INF100 H17 Semester Assignment 3

Due on Friday, November 10, 2017

Albin Severinson, Dag Haugland

Abstract

For the final semester assignment we will create a system to handle sending parcels between people. This assignment we will be creating classes rather than methods in the problems. The final program will depend on all of the classes. You should only hand in your final program. We will only grade this final program. The program is graded on a 0 to 100 points scale. If you have questions specific to this assignment you should send them to me at `albin.severinson@uib.no`.

Submission

1. Add your program file to a `.zip` archive named `sx3_firstname_lastname.zip`. You do this by right clicking on the file ending with `.java` that contains your program code and choosing compress or add to archive (exactly what it is called varies by operating system and version). You should only add your final program, i.e., the one for problem 7. Remember that the file name must match the class name and that you therefore can not rename the `.java` file! Ask your group leader if you are having issues with this. Note that handing in the wrong file may cause you to fail the assignment!
2. If you complete the voluntary getting ahead part of the assignment, it should be handed in separately. If you complete this part you thus need to hand in two files (the `.java` file for problem 6 and the `.java` file for the getting ahead part). Put both files into the same `.zip` archive.
3. Log in to `https://mitt.uib.no` and press the box marked with "INF100" to enter the page for this course.
4. Press "Oppgåver" on the left. Next, select "Semester Assignment 3".
5. Press "Lever oppgåve" and upload the `.zip` file you created in step 1.

Grading

- This assignment is worth a total of 100 points. Each problem is worth about 14 points $\left(\frac{100}{7}\right)$.
- We will deduct points for poor code. For example, lack of comments, poorly chosen variable names (please do not use non-ASCII characters such as Ø in variable names :)), incorrect indentation, and code repetition may be reason to deduct points.
- Have a look at the Google Java style guide if you are unsure on how to structure some part of your code: `https://google.github.io/styleguide/javaguide.html`. See the next section for comment guidelines.
- Make sure your solution actually does what the problem asks for. We may deduct points if you implement something other than what the problem asks for.

Comment Guidelines

These comment guidelines are adapted from the guidelines of David Eck at Hobart and William Smith Colleges (see `http://math.hws.edu/eck/cs124/f11/style_guide.html`).

- Every variable that has a non-trivial role in the program should have a comment that explains its purpose. For-loop variables and other local utility variables do not, in general, have to be commented.
- Comments can be included in the body of a method when they are needed to explain the logic of the code. In general, well-written code needs few comments.
- Comments should never be used to explain the Java language. A comment such as "declare an int variable named x" or "increment the variable ct" is worse than useless. Assume that your reader knows Java! (Note that such comments are sometimes used in programming textbooks or on the blackboard, but never in real programs.)

General Guidelines

These are general guidelines that will (hopefully) help you maintain your sanity through your programming endeavours.

- Have your computer indent your code for you. You do this in DrJava by selecting all of your code, right clicking, and selecting `indent lines`.
- You should make a copy of all your code after completing each problem so that you can roll back to a previous version when you inadvertently break something. Even the developers at Google sometimes have to roll back.
- Set aside time to clean up your code. The number that gets bandied around on the internet is that the best programmers spend roughly half their time cleaning up and rewriting their code, i.e., you are halfway done when you get it working.
- You save time by making sure you properly understand your current program before moving on.
- You should probably be spending a significant part of your time on stackoverflow.
- Google is your friend when you want to figure out how a standard class or method works (`Scanner`, say). For example, searching for “java Scanner” gives you its Java docs page as the first result.
- When in doubt, look at the Google Java style guide to figure out how to write something.

Postal Service

You have been employed as the manager of the local post office. Confident in your programming skills you have already fired all of the employees with the intention of replacing them with robots. You figure that the first step is to write a Java program that handles registering parcels. You know that you need the following functionality:

- Register parcels, including sender and recipient names and addresses.
- Print the registered parcels to the display.
- Write the registered parcels to a text file.
- Clear the registered parcels.

