Week 8 – Webinar A

In this webinar session, we will look at some common techniques to validate user input. Although you won’t need any input validation for your assignment, you will be expected to be familiar with basic input validation techniques for your second year units.

# Learning Objectives

* Develop programs that feature robust input validation
* Implement static methods to validate user input

# Preparation

You will need to have **IntelliJ** open for this webinar. If you are using your own personal machine, you should download the **Community Edition** from [www.jetbrains.com/idea/download/](http://www.jetbrains.com/idea/download/) and install it. You are also free to use a similar IDE (e.g. Eclipse, VS Code) if you are familiar with one (but you shouldn’t use Processing).

If you cannot get IntelliJ working in time for the start of this webinar, you can use an online IDE such as <https://www.online-java.com/> for this webinar.

# Step 1 – Declaring a static method to serve as our validation function

We’ll start by creating a class named **ValidationDemo** with a main method, and then define a static method named **validateGrade** that is passed a scanner (the input stream the method will be working with) as a parameter and declare that this method will return an **int**. A grade (the integer the user will be typing in) should be a value between **0** and **100** inclusive.

public static int validateGrade(Scanner input)

Although we’ve mostly been passing primitives like integers as parameters to methods and constructors, we can have parameters which are objects (or, more specifically, a reference variable that points to an object).

# Step 2 – Validating an integer input is within a certain range

In the static method we’ve created, we’ll **prompt** the user for an integer and then **collect** it from the keyboard using the relevant Scanner method. We will then do some basic validation to see if the integer that the user has typed in is within a valid range – in our case, the integer types in will correspond to a grade for an assessment, so the valid values are between **0** and **100** inclusive.

If the input we’ve been given is not a valid integer (i.e. it is less than 0 or more than 100), then we should keep prompting the user to give us the correct input (Figure 1).

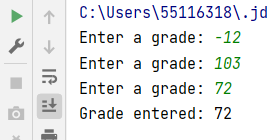


Figure Example Input Validation - Invalid Inputs are Rejected and User Re-enters Input

This input validation technique simply checks if the next integer is within a certain range, and if it is, then the while loop will not run. An issue with this, however, is that if a user gives us something which is not an integer, the program will crash (Figure 2).



Figure Invalid Input *type* Entered

# Step 3 – Validating an input stream has an integer in it

We’ve seen in the lecture and lab how a scanner can be used to *collect* input from an input stream, but the Scanner class also has methods to *check* what the next thing is in the input stream, without collecting it – **hasNextInt()** is just one of these helpful methods.

We can make use of this method like so: *“if the next thing in the input stream is an integer, fetch it and store it inside the grade variable. Otherwise, throw the next thing in the input stream away – since we’re not interested in it”*.

# Step 4 – Using the validateGrade to calculate the unit grade

Now that we have a method that returns an integer which is valid (i.e. is within a valid range and is definitely an integer), we can make use of it to calculate the grade for a student. We’ll assume a unit like this one, where the test is worth 50%, and the 2D game is worth the remaining 50%.

In the main method, rename the **grade** variable to **testGrade** (to represent the grade a student has gotten in the test). Create a second variable named **gameGrade**, which calls the **validateGrade** method as part of its assignment. Finally, calculate and print out the unit grade (the test and game are worth the same mark – 50% - so you can simply get the average of the two grades).

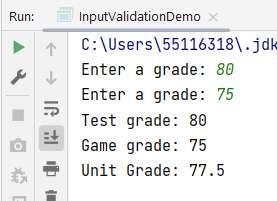


Figure Calculating Unit Grade (weighted average of testGrade and gameGrade)