EGG

Programación Orientada a Objetos

Paradigma Orientado a Objetos

2020

Guía de Paradigma Orientado a Objetos

Clases y Objetos

Una clase es un molde para crear múltiples objetos que encapsula datos y comportamiento. Una clase es una combinación específica de atributos y métodos y puede considerarse un tipo de dato de cualquier tipo no primitivo. Así, una clase es una especie de plantilla o prototipo de objetos: define los atributos que componen ese tipo de objetos y los métodos que pueden emplearse para trabajar con esos objetos. En su forma más simple, una clase se define por la palabra reservada *class* seguida del nombre de la clase. El nombre de la clase debe empezar por mayúscula. Si el nombre es compuesto, entonces cada palabra debe empezar por mayúscula. La definición de la clase se pone entre las llaves de apertura y cierre.

```
public class NombreClase {
      // miembros dato o atributos
      // constructores
      // funciones miembro
}
```

Una vez que se ha declarado una clase, se pueden crear objetos a partir de ella. A la creación de un objeto se le denomina instanciación. Es por esto que se dice que un objeto es una instancia de una clase y el término instancia y objeto se utilizan indistintamente. Para crear objetos, basta con declarar una variable de alguno de los tipos de las clases definidas.

```
NombreClase nombreObjeto;
```

Para crear el objeto y asignar un espacio de memoria es necesario realizar la instanciación con el operador *new*. El operador *new* reserva espacio en memoria para los miembros dato y devuelve una referencia que se guarda en la variable.

```
nombreObjeto = new nombreClase();
```

Tanto la declaración de un objeto como la asignación del espacio de memoria se pueden realizar en un solo paso:

```
NombreClase nombreObjeto = new NombreClase();
```

A partir de este momento los objetos ya pueden ser referenciados por su nombre.

Acceso a los miembros

Desde un objeto se puede acceder a los miembros mediante la siguiente sintaxis:

```
nombreObjeto.miembro
```

Estado y Comportamiento

En términos más generales, un *objeto* es una abstracción conceptual del mundo real que se puede traducir a un lenguaje de programación orientado a objetos. Los objetos del mundo real comparten dos características: Todos poseen estado y comportamiento. Por ejemplo, el perro tiene estado (color, nombre, raza, edad) y el comportamiento (ladrar, caminar, comer, acostarse, mover la cola). Por lo tanto, un estado permite informar cuáles son las características del objeto y lo que éste representa, y el comportamiento, consiste en decir lo que sabe hacer.

El *estado* de un objeto es una lista de variables conocidas como sus *atributos*, cuyos valores representan el estado que caracteriza al objeto.

El *comportamiento* es una lista de métodos, procedimientos, funciones u operaciones que un objeto puede ejecutar a solicitud de otros objetos. Los objetos también se conocen como instancias.

Elementos de una Clase

Una clase describe un tipo de objetos con características comunes. Es necesario definir la información que almacena el objeto y su comportamiento.

Atributos

La información de un objeto se almacena en atributos. Los atributos pueden ser de tipos primitivos de Java (descriptos en el Apéndice A) o del tipo de otros objetos. La declaración de un atributo de un objeto tiene la siguiente forma:

```
<modificador>* <tipo> <nombre> [ = <valor inicial> ];
```

- <nombre>: puede ser cualquier identificador válido y denomina el atributo que está siendo declarado.
- <modificador>: si bien hay varios valores posibles para el <modificador>, por el momento solo usaremos modificadores de visibilidad: public, protected, private.
- <tipo>: indica la clase a la que pertenece el atributo definido.
- <valor inicial>: esta sentencia es opcional y se usa para inicializar el atributo del objeto con un valor particular.

Constructores

Además de definir los atributos de un objeto, es necesario definir los métodos que determinan su comportamiento. Toda clase debe definir un método especial denominado constructor para instanciar los objetos de la clase. Este método tiene el *mismo nombre de la clase*. La declaración básica toma la siguiente forma:

```
[<modificador>] <nombre de clase> ( <argumento>* ) {
   <sentencia>*
}
```

- <nombre de clase>: El nombre del constructor debe ser siempre el mismo que el de la clase.
- <modificador>: Actualmente, los únicos modificadores válidos para los constructores son public, protected y private.
- <argumentos>: es una lista de parámetros que tiene la misma función que en los métodos.

El método constructor se ejecuta cada vez que se instancia un objeto de la clase. Este método se utiliza para inicializar los atributos del objeto que se instancia.

Para diferenciar entre los atributos del objeto y los identificadores de los parámetros del método constructor, se utiliza la palabra *this*. De esta forma, los parámetros del método pueden tener el mismo nombre que los atributos de la clase.

La instanciación de un objeto consiste en asignar un espacio de memoria al que se hace referencia con el nombre del objeto. Los identificadores de los objetos permiten acceder a los valores almacenados en cada objeto.

El Constructor por Defecto

Cada clase tiene al menos un constructor. Si no se escribe un constructor, el lenguaje de programación Java le provee uno por defecto. Este constructor no posee argumentos y tiene un cuerpo vacío. Si se define un constructor que no sea vacío, el constructor vacío se pierde, salvo que explícitamente lo definamos.

Métodos

Los métodos son funciones que determinan el comportamiento de los objetos. Un objeto se comporta de una u otra forma dependiendo de los métodos de la clase a la que pertenece. Todos los objetos de una misma clase tienen los mismos métodos y el mismo comportamiento. Para definir los métodos, el lenguaje de programación Java toma la siguiente forma básica:

```
<modificador>* <tipo de retorno> <nombre> ( <argumento>>* ) {
    <sentencias>*
    return valorRetorno;
}
```

- <nombre>: puede ser cualquier identificador válido, con algunas restricciones basadas en los nombres que ya están en uso.
- <modificador>: el segmento es opcional y puede contener varios modificadores diferentes incluyendo a public, protected y private. Aunque no está limitado a estos.
- <tipo de retorno>: el tipo de retorno indica el tipo de valor devuelto por el método. Si el método *no devuelve un valor*, *debe ser declarado void*. La tecnología Java es rigurosa acerca

de los valores de retorno. Si el tipo de retorno en la declaración del método es un int, por ejemplo, el método debe devolver un valor int desde todos los posibles caminos de retorno (y puede ser invocado solamente en contextos que esperan un int para ser devuelto). Se usa la sentencia *return* dentro de un método para devolver un valor.

- <argumento>: permite que los valores de los argumentos sean pasados hacia el método. Los elementos de la lista están separados por comas y cada elemento consiste en un tipo y un identificador.

Existen tres tipos de métodos: *métodos de consulta, métodos modificadores y operaciones*. Los métodos de consulta sirven para extraer información de los objetos, los métodos modificadores sirven para modificar el valor de los atributos del objeto y las operaciones definen el comportamiento de un objeto.

Para acceder a los atributos de un objeto se definen los métodos *get* y *set*. Los métodos *get* se utilizan para consultar el estado de un objeto y los métodos *set* para modificar su estado. Un método *get* se declara *public* y a continuación se indica el tipo de dato que devuelve. Es un *método de consulta*. La lista de parámetros de un método *get* queda vacía. En el cuerpo del método se utiliza *return* para devolver el valor correspondiente al atributo que se quiere devolver, y al cual se hace referencia como *this.nombreAtributo*.

Por otra parte, un método set se declara *public* y devuelve *void*. La lista de parámetros de un método set incluye el tipo y el valor a modificar. Es un *método modificador*. El cuerpo de un método set asigna al atributo del objeto el parámetro de la declaración.

Por último, un *método de tipo operación* es aquel que realiza un cálculo o modifica el estado de un objeto. Este tipo de métodos pueden incluir una lista de parámetros y puede devolver un valor o no. Si el método no devuelve un valor, se declara *void*.

Invocación de métodos

Un método se puede invocar dentro o fuera de la clase donde se ha declarado. Si el método se invoca dentro de la clase, basta con indicar su nombre. Si el método se invoca fuera de la clase entonces se debe indicar el nombre del objeto y el nombre del método. Cuando se invoca a un método ocurre lo siguiente:

- En la línea de código del programa donde se invoca al método se calculan los valores de los argumentos.
- Los parámetros se inicializan con los valores de los argumentos.
- Se ejecuta el bloque código del método hasta que se alcanza return o se llega al final del bloque.
- Si el método devuelve un valor, se sustituye la invocación por el valor devuelto.
- La ejecución del programa continúa en la siguiente instrucción donde se invocó el método.

Parámetros y argumentos

Los parámetros de un método definen la cantidad y el tipo de dato de los valores que recibe un método para su ejecución. Los argumentos son los valores que se pasan a un método durante su invocación. El método recibe los argumentos correspondientes a los parámetros con los que ha sido declarado. Un método puede tener tantos parámetros como sea necesario. La lista de parámetros de la cabecera de un método se define con la siguiente sintaxis:

```
tipo nombre [,tipo nombre ]
```

Durante la invocación de un método es necesario que el número y el tipo de argumentos coincidan con el número y el tipo de parámetros declarados en la cabecera del método. Durante el proceso de compilación se comprueba que durante la invocación de un método se pasan tantos argumentos como parámetros tiene declarados y que además coinciden los tipos. Esta es una característica de los lenguajes que se denominan "*strongly typed*" o "*fuertemente tipados*".

Paso de parámetros

Cuando se invoca un método se hace una copia de los valores de los argumentos en los parámetros. Esto quiere decir que, si el método modifica el valor de un parámetro, nunca se modifica el valor original del argumento.

Cuando se pasa una referencia a un objeto se crea un nuevo alias sobre el objeto, de manera que esta nueva referencia utiliza el mismo espacio de memoria del objeto original y esto permite acceder al objeto original.

El valor de retorno

Un método puede devolver un valor. Los métodos que no devuelven un valor se declaran void, mientras que los métodos que devuelven un valor indican el tipo que devuelven: int, double, char, String o un tipo de objeto.

Sobrecarga de métodos

La sobrecarga de métodos es útil para que el mismo método opere con parámetros de distinto tipo o que un mismo método reciba una lista de parámetros diferente. Esto quiere decir que puede haber dos métodos con el mismo nombre que realicen dos funciones distintas. La diferencia entre los métodos sobrecargados está en su declaración, y más específicamente, en la cantidad y tipos de datos que reciben.

