2️⃣

Control de excepciones

Una excepción es un error que ocurre en tiempo de ejecución.

Utilizando el subsistema de manejo de excepciones de Java, puede, de una manera estructurada y controlada, manejar los errores de tiempo de ejecución.

Aunque la mayoría de los lenguajes de programación modernos ofrecen algún tipo de manejo de excepciones, el soporte de Java es fácil de usar y flexible.

El manejo de excepciones agiliza el manejo de errores al permitir que tu programa defina un bloque de código, llamado manejador de excepción, que se ejecuta automáticamente cuando ocurre un error. No es necesario verificar manualmente el éxito o el fracaso de cada operación específica o llamada a un método. Si se produce un error, será procesado por el manejador de excepciones.

▼ Jerarquía de excepciones

En Java, todas las excepciones están representadas por clases. Todas las clases de excepción se derivan de una clase llamada Throwable. Por lo tanto, cuando se produce una excepción en un programa, se genera un objeto de algún tipo de clase de excepción.

Hay dos subclases directas de Throwable:

- Error: las excepciones de tipo Error están relacionadas con errores que ocurren en la Máquina Virtual de Java y no en tu programa. Este tipo de excepciones escapan a su control y, por lo general, tu programa no se ocupará de ellas. Por lo tanto, este tipo de excepciones no se describen aquí.
- Exception: los errores que resultan de la actividad del programa están representados por subclases de Exception. Por ejemplo, dividir por cero, límite de matriz y errores de archivo caen en esta categoría. En general, tu programa debe manejar excepciones de estos tipos. Una subclase importante de Exception es RuntimeException, que se usa para representar varios tipos comunes de errores en tiempo de ejecución.

▼ Fundamentos de manejo de excepciones

El manejo de excepciones Java se gestiona a través de cinco palabras clave: try, catch, throw, throws y finally.

Las declaraciones de programa que desea supervisar para excepciones están contenidas dentro de un bloque try. Si se produce una excepción dentro del bloque try, se lanza. El código puede atrapar esta excepción usando catch y manejarlo de una manera racional. Las excepciones generadas por el sistema son lanzadas automáticamente por el sistema en tiempo de ejecución de Java. Para lanzar manualmente una excepción, se usa la palabra clave throw. En algunos casos, una excepción arrojada por un método debe ser especificada como tal por una cláusula throws. Cualquier código que debe ejecutarse al salir de un bloque try se coloca en un bloque finally.

▼ Uso de try y catch

En el centro del manejo de excepciones están try y catch. Estas palabras clave trabajan juntas; no puedes atrapar (catch) sin intentarlo (try). Aquí está la forma general de los bloques de manejo de excepciones try/catch:

```
public static void main(String[] args) {
  try {
    //Bloque de codigo para monitorizar errores
  } catch(TipoExcepcion1 exOb) {
    //Manejador para TipoExcepcion1
  } catch(TipoExcepcion2 exOb) {
    //Manejador para TipoExcepcion2
  }
}
```

Aquí, TipoExcepcion es el tipo de excepción que ha ocurrido. Cuando se lanza una excepción, es atrapada por su instrucción catch correspondiente, que luego procesa la excepción. Como muestra la forma general, puede haber más de una declaración catch asociada con un try. El tipo de la excepción determina qué declaración de captura se ejecuta. Es decir, si el tipo de excepción especificado por una instrucción catch coincide con el de la excepción, entonces se ejecuta esa instrucción de catch (y todos los demás se anulan). Cuando se detecta una excepción, exOb recibirá su valor.

Si no se lanza una excepción, entonces un bloque try finaliza normalmente, y todas sus declaraciones catch se pasan por alto. La ejecución se reanuda con la primera instrucción después del último catch. Por lo tanto, las declaraciones catch se ejecutan solo si se lanza una excepción.

▼ Un ejemplo de excepción simple:

```
public class ExcDemo {
  public static void main(String[] args) {
    int nums[]=new int[4];
    try {
      System.out.println("Antes de que se genere la excepcion");
      //Generar una excepcion de indice fuera de limites
      num[7]=10;
    } catch(ArrayIndesOutOfBoundsException exc) {
      //Capturando excepcion
      System.out.println("Indice fuera de los limites");
    }
      System.out.println("Despues de que se genere la excepcion");
  }
}
```

El programa anterior ilustra varios puntos clave sobre el manejo de excepciones:

- Primero, el código que desea revisar para detectar errores está dentro de un bloque try.
- En segundo lugar, cuando se produce una excepción (en este caso, debido al intento de indexar nums más allá de sus límites), la excepción se emite desde el bloque try y es atrapada por la instrucción catch. En este punto, el control pasa al catch, y el bloque try finaliza.
- Es decir, no se llama a catch. Por el contrario, la ejecución del programa se transfiere a él. Por lo tanto, la instrucción que sigue a nums[7]=10; nunca se ejecutará.
- Después de que se ejecuta la instrucción catch, el control del programa continúa con las declaraciones que siguen el catch. Por lo tanto, es el trabajo de tu controlador de excepción remediar el problema que causó la excepción para que la ejecución del programa pueda continuar normalmente.

