Lanzar excepciones

DigitalHouse >
Coding School

Certified Tech Developer
The Ultimate Degree

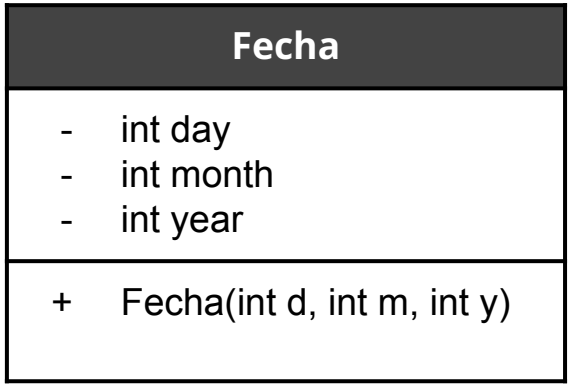
Índice

1. Proteger integridad
2. RuntimeException
3. Lanzar una excepción

1 Proteger integridad

Proteger la integridad de una clase

Cuando creamos una clase, estamos tratando de representar algo que tiene un cierto comportamiento. Los valores que se guardan en sus atributos pueden tener que respetar un rango de valores, en ese caso, tenemos que proteger la integridad de la clase, veamos un ejemplo.

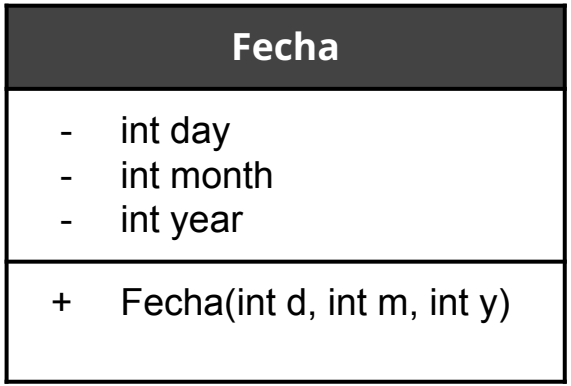


Usamos los nombres de atributos en inglés para no tener que usar la "**ñ**".

Proteger la integridad de una clase

Una fecha es algo bien conocido por todos, pero si la clase representa algo no tan habitual, quien tiene que utilizar la clase no tiene por qué saber con qué rango de valores se debe trabajar.

A fines prácticos, vamos a establecer que los días pueden estar entre 1 y 31 sin importar el mes y los meses entre 1 y 12.



2 | RuntimeException

RuntimeException

```
class Fecha{
     private int day;
     private int month;
     private int year;

     public Fecha(int d, int m, int y){
          day=d;
          month=m;
          year=y;
     }
}
```

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

     Fecha fecha= new Fecha(100,-100,1000);
}
```

Al usar la clase Fecha, para crear una nueva fecha, es necesario pasar 3 valores enteros.

Esto se está cumpliendo en el ejemplo, pero quien usa una clase no necesariamente entiende el comportamiento de esa clase.

La clase se debe proteger a sí misma y evitar que le lleguen valores que no estén en el rango esperado.

RuntimeException

```
class Fecha{
     private int day;
     private int month;
     private int year;

     public Fecha(int d, int m, int y){
          if (d<1||d>31||m<1||m>12)
               throw new RuntimeException("Los valores no son válidos");
          day=d;
          month=m;
          year=y;
     }
}
```

Con throw lanzamos una excepción en ejecución, para hacerlo la creamos con el new la nueva excepción.

RuntimeException

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

     Fecha fecha= new Fecha(100,-100,1000);
}
```

```
Exception in thread "main" java.lang.RuntimeException: Los valores no son válidos
```

Ahora, si intentamos crear con valores inválidos, nos genera una excepción. Las excepciones de tipo RuntimeException no es obligatorio protegerlas con los bloques try / catch.

3 Lanzar una excepción

Lanzar una excepción por código

```
class Fecha{
     private int day;
     private int month;
     private int year;

     public Fecha(int d, int m, int y){
          day=d;
          month=m;
          year=y;
     }
}
```

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

     Fecha fecha= new Fecha(100,-100,1000);
}
```

Veamos ahora otra forma de proteger este código. Ahora vamos a hacerlo de forma que quien utilice el método esté forzado a protegerlo con try / catch.

Lanzar una excepción

```
class Fecha{
     private int day;
     private int month;
     private int year;

     public Fecha(int d, int m, int y) throws Exception{
          if (d<1||d>31||m<1||m>12)
               throw new Exception("Los valores no son válidos");
          day=d;
          month=m;
          year=y;
     }}
```

De la misma forma que antes lanzamos la excepción con **throw**, pero en este caso es de tipo **Exception**, ya que no hay definida una excepción que diga fuera del rango esperado.
Se agrega que debemos avisar que el método puede generar una excepción, agregamos, a continuación de la firma **throws Exception**. Es un método **throweable**

RuntimeException

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

     try{

          Fecha fecha= new Fecha(100,-100,1000);
     catch (Exception e) {
          System.err.println("No son valores válidos para una fecha");
     }

}
```

Por ser un método throweable, nos obliga a protegerlo con try / catch.


DigitalHouse>
Coding School