CURSO DE PROGRAMACIÓN FULL STACK

MANEJO DE EXCEPCIONES

PARADIGMA ORIENTADO A OBJETOS

GUÍA DE MANEJO DE EXCEPCIONES

EXCEPCIONES

El término excepción es una abreviación de la frase "Evento Excepcional". Una *excepción* es un evento que ocurre durante la ejecución de un programa que interrumpe el flujo normal de las instrucciones del programa.

Existen muchas clases de errores que pueden provocar una excepción, desde un desbordamiento de memoria o un disco duro estropeado hasta un intento de dividir por cero o intentar acceder a un vector fuera de sus límites. Cuando esto ocurre, la máquina virtual Java crea un objeto de la clase *exception* o *error* y se notifica el hecho al sistema de ejecución. Se dice que se ha *lanzado una excepción* ("*Throwing Exception*"). Luego, el objeto, llamado excepción, contiene información sobre el error, incluyendo su tipo y el estado del programa cuando el error ocurrió.

Después de que un método lanza una excepción, el sistema, en tiempo de ejecución, intenta encontrar algo que maneje esa excepción. El conjunto de posibles "algo" para manejar la excepción es la lista ordenada de los métodos que habían sido llamados hasta llegar al método que produjo el error. Esta lista de métodos se conoce como *pila de llamadas*. Luego, el sistema en tiempo de ejecución busca en la pila de llamadas el método que contenga un bloque de código que pueda manejar la excepción. Este bloque de código es llamado *manejador de excepciones*.

Concretamente, una *excepción* en java es *un objeto que modela un evento excepcional*, el cual *no debería haber ocurrido*. Como observamos anteriormente, al ocurrir estos tipos de evento la máquina virtual no debe continuar con la ejecución normal del programa. Es evidente que las excepciones son objetos especiales, son objetos con la capacidad de cambiar el flujo normal de ejecución. Cuando se detecta un error, una excepción debe ser lanzada.

Ejemplos de situaciones que provocan una excepción:

- No hay memoria disponible para asignar
- Acceso a un elemento de un array fuera de rango
- Leer por teclado un dato de un tipo distinto al esperado
- Error al abrir un fichero
- División por cero

