Parcial Teórico

1. **¿Cómo se declara una clase que implementa una interfaz?**

a) Class **nombre_comun** implements **Interfaz1**

b) Import class **nombre_comun** implements **Interfaz1**

c) Public class **nombre_comun** extends **Interfaz1**

d) Public class **nombre_comun** implements **Interfaz1**

e) Public interface **Interfaz1** extends **nombre_comun**

2. ¿Qué son los modificadores de acceso y cuantos existen?

3. ¿A qué fundamento de la poo hace referencia la siguiente línea de código?

```
public clase1(int a, String b, boolean c, char d) {
    this.a = a;
    this.b = b;
    this.c = c;
    this.d = d;
}
no usages
public clase1(boolean c, char d) {
    this.a = 3;
    this.b = "Hola como estas";
    this.c = c;
    this.d = d;
}
```

a. Herencia.
b. Polimorfismo.
c. Abstracción.
d. Encapsulamiento.

4. ¿Para qué sirve una interfaz?

a. Para crear atributos y métodos que luego heredarán las clases hijas pero sin necesidad de instanciar ningún objeto.

b. Para modelar un objeto de la vida real y llevarlo al código.

c. Para crear métodos que puedan ser implementados por distintas clases y que estas no estén obligadas a heredar de la misma clase padre.

d. Para poder implementar un programa que no es nuestro sin necesidad de conocer su funcionamiento interno.

5. ¿Qué permite la abstracción en el diseño de software?

a. definir múltiples métodos con el mismo nombre pero diferenciándose en los parámetros.

b. Simplificar sistemas complejos al mostrar solo lo esencial.

c. Heredar métodos y atributos de una clase a otra.

d. Crear objetos sin definir su tipo específico.

e. Proteger partes del código haciendo que solo sea posible ser accedido a ellos en determinados lugares y no en la totalidad del código.

6. ¿Un Enumerador puede tener constructor?

a. si, pero debe ser privado.

b. no, solo se ponen las palabras.

c. si, es igual a cualquier constructor.

d. no, los Enums son constantes solo se igualan.

7. Nombrar los fundamentos de la POO y explicar cada uno, de forma concreta.

8. ¿Las funciones, como en C, y los métodos son lo mismo? ¿Por qué?

a. Verdadero.

b. Falso.

9. ¿Cual es la diferencia entre un atributo y un método declarado como static?

a. Ninguna, son maneras de hacer lo mismo.

b. Un método no puede ser declarado como static.

c. Un atributo no puede ser declarado como static.

d. Un atributo static es aquel que puede usarse sin instanciar un objeto de una clase mientras que un método static es aquel que permanece igual para cada instancia de ese objeto.

e. Un método static es aquel que puede usarse sin instanciar un objeto de una clase mientras que un atributo static es aquel que permanece igual para cada instancia de ese objeto.

10. ¿Cómo se declara una clase abstracta en java?

a. public class **Clase10** implements Abstract.

b. private abstract class **Clase10**.

c. public abstract class **Clase10**.

d. private class **Clase10** extends Abstract.

e. public class abstract **Clase10**.

11. ¿Una clase abstracta necesita que todos sus métodos sean abstractos?

a. Verdadero, pero no es necesario que las clases hijas implementen todos sus métodos.
b. Verdadero, pero los métodos que no implementen sus clases hijas deberán tener un cuerpo en la clase padre.
c. Falso, sin embargo, sus clases hijas deberán implementar todos sus métodos, sean abstractos o no.
d. Falso, sin embargo, sus métodos abstractos no deben tener un cuerpo.

12. ¿Qué palabra/s utilizamos cuando queremos que ninguna clase pueda heredar (ser hija) de cierta clase?

a. final.
b. abstract.
c. no hereditary.
d. private.
e. Inheritance.
f. static.

13. ¿Cómo se puede aplicar el principio de encapsulamiento para mejorar la mantenibilidad de un sistema?

a. Heredando de múltiples clases para compartir comportamiento.

b. Permitiendo el acceso directo a todos los atributos de la clase

c. Restringiendo el acceso directo a los atributos y proporcionando métodos de acceso y modificación.

d. Usando sólo métodos estáticos en las clases.

