★ Natalia Denise Fuentes Rubio
★ Semana 2 – Evidencias
★ Academia Java

Cuestionario Java

1. **Indica a JUnit que la propiedad que usa esta anotación es una simulación y, por lo tanto, se inicializa como tal y es susceptible de ser inyectada por @InjectMocks.**
   - ☐ Mockito
   - ☑ Mock @Mock es la anotación que se utiliza para indicar que una propiedad debe ser una simulación y debe ser inyectada.
   - ☐ Inject
   - ☐ InjectMock

2.

```
Given

public static void main(String[] args){
      int[][] array2D = {{0,1,2}, {3,4,5,6}};
      System.out.print(array2D[0].length + "");
      System.out.print(array2D[1].getClass().isArray() + "");
      System.out.print(array2D[0][1]);
}
What is the result?
```

   - ☐ 3false3
   - ☐ 3false1
   - ☐ 2false1
   - ☑ 3true1. La longitud del primer array es de 3, por lo que el primer resultado impreso será 3. Para el segundo valor se verifica que el segundo array sea, en efecto, un array, por lo cuál se imprime un true. Para el tercer valor nos indica que quiere el segundo elemento del primer array, el cuál es un 1.
   - ☐ 2true3

3. **¿Qué enunciados son verdaderos?**
   - ☐ An interface CANNOT be extended by another interface.
   - ☑ An abstract class can be extended by a concrete class. Una clase abstracta puede ser extendida mediante una clase concreta, la cual implementará los métodos de la clase abstracta.
   - ☐ An abstract class CANNOT be extended by an abstract class.
   - ☐ An interface can be extended by an abstract class.
   - ☑ An abstract class can implement an interface. La clase abstracta debe proporcionar las implementaciones para los métodos de la interfaz o dejar que sus subclases lo hagan.
   - ☐ An abstract class can be extended by an interface.

4. ¿Cuáles métodos, insertados independientemente en la línae 5, compilarán? (Escoge 5)

```
1 public class Blip{
2       protected int blipvert(int x){ return 0
3 }
4 class Vert extends Blip{
5       //insert code here
6 }
```

- ☐ Private int blipvert(long x) { return 0; }
- ☑ Protected int blipvert(long x) { return 0; } Al ser protected, permite sobrescribir el método, además cuenta con un parámetro diferente, lo que evita que tenga conflicto con la superclase.
- ☑ Protected long blipvert(int x, int y) { return 0; } Al tener dos parámetros, nos dará un retorno diferente lo que no afecta la sobrecarga.
- ☑ Public int blipvert(int x) { return 0; } El método tiene el mismo nombre y tipo de parámetro que el método protected, por lo que se puede sobrescribir en la superclase, y al hacerlo public brinda un acceso más amplio.
- ☐ Private int blipvert(int x) { return 0; }
- ☐ Protected long blipvert(int x) { return 0; }
- ☑ Protected long blipvert(long x) { return 0; } Al tener un tipo de parámetro diferente, la sobrecarga es válida.

5. Problema

```
Given:

1. class Super{
2.    private int a;
3.    protected Super(int a){ this.a = a; }
4. }
...
11. class Sub extends Super{
12.   public Sub(int a){ super(a);}
13.   public Sub(){ this.a = 5;}
14. }
Which two independently, will allow Sub to compile? (Choose two)
```

- ☐ Change line 2 to: public int a;
- ☑ Change line 13 to: public Sub(){ super(5);} Esto llamará a Super con el valor 5, se llama de forma correcta, por lo que compilará sin problema.
- ☐ Change line 2 to: protected int a;
- ☑ Change line 13 to: public Sub(){ this(5);}
- ☐ Change line 13 to: public Sub(){ super(a);}

6. ¿Qué es cierto acerca de la clase Wow?

```
public abstract class Wow {
        private int wow;
        public Wow(int wow) { this.wow = wow; }
        public void wow() { }
        private void wowza() { }
}
```

- ☐ It does not compile because an abstract class cannot have private methods
- ☐ It does not compile because an abstract class cannot have instance variables.
- ☑ It compiles without error. Compilará sin error ya que, el int wow es privado y solo se puede accesar dentro de la misma clase, el constructos es public, por lo que todo lo que pueda acceder a la clase wow puede instanciarlo, pero al ser wow privado, solo se puede mediante una subclase. El método wowza no será sobrescrito en las subclases.
- ☐ It does not compile because an abstract class must have at least one abstract method.
- ☐ It does not compile because an abstract class must have a constructor with no arguments.