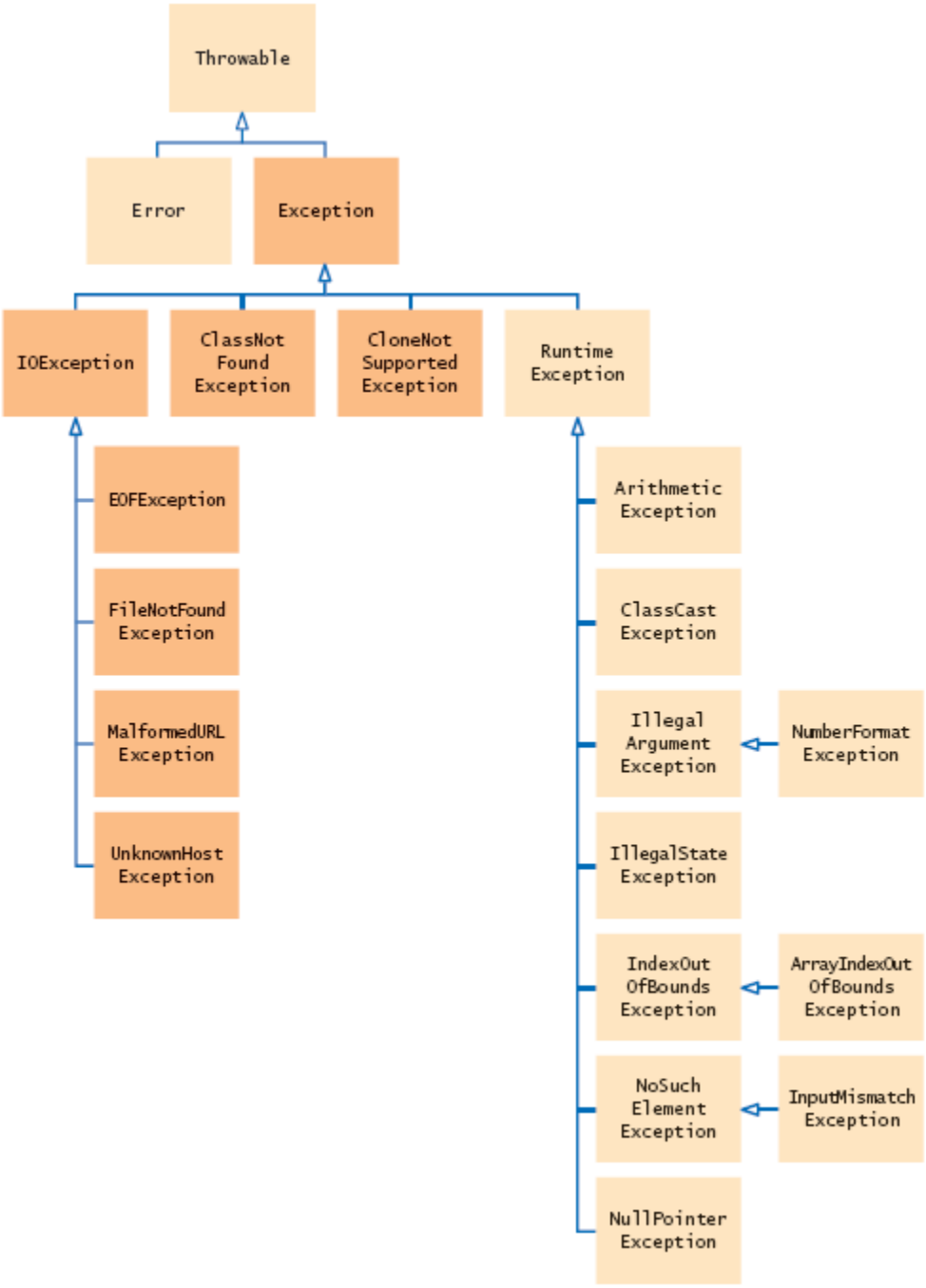
JERARQUIA DE EXCEPCIONES

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: Exception y Error:

1. 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.

2. 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.



MANEJADOR DE EXCEPCIONES

El manejo de excepciones Java se gestiona a través de cinco palabras clave: **try, catch, throw, throws,** y **finally**. Forman un subsistema interrelacionado en el que el uso de uno implica el uso de otro.

Las declaraciones del programa que desea supervisar para excepciones están contenidas dentro de un bloque try. Si se produce una excepción dentro del bloque **try**, se lanza. Tu 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 de tiempo de ejecución de Java. Para lanzar manualmente una excepción, use 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**.

Ahora vamos a ver en detalle cada palabra clave dentro del manejo de excepciones.

El bloque try

Lo primero que hay que hacer para que un método sea capaz de tratar una excepción generada por la máquina virtual Java o por el propio programa mediante una instrucción *throw* es encerrar las instrucciones susceptibles de generarla en un bloque **try**. En el bloque try vamos a poner una serie de instrucciones que creemos que puede llegar a tirar una excepción durante su ejecución y queremos manejarla para evitar la finalización del programa.

```
try {
  Instrucción1;
  Intruscción2;
  Instrucción3;
  Instrucción4
  ...
}
```

Cualquier excepción que se produzca por alguna instrucción, dentro del bloque *try* será analizada por el bloque o bloques *catch*. En el momento en que se produzca la excepción, se abandona el bloque try, y las instrucciones que sigan al punto donde se produjo la excepción no son ejecutadas. Cada bloque try debe tener asociado por lo menos un bloque catch.

