Unidad 3
Programación Orientada a Objetos. Programación JAVA. Parte I:

Nota: Parte I (_____/5), Parte II (_____/3.5) , Parte III-WEB (_____/1.5)
Alumno: _____

1. Corregir los errores del siguiente código JAVA. (0.5p):

```
// Obtener el menor divisible por 3 y el mayor divisible por 5 de un vector
public class Ejercicio1 {
   public static void main(String[] args) {
      int vector[];
      int mayor= Integer.MIN_VALUE;
      int menor= Integer.MAX_VALUE;
      BufferedReader in=new BufferedReader(new InputStreamReader(System.in));

      System.out.println("¿.....?");
      int longitud=Integer.valueOf(in.readLine()).intValue();
      vector=new int[longitud];

      for(int i=0 ; i<=vector.length ; i++)
      {
         System.out.println(".....:");
         vector[i] = Integer.valueOf(in.readLine()).intValue();
         if ((vector[i] % 3) == 0) && (vector[i] > menor))
               menor= vector[i];
         if ((vector[i] % 5) == 0) &&  (vector[i] < mayor))
               mayor = vector[i];
      }

      System.out.println("....: "+mayor);
      System.out.println("....: "+menor);
   }
}
```

2. Corregir los errores del siguiente código JAVA (0.5p):

```
import java.io.*;

// Clase que me permita seber si un número dado por el usuario es primo o no

public class XX {

   public static es_primo(int num) {
      boolean encontrado = false;
      while(divisor < num && num % divisor != 0)
      {
         divisor++;
       }
       if(num = divisor)
            encontrado = true;
       else
            encontrado = false;
      return(encontrado);
   }

   public static void main(String[] args) throws IOException{

      BufferedReader in=new BufferedReader(new InputStreamReader(System.in));

      System.out.println("Introduce un numero: ");
      num = Integer.valueOf(in.readLine().trim()).intValue();

      if (es_primo(num) == false)
         System.out.println("El numero "+num+" es primo");
      else
         System.out.println("EL numero "+num+" no es primo");
      }
}
```

3. Queremos modelar una casa con muchas bombillas mediante la clase Iluminación (vector de Bombillas), de forma que cada bombilla se puede encender o apagar individualmente. Para ello tenemos una clase Bombilla con una variable privada que indique si está encendida o apagada, métodos para pagagar y encendar, un método que nos dice si una bombilla concreta está o no encendida y el constructor que al crear/new un objeto Bombilla la sitúa es estado de apagada. Tenemos definida una clase Iluminación que necesita los siguientes métodos (1p – 0.25p/método):

```
class Bombilla
{
private ...; // interruptor
public void enciender () { ... }
public void apagar () { ... }
public boolean encendida () { .}
public Bombilla (){ ... }
}
```

```
class Iluminacion
{
private ...;  // Bombillas de la casa
private ...;  // Número de bombillas de la casa
public void apagar_bombilla() { ... }
public void encender_bombilla() { ... }
public boolean estado_bombilla() { ...}
public int numero_bombillas_encendidas (){ ... }
public numero_bombillas_apagadas (){ ... }
public Iluminacion() { ... }
}
```

4. ¿Qué resuelve este código JAVA? (0.5p):

```
import java.io.*;
public class XXXX {

   public static int x(int num){
      int a = 1;
      for (int b=2 ; b <= num ; b++)  a = a * b;
      return(a);
   }

   public static void main(String[] args) throws IOException{
      BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
      System.out.println("……..: ");
      int num = Integer.valueOf(in.readLine().trim()).intValue();
      System.out.println("…….: "+x(num));
   }
}
```

5. Escriba en la clase Java Gpoligono el siguiente método (0.5p):

Atributos:

- ✓ Números de lados (*int*). int nlados;
- ✓ Vector de (*class* ***Punto***). Punto [] posicionamiento;
- ✓ Color de relleno (*class* ***Rgb***). Rgb color;

Métodos:

- ✓ public gpoligono(){// Constructor por defecto}
- ✓ public gpoligono(int num, Rgb c){/* Mediante este constructor sobrecargado se proporciona el número de lados y su color. Dentro del cuerpo de este método te preguntará por cada uno de los puntos (X,Y) que formarán su posicionamiento en el espacio.*/}
- ✓ public int getlados(){// Método que devuelve los lados del Gpoligono.}
- ✓ public Rgb getcolor(){// Método que devuelve el color del Gpolígono.}
- ✓ public boolean compareTopoligono(Gpoligono a){/* Método que dado un polígono determina si es igual a él. Son iguales cuando coinciden el color, número de lados y su posicionamiento. */}

Nuevo Método:

- ✓ public Gpoligono mayorTopoligonos(Gpoligono [] a){/* Método que dado un vector de polígonos devuelva el polígono con mayor número de lados. */}

```
public Gpoligono mayorTopoligonos(Gpoligono [] a) {




} // Fin del new método
```

