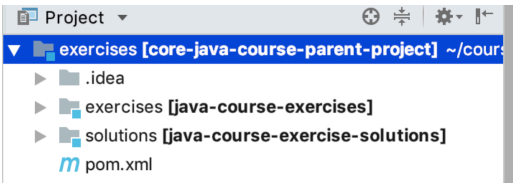
Exercise Setup and Structure

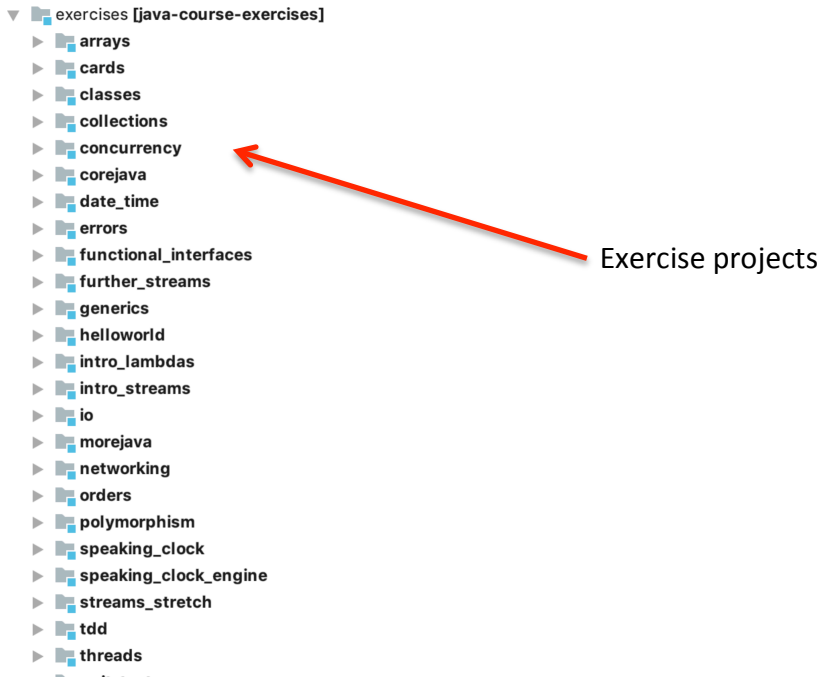
Overview

The exercise and solution files are provided for you. With the zip file, download it and extract it in a location you wish the files to reside in. The projects are all maven projects and have dependencies configured for you.

There are separate folders for the exercise projects where you will work and the solutions projects as shown below.



You will do your work in the java-course-exercises section. There are projects define for each exercise there as shown below.



There are corresponding solution folders where you can look for tips and compare your solutions.

Exercise 1: Hello World

Overview
This exercise enables you to write your first Java program using your IDE.

1. Start your IDE.
2. Open the project helloworld.
3. Add a new class named Driver to the project and add a main method to the class.
4. In the main method, output a message of your choice to the console.
5. Run your program.

This is the end of the exercise

Exercise 2: Simple Maths

Overview

In this exercise you will work with creating variables and using the arithmetic operators.

Core Exercise

1. In your IDE open the project named corejava
2. Add a new class named SimpleMath to your project.
3. In a main method create 4 variables of type double named a, b, c, and d. Initialize these variables to values of your choice.
4. Calculate the following values in main and display the results to the console:

```
a+b
d-c
c/a
b*d
a%d
```

6. Run your program and verify that the results are as expected.

Additional Exercise (If Time Permits)

7. Calculate the average of the 4 variables you have created and display this value to the console.
8. Calculate the value of a+b divided by c – d. Display the result to the console. Run your program and confirm it outputs the correct result.

This is the end of the exercise

Exercise 3: Making Decisions

Overview

In this exercise, you will gain experience of making decisions in Java.

Core Exercise

1. In your IDE, add a new class to the project corejava named Decisions. Make sure the class Decisions has a main method.

2. In the main method, prompt the user to enter a number between one and 100. If the number is not in the above range, you should tell them via an error message and exit the program.

3. Modify the program so that it prints the above number and the word odd or even depending on whether the number is odd or even.

4. Continuing from above, add a second prompt to the user to enter a number from 1 to 9 as a digit. Then output the number as a word; i.e., 1-> one, 2->two, etc.

Additional Exercise (If Time Permits)

5. Extend the above program so it prompts the user for two values and prints two values to the console and highlights which is the larger of the two numbers.

This is the end of the exercise

Exercise 4: Iteration

Overview

In this exercise, you will gain experience of performing iteration in Java.

Core Exercise

1. Add a new class to the project corejava named Iteration. Make sure Iteration has a main method.
2. In the main method write a program so it prompts the user for two integer values, a minimum and maximum value (between 1 and 100) and then prints all the integer numbers (in increments of 1) between the minimum and maximum entered by the user. If the numbers entered are not between 1 and 100 display an error message and exit the program.
3. Modify the above program so it only prints the odd numbers in the range also.

