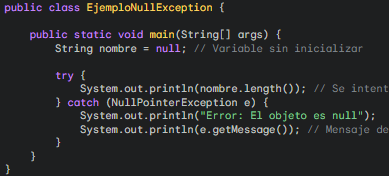
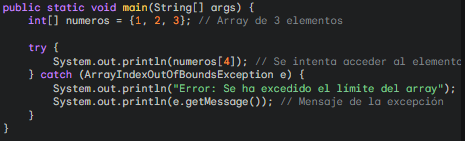
Tipos de excepciones:

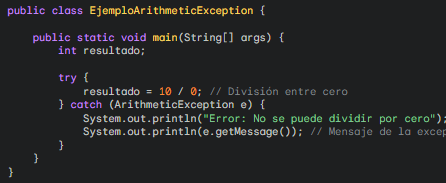
1. **NullPointerException:** Esta excepción se produce cuando se intenta acceder a un miembro o método de un objeto nulo. Es una de las excepciones más comunes en Java y, a menudo, se debe a un error en la programación del programa.



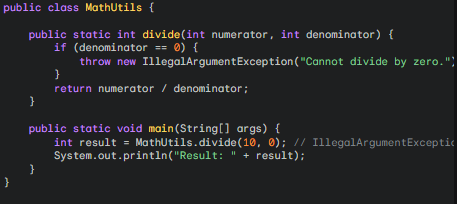
1. **ArrayIndexOutOfBoundsException:** Esta excepción se produce cuando se intenta acceder a un elemento de un array fuera de sus límites. Por ejemplo, si un array tiene una longitud de 10 y se intenta acceder al elemento 11, se lanzará esta excepción.



1. ArithmeticException: Esta excepción se produce cuando se produce un error aritmético, como la división por cero o el desbordamiento de enteros.



1. IllegalArgumentException: Esta excepción se produce cuando se pasa un argumento no válido a un método. Por ejemplo, si un método espera un número entero y se le pasa una cadena, se lanzará esta excepción.



1. **FileNotFoundException:** Esta excepción se produce cuando se intenta abrir un archivo que no existe.

public class EjemploIllegalArgumentException {

public static void main(String[] args) {

int edad = -10;

try {

validarEdad(edad); // Se pasa un argumento inválido

} catch (IllegalArgumentException e) {

System.out.println("Error: La edad debe ser mayor o igual a cero");

System.out.println(e.getMessage());

}

}

static void validarEdad(int edad) throws IllegalArgumentException {

if (edad < 0) {

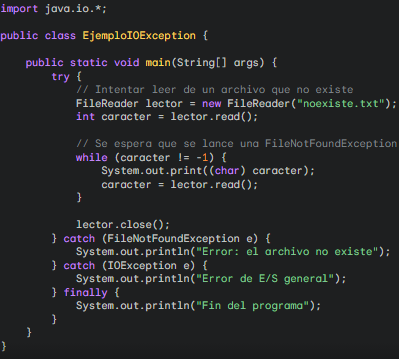
throw new IllegalArgumentException("La edad debe ser mayor o igual a cero");

}

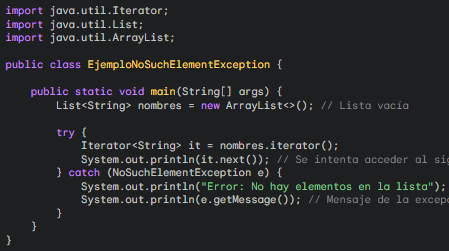
}

}

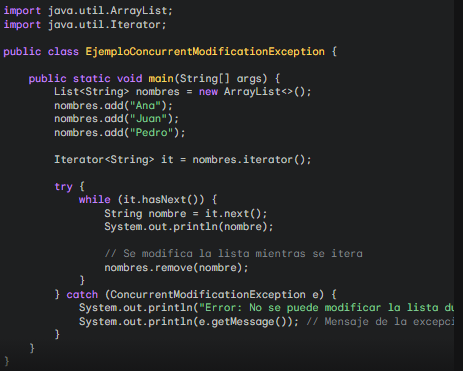
1. **IOException:** Esta excepción es una clase base para varias excepciones relacionadas con E/S, como FileNotFoundException y FileNotfoundException.



