Primer Certamen

Primera parte, sin apuntes (30 minutos; 1/3 de la nota):

1.- Responda brevemente y entregue en hoja con su nombre.

a. Al redefinir un método en una subclase, ¿es posible retornar un objeto de distinta clase al retornado en el método de la super clase? Explique.
Sí es posible. Se debe cumplir que la clase del objeto retornado por la redefinición debe ser subclase de la clase del objeto retornado por el método redefinido.

b. ¿Cuándo se usa la sentencia import javax.swing.JOptionPane;? ¿Podríamos omitir este "import"?
La sentencia import javax.swing.JOptionPane; se usa cuando deseamos crear instancias de la clase JOptionPane, como en:
JOptionPane obj=new JOptionPane();
Sí se puede omitir, pero cada vez que deseemos crear una instancia de esa clase debemos usar la especificación completa como en:
javax.swing.JOptionPane obj=new javax.swing.JOptionPane();

c. Alguien dice: un cuadrado es un rectángulo de lados iguales. Al ver la forma es-un, piensa en herencia. Dé dos razones por las que no es bueno definir la clase Cuadrado como heredando de Rectángulo.
i) Al hacerlo estaríamos usando más memoria de la necesaria para almacenar cuadrados.
ii) Habría problemas al aplicar el principio de sustitución. Si un método espera un rectángulo para crear la portada de un libro, el método no podrá definir un alto distinto de un ancho si el objeto fuera instancia de cuadrado.

d. La expresión "hola"=="hola", ¿es verdadera o falsa? Explique.
Es verdadera. Los strings son objetos en Java, pero por tratarse de strings literales, es decir secuencia de caracteres entre "", éstos quedan internos en la clase String y el compilador asocia una única referencia a iguales contenidos. Ver: http://profesores.elo.utfsm.cl/~agv/elo329/1s09/lectures/ComparacionStrings.html

e. Si las técnicas de orientación a objeto pueden dar solución a los mismos problemas que los lenguajes no orientados a objetos, ¿Dé dos razones para su uso común en los lenguajes actuales?
i) La orientación a objeto facilita la reutilización de código.
ii) La orientación a objeto facilita el trabajo en equipo.

f. Juan instaló el JDK de java, Pedro instaló JRE. Mencione dos cosas que puede hacer Juan pero no Pedro.
i) Juan puede compilar programas con javac.
ii) Juan puede extraer la documentación de un programa con javadoc.

g. La clase B es derivada de A. En B redefinimos el método público getEdad() de A. ¿Cómo podemos invocar getEdad() implementado por A desde otro método en B? ¿Cómo desde este método en B podemos invocar getEdad() implementado en B?
i) Para invocar implementación de A, en el método de B pongo:
super.getEdad(); // con esto invocamos la implementación de A.
ii) Para invocar implementación de B, pongo:
getEdad(); // Así invocamos la implementación en B desde un método de B.

h. ¿Es posible impedir que un método sea redefinido en una clase derivada? Justifique.
Sí es posible, basta poner agregar el calificador final al método, como en:

```
class A {
  public final int getEdad() {....}
....}
```

Segunda Parte, con apuntes (60 minutos, 2/3 de la nota. 1/3 + 1/3):

2.a) En clase MiTarea implemente un método estático para calcular ~~calcular~~ n!/m!, suponga $n \geqslant m$.
b) Extienda su implementación previa para lanzar una excepción cuando el método sea invocado con n < m.
c) Haga un programa Java que pida dos enteros n y m por consola y usando su implementación de b) retorne el valor de n!/m!. Si el usuario ingresa n<m, el programa imprime por pantalla "Detecté un problema que impide hacer su cálculo." y termina.

```
a)
public class p2a {
  public static long nFacSobreMFac(int n, int m){
      long result = 1;
      for (int i=n; i>m; i--)
        result*=i;
      return result;
    }
}

b)
public class p2b {
  public static long nFacSobreMFac(int n, int m) throws Exception {
      long result = 1;
      if ( n < m ) throw new Exception ("n < m");  // podría ser más elaborado creando otra clase
      for (int i=n; i>m; i--)
        result*=i;
      return result;
    }
}

c)
import java.util.Scanner;

class p2b {
  public static long nFacSobreMFac(int n, int m) throws Exception {
      long result = 1;
      if ( n < m ) throw new Exception ("n < m");
      for (int i=n; i>m; i--)
        result*=i;
      return result;
    }
}

public class p2c {
  public static void main (String[] argv) {
    Scanner in = new Scanner(System.in);

    System.out.print("n = ");
    int n = in.nextInt();
    System.out.print("m = ");
    int m = in.nextInt();
```

```
    try {
      System.out.println("n!/m! = "+p2b.nFacSobreMFac(n,m));
    }catch(Exception e){
      System.out.println("Detecté un problema que impide hacer su cálculo.");
      System.exit(-1);
    }
  }
}
```

3.
a) Para un juego se le pide crear dados de N caras. Estos dados pueden tomar valores dentro del conjunto {1,2,3,..., N}. Cree la clase Dado. Ésta permite definir el número de caras en su constructor. Si no se señala, se asume 6 caras. Estos dados deben permitir ser lanzados y en cualquier momento se puede consultar el valor del último lanzamiento del dado.
b) Desarrolle Juego, una aplicación Java que usa la clase definida en a) y muestra una interfaz gráfica con un botón, el cual lanza un dado al ser presionado, y un área para mostrar el resultado del lanzamiento. El número de caras es ingresado por consola vía un argumento en la línea de ejecución de esta aplicación.
c) Señale qué cambio debe hacer a su clase Dado para implementar la interfaz Comparable basado en el último lanzamiento de los dados.

```
a)
public class Dado {
  private final int numCaras;
  private int valor;

  public Dado(){
    this(6);
  }
  public Dado(int caras){
    numCaras=caras;
    lanzar();
  }
  public int lanzar(){
    valor=(int)(Math.random()*numCaras) +1;
    return valor;
  }
  public int getValor(){
    return valor;
  }
}

b)
import java.awt.event.*;
import javax.swing.*;

public class Juego {
  public static void main(String argv[]){
    JFrame f= new JuegoFrame(Integer.parseInt(argv[0]));
    f.setVisible(true);
  }
```

```
}
class JuegoFrame extends JFrame {
  public JuegoFrame(int caras){
    addWindowListener( new WindowAdapter() { /* Clase anonima */
        public void windowClosing( WindowEvent e) {
          System. exit( 0);
        }
      });
    getContentPane().add(new JuegoPanel(caras));
    pack();
  }
}
class JuegoPanel extends JPanel{
  private Dado dado;
  private JLabel label;

  public JuegoPanel (int caras){
      dado = new Dado(caras);
      JButton boton = new JButton("Lanzar Dado");
      add(boton);
      label = new JLabel(" "+dado.getValor());
      add(label);
      boton.addActionListener( new ActionListener (){
       public void actionPerformed(ActionEvent event) {
         dado.lanzar();
         label.setText(Integer.toString(dado.getValor()));
       };
      });
    }
}

c)
public class DadoComparable implements Comparable {  // cambia
  private final int numCaras;
  private int valor;

  public DadoComparable(){
    this(6);
  }
  public DadoComparable(int caras){
    numCaras=caras;
    lanzar();
  }
  public int compareTo(Object obj){                  // debe ser agregado
    DadoComparable d=(DadoComparable)obj;
    return (valor-d.valor);
  }
  public int lanzar(){
    valor=(int)(Math.random()*numCaras) +1;
    return valor;
  }
  public int getValor(){
```

```
    return valor;
  }
}
```