Abstracción y Encapsulamiento

La *abstracción* es la habilidad de ignorar los detalles de las partes para enfocar la atención en un nivel más alto de un problema. El *encapsulamiento* sucede cuando algo es envuelto en una capa protectora. Cuando el encapsulamiento se aplica a los objetos, significa que los datos del objeto

están protegidos, "ocultos" dentro del objeto. Con los datos ocultos, ¿cómo puede el resto del programa acceder a ellos? (El acceso a los datos de un objeto se refiere a leerlos o modificarlos.) El resto del programa no puede acceder de manera directa a los datos de un objeto; lo tiene que hacer con ayuda de los métodos del objeto. Al hecho de proteger los datos o atributos con los métodos se denomina *encapsulamiento*.

Abstracción

La abstracción es la propiedad que considera los aspectos más significativos o notables de un problema y expresa una solución en esos términos. La abstracción posee diversos grados o niveles de abstracción, los cuales ayudan a estructurar la complejidad intrínseca que poseen los sistemas del mundo real. La abstracción encarada desde el punto de vista de la programación orientada a objetos es el mecanismo por el cual se proveen los límites conceptuales de los objetos y se expresan sus características esenciales, dejando de lado sus características no esenciales. Si un objeto tiene más características de las necesarias los mismos resultan difíciles de usar, modificar, construir y comprender. En el análisis hay que concentrarse en ¿Qué hace? y no en ¿Cómo lo hace?

Encapsulamiento

La encapsulación o encapsulamiento significa reunir en una cierta estructura a todos los elementos que a un cierto nivel de abstracción se pueden considerar pertenecientes a una misma entidad, y es el proceso de agrupamiento de datos y operaciones relacionadas bajo una misma unidad de programación, lo que permite aumentar la cohesión de los componentes del sistema. El encapsulamiento oculta lo que hace un objeto de lo que hacen otros objetos y del mundo exterior por lo que se denomina también ocultación de datos. Un objeto tiene que presentar "una cara" al mundo exterior de modo que se puedan iniciar sus operaciones.

Los métodos operan sobre el estado interno de un objeto y sirven como el mecanismo primario de comunicación entre objetos. Ocultar el estado interno y hacer que toda interacción sea a través de los métodos del objeto es un mecanismo conocido como encapsulación de datos.

Modificadores de Acceso

Todas las clases poseen diferentes niveles de acceso en función del modificador de acceso (visibilidad). A continuación, se detallan los niveles de acceso con sus símbolos correspondientes:

- ***Public***: Este modificador permite a acceder a los elementos desde cualquier clase, independientemente de que esta pertenezca o no al paquete en que se encuentra el elemento.
- ***Private***: Es el modificador más restrictivo y especifica que los elementos que lo utilizan sólo pueden ser accedidos desde la clase en la que se encuentran. Este modificador *sólo puede utilizarse sobre los miembros de una clase* y *sobre interfaces y clases internas*, no sobre clases o interfaces de primer nivel, dado que esto no tendría sentido. Es importante destacar

también que el modificador private convierte los elementos en privados para otras clases, no para otras instancias de la clase. Es decir, un objeto de una determinada clase puede acceder a los miembros privados de otro objeto de la misma clase.

- ***Protected***: Este modificador indica que los elementos sólo pueden ser accedidos desde su mismo paquete y desde cualquier clase que extienda la clase en que se encuentra, independientemente de si esta se encuentra en el mismo paquete o no. Este modificador, como private, no tiene sentido a nivel de clases o interfaces no internas.

Si no especificamos ningún modificador de acceso se utiliza el nivel de acceso por defecto (Default), que consiste en que el elemento puede ser accedido sólo desde las clases que pertenezcan al mismo paquete. Los distintos modificadores de acceso quedan resumidos en la siguiente tabla:

Visibilidad	Public	Private	Protected	Default
Desde la misma Clase	SI	SI	SI	SI
Desde cualquier Clase del mismo Paquete	SI	SI	SI	SI
Desde una Subclase del mismo Paquete	SI	SI	SI	SI
Desde una Subclase fuera del mismo Paquete	SI	NO	SI, a través de la herencia	NO
Desde cualquier Clase fuera del Paquete	SI	NO	NO	NO

Atributos y Métodos Estáticos

Un atributo o un método de una clase se puede modificar con la palabra reservada *static* para indicar que este atributo o método no pertenece a las instancias de la clase si no a la propia clase. Se dice que son atributos de clase si se usa la palabra clave *static*: en ese caso la variable es única para todas las instancias (objetos) de la clase (ocupa un único lugar en memoria), es decir que, si se poseen múltiples instancias de una clase, cada una de ellas no tendrán una copia propia de este atributo, si no que todas estas instancias compartirán una misma copia del atributo. A veces a las variables de clase se les llama variables estáticas. Si no se usa static, el sistema crea un lugar nuevo para esa variable con cada instancia (la variable es diferente para cada objeto).
En el caso de una constante no tiene sentido crear un nuevo lugar de memoria por cada objeto de una clase que se cree. Por ello es adecuado el uso de la palabra clave static. Cuando usamos "*static final*" se dice que creamos una constante de clase, un atributo común a todos los objetos de esa clase.

Atributos Finales

En este contexto indica que una variable es de tipo constante: no admitirá cambios después de su declaración y asignación de valor. La palabra reservada *final* determina que un atributo no puede ser sobrescrito o redefinido, es decir, no funcionará como una variable "tradicional", sino como una constante. Toda constante declarada con final ha de ser inicializada en el mismo

momento de declararla. El modificador *final* también se usa como palabra clave en otro contexto: una *clase final* es aquella que no puede tener clases que la hereden. Lo veremos más adelante cuando hablemos sobre herencia.
Cuando se declaran constantes es muy frecuente que los programadores usen letras mayúsculas (como práctica habitual que permite una mayor claridad en el código), aunque no es obligatorio.

Colecciones

Una colección representa un grupo de objetos. Esto objetos son conocidos como elementos. Cuando queremos trabajar con un conjunto de elementos, necesitamos un almacén donde poder guardarlos. En las colecciones se puede almacenar cualquier tipo de objeto y se puede utilizar una serie de métodos comunes, como por ejemplo: agregar, eliminar, obtener el tamaño de la colección. Las colecciones son una especie de arrays de tamaño dinámico. Java Collections Framework es el marco de trabajo que permite trabajar con objetos contenedores de objetos tales como listas, conjuntos y mapas.

Las operaciones básicas de una collection son:

- *add(T):* Añade un elemento.
- *iterator():* Obtiene un "iterador" que permite recorrer la colección visitando cada elemento.
- *size():* Obtiene la cantidad de elementos que esta colección almacena.
- *contains(t):* Pregunta si el elemento t ya está dentro de la colección.

Listas

Las listas modelan una colección de objetos ordenados por posición. La principal diferencia entre las listas y los arreglos tradicionales, es que la lista crece de manera *dinámica* a medida que se van agregando objetos. No necesitamos saber de antemano la cantidad de elementos con la que vamos a trabajar. El framework trae varias implementaciones de distintos tipos de listas tales como ArrayList, LinkedList y Vector.

- ArrayList: se implementa como un arreglo de tamaño variable. A medida que se agregan más elementos, su tamaño aumenta dinámicamente.
- LinkedList: se implementa como una lista de doble enlace. Su rendimiento al agregar y quitar es mejor que Arraylist, pero peor en los métodos *set* y *get*.
- Vector: es similar a un ArrayList, pero está sincronizado.

Conjuntos

Los conjuntos modelan una colección de objetos de una misma clase donde cada elemento aparece sólo una vez, a diferencia de una lista donde los elementos podían repetirse. El framework trae varias implementaciones de distintos tipos de conjuntos:

- HashSet, se implementa utilizando una tabla hash. Los elementos no están ordenados. Los métodos de agregar, eliminar y contiene tienen una complejidad de tiempo constante por lo que tienen mejor performance que el Treeset.
- TreeSet se implementa utilizando una estructura de árbol (árbol rojo-negro en el libro de algoritmos). Los elementos de un conjunto están ordenados, pero los métodos de agregar, eliminar y contiene tienen una complejidad peor a la de HashSet. Ofrece varios métodos para tratar con el conjunto ordenado como primero (), último (), headSet (), tailSet (), etc.
- LinkedHashSet está entre HashSet y TreeSet. Se implementa como una tabla hash con una lista vinculada que se ejecuta a través de ella, por lo que proporciona el orden de inserción.

Mapas

Los mapas son una de las estructuras de datos importantes del Framework de Collections. Las implementaciones de mapas son HashMap, TreeMAp, LinkedHashMap y HashTable.

- HashMap es un mapa implementado a través de una tabla hash, las llaves se almacenan utilizando un algoritmo de hash.
- TreeMap es un mapa implementado a través de un árbol, es decir que las llaves se almacenan en una estructura de árbol, y por lo tanto, se almacenan ordenadas.
- LinkedHashMap es un HashMap que conserva el orden de inserción.
- Hashtable está sincronizado, en contraste con HashMap. Tiene una sobrecarga para la sincronización.

Relaciones entre Clases

Las relaciones existentes entre las distintas clases indican como se están comunicando las clases entre sí. Dentro de las relaciones entre clases que existen vamos a definir las siguientes:

Uso o Asociación

Representa un tipo de relación muy particular, en la que una clase es instanciada por otro objeto y clase. La asociación se podría definir como el momento en que dos objetos se unen para trabajar juntos y así, alcanzar una meta. Un punto a tomar muy en cuenta es que ambos objetos son independientes entre sí. Para validar la asociación, la frase “Usa un”, debe tener sentido, por ejemplo: El programador usa una computadora. El objeto computadora va a existir más allá de que exista o no el objeto programador.

Composición

La composición es una relación más fuerte que la asociación, es una "relación de vida", es decir, que el tiempo de vida de un objeto está condicionado por el tiempo de vida del objeto que lo incluye. La composición es un tipo de relación dependiente en dónde un objeto más complejo es conformado por objetos más pequeños. En esta situación, la frase “*Tiene un*”, debe tener sentido,

por ejemplo: el cliente tiene una cuenta bancaria. Esta relación es una composición, debido a que al eliminar el cliente la cuenta bancaria no tiene sentido y también se debe eliminar, es decir, la cuenta existe sólo mientras exista el cliente.

Herencia

La herencia es una relación fuerte entre dos clases donde una clase es padre de otra. Las propiedades comunes se definen en la superclase (clase padre) y las subclases heredan estas propiedades (Clase hija). En esta relación, la frase "Un objeto *es un-tipo-de* una superclase" debe tener sentido, por ejemplo: un perro es un tipo de animal, o también, una heladera es un tipo de electrodoméstico.