7. ¿Cuál será el resultado?

```
class Atom {
        Atom() { System.out.print("atom "); }
}
class Rock extends Atom {
        Rock(String type) { System.out.print(type); }
}
public class Mountain extends Rock {
        Mountain() {
                super("granite ");
                new Rock("granite ");
        }
        public static void main(String[] a)  { new Mountain(); }
}
```

- ☑ Compilation fails. Para que compile, se deben hacer Rock y Mountain clases estáticas, o en su defecto, crear una instancia de Atom, para luego crear instancias de Rock y Mountain.
- ☐ Atom granite.
- ☐ Granite granite.
- ☐ Atom granite granite.
- ☐ An exception is thrown at runtime
- ☑ Atom granite atom granite

8. ¿Qué se imprime cuando se ejecuta?

```
public class MainMethod {
        void main() {
                System.out.println("one");
        }
        static void main(String args) {
                System.out.println("two");
        }
        public static final void main(String[] args) {
                System.out.println("three");
        }
        void mina(Object[] args) {
                System.out.println("four");
        }
}
```

- ☐ one
- ☐ two
- ☑ three. Solo éste método está definido correctamente, los demás presentan algún error en sus sintaxis.
- ☐ four
- ☐ There is no output.

9. ¿Cuál será el resultado?

```
import java.util.*;
public class MyScan {
        public static void main(String[] args) {
                String in = "1 a 10 . 100 1000";
                Scanner s = new Scanner(in);
                int accum = 0;
                for (int x = 0; x < 4; x++) {
                        accum += s.nextInt();
                }
                System.out.println(accum);
        }
}
```

- ☐ 11
- ☐ 111
- ☑ 1111 Dentro del bucle for, se itera cuatro veces lo que hay en s, que son los cuatro números declarados en in. Por lo tanto, se van a sumar 1+10+100+1000
- ☐ An exception is thrown at runtime.

10. ¿Qué enunciado es cierto?

```
class ClassA {
        public int numberOfInstances;
        protected ClassA(int numberOfInstances) {
                this.numberOfInstances = numberOfInstances;
        }
}
public class ExtendedA extends ClassA {
        private ExtendedA(int numberOfInstances) {
                super(numberOfInstances);
        }
        public static void main(String[] args) {
                ExtendedA ext = new ExtendedA(420);
                System.out.print(ext.numberOfInstances);
        }
}
```

- ☑ 420 is the output. Se crea la instancia de ExtendedA con numberOfInstances que es inicializado en 420, y luego se manda a imprimir dicho valor.
- ☐ An exception is thrown at runtime.
- ☐ All constructors must be declared public.
- ☐ Constructors CANNOT use the private modifier.
- ☐ Constructors CANNOT use the protected modifier.

11. ¿Qué permite el patrón Singleton? :

- ☐ Have a single instance of a class and this instance cannot be used by other classes
- ☑ Having a single instance of a class, while allowing all classes have access to that instance. Asegura que una clase tenga una única instancia y proporciona un punto de acceso global a esa instancia.
- ☐ Having a single instance of a class that can only be accessed by the first method that calls it.

12. ¿Cuál es el resultado?

```
import java.text.*;
public class Align {
        public static void main(String[] args) throws ParseException {
                String[] sa = {"111.234", "222.5678"};
                NumberFormat nf = NumberFormat.getInstance();
                nf.setMaximumFractionDigits(3);
                for (String s : sa) { System.out.println(nf.parse(s)); }
        }
}
```

- ☐ 111.234 222.567
- ☐ 111.234 222.568 Se analizan las cadenas con NumberFormat y las convierte en números, como se tiene una condición o configuración con el setMaximumFractionDigits a 3, solo se podrán tener número con tres dígitos
- ☑ 111.234 222.5678
- ☐ An exception is thrown at runtime.

