1. PREGUNTA: ¿Una clase puede manejar objetos?

   RESPUESTA Sí, una clase puede manejar objetos. Aunque esta frase es bastante ambigua, apréndetela de memoria para el examen y no le des más vueltas.

2. PREGUNTA: ¿Pueden crearse objetos del tipo definido por una clase sin constructor declarado?

   RESPUESTA: Sí, en caso de que no haya constructor declarado, java aplica por defecto el criterio de permitir la creación de objetos como si existiera el constructor vacío (sin código).

3. PREGUNTA: ¿Una clase puede tener como nombre un número o su nombre empezar por un número?

   RESPUESTA: No. Al igual que no es válido que el nombre de una variable empiece con un número, tampoco el nombre de una clase puede empezar con un número.

4. PREGUNTA: Dado este fragmento de código, ¿cuál sería el resultado de compilarlo / ejecutarlo?

Código: ~~[Seleccionar]~~

```
public static void main (String Args[] ) {
int valor = 5;
cambiarValor(valor);
System.out.println (valor);
}

public static void cambiarValor (int valor) {
private int valor = valor * 2;
}
```

RESPUESTA: Hemos de suponer que el código está dentro de una clase, porque si no no se puede compilar ni ejecutar. El resultado es error en la línea 10 porque dentro de un método no pueden declararse variables locales precedidas de modificadores de acceso (public/private/protected)

5. PREGUNTA: ¿Este código compila y si compila, qué sentido tiene?

   private class MiClase1 { }

   RESPUESTA: No, el código no compila. Solo se admite public. Podría compilar si se tratara de una clase interna dentro de otra clase, pero tal y como está no.

6. PREGUNTA: ¿Este código compila y si compila, qué sentido tiene?

   protected class MiClase1 { }

   RESPUESTA: No, el código no compila. Solo se admite public. Podría compilar si fuera una clase interna a otra clase.

7. PREGUNTA: ¿Este código compila y si compila, qué sentido tiene?

Código: ~~[Seleccionar]~~

```
class MiClase1 { }

class MiClase2 { }

class MiClase3 { }
```

RESPUESTA: Sí, el código compila. La clase se denomina con el nombre de la primera declaración que aparezca, es decir: MiClase1. Si hay varias clases en un mismo fichero, java sólo permite que se pueda usar una de ellas desde código que esté fuera del fichero.
MiClase1 define un tipo.
Al no declararse usando public, el compilador le da visibilidad dentro del package en que esté definida. El resto de clases que se declaran tienen consideración de clases solo accesibles por otras clases dentro del fichero.

8. PREGUNTA: ¿Este código compila y si compila, qué sentido tiene?

   class MiClase1 { }

```
public class MiClase2 { }

class MiClase3 { }
```

RESPUESTA: Sí, el código compila. La clase (archivo) se denomina con el nombre de la clase que se declara como pública, es decir: MiClase2. Las otras clases se consideran clases privadas dentro de esta clase.

9. PREGUNTA: ¿Este código compila y si compila, qué sentido tiene?

```
public class MiClase1 { }

public class MiClase2 { }

class MiClase3 { }
```

RESPUESTA: No, el código no compila. El compilador no sabe qué nombre ponerle a la clase si se declaran dos nombres como público. En un archivo se pueden definir varias clases, pero en un archivo no puede haber más que una clase public. Este archivo se llamará como la clase public que contiene con extensión *.java. Con excepciones, lo habitual es escribir una sola clase por archivo.

10. PREGUNTA: ¿Este código compila y si compila, qué sentido tiene?

```
public class MiClase1 { }
class MiClase1 {}
```

RESPUESTA: No, el código no compila. En un fichero no puede haber dos clases con el mismo nombre.

11. PREGUNTA: ¿Este código compila y si compila, qué sentido tiene?

```
public class MiClase1 {
private MiClase2 prueba = new MiClase2();
public void mostrarMiClase1 () {
System.out.println (prueba.getContenido());
}
}
class MiClase2 {
String soy2 = "Hola soy 2";
public String getContenido() { return soy2; }
}
class MiClase3 {
MiClase1 pp;
MiClase2 jj;
```

```
}
```

RESPUESTA: Sí, el código compila. El fichero lleva el nombre de la clase pública. Las otras clases dentro del fichero se conocen entre sí, pero no pueden ser usadas fuera del fichero.

