Primer Certamen

En este certamen usted no podrá hacer preguntas. Si algo no está claro, indíquelo en su respesta, haga usted una suposición razonable y resuelva conforme a ello.

Primera parte, sin apuntes (32 minutos; 32 puntos):

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

a) Explique por qué se recomienda reemplazar los "tabs" por espacios para indentar código fuente.
Así la indentación se visualiza de igual forma en distintos editores y sus versiones impresas. El tamaño de cada tab es definido por el entorno, luego lo que se ve indentado en uno puede no estarlo en otro.

b) Nombre y explique tres características que comparten los lenguajes orientados a objetos.
Herencia: la posibilidad de definir una clase o tipo de objeto a partir de la definición de otra.
Subtipo: la posibilidad de usar instancias de clases derivadas donde se espera una instancia de la clase base.
Ligado dinámico: el código invocado ante un llamado es definido en tiempo de ejecución permitiendo así que el resultado dependa del objeto específico que recibe el llamado. En conjunto con subtipo, permite generar códigos reusables.

c) La clase Timer está definida en java.util.Timer y en javax.swing.Timer. ¿Qué definición de import debe hacer en su código para usar Timer del paquete java.util?
import java.util.Timer;
¿Cómo lo hace si además de esta clase, en su programa usted requiere crear instancias de Timer el paquete javax.swing?
Como antes al inicio uso:
import java.util.Timer;
Luego cada vez que desee usar una instancia de Timer de javax.swing, uso la versión completa para la clase, esto es:
javaxswing.Timer miTiner; /* así cada vez que requiero referirme a instancias de esta clase*/

d) La clase C hereda de B, y B hereda de A. A posee un método protegido getPeso(), el cual no es redefinido por B. ¿Es posible redefinir este método en C?
Sí. Los métodos protegidos dan visibilidad a las clases derivadas.
Si su respuesta es afirmativa, ¿cómo podemos llamar la implementación de A de getPeso() desde algún método de C? O
La invocación la haríamos con super.getPeso();
Esto funciona pues la versión de getPeso() visible en la clase B es justamente aquella definida en A.
Si su respuesta es negativa, justifique.

e) Alguien dice: "Sea ***emp*** una instancia de Empleado, al asignar ***emp=null;*** en todos los casos el espacio de memoria referenciado por ***emp*** será recolectado por el recolector de basura de java. ¿Está de cuerdo? Explique.
No, debido a que podemos tener más de un nombre para un objeto; es decir, pude haber más de una referencia apuntando a un objeto. El recolector de basura sólo recupera el espacio de memoria cuando no hay referencias a ese espacio.

f) ¿Qué imprime el siguiente programa?

```
public class A{
 public int a;
 public float b=2.0f;
 public static void main(String args[]){
     A a = new A();
    System.out.println(a.a, a.b);
```

```
 }
 public A(){
  System.out.println(a, b);
   b=3.0f;
 }
}
```

Hay un error en esta pregunta, en lugar de coma debió ir +. Se considerarán 4 puntos base debido a este error.

Primero al crear el objeto **a** instancia de A, sus atributos adoptan los valores a=0, y b=2.0, luego se ejecutan las instrucciones del constructor, por lo cual se imprime:

2.0

y b pasa a ser 3.0. Luego se ejecuta el la salida a consola del método main imprimiendo:

3.0

g) Alguien dice: "Si en un archivo tenemos definidas dos clases A y B, sólo una de ellas puede contener el método public static void main (String argv[]) {….. }" ¿Está usted de acuerdo? Justifique.

No estoy de acuerdo. El método main puede estar en más de una clase. Aquella que use al correr el programa definirá qué método main es usado.

h) Alguien dice: Si B hereda de A y hemos declarado:

B b;

entonces será un error de compilación si en alguna parte del programa hacemos:

A a= (A) b;

¿Está usted de acuerdo? Justifique.

No estoy de acuerdo, pues eso es perfectamente posible. b hereda de A, luego por subtipo puede tomar el lugar donde se espera una instancia de A. Incluso es válido poner:

B b2 = (B) a;

debido a que a bien podría ser referencia a instancias de B, en cuyo caso el casteo es posible.

Segunda Parte, con apuntes (68 minutos)

2.- (24 puntos) En la documentación de la clase Math se tiene:

static double random()	Returns a double value with a positive sign, greater than or equal to 0.0 and less than 1.0.
static long round(double a)	Returns the closest long to the argument.