El bloque catch

Por cada bloque *try* pueden declararse uno o varios bloques *catch*, cada uno de ellos capaz de tratar un tipo u otro de excepción. Para declarar el tipo de excepción que es capaz de tratar un bloque *catch*, se declara un objeto cuya clase es la clase de la excepción que se desea tratar o una de sus superclases.

```
try {
	BloqueDeIntrucciones
} catch (TipoExcepción nombreVariable) {
	BloqueCatch
} catch (TipoExcepción nombreVariable) {
	BloqueCatch
}
```

Al producirse la excepción dentro de un bloque *try*, la ejecución del programa se pasa al primer bloque *catch*. Si la clase de la excepción se corresponde con la clase o alguna subclase de la clase declarada en el bloque catch, se ejecuta el bloque de instrucciones catch y a continuación se pasa el control del programa a la primera instrucción a partir de los bloques *try-catch*. Lo más adecuado es utilizar excepciones lo más cercanas al tipo de error previsto, ya que lo que se pretende es recuperar al programa de alguna condición de error y si "se meten todas las condiciones en la misma bolsa", seguramente habrá que averiguar después qué condición de error se produjo para poder dar una respuesta adecuada.

```
try {
// Se intenta hacer la división
int division = 10 / 0;
} catch (ArithmeticException a) {
// Si la división falla el programa va al bloque catch y se ejecuta el
System.out.println
System.out.println("Error: división por cero");
}
```

En este ejemplo en el bloque try hacemos una división por cero, las divisiones por cero generan un tipo de excepción llamado, ArithmeticExcepction. En el bloque catch ponemos como tipo de excepción la ArithmeticException y dentro del bloque ponemos un mensaje que explique cual ha sido el error.

Métodos Throwable

Dentro del bloque catch, utilizamos un System.out.print para mostrar el error, 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.

Estos métodos son:

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

fillInStackTrace	Throwable fillInStackTrace()	Devuelve un objeto Throwable que contiene un seguimiento de pila completo. Este objeto se puede volver a lanzar.
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.

```
try {
int division = 10 / 0;
} catch (ArithmeticException a) {
System.out.println(“Error:” + a.getMessage());
System.out.println(“Error:” + a)
System.out.println(a.fillStrakTrace());
}
Resultado
Error: / by zero
Error: / by zero
Error: java.lang.ArithmeticException: / by zero
java.lang.ArithmeticException: / by zero
```

El bloque finally

El bloque finally se utiliza para ejecutar un bloque de instrucciones sea cual sea la excepción que se produzca. Este bloque se ejecutará siempre, incluso si no se produce ninguna excepción. Sirve para no tener que repetir código en el bloque try y en los bloques catch. El bloque finally es un buen lugar en donde liberar los recursos tomados dentro del bloque de intento.

```
try {
	BloqueDeIntrucciones
} catch (TipoExcepción nombreVariable) {
	MensajeDeError
} catch (TipoExcepción nombreVariable) {
	MensajeDeError
} finally {
	CodigoFinal
}
```

```
try {

// Se intenta hacer la división

int division = 10 / 0;

} catch (ArithmeticException a) {

// Si la división falla el programa va al bloque catch y se ejecuta el
System.out.println

System.out.println(“Error: división por cero”);

} finally {

// Si el programa hizo la división o no, este System.out.print se va a ejecutar
igual

System.out.println(“Saliendo del try”);

}
```

La claúsula throws

La cláusula *throws* lista las excepciones que un método puede lanzar. Los tipos de excepciones que lanza el método se especifica después de los paréntesis de un método, con una cláusula *throws*. Un método puede lanzar objetos de la clase indicada o de subclases de la clase indicada.

Java distingue entre las excepciones verificadas y errores. Las excepciones verificadas deben aparecer en la cláusula *throws* de ese método. Como las RuntimeExceptions y los Errores pueden aparecer en cualquier método, no tienen que listarse en la cláusula *throws* y es por esto que decimos que no están verificadas. Todas las excepciones que no son RuntimeException y que un método puede lanzar deben listarse en la cláusula *throws* del método y es por eso que decimos que están verificadas. El requisito de atrapar excepciones en Java exige que el programador atrape todas las excepciones verificadas o bien las coloque en la cláusula *throws* de un método.