12. PREGUNTA: Dado este fragmento de código, ¿cuál será su salida?

```
class Cantante { public static String cantar() { return "la"; } }
public class Tenor extends Cantante {
public static String cantar () { return "fa"; }
public static void main (String [] args ) {
Tenor t = new Tenor ();
Cantante s = new Tenor ();
System.out.println (t.cantar () + " " + s.cantar() );
}
}
```

RESPUESTA: Do, re, mi, fa, sol... qué simpática la pregunta. Qué derroche de imaginación y simpatía. Veamos el fondo del asunto.
Primera cuestión:
El fichero lleva el nombre de la clase pública. La otra clase no es conocida fuera del fichero.
Al crear un Tenor no ocurre nada puesto que no hay constructor ni campos.
El método cantar de Tenor es estático: esto significa que es un método de clase y que se puede invocar tanto sobre un objeto como sobre el nombre de la clase. Idem para el método cantar de Cantante.
Aunque parece que hay una sobreescritura de métodos, en realidad no la hay: un método estático no es sobreescribible y no funciona la ligadura dinámica de métodos. Si el método de la superclase fuera estático y el de la subclase no, saltaría un error “overridden method is static”.
Al crear un Cantante s con tipo Tenor, el tipo estático es Cantante, y el dinámico Tenor. Al invocar el método cantar sobre t se nos devuelve fa, y al invocar cantar sobre s, dado que s dispone de un método estático de s nombre se ejecuta y devuelve la. Por tanto la salida es : fa-la

13. PREGUNTA: ¿Qué se obtiene al ejecutar el siguiente código?

```
public class MiClase1 {
private MiClase2 prueba;
public void mostrarMiClase1 () {
System.out.println (prueba);
}
}
class MiClase2 {
```

```
String soy2 = "Hola soy 2";
}
```

RESPUESTA: Obtenemos como salida por pantalla: null.
¿Por qué? Porque el atributo prueba, que es un objeto, no está inicializado, sino solo declarado.
Si lo inicializáramos con una declaración como `private MiClase2 prueba = new MiClase2();` la salida por pantalla sería del tipo MiClase2@1428ea
¿Por qué? Porque estamos pidiendo la salida por pantalla de un objeto, y lo que nos sale es la referencia de memoria de dónde se encuentra alojado el objeto. Usando `System.out.println (prueba.toString());` el resultado sería el mismo. Para rescatar el contenido "Hola soy 2" necesitaríamos un método get e invocar este método. Por ejemplo: `public String getContenido() { return soy2; }`, y en la otra clase `System.out.println (prueba.getContenido());`

14. PREGUNTA: ¿Qué se obtiene al ejecutar el siguiente código?

```
class Vehiculo {
public void imprimirSonido() {
System.out.print("Vehiculo");
}
}
class Coche extends Vehiculo {
public void imprimirSonido() {
System.out.print ("Bicicleta");
}
}
class Bicicleta extends Vehiculo {
public void imprimirSonido() {
System.out.print ("Bicicleta");
}
}
public class Test {
public static void main (String [] args) {
Vehiculo v = new Coche();
Bicicleta b = (Bicicleta) v;
v.imprimirSonido();
b.imprimirSonido();
}
}
```

RESPUESTA: El código compila, pero se produce una excepción en tiempo de ejecución. El error nos indica java.lang.ClassCastException: Coche cannot be cast to Bicicleta at Test.main(Test.java:22). Esto está relacionado con conversiones de tipo permitidas y no permitidas.
Tanto coche como bicicleta heredan de vehiculo, podríamos decir que

coche y bicicleta "son hermanos". El código compila porque un vehiculo podría contener una bicicleta (polimorfismo de vehiculo). Sin embargo, en tiempo de ejecución lo que realmente contiene v es un coche. El resultado, que se detecta un intento de conversión de lado a lado (entre hermanos) y nos salta el error en tiempo de ejecución. Bueno, no sé si lo he explicado muy claro pero por lo menos lo he intentado.

