Facultad de Matemática, Astronomía y Física
Universidad Nacional de Córdoba
Asignatura: Ingeniería del Software I
Segundo Cuatrimestre de 2011

Examen Parcial No. 2

Parte teórica

Ej. 1. ¿Cuáles son las diferencias esenciales entre el *análisis orientado a objetos* y el *diseño orientado a objetos*? ¿Es posible, en su opinión, distinguir claramente una actividad de la otra? Justifique.

Ej. 2. Uno de los pasos definidos en la metodología de diseño orientada a objetos (OMT), mencionada en "An Integrated Approach to Software Engineering", es *Optimizar y Empaquetar (optimize and package)*. Describa en qu consiste este paso y las razones por las cuales es considerado importante en el proceso de diseño.

Ej. 3. ¿Qué expresa el *Principio de Sustitución de Liskov* y cuál es su relación el diseño orientado a objetos y la herencia?

Ej. 4. Describa brevemente la métrica denominada *Profundidad del árbol de herencia* (*Depth of Inheritance Tree*, DIT). ¿De qué manera puede interpretarse este valor, y cuál es su importancia en la evaluación de un diseño?

Ej. 5. ¿Qué es un criterio de testing y cuál es su finalidad?

Ej. 6. Describa sintéticamente en qué consiste mutation testing.

Parte práctica

Ej. 7. Considere la siguiente clase parcialmente implementada:

```
public class EstadoTaTeTi {

    public int[][] tablero; // 0: casilla vacia
                            // 1: casilla con una X
                            // -1: casilla con un 0
    public int turno; // 1 o -1 de acuerdo a si el turno es de
                      // X o 0, respectivamente.

    // Constructor de la clase
    public EstadoTaTeTi() {
        tablero = new int[3][3];
        turno = 1;
    }

    // retorna true ssi el estado es valido para el juego.
    public boolean repOK() {
        if (!(turno==-1 || turno==1)) return false;
        for (int i=0; i<3; i++) {
```

```
            for (int j=0; j<3; j++) {
                if (tablero[i][j]<-1 || tablero[i][j]>1)
                        return false;
            }
        }
        return true;
    }
    ...
}
```

La intención de esta clase es representar el estado del juego *TaTeTi*, como se indica en los comentarios dentro de la clase. ¿Tiene esta clase o alguno de sus métodos algún problema de diseño o codificación? Describa cuáles, qué concepto de diseño afecta (acoplamiento, cohesión, etc.) y cómo lo resolvería.

Ej. 8. Considere el siguiente fragmento de código:

```
public class InterfaceNames {
    public static final int noDeJugadores = 4;

    public static final int duracionTurno = 60;

    public static final String[] jugadores = {"John","Paul","George","Ringo"};

    ...
}

public class EstadoJuego extends InterfaceNames {

    ...
    public void setUp() {
        ...
        for (int i=0; i<noDeJugadores; i++) {
            ...
            mensaje = "puntos de " + jugadores[i] +": " + puntos[i];
            ...
        }
        ...
    }
    ...
}
```

En este ejemplo, se aprovecha herencia para poder mantener en una ubicación particular (la clase `InterfaceNames`) todos los parámetros de juego, nombres a ser usados en la interfaz, etc., en lugar de tener estos valores diseminados en la aplicación. Todas las clases que necesiten estos valores, como en el caso de `EstadoJuego`, deben heredar de `InterfaceNames` para poder acceder a los mismos.

¿Considera que este uso de herencia es correcto? De ser así, justifique. Si considera que no es correcto, indique cuáles son los conceptos o principios de diseño que no se respetan, y proponga un diseño alternativo para resolverlos.

Ej. 9. El método `repOK()` de la clase `TaTeTi` del ejercicio 7 tiene un único parámetro implícito, el objeto al cual se aplica. Se desea testear este método, para poder lograr cobertura de ramas. Dé diferentes objetos (tablero+turno) de manera tal de satisfacer este criterio de cobertura.