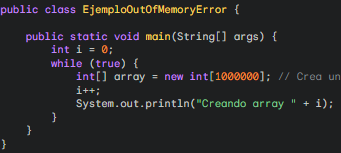
1. **NoSuchElementException:** Esta excepción se produce cuando se intenta acceder a un elemento de una colección vacía o cuando se ha iterado sobre todos los elementos de una colección y se intenta obtener el siguiente.



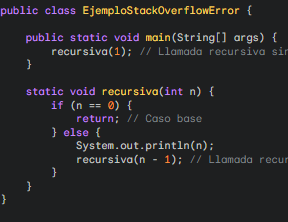
1. **ConcurrentModificationException:** Esta excepción se produce cuando se intenta modificar una colección mientras se está iterando sobre ella.



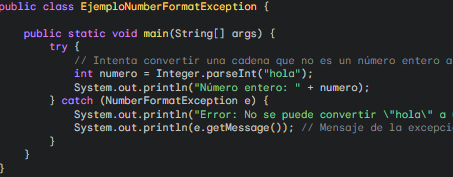
1. **OutOfMemoryError:** Esta excepción se produce cuando el programa se queda sin memoria.



1. **StackOverflowError:** Esta excepción se produce cuando la pila de llamadas se desborda, lo que suele deberse a una recursión infinita.



1. **NumberFormatException:** Esta excepción se produce cuando se intenta convertir un string a un tipo numérico, como un int o un double, pero el formato del string no es válido. Por ejemplo, si se intenta convertir el string "abc" a un int, se lanzará esta excepción.



1. **SecurityException:** Esta excepción se produce cuando se viola una política de seguridad, como cuando un programa intenta acceder a un recurso al que no tiene permiso.

import java.io.File;

public class EjemploSecurityException {

public static void main(String[] args) {

try {

File archivo = new File("/etc/passwd");

archivo.delete();

} catch (SecurityException e) {

System.out.println("Error: No se tiene permiso para eliminar el archivo");

System.out.println(e.getMessage());

}

}

}

1. **UnsupportedOperationException:** Esta excepción se produce cuando se intenta realizar una operación que no está soportada por un objeto o clase. Por ejemplo, si se intenta modificar un objeto que es inmutable, se lanzará esta excepción.

import java.util.Collections;

import java.util.List;

public class EjemploUnsupportedOperationException {

public static void main(String[] args) {

List<String> nombres = new ArrayList<>();

nombres.add("Ana");

nombres.add("Juan");

nombres.add("Pedro");

List<String> nombresInmutables = Collections.unmodifiableList(nombres);

try {

nombresInmutables.add("Diego");

} catch (UnsupportedOperationException e) {

System.out.println("Error: No se puede modificar la lista inmutable");

System.out.println(e.getMessage());

}

}

}

1. **IllegalStateException:** Esta excepción se produce cuando se intenta realizar una operación en un estado ilegal. Por ejemplo, si se intenta cerrar un archivo que ya está cerrado, se lanzará esta excepción.

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

public class EjemploIllegalStateException {

public static void main(String[] args) throws IOException {

BufferedReader lector = new BufferedReader(new FileReader("archivo.txt"));

try {

String linea;

while ((linea = lector.readLine()) != null) {

System.out.println(linea);

}

} finally {

lector.close(); // Cerrar el lector

}

String linea = lector.readLine(); // Lanzará IllegalStateException

System.out.println(linea);

}

}

1. **MalformedURLException:** Esta excepción se produce cuando se intenta crear una URL a partir de un string que no tiene un formato de URL válido. Por ejemplo, si se intenta crear una URL a partir del string "abc://def", se lanzará esta excepción.

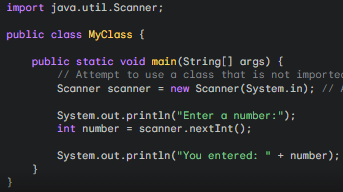


1. NoClassDefFoundError: Esta excepción se produce cuando el programa no puede encontrar la definición de una clase. Esto puede ocurrir si la clase no está en el classpath del programa o si está corrupta.

