## Task 1- Building and Using Methods

For the first task, we're going to fill out a simple program by adding suitable methods to complete the functionality. The code will be spread over two classes, but some of the basics have already been provided.

The program consists of two classes, NumberSequence and Runner. The program starts by reading some simple numerical data (integers) in from a file (this part is provided) and storing this inside an array in an object of type NumberSequence. From this starting point, your task is to add methods and code to NumberSequence and Runner so that the program prints out the following (in this order, each on a separate line):

"There are n*n* numbers." where n*n* is the number of numbers in the sequence.

"The maximum is x*x*." where x*x* is the greatest number in the sequence.

"The minimum is y*y*." where y*y* is the least number in the sequence.

"The mean is a*a*." where a*a* is the arithmetic mean (what we normally call the average). Note that the mean may not be an integer (use double for decimals in this case).

One of the following three:

"The sequence is increasing." if each number is strictly larger than the previous.

"The sequence is decreasing." if each number is strictly smaller than the previous.

"The sequence is wobbly or short." if it has less than two numbers or neither of the other two apply.

Runner Java:

*public class Runner {*

*public static void main(String[] args) {*

*NumberSequence numbers = new NumberSequence(args[0]);*

*// If you plan your methods carefully, you should be able to*

*// just write a few relatively simple println statements here*

*// (that use numbers of course).*

*}*

*}*

Number Sequence .java

*import java.util.List;*

*import java.io.FileReader;*

*import java.io.BufferedReader;*

*import java.io.IOException;*

*import java.io.FileNotFoundException;*

*import java.util.stream.Collectors;*

*public class NumberSequence {*

*/\*\**

*\* The following section of code does not need to*

*\* be modified. The section to complete is indicated*

*\* below.*

*\*\*/*

*//The array the data will be stored in*

*private final int[] sequence;*

*//The constructor for the class*

*public NumberSequence(String filename) {*

*this.sequence = NumberSequence.readFile(filename);*

*}*

*//A helper method to read the data in from the file.*

*//See the example file for the format if you want to*

*//write your own tests.*

*private static int[] readFile(String filename) {*

*try (BufferedReader reader = new BufferedReader(new FileReader(filename))) {*

*List<String> rawData = reader.lines().collect(Collectors.toList());*

*int[] sequence = new int[rawData.size()];*

*for (int i = 0; i < rawData.size(); i++) {*

*sequence[i] = Integer.parseInt(rawData.get(i).trim());*

*}*

*return sequence;*

*}*

*catch (FileNotFoundException e) {*

*System.err.println("There is no file with the name: " + filename);*

*System.err.println(e.getMessage());*

*}*

*catch (IOException e) {*

*System.err.println("Something went wrong reading data from the the file " + filename +".");*

*System.err.println(e.getMessage());*

*}*

*return null;*

*}*

*/\*\**

*\* Write your code below here.*

*\*\*/*

*}*

Example.data

*1 1*

*2 2*

*3 3*

*4 4*

*5 5*

You are given some example test data, but you should not assume that all test data will follow this exact pattern (but you are guaranteed there is at least one number, and all numbers are integers).

You should not alter any of the existing code given in the scaffold, just add to it. **You may need to add code to both classes.**

## Task 2 - Triangle 2.0 (The name is a hint)

For the task we'll build a class representing a Rectangle. The tests for this task will not check what's happening inside the class you create, so it's up to you as to what data members you use. The tests will also assume you can write methods and constructors correctly. **This means that you have to get the names, returns and parameters correct before the tests will even run**.

Okay, now the task itself.

Your company is doing some consulting work and the client has asked for a new, more exciting version of their existing Triangle. They want a Triangle, but with an extra side, which marketing are calling *Triangle 2.0*.

To meet the specification you need to create a class called Rectangle with the following methods:

A no-parameter constructor that sets all side lengths to 1.

A one parameter constructor that takes an int, and sets all side lengths to that value.

A two parameter constructor, that takes two ints, and sets two opposite sides to one length, and the other pair of opposite sides to the other length.

A method getShortSide that takes no parameters and returns the length of the shortest side as an int.

A method getLongSide that takes no parameters and returns the length of the longest side as an int.

A method isSquare that takes no parameters and returns true if the rectangle is a square, and false otherwise.