More specifically, this is how you want your program to behave:

```
Enter command (0 parcel(s) registered)
r: register parcel
p: print parcels to display
w: write parcels to file
c: clear parcel queue
q: quit
> r
Register sender
----------------
Enter person name:
> Dag
Enter street:
> Thormohlensgt
Enter street number:
> 55
Enter postal code:
> 5020
Enter town:
> Bergen
Enter country:
> Norway

Register recipient
----------------
Enter person name:
> Albin
Enter street:
> Fosswinckelsgt
Enter street number:
> 10
Enter postal code:
> 5007
Enter town:
> Bergen
Enter country:
> Norway

[Parcel registered]
Enter command (1 parcel(s) registered)
r: register parcel
p: print parcels to display
```

```
w: write parcels to file
c: clear parcel queue
q: quit
> r
Register sender
----------------
Enter person name:
> foo
Enter street:
> foostreet
Enter street number:
> 1337
Enter postal code:
> 7331
Enter town:
> footown
Enter country:
> foocountry

Register recipient
----------------
Enter person name:
> bar
Enter street:
> barstreet
Enter street number:
> 72
Enter postal code:
> 73
Enter town:
> bartown
Enter country:
> barcountry

[Parcel registered]
Enter command (2 parcel(s) registered)
r: register parcel
p: print parcels to display
w: write parcels to file
c: clear parcel queue
q: quit
> p
Barcode: eda8ffb1-ed21-40bc-98b0-32564529568c
Sender: Dag
Thormohlensgt 55
5020, Bergen
Norway
Recipient: Albin
Fosswinckelsgt 10
5007, Bergen
Norway

Barcode: 20022a84-8922-4f43-9258-b4fd6619bd04
```

```
Sender: foo
foostreet 1337
7331, footown
foocountry
Recipient: bar
barstreet 72
73, bartown
barcountry

--------------------
Enter command (2 parcel(s) registered)
r: register parcel
p: print parcels to display
w: write parcels to file
c: clear parcel queue
q: quit
> w
Enter filename
> parcels.txt
[Parcels written to file]

--------------------
Enter command (2 parcel(s) registered)
r: register parcel
p: print parcels to display
w: write parcels to file
c: clear parcel queue
q: quit
> c
Are you sure you want to delete all: 2 parcel(s)?
YES / NO
> YES
[Cleared registered parcels]

--------------------
Enter command (0 parcel(s) registered)
r: register parcel
p: print parcels to display
w: write parcels to file
c: clear parcel queue
q: quit
> p

--------------------
Enter command (0 parcel(s) registered)
r: register parcel
p: print parcels to display
w: write parcels to file
c: clear parcel queue
q: quit
> q
[Quitting]
```

We will now look into how to implement this. Objects will be central to this assignment. These two videos are good reminders for the concepts we need

- `https://www.youtube.com/watch?v=MK2SMJZbUmU&t=4s`
- `https://www.youtube.com/watch?v=l0N6WvIVoUI&t=237s`

Problem 1 ($\frac{100}{7}$ points)

The first step to sending parcels is dealing with addresses. Write a class `Address` to represent addresses. The class should have fields

```java
private String street;
private int streetNumber;
private int postalCode;
private String town;
private String country;
```

All of these fields should be given as arguments to the class constructor. Furthermore, the class should have a `toString` method that returns a string with the following format

```
street StreetNumber
postalCode, town
Country
```

Remember that this method gets called automatically when you print the object using, for example, `System.out.println`.

