COMP503/ENSE502/ENSE602: Week 09 – Exercises

➢ Exercise 1: WATCH_DOGS

WATCH_DOGS is a video game in which players spy on others to gather pieces of information and prevent crimes. The player starts the game with no information and no crimes prevented.

Create the **WDPlayer** class comprising

- data storing the ***player name***, number of information pieces ***nPieces*** collected and crimes prevented ***nPrevent***
- get and set methods for all instance variables
- `public Double rank()`method which computes and returns the player's rank according to the following equation:
  - rank = 1-1/(nPieces*nPrevent)
  - and rank = 0 if this quantity is undefined (e.g. when nPieces*nPrevent is 0).
- an implementation of **Comparable<WDPlayer>** based on the player's ranking. You will need to implement the ***compareTo*** method (i.e. `public int compareTo(WDPlayer o)`). Compare WDPlayer objects based on the rank (i.e. `this.rank()`). Note that Integer objects implement compareTo.
- an `public static ArrayList<WDPlayer> inputPlayer()`method which asks the user to input the name, number of information pieces and number of crimes prevented using a Scanner. Instantiate **WDPlayer** objects with this data and add them to an **ArrayList<WDPlayer>**. Once the user types "stop" for the name, return the sorted collection.

COMP503/ENSE502/ENSE602: Week 09 – Exercises

➢ Exercise 2: Customer Pay Roll Sorter

- Create the **Customer** class with attributes for storing a ***name*** and a numeric value representing a ***salary***; e.g. $60,000. Implement the **Comparable** interface and provide a **compareTo** method which compares customers based on their name.
- Create the **CustomerPayRoll** class with an instance variable to store a ***list of Customer objects***. Write a constructor to instantiate an empty list and include a **get** method for the list.
- Write the `public static ArrayList<Customer> inputCustomer()` method which asks the user to input names, via the **Scanner**. Instantiate **Customer** objects and **add** them to a collection of customers e.g. **ArrayList<Customer>**. Once the user types ”stop”, sort the list of customers.

COMP503/ENSE502/ENSE602: Week 09 – Exercises

➢ Exercise 3: ContactDetails ArrayList Comparison

- Create the **ContactDetails** class which stores the contact details of a Person: their ***relationship, first name, last name*** and ***company name***, with suitable getters, setters and constructors. The ***relationship*** data should be an enumerated type with the following: **FAMILY**, **FRIEND**, **COLLEAGUE**.

- The **ContactDetails** class should implement the **Comparable** interface. The **compareTo** method should compare last names of the **ContactDetails** objects.

- Write a main method which instantiates at least five different **ContactDetails** objects. Store these objects in **ArrayList<ContactDetails>** and invoke the following **Collections** methods:
  - sort
  - min
  - max
  - shuffle

  Print out the results of each method.
- create the static method **compareByRelationship** which defines a **Comparator** object, comparing **ContactDetails** objects by their relationship. Use this method to sort the **ArrayList** by relationship.

➢ Exercise 4: HashSet Filtering

Create the class **SetFilter**.

Write **filter** such that given an input **set** and **term** the method instantiates and returns a new **HashSet** of only the strings in set containing **term**.
Sample usage of filter is shown below:

```
HashSet<String> words = new HashSet<String>();
words.add("abc");
words.add("defgabc123");
words.add("qwerty");

HashSet<String> filterWords = filter(words,"abc");
System.out.println(filterWords);

> [abc, defgabc123]
```

➢ Exercise 5: Contact Details HashMap Lookup

Create the **ContactDetails** class which stores the contact details of a Person: their relationship, first name, last name and company name, with suitable getters, setters and constructors. The relationship data should be an enumerated type with the following: FAMILY, FRIEND, COLLEAGUE.
Next, create an **AddressBook** class with the instance variable

HashMap<String,ContactDetails> contacts

Create the following **AddressBook** methods

- A constructor to initialize contacts as an empty hashmap
- *void store(ContactDetails)*, which uses hashmap's put method to associate a person's last name with their contact details
- *ContactDetails lookup(String lastname)*, which searches the set of keys of contacts to find a matching last name (use contains to compare strings). If found, get the associated ContactDetails. For this method, you will need to use keySet and get methods in the HashMap class.

Create a ContactDetailsTest class for checking the AddressBook functionality: store and lookup.