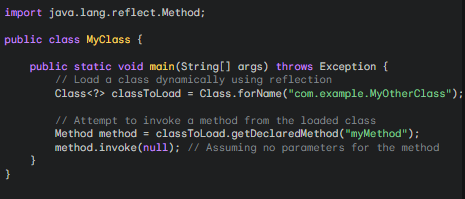
Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

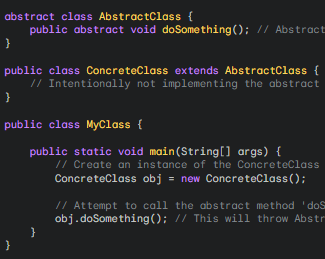
1. ClassNotFoundException: Esta excepción se produce cuando el programa no puede encontrar una clase específica. Esto puede ocurrir si la clase no existe o si no está en el classpath del programa.



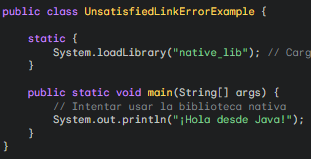
1. **LinkageError:** Esta excepción se produce cuando hay un problema con la vinculación de clases, como cuando las clases no se pueden cargar correctamente o cuando hay conflictos entre las clases.



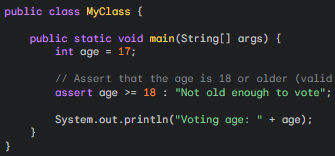
1. **AbstractMethodError:** Esta excepción se produce cuando se intenta llamar a un método abstracto. Esto solo puede ocurrir si se está intentando instanciar una clase abstracta o si se está llamando a un método abstracto desde una subclase que no lo ha implementado.



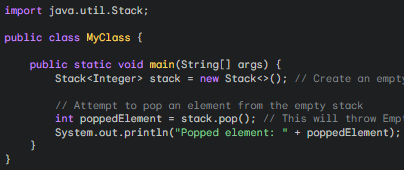
1. **UnsatisfiedLinkError:** Esta excepción se produce cuando el programa no puede encontrar una biblioteca nativa que necesita. Esto puede ocurrir si la biblioteca no está instalada en el sistema o si no es compatible con la plataforma del sistema.



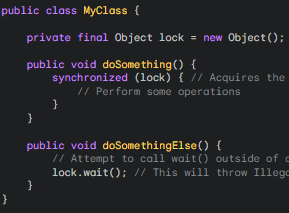
1. **AssertionError:** Esta excepción se produce cuando una aserción falla. Las aserciones se utilizan para verificar suposiciones sobre el estado de un programa. Si una aserción falla, significa que se ha producido un error inesperado y el programa no puede continuar.



1. **EmptyStackException:** Esta excepción se produce cuando se intenta acceder a un elemento de una pila vacía. Esto puede ocurrir si se intenta desapilar elementos de una pila que ya está vacía.



1. **IllegalMonitorStateException:** Esta excepción se produce cuando se intenta sincronizar en un objeto que no se posee o cuando se intenta realizar una operación en un objeto sincronizado que no está permitida.



1. **InterruptedException:** Esta excepción se produce cuando un hilo se interrumpe mientras está ejecutando un código. Esto puede ocurrir si otro hilo llama al método interrupt() del hilo interrumpido.

import java.lang.Thread;

public class MyClass {

public static void main(String[] args) {

Thread thread = new Thread(() -> {

try {

// Simulate a long-running operation

Thread.sleep(5000); // Sleep for 5 seconds

} catch (InterruptedException e) {

System.out.println("Thread interrupted: " + e.getMessage());

}

});

thread.start(); // Start the thread

try {

// Wait for 2 seconds

Thread.sleep(2000);

} catch (InterruptedException e) {

e.printStackTrace();

}

// Interrupt the thread

thread.interrupt();

}

}

1. **InvalidClassException:** Esta excepción se produce cuando el programa intenta cargar una clase que está corrupta o no es válida.

import java.io.\*;

public class MyClass {

public static void main(String[] args) throws IOException, ClassNotFoundException {

MySerializableObject obj = new MySerializableObject();

FileOutputStream fos = new FileOutputStream("serializable.data");