Example

The following code

```java
public static void main(String[] args) {
    String street = "Fake Street";
    int streetNumber = 123;
    int postalCode = 32145;
    String town = "Bergen";
    String country = "Norway";
    Address address = new Address(street, streetNumber,
                              postalCode, town, country);
    System.out.println(address);
}
```

Should print the following to the terminal

```
Fake Street 123
32145, Bergen
Norway
```

Problem 2 ($\frac{100}{7}$ points)

Now that we have addresses working we have to deal with people. Write a class `Person`. The class should have fields

```
private String name;
private Address address;
```

These fields should be given as arguments to the class constructor. Furthermore, the class should have a `toString` method that returns a string with format

```
name
address
```

Example

The following code

```
public static void main(String[] args) {
    String street = "Fake Street";
    int streetNumber = 123;
    int postalCode = 32145;
    String town = "Bergen";
    String country = "Norway";
    Address address = new Address(street, streetNumber,
                                  postalCode, town, country);
    String name = "Dr. Java";
    Person person = new Person(name, address);
    System.out.println(person);
}
```

Should print the following to the terminal

```
Dr. Java
Fake Street 123
32145, Bergen
Norway
```

Problem 3 ($\frac{100}{7}$ points)

That is addresses and people settled! Now we can finally write a class representing a parcel. Write a class `Parcel` with fields

```
private Person sender;
private Person recipient;
private UUID barcode;
```

The `sender` and `recipient` should be given as arguments to the constructor. The `barcode` value should not be given as an argument. Rather a random value should be generated in the constructor using `UUID.randomUUID()` (see `https://docs.oracle.com/javase/7/docs/api/java/util/UUID.html`). Remember that you need to import `java.util.UUID`. The class `toString()` method should return a string with the following format

```
Barcode: barcode
Sender: sender
Recipient: recipient
```

Problem 3 ($\frac{100}{7}$ points) continued on next page. . .

Example

The following code

```java
public static void main(String[] args) {
    String street = "Fake Street";
    int streetNumber = 123;
    int postalCode = 32145;
    String town = "Bergen";
    String country = "Norway";
    Address senderAddress = new Address(street, streetNumber,
                                        postalCode, town, country);
    String senderName = "Dr. Java";
    Person sender = new Person(senderName, senderAddress);

    street = "News Street";
    streetNumber = 999;
    postalCode = 32145;
    town = "Bergen";
    country = "Norway";
    Address recipientAddress = new Address(street, streetNumber,
                                           postalCode, town, country);
    String recipientName = "Mr. Python";
    Person recipient = new Person(recipientName, recipientAddress);

    Parcel parcel = new Parcel(sender, recipient);
    System.out.println(parcel);
}
```

Should print the following to the terminal

```
Barcode: 20022a84-8922-4f43-9258-b4fd6619bd04
Sender: Dr. Java
Fake Street 123
32145, Bergen
Norway
Recipient: Mr. Python
News Street 999
32145, Bergen
Norway
```

Note that your barcode will differ as it is generated randomly.

Interlude

That is all the classes we need! Next step is to write the user interface that creates objects as needed. As we mentioned earlier, you want the following functionality:

- Register parcels, including sender and recipient names and addresses.
- Print the registered parcels to the display.
- Write the registered parcels to a text file.
- Clear the registered parcels.

The remaining problems will consist of implementing these features. First, remove any main method you already have. For the rest of the assignment your `main` method should look as below. The 4 remaining problems will consist of filling in the missing parts. You can copy-paste the code below into your program, but you will understand it better if you write it in manually.

```java
public static void main(String args[]) {
    ArrayList<Parcel> registeredParcels = new ArrayList<Parcel>();
    Scanner sc = new Scanner(System.in);
    boolean done = false;
    while (!done) {
        System.out.printf("Enter command (%d parcel(s) registered)"
                          + "%nr: register parcel"
                          + "%np: print parcels to display"
                          + "%nw: write parcels to file"
                          + "%nc: clear parcel queue"
                          + "%nq: quit%n> ", registeredParcels.size());
        String command = sc.next();
        if (command.equals("r")) {
            // register parcel
        } else if (command.equals("p")) {
            // print registered parcels to display
        } else if (command.equals("w")) {
            // write registered parcels to file
        } else if (command.equals("c")) {
            // clear registered parcels
        } else if (command.equals("q")) {
            System.out.println("[Quitting]");
            done = true;
        } else {
            System.out.println("[Unknown command]");
        }
    }
}
```