6. Escriba los nuevos métodos de la clase Java que represente un Viaje(0.5):

Atributos:

- ✓ Números de trayectos (int). int ntra;
- ✓ Vector de (class Trayecto). Trayecto [] sectrayectos;

Clase Trayectoria

```
public class Trayecto {
        public String origen;
        public String destino;
        public double distancia;
        .....}
```

Métodos:

- ✓ public viaje(){// Constructor por defecto}
- ✓ public viaje(int num){/* Mediante este constructor sobrecargado se proporciona el número de trayectos. Dentro del cuerpo de este método te preguntará cada uno de los trayectos que forman el viaje.*/}
- ✓ public Trayecto mayortrayecto()){/* Devuelve el mayor trayecto en longitug.*/}
- ✓ public double distanciaviaje(){// Resultado de la suma de los trayectos}
- ✓ public boolean combinables(Viaje v1) {/*Si coincide el destino de último trayecto de v1 con el origen del primer trayecto de this, los trayectos son combinables true*/}.

Nuevo Método:

- ✓ public double distancia_combinable(Viaje a){/* Método que dado un viaje devuelve la distancia de la combinación del mismo con this si así fuera posible sino devuelve -1 . */}

```
public double distancia_combinable(Viaje a)){




} // Fin del new método
```

7. Escriba los nuevos métodos de la clase Java que represente un ArrayAsir:

Clase ArrayAsir
/ clase que moldea y contiene las operaciones relacionadas con un vector de enteros (int v[]) */*

```
public class ArrayAsir{
        int [] elementos;
        int num_elemente;
        .....}
```

Nuevo Método (0.5p):

- ✓ public boolean igualesTo(ArrayAsir aux);
  /* Método que devuelve true sin son iguales en número de elementos y contienen los mismos valores en las mismas posiciones. */

```
public boolean igualesTo(ArrayAsir aux); {




} // Fin del new método
```

8. Cuestionario tipo test (1p)(Las cuestiones erróneas no quitan cuestiones correstas):
   - Dpto. Ingeniería de Sistemas Telemáticos Universidad Politécnica de Madrid -
   http://www.lab.dit.upm.es/~fprg/examenes/

1) ¿ Qué ocurre cuando se compila y ejecuta la siguiente clase ?

```
class MiClase {
public static void main (String[] args) {
String s1[] = new String[5];
String str = s1[0].toUpperCase();
System.out.println(str);
}
}
```

   a. Imprime NULL.
   b. Da un error al compilar.
   c. Imprime null.
   d. Da una excepción NullPointerException al ejecutar.

2) Qué se escribe por pantalla con la siguiente línea de código: System.out.println ((int) Math.PI);

   a. 3.1415926
   b. 3
   c. nada, hay un error al compilar.
   d. nada, hay un error durante la ejecución.

3) Dada la siguiente definición: ¿Cuál debería ser el contenido del constructor?

```
class Temperatura {
double t;
Temperatura(double t) { ¿? }
}
```

   a. t=t;
   b. double t=t;
   c. this.t=t;
   d. No se puede llamar igual el parámetro del constructor que el atributo de la clase

4) "for (x= 5; x <100; x*= 2) { cosas; }" es equivalente a ...

   a. x= 5; do { cosas; x*= 2; } while (x <100);
   b. x= 5; while (x <100) { cosas; x*= 2; }

5) 2. Dado el siguiente fragmento de programa: Indique que afirmación es cierta:

```
int k;
for (k=5 ; k>0 ; k--)
System.out.print(k);
System.out.print(k);
```

   a. Se imprime 543210
   b. Se imprime 5432100
   c. Se imprime 554433221100
   d. Se imprime 543210-1

6) Indique el elemento que no es obligatorio en una declaración de variable:

   a. La asignación del valor inicial.
   b. El identificador o nombre.
   c. El punto y coma.
   d. El tipo.

7) Indique la salida de:

```
int a= 7, b= 3;
System.out.println ((++a) * b);
```

a. 24
b. 21
c. 10
d. 73

8) Evalúe el valor final que toma la variable "s":

```
int n= 1; s= 0;
while (n <= 9) s+=n;
```

a. 45
b. 0
c. 9
d. el programa no termina nunca

9) ¿ Qué imprime este programa ?

```
int metodo (int v1) { return v1*v1; }
  ……
int v = 3;
System.out.print (metodo (metodo (v))); ...
```

a. Tiene errores que impiden su compilación
b. 9
c. 81
d. Compila; pero tiene errores que impiden su ejecución

10) ¿Qué se imprimirá al ejecutar el siguiente bucle?

```
for (int i=0; i < 5; i++) {
if (i==3) { i=5; }
System.out.println (i + " ");
}
```

a. 0 1 2 3 4
b. 0 1 2 5
c. 0 1 2 4
d. 0 1 2

11) Indique qué ocurre con el siguiente código:

```
int a[] = new int[10];
for (int i=0 ; i<=a.length ; i++)
  System.out.print(a[i]);
```

a. Error de ejecución: índice fuera de rango.
b. Error de compilación: no se asignaron valores al array.
c. Escribe los 10 valores almacenados en a.
d. Escribe 10 ceros.