ObjectOutputStream oos = new ObjectOutputStream(fos);

oos.writeObject(obj);

oos.close();

FileInputStream fis = new FileInputStream("serializable.data");

ObjectInputStream ois = new ObjectInputStream(fis);

MySerializableObject deserializedObj = (MySerializableObject) ois.readObject();

ois.close();

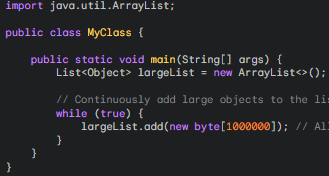
}

}

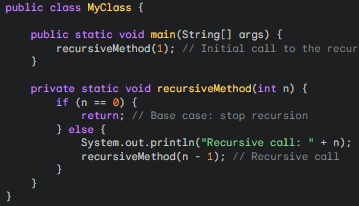
class MySerializableObject implements Serializable {

}

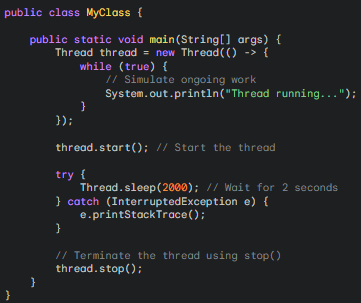
1. **OutOfMemoryError:** Esta excepción se produce cuando el programa se queda sin memoria. Esto puede ocurrir si el programa intenta asignar más memoria de la que está disponible.



1. **StackOverflowError:** Esta excepción se produce cuando la pila de llamadas se desborda. Esto suele deberse a una recursión infinita.



1. **ThreadDeathError:** Esta excepción se produce cuando un hilo se termina de forma anormal. Esto puede ocurrir si el hilo se interrumpe o si se produce una excepción no controlada en el hilo.



1. **UnknownHostException:** Esta excepción se produce cuando el programa no puede resolver un nombre de host a una dirección IP. Esto puede ocurrir si el nombre de host no es válido o si el programa no tiene acceso a un servidor DNS.

import java.net.\*;

public class MyClass {

public static void main(String[] args) {

try {

InetAddress address = InetAddress.getByName("invalid-domain.com");

System.out.println("IP address: " + address.getHostAddress());

} catch (UnknownHostException e) {

System.out.println("Error: Unknown host " + e.getMessage());

}

}

}

1. **UnsupportedEncodingException:** Esta excepción se produce cuando el programa intenta codificar o decodificar un string utilizando una codificación de caracteres no compatible.

import java.io.\*;

import java.net.URLDecoder;

public class MyClass {

public static void main(String[] args) {

String encodedString = "https%3A%2F%2Fwww.example.com%2Fpath%2Fwith%20spaces%2F";

try {

String decodedString = URLDecoder.decode(encodedString, "InvalidEncoding");

System.out.println("Decoded string: " + decodedString);

} catch (UnsupportedEncodingException e) {

System.out.println("Error: Unsupported encoding: " + e.getMessage());

}

}

}

1. **AccessControlException:** Esta excepción se produce cuando un programa intenta acceder a un recurso al que no tiene permiso. Esto puede ocurrir si el programa no tiene los permisos adecuados o si el recurso está protegido por una política de seguridad.

package com.example.restricted;

import java.security.AccessControlException;

public class RestrictedClass {

private static final String SECRET\_MESSAGE = "This is a secret message.";

public static void main(String[] args) {

RestrictedClass obj = new RestrictedClass();

System.out.println(obj.getSecretMessage());

}

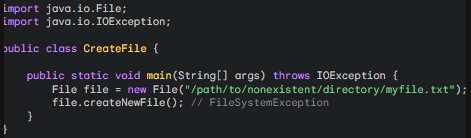
public String getSecretMessage() {

return SECRET\_MESSAGE;

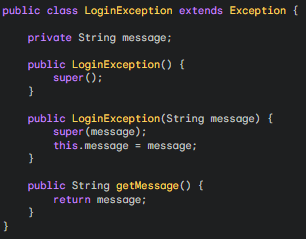
}

}

1. **FileSystemException:** Esta excepción es una clase base para varias excepciones relacionadas con el sistema de archivos, como FileNotFoundException y FileNotfoundException.



