## Assignment - Section B Part 1 - A Simple Set

## A Simple Set Data Structure

To build a Set from scratch, first we must create the universe. Luckily, we can build it in Java, which saves a lot of time.

For this part, we will build a SimpleSet class that implements the basic operations necessary for a functioning data structure. To keep things simple, we're only going to build sets that hold ints. To this end you need to create a class called SimpleSet with the following:

A private array of ints called data that will hold the actual numbers.

It will also need the following methods:

A no-parameter constructor that initialises the array, and any other variables you might have.

A method called add that takes a single int parameter, and puts it in data if it's not there, but does nothing if it's already there.

A method called remove that takes a single int parameter that removes that value if it's in data, but does nothing if it's not there.

A method call contains that takes a single int and returns true if that element is in the set, and false otherwise.

A method called size that returns the number of elements in the set.

A method called isEmpty that returns true if there are no elements in the set, and false otherwise.

A method called toArray that returns an array containing exactly the elements of the set.

A toString method that returns a String representation of the SimpleSet in the format "{x, y, z, ...}" where x, y, z, ... are replaced with the actual elements of the set.

When implementing these, you may want to think about the following questions:

How big should data be?

What happens when data is full and we add a new element?

What happens to the elements stored in data when we remove one?

Is the size of the set the same as the length of data?

You have seen answers to these questions throughout the workshops and lectures, so think about where we might have done similar things before. Note that there is more than one right answer, the point here is for you to think about what you're trying to do, and pick a consistent approach.

Note that there is a main method provided in the Runner class, but this does not form part of the tests, it's just there so you can run your code and test it manually.

Finally, but not least, as the point here is for you to implement your own data structure an learn about how these things actually work:

右侧是Runner.Java

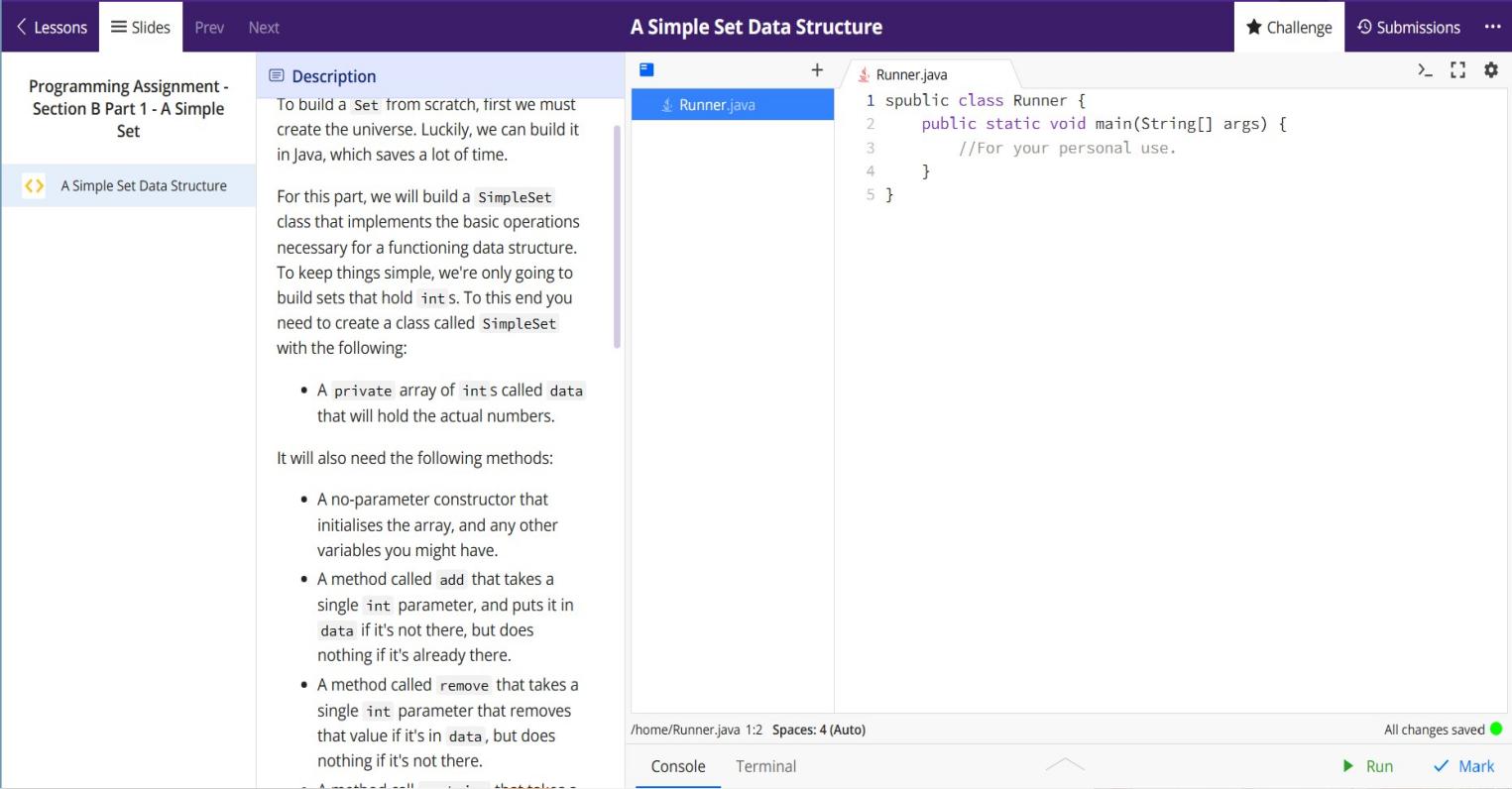
*1public class Runner {*

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

*3 //For your personal use.*

*4 }*

*5 }*



## Assignment - Section B Part 2 - Working With Sets

## Building Some Operations to Manipulate Sets

Now that we have a working SimpleSet class, we can start writing code that works *with* sets. If you are familiar with mathematical sets, these operations should also be familiar - they're the standard things you can do with sets.

For this task you need to create a class called SimpleSetOperations, which will have the following methods:

union, which takes two SimpleSets as parameters, and produces a new SimpleSet that contains all the elements in are in at least one of the sets.

intersection, which takes two SimpleSets as parameters, and produces a new SimpleSet that contains only the elements that are in both starting sets.

difference, which takes two SimpleSets as parameters, and produces a new SimpleSet that contains only the elements that are in one but not both starting sets.

setMinus, which takes two SimpleSets as parameters, and produces a new SimpleSet that contains only the elements that are in the first set and *not* in the second.

subset, which takes two SimpleSets as parameters, and returns true if the first is a subset of the second, and false otherwise.

properSubset, which takes two SimpleSets as parameters and returns true if the first is a *proper* subset of the second, and false otherwise. (A set A*A* is a proper subset of B*B* if it is a subset, and they're not the same).

equals, which takes two SimpleSets as parameters and returns true if they contain the same elements, and false otherwise.

If you are not clear on these, ask on the discussion board ([or read the source of all true knowledge](https://en.wikipedia.org/wiki/Set_(mathematics)" \t "https://edstem.org/au/courses/10153/lessons/28648/slides/_blank)).

These methods should all be public and static. You may assume that null will never be passed as an argument to any of the parameters.

You have been provided with a working SimpleSet implementation, so you don't need to copy your solution over, but you're welcome to if you like.

As with Part 1, you are provided with a main method in the Runner class, but it's not part of the tests, it's there for you to run your code to test it yourself.

As before:

#### 右侧是两个文件 Runner.Java/Simpleset.Java

*Runner.Java*

*1public class Runner {*

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

*3 //Test away!*

*4 }*

*5 }*

Simpleset.Java

*public class SimpleSet {*

*private int[] data;*

*private int size;*

*public SimpleSet() {*

*this.data = new int[10];*

*this.size = 0;*

*}*

*public void add(int element) {*

*if (!contains(element)) {*

*if (size >= data.length) {*

*resizeData();*

*}*

*data[size++] = element;*

*}*

*}*

*public boolean contains(int element) {*

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

*if (data[i] == element) return true;*

*}*

*return false;*

*}*

*public boolean isEmpty() {*

*return size == 0;*

*}*

*public int size() {*

*return size;*

*}*

*public void remove(int element) {*

*int position = findPosition(element);*

*if (position >= 0) {*

*for (int i = position; i < size - 1; i++) {*

*data[i] = data[i + 1];*

*}*

*size--;*

*}*

*}*

*public int[] toArray() {*

*int[] set = new int[size];*

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

*set[i] = data[i];*

*}*

*return set;*

*}*

*@Override*

*public String toString() {*

*String output = "{";*

*for (int i = 0; i < size - 1; i++) output += data[i] + ", ";*

*if (size > 0) output += data[size - 1];*

*output += "}";*

*return output;*

*}*

*private void resizeData() {*

*int[] newData = new int[this.data.length\*2];*

*for (int i = 0; i < this.data.length; i++) {*

*newData[i] = this.data[i];*

*}*

*this.data = newData;*

*}*

*private int findPosition(int element) {*

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

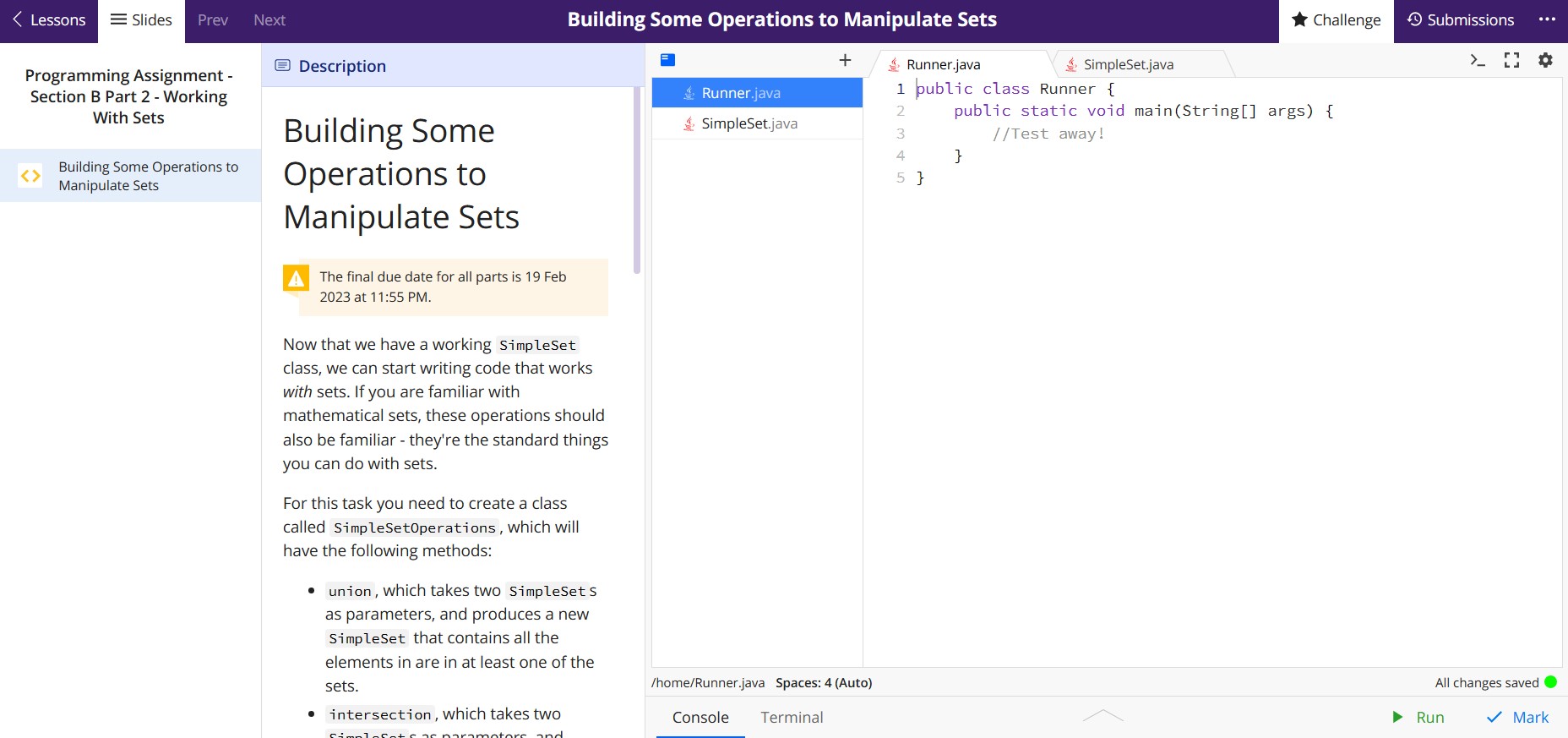
*if (data[i] == element) return i;*

*}*

*return -1;*

*}*

*}*

**

## Assignment - Section B Part 3 - Advanced Operations

## Doing Things To Sets

For the final part of the assignment, we're going to take a step somewhat sideways. The methods we're going to write in this part are what are called "Higher Order Functions" because they take as one of the parameters *another* function and the result of the method is the result of applying the other function to the supplied set (in the correct way of course).

As Java is strongly object oriented, it doesn't quite support just defining a parameter to be a different method. The way Java gets around this is to define interfaces that specify a single method that has the right properties. Then each function passed as an argument to the parameter is instantiated as an object that implements that interface, either explicitly, or if you're being fancy, as an instance of an anonymous class.

To complete this part, you do not need to understand any of that! Practically what we have is that all the methods have a parameter whose type is a class with and apply method, so you take the object passed in to that parameter (say f, and call f.apply(...) where the ... is replaced by whatever the suitable parameters are for that function.

For this part, your task is to complete the six methods in the HigherSetOperations class. To help you have three predefined "function" types, defined by the the interfaces UnaryFunction, BinaryFunction and BooleanFunction.