15. PREGUNTA: ¿Qué ocurre al ejecutar el siguiente código?

```
class Padre {}
class Hijo extends Padre {}
class Hijo2 extends Padre {}
public class CEx {
public static void main (String [] args) {
Padre p = new Padre();
Hijo h = (Hijo) p;
}
}
```

RESPUESTA: No sabemos si lo de (Hijo) p tiene algún doble sentido. El código compila, ya que el compilador no llega a conocer el tipo que almacena la variable p. En tiempo de ejecución, para hacer el casting Hijo h = (Hijo) p; sería necesario que p almacenara dinámicamente una variable tipo Hijo. Como no es el caso, salta un error en tiempo de ejecución tipo java.lang.ClassCastException: Padre cannot be cast to Hijo.
Este código sí compila y se ejecuta:

```
class Padre {}
class Hijo extends Padre {}
class Hijo2 extends Padre {}
public class Preg8CEx {
public static void main (String [] args) {
Padre p = new Hijo();
Hijo h = (Hijo) p;
}
```

16. PREGUNTA: ¿Qué ocurre al ejecutar el siguiente código?

```
public void goo() {
foo f = new foo();
System.out.println (f);
}
public class foo {
String f = "22";
public String toString() {
return "44";
```

```
}
public foo() {}
}
```

RESPUESTA: Error de compilación: un método no puede estar fuera de una clase como algo independiente.

17. PREGUNTA: ¿Cuál sería el resultado de ejecutar el método goo?

```
public void goo() {
foo f = new foo();
System.out.println (f);
}
public class foo {
String f = "22";
public String toString() {
return "44";
}
public foo() {}
}
```

RESPUESTA: Esta pregunta es un enunciado ininteligible, ya que no queda claro dónde está el método, y sin saber dónde está... De todas formas, se considera que hay que suponer que al no preguntar por la ejecución del código, sino del método, tenemos que considerar que estuviera dentro de la clase... Si el método estuviera dentro de la clase, sí compilaría, y el resultado de ejecutar el método sería: 44. Esto es así, porque en el println lo que se ejecuta es el método toString(). Y al tener sobreescrito el método toString() para que devuelva 44, esto es lo que obtenemos. A esta pregunta hay que reconocerle mérito: ole la mente brillante a la que se le ocurrió.

18. PREGUNTA: Estudia el código que se muestra a continuación. ¿Compila este código?

```
public class Preg15 {
public void prueba () {
try {
int variableLocal = 55;
} catch (ArithmeticException e) {
System.out.println( variableLocal);
}
finally { }
}
}
```

RESPUESTA: No, una variable local declarada dentro de un bloque try catch no es conocida ni en el catch ni en el finally. En resumen, si la

variable está declarada fuera del bloque try, sí se puede usar en el catch o en el finally. Si está dentro del bloque try, nos saltará un error de compilación.

19. PREGUNTA: ¿Compila este código?

```
public class Preg15 {
public void prueba () {
int variableLocal = 55;
try {
} catch (ArithmeticException e) {
System.out.println( variableLocal);
}
finally { }
}
}
```

RESPUESTA: Sí, una variable local declarada fuera de un bloque try-catch es conocida dentro del catch y también dentro del finally.

20. PREGUNTA: Dadas estas declaraciones de variables, indicar cuáles son correctas y cuáles no:

```
float foo = -1;
float foo1 = 1.0;
float foo2 = 42e1;
float foo3 = 2.02f;
float foo4 = 3.03d;
float foo5 = 0x0123;
double foo6 = 3.03d;
double foo7 = 0x0123;
double foo8 = 0x012345;
double foo9 = 0x012345678;
double foo10 = 0x0123456789;
```

RESPUESTA:

```
float foo = -1; // ---> Sí compila, declaración correcta
//float foo1 = 1.0; -->No compila Error: possible lost of precission:
found double required float
//float foo2 = 42e1; --> No compila: Error: possible lost of precission:
found double required float
float foo3 = 2.02f; // ---> Sí compila, declaración correcta
//float foo4 = 3.03d; --> No compila: Error: possible lost of precission:
found double required float
```

float foo5 = 0x0123; // ---> Sí compila, declaración correcta
double foo6 = 3.03d; // ---> Sí compila, declaración correcta
double foo7 = 0x0123; // ---> Sí compila, declaración correcta
double foo8 = 0x012345; // ---> Sí compila, declaración correcta
double foo9 = 0x012345678; // ---> Sí compila, declaración correcta
// double foo10 = 0x0123456789; ---> No compila. Error: integer number 0123456789 too large

Nota: una referencia a una posición de memoria ha de empezar con 0x seguida de hasta 9 dígitos. No se admite otra forma de expresar una referencia a una posición de memoria.