Si la excepción no se trata, el manejador de excepciones realiza lo siguiente:

- Muestra la descripción de la excepción.
- Muestra la traza de la pila de llamadas.
- Provoca el final del programa.

Colocar una excepción en la cláusula *throws* obliga a otros métodos a ocuparse de la excepción. Esto se puede hacer colocando otro *throws* al método que llama al método, con el tipo de excepción que podría tirar o rodear el llamado del método con un try-catch, y de esa manera que el try-catch se encargue de manejar la excepción que podría tirar el método.

```
[acceso][modificador][tipo] nombreFuncion() throws TipoDeExcepcion {

Bloque de instrucciones

}
```

```
// Tenemos un método que devuelve un resultado y que tira una ArithmeticException

public int division() throws ArithmeticException {

int division;

// Hacemos la división, si la división tira una excepción la manejará el try/catch del llamado a la función

division = 20 / 0;

// Si tira una excepción el método no va a devolver ningún resultado

return division;

}

Main

try {

// Llamamos a la función, si tira una excepción va al bloque catch y ejecuta el mensaje de error, sino imprime el resultado de la división.

System.out.println(division());

} catch (ArithmeticException a) {

System.out.println("Error: división por cero");

}
```

La palabra throw

Los programas escritos en Java pueden lanzar excepciones explícitamente mediante la instrucción *throw*, lo que facilita la devolución de un "código de error" al método que invocó el método que causó el error. La cláusula *throw* debe ir seguida del tipo de excepción que queremos que lance el método. Puede lanzarse cualquier tipo de excepción que implemente la interfaz *Throwable*.

Cuando se lanza una excepción usando la palabra throw, el flujo de ejecución del programa se detiene y el control se transfiere al bloque try-catch más cercano que coincida con el tipo de excepción lanzada. Si no se encuentra tal coincidencia, el controlador de excepciones predeterminado finaliza el programa. La palabra clave throw es útil para lanzar excepciones basadas en ciertas condicione, por ejemplo, si un usuario ingresa datos incorrectos. También es útil para lanzar excepciones personalizadas específicas para un programa o aplicación.

Cuando utilicemos la palabra throw en un método, vamos a tener que agregarle la palabra throws al método con la excepción que va a tirar nuestro throw. De esa manera avisamos que cuando se llame al método hay que manejar una posible excepción.

```
throw new TipoExcepcion("Mensaje de error");
```

```
// En este método recibimos una lista y un numero para agregar a dicha lista. El método contiene la palabra throws para avisar que este método puede tirar una excepción

public void agregarNumeroLista(List<Integer> lista, int numero) throws Exception{

// Validamos si la lista ya tiene el numero a agregar

if(lista.contains(numero){

// Si lo tiene tiramos un excepción de tipo Exception, poniéndole un mensaje entre los paréntesis

throw new Exception(“El numero ya está en la lista);

}

// Si no contiene el numero, lo agregamos a la lista

lista.add(numero);

Main

List<Integer> lista = new ArrayList();

try{

//Llamamos al método dentro de un try/catch para manejar la posible excepción

agregarNumeroLista(lista, 1);

catch (Exception e){

//Usamos el metodo getMessage, para obtener el mensaje que pusimos en el throw

System.out.println(e.getMessage);

}
```

PREGUNTAS DE APRENDIZAJE

1) La clase Error maneja errores:

   a) Del código
   b) De la Maquina Virtual de Java
   c) De Netbeans
   d) Ninguna de las anteriores

2) La clase Exception maneja errores:

   a) Del código
   b) De la Maquina Virtual de Java
   c) De Netbeans
   d) Ninguna de las anteriores

