Data Validation Overview. V1

Validation first, followed by verification.

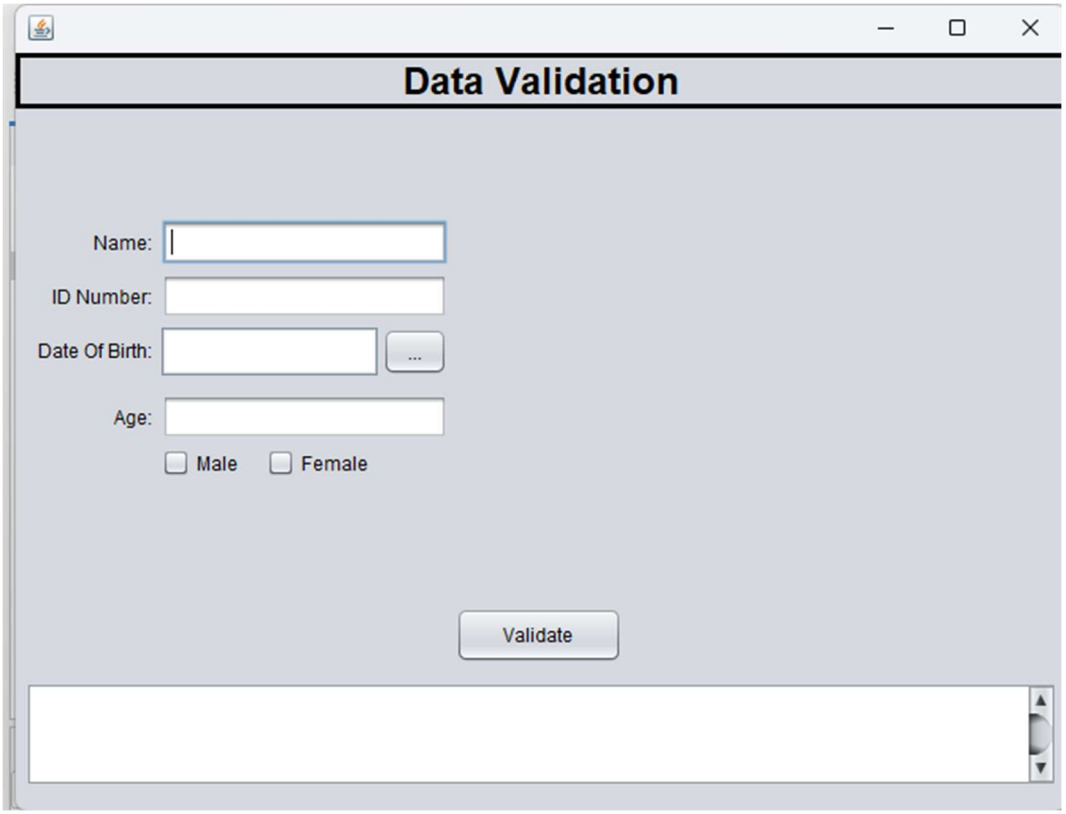
1. **Validation** – does the data match the template?
   Example: The date template could be "yyyy-MM-dd"
2. **Verification** – the data does match the template but is it valid?
   Example: "2026-14-13" matches the template but is not valid because no year has 14 months.

We validate in three ways.

- Object validation.
- String validation – always start with trim( )[1]
- Number validation – always start with trim( ) because data from a form starts off as a String.

Forms – GUIs – Using JFrameForm

All forms, like the one below, only accept text (String) into the open white fields. Therefore numbers (integers, doubles, dates, times) start off as String first and are then parsed to a number using "Integer.parseInt" , "Double.parseDouble", "LocalDate.parse", "LocalTime.parse" etc



[1] The String method trim( ) removes rubbish space characters before and/or after a String

Missing data, normal data, extreme data and abnormal data

We must trap (validate) the errors (or omissions) that a person makes when filling in an online form. Our code must only accept normal data.