21. PREGUNTA: Dado el siguiente código ¿cuál es el contenido de la variable foo?

```
int index = 1;
boolean [] test = new boolean [3];
boolean foo = test [index];
System.out.println (foo);
```

RESPUESTA: El contenido es false. Motivo: desde que se crea un array con la instrucción new, queda relleno en todas sus posiciones con cero para valores numéricos, false para booleanos, null para objetos incluido strings, o carácter nulo para tipo char.

22. PREGUNTA: Dado el siguiente código ¿cuál es el contenido de la variable c?

```
char c = '\0';
System.out.println (c);
```

RESPUESTA: El contenido es el carácter nulo. No tiene representación por pantalla, al igual que el string nulo “” no tiene representación por pantalla.

23. PREGUNTA: ¿Qué devuelve esta expresión en Java?

```
(1>1) && (1 > 1) == (1 > 1) == false
```

RESPUESTA: Evaluemos por partes:

false && false == false == false;
false && false = false;
false == false; --> vale true;
true == false; --> vale false;

Luego la expresión devuelve false

24. PREGUNTA: ¿Qué devuelve esta expresión en Java?

    (1 ==1) | (10 > 1) == true | true == true

    RESPUESTA: Evaluemos por partes:
    1 == 1 devuelve true;
    10 > 1 devuelve true;
    true | true == true | true == true
    true == true | true == true
    true | true

    Luego la expresión devuelve true

    La diferencia entre || y | es que con || se deja de evaluar la expresión desde que es verdadero el primer caso, mientras que con | se evalua también el segundo caso.

25. PREGUNTA: Dado el siguiente código java, ¿Cuál es su resultado?

```
class Top {
public Top (String s) { System.out.print ("B"); }
}

public class Bottom2 extends Top {
public Bottom2 (String s) { System.out.print ("D"); }
public static void mian (String [] args) {
Bottom2 obj = new Bottom2("C");
System.out.println (" ");
}
}
```

    RESPUESTA: Se obtiene un error de compilación debido a que el constructor de Bottom2 contiene instrucciones y omite la instrucción super(s), es decir, no se incluye el código necesario para que al construir un objeto de la clase Bottom2 se ejecute previamente el constructor de Top;. Si la incluyera el código devolvería BD.
    Siempre es obligatorio super(parámetros) excepto cuando existe un constructor de la clase superior sin parámetros. En este caso escribiríamos super(), pero si no lo incluimos el compilador lo hará automáticamente evitando que salte un error.

26. PREGUNTA: Dado el siguiente código java, ¿Cuál es su resultado?

```
class Hotel {
public int reservas;
public void reservar() { reservas++; }

}

public class SuperHotel extends Hotel {
public void reservar() {reservas--;}

public void reservar (int size) {
reservar();
super.reservar();
reservas += size;
}

public static void main (String [] args) {
SuperHotel hotel = new SuperHotel();
hotel.reservar(2);
System.out.print (hotel.reservas);
}
}
```

RESPUESTA: Por carecer ambas clases de constructor no hay problema. Al crear el objeto y manipular una variable que no ha sido inicializada (mala práctica, pero compilable y ejecutable sin error), la variable opera con su valor por defecto que en el caso de un int es cero. El resultado del método reservar (int size) es que la variable reservas se incrementa en lo que indica size. Por tanto el resultado de la ejecución es 2.

27. PREGUNTA: ¿Cuál es la definición de depuración?

RESPUESTA: "La depuración es el intento de apuntar con precisión y corregir un error en el código". Añadiremos que esto es una definición un tanto críptica y ambigua, pero no le des muchas vueltas. Apréndetela para el examen y punto.

28. PREGUNTA: Diga si es cierto o falso:

a) El encapsulamiento reduce la cohesión.
b) El encapsulamiento reduce el acoplamiento.

RESPUESTA:

a) Falso
b) Verdadero

29. PREGUNTA: ¿Son las llamadas a métodos en java polimórficas?

    RESPUESTA: Sí. Decimos que sí en el sentido de que un método sobreescrito en una subclase implica que una invocación al método pueda invocar al método de la subclase o al de la superclase, dependiendo del tipo dinámico de la variable.

30. PREGUNTA: ¿Cuál es el nombre del fichero java y de la clase según este código?

```
class BaseClass {
private float x = 1.0f;
protected float getVar () { return x; }
}

class Subclass extends BaseClass {
private float x = 2.0f;
}
```

    RESPUESTA: El nombre es BaseClass. A falta de especificación de una clase como public, siempre se coge como nombre el nombre de la primera clase que aparezca.