3) Las excepciones se manejan con el bloque:

   a) Finally
   b) Throws
   c) Try Catch
   d) Throw

4) Para cerrar recursos después de una operación usamos el bloque:

   a) Finally
   b) Throws
   c) Try Catch
   d) Throw

5) Cuando queremos informar que un método puede tirar una excepción usamos el bloque:

   a) Finally
   b) Throws
   c) Try Catch
   d) Throw

6) Cuando queremos lanzar una excepción de manera explicita usamos el bloque:

   a) Finally
   b) Throws
   c) Try Catch
   d) Throw

7) En Java la diferencia entre throws y throw es:

   a) throws arroja una excepción y throw indica el tipo de excepción que no maneja el método
   b) throws se usa en los metodos y throw en los constructores
   c) throws indica el tipo de excepción que no maneja el método y throw arroja una excepción
   d) Ninguna de las anteriores

EJERCICIOS DE APRENDIZAJE

En este módulo vamos a empezar a manejar los errores y las excepciones de nuestro código para poder seguir trabajando sin que el código se detenga

MANEJO DE EXCEPCIONES

VER VIDEOS:

A. Excepciones I
B. Excepciones II

1. Inicializar un objeto de la clase Persona ejercicio 7 de la guía POO, a null y tratar de invocar el método esMayorDeEdad() a través de ese objeto. Luego, englobe el código con una cláusula try-catch para probar la nueva excepción que debe ser controlada.

2. Definir una Clase que contenga algún tipo de dato de tipo array y agregue el código para generar y capturar una excepción ArrayIndexOutOfBoundsException (índice de arreglo fuera de rango).

3. Defina una clase llamada DivisionNumero. En el método main utilice un Scanner para leer dos números en forma de cadena. A continuación, utilice el método parseInt() de la clase Integer, para convertir las cadenas al tipo int y guardarlas en dos variables de tipo int. Por ultimo realizar una división con los dos numeros y mostrar el resultado.
   Todas estas operaciones puede tirar excepciones a manejar, el ingreso por teclado puede causar una excepción de tipo InputMismatchException, el método Integer.parseInt() si la cadena no puede convertirse a entero, arroja una NumberFormatException y además, al dividir un número por cero surge una ArithmeticException. Manipule todas las posibles excepciones utilizando bloques try/catch para las distintas excepciones

4. Escribir un programa en Java que juegue con el usuario a adivinar un número. La computadora debe generar un número aleatorio entre 1 y 500, y el usuario tiene que intentar adivinarlo. Para ello, cada vez que el usuario introduce un valor, la computadora debe decirle al usuario si el número que tiene que adivinar es mayor o menor que el que ha introducido el usuario. Cuando consiga adivinarlo, debe indicárselo e imprimir en pantalla el número de veces que el usuario ha intentado adivinar el número. Si el usuario introduce algo que no es un número, se debe controlar esa excepción e indicarlo por pantalla. En este último caso también se debe contar el carácter fallido como un intento.

5. Dado el método metodoA de la clase A, indique:

a) ¿Qué sentencias y en qué orden se ejecutan si se produce la excepción MioException?

b) ¿Qué sentencias y en qué orden se ejecutan si no se produce la excepción MioException?

```
class A {
void metodoA() {
sentencia_a1
sentencia_a2
try {
 sentencia_a3
 sentencia_a4
} catch (MioException e){
sentencia_a6
}
sentencia_a5
}
}
```

6. Dado el método metodoB de la clase B, indique:

a) ¿Qué sentencias y en qué orden se ejecutan si se produce la excepción MioException?
b) ¿Qué sentencias y en qué orden se ejecutan si no se produce la excepción MioException?

```
class B {
void metodoB() {
      sentencia_b1
try {
sentencia_b2
} catch (MioException e){
sentencia_b3
}
finally
      sentencia_b4
}
}
```

