PRUEBAS PARA LA OBTENCIÓN DE TÍTULOS DE TÉCNICO Y TÉCNICO SUPERIOR

Convocatoria correspondiente al curso académico 2020-2021

(RESOLUCIÓN de 12 de enero de 2021, de la Dirección General de Educación Secundaria, Formación Profesional y Régimen Especial)

DATOS DEL ASPIRANTE			CALIFICACIÓN
APELLIDOS:			
Nombre:	D.N.I.:	Fecha: **11-05-2021**	

Código del ciclo: IFCS02	Denominación completa del ciclo formativo: DESARROLLO DE APLICACIONES MULTIPLATAFORMA
Clave del módulo: 11	Denominación completa del módulo profesional: Programación de Servicios y Procesos

INSTRUCCIONES GENERALES PARA LA REALIZACIÓN DE LA PRUEBA

1) Sobre la mesa de examen sólo podrá haber:
   - El examen.
   - Bolígrafo azul o negro
   - DNI.
2) **En ningún caso,** está permitido el **uso de teléfonos móviles, ni ningún otro dispositivo electrónico, deberán estar apagados y guardados**.
3) Durante la realización de la prueba se observarán todas las normas elementales de comportamiento. Todos permanecerán en silencio. Para preguntar o entregar exámenes se levantará la mano.
4) Se debe rellenar los datos del aspirante tanto en esta primera página, como en la hoja de respuestas.
5) La prueba consta de 40 preguntas de respuesta múltiple de las que sólo una de ellas es correcta.
6) Cada pregunta se responderá en el espacio dejado al efecto, en la hoja de respuestas, la hoja 2. Se usarán X en los recuadros para señalar la respuesta seleccionada.
7) Si se quiere rectificar una respuesta contestada, se rellenará toda la casilla de la respuesta incorrecta, tal y como se puede apreciar aquí:

   ☐a ■b ☐c ☒d

   Cualquier tachadura o borrón en una respuesta podrá invalidar toda la puntuación de esta.
8) Se dispondrá de una hoja para borrador (o de varias si se requieren), que será proporcionada por el centro. Esa hoja se entregará obligatoriamente al final junto con el examen, si bien nada de lo escrito en la hoja de borrador se valorará en la corrección.

La duración de la prueba será de **90 minutos**.

CRITERIOS DE CALIFICACIÓN Y VALORACIÓN

1) La prueba se calificará sobre 40 puntos.
2) Las respuestas correctas tienen calificación de 1 punto cada una, las incorrectas restan 0,25 puntos cada una y las respuestas en blanco 0 puntos.

CALIFICACIÓN FINAL DEL MÓDULO PROFESIONAL

La nota del módulo será el resultado de truncar al entero más próximo por debajo la puntuación obtenida en la prueba dividida entre cuatro.

DATOS DEL ASPIRANTE	CALIFICACIÓN
APELLIDOS:	

IES ALONSO DE AVELLANEDA

CONSEJERÍA DE EDUCACIÓN

Comunidad de Madrid

Fondo Social Europeo

Nombre:	D.N.I.:	Fecha: **11-05-2021**		

Código del ciclo: IFCS02	Denominación completa del ciclo formativo: DESARROLLO DE APLICACIONES MULTIPLATAFORMA
Clave del módulo: 11	Denominación completa del módulo profesional: Programación de Servicios y Procesos