Additional Exercise (If Time Permits)

4. Write a program that prints out the multiplication table for all numbers 1 to 10. The multiplication should consider all values up to and including 12.
5. Write a program that prompts the user to enter 3 integer numbers and then outputs these numbers in ascending order to the console.

This is the end of the exercise

Exercise 5: Working with Arrays

Overview

In this exercise, you will gain experience of working with arrays in Java.

Core Exercise

1. Open the project named arrays.
2. Define a class named Arrays with a main method.
3. In main write a program that prompts the user for a number of students. The program should then prompt the user for a single grade each of the students has achieved and store them in an array. Print out all the grades entered and also the average grade entered.
4. Modify the above program to also output the lowest and highest values entered.

Additional Exercise (If Time Permits)

5. Define a new class named Operators in the project arrays. In the class Operators write a program that receives 3 command line arguments – two numbers and the third a character which must be one of +-/. Output the result of applying the third character operator to the two numbers to the user.
6. Modify the program above that prompts the user for the grades of a student so that each student has three grades. Read in the student grades and then store them in a two dimensional array. Print out the grades for each student.

This is the end of the exercise

Exercise: Working with Strings

Overview

In this exercise, you will gain experience of working with String objects in Java.

Core Exercise

1. Open the project named classes.
2. Define a class named Strings with a main method **.**
3. In main write a program that prompts the user for a String and prints the reverse of the String. You should do this by writing code that treats the String as an array of char's and iterates over the array.
4. On your phone keypad, the alphabet is mapped to digits as follows : ABC(2), DEF(3), GHI(4), JKL(5), MNO(6), PQRS(7), TUV(8), WXYZ(9). Modify main above so that it prompts the user for an alphabetic string and converts it to a sequence of digits. E.g. AGRZ would be converted to 2479.

Additional Exercise (If Time Permits)

5. A word that reads the same forward and backward is called a palindrome. Write a program that prompts the user for a word and then outputs whether the word is a palindrome or not. Test your program with words such as Madam, dad, mum, racecar, radar.
6. Write a program named Bin2Dec to convert an input binary string to its equivalent decimal value. So 1011 would be 11, 1111 would be 15 etc. You may wish to use the Math.pow() method.

This is the end of the exercise

Exercise: Defining Classes

Overview

In this exercise, you will gain experience of defining classes and creating objects in Java.

Core Exercise

1. In the project classes define a new class named calculator. The class should have four methods, add, subtract, divide and multiply. Each method should take two integer arguments and return an integer result.
2. Define a class named CalculatorDriver with a main method
3. In main write a program that creates a Calculator object and calls the four methods of the Calculator class. Print the results to the console to verify that they work as expected.

Additional Exercise (If Time Permits)

4. Extend the Calculator class so that it can also add, subtract, multiply and divide floating-point values. Verify that the methods work.

This is the end of the exercise

Exercise: Working with Constructors

Overview

In this exercise, you will gain experience of defining multiple constructors in a Java class.

Core Exercise

1. Open the project named orders.
2. In the project define a class named Order. The class should have a field for the stock symbol, the currency and the amount the order is for.
3. Add to your class a constructor that takes 3 parameters to initialize the fields.
4. Add to your class a method named `toString()` that returns a String containing the three field values.
5. Write a Driver class that creates an Order object, using the above constructor.
6. Print to the console the return value of the Order's `toString()` method.
7. Add a second constructor to Order. This constructor should take a stock symbol only and set the amount to zero and the currency to USD. Use the constructor defined in step 3 in your implementation of the new constructor.
8. In main create a second Order passing to the constructor a symbol of "TWTR" only and print the value returned from its `toString()` method of this second order.

Additional Exercise (If Time Permits)

9. Add a default constructor to the Order class and set the stock to the string "UNDEFINED" and the amount to zero and the currency to USD.
10. In main create a third Order using the default constructor and print the value returned from its `toString()` method to the console.

This is the end of the exercise

Exercise: More Classes

Overview

In this exercise, you will gain experience of using static data members and static methods.

Core Exercise

1. You will work with the Order class in the project orders from the previous exercise. Add a static field to the Order class that can be used to keep track of the number of Order objects created.
2. To the Order class add a static method named `getNumberOfOrders()`.
3. Modify the code in the Order class so that every time an order is created the count of number of orders created is incremented by one.
4. Print out the number of orders at the beginning of the main method and also at the end. Do the number of orders displayed reflect the number you actually created?

Additional Exercise (If Time Permits)

5. Modify the Order class so that the number of orders is decremented when the garbage collector removes an order object.