A method area that takes no parameters and returns the area of the rectangle as an int.

A method diagonal that takes no parameters and returns the length of the diagonals as a double. The library function Math.sqrt(double) may be useful here.

A method bigger that takes a Rectangle as a parameter and returns true if the current Rectangle has an area strictly greater than the area of one given in the parameter.

You don't have to worry about not-right-angled quadrilaterals (so there's only squares and rectangles, no parallelograms, rhombi, trapezoids etc.).

You also start with nothing in the scaffold. You will have to create a suitable file for the class. The run button will assume that Rectangle has a main method, but it's not part of the tests, so you don't have to write it if you don't care about it.

## Task 3 - GOTY 1964

In this task, you have been lumped with the task of completing a half finished game left by a "rock-star" developer who can't be bothered doing anything more.

The game has simple rules:

There are two players, A and B.

A always goes first.

There are three piles of tokens, each starts with 20 tokens.

Each player takes turns removing *any* (valid) number of tokens from a *single* pile. The player has to take at least one token, and may not take more tokens than there are in the selected pile.

The player that takes the last token wins.

The logic for the game play has already been implemented in the GameLogic class, it is your task to complete the GameState class which will keep track of the current state of the game. GameState needs to record how many tokens are in each pile, which player has won (if any) and how many turns it took to get to the end of the game. It will also need to be able to represent the current state of the piles as a String. To do this you will need to:

Create the GameState class itself (the file GameState.java exists, but is empty).

Create a constructor that sets up the initial condition of the game. You are free to choose how to represent the piles, but it is a good idea to keep it simple. You will almost certainly need other variables to track relevant information.

Implement the following methods:

aWinner, which takes no parameters and returns true if player A has won the game, and false otherwise.

bWinner, which takes no parameters and returns true if player B has won the game, and false otherwise.

isValidMove which takes two int parameters, the first representing the pile with a number from 1 to 3, and the second representing the number of tokens to remove from that pile, and returns true if that move is possible (note you will have to check the validity of both parameters values).

moveA which takes two int parameters, the first representing the pile and the second the number of tokens to remove, and attributes the move to player A (think carefully about when it's important for GameState to know which player made the move). You may assume that the parameter values are valid for this method.

moveB which takes two int parameters, the first representing the pile and the second the number of tokens to remove, and attributes the move to player B. You may assume that the parameter values are valid for this method.

toString which takes no parameters and returns a String in the format "Current State: Pile 1 - x*x*, Pile 2 - y*y*, Pile 3 - z*z*." where x*x*, y*y* and z*z* are replaced with the number of tokens in piles 1, 2 and 3 respectively.

turns which takes no parameters, and returns the number of turns taken in the game so far.

**You do not need to change anything in GameLogic.java.**

The tests will directly test your GameState class, this has two immediate consequences:

The tests won't run until you have the skeleton of the class and methods (so put those in first with dummy return values where necessary).

You need all the described functionality to be in GameState - so using information in GameLogic won't solve the exercise (it would make it more complicated to do it that way anyway).

If your code is working, and you have not modified GameLogic, an example game would look something like (ignore any syntax highlighting you may see):

Current State: Pile 1 - 20, Pile 2 - 20, Pile 3 - 20. It's Player A's turn: 1 20 Current State: Pile 1 - 0, Pile 2 - 20, Pile 3 - 20. It's Player B's turn: 2 19 Current State: Pile 1 - 0, Pile 2 - 1, Pile 3 - 20. It's Player A's turn: 3 15 Current State: Pile 1 - 0, Pile 2 - 1, Pile 3 - 5. It's Player B's turn: 3 4 Current State: Pile 1 - 0, Pile 2 - 1, Pile 3 - 1. It's Player A's turn: 1 1 Invalid move. It's Player A's turn: 2 1 Current State: Pile 1 - 0, Pile 2 - 0, Pile 3 - 1. It's Player B's turn: 3 1 Current State: Pile 1 - 0, Pile 2 - 0, Pile 3 - 0. Game Over! Player B wins! The game took 6 turn(s).

For anyone who doesn't recognise it, this is a variant of the game [Nim](https://en.wikipedia.org/wiki/Nim" \t "https://edstem.org/au/courses/10153/lessons/28646/slides/_blank), which is more interesting in its less constrained version, but only marginally.