Haga un programa que pida por consola hacer la suma de dos valores aleatorios positivos menor o igual a 100, luego lea el resultado ingresado por el usuario dentro de 10 segundos y envía un mensaje:
"Resultado correcto, dentro del tiempo"
"Resultado incorrecto, dentro del tiempo"
Si a los 10 segundos el usuario no ha respondido, se muestra:
"Tiempo excedido"
El programa termita luego de mostrar el mensaje que corresponda.

```
import java.util.Scanner;
import javax.swing.Timer;
import java.awt.event.*;
public class SumaConTiempo    // 4 pts por definir clase y su main
{
  public static void main(String args[])
  {
   Scanner s = new Scanner (System.in); // 4 pts
   long a = Math.round(Math.random()*101);  // usar 100 es OK
   long b = Math.round(Math.random()*101); // 4 pts
   Timer timer = new Timer(10000, new ActionListener() {  // 6 pts mecanismo
                                        // para abortar en 10 [s].
        public void actionPerformed(ActionEvent event)
        {
          System.out.print("Tiempo excedido.");
          System.exit(0);
        }
      });
   timer.start(); // 2 pts  mecanismo para bortar en 10 [s]
   System.out.print(a + " + " + b + " = ");
  int respuesta = s.nextInt();
   if (respuesta == a+b) // 4 pts por if completo
     System.out.print("Respuesta correcta dentro del tiempo.");
   else
     System.out.print("Respuesta errada dentro del tiempo.");
  }
}
```

3.- (26 puntos) Considere las clases de las hojas adjuntas. Cree la clase Vector2D que permita una compilación adecuada con las clases dadas. La semántica de las operaciones debe permitir el cálculo de la dinámica de bloques conectados a uno o dos resortes. El propósito de la clase Vector2D pedida es el mismo de la tarea 1.
Al revisar el código adjunto se puede identificar los siguientes métodos para la clase Vector2D.
De Block.java
force = force.plus(springA.getForce(this)); => public Vector2D plus(Vector2D)
Vector2D a = force.times(1/mass); => public Vector2D times(float)
speed_tPlusDelta = speed_t.plus(a.times(delta_t)); => public Vector2D plus(Vector2D) y public Vector2D times(double)
getPosition().getDescription() => public String getDescription()
return getPosition()+","; => public String toString()
De Spring.java
new Vector2D(0,restLength); => public Vector2D (float, float)
return b.minus(a); => public Vector2D minus(Vector2D)
double stretch = springVector.module() => public double module()
force = springVector.unitary() => public Vector2D unitary()
return new Vector2D(); => public Vector2D ()
Luego:

```
public class Vector2D {    //3 pts
  private double x, y;  // we will use cartesian coordinates    // 2 pts
  public Vector2D () {    // ambos constructores 3 pts
    x = y = 0;
  }
  public Vector2D (float x, float y) {
    this.x = x;      this.y = y;
  }
  public Vector2D plus(Vector2D v) {              // 3 pts
     if (v==null)   return new Vector2D(x,y);
     else              return new Vector2D(x+v.x, y+v.y);
  }
  public Vector2D times(double scalar) {  // include times (float)
    return new Vector2D(x*scalar, y*scalar);
  }
  public Vector2D minus(Vector2D v) {    // 3 pts
     if (v==null)   return new Vector2D (x,y);
     else              return new Vector2D(x-v.x, y-v.y);
  }
  public double module() {    // 3 pts
    return Math.sqrt(x*x+y*y);
  }
  public Vector2D unitary() {     // 3 pts
    return times(1/module());
  }
  public static String getDescription() {   // dos siguientes 3 pts
    return "(x,y)";
  }
  public String toString() {
    return x+","+y;
  }
}
```

4.- (18 puntos) Una cuerda puede ser modelada como un resorte con constante elástica muy grande cuando la cuerda tiende a crecer respecto de su largo en reposo y constante elástica cero cuando la cuerda es más corta que su largo en reposo. Así la fuerza ejercida en la dirección lineal de cuerda es:

$$F = \begin{cases} k * \Delta x & \Delta x > 0 \\ 0 & \Delta x \leqslant 0 \end{cases}$$