This is the end of the exercise

Exercise: Inheritance

Overview

In this exercise, you will gain experience of working with inheritance and polymorphism. You will define two subclasses of Order, one a MarketOrder, the other a LimitOrder. Each will have its own specific implementation of a match method.

Core Exercise

1. You will continue working on the project named orders. To the Order class add an abstract method with the following signature:

   ```
   public boolean match(Order order);
   ```

2. Now define a new class named MarketOrder which extends Order. In MarketOrder implement the match method and in the implementation just output a string to the console that says "in MarketOrder match".
3. Implement the appropriate constructors for MarketOrder.
4. In the main method, comment out the code that created Order objects. Create an array of two Order references and add two MarketOrders to the array.
5. Iterate over your array of orders calling the match method. You will need an Order to pass to the match method—which order can you pass to it?
6. Verify that your code runs as expected.
7. Define a second class named `LimitOrder` which extends Order. In `LimitOrder` implement the match method and in the implementation just output a string to the console that says "in LimitOrder match".
8. Extend the array of orders in the main method to now be of size 3 and in the third element add a reference to a `LimitOrder`.
9. Run the program and verify that it behaves as expected.

Additional Exercise (If Time Permits)

10. Modify the Order class so that it only has one constructor that takes three parameters for the currency, amount, and stock symbol. What effect does this have on the `MarketOrder` and `LimitOrder` classes?
11. Correct your code and run it and verify it works as expected.

This is the end of the exercise

Exercise: Interfaces

Overview

In this exercise, you will gain experience of working with interfaces.

Core Exercise

1. You will continue working on the project named orders. In the project define an interface named Transferable. The interface should have one method with the following signature:

   ```
   void transfer(Broker targetExchange);
   ```

   You will need to define a Broker class for this—which can just be an empty class for now.

2. Modify the `LimitOrder` class to make it implement Transferable.
3. In the main method of the class Driver in this project, add a loop that iterates over the array of orders and outputs only those that are Transferable.
4. Run your code and verify it works as expected.

This is the end of the exercise

Exercise: More Java

Overview

In this exercise, you will gain experience of using variable length arguments and enumerations,

Core Exercise

1. Open the project named morejava.
2. In the project create a new class named Math. In this class provide static methods to add a variable number of integers and doubles.
3. Create a new class named Driver in the morejava project with a main method that uses the two methods defined above and verifies they work as expected.

Additional Exercise (If Time Permits)

4. Create a new java project named cards. The following instructions should add code to this new project.
5. Define an enumeration named Suit that represents the suits of a set of playing cards (hearts, diamonds, club and spades). Define a second enumeration that represents the rank of a playing card from 2 to ace.
6. Define a class named Card that represents an instance of a playing card. It should have a rank and suit and a toString() method that returns its rank and suit.
7. Define a class named Deck that represents a deck of 52 playing cards. The constructor should create and initialize the 52 cards. Implement a `toString()` method that returns the full 52 cards rank and suit. Verify this works in a main method

Additional Exercise (If Time Permits)

8. Provide a shuffle method that will shuffle your card in the deck. Using the `toString()` method verify the cards have been shuffled.
9. Provide a `deal()` method that will deal the next available card in the deck.

This is the end of the exercise

Exercise: Working with Exceptions

Overview

In this exercise, you will gain experience of working with Java exceptions, including defining a custom exception class, throwing exceptions and handling exceptions.

Core Exercise

1. Open the project named errors.
2. Define a class named Calculator with four methods, add, subtract, divide and multiply. These should work with two integer values. You can copy the one from an earlier exercise.
3. Define a class named Driver and verify that your Calculator works as expected.
4. Define an Exception class named CalculatorException that extends Exception. The constructor should take an error message string.
5. In your calculator class restrict it to work only with integers in the range -100 to 100 and to throw a CalculatorException if data outside this range is passed into any of the methods. Modify your main method to handle the exception and print the exception message and its stack trace to the console.

Additional Exercise (If Time Permits)

6. Define an Exception class named CalculatorDivideByZeroException that extends Java's RuntimeException. The constructor should take an error message string.
7. Modify your calculator to throw the CalculatorDivideByZeroException when appropriate. Are any changes required in the main method that uses the calculator? Why or why not?
8. What happens if your code generates a CalculatorDivideByZeroException? How can this be avoided/improved?

This is the end of the exercise

Exercise: Working with Java Input and Output Classes

Overview

In this exercise, you will gain experience of working with the Java classes for reading and writing text files.

Core Exercise

1. Open the project named io.
2. Write a program that will read lines of text from the console and write them to a file whose name is also entered from the command line. Your program should use Readers and Writers.
3. Verify your program works as expected.

Additional Exercise (If Time Permits)

4. Write a new program that will open your file created above and add your name to the bottom of the file

This is the end of the exercise

Exercise: Working with Java Collections

Overview
In this exercise, you will gain experience of working with the core Java collection classes.

