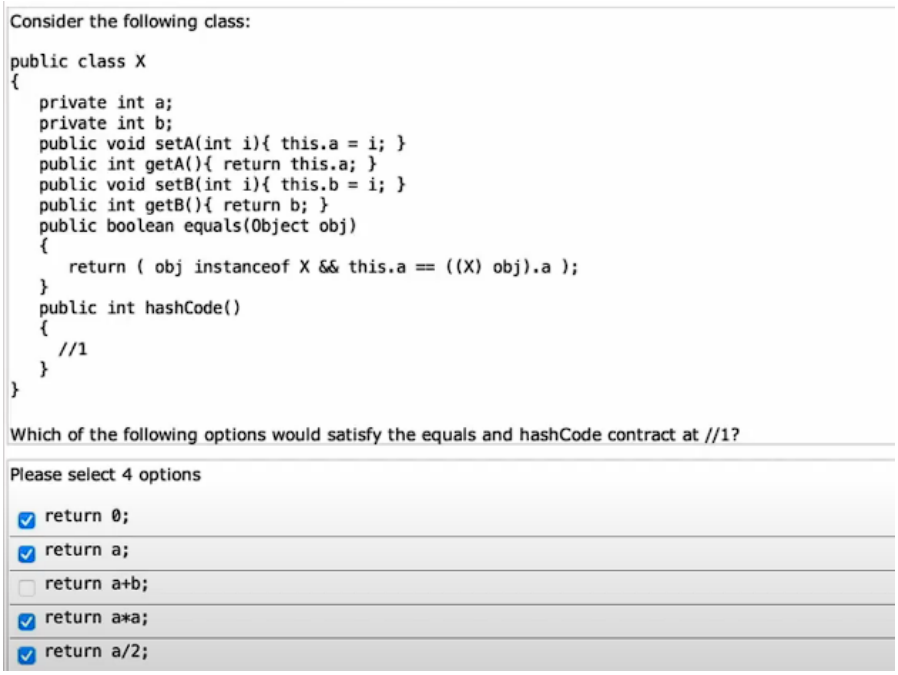
PREGUNTAS VIDEO 20231008_01

1. WHICH OF THE FOLLOWING OPTIONS WOULD SATISFY THE EQUALS AND HASHCODE CONTRACT AT //1?



Consider the following class:

```
public class X
{
   private int a;
   private int b;
   public void setA(int i){ this.a = i; }
   public int getA(){ return this.a; }
   public void setB(int i){ this.b = i; }
   public int getB(){ return b; }
   public boolean equals(Object obj)
   {
      return ( obj instanceof X && this.a == ((X) obj).a );
   }
   public int hashCode()
   {
     //1
   }
}
```

Which of the following options would satisfy the equals and hashCode contract at //1?

Please select 4 options

- [x] return 0;
- [x] return a;
- [ ] return a+b;
- [x] return a*a;
- [x] return a/2;

EXPLICACIÓN:

Respuesta correcta:

El contrato de equals y hashCode establece que si dos objetos son iguales según el método equals, entonces deben tener el mismo valor de hash code.

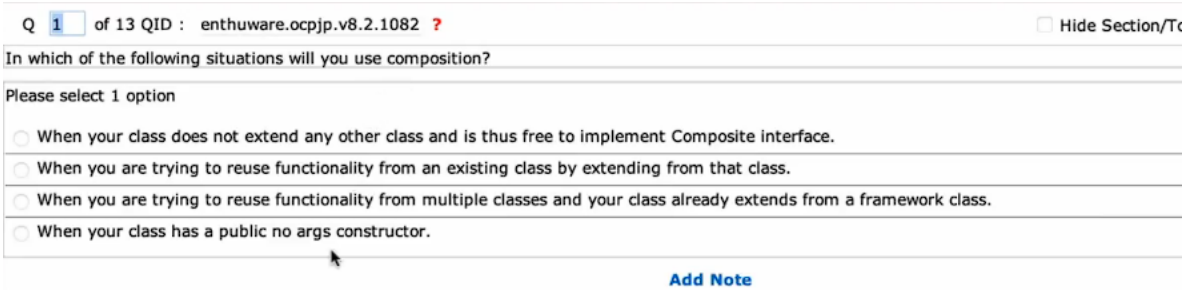
Análisis de las Opciones

1. **Opción a: return 0;**
    - Esta opción hará que todos los objetos tengan el mismo hash code, independientemente del valor de a.
    - No es una buena práctica porque reduce la eficiencia de las estructuras de datos que usan hash code, pero cumple con el contrato porque si dos objetos son iguales según equals, tendrán el mismo hash code (0).
    - **Cumple el contrato, pero es ineficiente.**
2. **Opción b: return a;**
    - Esta opción es correcta porque usa el valor del campo a, que es el mismo campo que se usa en el método equals para determinar la igualdad.
    - Cumple el contrato**.**
3. **Opción c: return a + b;**
    - Esta opción incluye el campo b, que no se usa en el método equals para determinar la igualdad.

- Si dos objetos tienen el mismo valor para a pero diferentes valores para b, equals dirá que son iguales, pero hashCode dará resultados diferentes.
- No cumple el contrato.

4. **Opción d: return a * a;**
    - Esta opción usa solo el campo a y es una transformación de a. Si dos objetos tienen el mismo valor para a, a * a será el mismo.
    - Cumple el contrato.
5. **Opción f: return a / 2;**
    - Esta opción usa solo el campo a y es una transformación de a. Si dos objetos tienen el mismo valor para a, a / 2 será el mismo.
    - Cumple el contrato

2. IN WHICH OF THE FOLLOWING SITUATIONS WILL YOU USE COMPOSITION?



EXPLICACIÓN:

Respuesta:

When you are trying to reuse functionality from multiple classes and your class already extends from a framework class.

La composición es una técnica de diseño en Java para implementar una relación de tipo "Has-a" . La herencia de Java se utiliza con fines de reutilización de código y lo mismo podemos hacer mediante la composición. La composición se logra mediante el uso de una variable de instancia que hace referencia a otros objetos. Si un objeto contiene a otro objeto y el objeto contenido no puede existir sin la existencia de ese objeto, entonces se denomina composición. En palabras más específicas, la composición es una forma de describir la referencia entre dos o más clases mediante una variable de instancia y se debe crear una instancia antes de usarla.

Ciertos puntos clave de la composición en Java son los siguientes:

- En su composición ambas entidades dependen una de la otra.
- Cuando hay una composición entre dos entidades, el objeto compuesto no puede existir sin la otra entidad. Por ejemplo, una biblioteca puede tener 100 libros **sobre** el mismo tema o sobre temas diferentes. Por lo tanto, si la biblioteca se destruye, todos los libros dentro de esa biblioteca en particular serán destruidos. Esto se debe a que los libros no pueden existir sin una biblioteca.
- La composición se logra mediante el uso de una variable de instancia que hace referencia a otros objetos.
- Tenemos que favorecer la composición sobre la herencia.