7. Indique que se mostrará por pantalla cuando se ejecute cada una de estas clases:

```
class Uno{
    private static int metodo() {
           int valor=0;
           try {
                  valor = valor+1;
                  valor = valor + Integer.parseInt (”42”);
                  valor = valor +1;
                  System.out.println(”Valor final del try:” + valor) ;
           } catch (NumberFormatException e) {
```

```
                Valor = valor + Integer.parseInt(”42”);


                System.out.println(“Valor final del catch:” + valor);
        } finally {
                valor = valor + 1;
                System.out.println(”Valor final del finally: ” + valor) ;
        }
        valor = valor +1;
        System.out.println(”Valor antes del return: ” + valor) ;
        return valor;
    }

    public static void main (String[] args) {
        try {
                System.out.println (metodo()) ;
        }catch(Exception e) {
                System.err.println(”Excepcion en metodo() ”) ;
                e.printStackTrace();
        }
    }
}

class Dos{
    private static metodo() {
        int valor=0;
        try{
                valor = valor + 1;
                valor = valor + Integer.parseInt (”W”);
                valor = valor + 1;
                System.out.println(”Valor final del try: ” + valor) ;
        } catch ( NumberFormatException e ) {
                valor = valor + Integer.parseInt (”42”);
                System.out.println(”Valor final del catch: ” + valor) ;
        } finally {
                valor = valor + 1;
                System.out.println(”Valor final del finally: ” + valor) ;
```

```
            }
            valor = valor + 1;
            System.out.println(”Valor antes del return: ” + valor) ;
            return valor;
    }

    public static void main (String[] args) {
            try{
                    System.out.println ( metodo ( ) ) ;
            } catch(Exception e) {
                    System.err.println ( ” Excepcion en metodo ( ) ” ) ;
                    e.printStackTrace();
            }
    }
}

class Tres{
        private static metodo( ) {
                int valor=0;
                try{
                        valor = valor + 1;
                        valor = valor + Integer.parseInt (”W”);
                        valor = valor + 1;
                        System.out.println(”Valor final del try: ” + valor);
                } catch(NumberFormatException e) {
                        valor = valor + Integer.parseInt (”W”);
                        System.out.println(”Valor final del catch: ” + valor);
                } finally{
                        valor = valor + 1;
                        System.out.println(”Valor final del finally:” + valor);
                }
                valor = valor + 1;
                System.out.println(”Valor antes del return: ” + valor) ;
                return valor;
                }
```

```
        public static void main (String[] args) {
                try{
                        System.out.println( metodo ( ) ) ;
                } catch(Exception e) {
                        System.err.println("Excepcion en metodo ( ) " ) ;
                        e.printStackTrace();
                }
        }
}
```

8. Dado el método metodoC de la clase C, indique:

a) ¿Qué sentencias y en qué orden se ejecutan si se produce la excepción MioException?
b) ¿Qué sentencias y en qué orden se ejecutan si no se produce la excepción MioException?
c) ¿Qué sentencias y en qué orden se ejecutan si se produce la excepción TuException?

```
class C {
    void metodoC() throws TuException{
            sentencia_c1
            try {
                    sentencia_c2
                    sentencia_c3
            } catch (MioException e){
                    sentencia_c4

            } catch (TuException e){
                    sentencia_c5
                    throw (e)
            }
            finally
                    sentencia_c6
    }
}
```

IMPORTANTE: A partir de la próxima guía se debe aplicar en todos los ejercicios el manejo de excepciones cada vez que sea necesario controlar una posible excepción.

EJERCICIO INTEGRADOR COMPLEMENTARIO

Este ejercicio va a requerir que utilicemos todos conocimientos previamente vistos en esta y otras guías. Estos pueden realizarse cuando hayas terminado todas las guías y tengas una buena base sobre todo lo que veníamos trabajando. Además, si ya terminaste la guía y te queda tiempo libre en las mesas, podes continuar con este ejercicio complementario, recordando siempre que no es necesario que los termines para continuar con el tema siguiente. Este ejercicio, no lleva nota y es solamente para medir nuestros conocimientos. Por ultimo, recordá que la prioridad es ayudar a los compañeros de la mesa y que cuando tengas que ayudar, lo más valioso es que puedas explicar el ejercicio con la intención de que tu compañero lo comprenda, y no sólo mostrarlo. ¡Muchas gracias!

