

# Final Practical Exam

**Due** 15 Jun by 18:00      **Points** 100

**Available** 15 Jun at 16:00 - 15 Jun at 18:30 2 hours and 30 minutes

This assignment was locked 15 Jun at 18:30.

## Introduction

**This session ends 18.00 pm**

**Make sure to complete your submission in time.**

**Submit to SVN & Websubmission (for every problem solved)**

## Assessment

- **This is a practical exam - your work must be entirely your own.**
- **This session starts at 16:10 (4:10pm)**
- **Due date:** End of this section (18:00 pm).
- **Don't forget** to take notes in a notes.txt file and commit this to this prac exam directory in svn as a backup if you score less than 50% for functionality (see details at end of this page).
- **Do Not Forget** To Submit on [WebSubmission](https://cs.adelaide.edu.au/services/websubmission/) (<https://cs.adelaide.edu.au/services/websubmission/>) **OFTEN.**
- The script mark is worth 80%.
- **Late penalties:** Only work submitted during your enrolled practical session will be marked.

Regarding functional marks, please consider:

- (1) only codes that can compile will be marked;
- (2) only codes that are in the suggested directory will be marked;
- (3) only codes submitted to SVN and WebSubmission before the deadline will be marked;

## Exam conditions:

- No discussion - except with tutors

- No discussion - except with tutors.
- No copying or use of other's work.
- No talking

Link to Translate (if useful):

- [translate.google.com](https://translate.google.com)  (<http://translate.google.com/>)

## Submission

```
https://version-control.adelaide.edu.au/svn/<YOUR-ID>/2022/s1/fcs/week-13/final-exam
```

Submit often. There are marks even for partial implementations of some classes. Look at any messages carefully - especially the ones that refer to expected input and output these can help you debug. Also make use of the driver files for each questions - these can really help.

## Repository

All your files must be in the directory above (**no need for sub-directories in this exam**)

## Exam Specification

Note that **you are allowed to use built-in data structures** in this exam (these should only be relevant to the third question below).

### Part1 - Array Manipulation (25%)

Define and implement a code to **Handle Arrays**.

```
public class HandlingArrays {  
    public static void printArray(double [] testArray) {  
        // your code goes here  
    }  
  
    public static double[] divElements(double [] Array, double [] factors) {  
        // your code goes here  
    }  
  
    public static int peakArrays(double [] Array) {  
        // your code goes here  
    }  
}
```

You are asked to define:

- Define and implement the `printArray`, for a given array;

- Define and implement `divElements`, which divides **element-wise** two arrays; The term element-wise means that you want to perform an operation on each element of a vector while doing a computation. In this case you want to divide each element `Array[i]` by the `factor[i]`. Assume the elements of factor are never zero.
- You might use `java.lang.RuntimeException` as shown below;

Quick tip:

```
if( // condition checking whether an array is bigger than the other ){
    throw new RuntimeException("Error - Arrays different shape");
}else{
    // do something
}
```

- Define and implement the `peakArrays`, which reads an array and prints each peak (element that is greater than the previous one and the next one). It also counts and returns the number of peaks. See output expected for examples

#### Output Expected:

##### PrintArray method:

Given array `{1.0,2.0,3.0,4.0}`, for instance. The output of `printArray` should be:  
*printing Array: [1.0,2.0,3.0,4.0]*

##### divElements method:

Given array `{3.0, 4.0, 6.0}` and `{3.0, 1.0, 4.0}`, the output of `divElements` should be:  
`{1.0, 4.0, 1.5}`

Given two arrays of different size, the output of `divElements` should be:  
*Error - Arrays different shape.*

##### peakArrays method:

Given array `{1.0, 4.0, 3.0, 2.0, 8.0, 6.0}`, the method should return 2 and its output should be:

*element 1 (4.0) is a peak*  
*element 4 (8.0) is a peak*

#### Important

1. The class `TestArray` **MUST NOT** contain a ***public static main*** function.
2. In order to test your code, please create another class `Main.java` and perform your test from this class.
3. The method `printArray` display information on the screen. The other methods just return an array;

#### Repository

save this project as `HandlingArrays.java`  
 add to your svn repository


The classes **MUST NOT** contain a ***public static main*** function.