Implemente la clase Cuerda como clase heredada de Spring y haga los cambios que correspondan para reflejar el comportamiento de cuerdas.

```
public class Cuerda extends Spring  {  // 2pts
  // luego de pasar los atributos privados de Spring a protected   // 2pts
   private Cuerda(){     // ambos constructores 3pts
      super();
   }
   public Cuerda(float restLength, float stiffness){
      super (restLenght, stiffness);
   }
   public Vector2D getForce(Block block) {  // debo redefinir la fuerza   // 2 pts
     Vector2D force;
     Vector2D cuerdaVector = getVector();
     double stretch = springVector.module() - restLength;    // hasta aquí  2pts
     if (stretch < 0) return Vector2D();  // fuerza nula    // 2pts
     esle {
       force = cuerdaVector.unitary().times(stretch*stiffness);   // 4 pts
       if (block == block_a) return force;
       if (block == block_b) return force.times(-1);
       else return new Vector2D();
     }
   }
   public String getDescription() {   // 2 pts.
      return "Cuerda#"+ getId()+": "+Vector2D.getDescription()+
           ","+Vector2D.getDescription()+",";
   }
}
```

==========

```
public abstract class PhysicsElement {
  private int myId;

   protected PhysicsElement( int id){
     myId = id;
   }
   protected int getId() {
     return myId;
   }
   public abstract void computeNextState(double delta_t);
   public abstract void updateState();
   public abstract String getDescription();
   public abstract String getState();
}
```

```
=======================================================
import java.util.*;
public class Block extends PhysicsElement  {
  private static int id=0;
  private final float mass;
  protected Vector2D pos_t, pos_tPlusDelta;
  private Vector2D speed_t, speed_tPlusDelta;
  private Spring springA, springB;

  private Block(){ // nobody can create a block without state
    super(id++);
    mass=0;
  }
  public Block(float mass, Vector2D position, Vector2D speed){
    super(id++);
    this.mass = mass;
    pos_t = position;
    this.speed_t = speed;
    springA = springB = null;
  }
  public void attachSpring (Spring spring) {
   if (springA==null) springA = spring;
   else springB = spring;
  }
  public Vector2D getPosition() {
    return pos_t;
  }
  public void computeNextState(double delta_t) {
   Vector2D force = new Vector2D();
   if (springA != null) force = force.plus(springA.getForce(this));
   if (springB != null) force = force.plus(springB.getForce(this));
   Vector2D a = force.times(1/mass);
   pos_tPlusDelta = pos_t.plus(speed_t.times(delta_t)).plus(a.times(delta_t*delta_t/2));
   speed_tPlusDelta = speed_t.plus(a.times(delta_t));
  }
  public void updateState(){
   pos_t = pos_tPlusDelta;
   speed_t = speed_tPlusDelta;
  }
  public String getDescription() {
    return "Block#" + getId() + ": "+getPosition().getDescription()+",";
  }
  public String getState() {
    return getPosition()+",";
  }
}
=============================================================
public class Spring extends PhysicsElement  {
  private static int id=0;  // Spring identification
  protected float restLength;
  private final float stiffness;
  private Block block_a, block_b;
```

```
  private Spring(){    // nobody can create a block without state
    super(id++);
    stiffness=0;
  }
  public Spring(float restLength, float stiffness){
    super(id++);
    this.restLength = restLength;
    this.stiffness = stiffness;
    block_a = block_b = null;
  }
  public void attachBlock_a (Block a) {
    block_a = a;
    block_a.attachSpring(this);
  }
  public void attachBlock_b (Block b) {
    block_b = b;
    block_b.attachSpring(this);
  }
  protected Vector2D getPositionA() {
    if (block_a != null)
      return block_a.getPosition();
    return new Vector2D(0,restLength);
  }
  protected Vector2D getPositionB() {
    if (block_b != null)
      return block_b.getPosition();
    return new Vector2D(restLength, restLength);
  }
  protected Vector2D getVector() {
    Vector2D b = getPositionB();
    Vector2D a = getPositionA();
    return b.minus(a);
  }
  public Vector2D getForce(Block block) {
   Vector2D force;
   Vector2D springVector = getVector();
   double stretch = springVector.module() - restLength;
   force = springVector.unitary().times(stretch*stiffness);
   if (block == block_a) return force;
   if (block == block_b) return force.times(-1);
   else return new Vector2D();
  }
  public void computeNextState(double delta_t) {
  }
```

```
  public void updateState(){
  }
  public String getDescription() {
    return "Spring#"+ getId()+": "+Vector2D.getDescription()+
         ","+Vector2D.getDescription()+",";
  }
  public String getState() {
    String s;
    if (block_a != null)
      s = block_a.getPosition()+",";
    else if (block_b != null)
          s = block_b.getPosition().minus(new Vector2D(restLength,0))+",";
        else
          s = new Vector2D()+",";  // up to here left end is determined
    if (block_a != null)
      s += block_a.getPosition();
    else if (block_b != null)
          s += block_b.getPosition().plus(new Vector2D(restLength,0));
        else
          s += new Vector2D();
    return s+",";
  }
}
```