31. Ahí vamos con otra.

    PREGUNTA: ¿Qué dos formas hay de sobreescribir el método getVar() definido en el código, dentro de la subclase?

```
class BaseClass {
private float x = 1.0f;
protected float getVar () { return x; }
}

class Subclass extends BaseClass {
private float x = 2.0f;
}
```

El método lleva como modificador de acceso protected, lo que significa que es visible en la clase, el package y por las subclases. Protected podríamos decir que es cuasi-público. Para reescribirlo hemos de usar un modificador igual o que dé mayor acceso. Por tanto hay dos formas de redefinirlo:

a) protected float getVar() {return x;}
b) public float getVar() {return x;}

Las formas sin modificador y private son más restrictivas y por tanto no pueden usarse para sobreescribir el método.

32. PREGUNTA: ¿Cuál es la definición de programación defensiva, programación extrema y programación por parejas?

RESPUESTA: yo en pareja prefiero hacer otras cosas que no son programar. Bueno, vamos con las definiciones verdaderas.

Programación defensiva: técnica de programación basada en maximizar las verificaciones suponiendo que las peticiones de clientes o entradas de usuario no tienen por qué responder a lógica alguna. Busca mejorar la calidad del software evitando errores, pero supone más código y uso de recursos en verificaciones.

Programación extrema: considerar que los cambios de requisitos sobre la marcha son un aspecto natural, inevitable e incluso deseable del desarrollo de proyectos. Según esta técnica, ser capaces de adaptarnos a los cambios de requisitos en cualquier punto de la vida del proyecto es una aproximación mejor y más realista que intentar definir todos los requisitos al comienzo del proyecto. Una técnica dentro de la programación extrema es la programación por parejas.

Programación por parejas: técnica que se basa en que la implementación de una clase no la realiza una persona sola para después pasar a revisión por otra, sino que el desarrollo de la clase se hace entre dos personas con comunicación constante desde el primer momento.

33. PREGUNTA: ¿Qué interface y clase hemos de usar para interceptar el evento de cierre de una ventana?

RESPUESTA: Interface WindowListener, clase WindowAdapter. Mucha gente piensa que se usaría un ActionListener, pero JFrame no tiene

disponible el método addActionListener, sí en cambio el método addWindowListener. Lo mejor es hacer un pequeño código y verlo con nuestros propios ojos, porque si no lo hacemos memorizarlo sin más es un poco difícil. Queda dicho.

34. PREGUNTA: ¿Qué resultado se obtiene al ejecutar este código Java?

```
public class test {
public static void add3 (Integer i) {
int val = i.intValue();
val += 3;
i = new Integer (val);
}

public static void main (String args[]) {
Integer i = new Integer (0);
add3 (i);
System.out.println (i.intValue ( ) );
}
}
```

RESPUESTA:

Esta pregunta tiene su interés, ya que en el fondo está la forma en que se pasan los parámetros a los métodos y constructores: ¿por valor o por variable? Por pantalla se imprime un 0. i = new Integer (val) es una modificación de la referencia a objeto que es la variable. Como la referencia a objeto se pasa por valor (una copia), esta modificación no es conocida luego en el main. Resultado, se imprime un cero.

Por si no ha quedado claro: los parámetros en Java se pasan por valor. Lo que pasa es que en java al tratarse de programación orientada a objetos sacar conclusiones de esta afirmación no es igual de fácil que en otros lenguajes. Este es uno de los motivos por los que se dice que la programación orientada a objetos es un poco más complicada.

35. PREGUNTA: Dado el siguiente código, indicar si el método `private void setVar (int a, float c, int b) { }` constituiría una sobrecarga válida del método setVar.

```
public class Preg15testJunio {
public void setVar (int a, int b, float c) { }
// Nuevo código
}
```

RESPUESTA: Sí, este método sería una sobrecarga válida desde el

momento en que los tipos de los parámetros no coinciden con los tipos del método ya escrito. En concreto en el ya escrito tenemos int, int, float y en el nuevo tenemos int, float, int. Sobrecargar implica que constituyan métodos distintos, y esto se produce. Sobreescribir sería otra cosa: que coincidan el nombre del método, de los parámetros y el tipo de los parámetros.