14. De las siguientes Collection, ¿Cuál permite elementos duplicados? (puede elegir más de una si lo necesita)

a. Map (en ambos atributos).

b. List.

c. Set.

d. Map (solo en la Key).

15. ¿Cómo se define el método constructor en una clase “CLASE15” en Java?

a. public **CLASE15**().
b. public create **CLASE15**().
c. constructor **CLASE15**().
d. public static CLASE15 **crearObjeto**().
e. public constructor **CLASE15**().

16. ¿Cual es una desventaja de usar herencia y cuál es su solución?

a. Los métodos y atributos creados en la clase padre pueden ser utilizados por las clases hijas. La solución es crear una clase envolvente.

b. Hay métodos que necesitan estar en dos clases pero que estas no tienen nada que ver por lo que no pueden heredar de lo mismo. La solución es usar una interfaz.

c. Las clases de un mismo padre no compartirán nada y por ello no podrán relacionarse entre sí. La solución es usar una interfaz.

d. Los nombres de las clases hijas podrán resultar muy confusos haciendo que se pierda el significado específico del programa realizado. La solución es usar un enum.

17. ¿Qué es una Collection en Java?

a. Una clase abstracta para manejar datos.

b. Una interfaz para manejar conexiones de red.

c. Una interfaz que representa un grupo de objetos.

d. una clase que implementa la interfaz Map.

18. ¿Cuál es la principal diferencia entre un atributo y un método?

a. Los métodos definen el comportamiento mientras que los atributos definen las características.
b. Los atributos definen el comportamiento mientras que los métodos definen las características.
c. Los métodos definen variables locales y los atributos son esas variables.
d. Los atributos son creados dentro de los métodos para facilitar el comportamiento de estos y estos últimos definen el comportamiento del objeto modelado.

19. ¿Cual es la principal diferencia entre una clase abstracta y una interfaz?

a. Una clase abstracta no puede tener métodos implementados, mientras que una interfaz sí puede.

b. Una clase puede heredar múltiples clases abstractas, pero solo puede implementar una interfaz.

c. Una clase puede implementar múltiples interfaces pero solo puede heredar de una clase abstracta.

d. Una interfaz puede tener constructores, mientras que una clase abstracta no puede.

20. ¿Para qué sirve la palabra reservada “super”?

a. Sirve para definir un único atributo primario de una clase.

b. Para únicamente llamar al constructor de la clase padre.

c. Sirve para definir uno o más atributos primarios en una clase.

d. Para crear una clase padre de las que hereden todas las clases de java sin necesidad de usar “extends”.

e. Sirve para referirse a la clase padre.

21. ¿Cuál de estas opciones es una característica de la Collection “Map”?

a. Hacen referencia a una lista doblemente enlazada.

b. Contienen dos tipos de datos dentro, uno que no se puede repetir y otro si.

c. Carecen del método *get()***.**

d. La clave y el valor no pueden ser el mismo dato.

e. Los valores dentro del mapa deben ser si o si valores primitivos.

22. ¿Qué sucede cuando se declara un método *static* como *final*?

a. No puede ser llamado desde la subclase.

b. Solo pueden ser usados atributos o métodos estáticos dentro de él.

c. Se lanza una excepción en tiempo de compilación.

d. no puede ser sobreescrito ni redefinido en una subclase.

23. ¿Cuándo es útil usar una Collection Set?

a. Cuando quiero elementos repetidos.

b. Cuando quiero que los elementos tengan un índice.

c. Cuando quiero que los elementos están separados por un dato único e irrepetible.

d. Cuando quiero elementos están ordenados por una tabla hash.

e. Cuando quiero elementos que no se puedan repetir.

24. ¿En un Arraylist es posible guardar datos nulos y directamente datos primitivos?

a. Puedes guardar nulos pero no datos primitivos.

b. En un Arraylist no pueden guardarse ninguno de los dos

c. Puedes guardar datos primitivos pero no nulos.

d. En un Arraylist pueden guardarse ambos sin problema.

