Catálogo de restricciones sobre los atributos de un objeto

Autor: Mariano González. **Revisores**: Fermín Cruz, José Riquelme, Toñi Reina. **Última modificación**: 26/3/2020

Este documento contiene un catálogo de las restricciones más habituales que nos podemos encontrar en nuestros tipos, sin pretender ser exhaustivo.

A) Restricciones generales

- El valor no puede ser nulo

B) Restricciones según el tipo del atributo

Tipos numéricos

- El número debe ser mayor o menor que un valor dado
- El número debe estar dentro de un rango de valores dado
- El número solo puede tomar unos valores dados

Tipo String

- La cadena no debe superar una longitud dada
- La cadena debe contener un carácter dado
- La cadena debe comenzar o terminar por un carácter dado
- La cadena debe comenzar o terminar por un conjunto de caracteres dado
- La cadena debe ser diferente a un valor dado

Tipos fecha y hora

- La fecha (hora) debe ser anterior o posterior a una fecha (hora) dada
- La fecha (hora) debe estar dentro de un intervalo de fechas (horas) dado

Notas:

- El valor al que se hace referencia en las restricciones puede ser un valor constante (el número 0, la cadena vacía, la fecha de hoy…) o el valor de otro atributo del objeto (la velocidad media no puede superar la máxima, el destino debe ser distinto al origen, la fecha final debe ser posterior a la inicial…)
- La restricción puede aplicarse a una parte del atributo. Por ejemplo, la restricción sobre una fecha puede ser que el año de la fecha sea mayor que un valor dado.
- Las restricciones se pueden combinar. Por ejemplo, una cadena puede comenzar por una letra dada y no superar una longitud dada. En tal caso deben cumplirse ambas restricciones.

C) Restricciones complejas

A veces, comprobar la restricción no es tan simple como comparar dos valores, sino que puede conllevar la realización de un **tratamiento secuencial**. Por ejemplo, ver si una cadena de caracteres sigue un patrón dado.

Ejemplos

Para ilustrar las restricciones usaremos como ejemplo el tipo Vuelo, que tiene las siguientes propiedades:

- **Destino**, de tipo String
- **Precio**, de tipo Double
- **Número** de plazas, de tipo Integer
- **Número** de pasajeros, de tipo Integer
- **Código**, de tipo String
- **Fecha**, de tipo LocalDate
- **Duración**, de tipo Duration

Cada restricción se implementará de dos formas diferentes:

1. Utilizando un método privado.
2. Utilizando la clase Checkers. En este caso, a veces también crearemos un método privado, cuando la condición sea más compleja.

En el primer caso, se muestra el código del método privado, que hay que invocar desde los métodos que modifiquen el atributo del objeto, que son los métodos constructores y los métodos set. En el segundo caso, se muestra la invocación al método de la clase Checkers, que hay que colocar en el constructor o el método set.

A) Restricciones generales

Ejemplo: la fecha del Vuelo no puede ser nula.

Con método privado:

```
private void checkFecha(LocalDate fecha) {
	if (fecha == null) {
		throw new IllegalArgumentException(
			"La fecha no debe ser nula");
	}
}
```

Con la clase Checkers:

```
Checkers.checkNoNull(fecha);
```

B) Restricciones según el tipo del atributo

Tipos numéricos

Ejemplo: el precio del Vuelo debe ser mayor que 0.

Con método privado:

```
private void checkPrecio(Double precio) {
	if (precio <= 0) {
		throw new IllegalArgumentException(
			"El precio debe ser positivo");
	}
```

```
}
```

Con la clase Checkers:

```
Checkers.check("El precio debe ser positivo", precio > 0.0);
```

Ejemplo: el número de pasajeros del Vuelo debe ser mayor que cero e igual o inferior que el número de plazas.

Con método privado:

```
private void checknumPasajeros(Integer numPasajeros,
            Integer numPlazas) {
      if (numPasajeros <= 0 || numPasajeros > numPlazas) {
            throw new IllegalArgumentException(
                  "El número de pasajeros debe ser positivo y "
                  + "menor o igual que el número de plazas");
      }
}
```

Con la clase Checkers:

```
Checkers.check("El número de pasajeros debe ser positivo y "
            + "menor o igual que el número de plazas",
            numPasajeros > 0 && numPasajeros <= numPlazas);
```

Ejemplo: el número de plazas del Vuelo solo puede tomar los valores 170 y 220.

Con método privado:

```
private void checkNumPlazas(Integer numPlazas) {
      if (!numPlazas.equals(170) && !numPlazas.equals(220)) {
            throw new IllegalArgumentException(
                  "El número de plazas debe ser 170 o 220");
      }
}
```

Con la clase Checkers:

```
Checkers.check("El número de plazas debe ser 170 o 220",
            numPlazas.equals(170) || numPlazas.equals(220));
```

Tipo String

Ejemplo: el destino del Vuelo no puede tener más de 20 caracteres.

Con método privado:

```
private void checkDestino(String destino) {
      if (destino.length() > 20) {
            throw new IllegalArgumentException(
                  "El destino no puede tener más de 20 caracteres");
      }
}
```

Con la clase Checkers:

```
Checkers.check("El destino no puede tener más de 20 caracteres",
            destino.length() <= 20);
```

Ejemplo: el código del Vuelo debe contener el carácter '#'.

