Parte #1

Ventajas de realizar la captura de errores:

1. Detección y diagnóstico de errores: Al capturar y manejar errores, puedes obtener información valiosa sobre la causa y la ubicación del error. Esto facilita el proceso de depuración y resolución de problemas en tu código. La captura de errores te permite generar mensajes de error más descriptivos y específicos, lo que ayuda a identificar rápidamente los problemas y solucionarlos de manera efectiva.
2. Mejora de la robustez y la estabilidad: Al manejar adecuadamente los errores, puedes evitar que tu programa se bloquee o se comporte de manera inesperada frente a situaciones excepcionales o datos incorrectos. La captura de errores te permite implementar mecanismos de recuperación, como ofrecer al usuario una opción alternativa o realizar acciones correctivas, para mantener la estabilidad y la funcionalidad de tu programa incluso en condiciones inesperadas.

Objetivos de Debuggear nuestro código fuente Java:

1. Identificar y corregir errores lógicos: Uno de los principales objetivos al debuggear el código fuente Java es encontrar y corregir errores lógicos o "bugs". Estos errores pueden provocar un comportamiento inesperado o incorrecto del programa. Al depurar, puedes utilizar herramientas como puntos de interrupción (breakpoints) para detener la ejecución del programa en puntos específicos y examinar los valores de las variables, así como seguir el flujo de ejecución. Esto te permite analizar el código paso a paso y detectar dónde se produce el error lógico. Una vez identificado, puedes corregir el error y asegurarte de que el programa funcione según lo previsto.
2. Optimizar el rendimiento: Otro objetivo importante al debuggear el código fuente Java es mejorar el rendimiento del programa. Durante el proceso de depuración, puedes detectar secciones de código que consumen muchos recursos o ejecutan instrucciones innecesarias. Al identificar estas áreas problemáticas, puedes realizar ajustes en el código para optimizar su rendimiento. Esto podría implicar la refactorización del código, la eliminación de bucles redundantes, el uso de algoritmos más eficientes, entre otras técnicas. El objetivo es lograr que el programa se ejecute de manera más rápida y eficiente, lo que puede resultar en tiempos de respuesta más cortos y un mejor uso de los recursos del sistema.

Concepto de herencia:

En el contexto de la programación orientada a objetos, la herencia es un concepto fundamental que permite crear nuevas clases basadas en clases existentes, lo que permite la reutilización de código y la organización jerárquica de las clases. Cuando una clase hereda de otra, adquiere automáticamente los atributos y métodos de la clase padre, lo que proporciona una serie de beneficios importantes. A continuación, se presentan tres beneficios de aplicar el concepto de herencia en la programación orientada a objetos:

1. Reutilización de código: La herencia permite reutilizar el código existente al crear nuevas clases. Al heredar de una clase base, la clase derivada hereda automáticamente todos los atributos y métodos de la clase padre. Esto evita la necesidad de volver a escribir el mismo código en múltiples lugares y promueve la modularidad y mantenibilidad del código.
2. Extensibilidad: La herencia permite extender y especializar las funcionalidades de una clase base. Una clase derivada puede agregar nuevos atributos y métodos, o incluso modificar el comportamiento de los métodos heredados. Esto permite adaptar y ampliar la funcionalidad de una clase base sin modificar directamente su implementación, lo que facilita la creación de programas flexibles y adaptables.
3. Polimorfismo: El polimorfismo es otro beneficio clave de la herencia en la programación orientada a objetos. Permite tratar objetos de diferentes clases de manera uniforme cuando comparten una clase base común. Esto significa que se puede escribir código que funcione con la clase base y luego utilizar ese mismo código con objetos de clases derivadas sin necesidad de conocer los detalles específicos de cada clase. Esto facilita la creación de programas genéricos y flexibles que pueden manejar diferentes tipos de objetos de manera transparente.