13. What is the result?

Given

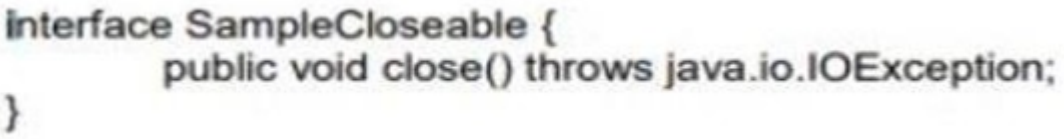
```
public class SuperTest {
        public static void main(String[] args) {
                //statement1
                //statement2
                //statement3
        }
}
class Shape {
        public Shape() {
                System.out.println("Shape: constructor");
        }
        public void foo() {
                System.out.println("Shape: foo");
        }
}
class Square extends Shape {
        public Square() {
                super();
        }
        public Square(String label) {
                System.out.println("Square: constructor");
        }
        public void foo() {
                super.foo();
        }
        public void foo(String label) {
                System.out.println("Square: foo");
        }
}
```

What should statement1, statement2, and statement3, be respectively, in order to produce the result?

Shape: constructor
Shape: foo
Square: foo

- ☐ Square square = new Square ("bar"); square.foo ("bar"); square.foo();
- ☐ Square square = new Square ("bar"); square.foo ("bar"); square.foo ("bar");
- ☐ Square square = new Square (); square.foo (); square.foo(bar);
- ☑ Square square = new Square (); square.foo (); square.foo("bar"); El statement1 crea la instancia Square. El statement 2 llamará al método foo de la instancia Square, que a su vez llama al método foo de shape, produciendo Shape : foo. Para el statement 3 se llama al método foo(bar) de Square, lo que producirá el Square : foo.
- ☑ Square square = new Square (); square.foo (); square.foo ();

14. ¿Qué tres implementaciones son válidas?



```
interface SampleCloseable {
        public void close() throws java.io.IOException;
}
```

- ☑ class Test implements SampleCloseable { public void close() throws java.io.IOException { // do something } } IOException es válida para lanzarse SampleCoseable declara que el método close la puede lanzar.
- ☐ class Test implements SampleCloseable { public void close() throws Exception { // do something } }
- ☑ class Test implements SampleCloseable { public void close() throws FileNotFoundException { // do something } } FileNotFoundException es una excepción específica de IOException, por lo que es válida mientras SampleCloseable la declare
- ☐ class Test extends SampleCloseable { public void close() throws java.io.IOException { // do something } }
- ☑ class Test implements SampleCloseable { public void close() { // do something }} Si SampleCoseable no declara ninguna excepción para el método close(), no es necesario implementar alguna excepción.

15. Un _____ proporciona una solución a un problema de diseño. Debe cumplir con diferentes características, como la efectividad al resolver problemas similares en ocasiones anteriores. Por lo tanto, debe de ser re-utilizable, es decir, aplicable a diferentes problemas en diferentes circunstancias.

- ☐ Reutilizando código
- ☐ Anti-patrón
- ☐ Metodología
- ☑ Patrón de diseño. Es una solución general y reutilizable para problemas comunes que se presentan durante el diseño de software.

16. ¿Cuál es el resultado?

```
class MyKeys {
        Integer key;
        MyKeys(Integer k) { key = k; }
        public boolean equals(Object o) {
                return ((MyKeys) o).key == this.key;
        }
}
```

And this code snippet:

```
Map m = new HashMap();
MyKeys m1 = new MyKeys(1);
MyKeys m2 = new MyKeys(2);
MyKeys m3 = new MyKeys(1);
MyKeys m4 = new MyKeys(new Integer(2));
m.put(m1, "car");
m.put(m2, "boat");
m.put(m3, "plane");
m.put(m4, "bus");
System.out.print(m.size());
```

- ☐ 2. Se crean las instancias de m1 a m4, se procede a insertar el HashMap que añade una entrada de la m1 a la m4 y les asocia un valor de string a cada una, m1 será igual que m3, y m2 será igual a m4, por lo que, ahora HashMap tendrá dos entrada m1/m3 con key de 1 y m2/m4 con key de 2. Por ende, al imprimir el tamaño de HashMap, el resultado será 2, porque solo tiene dos valores.
- ☐ 3
- ☑ 4
- ☐ Compilation fails