12) Indique qué escribe una llamada al método m()

```
void m() {
  int x = 0;
  try {
    System.out.print(x++);
    if (x>0) throw new Exception();
    System.out.print(x++);
  } catch (Exception e) {
    System.out.print(x++);
    } finally {
    System.out.print(x++);
  }
}
```

a. 012
b. 0
c. 01
d. 0123

13) Dados los siguientes fragmentos de código:

```
class ClaseC {
public void fmet (int i) { ... }
public int fmet (int i) { ... }
}  ...
ClaseC c = new ClaseC();
c.fmet(4);
```

Se produce:

a. la llamada al primer método fmet.
b. la llamada a ningún método porque hay sobrecarga.
c. un error al compilar.
d. un error al ejecutar.

14) Dado el método "imprime":

```
void imprime (boolean ok) {
 System.out.println ("La respuesta " + (ok?"NO":"") + " está mal");
}
```

¿qué imprime la llamada imprime(false)?

a. La respuesta está mal
b. La respuesta NO está mal
c. Se produce un error de compilación

15) En una sentencia try-catch-finally:

a. Los bloques catch se pueden repetir tantas veces como excepciones de distinto tipo se desee atrapar. El bloque finally debe aparecer al menos una vez y se ejecuta siempre.
b. Los bloques catch se pueden repetir tantas veces como excepciones de distinto tipo se desee atrapar. El bloque finally no es opcional y se ejecuta siempre.
c. Los bloques catch se pueden repetir tantas veces como excepciones de distinto tipo se desee atrapar. El bloque finally es opcional y solo puede aparecer una vez. Este bloque se ejecuta siempre.
d. d) Todas las afirmaciones son falsas

Unidad 3
Programación Orientada a Objetos. Programación JAVA. Parte II

Alumno: _____

Desarrolla un proyecto JAVA llamado nombre_examen_Asir que contenga los siguientes elementos (3.5p):

- Un paquete Ejercicio1_Examen.
- Un paquete Ejercicio2_Examen.
- Una clase Main.java que se utilizará para probar de manera opcional los ejercicios resueltos. Es evidente que el examen estará dividido en dos ejercicios y cada una de ellos se resolverá con una , dos ..etc clases que serán definidas en su correspondiente paquete.

Ejercicios 1 (1.75 puntos).- (Paquete Ejercicio1_Examen)

- Dpto. Ingeniería de Sistemas Telemáticos Universidad Politécnica de Madrid -
  http://www.lab.dit.upm.es/~fprg/examenes/

Se dispone de un dispositivo que es alimentado desde el exterior con instrucciones para que realice determinado tipo de acciones. Se desea modelar el comportamiento de este dispositivo en Java. Para ello, se utilizarán las clases

Instrucción y ColaInstrucciones que se detallan a continuación.

```
public class Instruccion (0.75p){
        private String codigo; // nombre de la instrucción
        public int prioridad; // prioridad de la instrucción
        public long tiempo; // instante en el que se invoca la instrucción
        // Contructor de inicializar cada uno de los atributos de una instrución
        public Instruccion (String codigo, int prioridad, long tiempo) { ... }
       // Devuelve el código/nombre de la insrtucción. Por ejemplo podría devolver "SUM"
        public String getCodigo () {…. }
       // Método que compara dos instrucciones devolviendo true si ambas tienen el mismo
       código/nombre de acción
       public boolean compareToIntruccion (Instrucción aux){…}
}
public class Dispositivo (1p) {
        private Instruccion[] cola; // vector de instrucciones
        private int numInstrucciones; // número de instrucciones en el dispositivo
       // Constructor tamaño inicial de 100
       public Dispositivo () {
            this.cola=new Instruccion[100];
            this.numInstrucciones=-1;          }
       // Constructor con tamaño inicial proporcionado por el usuario
       public Dispositivo (int tama_max) { …};
        // Método que se ejecuta cuando se quiere introducir una instrucción
        public void putInstruccion (Instruccion i) { ... }
       // Método que determina cuantas instrucciones son del nombre/código "XX"
        public int count_codigo (String codigo) { ... }
       // Método que ejecuta una instrucción. La ejecución de la instrucción cola[0] implica simularlo
       mostrando por pantalla las instrucción y a continuación desplazando todas la siguientes una
       posición a la izquierda y actualizando el valor de numInstrucciones. Si no hay instrucciones
       debe aparecer por pantalla "dispositivo vacío"
       public void ejecutarInstruccion(){….}
 }
 // fin clase
```

