Java Pattern Predicate nos puede ayudar a simplificar la forma que tenemos de trabajar con Java Streams a la hora de realizar filtrados. Normalmente cuando realizamos filtrados dentro de un Stream nos encontramos con que usamos algún tipo de Java Predicate . Vamos a ver un ejemplo sencillo de como usar Java Pattern Predicate , supongamos que tenemos un Stream de números.

```
package com.arquitecturajava;

import java.util.stream.Stream;

public class Principal {

  public static void main(String[] args) {
    Stream<Integer> stream=Stream.of(1,5,7,9,8,2);
    stream.filter(item->item>5).forEach(System.out::println);

  }

}
```

Es muy sencillo usar el high order function de filter y filtrar las notas que superan el 5 y son aprobados , si ejecutamos el código la consola los mostrará los valores mayores de 5.

```
7
9
8
```

Sin embargo existen situaciones que no son tan sencillas y que los valores pueden venir construidos de la siguiente forma.

```
package com.arquitecturajava;
```

```
import java.util.stream.Stream;

public class Principal2 {

  public static void main(String[] args) {
    Stream<Integer> stream=Stream.of(1,5,7,9,8,2);
    stream.filter(item->item>5).forEach(System.out::println);
    Stream<String> mistream=Stream.of("hola","4","6","7","estas","9");

  }

}
```

En este caso para obtener los valores numéricos tenemos que mezclar cadenas de texto y valores nunéricos que en este caso son cadenas de texto. ¿CÓmo podemos hacerlo? . Bueno una solución sencilla es usar expresiones regulares.

```
package com.arquitecturajava;

import java.util.regex.Matcher;
import java.util.regex.Pattern;
import java.util.stream.Stream;

public class Principal2 {

  public static void main(String[] args) {

    Stream<Integer> stream = Stream.of(1, 5, 7, 9, 8, 2);

    stream.filter(item -> item > 5).forEach(System.out::println);

    Pattern patron = Pattern.compile("\\d+");
```

```
    Matcher cumple = patron.matcher("7");

    System.out.println(cumple.matches());

  }

}
```

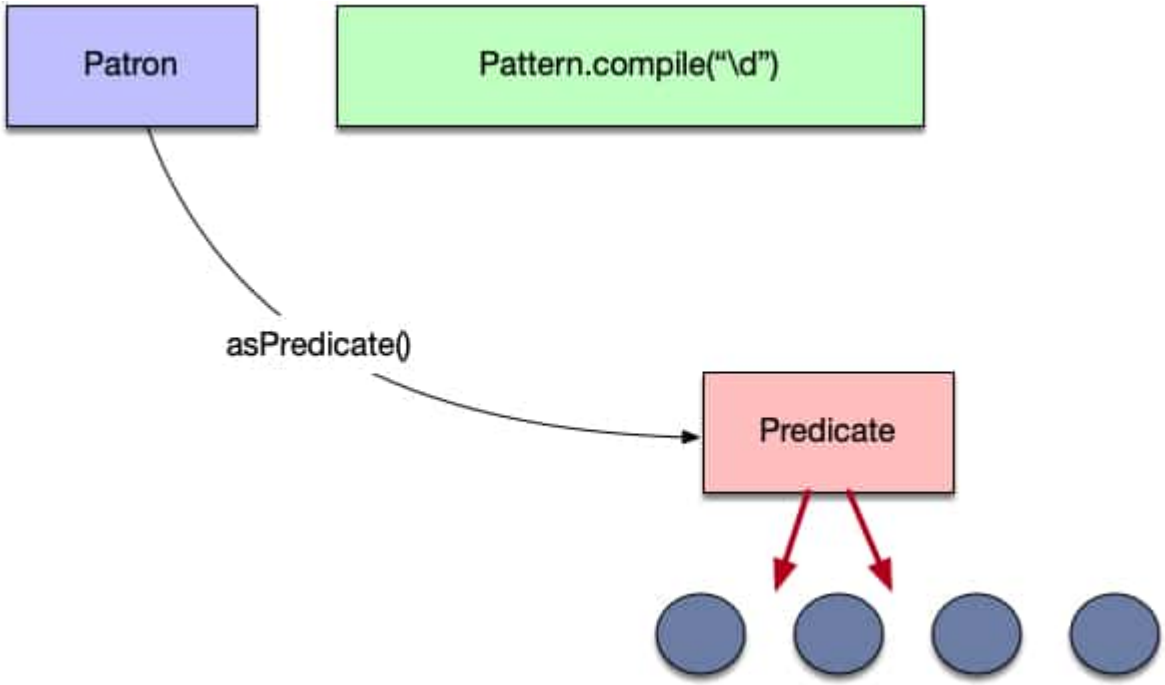
Con este código conseguimos a nivel de expresión regular validar que cumplimos con que el texto contiene uno o más digitos. Por lo tanto por la consola nos imprimirá true.

true

Nos queda combinarlo con el concepto de Predicate y Streams.

Java Pattern Predicate

Con Java 8 podemos realizar esta operación de forma muy sencilla ya que la clase Pattern aporta un método que convierte el patrón directamente en un predicate (Pattern.asPredicate) .



Así es muy sencillo convertir la expresión regular en un Predicado y utilizarla dentro de un Stream.

```
package com.arquitecturajava;

import java.util.regex.Pattern;
import java.util.stream.Stream;

public class Principal3 {

  public static void main(String[] args) {
    Pattern patron=Pattern
    .compile("\\d+");
    Stream<String> mistream=Stream.of("hola","4","6","7","estas","9");
    mistream.filter(patron.asPredicate())
    .map(Integer::parseInt)
    .filter(n->n>5)
    .forEach(System.out::println);
```

```
    }

}
```

El resultado únicamente incluirá los valores superiores a 5 acabamos de usar un Java Pattern Predicate.

```
6
7
9
```

Acabamos de ver otro uso del API de Java 8 para simplificar el trabajo con Java Stream y Lambdas.

Otros artículos relacionados

1. Java Stream Reduce , eliminando bucles
2. Java Stream Filter y Predicates
3. Java Stream String y Java 8
4. Java 8 Files Walk y Recursividad
5. Java Stream for loop y programación funcional
6. Expresiones Regulares