Add and commit your `HandlingArrays.java` file and submit your solution, then proceed to the next question.

## Part 2 - Solitaire Player Federation

Solitaire is a card game that you play against yourself. When you play solitaire you either win or you lose. Solitaire players are very organised. As a result the Australian Solitaire Players Federation is also very organised. The Australian Solitaire Players Federation keeps track of all registered solitaire players in Australia. This is so they can help organise clubs and work out player rankings.

## Player Class (25%)

Write a Java class to represent a solitary Player. This class should **extend** the Person class provided for you in [this file, \(https://myuni.adelaide.edu.au/courses/74996/files/11111274?wrap=1\)](https://myuni.adelaide.edu.au/courses/74996/files/11111274?wrap=1)  [\(https://myuni.adelaide.edu.au/courses/74996/files/11111274/download?download\\_frd=1\)](https://myuni.adelaide.edu.au/courses/74996/files/11111274/download?download_frd=1) with name and age attributes, and have at least the following additional attributes:

- an integer **id** representing the unique identifier for this player. When a new Player object is created they will get the next available id. To keep track of the next available id you might need a static variable in this class. Note that the very first player id is 1.
- an integer **numWins** that represents the number of wins this player has had in their career.
- an integer **numPlayed** that represents the number of times a player has played Solitaire in their career.

**Note that due to a quirk in the scoring of Solitaire it is possible for players to count more wins than games played! ( It is due to a weird historical rule - nobody remembers why).**

Of course because Player extends Person, Players also automatically have the name and age attributes of Person. Your Player class should have the following methods:

- a constructor that takes parameters representing the player's name, the player's age, the number of wins the player has had and the number of times the player has played. This constructor should then initialise the name and age (in the super class) and also initialise numWins and numPlayed and the player **id**. **Note**: that you will also need to update the static count of the next available id (so that the next player created gets the next id number).
- a void method called **win()** that takes no parameters and increments the number of wins and the number of games played by this player.
- a void method called **lose()** that takes no parameters and just increments the number of games played.
- a method called **getRanking()** that takes no parameters and returns an integer value representing this player's ranking score. This score is calculated as:

**numPlayed multiplied-by ( numWins divided-by numPlayed)**

Note that all the calculations above are done with **integers**. This means that (due to rounding)

the ranking is **not** always the same as **numWins**. Note that this scoring calculation is unconventional (its related to the quirk described above) but the calculation above faithfully implements the conventions of the Solitaire Federation. Note also that if a player has played zero times getRanking should just return zero.

- a method called **getId()** that just returns the integer id of this player.
- a **toString()** method which returns the String representing the person's attributes concatenated with "Id: " followed by the player's id and then " Ranking: " followed by the player's ranking.

To help test your Player class you can use the [following driver file](#).

<https://myuni.adelaide.edu.au/courses/74996/files/10490649/download?wrap=1> ↓

[https://myuni.adelaide.edu.au/courses/74996/files/10490649/download?download\\_frd=1](https://myuni.adelaide.edu.au/courses/74996/files/10490649/download?download_frd=1)

When you compile and run the driver file you should see the output:

```
Person: Sally Smith is age: 18 Id: 1 Ranking: 0
Person: Jane Austen is age: 23 Id: 2 Ranking: 6
Person: Spiro Agnew is age: 76 Id: 3 Ranking: 8
Person: Sarah Smith is age: 18 Id: 1 Ranking: 0
Person: Jane Austen is age: 23 Id: 2 Ranking: 4
Person: Spiro Agnew is age: 77 Id: 3 Ranking: 9
18
Jane Austen
77
9
0
2
```

Note how the id's increase automatically as the players are constructed and how the id's start at 1. As before, note that having your driver file work in the desired way does not guarantee you will pass all the tests - we will run more tests.

Add your Java files (both Person.java and Player.java) and commit and submit your solutions and then proceed to the next question.