17. ¿Qué valor de x, y, z dará el siguiente resultado? 1234,1234,1234 -----, 1234, ------

```
public static void main(String[] args) {
        // insert code here
        int j = 0, k = 0;
        for (int i = 0; i < x; i ++) {
                do {
                        k = 0;
                        while (k < z) {
                                k++;
                                System.out.print(k + " ");
                        }
                        System.out.println(" ");
                        j++;
                } while (j < y);
                System.out.println("---");
        }
}
```

- ☐ int x = 4, y = 3, z = 2;
- ☐ int x = 3, y = 2, z = 3;
- ☐ int x = 2, y = 3, z = 3;
- ☑ int x = 2, y = 3, z = 4; x para generar los dos bloque separados por guiones, y para imprimir 1234 tres veces por bloque, z para imprimir 1234 que tienen cuatro caracteres.
- ☐ int x = 4, y = 2, z= 3;

18.

```
Given:

class Alpha{ String getType(){ return "alpha";}}
class Beta extends Alpha{String getType(){ return "beta";}}
public class Gamma  extends Beta { String getType(){ return "gamma";}
      public static void main(String[] args) {
        Gamma g1 = (Gamma) new Alpha();
        Gamma g2 = (Gamma) new Beta();
        System.out.print(g1.getType()+ " " +g2.getType());
    }
}
What is the result?
```

- ☐ Gamma gamma
- ☐ Beta beta
- ☐ Alpha beta
- ☑ Compilation fails. No se compilará debido a que no se puede convertir directamente Alpha o Beta a una Gamma, ya que no es una superclase y no se están casteando.

19. ¿Cuáles son las tres malas prácticas?

- ☐ Checking for an IOException and ensuring that the program can recover if one occurs.
- ☑ Checking for ArrayIndexOutOfBoundsException and ensuring that the program can recover if one occurs. Es una mala práctica porque es una excepción no corroborada al no corroborar los índices de los arrays.
- ☐ Checking for FileNotFoundException to inform a user that a filename entered is not valid.
- ☑ Checking for Error and, if necessary, restarting the program to ensure that users are unaware problems. Intentar capturar Error y reiniciar el programa es generalmente una mala práctica, ya que podría ocultar problemas graves y llevar a un comportamiento impredecible.
- ☑ Checking for ArrayIndexOutOfBoundsException when iterating through an array to determine when all elements have been visited. En su lugar se recomienda usar bucles adecuados que eviten estas excepciones.

20. ¿Cuál es el resultado

```
public class Test {
        public static void main(String[] args) {
                int b = 4;
                b--;
                System.out.print(--b);
                System.out.println(b);
        }
}
```

- ☑ 2 2 El b - - es un postdecremento, por lo que el 4 se convierte en un 3, posteriormente se le aplica un predecremento con el –b, entonces la ser 3 pasa a ser 2 y ese es el primer valor impreso. Para el segundo valor b sigue siendo un 2, ya que no vuelve a sufrir modificaciones.
- ☐ 1 2
- ☐ 3 2
- ☐ 3 3

21. ¿Cuál es la diferencia entre throws y throw?:

- ☐ Throws throws an exception and throw indicates the type of exception that the method.
- ☐ Throws is used in methods and throw in constructors.
- ☑ Throws indicates the type of exception that the method does not handle and throw an exception. Throw se utiliza para lanzar una excepción dentro de un métodos, mientras que trhows se declara en el método para así lanzar ciertas excepciones.

22. ¿Cuál es el resultado?

```
class Feline {
        public String type = "f ";
        public Feline() {
                System.out.print("feline ");
        }
}
public class Cougar extends Feline {
        public Cougar() {
                System.out.print("cougar ");
        }
        void go() {
                type = "c ";
                System.out.print(this.type + super.type);
        }
        public static void main(String[] args) {
                new Cougar().go();
        }
}
```

- ☐ Cougar c f.
- ☑ Feline cougar c c.
- ☑ Feline cougar c f. Al crear al Cougar, el constructor de Feline, regresará un feline, después el constructor Cougar imprimirá un cougar, por último en el go(), llama al tipo c de la clase Cougar y una f como el super proveniente de feline.
- ☐ Compilation fails.