36. PREGUNTA: Dado el siguiente código, indicar si el método `protected void setVar (int a, int b, float c) { }` constituiría una sobrecarga válida del método setVar.

```
public class Preg15testJunio {
public void setVar (int a, int b, float c) { }
// Nuevo código
}
```

RESPUESTA: No, este método no sería una sobrecarga válida desde el momento en que coincide el nombre y los tipos de los parámetros.

37. PREGUNTA: Dado el siguiente código, indicar si el método `protected void setVar (int flin, int flan, float flon) { }` constituiría una sobrecarga válida del método setVar.

```
public class Preg15testJunio {
public void setVar (int a, int b, float c) { }
// Nuevo código
}
```

RESPUESTA: No, este método no sería una sobrecarga válida desde el momento en que coinciden los tipos de los parámetros y el nombre del método. Que el nombre de los parámetros no coincida es irrelevante, ya que esto no permite diferenciar un método de otro a la hora de invocarlo.

38. PREGUNTA: Dado el siguiente código, indicar si el método `public int setVar (float a, int b, int c) {return b;}` constituiría una sobrecarga válida del método setVar.

```
public class Preg15testJunio {
public void setVar (int a, int b, float c) { }
// Nuevo código
}
```

RESPUESTA: Sí, sería una sobrecarga válida desde el momento en que coincide el nombre pero los tipos de los parámetros son distintos. No sería válida si los tipos fueran coincidentes.

39. PREGUNTA: ¿Cuál es el resultado de ejecutar este código?

```
int i=0;
if (i) {
System.out.println ("Hello");
}
```

RESPUESTA: Error de compilación: en el if se espera un tipo boolean y se encuentra un tipo int. Tampoco sería válido i || i por el mismo motivo. Sí sería válido i ==i (imprimiría Hello) ó i!=i (no imprimiría nada por pantalla).

40. PREGUNTA: ¿Cuál es el resultado de ejecutar este código?

```
int i=1;
int j=2;
if(i==1 || j==2)
System.out.println ("OK");
```

RESPUESTA: Se muestra por pantalla OK. La ausencia de llaves en el if se admite, ejecutándose como consecuencia la primera instrucción a continuación del if.

41. PREGUNTA: ¿Cuál es el resultado de ejecutar este código?

```
public class Preg3Pass {
static int j=20;
public static void main (String[] Args) {
int i=10;
Preg3Pass p = new Preg3Pass();
p.ametodo(i);
System.out.println(i);
System.out.println(j);
}

public void ametodo (int x) {
x=x*2;
j=j*2;
}
}
```

RESPUESTA: Se muestra por pantalla 10 y 40. Razonamiento: la variable j es una variable de clase pero una variable: no es una

constante porque para serlo debería llevar la palabra clave final. Por tanto es conocida en toda la clase (constructor y métodos). En el main la variable i se pasa como parámetro al método por valor (una copia) con lo cual la modificación del parámetro no es conocida fuera del método. Sin embargo, la modificación de una variable de clase sí es conocida si se realiza la manipulación directamente.

42. PREGUNTA: ¿Cuál es el resultado de ejecutar este código?

```
public class Probando {
boolean prueba = false;

public Probando() {trocar(prueba);
System.out.println (prueba);}

public static void main (String[] Args) {
Probando p = new Probando ();
}

public boolean trocar (boolean v) {
v = !v;
return v;
}
}
```

RESPUESTA: Se muestra por pantalla false. Razonamiento: al crear un objeto se invoca el constructor. La ejecución del método devuelve true, pero esto no tiene efectos sobre la variable porque la variable se pasa por valor.

43. PREGUNTA: ¿Cuál es el resultado de ejecutar este código?

```
public class Probando {
boolean prueba = false;

public Probando() {trocar(prueba);
System.out.println (prueba);}

public static void main (String[] Args) {
Probando p = new Probando ();
}

public boolean trocar (boolean v) {
prueba = !prueba;
return prueba;
}
```

```
}
```

RESPUESTA: Se muestra por pantalla true. Razonamiento: al crear un objeto se invoca

el constructor. En el método se manipula la variable prueba directamente, no el parámetro que realmente no tiene funcionalidad tal y como está definido.