Ejercicios 2 (1.75 puntos).- (Paquete Ejercicio2_Examen)

Para multar a los conductores que conducen demasiado rápido por las autopistas de peaje, vamos a desarrollar un sistema que calcula la velocidad media a la que han circulado y, si superan la velocidad máxima permitida, el sistema procederá a multarles. Para ello, se tomarán los datos de los coches cuando entran y salen de la autopista. Con estos datos se calculará la velocidad media y se emitirán las multas pertinentes.

```
public class Registro (0.25p) {
        public String matricula; // matrícula del coche
        public double hora; // instante (hora) de entrada o de salida
        public double kilometro; // punto kilométrico de entrada o salida
        public Registro (String matricula, double hora, double kilometro) { ... }
}
```

Desarrollaremos la clase GestorDeMultas que procesa los registros creados a las entrada y salidas de los coches en la autopista, e imprime por pantalla las multas que deban ponerse. Esta clase es la siguiente:

```
public class GestorDeMultas (1.5p){
        // Velocidad media máxima permitida.
        public static final double MAX_VEL = 120;
        private Registro[] entradas;
        private Registro[] salidas;
       private int  numRegistrosentradas;
       private int  numRegistrossalidas;
        // Constructor que inicializa el tamaño de entradas y salidas a 100 posibles registros y el
       valores de numRegistrosentradas y numRegistrossalidas
        public GestorDeMultas() {...}
         // Metodo llamado cuando un coche entra en la autopista guardándose dicho registro.
        public void entrada(Registro reg) {...}
        // Metodo llamado cuando un coche sale de la autopista debe guardarse dicho registro en el
       array correspondiente y debe calcularse la velocidad media entre lesta salida y su
       correspondiente entrada, e imprimir por pantalla una multa si se ha excedido la velocidad
       media máxima permitida. El formato del mensaje de multa debe ser como el del siguiente
       ejemplo:
       Multa: 1234ABC Velocidad 193.54 [(220-100)/(11.12-10.5)] entre 10.5 y 11.12 horas, entre
       Kms 100 y 220
       public void salida(Registro reg) {...}
 }
```

```
//----------------Ejemplo de prueba en el Main() para ejemplo1------------------
public static void main(String [] args)
{
    Dispositivo ejemplo1=new Dispositivo(50);
    ejemplo1.putInstruccion(new Instruccion("SUM",2,100));
    ejemplo1.putInstruccion(new Instruccion("SUM",1,100));
    ejemplo1.putInstruccion(new Instruccion("GOTO",2,300));
    ejemplo1.putInstruccion(new Instruccion("CMP",5,100));
    System.out.println("El número de instrucciones \"SUM\" es:"+ejemplo1.count_codigo("SUM"));
    ejemplo1.ejecutarInstruccion();
    System.out.println("El número de instrucciones \"SUM\" es:"+ejemplo1.count_codigo("SUM"));
    ejemplo1.ejecutarInstruccion();
    ejemplo1.ejecutarInstruccion();
    ejemplo1.ejecutarInstruccion();
    ejemplo1.ejecutarInstruccion();
}
```

Salida:

```
El número de instrucciones "SUM" es: 2
Se ha ejecutado la Instrucción: "SUM" con prioridad 2 y tiempo 100
El número de instrucciones "SUM" es: 1
Se ha ejecutado la Instrucción: "SUM" con prioridad 1 y tiempo 100
Se ha ejecutado la Instrucción: "GOTO" con prioridad 2 y tiempo 300
Se ha ejecutado la Instrucción: "CMP" con prioridad 5 y tiempo 100
Dispositivo vacío
```

```
//----------------Ejemplo de prueba en el Main() para ejemplo2------------------

public static void main(String [] args)
{
    GestorDeMultas ejemplo2=new GestorDeMultas();
    ejemplo2.entrada(new Registro("HSM542",10.5,100));
    ejemplo2.entrada(new Registro("HDF545",11.0,120));
    ejemplo2.entrada(new Registro("ADF567",12,220));
    ejemplo2.salida(new Registro("HSM542",11.12,220));
    ejemplo2.salida(new Registro("HDF545",12.0,300));
    ejemplo2.salida(new Registro("ADF567",13,330));
}
```

Salida:

Multa: HSM542 Velocidad 193.54 [(220-100)/(11.12-10.5)] entre 10.5 y 11.12 horas, entre Kms 100 y 220
Multa: HDF545 Velocidad 200.0 [(320-120)/(12.0-11.0)] entre 11.0 y 12.0 horas, entre Kms 220 y 120