Armadura Iron Man:

J.A.R.V.I.S. es una inteligencia artificial desarrollada por Tony Stark. Está programado para hablar con voz masculina y acento británico. Actualmente se encarga de todo lo relacionado con la información doméstica de su casa, desde los sistemas de calefacción y refrigeración hasta los Hot Rod que Stark tiene en su garage.

Tony Stark quiere adaptar a J.A.R.V.I.S. para que lo asista en el uso de sus armaduras, por lo tanto, serás el responsable de llevar adelante algunas de estas tareas.

El objetivo de JARVIS es que analice intensivamente toda la información de la armadura y del entorno y en base a esto tome decisiones inteligentes.

En este trabajo se deberá crear en el proyecto una clase llamada Armadura que modele toda la información y las acciones que pueden efectuarse con la Armadura de Iron Man. La armadura de Iron Man es un exoesqueleto mecánico ficticio usado por Tony Stark cuando asume la identidad de Iron Man. La primera armadura fue creada por Stark y Ho Yinsen, mientras estuvieron prisioneros.

Las armaduras de Stark se encuentran definidas por un color primario y un color secundario. Se encuentran compuestas de dos propulsored, uno en cada bota; y dos repulsores, uno en cada guante (los repulsores se utilizan también como armas). Tony los utiliza en su conjunto para volar.

La armadura tiene un nivel de resistencia, que depende del material con el que está fabricada, y se mide con un número entero cuya unidad de medida de dureza es Rockwell (https://es.wikipedia.org/wiki/Dureza_Rockwell). Todas las armaduras poseen un nivel de salud el cual se mide de 0 a 100. Además, Tony tiene un generador, el cual le sirve para salvarle la vida en cada instante de tiempo alejando las metrallas de metal de su corazón y también para alimentar de energía a la armadura. La batería a pesar de estar en el pecho de Tony, es considerada como parte de la armadura.

La armadura también posee una consola en el casco, a través de la cual JARVIS le escribe información a Iron Man. En el casco también se encuentra un sintetizador por donde JARVIS susurra cosas al oído de Tony. Cada dispositivo de la armadura de Iron Man (botas, guantes, consola y sintetizador) tienen un consumo de energía asociado.

En esta primera etapa con una armadura podremos: caminar, correr, propulsar, volar, escribir y leer.

- Al caminar la armadura hará un uso básico de las botas y se consumirá la energía establecida como consumo en la bota por el tiempo en el que se camine.
- Al correr la armadura hará un uso normal de las botas y se consumirá el doble de la energía establecida como consumo en la bota por el tiempo en el que se corra.
- Al propulsarse la armadura hará un uso intensivo de las botas utilizando el triple de la energía por el tiempo que dure la propulsión.
- Al volar la armadura hará un uso intensivo de las botas y de los guantes un uso normal consumiendo el triple de la energía establecida para las botas y el doble para los guantes.
- Al utilizar los guantes como armas el consumo se triplica durante el tiempo del disparo.
- Al utilizar las botas para caminar o correr el consumo es normal durante el tiempo que se camina o se corra.
- Cada vez que se escribe en la consola o se habla a través del sintetizador se consume lo establecido en estos dispositivos. Solo se usa en nivel básico.
- Cada vez que se efectúa una acción se llama a los métodos usar del dispositivo se le pasa el nivel de intensidad y el tiempo. El dispositivo debe retornar la energía consumida y la armadura deberá informar al generador se ha consumido esa cantidad de energía.

Modele las clases, los atributos y los métodos necesarios para poder obtener un modelo real de la armadura y del estado de la misma.

Mostrando Estado

Hacer un método que JARVIS muestre el estado de todos los dispositivos y toda la información de la Armadura.

Estado de la Batería

Hacer un método para que JARVIS informe el estado de la batería en porcentaje a través de la consola. Poner como carga máxima del reactor el mayor float posible. Ejecutar varias acciones y mostrar el estado de la misma.

Mostrar Información del Reactor

Hacer un método para que JARVIS informe el estado del reactor en otras dos unidades de medida. Hay veces en las que Tony tiene pretensiones extrañas. Buscar en Wikipedia la tabla de transformaciones.