1. **LoginException:** Esta excepción se produce cuando un usuario no puede iniciar sesión en un sistema o aplicación. Esto puede ocurrir si el usuario introduce un nombre de usuario o contraseña incorrectos, o si la cuenta del usuario está deshabilitada.

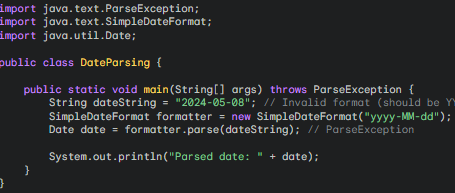


1. **NotSerializableException:** Esta excepción se produce cuando se intenta serializar un objeto que no es serializable. La serialización es el proceso de convertir un objeto en un formato de datos binario que se puede almacenar o transmitir.

Texto

Descripción generada automáticamente

1. **ParseException:** Esta excepción se produce cuando se intenta analizar un string en un formato específico, como un número o una fecha, pero el formato del string no es válido.



1. **RemoteException:** Esta excepción se produce cuando se produce un error al comunicarse con un objeto remoto. Esto puede ocurrir si el objeto remoto no está disponible o si hay un problema con la red.

import java.net.ServerSocket;

import java.net.Socket;

import java.rmi.RemoteException;

import java.rmi.server.UnicastRemoteObject;

public class Server implements RemoteInterface {

private static final int PORT = 1099;

public Server() throws RemoteException {

UnicastRemoteObject.exportObject(this);

}

@Override

public String getMessage() throws RemoteException {

return "Hello from the server!";

}

public static void main(String[] args) throws Exception {

ServerSocket serverSocket = new ServerSocket(PORT);

System.out.println("Server started on port " + PORT);

while (true) {

Socket clientSocket = serverSocket.accept();

new Thread(() -> handleClient(clientSocket)).start();

}

}

private static void handleClient(Socket clientSocket) {

try {

RemoteInterface remoteInterface = (RemoteInterface) UnicastRemoteObject.exportObject(new Server(), 0);

System.out.println("Client connected from " + clientSocket.getInetAddress().getHostAddress());

String message = remoteInterface.getMessage();

System.out.println("Received message from client: " + message);

clientSocket.close();

} catch (RemoteException e) {

System.err.println("Error communicating with client: " + e.getMessage());

} catch (Exception e) {

System.err.println("Unexpected error: " + e.getMessage());

}

}

}

1. **TimeoutException:** Esta excepción se produce cuando una operación tarda demasiado tiempo en completarse. Esto puede ocurrir si la operación está bloqueada o si hay un problema con la red.

import java.net.URL;

import java.net.URLConnection;

import java.io.IOException;

public class TimeoutExample {

public static void main(String[] args) throws IOException {

URL url = new URL("https://www.google.com"); // Replace with a URL that may take longer to load

URLConnection connection = url.openConnection();

connection.setConnectTimeout(1000); // Set a 1-second timeout

try {

connection.getInputStream();

} catch (IOException e) {

if (e instanceof java.net.SocketTimeoutException) {

System.out.println("Connection timed out after 1 second.");

} else {

System.err.println("Error opening connection: " + e.getMessage());

}

}

}

}

1. UnsupportedOperationException: Esta excepción se produce cuando se intenta realizar una operación que no está soportada por un objeto o clase. Por ejemplo, si se intenta modificar un objeto que es inmutable, se lanzará esta excepción.

import java.util.Collections;

import java.util.List;

public class UnmodifiableCollectionExample {

public static void main(String[] args) {

List<String> names = new ArrayList<>();

names.add("Alice");

names.add("Bob");

List<String> unmodifiableNames = Collections.unmodifiableList(names);

try {

unmodifiableNames.add("Charlie");

} catch (UnsupportedOperationException e) {

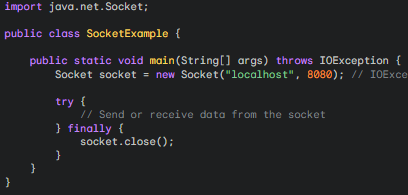
System.out.println("Error: Operation not supported on unmodifiable list.");

}

}

}

1. **IOException:** Esta excepción es una clase base para varias excepciones relacionadas con E/S, como FileNotFoundException y FileNotfoundException.