Con método privado:

```
private void checkCodigo(String codigo) {
	if (!codigo.contains("#")) {
		throw new IllegalArgumentException(
			"El código debe contener el carácter '#'");
	}
}
```

Con la clase Checkers:

```
Checkers.check("El código debe contener el carácter '#'",
		codigo.contains("#"));
```

Ejemplo: el código del Vuelo debe comenzar por una letra mayúscula.

Con método privado:

```
private void checkCodigo(String codigo) {
	if (!Character.isUpperCase(codigo.charAt(0))) {
		throw new IllegalArgumentException(
			"El código debe comenzar por una letra mayúscula");
	}
}
```

Con la clase Checkers:

```
Checkers.check("El código debe comenzar por una letra mayúscula",
		Character.isUpperCase(codigo.charAt(0)));
```

Ejemplo: el código del vuelo debe comenzar por la cadena "IB"

Con método privado:

```
private void checkCodigo(String codigo) {
	if (!codigo.startsWith("IB")) {
		throw new IllegalArgumentException(
			"El código debe comenzar por la cadena 'IB'");
	}
}
```

Con la clase Checkers:

```
Checkers.check("El código debe comenzar por la cadena 'IB'",
		codigo.startsWith("IB"));
```

Ejemplo: el destino del Vuelo no puede ser la cadena vacía.

Con método privado:

```
private void checkDestino(String destino) {
	if (destino.isEmpty()) {
		throw new IllegalArgumentException(
			"El destino no puede ser vacío");
	}
}
```

Con la clase Checkers:

```
Checkers.check("El destino no puede ser vacío",
            !destino.isEmpty());
```

Tipos fecha y hora

Ejemplo: la fecha de salida del Vuelo debe ser igual o posterior a la fecha actual.

Con método privado:

```
private void checkFecha(LocalDate fecha) {
      if (fecha.isBefore(LocalDate.now())) {
            throw new IllegalArgumentException(
                  "La fecha debe ser igual o posterior a la de hoy");
      }
}
```

Con la clase Checkers:

```
Checkers.check("La fecha debe ser igual o posterior a la de hoy",
            !fecha.isBefore(LocalDate.now()));
```

Ejemplo: la fecha de salida del Vuelo debe estar comprendida entre el 1 de julio y el 31 de agosto de 2020.

Con método privado:

```
private void checkFecha(LocalDate fecha) {
      LocalDate f1 = LocalDate.of(2020, 7, 1);
      LocalDate f2 = LocalDate.of(2020, 8, 31);
      if (fecha.isBefore(f1) || fecha.isAfter(f2)) {
            throw new IllegalArgumentException(
                  "La fecha debe estar comprendida entre "
                  + "el 1/7/2020 y el 31/8/2020");
      }
}
```

Con la clase Checkers:

```
Checkers.check("La fecha debe estar comprendida entre "
            + "el 1/7/2020 y el 31/8/2020",
            fechaEnIntervalo(fecha));


private boolean fechaEnIntervalo(LocalDate fecha) {
      LocalDate f1 = LocalDate.of(2020, 7, 1);
      LocalDate f2 = LocalDate.of(2020, 8, 31);
      return !(fecha.isBefore(f1) || fecha.isAfter(f2));
}
```

C) Restricciones complejas

Ejemplo: el código del Vuelo debe estar formado por 6 dígitos.

Con método privado:

```
private void checkCodigo(String codigo) {
	if (codigo.length() != 6 || !sonDigitos(codigo)) {
		throw new IllegalArgumentException(
			"El código debe estar formado por 6 dígitos");
	}
}

private boolean sonDigitos(String codigo) {
	boolean res = true;
	for (int i = 0; i < codigo.length(); i++) {
		res = res && Character.isDigit(codigo.charAt(i));
		if (!res) {
			break;
		}
	}
	return res;
}
```

Con la clase Checkers:

```
Checkers.check("El código debe estar formado por 6 dígitos",
		codigo.length() == 6 && sonDigitos(codigo));
```

(el método sonDigitos es el mismo que se ha visto más arriba)

Ejemplo: el código del Vuelo debe tener 6 caracteres, siendo los dos primeros caracteres alfabéticos en mayúsculas, y los cuatro siguientes dígitos. Por ejemplo, “IB2539”.

Con método privado:

```
private void checkCodigo(String codigo) {
	if (codigo.length() != 6
			|| !sonMayusculas(codigo.substring(0, 2))
			|| !sonDigitos(codigo.substring(2, 6))) {
		throw new IllegalArgumentException(
			"El código debe ajustarse al patrón AA0000");
	}
}

private boolean sonMayusculas(String s) {
	boolean res = true;
	for (int i = 0; i < s.length(); i++) {
		res = res && Character.isUpperCase(s.charAt(i));
		if (!res) {
			break;
		}
	}
	return res;
}
```

(el método sonDigitos es el mismo que se ha visto más arriba)

Con la clase Checkers:

```
Checkers.check("El código debe ajustarse al patrón AA0000",
		codigo.length() == 6
		&& sonMayusculas(codigo.substring(0, 2))
		&& sonDigitos(codigo.substring(2, 6)));
```

(los métodos sonDigitos y sonMayusculas son los mismos que se han visto más arriba)