Si no se lanza una excepción por un bloque try, no se ejecutarán declaraciones catch y el control del programa se reanudará después de la instrucción catch. Para confirmar esto, en el programa anterior, cambiamos la línea, `nums[7]=10;` por `nums[0]=10;`

```
class ExcDemo {
  public static void main(String[] args) {
    int nums[]=new int[4];
    try {
      System.out.println("Antes de que se genere la excepcion");
      //Generar una excepcion de indice fuera de limites
      num[0]=10;
    } catch(ArrayIndesOutOfBoundsException exc) {
      //Capturando excepcion
      System.out.println("Indice fuera de los limites");
    }
      System.out.println("Despues de que se genere la excepcion");
  }
}
```

Ahora, no se genera ninguna excepción, y el bloque catch no se ejecuta.

▼ Captura de excepciones de subclase

Hay un punto importante sobre declaraciones de múltiples catch que se relaciona con subclases. Una cláusula catch para una superclase también coincidirá con cualquiera de sus subclases.

Por ejemplo, dado que la superclase de todas las excepciones es Throwable, para atrapar todas las excepciones posibles, capture Throwable. Si desea capturar excepciones de un tipo de superclase y un tipo de subclase, coloque la subclase primero en la secuencia de catch. Si no lo hace, la captura de la superclase también atrapará todas las clases derivadas. Esta regla se autoejecuta porque poner primero la superclase hace que se cree un código inalcanzable, ya que la cláusula catch de la subclase nunca se puede ejecutar.

En Java, el código inalcanzable es un error.

```
//En las subclases deben preceder a las superclases en las declaraciones catch
public class ExcDemo {
  public static void main(String[] args) {
    //Aqui num, es mas grande que denom
    int nums[]={4,8,16,32,64,128,256,512};
```

```
    int denom[]={2,0,0,4,4,0,8};
    for(int i=0;i<nums.lentgh;i++) {
      try {
        System.out.println(nums[i]+" / "+denom[i]+" es "nums[i]/demon[i]);
      } catch(ArrayIndesOutOfBoundsException exc) {
        //Capturando excepcion (subclase)
        System.out.println("No se encontro ningun elemento");
      } catch(Throwable exc) {
        //Capturando excepcion (superclase)
        System.out.println("Alguna excepcion ocurrio");
      }
    }
  }
}
```

Un bloque try se puede anidar dentro de otro. Una excepción generada dentro del bloque try interno, que no está atrapada por un catch asociado con este try, se propaga al bloque try externo. Por ejemplo, aquí, la ArrayIndexOutOfBoundsException no es capturada por el catch interno, sino por el catch externo:

```
public class TryAnidado {
  public static void main(String[] args) {
    //Aqui num, es mas grande que denom
    int nums[]={4,8,16,32,64,128,256,512};
    int denom[]={2,0,0,4,4,0,8};
    try { //try externo
        for(int i=0;i<nums.lentgh;i++) {
          try { //try anidado
            System.out.println(nums[i]+" / "+denom[i]+" es "nums[i]/demon[i]);
          } catch(ArithmeticException exc) {
              //Capturando la excepcion
              System.out.println("No se puede dividir por cero");
          }
        }
    } catch(ArrayIndesOutOfBoundsException exc) {
        //Capturando excepcion
        System.out.println("Alguan excepcion ocurrio");
        System.out.println("Error: Programa terminado");
    }
  }
}
```

En este ejemplo, una excepción que puede ser manejada por el try interno, en este caso, un error de división por cero, permite que el programa continúe. Sin

embargo, un error de límite de matriz es capturado por la try externo, lo que hace que el programa finalice.

▼ Lanzar una excepción

Los ejemplos anteriores han estado capturando excepciones generadas automáticamente por la JVM. Sin embargo, es posible lanzar manualmente una excepción utilizando la instrucción throw.

Aquí, excepcOb debe ser un objeto de una clase de excepción derivada de Throwable. Aquí hay un ejemplo que ilustra la instrucción throw arrojando manualmente una ArithmeticException:

```
//Lanzar una excepcion manualmente
public class ThrowDemo {
  public static void main(String[] args) {
    try {
      System.out.println("Antes de lanzar excepcion");
      throw new ArithmeticException(); //Lanzar excepcion
    } catch(TipoExcepcion1 exOb) {
      //Capturando excepcion
      System.out.println("Excepcion capturada");
    }
    System.out.println("Despues del bloque try/catch");
  }
}
```

