# ENSF 409
# Winter Semester 2021
# Assignment 03

## Assignment Instructions

Complete and submit exercise 8.5 (Lesson 08). This assignment is due on February 1, 2021 by 5:59 PM Mountain Time.

# General Instructions

## Academic Integrity and Collaboration

This is an individual assignment. Your submission must be your own original work. You may not work with others to complete the assignment, although you may collaborate on exercises which are not submitted. It is not academic misconduct to ask for help on the discussion boards, but you may not copy code or complete answers from your peers.

## Submission

You must submit your assignment in the specified format. Due to the number of students in the course, we are using automated grading. If your submission does not have the correct folder structure and file names, it will not be marked. This is a strict requirement.

File and directory names are case-sensitive.

You must submit your assignment to the appropriate D2L dropbox folder. You may submit multiple times before the assignment deadline, but only the last submission will be graded. Previous uploads are discarded by D2L. Therefore, each upload must be a complete submission. Do not submit multiple files across separate submissions.

The assignment must be submitted as a single zip folder named with your student ID. Furthermore, when the file is unzipped, the contents must be in a folder with the same name. You may wish to verify that your zip file has been correctly generated (includes the student ID directory, contains all files). You can do this by copying the zip file into another folder, unzip it, and examine the structure.

Within the folder with your ID number, you must create a subdirectory for each exercise within the assignment. Use lowercase and employ a _ to separate sub exercise numbers. For example, Exercise 1.3 should be in a folder `exercise1_3`.

Below is an example for submitting Assignment 1, assuming a student ID of *1234765*.

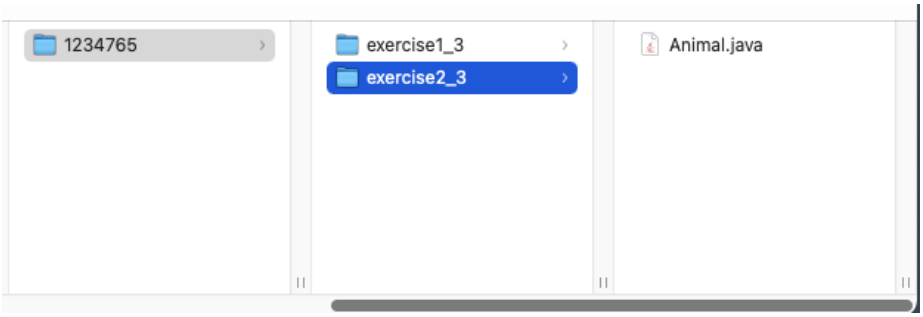


Figure 1: The folder structure for Assignment 1

```
ENSF409 > ls
1234765
ENSF409 > zip -r 1234765.zip 1234765
  adding: 1234765/ (stored 0%)
  adding: 1234765/exercise2_3/ (stored 0%)
  adding: 1234765/exercise2_3/Animal.java (stored 0%)
  adding: 1234765/exercise1_3/ (stored 0%)
  adding: 1234765/exercise1_3/Hello.java (stored 0%)
ENSF409 > ls
1234765         1234765.zip
ENSF409 >
```

Figure 2: Creating the zip file in Mac/Linux

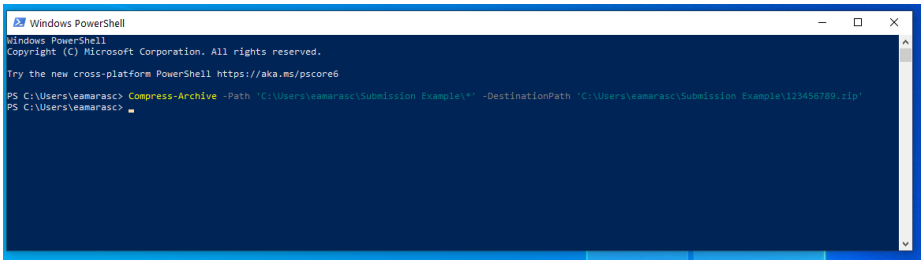


Figure 3: Creating the zip file in Windows

## Evaluation

Code which does not compile will not receive any points. Your code must compile and execute correctly to receive full marks. Assignments are graded using OpenJDK 11. While we do not anticipate that any of the assignments in this course will execute differently in any version of Java from 8 on, you should use this version for optimal results.

When style conventions are introduced in lessons, all subsequent submissions should adhere to the conventions.