RESPUESTAS

1 ☐a ☐b ☐c ☐d	11 ☐a ☐b ☐c ☐d	21 ☐a ☐b ☐c ☐d	31 ☐a ☐b ☐c ☐d
2 ☐a ☐b ☐c ☐d	12 ☐a ☐b ☐c ☐d	22 ☐a ☐b ☐c ☐d	32 ☐a ☐b ☐c ☐d
3 ☐a ☐b ☐c ☐d	13 ☐a ☐b ☐c ☐d	23 ☐a ☐b ☐c ☐d	33 ☐a ☐b ☐c ☐d
4 ☐a ☐b ☐c ☐d	14 ☐a ☐b ☐c ☐d	24 ☐a ☐b ☐c ☐d	34 ☐a ☐b ☐c ☐d
5 ☐a ☐b ☐c ☐d	15 ☐a ☐b ☐c ☐d	25 ☐a ☐b ☐c ☐d	35 ☐a ☐b ☐c ☐d
6 ☐a ☐b ☐c ☐d	16 ☐a ☐b ☐c ☐d	26 ☐a ☐b ☐c ☐d	36 ☐a ☐b ☐c ☐d
7 ☐a ☐b ☐c ☐d	17 ☐a ☐b ☐c ☐d	27 ☐a ☐b ☐c ☐d	37 ☐a ☐b ☐c ☐d
8 ☐a ☐b ☐c ☐d	18 ☐a ☐b ☐c ☐d	28 ☐a ☐b ☐c ☐d	38 ☐a ☐b ☐c ☐d
9 ☐a ☐b ☐c ☐d	19 ☐a ☐b ☐c ☐d	29 ☐a ☐b ☐c ☐d	39 ☐a ☐b ☐c ☐d
10 ☐a ☐b ☐c ☐d	20 ☐a ☐b ☐c ☐d	30 ☐a ☐b ☐c ☐d	40 ☐a ☐b ☐c ☐d

Correctas *Incorrectas* *No Puntuadas/Sin Contestar*

CONTENIDO DE LA PRUEBA

1) Indique cuál será la salida de compilar y ejecutar el siguiente programa:

```
class Tester8 extends Thread {
    public void run() {
            System.out.println("¡Hola Mundo!");
    }

    public static void main(String[] args) {
            Runnable c = new Tester8();
            Thread t = new Thread(c);
            t.run();
    }
}
```

a) Error de compilación.
b) El programa compila bien, pero no se produce ninguna salida.
c) Compila bien y escribe “¡Hola Mundo!”.
d) Compila bien pero lanza una excepción.

2) Dado el siguiente código, se produce un problema de descoordinación para cada cálculo ejecutado y la presentación de dicho cálculo en la salida por pantalla. Indique como lo solucionaría:

```
import java.util.Random;
class CalculadorAreas implements Runnable{
    float base, altura;
    public CalculadorAreas(int base, int altura){
            this.base=base;
            this.altura=altura;
    }
    @Override
    public void run() {
            float area=this.base*this.altura/2;
            System.out.print("Base: "+this.base);
            System.out.print("\t, Altura: "+this.altura);
            System.out.println("\t--->\tArea: "+area);
    }
}
public class AreasEnParaleloDescoordinado {
    public static void main(String[] args) throws InterruptedException {
            Random generador=new Random();
            int numHilos=10000;
            int baseMaxima=3;
            int alturaMaxima=5;
            for (int i=0; i<numHilos; i++){
                    int base=generador.nextInt(baseMaxima);
                    int altura=generador.nextInt(alturaMaxima);
                    CalculadorAreas ca=new CalculadorAreas(base, altura);
                    Thread hiloAsociado=new Thread(ca);
                    hiloAsociado.start();
            }
    }
}
```

a) Incorporando “hiloAsociado.join()” como última línea del for.
b) Incorporando el modificador “synchronized” en el método CalcularAreas().
c) Incorporando el modificador “synchronized” en el método run().
d) No es cierto, la salida está perfectamente coordinada, apareciendo en cada línea una base, una altura y el cálculo del área.

3) Indique cuál será la salida de compilar y ejecutar el siguiente programa:

```
public class Tester9 extends Thread {
    static int count = 0;
    public static void main(String argv[]) {
            Tester9 t = new Tester9();
```

```
                t.increment(count);
                t.start();
                Thread.sleep(1000);
                System.out.println(count);
    }

    public void increment(int count) {
                ++count;
    }

    public void run() {
                count = count + 5;
    }
}
```

a) 5
b) 6
c) Error de compilación.
d) Error en ejecución.

4) Indique cuál será la salida de compilar y ejecutar el siguiente programa:

```
public class Tester10 extends Thread {
    static int count = 0;
    public static void main(String argv[]) throws InterruptedException {
                Tester10 t = new Tester10 ();
                t.increment(count);
                t.start();
                Thread.sleep(1000);
                System.out.println(count);
    }

    public void increment(int count) {
                ++count;
    }

    public void run() {
                count = count + 5;
    }
}
```

a) 5
b) 6
c) Error de compilación.
d) Error en ejecución.