23. Which statement, when inserted into line " // TODO code application logic here", is valid in compilation time change?

```
public class SampleClass {
        public static void main(String[] args) {
                AnotherSampleClass asc = new AnotherSampleClass();
                SampleClass sc = new SampleClass();
                // TODO code application logic here
        }
}
class AnotherSampleClass extends SampleClass { }
```

- ☐ asc = sc;
- ☑ sc = asc; sc = asc; asigna la instancia asc a la variable sc. Y como AnotherSampleClass hereda de SampleClass, la asignación es válida
- ☐ asc = (Object) sc;
- ☐ asc= sc.clone();

24. ¿Cuál es el resultado?

```
public class Test {
        public static void main(String[] args) {
                int[][] array = { {0}, {0,1}, {0,2,4}, {0,3,6,9}, {0,4,8,12,16} };
                System.out.println(array[4][1]);
                System.out.println(array[1][4]);
        }
}
```

- ☐ 4 Null.
- ☐ Null 4.
- ☐ An IllegalArgumentException is thrown at run time.
- ☑ 4 An ArrayIndexOutOfBoundsException is thrown at run time. El programa imprimirá primero un 4, debido al elemento mencionado en el primer System.out, y después lanzará la excepción de Array fuera de los límites porque el array al que se refiere en el segundo System.out, no tiene esas dimensiones.

25. ¿Cuál es el resultado?

```
import java.util.*;
public class App {
        public static void main(String[] args) {
                List p = new ArrayList();
                p.add(7);
                p.add(1);
                p.add(5);
                p.add(1);
                p.remove(1);
                System.out.println(p);
        }
}
```

- ☐ [7, 1, 5, 1]
- ☑ [7, 5, 1] Dentro del código se añaden 7, 1, 5 y 1 con el p.add. Posterior a esto, con p.remove se quita el elemento del índice 1, por lo que el primer 1 se elimina, dejando así 7, 5, 1.
- ☐ [7, 5]
- ☐ [7, 1]

26. ¿Cuáles son las tres líneas que al ejecutarse imprimirán "Right on!"?

```
13.    public class Speak {
14.        public static void main(String[] args) {
15.                Speak speakIT = new Tell();
16.                Tell tellIt = new Tell();
17.                speakIT.tellItLikeItIs();
18.                (Truth) speakIT.tellItLikeItIs();
19.                ((Truth) speakIT).tellItLikeItIs();
20.                tellIt.tellItLikeItIs();
21.                (Truth) tellIt.tellItLikeItIs();
22.                ((Truth) tellIt).tellItLikeItIs();
23.        }
24.    }

class Tell extends Speak implements Truth {
        @Override
        public void tellItLikeItIs() {
                System.out.println("Right on!");
        }
}

interface Truth {
        public void tellItLikeItIs();
}
```

- ☐ Line 17
- ☐ Line 18
- ☑ Line 19 SpeakIt es casteado a Truth y el método tellItLikeItIs es implementado, y como speakIt es el objeto Tell, va a invocar al TellItLikeItIs.
- ☑ Line 20 Hace referencia al objeto Tell que llama directamente al método TellItLikeItIs.
- ☐ Line 21
- ☑ Line 22 Castea el TellIt al Truth, y llama al método TellItLikeItIs.

27. ¿Cuál es el resultado?

```
public class Bees {
        public static void main(String[] args) {
                try {
                        new Bees().go();
                } catch (Exception e) {
                        System.out.println("thrown to main");
                }
        }
        synchronized void go() throws InterruptedException {
                Thread t1 = new Thread();
                t1.start();
                System.out.print("1 ");
                t1.wait(5000);
                System.out.print("2 ");
        }
}
```

- ☐ The program prints 1 then 2 after 5 seconds.
- ☑ The program prints: 1 thrown to main. Arrojará 1, porque IllegalMonitorStateException es arrojado cuando el método wait es mal llamado
- ☐ The program prints: 1 2 thrown to main.
- ☐ The program prints:1 then t1 waits for its notification.