Problem 4 ($\frac{100}{7}$ points)

Implement the register parcel feature. The program should ask the user for the necessary information. Specifically, this is the information needed to construct the sender and receiver `Person` objects. You must make sure that the program does not crash if the user inputs invalid data. Returning to the main loop and letting the user re-start the registration by typing “r” again is an acceptable solution. Allowing the user to retry entering the data

is another (much better!) solution. You only need to verify the format of the data. You do not need to verify that the country and postal code actually exist. If you want to do that anyway, here is a list of all countries that you could use to validate the country name: `https://www.searchify.ca/list-of-countries/` :) You should implement this method by writing a method that you call from `main`. This method should have signature

```
public static Parcel registerParcel() {
    // create and return new Parcel object
    // the user should be asked to enter the required data
}
```

The `Parcel` object returned from this method should be added to the `registeredParcels` `ArrayList`. See the Postal Service section for an example on how this method should work.

Problem 5 ($\frac{100}{7}$ points)

Implement writing all registered parcels to the terminal. This feature should also be implemented as a separate method that you call from `main`. The method should have signature

```
public static void parcelsToTerminal(ArrayList<Parcel> registeredParcels) {
    // write all parcels to the terminal
}
```

See the Postal Service section for an example on how this method should work.

Problem 6 ($\frac{100}{7}$ points)

Implement writing all registered parcels to a file. The program should ask the user for the name of the file to write to. The exact same text should be written to the file as was written to the terminal in the previous problem. As always this should be implemented as a separate method. The method should have signature

```
public static void parcelsToFile(String filename,
                                 ArrayList<Parcel> registeredParcels)
    throws FileNotFoundException {
    // write all parcels to file with name filename
}
```

The program must not crash if there is a problem writing to the file. The user should be notified of the problem in this case. Implement this by catching the exception in `main`.

Problem 7 ($\frac{100}{7}$ points)

Implement the clear parcels feature. Typing “c” should prompt the user with a "Are you sure you want to delete all: “x” parcel(s)?" and if so remove all registered parcels, before returning to the main menu. This feature does not need a separate method.

Getting Ahead

This is it! Last semester assignment done. Well done! You have earned yourself a break. The mentality of always getting ahead should not stop with the last semester assignment though. For example, we can do a lot more interesting things with objects than we have so far! For the getting ahead part we will consider the case that there are many post offices that each have their own list of registered parcels. Furthermore, we will implement code for sending parcels between post offices.

This part is meant as preparation for the coming course material and will not be graded. However, we will still look at it and comment it! Furthermore, you never have to catch up if you are always getting ahead. This part should be handed in separately. If you complete this part you thus need to hand in two sets of files (the `.java` files for problem 7 and the `.java` files for this part). Put both files into separate folders in the same `.zip` archive.

Make sure to make a copy of all your code before moving forward with the getting ahead part as you need to hand them in separately. Write a class `PostOffice` to represent a post office. It should have fields

```
private Address address;
private Person manager;
private UUID officeId;
private ArrayList<Parcel> registeredParcels;
```

Furthermore, it should have methods for

- Register a parcel at this post office.
- Print the parcels registered at this post office to the display.
- Write the parcels registered at this post office to a text file.
- Clear the parcels registered at this post office.

As a first step you should re-implement your current `main` method to use a `PostOffice` object. At this point your program should work the same as previously from the point of view of the user.

Your next step is to implement creating new `PostOffice` objects from the terminal. You should also implement a feature that allows the user to select which `PostOffice` is currently used. All commands entered should call methods of the currently used `PostOffice`. Furthermore, the user should be able to transfer parcels from the currently used `PostOffice` to any other `PostOffice`.