1. .

a. En Java, la clase que define las excepciones es java.lang.Exception. Esta clase es la clase base para todas las excepciones chequeadas en Java.

1. b. La clase en Java que representa las excepciones que se producen al invocar un método en un objeto nulo es java.lang.NullPointerException. Esta excepción se lanza cuando intentas acceder o invocar un método en un objeto que no ha sido inicializado (es decir, es nulo).
2. c. El método printStackTrace() en el manejo de excepciones en Java proporciona información sobre la pila de llamadas en el momento en que se lanzó la excepción. Esta información incluye el nombre de la clase, el método y el número de línea donde ocurrió la excepción, lo que puede ser útil para depurar y entender la causa del error.
3. En Java, el mensaje asociado a una excepción se puede obtener utilizando el método getMessage() proporcionado por la clase base Throwable

*public class* Main {  
 *public static void* main(String[] args) {  
 *try* {  
 *// Intentamos dividir por cero, lo que lanzará una ArithmeticException  
 int* resultado = dividirPorCero();  
 System.out.println("El resultado de la división es: " + resultado);  
 } *catch* (ArithmeticException e) {  
 *// Capturamos la excepción y obtenemos el mensaje asociado* String mensaje = e.getMessage();  
 System.out.println("Se ha producido una excepción: " + mensaje);  
 }  
 }  
  
 *public static int* dividirPorCero() {  
 *// Intentamos dividir por cero, lo que provocará una excepción  
 return* 10 / 0;  
 }  
}

1. lanzará una excepción NumberFormatException al intentar convertir la cadena "hola" a un entero utilizando el método Integer.parseInt(). Esto sucede porque la cadena "hola" no puede ser convertida a un entero válido.

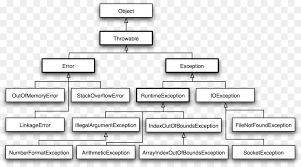
*public class* Main {  
 *public static void* main(String[] args) {  
 String aux = "hola";  
 *try* {  
 *int* aux2 = Integer.parseInt(aux);  
 System.out.println("El valor convertido es: " + aux2);  
 } *catch* (NumberFormatException e) {  
 System.out.println("Se ha producido una excepción: " + e.getMessage());  
 }  
 }  
}

1. .

*import* java.io.File;  
*import* java.io.FileReader;  
*import* java.io.IOException;  
  
*public class* Main {  
 *public static void* main(String[] args) {  
 *try* {  
 *// Intentamos abrir un archivo que no existe* FileReader fileReader = *new* FileReader("archivo\_que\_no\_existe.txt");  
 *// Si no ocurre ninguna excepción, continuamos con el flujo del programa* System.out.println("El archivo se ha abierto correctamente.");  
 } *catch* (IOException e) {  
 *// Capturamos la excepción y manejamos el error* System.out.println("Se ha producido un error al abrir el archivo:");  
 *// Imprimimos la información de la excepción* System.out.println(e.getMessage());  
 *// Opcionalmente, podríamos imprimir la traza completa de la excepción  
 // e.printStackTrace();* }  
 }  
}

1. .

*public class* Main {  
 *public static void* main(String[] args) {  
 *try* {  
 metodoAnidado();  
 } *catch* (Exception e) {  
 System.out.println("Se ha producido una excepción:");  
 System.out.println(e.getMessage());  
 }  
 }  
  
 *public static void* metodoAnidado() *throws* Exception {  
 *try* {  
 metodoUno();  
 } *catch* (ArithmeticException e) {  
 System.out.println("Se ha producido una ArithmeticException en metodoAnidado:");  
 System.out.println(e.getMessage());  
 } *catch* (NullPointerException e) {  
 System.out.println("Se ha producido una NullPointerException en metodoAnidado:");  
 System.out.println(e.getMessage());  
 } *catch* (ArrayIndexOutOfBoundsException e) {  
 System.out.println("Se ha producido una ArrayIndexOutOfBoundsException en metodoAnidado:");  
 System.out.println(e.getMessage());  
 }  
 }  
  
 *public static void* metodoUno() *throws* Exception {  
 *// Simulamos una excepción ArithmeticException  
 int* resultado = 10 / 0;  
  
 *// Simulamos una excepción NullPointerException* String cadena = *null*;  
 *int* longitud = cadena.length();  
  
 *// Simulamos una excepción ArrayIndexOutOfBoundsException  
 int*[] arreglo = *new int*[5];  
 *int* valor = arreglo[10];  
 }  
}

1. 

IOException: Esta es la clase base para excepciones producidas durante la operación de entrada/salida. Por ejemplo, al intentar leer o escribir archivos, trabajar con sockets, entre otros. Algunas de sus subclases notables incluyen FileNotFoundException, EOFException y SocketException.

NullPointerException: Esta excepción se lanza cuando intentas acceder a un miembro (método o campo) de un objeto que es nulo. Por ejemplo, si intentas llamar a un método en un objeto que no ha sido inicializado.

IllegalArgumentException: Esta excepción se lanza cuando se pasa un argumento ilegal o inapropiado a un método. Por ejemplo, si un método espera un valor dentro de un rango específico y se le pasa un valor fuera de ese rango, se lanzará esta excepción.

ArrayIndexOutOfBoundsException: Esta excepción se lanza cuando intentas acceder a un índice fuera del rango válido de un arreglo. Por ejemplo, si intentas acceder al elemento en la posición 5 de un arreglo que solo tiene 3 elementos, se lanzará esta excepción. También hay una subclase específica llamada StringIndexOutOfBoundsException, que se lanza al intentar acceder a un índice fuera del rango válido de una cadena.

1. public class CustomException extends Exception {

public CustomException(String message) {

super(message);

}

}

public class SubCustomException extends CustomException {

public SubCustomException(String message) {

super(message);

}

}

public class Main {

public static void main(String[] args) {

try {

// Lanzando CustomException

throw new CustomException("Este es un error personalizado");

} catch (CustomException e) {

System.out.println("Se ha producido una CustomException: " + e.getMessage());

}

try {

// Lanzando SubCustomException

throw new SubCustomException("Este es un error personalizado más específico");

} catch (SubCustomException e) {

System.out.println("Se ha producido una SubCustomException: " + e.getMessage());

} catch (CustomException e) {

System.out.println("Se ha producido una CustomException: " + e.getMessage());

}

}

}

1. El programa imprimirá 3 en la salida. Aquí está el flujo de ejecución:

Se llama al método devuelveNumero(1) desde el método main.

Dentro del método devuelveNumero, se comprueba si el número pasado es par (num % 2 == 0). Como 1 no es par, no se lanza la excepción y se procede a la declaración return 1;.

Antes de que se ejecute la declaración return 1;, el bloque finally se ejecuta independientemente de si se lanzó una excepción o no. En este caso, la declaración return 3; dentro del bloque finally sobrescribe cualquier valor de retorno anterior.

Por lo tanto, el método devuelveNumero retorna 3.

La salida del programa será 3.