5) Dado el siguiente código:

```
public class HiloQueCreaOtroHilo implements Runnable {
    public void run() {
                for (int i = 0; i < 5 ; i++) {
                            System.out.println(i + " " + Thread.currentThread().getName());
                            try {
                                        Thread.sleep(500);
                            } catch (InterruptedException e) {e.printStackTrace();}
                }
                System.out.println("Termina thread " + Thread.currentThread().getName());
                HiloQueCreaOtroHilo hiloLlamado = new HiloQueCreaOtroHilo();
                Thread thread = new Thread (hiloLlamado, "Juan");
                thread.setPriority(Thread.MAX_PRIORITY);
                thread.start();
    }

    public static void main (String [] args) {
                new Thread (new HiloQueCreaOtroHilo(), "Pepe").start();
                System.out.println("Termina thread main");
    }
}
```

a) El programa no finaliza porque se lanzan indefinidamente hilos “Pepe”.
b) El programa no finaliza porque se lanzan indefinidamente hilos “Juan”.
c) El programa no finaliza porque se lanzan indefinidamente hilos “Pepe” y “Juan”.
d) El programa sí finaliza.

6) Indique cuál de las siguientes salidas no será posible como resultado de la ejecución del siguiente programa:

```
class Escritor2 implements Runnable{
    String name ;

    Escritor2(String name){
            this.name = name;
    }

    public void run() {
            new  Tester12().firmar(name);
    }
}

public class Tester12 {
    public synchronized void firmar(String name) {
            System.out.print(name);
            System.out.print(name);
    }

    public static void main(String[] args) {
            new Thread(new Escritor2("Cervantes")).start();
            new Thread(new Escritor2("Lope")).start();
            new Thread(new Escritor2("Quevedo")).start();
    }
}
```

a) CervantesCervantesLopeQuevedoQuevedoLope
b) CervantesCervantesLopeLopeQuevedoQuevedo
c) LopeLopeQuevedoQuevedoCervantesCervantes
d) Todas son posibles.

7) Indique una posible salida para el siguiente programa:

```
public class Tester16 implements Runnable {
    Integer id = 0;

    public static void main(String[] args) {
            new Thread(new Tester16()).start();
            new Thread(new Tester16()).start();
    }

    public void run() {
            press(id);
    }

    synchronized void press(Integer id) {
            System.out.print(id.intValue());
            System.out.print((++id).intValue());
    }
}
```

a) 0101
b) 1010
c) 1100
d) 0123

8) Dado el bloque de sentencias:
int espera=1000;
Thread t=new Thread();
t.wait(espera);

a) Se producirá una espera de 1000 milisegundos.
b) Se producirá una espera de 1000 segundos.
c) Dicho bloque de sentencias requiere tratamiento de posible excepción.
d) Dicho bloque de sentencias es incorrecto.

9) ¿Qué hace el método sleep(long millis) en la clase Thread?
a) Hace que el hilo sobre el que es invocado, se duerma (cesa temporalmente su ejecución) durante el tiempo especificado de milisegundos.
b) Hace que el hilo que actualmente se está ejecutando se duerma (cesa temporalmente su ejecución) durante el tiempo especificado de milisegundos.
c) Hace que el hilo del main() se duerma (cesa temporalmente su ejecución) durante el tiempo especificado de milisegundos.
d) Hace que el hilo que actualmente se está ejecutando se duerma (cesa temporalmente su ejecución) durante el tiempo especificado de milisegundos y tras ello empiece de nuevo a ejecutarse.