44. PREGUNTA: ¿Cuál es el resultado de ejecutar este código?

```
public class Probando {
boolean prueba = false;
public Probando() { System.out.println (prueba);}
public static void main (String[] Args) {
Probando p = new Probando ();
System.out.println (!prueba);
}
}
```

RESPUESTA: Error de compilación: una variable no estática no puede ser utilizada en un contexto estático. Un método estático solo podrá trabajar con atributos estáticos, ya que un método de clase se invoca bajo el nombre de la clase, no bajo el nombre de un objeto. No tendría sentido manipular directamente atributos de objeto dentro de un método estático. En el contexto estático sí podremos crear y manipular objetos a través de los métodos propios de dichos objetos.

45. PREGUNTA: ¿Cuál es el resultado de n%6 para cualquier entero?

RESPUESTA: % es el operador mod, o resto de una división. 0%6 =6; 1%6=5; 2%6=4; 3%6=3; 4%6=2; 5%6=1; 6%6 =0; 7%6 = 1 y seguimos igual con 2, 3, 4…Tenemos enteros de 0 a 6, pero ¿qué ocurre con los números negativos? -1%6 =-1; -2%6= -2 …hasta -6%6=0. Por tanto el resultado de n%6 para cualquier entero es -5, -4, -3, -2, -1, 0, 1, 2, 3, 4 ó 5.

46. PREGUNTA: ¿Cuáles son los tres tipos estándar de diálogos que posee la clase de Swing JOptionPane?

RESPUESTA: Diálogo de mensaje, diálogo de entrada y diálogo de

confirmación. Así de breve 

47. PREGUNTA: ¿Qué es una excepción?

RESPUESTA: Una excepción es un objeto que contiene información relativa a los detalles de un error en la ejecución de un programa.

48. PREGUNTA: Si la clase Y es una subclase de la clase X. ¿Cuál de los siguientes códigos no compilará?

    a. X objeto=new X();
    b. Y objeto=new X();
    c. Y objeto= new Y();
    d. X objeto=new Y();

    RESPUESTA: La opción a compila sin problemas. La opción c también. X objeto = new Y (); compila puesto que las superclases son polimórficas y admiten tipos de las subclases. Un vehículo puede crearse definiéndolo como un coche.
    No compilará la opción b: un coche no puede crearse definiéndolo como un vehículo.

49. PREGUNTA: Si una variable calificada como final referencia a un objeto, ¿es posible modificar el objeto al que apunta dicha variable mientras el programa se está ejecutando?

    RESPUESTA: No, no es posible. Al estar declarado como constante no es posible modificación alguna.

50. PREGUNTA: ¿Cuál es el tipo de retorno para los métodos add (E e) de las interfaces Set y List?

    RESPUESTA: Podríamos pensar que el tipo es void, porque es una operación en la que usualmente no esperamos un retorno. Sin embargo, el tipo de retorno es boolean, y nos sirve para comprobar si la inserción efectivamente se hizo o no se hizo en el caso de que queramos hacerlo. Por ejemplo, el retorno será false si intentamos insertar un elemento duplicado en un set.

51. PREGUNTA: ¿Puede una clase contar cuántas instancias han sido creadas de dicha clase?

    RESPUESTA: Sí, bastaría con incluir una variable estática (de clase) que en el constructor se incrementara unitariamente cada vez que se crea una instancia de dicha clase.

52. Esta pregunta tiene interés comprenderla bien.

    PREGUNTA: ¿Cuál será el resultado de ejecutar este código?

```
List <String> list = Arrays.asList ("a", "b", "c");
System.out.println (list);
```

RESPUESTA: Este es un caso curioso, porque List es una interface, y por lo tanto no es instanciable. Podría parecer que por ello nos saltaría un error de compilación, pero no es así. El tipo devuelto por Arrays.asList es simplemente List, lo cual tampoco nos aclara mucho. No obstante, el resultado de ejecución es que se muestra por pantalla [a, b, c], lo cual muestra que se ha creado un objeto. ¿De qué tipo? Si nos ponemos a investigar…

```
if (list instanceof AbstractList) {System.out.println ("Es una instancia de AbstracList");}

if (list instanceof ArrayList) {System.out.println ("Es una instancia de ArrayList");}

if (list instanceof AbstractSequentialList) {System.out.println ("Es una instancia de AbstractSequentialList");}
```

El resultado es que por defecto Java, al menos en Java 6, le asigna por defecto el tipo AbstractList.

List <Integer> miListaEnteros = Arrays.asList (1, 5, 7, 13, 21); también es válido.