1. **ZipException:** Esta excepción se produce cuando se produce un error al manipular un archivo ZIP. Esto puede ocurrir si el archivo ZIP está corrupto o si no está en el formato correcto.

import java.io.FileInputStream;

import java.io.IOException;

import java.util.zip.ZipInputStream;

public class CorruptedZipFileExample {

public static void main(String[] args) throws IOException {

String zipFilePath = "corrupted.zip"; // Replace with the path to a corrupted ZIP file

try (FileInputStream fis = new FileInputStream(zipFilePath);

ZipInputStream zis = new ZipInputStream(fis)) {

} catch (ZipException e) {

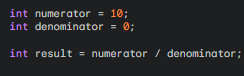
System.err.println("Error opening ZIP file: " + e.getMessage());

}

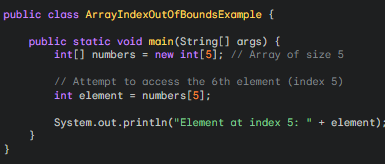
}

}

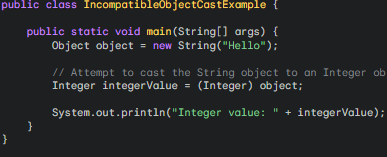
1. **ArithmeticException:** Esta excepción se produce cuando se produce un error aritmético, como la división por cero o el desbordamiento de enteros.



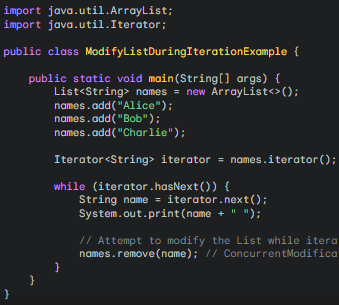
1. **ArrayIndexOutOfBoundsException:** Esta excepción se produce cuando se intenta acceder a un elemento de un array fuera de sus límites.



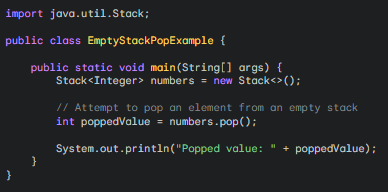
1. **ClassCastException:** Esta excepción se produce cuando se intenta convertir un objeto a un tipo incompatible. Por ejemplo, si se intenta convertir un objeto String a un objeto int, se lanzará esta excepción.



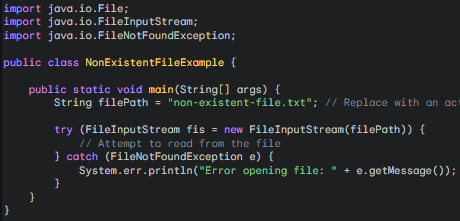
1. **ConcurrentModificationException:** Esta excepción se produce cuando se intenta modificar una colección mientras se está iterando sobre ella.



1. **EmptyStackException:** Esta excepción se produce cuando se intenta acceder a un elemento de una pila vacía.



1. **FileNotFoundException:** Esta excepción se produce cuando se intenta abrir un archivo que no existe.



1. **IllegalArgumentException:** Esta excepción se produce cuando se pasa un argumento no válido a un método. Por ejemplo, si un método espera un número entero y se le pasa una cadena, se lanzará esta excepción.

public class IllegalSwitchCaseValueExample {

public static void main(String[] args) {

int day = 10; // Invalid day value (10 is not a valid day of the week)

switch (day) {

case 1:

System.out.println("Monday");

break;

case 2:

System.out.println("Tuesday");

break;

// ... other cases for Wednesday, Thursday, Friday, Saturday, Sunday

default:

throw new IllegalArgumentException("Invalid day value: " + day);

}

}

}

1. **IllegalMonitorStateException:** Esta excepción se produce cuando se intenta sincronizar en un objeto que no se posee o cuando se intenta realizar una operación en un objeto sincronizado que no está permitida.

public class IllegalMonitorStateExceptionExample {

private Object lock = new Object();

public void waitWithoutLock() {

try {

lock.wait(); // IllegalMonitorStateException

} catch (InterruptedException e) {

e.printStackTrace();