10)
Dado el siguiente programa, entonces la salida:

```
public class Tester22 {
    public static void main(String[] args) throws InterruptedException {
            Runnable t1 = new Runnable() {
                    public void run() {
                            try {
                                    System.out.print("t1-Antes ");
                                    Thread.sleep(100);
                                    System.out.print("t1-Después ");
                            } catch (InterruptedException e) { }
        }
      };

      final Thread t2 = new Thread() {
        public void run() {
          try {
            System.out.print("t2-Antes ");
            //synchronized (this) {
              wait();
            //}
            System.out.print("t2-Después ");
          } catch (InterruptedException e) { }
        }
      };

      t2.start();
      new Thread(t1).start();
   }
}
```

a) “t1-Antes” y “t1-Después” formarán parte de la salida.
b) “t2-Antes” y “t2-Después” formarán parte de la salida.
c) “t1-Antes” y “t2-Después” formarán parte de la salida.
d) Se producirá una excepción IllegalMonitorStateException en tiempo de ejecución.

11) Si en el programa anterior descomentamos las dos líneas que aparecen comentadas, cuál de las siguientes salidas no es posible:
a) t2-Antes t1-Antes t1-Después
b) t1-Antes t2-Antes t1-Después
c) t1-Antes t2-Antes t1-Después
d) t1-Antes t2-Antes t2-Después

12) Dado el siguiente programa, se desea conseguir el efecto del clásico “Me quiere, no me quiere”, de manera perfectamente alterna hasta llegar a un límite de veces establecido. Para ello…

```
import java.io.Console;

class HiloDeshojando extends Thread {
    Tester24 pregunta;
    HiloDeshojando(Tester24 contract) {
        this.pregunta = contract;
    }

    public void run() {
        pregunta.preguntar();
    }
}

public class Tester24 {
    static StringBuilder cadenaDeshojando = new StringBuilder(" +Me Quiere+ ");
    boolean meQuiere = true;

    public void ponerNoMeQuiere() {
        if (meQuiere && !estoyCansado()) {
            cadenaDeshojando.append(" -No me quiere- ");
            meQuiere = false;
        }
    }

    public  void ponerMeQuiere() {
        if (!meQuiere && !estoyCansado()) {
            cadenaDeshojando.append(" +Me quiere+ ");
            meQuiere = true;
        }
    }

    public boolean estoyCansado() {
        return cadenaDeshojando.length() >= 100;
    }

    public void preguntar() {
        while (!estoyCansado()) {
            ponerNoMeQuiere();
            ponerMeQuiere();
        }
    }

    public static void main(String[] args) {
        Tester24 prueba = new Tester24();
        new HiloDeshojando(prueba).start();
        new HiloDeshojando(prueba).start();
        try {
            Thread.sleep(2000);
        } catch (InterruptedException e) { }
        System.out.println(""+cadenaDeshojando.toString());
    }
}
```

a) Debemos sincronizar preguntar()
b) Debemos sincronizar ponerMeQuiere()
c) Debemos sincronizar ponerNoMeQuiere()
d) Debemos sincronizar estoyCansado()

13) Otra alternativa para hacer lo pedido en el ejercicio anterior sería:
a) Añadir un Thread.sleep() al final del método preguntar().
b) Añadir un Thread.sleep() al principio y al final del método preguntar().
c) Sincronizar tanto ponerMeQuiere() como ponerNoMeQuiere().
d) No existe otra forma.

14) ¿Cuál será la salida del siguiente programa?

```
class Tester25{
    public static void main(String... args){
        class A implements Runnable{
            public void run(){
               display();
            }

            synchronized void display(){
               for(int i=0;i<5;i++){
                    System.out.print("Hola ");
                    try{
                       Thread.sleep(1000);
                    } catch(InterruptedException e){}
                    System.out.print(Thread.currentThread().getName());
               }
            }
        }

        A ob1=new A();
        A ob2=new A();
        Thread ob3=new Thread(ob1,"Don Pepito ");
        Thread ob4=new Thread(ob2,"Don José ");
        ob3.start();
        ob4.start();
    }
}
```

a) La salida será impredecible.
b) La salida mostrará “Hola Don Pepito Hola Don José” alternativamente cinco veces.
c) Se producirá un error de compilación.
d) Se producirá un error en tiempo de ejecución.

