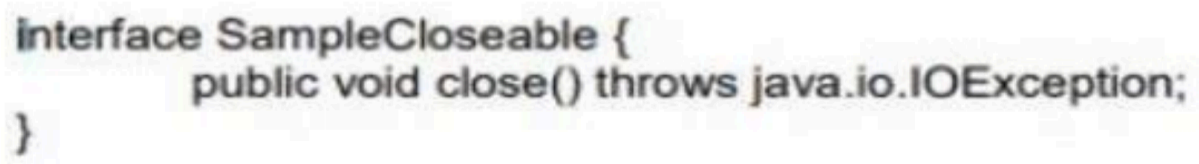
20231001_01

Contenido

Herencia con manejo de excepciones | Reglas de Exceptions en Override
Excepciones con hilos | synchronized - wait()

Herencia con manejo de excepciones

Given



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

Which three implementations are valid?

- class Test implements SampleCloseable { public void close() throws java.io.IOException { // do something } }
- class Test implements SampleCloseable { public void close() throws Exception { // do something } }
- class Test implements SampleCloseable { public void close() throws FileNotFoundException { // do something } }
- class Test extends SampleCloseable { public void close() throws java.io.IOException { // do something } }
- class Test implements SampleCloseable { public void close() { // do something }}

Puntos claves:

- Características de una Interface:
    - No puede ser instanciada.
    - Por naturaleza los métodos de una interfaz son abstractos y públicos.
- Los métodos que se sobreescriben no pueden reducir la visibilidad.
- La notación java.io.IOException es válida cuando no se tiene el import java.io.IOException.
- throw es utilizado para arrojar la excepción dentro del método, para definir en los métodos se utiliza throws.
- Reglas de herencia con excepciones en métodos:
    - El hijo puede lanzar la misma excepción que el padre.
    - El hijo puede no lanzar alguna la excepción.
    - El hijo puede lanzar una excepción subclase de la excepción padre.
- Aunque se lance una excepción tipo unchecked, no se está obligado a manejarla.
- Exception se importa de java.lang, y java.lang se importa por defecto.

Análisis de opciones

Opción 1: class Test implements SampleCloseable { public void close() throws java.io.IOException { // do something } }
Puntos a favor:

- Clase Test implementa correctamente la interfaz.
- Misma firma del método -> Override
- El método implementado es public.
- Indica que lanza la misma excepción definida en la interfaz.

Opción 2: class Test implements SampleCloseable { public void close() throws Exception { // do something } }}
Puntos a favor:

- Clase Test implementa correctamente la interfaz.
- Misma firma del método -> Override
- El método implementado es public.

Puntos en contra:

- El método indica que se lanza una excepción superclase respecto a la lanzada en el método definido en la interfaz.

Opción 3: class Test implements SampleCloseable { public void close() throws FileNotFoundException { // do something } }
Puntos a favor:

- Clase Test implementa correctamente la interfaz.
- Misma firma del método -> Override.
- El método implementado es public.
- El método indica que se lanza una excepción subclase respecto a la lanzada en el método definido en la interfaz.

Opción 4: class Test extends SampleCloseable { public void close() throws java.io.IOException { // do something } }
Puntos en contra:

- Clase Test no implementa correctamente la interfaz.

Opción 5: class Test implements SampleCloseable { public void close() { // do something }}
Puntos a favor:

- Clase Test implementa correctamente la interfaz.
- Misma firma del método -> Override.
- El método implementado es public.
- El método implementado omite el lanzamiento de la excepción.

Respuesta correcta:

- **Opción 1: class Test implements SampleCloseable { public void close() throws java.io.IOException { // do something } }**
- **Opción 3: class Test implements SampleCloseable { public void close() throws FileNotFoundException { // do something } }**
- **Opción 5: class Test implements SampleCloseable { public void close() { // do something }}**

Notas

Existe un error en la pregunta, en el código no se muestran los import y la excepción se está colocando con el nombre completo **java.io.IOException**, entonces la opción 3 únicamente es correcta si se añade un import o colocando el nombre completo **java.io.FileNotFoundException.**

Alternativas

- Cuando el método en la interfaz no arroja una excepción, el método implementado únicamente puede arrojar excepciones unchecked.

Excepciones con hilos

Given

```
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 ");
        }
}
```

What is the result?

- The program prints 1 then 2 after 5 seconds.
- The program prints: 1 thrown to main.
- The program prints: 1 2 thrown to main.
- The program prints:1 then t1 waits for its notification.

Puntos claves:

- InterruptedException pertenece a java.lang y java.lang se importa automáticamente.
- El método .wait() es quien lanza la checked exception InterruptedException.

- El orden de ejecución de los hilos es decisión de la JVM y se puede ejecutar en cualquier orden.
- En un catch es posible cachar al padre de la excepción
- synchronized asegura que únicamente un hilo pueda ejecutar un método a la vez en una instancia.
- Instancia de un hilo (Thread t1 = new Thread();), un hilo es una tarea separada que puede ejecutarse concurrentemente con otras tareas.
- El método .start() indica que un hilo comienza a ejecutarse.
- .wait() hace que el hilo actual espere y libere el monitor permitiendo que otros hilos lo adquieran..
- En programación concurrente un monitor es el mecanismo que solo un hilo a la vez pueda ejecutar un bloque de código. El monitor es como una llave que asegura que solo un hilo puede entrar a un bloque de código a la vez.
- En Java, los monitores están asociados con cada objeto y se utilizan para gestionar la sincronización.
- Cuando un hilo quiere ejecutar un bloque synchronized necesita adquirir el monitor del objeto. Si otro hilo ya tiene al monitor, el hilo actual debe esperar hasta que el monitor esté disponible.
- Cuando el hilo termina su ejecución libera el monitor, permitiendo que otros hilos puedan adquirirlo.
-

Análisis de código

- En el método main se crea una instancia de Bees y se llama al método go().
- El método go() es synchronized, lo que significa que cualquier hilo que lo llame debe adquirir el monitor del objeto Bees antes de ejecutarlo.
- Se crea un nuevo hilo y se inicia aunque no se le da ninguna tarea a ejecutar.
- Imprime 1.
- Intenta que el hilo t1 espere 5 segundos. Esto falla porque el hilo principal posee el monitor del objeto Bees, no de t1, esto causa un IllegalMonitorStateException.
- El programa ejecuta el catch, imprime y termina.

Respuesta correcta:

The program prints: 1 thrown to main.

Notas

A pesar que en el método go() se indica el lanzamiento de InterruptedException, la excepción que en realidad de lanza es IllegalMonitorStateException. El manejo de la excepción se logra debido a que en el catch se maneja Exception.