28. ¿Cuáles son las tres opciones válidas?

```
class ClassA {}
class ClassB extends ClassA {}
class ClassC extends ClassA {}
And:
ClassA p0 = new ClassA();
ClassB p1 = new ClassB();
ClassC p2 = new ClassC();
ClassA p3 = new ClassB();
ClassA p4 = new ClassC();
```

- ☑ p0 = p1;
- ☐ p1 = p2;
- ☑ p2 = p4;
- ☐ p2 = (ClassC)p1;
- ☑ p1 = (ClassB)p3; p3 es de tipo ClassA e intenta castear a ClassB, dado que p3 se inicializó con la ClassB, el cast es válido así como las asignación.
- ☑ p2 = (ClassC)p4; p4 es de tipo ClassA e intenta castear a ClassC, como p4 se inicializó en ClassC, el cast es válido.

29. ¿Cuáles son las opciones que describen correctamente la relación de clases?

```
class Class1 { String v1; }
class Class2 {
        Class1 c1;
        String v2;
}
class Class3 { Class2 c1; String v3; }
```

- ☐ Class2 has-a v3.
- ☐ Class1 has-a v2.
- ☑ Class2 has-a v2. Tiene una variable v2.
- ☑ Class3 has-a v1.
- ☐ Class2 has-a Class3.
- ☑ Class2 has-a Class1. Class2 si tiene una variable Class1

30. ¿Cuál es el resultado?

```
class MySort implements Comparator<Integer> {
        public int compare(Integer x, Integer y) {
                return y.compareTo(x);
        }
}

And the code fragment:

Integer[] primes = {2, 7, 5, 3};
MySort ms = new MySort();
Arrays.sort(primes, ms);
for (Integer p2 : primes) { System.out.print(p2 + " "); }
```

- ☐ 2 3 5 7
- ☐ 2 7 5 3
- ☑ 7 5 3 2. El comparador MySort, va a comparar los valores en forma descendente, por lo que los ordenará de mayor a menor.
- ☐ Compilation fails.

31. ¿Cuáles son los dos posibles outputs?

```
public class Main {
        public static void main(String[] args) throws Exception {
                doSomething();
        }
        private static void doSomething() throws Exception {
                System.out.println("Before if clause");
                if (Math.random() > 0.5) { throw new Exception();}
                System.out.println("After if clause");
        }
}
```

- ☑ Before if clause Exception in thread "main" java.lang.Exception at Main.doSomething (Main.java:21) at Main.main (Main.java:15). Este resultado ocurre si la condición if es verdadera y se lanza una excepción. El mensaje se imprimirá antes de que se lance la excepción y luego se mostrará dónde ocurrió.
- ☐ Before if clause Exception in thread "main" java.lang.Exception at Main.doSomething (Main.java:21) at Main.main (Main.java:15) After if clause.

- ☐ Exception in thread "main" java.lang.Exception at Main.doSomething (Main.java:21) at Main.main (Main.java:15).
- ☑ Before if clause After if clause. Ocurre si el if es falso, en este caso se imprimirá el Before y no lanza ninguna excepción, después se imprimirá el After.

32. ¿Cuál es el resultado?

```
public static void main(String[] args) {
        String color = "Red";
        switch (color) {
        case "Red":
                System.out.println("Found Red");
        case "Blue":
                System.out.println("Found Blue");
        case "White":
                System.out.println("Found White");
                break;
        Default:
                System.out.println("Found Default");
        }
}
```

- ☐ Found Red.
- ☐ Found Red Found Blue.
- ☑ Found Red Found Blue Found White. Como el color es Red, imprimirá el Found Red, al no encontrarse con algún break, continuará e imprimira el Found Blue y el White, aquí se encuentra con un break, por lo que detnedrá la impresión y no llegará al Default.
- ☑ Found Red Found Blue Found White Found Default.

33. ¿Qué sentencias son verdaderas?

- ☑ An abstract class can implement an interface. Una clase abstracta puede implementar una o más interfaces
- ☐ An abstract class can be extended by an interface.
- ☐ An interface CANNOT be extended by another interface.
- ☐ An interface can be extended by an abstract class.