## Deadline

All homework assignments are due at 17:59 (5:59 PM) Mountain Time. You are responsible for the conversion between Mountain Time and your local time zone, if applicable. Be aware that the switch from Mountain Standard Time to Mountain Daylight Time occurs during the term.

It is recommended that you do not leave your submission until the last minute in case of technical issues. Once the dropbox folder closes, you will not be able to submit your assignment. Late submissions will not be accepted.

If there is a technical problem and you are unable to submit via the dropbox, do not email your assignment. The University of Calgary email system will remove zip files. Instead, you may email a link to download your .zip folder from OneDrive. The email must be received by Dr. Barcomb and Dr. Marasco before the assignment deadline and the OneDrive folder should not show any changes to the .zip file after the deadline has passed.

# ENSF 409

Exercises - Lesson 6

The following exercises are described in the video for Lesson 6.

## Exercise 6.1

1. Create a recursive method which calculates the number of possible permutations without replacement. Permutations are calculated for the values contained in a String array.
2. The method should take two variables, the total number of elements in the array, and the number of items to be selected.
3. Include error handling if the number of items to be selected is negative, or greater than the array length.

- To calculate permutations without replacement, multiply the total number of items by the total number of items minus one, and repeat the process until the specified number of items is reached.
  - Example: If there are 5 items, and we want to select 4 of them, the number of possible permutations is 5 * 4 * 3 * 2
  - Example: If there are 6 items, and we want to select 3 of them, the number of possible permutations is 6 * 5 * 4
- Example input and return (1):
  - `String[] array = { "Apple", "Cabbage", "Lettuce", "Orange", "Radish"}`
  - `permutations(array.length, 3)` returns 60
- Example input and return (2):
  - `String[] array = {"Water", "Milk", "Soda", "Tea"}`
  - `permutations(array.length, 4)` returns 24

# ENSF 409
Exercises - Lesson 7

The following exercises are described in the video for Lesson 7.

## Exercise 7.1

1. Review the documentation for Boolean, Character, Integer, Long, Float, Double, Short, and Byte.

> To get started, here is the link to Character:
> `https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Character.html`

# ENSF 409
Exercises - Lesson 8

The following exercises are described in the video for Lesson 8.

## Exercise 8.1

1. Open the course git repository and find the example program called `ObjectCopying.java` in the setup folder of Lesson 8.
2. Create a second horse. Assign it a few skills and a rider.

```
...
import java.util.Arrays;

public class ObjectCopying {

    public static void main(String[] args){

        Animal horse1 = new Animal("Horse","Equidae","Chordata","Blaze");
        String[] horse1Skills = new String[]{"Jumping", "Dressage"};
        horse1.setAnimalSkills(horse1Skills);
        horse1.setRider("Alex", 4321, 30);

        System.out.println("Name: " + horse1.getAnimalName());
        System.out.println("Skills: " + Arrays.toString(horse1.skills));
        System.out.println("Rider: " + horse1.getRider().getRiderName());
    }
}
...
```

## Exercise 8.2

1. Create a second horse using the equals operator. Give it a different name, skills, and rider.
2. After copying and setting the values, print out the information for the first and second horse. Are the results what you expected?

```
...
public static void main(String[] args){

    Animal horse1 = new Animal("Horse","Equidae","Chordata","Blaze");
    String[] horse1Skills = new String[]{"Jumping", "Dressage"};
    horse1.setAnimalSkills(horse1Skills);
    horse1.setRider("Alex", 4321, 30);

    System.out.println("Original Horse1");
    System.out.println("Name: " + horse1.getAnimalName());
    System.out.println("Skills: " + Arrays.toString(horse1.skills));
    System.out.println("Rider: " + horse1.getRider().getRiderName());

    Animal horse2 = new Animal();
    horse2 = horse1;
    horse2.setAnimalName("Thunder");
    horse2.setRider("Taylor", 1234, 25);
    String [] horse2Skills = new String[]{"Racing"};
    horse2.setAnimalSkills(horse2Skills);

    System.out.println();
    System.out.println("Horse1");
    System.out.println("Name: " + horse1.getAnimalName());
    System.out.println("Skills: " + Arrays.toString(horse1.skills));
    System.out.println("Rider: " + horse1.getRider().getRiderName());
    System.out.println("Horse2");
    System.out.println("Name: " + horse2.getAnimalName());
    System.out.println("Skills: " + Arrays.toString(horse2.skills));
    System.out.println("Rider: " + horse2.getRider().getRiderName());
}
...
```

