Programación orientada a objetos. Primer parcial 2004.

Viernes 23 de abril de 2004.

Pongan el nombre sólo en la primera hoja. Contesten las preguntas en orden en que están formuladas agrupando las respuestas de las preguntas de la 1 a la 5 y de la 6 a la 9 porque serán corregidas por diferentes profesores. Escriban sólo del anverso de la hoja y eviten doblarlas. Gracias y mucha suerte.

1. Define **tipo**. ¿Cómo puedes definir tipos en Java?

   5 puntos.

2. En la descomposición orientada a objetos vemos el mundo como una colección de agentes autónomos que colaboran para exhibir algún nivel más elaborado de comportamiento. La colaboración aparece cuando un objeto cliente solicita un servicio de otro objeto servidor. Un contrato establece las obligaciones entre un cliente y un servidor.

   Explica en qué situación el cliente viola el contrato y en cuál el servidor viola el contrato usando los conceptos de **clase**, **mensaje**, **método**, **operación** y **tipo**.

   10 puntos.

3. Cuando se formaliza un contrato entre un objeto cliente y otro objeto servidor usando un tipo:

   a. ¿El cliente o el servidor debe tener el tipo?

   b. ¿Qué significa que el cliente o el servidor tiene el tipo?

   Justifica tus respuestas.

   10 puntos.

4. En un programa orientado a objetos los datos y la lógica están distribuidos en forma razonablemente equilibrada entre todos y cada uno de los objetos. ¿Compartes esta afirmación? ¿Qué ocurre cuando esto no sucede? Justifica tus respuestas.

   10 puntos.

5. Sean las siguientes clases e interfaz:

```
/* En algún lugar de java.util */
public interface Iterator {
  /* Es una operación opcional que no es soportada por el iterador
     retornado por el método iterator() de la clase Grupo por lo cual
     debe levantar una excepción UnsupportedOperationException.*/
  public void remove();

  /* Retorna true si la iteración tiene más elementos. En otras palabras,
     retorna true si next() retornará un elemento en lugar de levantar
     una excepción.*/
  public boolean hasNext();

  /* Retorna el próximo elemento en la iteración. Levanta una excepción
     NoSuchElementException si no hay más elementos sobre los cuales
     Iterar.*/
  public Object next();
}

/* Figura.java*/
public class Figura {
}
```

```
/* Grupo.java*/
import java.util.*;

public class Grupo {
  private Set m_figuras;
  private Set m_grupos;

  /* Crea un nuevo Grupo. Los grupos tienen por lo menos una figura. */
  public Grupo(Figura figura) {
    m_figuras = new HashSet();
    m_grupos = new HashSet();
    m_figuras.add(figura);
  }

  /* Agrega un Grupo a este grupo. */
  public void addGrupo(Grupo grupo) {
    m_grupos.add(grupo);
  }

  /* Agrega una Figura a este grupo. */
  public void addFigura(Figura figura) {
    m_figuras.add(figura);
  }

  /* Retorna un Iterator sobre las figuras de este grupo y las figuras de
     los grupos contenidos en este grupo. */
  public Iterator iterator() {
    return new GrupoIterator(this);
  }

  /* Retorna un Iterator sobre las figuras de este grupo. */
  public Iterator getFiguras() {
    return m_figuras.iterator();
  }

  /* Retorna un Iterator sobre los grupos de este grupo. */
  public Iterator getGrupos() {
    return m_grupos.iterator();
  }
}
```

Sea también el siguiente caso de prueba:

```
/* GrupoTest.java*/
import java.util.*;
import junit.framework.TestCase;

public class GrupoTest extends TestCase {

  Figura f1;
  Figura f2;
  Figura f3;
  Figura f4;
  Grupo g1;
  Grupo g2;
  Grupo g3;

  protected void setUp() throws Exception {
    super.setUp();
    super.setUp();
    f1 = new Figura();
    f2 = new Figura();
    f3 = new Figura();
    f4 = new Figura();
    g1 = new Grupo(f1);
    g2 = new Grupo(f3);
    g3 = new Grupo(f4);
    g1.addFigura(f2);
```

```
    g1.addGrupo(g2);
    g2.addGrupo(g3);
  }

  public void testIterator() {
    Set figuras = new HashSet();
    for (Iterator i = g1.iterator(); i.hasNext();) {
      figuras.add(i.next());
    }
    assertTrue(figuras.contains(f1));
    assertTrue(figuras.contains(f2));
    assertTrue(figuras.contains(f3));
    assertTrue(figuras.contains(f4));
    assertEquals(figuras.size(), 4);
  }
}
```

El código provisto está incompleto, razón por la cual no compila. Agrega lo que sea necesario para que compile y para que pase el caso de prueba.

20 puntos.

6. Dadas las siguientes clases e interfaces:

```
public interface Poligono {
  float getArea();
}

public interface IRectangulo extends Poligono {
  float getAncho();
  float getLargo();
}

public interface ICuadrado extends IRectangulo {
  float getLado();
}

public class Rectangulo implements IRectangulo {
  …
}

public class Cuadrado implements ICuadrado {
  …
}

public class Sumador {
  public float sumarArea (Poligono p, ICuadrado c) {
    return p.getArea() + c.getArea();
  }
}
```

Y las siguientes variables:

```
Poligono poli;
ICuadrado cuad;
IRectangulo rect;
Rectangulo rect2;
Cuadrado cuad2;
Sumador sum = new Sumador();
```

Define si las siguientes líneas de código son correctas. En caso contrario, explica la razón. Si es posible arreglar una línea incorrecta, agregando algo pero no cambiando ni quitando, hazlo.

1. `cuad = new Cuadrado();`
2. `poli = new Poligono ();`
3. `rect = new Cuadrado();`
4. `cuad2 = new Rectangulo();`

5. `rect2 = new Cuadrado();`
6. `cuad.getLargo();`
7. `rect.getAncho();`
8. `rect.getLado();`
9. `poli = cuad;`
10. `cuad2 = rect2;`
11. `sum.sumarArea(cuad, rect2);`
12. `sum.sumarArea(rect2, cuad);`

20 puntos.

7. ¿Cuantos tipos tiene el objeto referenciado por la variable `cuad` (sin contar `Object`) y cuáles son?

10 puntos.

8. ¿Es suficiente el uso de interfaces en Java para programar especificando completamente los contratos? ¿Por qué?

10 puntos.

9. Dadas las siguientes clases:

```
public class Impresora {
  public void imprimir (Letra l) {
    pintar(l.getContorno());
  }
  …
}

public class Pantalla {
  public void mostrar (Cuadrado c) {
    dibujar(c.getContorno());
  }
  …
}

public class Letra {
  public Contorno getContorno() { (...) }
  …
}

public class Cuadrado {
  public Contorno getContorno() { (...) }
  …
}
```

Permite que `Impresora` imprima `Cuadrados` y `Pantallas` muestre `Letras`. Modifica el código como desees.

20 puntos.