P03 Exceptional Climbing

Overview

In your Climbing Tracker in P01, several situations arose where we had to handle bad input to methods (or assume that input would be valid), which typically involved returning special output. In reality, it's easier to indicate an error condition occurred by throwing an exception from the method rather than returning a result.

This week we'll be modifying a Climbing Tracker implementation to incorporate **exceptions**.

Grading Rubric

5 points	**Pre-assignment Quiz**: accessible through Canvas until 11:59PM on **09/27**.
20 points	**Immediate Automated Tests**: accessible by submission to Gradescope. You will receive feedback from these tests *before* the submission deadline and may make changes to your code in order to pass these tests. Passing all immediate automated tests does **not** guarantee full credit for the assignment.
15 points	**Additional Automated Tests**: these will also run on submission to Gradescope, but you will not receive feedback from these tests until after the submission deadline.
10 points	**Manual Grading Feedback**: TAs or graders will manually review your code, focusing on algorithms, use of programming constructs, and style/readability.

Learning Objectives

The goals of this assignment are:

- Practice working with exceptions, both throwing and catching.
- Learning to use an exception's message to convey additional information about the error circumstance.
- Learning to use different types of exceptions to convey information about the error circumstance, and ordering catch statements appropriately.
- Exploring the different requirements of checked and unchecked exceptions.

Additional Assignment Requirements and Notes

Keep in mind:

- You may import ONLY java.util.zip.DataFormatException.

- You are allowed to define any **local** variables you may need to implement the methods in this specification.

- You are NOT allowed to add constants or variables outside of methods.

- All methods, public or private, must have their own Javadoc-style method header comments in accordance with the CS 300 Course Style Guide. The text of these comments may be copied from this writeup without attribution.

- Any source code provided in this specification may be included verbatim in your program without attribution.

- ALL test methods must be **public static**, take **zero** arguments, and return a **boolean** value. These methods MUST be contained in the ExceptionalClimbingTester class.

- Be careful when copying text directly from the spec; occasionally some characters will not be compatible with Java. If you run into compilation issues, check the error messages carefully.

CS 300 Assignment Requirements

You are responsible for following the requirements listed on both of these pages on all CS 300 assignments, whether you've read them recently or not. Take a moment to review them if it's been a while:

- Academic Conduct Expectations and Advice, which addresses such questions as:
  - How much can you talk to your classmates?
  - How much can you look up on the internet?
  - What do I do about hardware problems?
  - and more!
- Course Style Guide, which addresses such questions as:
  - What should my source code look like?
  - How much should I comment?
  - and more!

Getting Started

1. Create a new project in Eclipse, called something like **P03 Exceptional Climbing**.
   a. Ensure this project uses Java 11. Select "JavaSE-11" under "Use an execution environment JRE" in the New Java Project dialog box.
   b. Do **not** create a project-specific package; use the default package.

2. Import the following Java source file into that project's src folder:
   a. ExceptionalClimbing.java

3. Create **one (1)** Java source file within that project's src folder:
   a. ExceptionalClimbingTester.java (includes a main method)

All methods in this program will be **static** methods, as this program focuses on procedural programming.

Complete the implementation of ExceptionalClimbing

Your methods from the ClimbingTracker program will now perform more complete input validation, and use exceptions rather than special return values to indicate when bad input was provided.

sendClimb() and failClimb()

The error conditions from P01 for sendClimb() were as follows:

> If there is no room to add a new climb OR the grade input is invalid, the return value will be the same as the initial value of `numSend/numFail`. Do not modify the array.

In ExceptionalClimbing, these errors should instead result in the following behaviors, tested in this order:

1. **Grade input is invalid**: throw an IllegalArgumentException, where the exception's message is `"[input] is not a valid grade"`, where [input] is the value provided. For example, if the provided grade is "V8", the exception's message would be `"V8 is not a valid grade"`.