GameLogic.Java

*import java.util.Scanner;*

*public class GameLogic {*

*private static Scanner scanner = new Scanner(System.in);*

*public static void main(String args[]) {*

*GameState gameState = new GameState();*

*boolean AGo = true;*

*while (!gameState.aWinner() && !gameState.bWinner()) {*

*boolean moved = false;*

*System.out.println(gameState);*

*do {*

*System.out.print("It's Player " + (AGo? "A" : "B") + "'s turn: ");*

*int pile = scanner.nextInt();*

*int count = scanner.nextInt();*

*if (gameState.isValidMove(pile, count)) {*

*if (AGo) gameState.moveA(pile, count);*

*else gameState.moveB(pile, count);*

*moved = true;*

*}*

*else {*

*System.out.println("Invalid move.");*

*}*

*}*

*while (!moved);*

*AGo = !AGo;*

*}*

*System.out.println(gameState);*

*System.out.println("Game Over!");*

*if (gameState.aWinner()) {*

*System.out.println("Player A wins!");*

*}*

*else {*

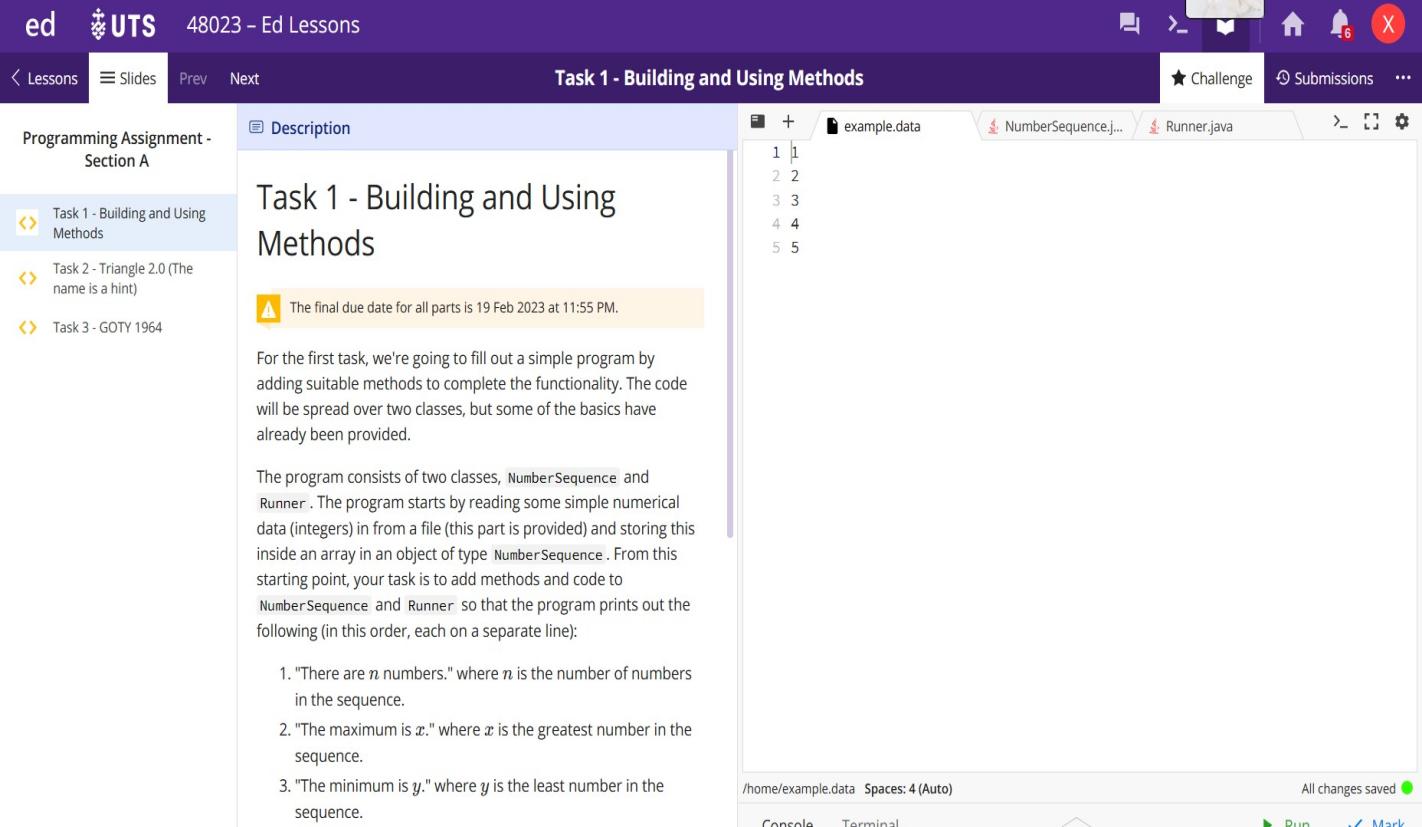
*System.out.println("Player B wins!");*

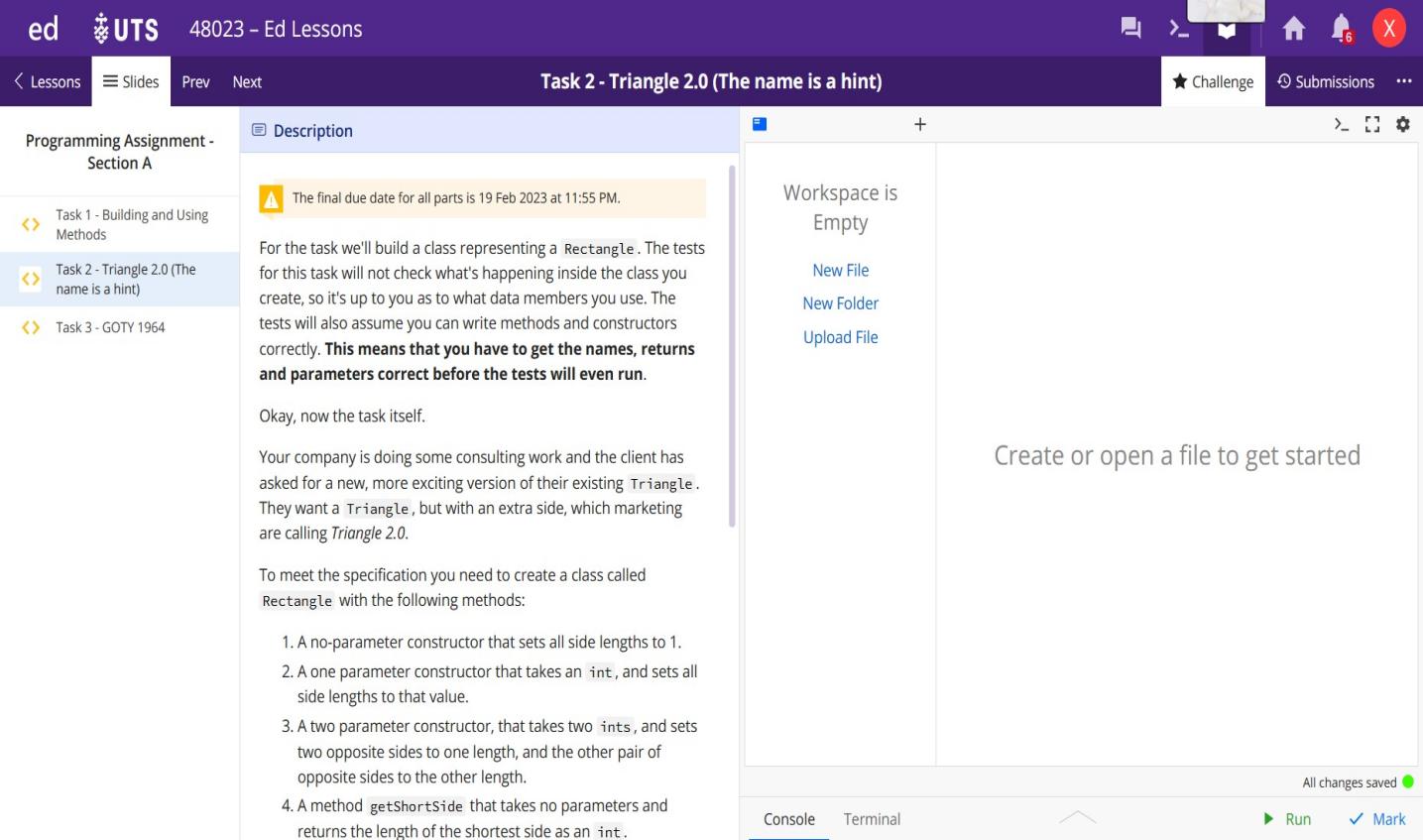
*}*

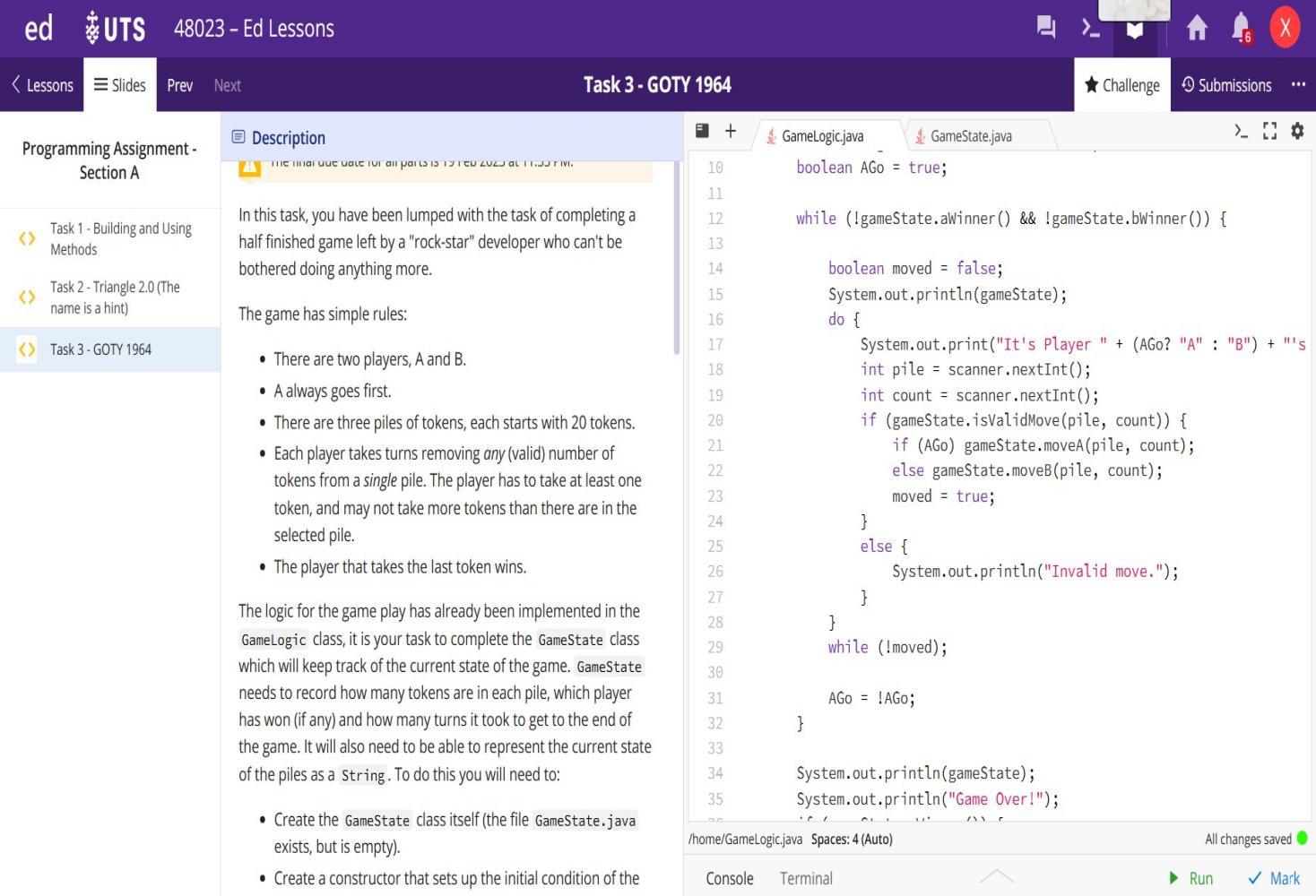
*System.out.println("The game took " + gameState.turns() + " turn(s).");*

*}*

*}*

**

**

**