- Missing data: Name:
- Normal data: Name = Steve Eilertsen.
- Extreme data: Name = Steve Eilertsennnnnnnnnnnnnnnnnnnnnnnnnnn.
- Abnormal data: Name = S$t&44 &e9*

Object validation

- Unwanted negative numbers and decimal values.
- File not found.
- Empty date.
- Empty time.
- Numbers with unwanted letters.

String Validation

- String is too long.
- String is null.
- String has no length (but is not null).
- String has strange characters.

Numbers

Validation

Using "try . . . catch with Integer.parseInt" or Double.parseDouble is an easy way to determine if a number is in a valid number format. If the format is acceptable then we can do further verification. See below.

- Numbers must be in an acceptable range
- Numbers cannot come from a String that is null

Verification

Once data has been validated it must be verified

Example: The date: 2026-14-17. This date will pass a validation test but not a verification test because there are not 14 months in a year.

```
 1 // Object validation. Objects - Date, Time, String, File. V1
 2
 3 // Integer.parseInt and Double.parseDouble with a try catch
 4 // to reject "bad" numbers.
 5
 6 // Objects with nothing in them - null value.
 7 // NumberFormatException when a number cannot be converted to a
different
 8 // datatype.
 9 // FileNotFoundException when the file cannot be found.
10
11 import java.time.LocalDate;
12 import java.time.LocalTime;
13 import java.util.Scanner;
14 import java.io.File;
15 import java.io.FileNotFoundException;
16
17 public class ValidationObject
18 {
19    public static void main(String[]args)
20    {
21       // Staring values
22       String st = "6";
23       st = st.trim();
24       int num = 0;
25       boolean error = false;
26
27 //===== ParseInt with an if statement to reject unwanted
   // negative numbers and decimal values ========================
29
30       try
31       {
32
33          if( Integer.parseInt(st) >=0)
34          {
35             int parseNum = Integer.parseInt(st);
36             System.out.println("The number is good");
37          }
38          else
39          {
40             System.out.println( "Error: Must be a positive whole
number");
41             error = true;
42          }
43
44       }
```

```
45
46       catch(NumberFormatException e)
47       {
48          System.out.println("Integer parseInt error");
49          error = true;
50
51       }
52
53 //========= File not found ====================================
54
55       try
56       {
57
58          Scanner scFile = new Scanner(new File("file.txt"));
59          System.out.println("File found");
60
61       }
62
63       catch(FileNotFoundException e)
64       {
65          System.out.println("File not found");
66          error = true;
67
68       }
69
70   //===== Empty date ============================================
71
72    LocalDate theDate = null, dob = null;
73    theDate = LocalDate.of(2024,4,2);
74    System.out.println("The date: " + theDate);
75
76    try
77    {
78          dob = theDate;
79          System.out.println("The dob is valid: " +
theDate.toString());
80    }
81    catch (NullPointerException e)
82    {
83          System.out.println("DOB is empty");
84          error = true;
85    }
86
87 //===== Empty time ========================================
88
89    LocalTime theTime = null, myTime = null;
90    theTime = LocalTime.of(13,14,15);
91    System.out.println("The time: " + theTime);
```

```
 92
 93    try
 94    {
 95          myTime = theTime;
 96          System.out.println("The time is valid: " +
theTime.toString());
 97    }
 98    catch (NullPointerException e)
 99    {
100          System.out.println("Time is empty");
101          error = true;
102    }
103
104 //====== parseInt - To reject numbers with unwanted letters. ===
105
106    String id = "20200115001803";
107    id = id.trim();
108
109    try
110    {
111       for(int x = 0; x < id.length(); x++)
112       {
113          char ch = id.charAt(x);
114          int check = Integer.parseInt("" + ch);
115
116       }
117
118       System.out.println("Id is good");
119
120    }
121    catch(NumberFormatException e)
122    {
123       System.out.println("The id must only have numbers
present");
124       error = true;
125
126    }
127
128
129 // ===== Final OUTPUT ========================================
130
131    System.out.println("Is an error present: " + error);
132