25. ¿Qué es una variable local?

a. Una variable declarada dentro de una clase.

b. Una variable declarada dentro de un método.

c. Una variable que comparten todas las instancias de la clase.

d. Una variable que solo puede ser leída, no modificada.

e. Una variable que es siempre pública.

26. ¿Es posible que una clase no implemente todos los métodos de una interfaz?

a. No, es obligatorio que implemente todos los métodos.

b. Si, dentro de la interfaz, se deben declarar abstractos los métodos que no sean obligatorios.

c. No, eso lanzaría una excepción del tipo “NoInterfaceCompletedExeption”.

d. Si, pero la clase debe ser abstracta.

e. Nunca es necesario implementar los métodos de una interfaz.

27. ¿Es posible que los Enumeradores tengan métodos?¿Por qué?

Verdadero.

Falso.

28. ¿Cuándo es apropiado utilizar un enumerador?

a. Cuando se necesita definir una clase abstracta.

b. Cuando se necesita definir una clase común.

c. Cuando se necesita implementar una interfaz.

d. Cuando se necesita definir métodos estáticos.

e. Cuando se necesita definir un grupo de valores constantes relacionados.

29. ¿Qué es una clase en programación orientada a objetos?

a. Una colección de variables locales.

b. Una forma de agrupar métodos en nuestro proyecto.

c. Un método de ordenamiento.

d. Un modelo o plantilla para crear objetos.

30. ¿Qué es la sobrecarga de métodos en Java?

a. Cuanto un método que carga una colección de datos y este sobrepasa el límite.

b. Crear un método sin ningún parámetro.

c. Usar un método de una clase padre en una clase hija.

d. Crear un método con el mismo nombre en dos clases diferentes.

e. Declarar dos métodos con el mismo nombre y diferentes parámetros.

Respuestas

1. d.

2. Los modificadores de acceso son palabras reservadas que forman parte del fundamento de encapsulamiento. Estas palabras sirven para poder restringir el acceso a métodos o atributos, esto con el fin de que únicamente puedan ser modificados o usados cuando el programa lo requiera. Las modificadores son 4 y son default, public, protected y private.

3. b.

4. c.

5. b.

6. a. (nota: aunque el IDE (intellij) te diga que ponerle private es innecesario es porque no es posible acceder al constructor de ningún lado y eso es básicamente un atributo privado por definición).

7. **Abstracción** : proceso de abstracción permite seleccionar las características relevantes dentro de un conjunto e identificar comportamientos comunes para definir nuevos tipos de entidades en el mundo real.
   Herencia: Las clases no se encuentran aisladas, sino que se relacionan entre sí, formando una jerarquía de clasificación. Los objetos heredan las propiedades y el comportamiento de todas las clases a las que pertenecen.
   Polimorfismo: Un mismo método -o acción- puede tener el mismo nombre pero un comportamiento diferente para el mismo o diferentes objetos. Se diferencian con la cantidad y tipos de parámetros de entrada pero el retorno (y el nombre) es el mismo.
   Encapsulamiento: Se refiere a la idea de ocultar los detalles internos de una clase y sólo exponer una interfaz pública mediante la cual otras clases pueden interactuar con ella. Esto se logra mediante el uso de modificadores de acceso (como private, public, protected) para controlar el acceso a los atributos y métodos de la clase.

8. Falso. Mientras que las funciones sirven para guardar pequeñas partes del código y luego poder usarlas a lo largo de nuestro programa principal los métodos definen los comportamientos de los objetos que estamos modelando y únicamente sirven para el contexto de ese mismo objeto.

9. e.

10. c. (nota: puede ser que en el id no se ponga “public” pero es porque es un caso similar a mi nota anterior, ponerlo como no ponerlo es en sí, innecesario).

11. d.

12. a.

13. c.

14. b. (nota: los map solo pueden tener duplicados en el value, era todo una trampa).

15. a.

16. b.

17. c.

18. a.

19. c.

20. e.

21. b.

22. d.

23. e.

24. a.

25. b.

26. d.

27. Verdadero. Se puede definir métodos dentro de un enum y cada constante del enum puede implementar estos métodos de forma individual. Esto permite que cada constante tenga un comportamiento específico.

28. e.

29. d.

30. e.