Core Exercise

1. Open the project named collections.
2. Copy your Order, `LimitOrder` and `MarketOrder` classes from the earlier exercise.
3. Define a class named Driver with a main method and in main create an ArrayList of Order's. Populate the array list with 5 different `MarketOrder`/`LimitOrder` objects.
4. Iterate over the array list of orders's and display the result of their toString() methods to the console.
5. Verify the code works as expected.
6. Using the Collections class `sort()` method, sort your array list of orders and iterate over the sorted array list printing the result of `toString()` to the console. You should sort them on amount in descending order. Hint: You need to implement Comparable or Comparator.

Additional Exercise (If Time Permits)

7. In main create a HashSet and add your 5 Order objects to it.
8. Iterate over the HashSet and output the result of each Orders's `toString()` to the console. Is the result as you expected?
9. Which implementation of Set will sort the Orders in order? Create one of these and verify it works.
10. Create a HashMap of Order objects. What can you use as a key for the order? Consider adding an id field to the Order class and set this internally in a way that each order has a unique id. Add your 5 orders to the HashMap.
11. Iterate over the HashMap and output the result of each Orders's `toString()` to the console. Is the result as you expected?
12. Which implementation of Map will sort the Orders in order? Create one of these and verify it works.

This is the end of the exercise

Exercise: Working with Generics

Overview

In this exercise, you will gain experience of working with the generics.

Core Exercise

1. Open the project named generics.
2. Define a class named MyArrayList that is a generic class and the type of items it works with are supplied when the variable is defined using <Type>. Your class should only store 10 items maximum in it.
3. Add methods add(T), T get(index) and remove(index) to your arraylist.
4. Define a new class named Driver and write some code to verify your generic class above works.

Additional Exercise (If Time Permits)

5. Modify your class above so that it implements the Iterable<T> interface.
6. In Driver, use the for each loop to iterate over your array list to verify the iterator implementation works.

This is the end of the exercise

Exercise: Working with Java Threads

Overview
In this exercise, you will gain experience with Java threads.

Core Exercise

1. In your IDE, open the project named threads.
2. Develop a class that can run as a separate thread of execution and will output the numbers start to end with x second delay between each output. Start, end and x should be input from the keyboard.
3. Develop a class named Driver that has a main method and will create an instance of your class from step 2 and run it as a separate thread.

Additional Exercise (If Time Permits)

4. Your task is to create two classes that can each run as separate threads. One class will be a producer and the second will be a consumer. The producer class will:
    a. Read lines of text from the keyboard and place them in a buffer.
    b. Notify any consumer threads that there is a line of text ready in the buffer

    Your second class will be a consumer that will wait for a producer to produce a line of text. It will wait until it is notified of the text then print it to the console.

    You can structure this code however you wish but you should use the java wait() and notify() methods.
5. You should create a Producer and Consumer object and run them as separate threads and verify they work as expected.

This is the end of the exercise

Exercise: Working with Java Concurrency

Overview

In this exercise, you will gain experience of working with the core Java concurrency classes. To introduce working with core Java concurrency classes. You will use the completion service to run a number of callables that will count the number of characters in a line of text. The overall program will return the number of characters in a file.

Core Exercise

1. Open the project named concurrency.
2. Write a class that implements Callable<Integer>. The class receives a String argument to its constructor and in the call method calculates and returns the number of characters in the string.
3. Create a class named CharacterCount that has a main method. In main, create an ExecutorService that will execute your Callables.
4. In main open a text file whose name is passed as a command line argument.
5. Read the file line by line, and for each line create a callable passing it the line read from the file.
6. When all the lines of text have been submitted from the file, use the completion service to gather the results passed back by the callables and sum them together. This is the number of characters in the file. Print this number to the console.

Additional Exercise (if time permits)

7. Modify the above program so that it receives two command line arguments – the name of the text file and either c or w. If c is received your program should count the number of characters in the file, if w then it should count and print the number of words in the file.

This is the end of the exercise

Exercise: Working with Java Networking

Overview

In this exercise, you will gain experience of working with Java networking classes. You will write a concurrent network server.

Core Exercise

1. In your IDE open the project networking.
2. Your first task is to write a date/time Server that can handle multiple connected clients and sends them the date/time as a String every second.
3. Your second task is to write a client that can connect to the date/time server and will display the date/time it receives to the console.
4. Run your server and a single client and verifies it works.
5. Run multiple clients simultaneously and verify they work as expected.
6. Modify the client/server so that the client can send a user selected repeat time/interval at which the sever can send the date/time. Each client can have its own period which should not be less than a second and should be in multiples of seconds.

Additional Exercise (if time permits)

7. Re-implement your date/time server using UDP. Is this a more appropriate solution than a connection based one for the type of data being sent?

This is the end of the exercise