15) ¿Cuál de los siguientes métodos no compilará?

```
class Tester26{

     Object o;

    public static void main(String... args){
        class A implements Runnable{
            public void run(){
               display();
            }

            synchronized void display(){
                  //Codigo
               }
        }
    }

    public synchronized void metodoA(){
      //Método A
    }

    private synchronized (this) metodoB(){
      //Método B
    }

    void metodoC(){synchronized (o){
      //Método C
     }
    }

     void metodoD(){synchronized(Object.class){
      //Método D
```

```
    }
   }
}
```

a) metodoA()
b) metodoB()
c) metodoC()
d) metodoD()

16) Para el caso en que queramos actualizar una barra de progreso mediante la clase SwingProgress:
a) Debemos enviar datos con el método publish() al método doInBackground().
b) Debemos enviar datos con el método doInBackground() al método process().
c) Debemos enviar datos con el método publish() al método process().
a) Debemos enviar datos con el método update() al método doInBackground().

17) ¿Qué es el EDT?
a) Es el hilo usado en Java para procesar eventos provenientes desde la cola de eventos de interfaz gráfica de las bibliotecas gráficas AWT y Swing.
b) Es el hilo usado en Java para procesar eventos de teclado.
c) Es el hilo usado en Java para procesar eventos de ratón.
d) Es el hilo usado en Java para procesar eventos relativos a conexiones de red.

18) Dado el clásico código de sincronización entre el productor y el consumidor para acceder a un dato de un contenedor:

```
public class ProductorConsumidorTest {
    public static void main(String[] args) {
            Contenedor c = new Contenedor();
            Productor produce = new Productor(c);
            Consumidor consume = new Consumidor(c);
            produce.start();
            consume.start();
    }
}

class Contenedor {
    int dato;
    public void put(int dato) { this.dato=dato; }
    public int get() { return dato; }
}

class Productor extends Thread {
    private Contenedor contenedor;
    public Productor(Contenedor c) {
            contenedor = c;
    }
    public void run() {
            for (int i = 0; i < 3; i++) {
                    contenedor.put(i);
                    System.out.println("Productor. put: " + i);
                    try {
                            sleep((int)(Math.random() * 100));
                    } catch (InterruptedException e) { }
            }
    }
}

class Consumidor extends Thread {
    private Contenedor contenedor;
    public Consumidor (Contenedor c) {
            contenedor= c;
    }
    public void run() {
            int value = 0;
            for (int i = 0; i < 3; i++) {
```

```
                    value = contenedor.get();
                    System.out.println("Consumidor. get: " + value);
                }
        }
}
```

a) La salida podrá ser:
Productor. put: 0
Consumidor. get: 0
Consumidor. get: 0
Consumidor. get: 0
Productor. put: 1
Productor. put: 2
b) La salida podrá ser:
Consumidor. get: 0
Productor. put: 0
Consumidor. get: 0
Consumidor. get: 0
Productor. put: 1
Productor. put: 2
c) La salida será:
Productor. put: 0
Consumidor. get: 0
Productor. put: 1
Consumidor. get: 1
Consumidor. get: 2
Productor. put: 2
d) Serán posibles la a) y la b)

19) En el ejercicio anterior podremos mejorar la sincronización entre el productor y el consumidor a través del modificador sychronized. ¿Dónde lo incorporaría?
a) En el método get() de la forma “public synchronized int get(){…}” y en el método put() de la forma “public synchronized void put(int dato) {…}”
b) En el método get() de la forma “public int synchronized get(){…}” y en el método put() de la forma “public void synchronized put(int dato) {…}”
c) Al iniciar la definición de la clase Productor de la forma “class synchronized Productor extends Thread {…}” y al iniciar la definición de la clase consumidor de la forma “class synchronized Consumidor extends Thread {…}”
d) Son válidas tanto la b) como la c).

20) Además para el ejercicio anterior debería crear una variable que controlara si hay un dato o no en el contenedor:
a) Es cierto, ya que si hay dato, no se puede usar put(), pero sí get(), y si no lo hay, se puede usar put() pero sí get().
b) Es cierto, ya que si hay dato, se puede usar put(), pero no get(), y si no lo hay, se puede usar get() pero no get().
c) Es falso ya que a través de synchronized ya se gestiona que un hilo alterne con otro en el acceso al contenedor.
d) Es falso ya que gracias a la variable “dato” que ya se usaba, los hilos Productor y Consumidor ya controlan su sincronización.