Sufriendo Daños

A veces los dispositivos de la armadura sufren daños para esto cada dispositivo contiene un atributo público que dice si el dispositivo se encuentra dañado o no. Al utilizar un dispositivo existe un 30% de posibilidades de que se dañe.

La armadura solo podrá utilizar dispositivos que no se encuentren dañados.

Modifique las clases que sean necesarias para llevar adelante este comportamiento.

Reparando Daños

Hay veces que se puede reparar los daños de un dispositivo, en general es el 40% de las veces que se puede hacer. Utilizar la clase Random para modelar este comportamiento. En caso de estar dentro de la probabilidad (es decir probabilidad menor o igual al 40%) marcar el dispositivo como sano. Si no dejarlo dañado.

Revisando Dispositivos

Los dispositivos son revisados por JARVIS para ver si se encuentran dañados. En caso de encontrar un dispositivo dañado se debe intentar arreglarlo de manera insistente. Para esos intentos hay un 30% de posibilidades de que el dispositivo quede destruido, pero se deberá intentar arreglarlo hasta que lo repare, o bien hasta que quede destruido.

Hacer un método llamado revisar dispositivos que efectúe lo anteriormente descrito, el mecanismo insistente debe efectuarlo con un bucle do while.

Radar Versión 1.0

JARVIS posee también incorporado un sistema que usa ondas electromagnéticas para medir distancias, altitudes, ubicaciones de objetos estáticos o móviles como aeronaves barcos, vehículos motorizados, formaciones meteorológicas y por su puesto enemigos de otro planeta.

Su funcionamiento se basa en emitir un impulso de radio, que se refleja en el objetivo y se recibe típicamente en la misma posición del emisor.

Las ubicaciones de los objetos están dadas por las coordenadas X, Y y Z. Los objetos pueden ser marcados o no como hostiles. JARVIS también puede detectar la resistencia del objeto, y puede detectar hasta 10 objetos de manera simultánea.

JARVIS puede calcular la distancia a la que se encuentra cada uno de los objetos, para esto siempre considera que la armadura se encuentra situada en la coordenada (0,0,0).

Hacer un método que informen a qué distancia se encuentra cada uno de los enemigos. Usar la clase Math de Java.

Simulador

Hacer un método en JARVIS que agregue en radar objetos, hacer que la resistencia, las coordenadas y la hostilidad sean aleatorios utilizando la clase random. Utilizar la clase Random.

¿Qué ocurre si quiero añadir más de 10 objetos?
¿Qué ocurre si cuando llevo 8 enemigos aumento la capacidad del vector?

Destruyendo Enemigos

Desarrollar un método para que JARVIS que analice todos los objetos del radar y si son hostiles que les dispare. El alcance de los guantes es de 5000 metros, si el objeto se encuentra fuera de ese rango no dispara.

JARVIS al detectar un enemigo lo atacará hasta destruirlo, la potencia del disparo es inversamente proporcional a la distancia al a que se encuentra el enemigo y se descontará de la resistencia del enemigo. El enemigo se considera destruido si su resistencia es menor o igual a cero.

JARVIS solo podrá disparar si el dispositivo está sano y si su nivel de batería lo permite. Si tiene los dos guantes sanos podrá disparar con ambos guantes haciendo más daño. Resolver utilizando un for each para recorrer el arreglo y un while para destruir al enemigo.

Acciones Evasivas

Desarrollamos un método para que JARVIS que analice todos los objetos del radar y si son hostiles que les dispare. Modificar ese método para que si el nivel de batería es menor al 10% se corten los ataques y se vuelve lo suficientemente lejos para que el enemigo no nos ataque. Deberíamos alejarnos por lo menos 10 km enemigo. Tener en cuenta que la velocidad de vuelo promedio es de 300 km / hora.

Bibliografía

Información sacada de las paginas:

- https://www.oracle.com/ar/database/what-is-a-relational-database/

- https://www.geeksforgeeks.org/sql-tutorial/