación que tiene el código**, explique la razón y refactorice el código para mejorar la calidad y legibilidad de dicho código. Puede crear las interfaces y clases que considere necesarias para la refactorización. **SOLID [5%], Code Smells [5%] y Refactorización [10%]**

```
 * Simulate a game of Rock, Paper, Scissors
 */
public class RPMGame {
    public static void main(String args[]) {
        Player p1 = new Player();
        Player p2 = new Player();
        boolean gameWon = false;
        int roundsPlayed = 0;    // Number of rounds played
        int p1Wins = p1.wins;
        int p2Wins = p2.wins;
        int draw = 0;
        String p1Choice;
        String p2Choice;
        // Game Loop
        do {
            System.out.println("***** Round: " +
                roundsPlayed + " ********************\n");
            System.out.println("Number of Draws: "+
                draw + "\n");
            p1Choice = p1.playerChoice();
            System.out.println("Player 1: " + p1Choice +
                "\t Player 1 Total Wins: " + p1Wins);
            p2Choice = p2.playerChoice();
            System.out.println("Player 2: " + p2Choice+
                "\t Player 2 Total Wins: " + p2Wins);
            if((p1Choice.equals("rock"))&&(p2Choice.equals("paper"))) {
                System.out.println("Player 2 Wins");
                p2Wins++;  // trying a couple different ways to get count to work
            } else if((p1Choice.equals("paper"))&&(p2Choice.equals("rock"))) {
                p1Wins++;
                System.out.println("Player 1 Wins");
            } else if((p1Choice.equals("rock"))&&(p2Choice.equals("scissors"))) {
                p1Wins = p1.setWins();
                System.out.println("Player 1 Wins");
            } else if((p1Choice.equals("scissors"))&&(p2Choice.equals("rock"))) {
                p2Wins = p2.setWins();
                System.out.println("Player 2 Wins");
            } else if((p1Choice.equals("scissors"))&&(p2Choice.equals("paper"))) {
                p1Wins = p1.setWins();
                System.out.println("Player 1 Wins");
            } else if((p1Choice.equals("paper"))&&(p2Choice.equals("scissors"))) {
                p2Wins = p2.setWins();
                System.out.println("Player 2 Wins");
            }
            if(p1Choice==p2Choice) {
                draw++;
                System.out.println("\n\t\t\t Draw \n");
            }
            roundsPlayed++;
            if((p1.getWins()>=3)| |(p2.getWins()>=3)) {
                gameWon = true;
                System.out.println("GAME WON");
            }
            System.out.println();
        } while(gameWon != true);
    }
}
```

```
class Player{
    int wins;       // # of wins
    int winTotal;
    /**
     * Randomly choose rock, paper, or scissors
     */
    public String playerChoice(){
        String choice = "";
        int c = (int)(Math.random()*3);
        switch(c) {
            case 0:
                choice = ("rock");
                break;
            case 1:
                choice = ("paper");
                break;
            case 2:
                choice = ("scissors");
                break;
        }
        return choice;
    }
    public int setWins() {
        int winTotal = wins++;
        return winTotal;
    }
    public int getWins() {
        return(wins);
    }
}
```