21) Además, para el ejercicio anterior, indique la afirmación correcta:
a) El método get() debe tener un wait() cuando haya dato en el contenedor y un notifyAll() para cuando acabe de meter un dato en dicho contenedor.
b) El método put() debe tener un wait() cuando haya dato en el contenedor y un notifyAll() para cuando acabe de meter un dato en dicho contenedor.
c) El método get() debe tener un notifyAll() cuando haya dato en el contenedor y un wait() para cuando acabe de meter un dato en dicho contenedor.
d) El método put() debe tener un notifyAll() cuando haya dato en el contenedor y un wait() para cuando acabe de meter un dato en dicho contenedor.

22) Indique la afirmación correcta para la clase Semaphore:
a) El método release() y release(1) son equivalentes.
b) El método release() y release(0) son equivalentes.
c) El método release() y release(true) son equivalentes.
d) El método release() y release(false) son equivalentes.

23) Dado el código siguiente, indique la afirmación correcta

```
public class ConteoHiloQueDuerme2 extends Thread {
    public void run(){
        for(int num =1; num<=3;num++){
            System.out.println("Hilo Secundario: " +num+" ");
            try {
                Thread.sleep(1000);
            }catch(InterruptedException e){       }
        }
        System.out.println("Hilo Secundario finalizado");
    }

    public static void main(String []args){
        ConteoHiloQueDuerme2 ejer = new ConteoHiloQueDuerme2();
        ejer.start();
        for(int num =1; num<=3;num++){
            System.out.println("Hilo Principal: " +num+" ");
            try {
                Thread.sleep(1000);
            }catch(InterruptedException e){       }
        }
        System.out.println("Hilo Principal finalizado");
    }
}
```

a) El hilo principal siempre finalizará antes que el secundario.
b) El hilo secundario siempre finalizará antes que el principal.
c) El hilo principal y el secundario finalizarán a la vez no garantizándose cuál se muestra encima en la salida estándar.
d) El hilo principal y el secundario no tienen por qué acabar a la vez.

24) Indique la afirmación correcta:
a) El método yield() nunca causará que un hilo pase a estado de espera/bloqueado/dormido, simplemente pasa de “ejecutándose (running)” a “listo para ejecutar (runnable)”.
b) El método yield() garantiza que siempre que el hilo actual ceda a los hilos a la espera la posibilidad de ejecutarse.
c) El método yield() garantiza que siempre que el hilo actual siga ejecutándose haciendo esperar a los otros hilos.
d) Todas son falsas.

25) ¿Cómo establecería la prioridad del hilo principal a valor 10?
a) Escribiendo “this.currentThread().setPriority(10);” en el main().
b) Escribiendo “Thread.currentThread().setPriority(Thread.MAX_PRIORITY);” en el main().
c) Escribiendo “Thread.setPriority(10);” en el main().
d) Escribiendo “this.Thread.setPriority(10);” en el main().

26) Si dentro del método run de un hilo nos encontramos con el siguiente código:

```
try {
    Thread.sleep(2000);
    System.out.println("Junio");
    Thread.sleep(2000);
    System.out.println("Julio");
    Thread.sleep(2000);
    System.out.println("Agosto");
} catch (Exception e) { }
```

a) El hilo principal cederá el control a dicho hilo, pasarán 2s, escribirá Junio, pasarán 2s, escribirá Julio, pasarán 2s y escribirá Agosto.
b) El hilo principal cederá el control a dicho hilo, escribirá Junio, pasarán 2s, escribirá Julio, pasarán 2s y escribirá Agosto.
c) El hilo principal cederá el control a dicho hilo, pasarán 2s y escribirá Junio, Julio y Agosto, en ese orden.
d) El hilo principal cederá el control a dicho hilo, pasarán 2s y se establecerá una competición por escribir Junio, Julio y Agosto, de forma que no se garantizará el orden.

27) El método enumerate() de la clase ThreadGroup:

a) Admite como parámetro un array de objetos Thread.
b) Admite como parámetro un ArrayList de objetos Thread.
c) Admite como parámetro tanto un array como un ArrayList de objetos Thread.
d) Todas son falsas.