UnaryFunction represents a function that takes an int and produces an int.

BinaryFunction represents a function that takes two ints and produces an int.

BooleanFunction represents a function that takes an int and produces a boolean.

The six method you will work on are (examples are described below):

map, which takes a SimpleSet and UnaryFunction and returns a new SimpleSet where every element has had the function represented by the UnaryFunction applied to it.

fold, which takes a SimpleSet, a BinaryFunction and an int and returns an int that is the result of progressively applying the BinaryFunction to a running value (starting with int parameter) each element of the set successively.

filter, which takes a SimpleSet and a BooleanFunction and returns a new SimpleSet which consists of the elements of the starting set that the function returns true on.

all, which takes a SimpleSet and a BooleanFunction and returns true if the function returns true for every element of the set and false otherwise.

none, which takes a SimpleSet and a BooleanFunction and returns true if the function returns false for every element of the set and false otherwise.

any, which takes a SimpleSet and a BooleanFunction and returns true if the function returns true for any element of the set and false otherwise.

To illustrate these, consider the starting set set = {1, 2, 3, 4, 5}, the UnaryFunction PlusOne whose apply(int) method adds one to its argument, the BinaryFunction Plus whose apply(int, int) method adds its arguments together, and the BooleanFunction IsEven whose apply(int) method returns true if the argument is even and false if it's odd.

Then:

map(set, new PlusOne()) should produce {2, 3, 4, 5, 6},

fold(set, new Plus(), 0) should produce 15,

filter(set, new IsEven()) should produce {2, 4},

all(set, new IsEven()) should produce false,

none(set, new IsEven()) should produce false, and

any(set, new IsEven()) should produce true.

To help (hopefully!) you are provided with the implementations of these three functions so you can test your code, and if you're feeling adventurous, write your own!

You are also provided with working versions of SimpleSet and SimpleSetOperations, so you don't bring your own over. The version of SimpleSet is also a more advanced version which supports iteration and the for-each construct (so you can do the for(int element : set) style for loop. It also has a second constructor, that allows expressions like new SimpleSet(1,2,3,4,5), which simplifies creating some small sets.

The final code for this part shouldn't turn out to be too complicated - the challenge here is understanding how to interrelate different parts of the code to produce a functional whole. Think carefully about the elements you already have that are spread over the other classes.

You do not need to modify anything other than the HigherSetOperations class.

You may assume that all test inputs are non-null and valid.

右侧是

Binary Function.Java/BooleanFunction.Java/HigherSetOperations.Java/IsEven.Java/Plus.Java/ PlusOne.Java/Runner.Java/SimpleSet.Java/SimpleSetOperations.Java/UnaryFunction.Java