▼ Una mirada más cercana a Throwable

Hasta este punto, hemos estado detectando excepciones, pero no hemos estado haciendo nada con el objeto de excepción en sí mismo. Como muestran todos los ejemplos anteriores, una cláusula catch especifica un tipo de excepción y un parámetro. El parámetro recibe el objeto de excepción. Como todas las excepciones son subclases de Throwable, todas las excepciones admiten los métodos definidos por Throwable. Varios de uso común se muestran en la siguiente tabla:

Método	Sintaxis	Descripción
getMessage	String getMessage()	Devuelve una descripción de la excepción

Método	Sintaxis	Descripción
getLocalizedMessage	String getLocalizedMessage()	Devuelve una descripción localizada de la excepción
toString	String toString()	Devuelve un objeto String que contiene una descripción completa de la excepción. Este método lo llama println() cuando se imprime un objeto Throwable
printStackTrace()	void printStackTrace()	Muestra el flujo de error estándar
printStackTrace	void printStackTrace(PrintStreams)	Envía la traza de errores a la secuencia especificada
printStackTrace	void printStackTrace(PrintWriters)	Envía la traza de errores a la secuencia especificada
fillInStackTrace	Throwable fillInStackTrace()	Devuelve un objeto Throwable que contiene un seguimiento de pila completo. Este objeto se puede volver a lanzar

De los métodos definidos por Throwable, dos de los más interesantes son printStackTrace() y toString().

- Puede visualizar el mensaje de error estándar más un registro de las llamadas a métodos que conducen a la excepción llamando a printStackTrace().
- Puede usar toString() para recuperar el mensaje de error estándar. El método toString() también se invoca cuando se usa una excepción como argumento para println().

El siguiente programa demuestra estos métodos:

```
public class ExcDemo {
  static void genException() {
    int nums[]=new int[4];
    System.out.println("Antes de lanzar excepcion");
    num[7]=10;
    System.out.println("Esto no se mostrara");
  }
}
class ExcDemo {
  public static void main(String[] args) {
    try {
```

```
      ExcDemo.genException();
    } catch(ArrayIndexOutOfBounds exc) {
      System.out.println("Mensaje estandar: ");
      System.out.println(exc);
      System.out.println("\nTraza de errores: ");
      exc.printStrackTrace();
    }
    System.out.println("Despues del bloque catch");
  }
}
```

▼ Uso de finally

Algunas veces querrá definir un bloque de código que se ejecutará cuando quede un bloque try/catch. Por ejemplo, una excepción puede causar un error que finaliza el método actual, causando su devolución prematura. Sin embargo, ese método puede haber abierto un archivo o una conexión de red que debe cerrarse.

Tales tipos de circunstancias son comunes en la programación, y Java proporciona una forma conveniente de manejarlos: finally. Para especificar un bloque de código a ejecutar cuando se sale de un bloque try/catch, incluya un bloque finally al final de una secuencia try/catch. Aquí se muestra la forma general de un try/catch que incluye finally.

```
try {
  //Bloque de codigo para monitorizar errores
} catch(TipoExcepcion1 exOb) {
  //Manejador para TipoExcepcion1
} catch(TipoExcepcion2 exOb) {
  //Manejador para TipoExcepcion2
} finally {
  //Codigo final
}
```

El bloque finally se ejecutará siempre que la ejecución abandone un bloque try/catch, sin importar las condiciones que lo causen. Es decir, si el bloque try finaliza normalmente, o debido a una excepción, el último código ejecutado es el definido por finally. El bloque finally también se ejecuta si algún código dentro del bloque try o cualquiera de sus declaraciones catch devuelve del método.

```
public class Finally {
  public static void main(String[] args) {
```

```
    int a=5;
    int b=0;
    try {
      int resultado=a/b;
      System.out.println(resultado);
    } catch(Exception e) {
      System.out.println("la aplicacion fallo");
      throw new NullPointerException();
    } finally {
      Sytem.out.println("se cierran los recursos");
    }
    System.out.println("la aplicacion ha finalizado");
  }
}
```

▼ Uso de throws

En algunos casos, si un método genera una excepción que no maneja, debe declarar esa excepción en una cláusula throws. Aquí está la forma general de un método que incluye una cláusula throws:

```
tipo-retorno nombreMetodo(lista-param) throws lista-excepc {
  // Cuerpo
}
```

Aquí, lista-excepc es una lista de excepciones separadas por comas que el método podría arrojar fuera de sí mismo.

Las excepciones que son subclases de Error o RuntimeException no necesitan ser especificadas en una lista de throws. Java simplemente asume que un método puede arrojar uno. Todos los otros tipos de excepciones deben ser declarados. De lo contrario, se produce un error en tiempo de compilación.

```
class Prueba {
  static void demoproc() throws IllegalAccessException {
    System.out.println("Dentro de demoproc");
    throw new IllegalAccessException("demo");
  }
  public static void main(String[] args) {
    try {
      demoproc();
    } catch(IllegalAccessException e) {
      System.out.println("Capturada de nuevo: "+e);
    }
```

```
  }
}
```