## Club Class (50%)

Last but not least, write a Java class file called Club.java that represents a solitaire club. A club keeps track of its members. Members can belong to more than one club (the Solitaire Players Federation is organised but not jealous!). Your class should have the following attribute:

- a collection to contain Players called members

It should have the following methods:

- a default **constructor** that takes no parameters.
- **addMember** a void method that takes a member as a parameter and adds the member to its collection of members (note you may want to keep the member list sorted in alphabetical order in to make printing easier).

a boolean method called **removeMemberById** which takes an integer parameter representing the

id of a Player and tries to remove a player with that id from its collection of members. If the removal is successful this method should return true. If the player can't be found then this method should return false.

- a method called **getHighestRankedPlayer** that returns the Player in the club that has the highest ranking. Note that if more than one player have the equal highest ranking the player, the youngest of them should be returned. If the club is empty this method should return null.
- a method called **printMembers()** that calls toString on each Player in the club to print out all the members in *alphabetical* order.
- a method called **countMembers()** that print out the number of junior (age < 21) and senior (age >= 21) members. An example of the output format is given below

```
The club has 1 junior and 2 senior members
```

- a method called **getHighestRankedJunior** that returns the Player under 21 years of age that has the highest ranking. If there is a tie, the younger player will be the winner. If there are no junior players in the club this method should return null.

To help test your Club class you can use the [following driver file.](#)

<https://myuni.adelaide.edu.au/courses/74996/files/10490552/download?wrap=1> 

[https://myuni.adelaide.edu.au/courses/74996/files/10490552/download?download\\_frd=1](https://myuni.adelaide.edu.au/courses/74996/files/10490552/download?download_frd=1) (note this driver is not testing the last two methods)

When you compile and run the driver file you should see the output:

```
Club 1
Person: Jane Austen is age: 23 Id: 2 Ranking: 6
Person: Sally Smith is age: 18 Id: 1 Ranking: 0
Person: Yasi Jones is age: 53 Id: 5 Ranking: 2
Club 2
Person: Jenny Lee is age: 33 Id: 4 Ranking: 8
Person: Sally Smith is age: 18 Id: 1 Ranking: 0
Person: Spiro Agnew is age: 76 Id: 3 Ranking: 8
Club 1
Person: Jane Austen is age: 23 Id: 2 Ranking: 4
Person: Sally Smith is age: 18 Id: 1 Ranking: 1
Person: Yasi Jones is age: 53 Id: 5 Ranking: 4
Club 2
Person: Jenny Lee is age: 33 Id: 4 Ranking: 9
Person: Sally Smith is age: 18 Id: 1 Ranking: 1
Person: Spiro Agnew is age: 76 Id: 3 Ranking: 10
true
false
Club 1
Person: Jane Austen is age: 23 Id: 2 Ranking: 4
Person: Yasi Jones is age: 53 Id: 5 Ranking: 4
Club 2
Person: Jenny Lee is age: 33 Id: 4 Ranking: 9
Person: Sally Smith is age: 18 Id: 1 Ranking: 1
Person: Spiro Agnew is age: 76 Id: 3 Ranking: 10
Highest Ranked Players
Person: Jane Austen is age: 23 Id: 2 Ranking: 4
Person: Spiro Agnew is age: 76 Id: 3 Ranking: 10
```

As before note that having your driver file work in the desired way does not guarantee you will

pass all the tests - we will run more tests.

Add your Java files and commit and submit your solutions.

## Optional: Your Notes: Can Help Make up Some Marks if your Programs Don't Pass the Tests

You should be documenting what you are trying to do as you write your program. This can help reveal your software development process. As you work should be updating a file called **notes.txt** in your svn directory for this exam in the final-exam directory. If you commit this file we can give you marks for a detailed and informed description of your software development process that can make up to 50% of the marks for functionality **if your marks for functionality fall short of 50%** (that is 35/70).

This file is a narrative so we would expect to see things such as:

- your own words describing what you are trying next.
- very specific ideas about your approach to solving each part of the problem and the test cases you will use to identify that component is working.
- notes on the bugs you encounter, what you think they are and the tests that you do to isolate them.
- notes on your design.
- anything that relates to process you are following to develop the software.

These notes are a narrative so they must be readable sequentially. They don't have to be well formatted or divided into themes they have to describe in detail what you are doing as you do it.

## End of Exam.