- ☑ An abstract class can be extended by a concrete class. Una clase concreta puede extender una clase abstracta y debe proporcionar implementaciones para todos los métodos abstractos definidos en la clase abstracta.
- ☐ An abstract class CANNOT be extended by an abstract class

34. ¿Cuál es el resultado?

```
class X {
        static void m(int i) {
                i += 7;
        }
        public static void main(String[] args) {
                int i = 12;
                m(i);
                System.out.println(i);
        }
}
```

- ☐ 7
- ☑ 12. El método m trabaja con una copia del valor de i, por lo que el verdadero valor de i, queda intacto y no se ve afectado por el método.
- ☐ 19
- ☐ Compilation fails.
- ☐ An exception is thrown at run time

35. ¿Cuál es la verdadera?

```
5.  class Building { }
6.      public class Barn extends Building {
7.          public static void main(String[] args) {
8.              Building build1 = new Building();
9.              Barn barn1 = new Barn();
10.             Barn barn2 = (Barn) build1;
11.             Object obj1 = (Object) build1;
12.             String str1 = (String) build1;
13.             Building build2 = (Building) barn1;
14.         }
15.     }
```

Which is true?

- ☐ If line 10 is removed, the compilation succeeds.
- ☐ If line 11 is removed, the compilation succeeds.
- ☑ If line 12 is removed, the compilation succeeds. build1 es una instancia de Building, por ende no puede ser convertida a string. Esto producirá un error de compilación.
- ☐ If line 13 is removed, the compilation succeeds.
- ☐ More than one line must be removed for compilation to succeed.

36. ¿Cuál es el resultado si el integer es 33?

```
public static void main(String[] args) {
        if (value >= 0) {
                if (value != 0) {
                        System.out.print("the ");
                } else {
                        System.out.print("quick ");
                }
                if (value < 10) {
                        System.out.print("brown ");
                }
                if (value > 30) {
                        System.out.print("fox ");
                } else if (value < 50) {
                        System.out.print("jumps ");
                } else if (value < 10) {
                        System.out.print("over ");
                } else {
                        System.out.print("the ");
                }
                if (value > 10) {
                        System.out.print("lazy ");
                } else {
                        System.out.print("dog ");
                }
                System.out.print("... ");
        }
}
```

- ☑ The fox jump lazy... Al tener el valor de 33, es != 0, por lo que imprime the, es >30 por lo que se imprime fox, es <50 por lo que imprime jump, es >10 por lo que imprime lazy y los … se imprimen sin condición.
- ☐ The fox lazy...
- ☐ Quick fox over lazy...

- ☐ Quick fox the…

37. ¿Cuál es el resultado?

```
11. class Person {
12. String name  = "No name";
13. public Person (String nm) { name = nm}
14. }
15.
16. class Employee extends Person {
17.  String empID = "0000";
18.  public Employee (String id) { empID "
id; }
19. }
20.
21. public class EmployeeTest {
22.  public static void main(String[] args)
{
23.   Employee e = new Employee("4321");
24.   System.out.println(e.empID);
25.  }
26. }
```

- ☐ 4321.
- ☐ 0000.
- ☐ An exception is thrown at runtime.
- ☑ Compilation fails because of an error in line 18.

38. ¿Qué parte del código es ilegal?

- ☐ Class Base1 { abstract class Abs1 { } }
- ☐ Abstract class Abs2 { void doit() { } }
- ☐ class Base2 { abstract class Abs3 extends Base2 { } }
- ☑ class Base3 { abstract int var1 = 89; } No se pueden declarar campos abstractos, pueden ser final, static, etc.

39. ¿Cuál es el resultado?

```
public static void main(String[] args) {
        System.out.println("Result: " + 2 + 3 + 5);
        System.out.println("Result: " + 2 + 3 * 5);
}
```

- ☐ Result: 10 Result: 30
- ☐ Result: 25 Result: 10
- ☑ Result: 235 Result: 215 Para el primer valor impreso, se concatenan los números, dando 235. Para el segundo valor, primero realiza la operación 3*5 dando un 15, posterior a eso se concatena al 2 obteniendo un 215.
- ☐ Result: 215 Result: 215
- ☐ Compilation fails.

40. ¿Cuál es el reemplazo para foo?

```
public static void main(String[] args) {
        Boolean b1 = true;
        Boolean b2 = false;
        int i = 0;
        while (foo) { }
}
```

- ☐ b1.compareTo(b2)
- ☐ i = 1
- ☐ i == 2? -1:0
- ☑ foo.equals("bar")

Fin en página 144