2. **Provided array is full:** throw an IllegalArgumentException, where the exception's message is `"cannot add new value to full length [n] array"`, where [n] is the size of the input array. For example, if send has length 8 and numSend is 8, the exception's message would be `"cannot add new value to full length 8 array"`.

3. **NEW ERROR CASE**: provided array has null elements between index 0 (inclusive) and index `numSend/numFail` (exclusive), OR passed negative value for `numSend/numFail`. If you detect either case, throw a DataFormatException with the message `"invalid oversize array"`.

Be careful!! The DataFormatException is a *checked* exception and will require special handling.

getStats()

The error conditions from P01 for getStats() were as follows:

> If either `send` or `fail` are empty, or `historyLength` is 0 or negative, see page 7 for appropriate output.

In ExceptionalClimbing, these errors should instead result in the following behaviors, tested in this order:

1. **BOTH provided arrays are empty**: throw a RuntimeException, where the exception's message is `"no climbs provided"`. If *only one* of the arrays is empty, proceed as in ClimbingTracker.

2. **historyLength is 0 or negative:** throw an IllegalArgumentException, where the exception's message is `"[n] is not a valid history length"`. For example, if historyLength were -27, the exception's message would be `"-27 is not a valid history length"`.

You should NOT check the new error case for a DataFormatException in getStats().

getHistogram()

The error conditions from P01 for getHistogram() were as follows:

> If both arrays are empty, return an error message. See page 7 for an example.

In ExceptionalClimbing, this error should instead result in the following behavior:

1. **BOTH provided arrays are empty**: throw a RuntimeException, where the exception's message is `"no climbs provided"`.

You should NOT check the new error case for a DataFormatException in getHistogram().

ExceptionalClimbingTester

Your tester class for this iteration of the program can focus on error and edge cases for the program rather than correct functionality. For example, testSendClimb() should ensure that the **correct exception** with the **correct message** is thrown when grade input is invalid, and that it is NOT thrown when grade input is valid, but you do not need to check that the size has been updated correctly or that the contents of the array are correct.

Any exceptions that are NOT expected must be caught, and must cause the test to fail.

Some suggested scenarios:

- `testSendClimb()`/`testFailClimb()`:
    - Valid input causes no exceptions
    - Invalid grade causes IllegalArgumentException
    - Full array causes IllegalArgumentException
    - Array with null elements or invalid size causes DataFormatException
    - Combinations of these error conditions cause the *first* listed applicable exception from the description above (be sure to validate the message contents!)
- `testGetStats()`:
    - Valid input causes no exceptions (this includes *one* empty array)
    - Both arrays empty causes RuntimeException
    - Negative or 0 historyLength causes IllegalArgumentException
    - Combinations of these error conditions cause the *first* listed applicable exception from the description above (be sure to validate the message contents!)
- `testGetHistogram()`:
    - Valid input causes no exceptions (this includes one empty array)
    - Both arrays empty causes RuntimeException

These methods should be called from `runAllTests()`, which should be called in `main()`, as in P01.

Commenting

You will be graded on your code commenting for this assignment; be sure to include all comments per the style guide.

In particular, you should note in the ExceptionalClimbing class which exceptions are thrown by which methods using an `@throws` block tag for each exception type, and describe the situation that would lead to the exception being thrown.

Assignment Submission

Hooray, you've finished this CS 300 programming assignment!

Once you're satisfied with your work, both in terms of adherence to this specification and the academic conduct and style guide requirements, submit your source code through Gradescope.

For full credit, please submit ONLY the following files (source code, *not* .class files):

- ExceptionalClimbing.java
- ExceptionalClimbingTester.java

Your score for this assignment will be based on the submission marked "**active**" prior to the deadline. You may select which submission to mark active at any time, but by default this will be your most recent submission.

Copyright Notice

This assignment specification is the intellectual property of Mouna Ayari Ben Hadj Kacem, Hobbes LeGault, and the University of Wisconsin–Madison and may not be shared without express, written permission.

Additionally, students are not permitted to share source code for their CS 300 projects on any public site.