Herencia y Constructores

Una diferencia entre los constructores y los métodos es que los constructores no se heredan, pero los métodos sí. Todos los constructores definidos en una superclase pueden ser usados desde constructores de las subclases a través de la palabra clave *super*. La palabra clave *super* es la que me permite elegir qué constructor entre los que tiene definida la clase padre es el que debo usar. Si la superclase tiene definido el constructor vacío y no colocamos una llamada explícita super se llamará el constructor vacío de la superclase.

Herencia y Métodos

Todos los métodos accesibles o visibles de una superclase se heredan a sus subclases. ¿Qué ocurre si una subclase necesita que uno de sus métodos heredados funcione de manera diferente? Los métodos heredados pueden ser redefinidos en las clases hijas. Este mecanismo se lo llama *sobreescritura*. La *sobreescritura* permite que las clases hijas sumen sus particulares en torno al funcionamiento y agrega coherencia al modelo. Lo que se espera es que a medida que desciendo por la jerarquía de herencia me encuentre con clases más específicas.

Polimorfismo

El término *polimorfismo* es una palabra de origen griego que significa "muchas formas". Este término se utiliza en la POO para referirse a la propiedad por la que es posible enviar mensajes sintácticamente iguales a objetos de tipos distintos, es decir, que la misma operación se realiza en las clases de diferente forma. Estas operaciones tienen el mismo significado y comportamiento, pero internamente, cada operación se realiza de diferente forma. El único requisito que deben cumplir los objetos que se utilizan de manera polimórfica es saber responder al mensaje que se les envía.

Modificadores de Clases y Métodos

Clases Finales

El modificador final puede utilizarse también como modificador de clases. Al marcar una clase como final impedimos que se generen hijos a partir de esta clase, es decir, cortamos la jerarquía de herencia.

Métodos Finales

El modificador final puede utilizarse también como modificador de métodos. La herencia nos permite reutilizar el código existente y uno de los mecanismos es la crear una subclase y sobrescribir alguno de los métodos de la clase padre. Cuando un método es marcado como final en una clase, evitamos que sus clases hijas puedan sobrescribir estos métodos.

Métodos Abstractos

Un método abstracto es un método declarado, pero no implementado, es decir, es un método del que solo se escribe su nombre, parámetros, y tipo devuelto, pero no su código de implementación. Estos métodos se heredan por las clases hijas y éstas son las responsables de implementar sus funcionalidades.

Clases Abstractas

En java se dice que una clase es abstracta cuando no se permiten instancias de esa clase, es decir que no se pueden crear objetos. Cuando una clase posee al menos un método abstracto esa clase necesariamente debe ser marcada como abstracta. Esto indica que no pueden existir instancias de esa clase debido a que poseen por lo menos un método sin implementar. Los hijos de una clase abstracta deben ser los encargados de sobrescribir los métodos abstractos e implementar las funcionalidades.

Interfaces

Una interfaz en Java es una clase que contiene todos sus métodos abstractos y sus atributos son sólo constantes. En las interfaces se especifica qué se debe hacer, pero no su implementación. Las clases que implementen una interface serán las responsables de describir la lógica del comportamiento de los métodos. La principal diferencia entre *interface* y *abstract* es que una interface proporciona un mecanismo de encapsulación de los métodos sin forzar a la utilización de una herencia.

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
- Problemas de Hardware

Manejador de excepciones

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 objeto que modela el error que ha ocurrido. Puede lanzarse cualquier objeto que implemente la interfaz *Throwable*. Existen dos implementaciones de esta interfaz (*Exception* y *Error*) en el API de Java.

- Error: Modela cualquier error del sistema grave. Estos errores no deberían ser capturados.
- Exception: Modela un error el cual el programador desea manejar. Nosotros vamos a trabajar con subclases de Exception.

En primer lugar, es necesario declarar todas las posibles excepciones que es posible generar en el método, utilizando la cláusula *throws* de la declaración de métodos. Para lanzar la excepción es necesario crear un objeto de tipo *Exception* o alguna de sus subclases (por ejemplo: *ArithmeticException*) y lanzarlo mediante la instrucción *throw*.

En Java se pueden tratar las excepciones previstas por el programador utilizando unos mecanismos, los manejadores de excepciones, que se estructuran en tres bloques:

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.

```
try {
	BloqueDeIntrucciones
}
```

Cualquier excepción que se produzca 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.

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) {
       BloqueCatch
} catch (TipoExcepción nombreVariable) {
       BloqueCatch
} …
} catch (TipoExcepción nombreVariable) {
       BloqueCatch
}
finally {
       BloqueFinally
}
```

La claúsula throws

La cláusula *throws* lista las excepciones que un método puede lanzar. Los tipos de excepciones que lanza el método se especifica en la 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. Desde luego, colocar una excepción verificada en la cláusula *throws* obliga a otros métodos a ocuparse también de la excepción verificada.

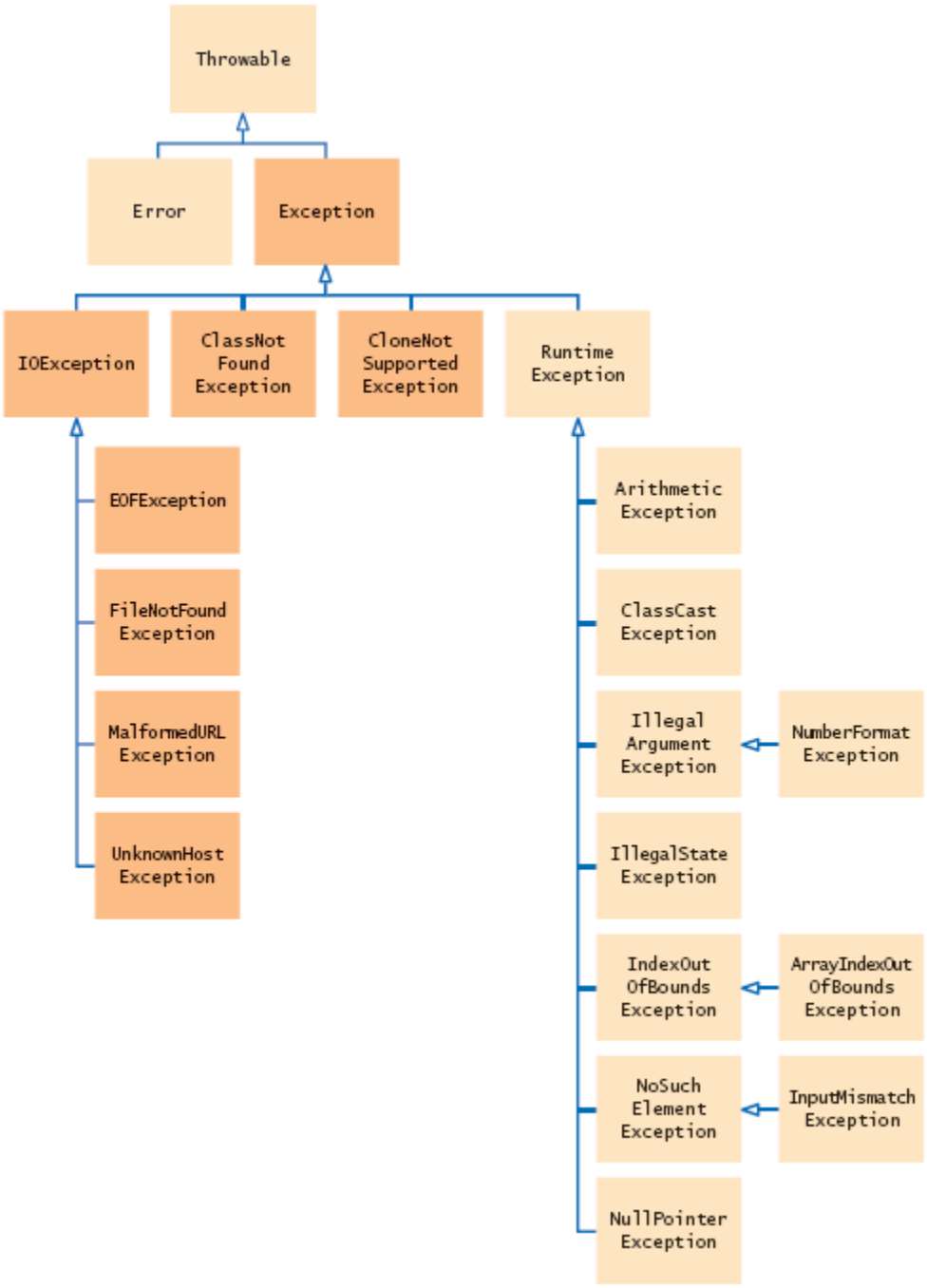
Si la excepción no se trata, el manejador de excepciones realiza lo siguiente:

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

Jerarquía de Excepciones

Todas las excepciones lanzadas automáticamente en un programa Java son objetos de la clase Throwable o de alguna de sus clases derivadas. La clase Throwable deriva directamente de Object y tiene dos clases derivadas directas: Error y Exception.

Resumen de la jerarquía de clases derivadas de Throwable.



Preguntas de Aprendizaje

1) Responda Verdadero (V) o Falso (F)

	V	F
Java es un lenguaje de tipado estático, por lo tanto, todas las variables deben ser declaradas antes de ser utilizadas.	( )	( )
El acceso a una variable local no inicializada no da error	( )	( )
Cuando se coloca la palabra final precediendo la declaración de una variable la misma se transforma en una constante	( )	( )
Una variable local no puede ser declarada en cualquier lugar del cuerpo de una clase o método	( )	( )
Tanto las operaciones de lectura como las de escritura son de la clase java.net	( )	( )
El método constructor de una clase puede tener cualquier nombre	( )	( )
Cuando un método no devuelve ningún valor se utiliza la palabra reservada void para indicar que no devuelve nada	( )	( )
La palabra reservada final determina que un atributo no puede ser redefinido	( )	( )
La devolución de un valor a través de un método se hace con la sentencia restore	( )	( )
El bloque try se utiliza para tratar algún tipo de excepción	( )	( )

2) Un lenguaje fuertemente tipado:
   a) Es un lenguaje en el que las variables pueden cambiar de tipos de datos si se especifica previamente.
   b) No es necesario declarar todas las variables siempre y cuando pasen como parámetros a los métodos.
   c) Deben declararse todas las variables que se utilicen.
   d) Ninguna de las anteriores.

3) El llamado de una librería en Java se hace usando el:
   a) Import
   b) Scanner
   c) String
   d) Ninguna de las anteriores

4) ¿Cuál es la descripción que crees que define mejor el concepto *clase* en la programación orientada a objetos?
   a) Es un concepto similar al de array
   b) Es un tipo particular de variable
   c) Es un modelo o plantilla a partir de la cual creamos objetos
   d) Es una categoría de datos ordenada secuencialmente

5) ¿Qué elementos crees que definen a un objeto?
   a) Su cardinalidad y su tipo
   b) Sus atributos y sus métodos
   c) La forma en que establece comunicación e intercambia mensajes
   d) Su interfaz y los eventos asociados

6) ¿Qué significa instanciar una clase?
   a) Duplicar una clase
   b) Eliminar una clase
   c) Crear un objeto a partir de la clase
   d) Conectar dos clases entre sí

7) Queremos crear una clase Java con variables miembro que puedan ser accedidas, ¿qué opción elegirías como la mejor?
   a) Variables miembro públicas
   b) Variables miembro static
   c) Variables miembro privadas con getters y setters
   d) Ninguna de las anteriores

8) ¿Qué permite agrupar la palabra package?
   a) Clases e interfaces
   b) Agrupar códigos
   c) Agrupar signos
   d) Agrupar secuencias

9) Los nombres de los paquetes son:
   a) Complejos
   b) Para agrupar clases y evitar conflictos con los nombres entre las clases importadas como también para ayudar en el control de la accesibilidad de clases y miembros.
   c) Para almacenar directorios
   d) Ninguna de las anteriores

10) ¿Cuáles son las dos formas para utilizar import?
   a) Para una clase y para todo un package
   b) Para dos clases de package
   c) Para Java.io
   d) Para Java.net

11) Se crean anteponiendo la palabra static a su declaración:
   a) Variables miembro de objeto
   b) Variables miembro de clase
   c) Variables finales
   d) Ninguna de las anteriores

12) No puede cambiar su valor durante la ejecución del programa:
   a) Variables miembro de objeto
   b) Variables miembro de clase
   c) Variables finales
   d) Todas las anteriores

13) ¿Qué es Netbeans?
   a) Una librería de Java
   b) Una versión de Java especial para servidores
   c) Un IDE para desarrollar aplicaciones
   d) Ninguna de las anteriores

14) ¿Qué es una excepción?
   a) Un error que lanza un método cuando algo va mal
   b) Un objeto que no puede ser instanciado
   c) Un bucle que no finaliza
   d) Un tipo de evento muy utilizado al crear interfaces

15) En el framework de colecciones de Java un Set es:
   a) Una colección que no puede contener elementos duplicados
   b) Una colección ordenada que puede contener elementos duplicados
   c) Un objeto que mapea conjuntos de clave valor y no puede contener valores duplicados
   d) Ninguna de las anteriores

16) En Java un Iterator es:
   a) Una interface que proporciona los métodos para borrar elementos de una colección
   b) Una interface que proporciona los métodos para recorrer los elementos de una colección y posibilita el borrado de elementos
   c) Una interface que proporciona los métodos para ordenar los elementos de la colección.
   d) Ninguna de las anteriores

17) ¿Qué código de los siguientes tiene que ver con la herencia?
   a) public class Componente extends Producto
   b) public class Componente inherit Producto
   c) public class Componente implements Producto
   d) public class Componente belong to Producto

18) ¿Cuál tipo de clase debe tener al menos un método abstracto?
   a) final
   b) abstract
   c) public
   d) Todas las anteriores

19) ¿De cuántas clases se puede derivar Java?
   a) Tres clases
   b) Dos clases
   c) Una clase
   d) Cinco clases

20) Una clase que termina la cadena de una herencia de la puede declarar como:
   a) final
   b) abstract
   c) public
   d) Ninguna de las anteriores

21) ¿Qué código asociarías a una Interfaz en Java?
   a) public class Componente interface Product
   b) Componente cp = new Componente (interfaz)
   c) public class Componente implements Printable
   d) Componente cp = new Componente.interfaz

22) ¿Qué significa sobrecargar (overload) un método?
   a) Editarlo para modificar su comportamiento
   b) Cambiarle el nombre dejándolo con la misma funcionalidad
   c) Crear un método con el mismo nombre, pero diferentes argumentos
   d) Añadirle funcionalidades a un método

23) 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

24) Responda las siguientes preguntas:
   a) ¿Un perro, es un objeto o una clase? Justique.
   b) Defina qué es un objeto. De ejemplos concretos
   c) Defina qué es una clase. De ejemplos concretos.
   d) ¿Cuál es la diferencia entre un objeto y una clase?
   e) ¿Cuál es la diferencia entre mensaje y método?

25) Dado el siguiente código y asumiendo que la clase Ordenador ya fue definida:

```
1 Ordenador escritorio; Ordenador portatil;
2 escritorio = new Ordenador();
3 escritorio.precio(900);
4 portatil = new Ordenador();
5 portatil.precio(1100);
6 portatil = escritorio;
7 escritorio = null;
```

Responda ¿qué afirmación es cierta? Justifique su respuesta

a) Cuando se ejecuta la línea 5, la instancia escritorio cuesta 1100.
b) Cuando se ejecuta la línea 5, la instancia portatil cuesta 1100.
c) Al final tanto el objeto escritorio como el objeto portatil apuntan a null.
d) Al final sólo queda un objeto de tipo Ordenador con precio 1100.
e) Al final sólo queda un objeto de tipo Ordenador con precio 900.
f) Al final hay dos objetos de tipo Ordenador, uno con precio 900 y otro con precio 1100.

26) Indique qué valores retornan las siguientes expresiones, o si dan error, por qué se producen:

```
s.length();
t.length(); 1 + a;
a.toUpperCase();
"Libertad".indexOf("r");
"Universidad".lastIndexOf('i');
"Quilmes".substring(2,4);
(a.length() + a).startsWith("a");
s == a;
a.substring(1,3).equals("bc");
```

Ejercicios de Aprendizaje

Antes de comenzar con esta guía, les damos algunas recomendaciones:

Este módulo es uno de los más divertidos ya que vamos a comenzar a modelar los objetos del mundo real con el lenguaje de programación Java. Es importante tener en cuenta que entender la programación orientada a objetos lleva tiempo y sobre todo PRÁCTICA, así que, a no desesperarse, con cada ejercicio vamos a ir entendiendo un poco más cómo aplicar este paradigma.

A partir de esta guía todos los ejemplos de los videos serán desarrollados siguiendo el ejemplo de un Tinder de Mascotas. A continuación se presenta el contexto del problema para entender los objetos del mundo real que vamos a modelar.

VER VIDEOS:

A. Introducción
B. Contexto del Tinder de Mascotas

Realizar los siguientes ejercicios:

VER VIDEOS:

A. Introducción a Objetos
B. Método Constructor
C. Encapsulamiento Parte 1
D. Encapsulamiento Parte 2

1. Crear una clase llamada *Fraccion* que contenga métodos para sumar, restar, multiplicar y dividir fracciones. Los argumentos de cada método son el numerador y denominador según corresponda. La clase también debe contener un método constructor con parámetros.

2. Crear una clase llamada *Libro* que contenga los siguientes atributos: ISBN, Título, Autor, Número de páginas, y un constructor que inicialice esos atributos con los valores pasados como parámetros. Definir una instancia para cargar un libro y luego informar mediante otro método el número de ISBN, el título y el autor del libro.

3. Declarar una clase llamada *Circunferencia* que tenga como atributo privado el radio de tipo real.
   A continuación se deben crear los siguientes métodos:

   a) Método constructor que inicialice el radio pasado como parámetro.
   b) Métodos get y set para el atributo radio de la clase Circunferencia.
   c) Método para calcular el área de la circunferencia ($Area = \pi * radio^2$).
   d) Método para calcular el perímetro ($Perimetro = 2 * \pi * radio$).

4. Definir una clase llamada *Punto* con un constructor que inicialice las coordenadas x e y. Generar dos instancias, es decir, crear dos objetos llamados punto1 y punto2 y comprobar la distancia que existe entre ambos. Para conocer como calcular la distancia entre dos puntos consulte el siguiente link: http://www.matematicatuya.com/GRAFICAecuaciones/S1a.html.

5. Definir la clase *Tiempo*, la cual tendrá la hora, minutos y segundos. Definir dos constructores: un constructor vacío y otro con la hora, minutos y segundos ingresado por el usuario. Deberá definir además, los métodos getters y setters correspondientes, y el método imprimirHoraCompleta().

6. Desarrollar una clase *Cancion* con los siguientes atributos: título y autor. Se deberá definir además dos constructores: uno vacío que inicializa el título y el autor a cadenas vacías y otro que reciba como parámetros el título y el autor de la canción. Se deberán además definir los métodos getters y setters correspondientes.

7. Crear una clase *Rectángulo* que modele rectángulos por medio de cuatro puntos (los vértices). La clase dispondrá de dos constructores: uno que cree un rectángulo partiendo de sus cuatro vértices y otro que cree un rectángulo partiendo de la base y la altura, de forma que su vértice inferior izquierdo esté en (0,0). La clase también incluirá un método para calcular la superficie y otro que desplace el rectángulo en el plano. Se deberán además definir los métodos getters y setters correspondientes ($Superficie = base * altura$).

VER VIDEO: Clases Entidad y Control

8. Realizar una clase llamada *Cuenta* (bancaria) que debe tener como mínimo los atributos: numeroCuenta (entero), el DNI del cliente (entero largo), el saldo actual y el interés anual que se aplica a la cuenta (porcentaje). Las operaciones asociadas a dicha clase son:
   - Constructor por defecto y constructor con DNI, saldo, número de cuenta e interés.
   - Método actualizarSaldo(): actualizará el saldo de la cuenta aplicándole el interés diario (interés anual dividido entre 365 aplicado al saldo actual)
   - Método ingresar(double): permitirá ingresar una cantidad en la cuenta bancaria.
   - Método retirar(double): permitirá sacar una cantidad de la cuenta (si hay saldo).
   - Método consultarSaldo(): permitirá consultar el saldo disponible en la cuenta.
   - Método consultarDatos(): permitirá mostrar todos los datos de la cuenta.
   - Agregar los métodos getters y setters correspondientes

9. Vamos a realizar una clase llamada *Raices*, donde representaremos los valores de una ecuación de 2º grado. Tendremos los 3 coeficientes como atributos, llamémosles a, b y c. Hay que insertar estos 3 valores para construir el objeto a través de un método constructor. Luego, las operaciones que se podrán realizar son las siguientes:
   - Método obtenerRaices(): imprime las 2 posibles soluciones
   - Método obtenerRaiz(): imprime una única raíz. Es en el caso en que se tenga una única solución posible.

- Método getDiscriminante(): devuelve el valor del discriminante (double). El discriminante tiene la siguiente formula: (b^2)-4*a*c
- Método tieneRaices(): devuelve un booleano indicando si tiene dos soluciones, para que esto ocurra, el discriminante debe ser mayor o igual que 0.
- Método tieneRaiz(): devuelve un booleano indicando si tiene una única solución, para que esto ocurra, el discriminante debe ser igual que 0.
- Método calcular(): mostrará por pantalla las posibles soluciones que tiene nuestra ecuación, y en caso de no existir solución, se mostrará un mensaje.

*Nota: Formula ecuación 2º grado: (-b±√((b^2)-(4*a*c)))/(2*a)*
Solo varia el signo delante de -b

10. Crear una clase *Fecha* con atributos para el día, mes y año. Se debe incluir al menos los siguientes métodos:
    - Constructor predeterminado con el 1-1-1900 como fecha por defecto
    - Constructor parametrizado con día, mes y año ingresados por el usuario.
    - Método leer() para pedir al usuario el día (1 a 31), el mes (1 a 12) y el año (1900 a 2050).
    - Método bisiesto() para indicar si el año es bisiesto o no.
    - Método diasMes(int) para devolver el número de días del mes indicado (para el año de la fecha).
    - Método validar() para comprobar si la fecha es correcta (entre el 1-1-1900 y el 31-12-2050). Si el día no es correcto, se pondrá en 1; si el mes no es correcto se pondrá en 1; y si el año no es correcto lo pondrá en 1900. Esto último se realizará mediante los métodos setters de cada atributo. Los setters también se llamarán en el constructor parametrizado y en el método leer().
    - Método diasTranscurridos() para devolver la cantidad de días transcurridos desde el 1-1-1900 hasta la fecha ingresada por el usuario.
    - Método diasEntre(Fecha) para devolver el número de días entre la fecha ingresada por el usuario y la proporcionada como parámetro.
    - Método siguiente() para devolver el día siguiente del día de la fecha ingresada.
    - Método anterior() para devolver el día anterior del día de la fecha ingresada.

11. Programa *Nespresso*. Desarrolle una clase Cafetera con los atributos capacidadMaxima (la cantidad máxima de café que puede contener la cafetera) y cantidadActual (la cantidad actual de café que hay en la cafetera). Implemente, al menos, los siguientes métodos:
    - Constructor predeterminado: establece la capacidad máxima en 1000 (c.c.) y la actual en cero (cafetera vacía).
    - Constructor con la capacidad máxima de la cafetera; inicializa la cantidad actual de café igual a la capacidad máxima.
    - Constructor con la capacidad máxima y la cantidad actual. Si la cantidad actual es mayor que la capacidad máxima de la cafetera, la ajustará al máximo.
    - Métodos getters y setters.
    - Método llenarCafetera():hace que la cantidad actual sea igual a la capacidad.
    - Método servirTaza(int): simula la acción de servir una taza con la capacidad indicada. Si la cantidad actual de café “no alcanza” para llenar la taza, se sirve lo que quede.

- Método vaciarCafetera(): pone la cantidad de café actual en cero.
- Método agregarCafe(int): añade a la cafetera la cantidad de café indicada

12. *Dígito Verificador*. Crear una clase Letra que se usará para mantener DNIs con su correspondiente letra. Los atributos serán el número de DNI (entero largo) y la letra que le corresponde. La clase dispondrá de los siguientes métodos:
    - Constructor predeterminado que inicialice el nº de DNI a 0 y la letra a espacio en blanco (será un NIF no válido).
    - Constructor que reciba el DNI y establezca la letra que le corresponde.
    - Métodos getters y setters para el número de DNI (que ajuste también automáticamente la letra).
    - Método leer(): para pedir el número de DNI (ajustando automáticamente la letra)
    - Método que nos permita mostrar el NIF (ocho dígitos, un guión y la letra en mayúscula; por ejemplo: 00395469-F).

    La letra correspondiente al dígito verificador se calculará a través de un método que funciona de la siguiente manera: Para calcular la letra se toma el *resto* de dividir el número de DNI por 23 (el resultado debe ser un número entre 0 y 22). El método debe buscar en un array de caracteres la posición que corresponda al resto de la división para obtener la letra correspondiente. La tabla de caracteres es la siguiente:

POSICION	LETRA
0	T
1	R
2	W
3	A
4	G
5	M
6	Y
7	F
8	P
9	D
10	X
11	B
12	N
13	J
14	Z
15	S
16	Q
17	V
18	H
19	L
20	C
21	K
22	E

13. *Juego Ahorcado:* Crear una clase *Ahorcado* (como el juego), la cual deberá contener un vector con la palabra a buscar y la cantidad jugadas máximas que puede realizar el usuario. Definir los siguientes métodos con los parámetros que sean necesarios:
    - Un método para mostrar la longitud de la palabra que se debe encontrar.
    - Un método para buscar si una letra ingresada por el usuario es parte de la palabra o no.
    - Un método para informar error o acierto.
    - Un método para mostrar cuantas oportunidades le queda al jugador.
    - Un método que al pedir ingresar una letra muestre que letras han sido encontradas y en qué posición. Además, se debe mostrar también cuántas oportunidades quedan.
    - Un método para buscar si se encontró la palabra completa.
    - En el método main() se deberá pedir al usuario una letra hasta que el usuario haya gastado todas sus oportunidades o bien hasta que encuentre la palabra.

      Un ejemplo de salida puede ser así:

      Longitud de la palabra: 6
      Ingrese una letra:
      a
      Mensaje: La letra pertenece a la palabra
      Número de letras (encontradas,faltantes): (3,4)
      Número de oportunidades restantes: 3
      Estado Actual: a a a

      Ingrese una letra:
      z
      Mensaje: La letra no pertenece a la palabra
      Número de letras (encontradas,faltantes): (3,4)
      Número de oportunidades restantes: 2
      Estado Actual: a a a

      Ingrese una letra:
      b
      Mensaje: La letra no pertenece a la palabra
      Número de letras (encontradas,faltantes): (3,4)
      Número de oportunidades restantes: 2
      Estado Actual: a a b a

      Ingrese una letra:
      u
      Mensaje: La letra no pertenece a la palabra
      Número de letras (encontradas,faltantes): (3,4)
      Número de oportunidades restantes: 1
      Estado Actual: a a a

      Ingrese una letra:
      q

Mensaje: La letra no pertenece a la palabra
Mensaje: Lo sentimos, no hay más oportunidades

14. Realizar una clase llamada *Persona* que tenga los siguientes atributos: nombre, edad, sexo ('H' hombre, 'M' mujer, 'O' otro), peso y altura. Si el alumno desea añadir algún otro atributo, puede hacerlo.
Se implementarán tres constructores:
- Un constructor por defecto.
- Un constructor con el nombre, edad y sexo recibidos como parámetro; y el resto de los atributos se inicializarán con valores por defecto.
- Un constructor con todos los atributos como parámetro.

Los métodos que se implementarán son:
- Método calcularIMC(): calculara si la persona está en su peso ideal (peso en kg/(altura^2 en m)). Si esta fórmula da por resultado un valor menor que 20, la función devuelve un -1. Si la fórmula da por resultado un número entre 20 y 25 (incluidos), significa que el peso está por debajo de su peso ideal y la función devuelve un 0. Finalmente, si el resultado de la fórmula es un valor mayor que 25 significa que la persona tiene sobrepeso, y la función devuelve un 1. Se recomienda hacer uso de constantes para devolver estos valores.
- Método esMayorDeEdad(): indica si la persona es mayor de edad. La función devuelve un booleano.
- Método comprobarSexo(char sexo): comprueba que el sexo introducido sea correcto, es decir, H M ó O. Si no es correcto se deberá mostrar un mensaje.
- Métodos getters y setters de cada atributo.

A continuación, crear una clase ejecutable que haga lo siguiente:

Pedir por teclado el nombre, la edad, sexo, peso y altura. Luego se crearán 3 objetos de la clase Persona de la siguiente manera: el primer objeto obtendrá los valores pedidos por teclado, el segundo objeto obtendrá del usuario todos los atributos menos el peso y la altura, y el tercer objeto será inicializado con valores por defecto. Para este último utiliza los métodos set para darle a los atributos un valor.

Para cada objeto, deberá comprobar si la persona está en su peso ideal, tiene sobrepeso o está por debajo de su peso ideal. Mostrar un mensaje.

Indicar para cada objeto si la persona es mayor de edad.

Por último, se debe mostrar la información completa de cada objeto, es decir, los valores de todos sus atributos.

Clases de Utilidad en Java

Los métodos disponibles para las clases de utilidad *String, Integer, Math, Array, Date, Calendar* y *GregorianCalendar* se pueden consultar el "***Apéndice B***".

VER VIDEO: Clases String e Integer

Ver más acerca de la clase String

Ver más acerca de la clase Integer

15. Realizar una clase llamada *Cadena* que tenga como atributos una frase (de tipo de cadena) y su longitud. La clase deber tener un constructor mediante el cual se inicialice la frase, con una o más palabras ingresadas por el usuario (separadas entre sí por un espacio en blanco) y se inicialice de manera automática su longitud. Deberá además implementar los siguientes métodos:
   - Método mostrarVocales(), deberá contabilizar la cantidad de vocales que tiene la frase ingresada.
   - Método invertirFrase(), deberá invertir la frase ingresada y mostrar la frase invertida por pantalla. Por ejemplo: Entrada: "casa blanca", Salida: "acnalb asac".
   - Método vecesRepetido(), deberá recibir por parámetro un carácter ingresado por el usuario y contabilizar cuántas veces se repite el carácter en la frase, por ejemplo: Entrada: frase = "casa blanca", car = 'a', Salida: El carácter 'a' se repite 4 veces.
   - Método compararLongitud(), deberá comparar la longitud de la frase que compone la clase con otra nueva frase ingresada por el usuario.
   - Método unirFrases(), deberá unir la frase contenida en la clase Cadena con una nueva frase ingresada por el usuario y mostrar la frase resultante.
   - Método reemplazar(), deberá reemplazar todas las letras “a” que se encuentren en la frase, por algún otro carácter seleccionado por el usuario.

VER VIDEO: Método Static y Clase Math

16. Realizar una clase llamada *Matemática* que tenga como atributos dos números reales con los cuales se realizarán diferentes operaciones matemáticas. La clase deber tener un constructor mediante el cual se inicialicen ambos valores en cero. Deberá además implementar los siguientes métodos:
   - Método devolverMayor() para retornar cuál de los dos atributos tiene el mayor valor
   - calcular potencia() para calcular la potencia del mayor valor de la clase elevado al menor número. Previamente se deben redondear ambos valores.
   - Método calculaRaiz(), para calcular la raíz cuadrada del menor de los dos valores. Antes de calcular la raíz cuadrada se debe obtener el valor absoluto del número.
   - Método sumaAngulos(), para calcular la suma de dos ángulos:
     *sen(a+b) = sen(a)*cos(b)+cos(a)*sen(b)*
     donde a y b son los dos valores que componen la clase.

VER VIDEO: Clase Arrays

Ver más acerca de la clase Arrays

17. Crea una clase en Java donde declares una variable de tipo array de Strings que contenga los doce meses del año, en minúsculas. A continuación declara una variable mesSecreto de tipo String, y hazla igual a un elemento del array (por ejemplo mesSecreto = mes[9]. El programa debe pedir al usuario que adivine el mes secreto. Si el usuario acierta mostrar un mensaje, y si no lo hace, pedir que vuelva a intentar adivinar el mes secreto. Un ejemplo de ejecución del programa podría ser este:

    Adivine el mes secreto. Introduzca el nombre del mes en minúsculas: febrero
    No ha acertado. Intente adivinarlo introduciendo otro mes: agosto
    No ha acertado. Intente adivinarlo introduciendo otro mes: octubre
    ¡Ha acertado!

18. Realizar un programa en Java donde se creen dos arreglos: el primero será un arreglo de 50 números reales, y el segundo, un arreglo de 20 números, también reales. El programa deberá inicializar el arreglo con números aleatorios y mostrarlo por pantalla. Luego, el arreglo se debe ordenar de menor a mayor y copiar los primeros 10 números ordenados al segundo arreglo de 20 elementos, y rellenar los 10 últimos elementos con el valor 0.5. Mostrar los dos arreglos resultantes: el ordenado de 50 elementos y el combinado de 20.

VER VIDEOS:

A. Clase Date

B. Clases Calendar y GregorianCalendar

Ver más acerca de la clase Date

Ver más acerca de las clases Calendar y GregorianCalendar

19. Implemente la clase *Persona*. Una persona tiene un nombre y una fecha de nacimiento, por lo que debe ser posible pedirle su nombre, fecha de nacimiento y edad.
    - Escribir un método calcularEdad() a partir de la fecha de nacimiento ingresada. Tener en cuenta que para conocer la edad de la persona también se debe conocer la fecha actual.
    - Agregar a la clase el método menorQue(Persona persona) que recibe como parámetro a otra persona y retorna true en caso de que el receptor tenga menor edad que la persona que se recibe como parámetro, o false en caso contrario.
    - Agregar un método de creación del objeto que respete la siguiente firma: Persona(String nombre, Date fechaNacimiento). Este método recibe como parámetros el nombre y la fecha de nacimiento de la persona a crear y retorna un objeto inicializado con los valores recibidos como parámetro.

Colecciones en Java

VER VIDEO: Lista

20. Crear una clase llamada *Palabra* que mantenga información sobre diferentes palabras que se le van a ir agregando por medio del método add(String). Al final, el programa debe permitirnos conocer el conjunto de palabras de una determinada longitud ingresada por el usuario. Este conjunto deberá estar ordenado alfabéticamente. Crear una aplicación que muestre toda la información que disponga la clase Palabra.

21. Diseñar un programa que lea una serie de valores numéricos enteros desde el teclado y los guarde en un ArrayList de tipo Integer. La lectura de números termina cuando se introduzca el valor -99. Este valor no se guarda en el ArrayList. A continuación, el programa mostrará por pantalla el número de valores que se han leído, su suma y su media (promedio). Por último se mostrarán todos los valores leídos, indicando cuántos de ellos son mayores que la media. Utilizar los siguientes 3 métodos para resolver el ejercicio:
    - Método leerValores(): pide por teclado los números y los almacena en el ArrayList. La lectura finaliza cuando se introduce el valor -99. El método devuelve mediante return el ArrayList con los valores introducidos.
    - Método calcularSuma(): Recibe como parámetro el ArrayList con los valores numéricos y calcula y devuelve su suma. En este método se utiliza un Iterator para recorrer el ArrayList.
    - Método mostrarResultados(): Recibe como parámetro el ArrayList, la suma y la media aritmética. Este método muestra por pantalla todos los valores, su suma y su media y calcula y muestra cuantos números son superiores a la media.

22. Crear una clase llamada *ListaCantantesFamosos* que disponga de un atributo ArrayList que contenga objetos de tipo *CantanteFamoso*.
    La clase debe tener un método que permita añadir objetos de tipo CantanteFamoso a la lista. La clase CantanteFamoso tendrá como atributos el nombre y discoConMasVentas, y los métodos getters y setters.
    Se debe además crear una clase Test con el método main que inicialice un objeto ListaCantantesFamosos y agregue manualmente 5 objetos de tipo CantanteFamoso a la la lista. Luego, se debe mostrar los nombres de cada cantante y su disco con más ventas por pantalla. Además, se debe pedir al usuario un nombre y disco con más ventas de otro cantante famoso, y una vez introducidos los datos mostrar la lista actualizada. Una vez mostrada la lista actualizada, se debe dar opción a elegir entre volver a introducir los datos de otro cantante o salir del programa. Se podrán introducir tantos datos de cantantes como se desee.

VER VIDEO: Conjunto

23. Se necesita implementar un sistema en el que se puedan cargar alumnos, a los cuales los caracterizan el nombre y apellido, el número de legajo, el sexo, condición (regular o condicional) y la nota final. Estos alumnos se deben cargar en una asignatura, llamada Curso

Programación Egg. Implemente las clases y métodos necesarios para esta situación, teniendo en cuenta lo que se pide a continuación:

- Mostrar en pantalla todos los alumnos que se encuentren en la asignatura.
- Mostrar en pantalla los alumnos que se encuentren como condicional y su cantidad.
- Ordenar los alumnos de acuerdo a su nota (de mayor a menor) y mostrarlo en pantalla.
- Ordenar los alumnos de acuerdo a su nota (de menor a mayor) y mostrarlo en pantalla.
- Ordenar los alumnos por nombre y apellido y mostrarlo en pantalla

Nota: para los ordenamientos utilizar las facilidades provistas por la plataforma Java.

24. Implemente un programa para una *Librería* haciendo uso de conjuntos para evitar datos repetidos**.** Para ello, se debe crear una clase llamada Libro que guarde la información de cada uno de los libros de una Biblioteca. La clase Libro debe guardar el título del libro, autor, número de ejemplares del libro y número de ejemplares prestados.
La clase Librería contendrá además los siguientes métodos:
    - Constructor por defecto.
    - Constructor con parámetros.
    - Métodos Setters/getters
    - Método *préstamo* que incrementa el atributo correspondiente cada vez que se realice un préstamo del libro. No se podrán prestar libros de los que no queden ejemplares disponibles para prestar. Devuelve true si se ha podido realizar la operación y false en caso contrario.
    - Método *devolución* que decremente el atributo correspondiente cuando se produzca la devolución de un libro. No se podrán devolver libros que no se hayan prestado. Devuelve true si se ha podido realizar la operación y false en caso contrario.
    - Método *toString* para mostrar los datos de los libros.

VER VIDEO: Mapa

25. Se necesita una aplicación para una tienda en la cual queremos almacenar los distintos productos que venderemos y el precio que tendrán. Además, se necesita que la aplicación cuente con las funciones básicas, como introducir un elemento, modificar su precio, eliminar un producto y mostrar los productos que tenemos con su precio (Utilizar Hashmap).

26. Desarrollar un simulador del sistema de votación de facilitadores de EGG.
    - La clase *Simulador* debe ser tener un método que retorna un listado de alumnos generados de manera aleatoria. Las combinaciones de nombre, apellido y dni deben ser generadas de manera aleatoria. Se sugiere almacenar una lista con nombres y una lista con apellidos, y luego, que el programa genere automáticamente de manera aleatoria combinaciones de los mismos.
    - Las combinaciones de DNI posibles deben estar dentro de un rango real de números de documentos.
    - Se debe imprimir por pantalla el listado de alumnos.
    - El método simular de la clase Simulator recibe el listado de alumnos y para cada alumno genera tres votos de manera aleatoria. Tener en cuenta que un alumno no puede votarse

a sí mismo o votar más de una vez al mismo alumno. Utilizar un hashset para resolver esto. Todos los alumnos deben guardarse en una lista de alumnos.

- El recuento de votos recibe la lista de alumnos votados y comienza a hacer el recuento de votos utilizando la clase voto. Para hacer el recuento de votos utilizar la clase hashmap.
- Se deben crear 5 facilitadores con los 5 primeros alumnos votados y se deben crear 5 facilitadores suplentes con los 5 segundos alumnos más votados.

Relaciones entre Clases

VER VIDEOS:

A. Identificar nuevas clases

B. Uso y Composición

27 Nos piden hacer un programa sobre un *Cine* (solo de una sala) que tiene un conjunto de asientos (8 filas por 9 columnas). Del cine nos interesa conocer la película que se está reproduciendo y el precio de la entrada. Luego, de las películas nos interesa saber el título, duración, edad mínima y director, y por último, del espectador, nos interesa saber su nombre, edad y el dinero que tiene disponible.

Los asientos son etiquetados por una letra (columna) y un número (fila), la fila 1 empieza al final de la matriz como se muestra en la tabla. También deberemos saber si está ocupado o no el asiento.

8 A 8 B 8 C 8 D 8 E 8 F 8 G 8 H 8 I
7 A 7 B 7 C 7 D 7 E 7 F 7 G 7 H 7 I
6 A 6 B 6 C 6 D 6 E 6 F 6 G 6 H 6 I
5 A 5 B 5 C 5 D 5 E 5 F 5 G 5 H 5 I
4 A 4 B 4 C 4 D 4 E 4 F 4 G 4 H 4 I
3 A 3 B 3 C 3 D 3 E 3 F 3 G 3 H 3 I
2 A 2 B 2 C 2 D 2 E 2 F 2 G 2 H 2 I
1 A 1 B 1 C 1 D 1 E 1 F 1 G 1 H 1 I

Se debe realizar una pequeña simulación, en la que se generen muchos espectadores y se los ubique en los asientos aleatoriamente (no se puede ubicar un espectador donde ya este ocupado el asiento). Los espectadores serán ubicados de uno en uno.
Tener en cuenta que sólo se podrá sentar a un espectador si tiene el dinero suficiente para pagar la entrada, si hay espacio libre en la sala y si tiene la edad requerida para ver la película. En caso de que el asiento este ocupado se le debe buscar uno libre.

28 Realizar el juego de la ruleta rusa en Java. Como muchos saben, el juego se trata de un número de jugadores que con un revolver que posee una sola bala en el tambor se dispara en la cabeza. Las clases a hacer son del juego son las siguientes:

Clase Revolver: esta clase posee los siguientes atributos: posición actual (posición del tambor donde se dispara, puede que esté la bala o no) y posición bala (la posición del tambor donde se encuentra la bala). Estas dos posiciones, se generarán aleatoriamente.
Métodos:

- disparar(): devuelve true si la bala coincide con la posición actual
- siguienteBala(): cambia a la siguiente posición del tambor
- toString(): muestra información del revolver (posición actual y donde está la bala)

Clase Jugador: esta clase posee los siguientes atributos: id (representa el número del jugador, empieza en 1), nombre (Empezara con Jugador más su ID, "Jugador 1" por ejemplo) y vivo (indica si está vivo o no el jugador)
Métodos:

- disparar(Revolver r): el jugador se apunta y se dispara, si la bala se dispara, el jugador muere.

Clase Juego: esta clase posee los siguientes atributos: Jugadores (conjunto de Jugadores) y Revolver
Métodos:

- finJuego(): cuando un jugador muere, devuelve true
- ronda(): cada jugador se apunta y se dispara, y luego se informara del estado de la partida (El jugador se dispara, no ha muerto en esa ronda, etc.).

El número de jugadores será decidido por el usuario, pero debe ser entre 1 y 6. Si no está en este rango, por defecto será 6. En cada turno uno de los jugadores, dispara el revólver, si éste tiene la bala el jugador muere y el juego termina.

29. Realizar una baraja de cartas españolas orientada a objetos. Una carta tiene un número entre 1 y 12 (el 8 y el 9 no los incluimos) y un palo (espadas, bastos, oros y copas). Esta clase debe contener un método toString() que retorne el número de carta y el palo. La baraja estará compuesta por un conjunto de cartas, 40 exactamente.

Las operaciones que podrá realizar la baraja son:

- barajar(): cambia de posición todas las cartas aleatoriamente.
- siguienteCarta(): devuelve la siguiente carta que está en la baraja, cuando no haya más o se haya llegado al final, se indica al usuario que no hay más cartas.
- cartasDisponibles(): indica el número de cartas que aún se puede repartir.
- darCartas(): dado un número de cartas que nos pidan, le devolveremos ese número de cartas. En caso de que haya menos cartas que las pedidas, no devolveremos nada pero debemos indicárselo al usuario.
- cartasMonton(): mostramos aquellas cartas que ya han salido, si no ha salido ninguna indicárselo al usuario
- mostrarBaraja(): muestra todas las cartas hasta el final. Es decir, si se saca una carta y luego se llama al método, este no mostrara esa primera carta.

VIDEO: Herencia y Polimorfismo

30 Tenemos una clase padre *Animal* junto con sus 3 clases hijas *Perro*, *Gato*, *Caballo*. Crear un método abstracto en la clase Animal a través del cual cada clase hija deberá mostrar luego un mensaje por pantalla informando de que se alimenta. Generar una clase Main que realice lo siguiente:

```
1  public class Main {
2
3      public static void main(String[] args) {
4
5          //-->Declaracion del objeto PERRO
6              Animal perro = new Perro("Stich","Carnivoro",15,"Doberman");
7          perro.Alimentarse();
8          //-->Declaracion de otro objeto PERRO
9          Perro perro1 = new Perro("Teddy","Croquetas",10,"Chihuahua");
10         perro1.Alimentarse();
11
12
13         //-->Declaracion del objeto Gato
14         Animal gato = new Gato("Pelusa","Galletas",15,"Siames");
15         gato.Alimentarse();
16         //-->Declaracion del objeto Caballo
17         Animal caballo = new Caballo("Spark","Pasto",25,"Fino");
18         caballo.Alimentarse();
19
20     }
21 }
```

31 Crear una superclase llamada *Electrodomestico* con los siguientes atributos: precio base, color, consumo energético (letras entre A y F) y peso. Por defecto, el color será blanco, el consumo energético sera F, el precioBase de $1000 y el peso de 5 kg. Los colores disponibles para los electrodomésticos son blanco, negro, rojo, azul y gris. No importa si el nombre está en mayúsculas o en minúsculas.

Los constructores que se deben implementar son los siguientes:

- Un constructor por defecto.
- Un constructor con el precio y peso. El resto de los atributos por defecto.
- Un constructor con todos los atributos pasados por parámetro.

Los métodos a implementar son:

- Métodos getters y setters de todos los atributos.
- Método comprobarConsumoEnergetico(char letra): comprueba que la letra es correcta, sino es correcta usara la letra por defecto. Este método se debe invocar al crear el objeto y no será visible.
- Método comprobarColor(String color): comprueba que el color es correcto, y si no lo es, usa el color por defecto. Este método se invocará al crear el objeto y no será visible.
- Método precioFinal(): según el consumo energético y su tamaño, aumentará el precio. Esta es la lista de precios:

LETRA	PRECIO
A	$1000
B	$800
C	$600
D	$500
E	$300
F	$100

TAMAÑO	PRECIO
Entre 0 y 19 kg	$100
Entre 20 y 49 kg	$500
Entre 50 y 79 kg	$800
Mayor que 80 kg	$1000

A continuación, se debe crear una subclase llamada *Lavadora* con el atributo carga, además de los atributos heredados. Por defecto, la carga es de 5 kg.

Los constructores que se implementarán serán:

- Un constructor por defecto.
- Un constructor con el precio y peso. El resto por defecto.
- Un constructor con la carga y el resto de atributos heredados. Recuerda que debes llamar al constructor de la clase padre.

Los métodos que se implementara serán:

- Método get y set del atributo carga.
- Método precioFinal(): si tiene una carga mayor de 30 kg, aumentará el precio en $500, si la carga es menor o igual, no se incrementará el precio. Este método debe llamar al método padre y añadir el código necesario. Recuerda que las condiciones que hemos visto en la clase Electrodomestico también deben afectar al precio.

Se debe crear también una subclase llamada *Televisor* con los siguientes atributos: resolución (en pulgadas) y sintonizador TDT (booleano), además de los atributos heredados. Por defecto, la resolución será de 20 pulgadas y el sintonizador será false.

Los constructores que se implementarán serán:

- Un constructor por defecto.
- Un constructor con el precio y peso. El resto por defecto.
- Un constructor con la resolución, sintonizador TDT y el resto de atributos heredados. Recuerda que debes llamar al constructor de la clase padre.

Los métodos que se implementara serán:

- Método get y set de los atributos resolución y sintonizador TDT.
- Método precioFinal(): si el televisor tiene una resolución mayor de 40 pulgadas, se incrementará el precio un 30% y si tiene un sintonizador TDT incorporado, aumentará

$500. Recuerda que las condiciones que hemos visto en la clase Electrodomestico también deben afectar al precio.

Finalmente, se debe crear una clase ejecutable que realice lo siguiente:
Crear un array de Electrodomesticos de 10 posiciones. Asignar a cada posición un objeto de las clases anteriores con los valores que desees. Luego, recorrer este array y ejecutar el método precioFinal(). Se deberá también mostrar el precio de cada tipo de objeto, es decir, el precio de todos los televisores, por un lado, el de las lavadoras por otro, y la suma de todos los Electrodomesticos. Por ejemplo, si tenemos una lavadora con un precio de 2000 y un televisor de 5000, el resultado final será de 7000 (2000+5000) para electrodomésticos, 2000 para lavadora y 5000 para televisor.

32. Ahora se debe realizar unas mejoras a la clase Baraja del ejercicio 29. Lo primero que haremos es que nuestra clase Baraja será la clase padre y será abstracta. Le añadiremos el número de cartas en total y el número de cartas por palo. El método crearBaraja() será abstracto.
La clase Carta tendrá un atributo genérico que será el palo de nuestra versión anterior.
Creamos dos Enumeraciones (ver: https://javadesdecero.es/avanzado/enumerados-enum-ejemplos/):

PalosBarEspañola:
OROS
COPAS
ESPADAS
BASTOS

PalosBarFrancesa:
DIAMANTES
PICAS
CORAZONES
TREBOLES

Crear dos clases hijas:

BarajaEspañola: tendrá un atributo boolean para indicar si queremos jugar con las cartas 8 y 9 (total 48 cartas) o no (total 40 cartas).

BarajaFrancesa: no tendrá atributos, el total de cartas es 52 y el número de cartas por palo es de 13. Esta clase tendrá dos métodos llamados:

- cartaRoja(Carta<PalosBarFrancesa> c): si el palo es de corazones y diamantes.
- cartaNegra(Carta<PalosBarFrancesa> c): si el palo es de tréboles y picas.

De la carta modificaremos el método toString().

Si el palo es de tipo PalosBarFrancesa:
La carta número 11 será Jota
La carta numero 12 será Reina

La carta numero 13 será Rey
La carta numero 1 será As

Si el palo es de tipo PalosBarEspañola:
La carta numero 10 será Sota
La carta numero 12 será Caballo
La carta numero 13 será Rey
La carta numero 1 será As

33. En un puerto se alquilan amarres para barcos de distinto tipo. Para cada *Alquiler* se guarda el nombre y DNI del cliente, las fechas inicial y final de alquiler, la posición del amarre y el barco que lo ocupará. Un *Barco* se caracteriza por su matrícula, su eslora en metros y año de fabricación. Un alquiler se calcula multiplicando el número de días de ocupación (incluyendo los días inicial y final) por un valor módulo de cada barco (obtenido simplemente multiplicando por 10 los metros de eslora) y por un valor fijo (2 es en la actualidad). Sin embargo, se pretende diferenciar la información de algunos tipos de barcos:

- Número de mástiles para veleros
- Potencia en CV para embarcaciones deportivas a motor
- Potencia en CV y número de camarotes para yates de lujo.

El módulo de los barcos de un tipo especial se obtiene como el módulo normal más:

- El número de mástiles para veleros
- La potencia en CV para embarcaciones deportivas a motor
- La potencia en CV más el número de camarotes para yates de lujo.

Utilizando la herencia de forma apropiada, diseñe el diagrama de clases y sus relaciones, con detalle de atributos y métodos necesarios. Luego, programe en Java los métodos que permitan calcular el alquiler de cualquier tipo de barco.

34. Se plantea desarrollar un programa que permita representar la siguiente situación. Una instalación deportiva es un recinto delimitado donde se practican deportes. Se debe disponer de un método getTipoDeInstalacion(). Un edificio es una construcción (instalación) cubierta y se requiere disponer de un método getSuperficieEdificio(). Un polideportivo es al mismo tiempo una instalación deportiva y un edificio; y lo que interesa conocer es la superficie que tiene y su nombre. Un edificio de oficinas es un edificio; interesa conocer el número de oficinas que tiene. Definir las interfaces y las clases necesarias para representar la situación anterior. Luego, crear una clase Test con el método main y dos ArrayList. El primer ArrayList debe contener tres polideportivos, y el segundo, dos edificios de oficinas. Utilizar un iterator para recorrer las colecciones y mostrar los atributos de cada elemento.

Una vez realizado el ejercicio responda: ¿Entre qué clases existe una relación que se asemeja a la herencia múltiple?

35. Una compañía de promociones turísticas desea mantener información sobre la infraestructura de alojamiento para turistas, de forma tal que los clientes puedan planear sus vacaciones de acuerdo a sus posibilidades. Los alojamientos se identifican por un nombre,

una dirección, una localidad y un gerente encargado del lugar. La compañía trabaja con dos tipos de alojamientos: Hoteles y Alojamientos Extrahoteleros.

Los Hoteles pueden ser de tres, cuatro o cinco estrellas. Las características de las distintas categorías son las siguientes:

- Hotel *** Cantidad de Habitaciones, Número de Camas, Cantidad de Pisos, Precio de Habitaciones.
- Hotel **** Cantidad de Habitaciones, Número de camas, Cantidad de Pisos, Gimnasio, Nombre del Restaurante, Capacidad del Restaurante, Precio de las Habitaciones.
- Hotel ***** Cantidad de Habitaciones, Número de camas, Cantidad de Pisos, Gimnasio, Nombre del Restaurante, Capacidad del Restaurante, Cantidad Salones de Conferencia, Cantidad de Suites, Cantidad de Limosinas, Precio de las Habitaciones.

Los gimnasios pueden ser clasificados por la empresa como de tipo “A” o de tipo “B”, de acuerdo a las prestaciones observadas. Las limosinas están disponibles para cualquier cliente, pero sujeto a disponibilidad, por lo que cuanto más limosinas tenga el hotel, más caro será.

El precio de una habitación debe calcularse de acuerdo a la siguiente fórmula: PrecioHabitación = $50 + ($1 x capacidad del hotel) + (valor agregado por restaurante) + + (valor agregado por gimnasio) + (valor agregado por limosinas).

Donde:

Valor agregado por el restaurante:

- $10 si la capacidad del restaurante es de menos de 30 personas.
- $30 si está entre 30 y 50 personas.
- $50 si es mayor de 50.

Valor agregado por el gimnasio:

- $50 si el tipo del gimnasio es A.
- $30 si el tipo del gimnasio es B.

Valor agregado por las limosinas:

- $15 por la cantidad de limosinas del hotel.

En contraste, los Alojamientos Extra hoteleros proveen servicios diferentes a los de los hoteles, estando más orientados a la vida al aire libre y al turista de bajos recursos. Por cada Alojamiento Extrahotelero se indica si es privado o no, así como la cantidad de metros cuadrados que ocupa. Existen dos tipos de alojamientos extrahoteleros: los Camping y las Residencias. Para los Camping se indica la capacidad máxima de carpas, la cantidad de baños disponibles y si posee o no un restaurante dentro de las instalaciones. Para las residencias se indica la cantidad de habitaciones, si se hacen o no descuentos a los gremios y si posee o no campo deportivo. Realizar un programa en el que se representen todas las relaciones descriptas. Realizar un sistema de consulta que le permite al usuario consultar por diferentes criterios:

- todos los alojamientos
- todos los hoteles de una determinada localidad
- todos los campings de una determinada localidad

36. Sistema *Gestión Facultad*. Se pretende realizar una aplicación para una facultad que gestione la información sobre las personas vinculadas con la misma y que se pueden clasificar en tres tipos: estudiantes, profesores y personal de servicio. A continuación, se detalla qué tipo de información debe gestionar esta aplicación:
- Por cada persona, se debe conocer, al menos, su nombre y apellidos, su número de identificación y su estado civil.
- Con respecto a los empleados, sean del tipo que sean, hay que saber su año de incorporación a la facultad y qué número de despacho tienen asignado.
- En cuanto a los estudiantes, se requiere almacenar el curso en el que están matriculados.
- Por lo que se refiere a los profesores, es necesario gestionar a qué departamento pertenecen (lenguajes, matemáticas, arquitectura, ...).
- Sobre el personal de servicio, hay que conocer a qué sección están asignados (biblioteca, decanato, secretaría, ...).

El ejercicio consiste, en primer lugar, en definir la jerarquía de clases de esta aplicación. A continuación, debe programar las clases definidas en las que, además de los constructores, hay que desarrollar los métodos correspondientes a las siguientes operaciones:
- Cambio del estado civil de una persona.
- Reasignación de despacho a un empleado.
- Matriculación de un estudiante en un nuevo curso.
- Cambio de departamento de un profesor.
- Traslado de sección de un empleado del personal de servicio.
- Imprimir toda la información de cada tipo de individuo. Incluya un programa de prueba que instancie objetos de los distintos tipos y pruebe los métodos desarrollados.

37. Extender el ejercicio anterior incluyendo una clase que represente al centro docente. Esa clase incluirá 3 contenedores, uno por cada tipo de persona vinculada con el centro. En una primera fase deben incluirse los siguientes métodos:
- Uno para dar de alta una persona, que incorporará a la persona en la lista correspondiente.
- Otro para dar de baja una persona, dado su DNI. Añada un método a la clase persona para poder obtener el DNI de un objeto de esa clase.
- Uno último para imprimir toda la información de las personas vinculadas con el centro.

Dado que hay múltiples alternativas a la hora de afrontar este problema, a continuación, se explican las características de la solución implementada, aunque el alumno podrá realizar el diseño que considere oportuno:
- Se usan como contenedores listos de estudiantes, profesores y personal de servicio, respectivamente.
- El método de alta recibe como parámetro un objeto del tipo persona correspondiente, existiendo, por tanto, tres versiones sobrecargadas del mismo.

- Cada versión del método crea un objeto del tipo que le corresponde y lo inserta en la lista.
- El método de baja busca, en primer lugar, un objeto en las listas que tenga el DNI recibido como parámetro. Una vez encontrado, lo elimina de la lista.

38. Guarde el ejercicio anterior y realice este ejercicio sobre una copia de la misma. En la aplicación anterior, se detecta que en el futuro se va a necesitar crear nuevos tipos de empleados (por ejemplo, investigadores) y distinguir entre distintos tipos de estudiantes. Por tanto, para que el diseño se pueda adaptar a estas necesidades futuras, se plantea unificar todos los contenedores del tipo centro, de manera que sólo haya un único contenedor de personas, y usar polimorfismo para gestionar los distintos tipos de personas. Modifique el programa anterior para adaptarlo a este nuevo diseño. Tenga en cuenta que debe desaparecer cualquier referencia a profesores, personal de servicio y estudiantes en el código de la clase que representa al centro. Asimismo, haga que las clases que corresponden a personas y empleados sean abstractas.

39. En la versión correspondiente al ejercicio anterior, se va a añadir un nuevo método a las clases profesor y personal de servicio para calcular su salario. Se va a suponer que la forma de calcular el salario para estos dos tipos de empleados es totalmente distinta y que no se puede sacar ningún código común para incluirlo en la clase empleado. Haga la suposición que considere oportuna sobre cómo calcular el sueldo de cada tipo de empleado, dado que no es importante para el ejemplo. Aquí va una propuesta bastante ridícula: el personal de servicio cobra una cantidad fija más un 5% si están casados; el personal docente gana un fijo más un 8% si su fecha de incorporación es anterior al año 2000. Habilite los métodos en la clase base que necesite para obtener los datos que requiera para el cálculo del sueldo. Una vez añadidos los métodos para calcular el salario en cada una de las dos clases, debe añadir un método en la clase que representa el centro que imprima el salario de todos los empleados del centro (nombre y apellidos más el sueldo).

Manejo de Excepciones

VER VIDEOS:

A. **Excepciones I**
B. **Excepciones II**

40. Inicializar un objeto de la clase *Persona* ejercicio 14 a null y tratar de invocar el método esMayorDeEdad() a través de esta referencia. Luego, englobe el código con una cláusula try-catch para probar la nueva excepción que debe ser controlada.

41. 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 matriz fuera de rango).

42. 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.

43 Defina una clase llamada DivisionNumero. En el método main utilice el método readLine() para leer dos números en forma de cadena. A continuación, utilice el método parseInt() para convertir las cadenas al tipo int y guardarlas en dos variables de tipo int (numero1 y numero2). Divida numero1 por numero2 y muestre el resultado. El ingreso por teclado puede causar una excepción de tipo IOException. Recordá que el método Integer.parseInt() convierte números de tipo String a int. Si la cadena no puede convertirse a entero, arroja una NumberFormatException. Además, al dividir un número por cero surge una ArithmeticException. Manipule las posibles excepciones utilizando un bloque de región segura try-catch.

44 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
    }
}
```

45 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
    }
}
```

46. 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();
		}
	}
}
```

47 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

48 *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.