}

}

public void notifyWithoutLock() {

lock.notify(); // IllegalMonitorStateException

}

public static void main(String[] args) {

IllegalMonitorStateExceptionExample example = new IllegalMonitorStateExceptionExample();

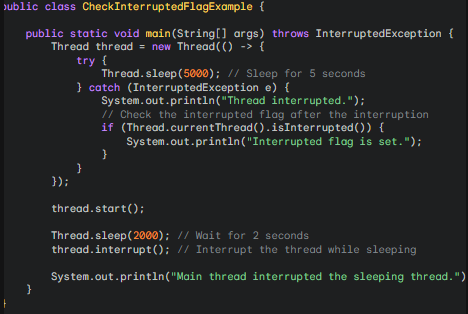
example.waitWithoutLock();

example.notifyWithoutLock();

}

}

1. **InterruptedException:** Esta excepción se produce cuando un hilo se interrumpe mientras está ejecutando un código.



1. **InvalidClassException:** Esta excepción se produce cuando el programa intenta cargar una clase que está corrupta o no es válida.

import java.io.\*;

public class InvalidClassVersionExample {

public static void main(String[] args) throws IOException, ClassNotFoundException, InvalidClassException {

// Read serialized data from a file

FileInputStream fis = new FileInputStream("serialized\_data.txt");

ObjectInputStream ois = new ObjectInputStream(fis);

try {

// Attempt to deserialize the object

Object deserializedObject = ois.readObject();

} catch (ClassNotFoundException e) {

System.err.println("Error: Class not found: " + e.getMessage());

} catch (InvalidClassException e) {

System.err.println("Error: Invalid class version: " + e.getMessage());

} finally {

ois.close();

fis.close();

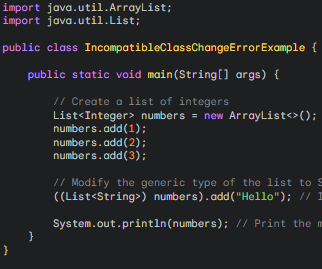
}

}

}

Excepciones de herencia:

1. **IncompatibleClassChangeError:** Esta excepción se produce cuando se intenta cambiar la firma de un método de una clase de una manera que no es compatible con las subclases existentes. Esto puede ocurrir si se cambia el tipo de retorno del método, el número o los tipos de parámetros, o si se elimina un método de la clase base.



1. **UnsupportedClassVersionError:** Esta excepción se produce cuando se intenta cargar una clase que tiene una versión incompatible con la versión del entorno de ejecución de Java (JVM) que se está utilizando. Esto puede ocurrir si la clase se compiló con una versión más reciente de Java que la que se está utilizando para ejecutarla.

import java.io.File;

import java.io.FileInputStream;

import java.io.IOException;

public class UnsupportedClassVersionErrorExample {

public static void main(String[] args) throws IOException {

String classFilePath = "/path/to/class/file.class"; // Replace with an actual path

try (FileInputStream fis = new FileInputStream(classFilePath)) {

Class<?> clazz = ClassLoader.getSystemClassLoader().loadClass(classFilePath);

// Create an instance of the class (will throw UnsupportedClassVersionError)

Object object = clazz.newInstance();

}

}

}

Excepciones de reflection:

1. IllegalAccessException: Esta excepción se produce cuando se intenta acceder a un miembro o recurso de un objeto al que no se tiene permiso a través de la reflexión. Esto puede ocurrir si el miembro o recurso es privado o si el programa no tiene los permisos adecuados.

public class PrivateMemberAccessExample {

private int privateField = 10; // Private field

public void accessPrivateField() {

System.out.println("Private field value: " + privateField); // IllegalAccessException

}

public static void main(String[] args) {

PrivateMemberAccessExample example = new PrivateMemberAccessExample();

example.accessPrivateField(); // IllegalAccessException

}

}

1. **NoSuchMethodException:** Esta excepción se produce cuando se intenta obtener un método de una clase a través de la reflexión, pero el método no existe. Esto puede ocurrir si el nombre del método o los tipos de parámetros son incorrectos.