No obstante, el uso de esta sintaxis da lugar a comportamientos extraños. Por ejemplo, se permite la modificación de la lista con el método set pero no se permite su ampliación con el método add.

53. ¿Cuál es el resultado de ejecutar este código?

```
List <String> list = new ArrayList<String> (Arrays.asList ("a", "b", "c")) ;
List <Integer> miListaEnteros = new ArrayList<Integer>
( Arrays.asList (1, 5, 7, 13, 21)) ;
```

RESPUESTA: El código compila y se crean los ArrayList correctamente.

54. PREGUNTA: ¿Qué faltaría en el siguiente código para que sea válido?

```
public String getNombre () {
if (nombre==null) {trhow new NullPointerException(); } else {
return this.nombre; }
```

RESPUESTA: Podría parecer que falta una sentencia return en la primera parte del condicional, pero en realidad no es así. El lanzamiento de una excepción con throw interrumpe la ejecución del método, por lo que no es necesaria sentencia return en estos casos. Por tanto no falta nada. Pregunta con truco!!!

55. PREGUNTA: Durante la ejecución de un constructor, ejecutada la mitad del código del mismo, se alcanza una sentencia throw. ¿Llega a crearse el objeto?

    RESPUESTA: No, si se alcanza una sentencia throw durante la ejecución del constructor no llega a crearse el objeto.

56. Un metodo de clase static. ¿se podría llamar sin instanciar un objeto de la clase?.
    a.- No, siempre hay que declarar el objeto y luego llamar al metodo.
    b.- No, siempre hay que hay que declarar el objeto, crearlo y a continuación llamar al metodo.
    c.- Sí, se podría llamar al metodo desde la misma clase.
    d.- Sí, se podría llamar al metodo pero sólo si la clase es abstracta.

    RESPUESTA: Te doy mi opinión: creamos una clase denominada prueba con este contenido

```
import java.util.Scanner;

public class prueba {
public static void miMetodo () {
System.out.println ("Introduzca el primer número (entero):");
Scanner ent1 = new Scanner(System.in);
int num1 = ent1.nextInt();

System.out.println ("Introduzca el segundo número (real)");
Scanner ent2 = new Scanner(System.in);
double num2 = ent2.nextDouble();

System.out.println ("El producto de los dos números es " + (num1 *
num2) );
}
}
```

    A continuación creamos una clase denominada sumaDosNumeros con este contenido:

```
public class sumaDosNumeros {
public static void main (String[] Args) {
prueba.miMetodo();
```

```
}

}
```

Resultado: ejecutamos el main y el programa se ejecuta correctamente.

Conclusión:

Se puede llamar al método sin instanciar un objeto de la clase.

Yo no veo ninguna de las respuestas clara, si tuviera que responder respondería la c) por eliminación.

57. PREGUNTA: ¿Cuáles de las siguientes declaraciones nunca podrían generar dos objetos de la misma clase con los mismos valores?.
a.- Triangulo objCir1 = new Triangulo(5,8);
Triangulo objCir2 = new Triangulo(objCir1);
b.- Triangulo objCir1 = new Triangulo(5,8);
Triangulo objCir2 = objCir1;
c.- Triangulo objCir1 = new Triangulo(5,8);
Triangulo objCir2 = new Triangulo(5,8);
d.- Triangulo objCir1 = new Triangulo(5);
Triangulo objCir2 = new Triangulo(5);

RESPUESTA: Veamos las opciones que sí pueden generar dos objetos de la misma clase con los mismos valores:

a) La clase Triangulo podría tener dos constructores, uno que reciba dos parámetros y otro que reciba un objeto de la clase. El constructor que recibe el objeto de la clase podría servir para crear otro objeto con los mismos valores.

c) Estamos generando dos objetos de la misma clase en principio con los mismos valores porque le pasamos los mismos parámetros

d) Estamos generando dos objetos de la misma clase en principio con los mismos valores porque le pasamos los mismos parámetros

Ahora nos queda la opción b) ¿Qué es lo que hace?

Declara una variable apuntadora objCir1 y crea un objeto (espacio de memoria) al que apunta la variable.

Luego declara una variable apuntadora objCir2 y dice que esa variable apuntadora apunta al mismo objeto al que está apuntando la otra variable.

Por tanto de esta manera nunca creamos dos objetos ya que lo que hacemos es crear un objeto y tener dos variables que apuntan a él.