## Exercise 8.3

1. Create a second horse using the clone method. Give it a different name, skills, and rider.
2. After copying and setting the values, print out the information for the first and second horse. Are the results what you expected?

```
...
public static void main(String[] args){

    Animal horse1 = new Animal("Horse","Equidae","Chordata","Blaze");
    String[] horse1Skills = new String[]{"Jumping", "Dressage"};
    horse1.setAnimalSkills(horse1Skills);
    horse1.setRider("Alex", 4321, 30);

    System.out.println("Original Horse1");
    System.out.println("Name: " + horse1.getAnimalName());
    System.out.println("Skills: " + Arrays.toString(horse1.skills));
    System.out.println("Rider: " + horse1.getRider().getRiderName());

    Animal horse2 = (Animal)horse1.clone();
    horse2.setAnimalName("Thunder");
    horse2.setRider("Taylor", 1234, 25);
    String [] horse2Skills = new String[]{"Racing"};
    horse2.setAnimalSkills(horse2Skills);

    System.out.println();
    System.out.println("Horse1");
    System.out.println("Name: " + horse1.getAnimalName());
    System.out.println("Skills: " + Arrays.toString(horse1.skills));
    System.out.println("Rider: " + horse1.getRider().getRiderName());
    System.out.println("Horse2");
    System.out.println("Name: " + horse2.getAnimalName());
    System.out.println("Skills: " + Arrays.toString(horse2.skills));
    System.out.println("Rider: " + horse2.getRider().getRiderName());
}
...
```

## Exercise 8.4

1. Write a copy constructor for the `Animal` class and `Rider` class.
2. Create a second horse using the copy constructor. Give it a different name, skills, and rider.
3. After copying and setting the values, print out the information for the first and second horse. Are the results what you expected?

```
...
public static void main(String[] args){

    Animal horse1 = new Animal("Horse","Equidae","Chordata","Blaze");
    String[] horse1Skills = new String[]{"Jumping", "Dressage"};
    horse1.setAnimalSkills(horse1Skills);
    horse1.setRider("Alex", 4321, 30);

    System.out.println("Original Horse1");
    System.out.println("Name: " + horse1.getAnimalName());
    System.out.println("Skills: " + Arrays.toString(horse1.skills));
    System.out.println("Rider: " + horse1.getRider().getRiderName());

    Animal horse2 = new Animal(horse1);
    horse2.setAnimalName("Thunder");
    horse2.setRider("Taylor", 1234, 25);
    String [] horse2Skills = new String[]{"Racing"};
    horse2.setAnimalSkills(horse2Skills);

    System.out.println();
    System.out.println("Horse1");
    System.out.println("Name: " + horse1.getAnimalName());
    System.out.println("Skills: " + Arrays.toString(horse1.skills));
    System.out.println("Rider: " + horse1.getRider().getRiderName());
    System.out.println("Horse2");
    System.out.println("Name: " + horse2.getAnimalName());
    System.out.println("Skills: " + Arrays.toString(horse2.skills));
    System.out.println("Rider: " + horse2.getRider().getRiderName());
}
...
```

**Tip: As of JDK 10, you can use var to declare variables. e.g. `String myStr = "Horses";` `var myStr = "Horses";`**

# Exercise 8.5

1. Write a program which can place any number of queens, up to a maximum of 8, on a chessboard such that no queen can attack any other queens.
2. Reminder:
    - A chessboard is an 8x8 grid.
    - A queen is a chess piece which can attack any piece located in the same row, same column, or in any of the four diagonals.

- Name the class `EightQueens`
- Make all the required methods public
- Represent the chessboard as an instance variable, a two-dimensional char array. Use the char **Q** for a queen and **o** (lower case letter O) for an empty space
- The class must support a deep copy through a method called clone()
- When an object is created, it should return an empty chessboard (that is, one filled with **o**)
- `getBoard()` returns the current state of the chessboard
- `setQueen(int row, int column)` marks the square as **Q**
- `emptySquare(int row, int column)` marks the square as **o**
- `setQueens(int queensRemaining)` returns `boolean` (success or failure of placement). It sets the specified number of queens in allowed positions on the board. Note that the board may already have queens placed on it
- The solution should employ recursion
- Placement of the queens must be calculated by your program, rather than hardcoded

**Tip: Create a method which will mark squares as threatened. Just remember to remove the marks!**

**Important: Ensure your recursion has a base case!**