public class NoSuchMethodExceptionExample {

public static void main(String[] args) {

try {

// Attempt to call a non-existent method

Class<?> clazz = Class.forName("com.example.MyClass");

Object obj = clazz.newInstance();

clazz.getMethod("nonExistentMethod").invoke(obj);

} catch (ClassNotFoundException | InstantiationException | IllegalAccessException | NoSuchMethodException | InvocationTargetException e) {

e.printStackTrace();

}

}

}

Excepciones de serialización:

1. **NotSerializableException:** Esta excepción se produce cuando se intenta serializar un objeto que no es serializable. La serialización es el proceso de convertir un objeto en un formato de datos binario que se puede almacenar o transmitir.

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.ObjectOutputStream;

public class NotSerializableExceptionExample {

public static void main(String[] args) throws IOException {

// Create an object of a class that does not implement Serializable

MyCustomClass obj = new MyCustomClass();

// Attempt to serialize the object

try (FileOutputStream fos = new FileOutputStream("serialized\_object.txt");

ObjectOutputStream oos = new ObjectOutputStream(fos)) {

oos.writeObject(obj); // NotSerializableException

}

}

}

class MyCustomClass {

// No implementation of Serializable interface

}

1. **SerializationException:** Esta excepción es una clase base para varias excepciones relacionadas con la serialización, como NotSerializableException y InvalidClassException.

import java.io.\*;

import java.util.ArrayList;

import java.util.List;

public class SerializationExceptionExample {

public static void main(String[] args) throws IOException, ClassNotFoundException {

// Create a list containing an incompatible object

List<Object> list = new ArrayList<>();

list.add(1); // Integer object

list.add("Hello"); // String object - Incompatible for serialization

// Attempt to serialize the list

try (FileOutputStream fos = new FileOutputStream("serialized\_list.txt");

ObjectOutputStream oos = new ObjectOutputStream(fos)) {

oos.writeObject(list); // SerializationException

}

}

}

Excepciones de concurrencia:

1. **TimeoutException:** Esta excepción se produce cuando una operación tarda demasiado tiempo en completarse. Esto puede ocurrir si la operación está bloqueada o si hay un problema con la red.

import java.io.IOException;

import java.net.HttpURLConnection;

import java.net.URL;

import java.util.concurrent.TimeUnit;

public class TimeoutExceptionExample {

public static void main(String[] args) throws IOException, InterruptedException {

// Set the timeout value in milliseconds

long timeoutInMillis = 1000; // 1 second

try (HttpURLConnection connection = (HttpURLConnection) new URL("https://www.example.com").openConnection()) {

// Set the connection timeout

connection.setConnectTimeout((int) timeoutInMillis);

// Set the read timeout

connection.setReadTimeout((int) timeoutInMillis);

// Send the GET request

connection.setRequestMethod("GET");

connection.connect();

// Wait for the response

connection.getInputStream().read(); // TimeoutException

}

}

}

1. **DeadLockError:** Este error se produce cuando dos o más hilos se bloquean esperando a que el otro libere un recurso. Esto puede ocurrir si cada hilo tiene un recurso que el otro necesita.

public class DeadLockErrorExample {

public static void main(String[] args) {

final String resource1 = "resource1";

final String resource2 = "resource2";

// Create and start two threads

Thread thread1 = new Thread(() -> {

synchronized (resource1) {

System.out.println("Thread 1: Lock acquired: " + resource1);

try {

Thread.sleep(1000); // Simulate delay

} catch (InterruptedException e) {

e.printStackTrace();

}

System.out.println("Thread 1: Waiting for lock: " + resource2);

synchronized (resource2) {

System.out.println("Thread 1: Lock acquired: " + resource2);

}

}

});

Thread thread2 = new Thread(() -> {

synchronized (resource2) {

System.out.println("Thread 2: Lock acquired: " + resource2);

try {

Thread.sleep(1000); // Simulate delay

} catch (InterruptedException e) {

e.printStackTrace();

}

System.out.println("Thread 2: Waiting for lock: " + resource1);

synchronized (resource1) {

System.out.println("Thread 2: Lock acquired: " + resource1);

}

}

});

// Start both threads

thread1.start();

thread2.start();

}

}