133    } // end main
134
135 }
```

```
 1 // String validation - Always start with trim. V1
 2 // Strings that are too long or too short or null
 3 // String with abnormal characters
 4
 5
 6 public class ValidateString
 7 {
 8
 9    public static void main(String[]args)
10    {
11
12       boolean error = false;
13
14 // ======= Use trim as a standard default to get rid of unwanted
 spaces in the beginning and end of a string. ======================
16
17    String myStringTrim = "     Bob the builder     ";
18    myStringTrim = myStringTrim.trim();
19    System.out.println(myStringTrim);
20
21
22 // =========== String is too long. ============================
23
24       String longName = "aaaaaaaaaaa bbbbbbbbb";
25       longName = longName.trim();
26
27       if(longName.length() > 35)
28       {
29          System.out.println("Name is too long");
30          error = true;
31
32       }
33       else
34       System.out.println("Name is correct length");
35
36
37 // ========== String is null. =========================
38
39    String theString = null, myString = null;
40    theString = "ABCDEFG";
41    theString = theString.trim();
42    System.out.println("The string: " + theString);
43
44    try
45    {
46          myString = theString;
47          System.out.println("The string is valid: " +
theString.toString());
```

```
 48    }
 49    catch (NullPointerException e)
 50    {
 51          System.out.println("String is empty");
 52          error = true;
 53    }
 54
 55
 56// ===== String has no length - no error message. Do not use this
 57
 58    String theString2 = null, myString2 = null;
 59    theString2 = "";
 60    System.out.println("The string: " + theString2);
 61
 62    try
 63    {
 64          myString2 = theString2;
 65          System.out.println("The string is valid: " +
theString2.toString());
 66    }
 67    catch (NullPointerException e)
 68    {
 69          System.out.println("String is empty");
 70          error = true;
 71    }
 72
 73 // ====== String has no length - Error message. Use this. ======
 74
 75    String theString3 = null, myString3 = null;
 76    theString3 = "W";
 77    theString3 = theString3.trim();
 78    System.out.println("The string: " + theString3);
 79
 80    myString3 = theString3;
 81    if(myString3.length() == 0)
 82    {
 83       System.out.println("Error. String has no length");
 84       error = true;
 85    }
 86
 87 // ===== String has strange characters - Error message. ========
 88
 89    String theString4 = null, myString4 = null;
 90    theString4 = "The end of the # world";
 91    theString4 = theString4.trim();
 92    System.out.println("The string: " + theString4);
 93
 94    myString4 = theString4;
```

```
 95    if(myString4.contains("$") || myString4.contains("%") ||
myString4.contains("#") ||
 96       myString4.contains("@"))
 97    {
 98       System.out.println("Error. String has abnormal
character(s).");
 99       error = true;
100    }
101
102 //================= FINAL OUTPUT. ============================
103    System.out.println("Is an error present: " + error);
104
105
106    } // end main
107
108 }
```

```
1 // Number validation - still use trim on the String before
2 // parsing to a number. V1
3
4 // Numbers that are too big, too small or negative.
5 // Using the correct datatype ie int versus double.
6
7 public class Validatenumber
8 {
9
10    public static void main(String[]args)
11    {
12
13       boolean error = false;
14
15
16 // ===== Can the number be parsed to a double number? ===========
17
18
19    String st = "10.6";
20    st = st.trim();
21
22    if(st == null)
23    {
24       error = true;
25    }
26
27    if(st != null)
28    {
29       try
30       {
31             double parseNum = Double.parseDouble(st);
32       }
33
34       catch(NumberFormatException e)
35       {
36          System.out.println("Double parse error");
37          error = true;
38       }
39
40    }
41
```

```
42 // ======== Numbers must be in an acceptable range. ============
43
44       int age = 15;
45
46       System.out.println("The age is :" + age);
47
48       if(age <= 0 || age >= 100)
49       {
50          System.out.println("Possible age error. " + age);
51          error = true;
52       }
53
54
55 //========= Final output.===================
56
57       System.out.println("Number error present : " + error);
58
59    } // end main
60 }
```