28) ¿Cuál de los siguientes métodos no corresponde a la interfaz BlockingQueue?
a) offer()
b) insert()
c) put()
d) add()

29) La clase Exchanger:
a) Permite la sincronización e intercambio de variables entre dos hilos.
b) Permite la sincronización e intercambio de variables entre dos o más hilos.
c) Permite la sincronización e intercambio de variables entre un número ilimitado de hilos.
d) Todas son ciertas.

30) Indique cuál de las siguientes no es una clase sincronizadora de Java:
a) CountDownLatch
b) CyclicBarrier
c) Exchanger
d) ActivateSyncThread

31) Indique la afirmación correcta en relación a los sockets:
a) Los procesos de las aplicaciones y los procesos de los protocolos de transporte forman parte del SO.
b) Los procesos de las aplicaciones y los procesos de los protocolos de transporte residen en el espacio de usuario.
c) Los procesos de las aplicaciones residen en el espacio de usuario mientras que los procesos de los protocolos de transporte forman parte del SO.
d) Los procesos de las aplicaciones forman parte del SO mientras que los procesos de los protocolos de transporte residen en el espacio de usuario.

32) La clase InetAddress de Java:
a) Sólo permite resolución directa de nombres de dominio.
b) Sólo permite resolución inversa de nombres de dominio.
c) No permite resolución directa ni inversa de nombres de dominio.
d) Permite resolución directa e inversa de nombres de dominio.

33) La clase InetAddress de Java, a través de su método getByName(String host) devuelve:
a) Una cadena con el nombre de dominio del host pasado como parámetro.
b) Una cadena con la dirección IP del host pasado como parámetro.
c) Un objeto de la clase InetAdress que representa al host pasado como parámetro.
d) Son todas falsas.

34) Considerando que mi equipo se llama “DESKTOP-7B4KSO6”, el aspecto que tendrá la salida del código siguiente, será:
```
InetAddress address = InetAddress.getLocalHost();
System.out.println(address);
```

a) DESKTOP-7B4KSO6
b) 192.168.1.131
c) DESKTOP-7B4KSO6/192.168.1.131
d) No se puede mostrar ya que no podemos mostrar por salida estándar un objeto InetAddress

35) ¿Qué clase de Java permite establecer un sistema de escucha de peticiones a través de la red?
a) ServerSocket.

b) Socket.
c) Accept.
d) Bind.

36) Indique la afirmación errónea:
a) La clase DatagramSocket garantiza que la transmisión de paquetes siempre llegue a su destino.
b) La clase DatagramSocket se basa en el protocolo UDP.
c) Los sockets de tipo datagrama son menos seguros que los sockets de tipo stream.
d) Los sockets de tipo datagrama se basan en el protocolo UDP.

37) Indique la afirmación incorrecta:
a) Un Socket se utiliza para transmitir y recibir datos.
b) Un ServerSocket no se utiliza para transmitir datos.
c) Un ServerSocket, en el servidor, espera a que un cliente quiera establecer una conexión con dicho servidor.
d) Un Socket se utiliza exclusivamente para recibir datos.

38) Para acceder al contenido de una página de HTTP:
a) Podemos hacerlo usando la clase URL y la clase Socket.
b) Solo podemos hacerlo usando la clase URL.
c) Solo podemos hacerlo usando la clase Socket.
d) No podemos hacerlo usando la clase URL ni la clase Socket.

39) Los algoritmos de cifrado que utilizaban los antiguos romanos eran de clave:
a) simétrica
b) asimétrica
c) diferencial
d) clave simétrica o asimétrica, dependiendo del algoritmo

40) Los sistemas de cifrado simétrico se basan en:
a) Cifrar con la clave privada y descifrar con la clave pública, para garantizar la autenticidad del emisor.
b) Cifrar con una sola clave, y no descifrar en el destino, para salvaguardar la integridad.
c) Cifrar y descifrar con la misma clave.
d) Cifrar con la clave pública y descifrar con la privada, para garantizar la confidencialidad.