Exercise 2 - Contact Manager
Unassessed

COMP70055 Software Engineering Design

In this tutorial you will write Java code with a focus on using inheritance, interfaces and abstract classes. You will also use some generic collections.

Being unassessed, **submission is not mandatory**. If you wish to get feedback about your solution to this exercise you must submit by the deadline.

Contact Manager

A Brief Description

Your task is to write some classes of a larger system for managing and interacting with contacts.

A contact is a pair consisting of a person and contact information. The latter may be an email address, a phone number or a mobile phone number. You will write a `ContactManager` class that allows adding contacts, querying the contact details available for a specific person, and sending a message to all based on the capabilities of the communications device.

Helper Classes

We provide classes `Person` and `Audio`. You should not edit these classes. In addition, we provide test classes with tests that your code must pass.

What to Do

1. Write an abstract class `ContactInfo` with three abstract methods:
   - `String contactInfo()` that returns information required to make contact (e.g., +671 0793 434 241, myemail@domain.com, etc.)
   - `String contactInfoType()` that returns thecommunication device the contact information refers to (e.g., phone, email, etc.)
   - `void sendMessage(String msg)` that sends a message.
2. Write a `Contact` class that combines a `Person` with a `ContactInfo`. It should include a constructor `Contact(Person, ContactInfo)`
3. Write interface `TextMessageEnabled` that requires a method `void sendTextMessage(String msg)`.
4. Write class `Email` as a subclass of `ContactInfo` that implements interface `TextMessageEnabled`.
   - Include a constructor that has only one parameter, a string containing the email address.
   - The implementation of `sendTextMessage` should print to `System.out` both the message and the email address.

- The implementation of `sendMessage` should behave like `sendTextMessage`.

5. Write interface `AudioMessageEnabled` that requires a method `void sendAudioMessage(Audio msg)`.
6. Write class `Phone` as a subclass of `ContactInfo` that implements interface `AudioMessageEnabled`.
   - Include a constructor that has only one parameter, a string containing the phone number.
   - The implementation of `sendAudioMessage` should print out to `System.out` the audio message (using `Audio.toString()`) and the phone number.
   - The implementation of `sendMessage` should behave like `sendAudioMessage`.
7. Write class `MobilePhone` as a subclass of `Phone` that implements **both** interfaces.
   - Methods `sendAudioMessage` and `sendTextMessage` should behave as in `Phone` and `Email` respectively. Method `sendMessage` should send a text message.
8. Write a `ContactManager` class that:
   - Stores in a `List<Contact>` contacts added with method `void add(Person, ContactInfo)`.
   - Provides a method `contactDetails(Person)` that returns in a `List<ContactInfo>` all contact details available for a given person.
   - Implements `void spam(String msg)` that sends a message for all contacts.

Getting The Skeleton Code

Clone from GitLab the skeleton code

`https://gitlab.doc.ic.ac.uk/lab2021_spring/sed_ex2_`*`login`*`.git`

You may work in a pair, but only one person needs to clone the repository. Normally, pair-programming involves two people sitting at one computer. If you need to pair-program whilst working remotely, we suggest sharing your screen in Teams, Zoom or other video platforms, working on your solution together on one computer, and making commits from there back to the repository.

Project Structure

When you clone from GitLab, you should find a structure similar to this one.

- `src`
  - `Person.java`
  - `Audio.java`
- `test`
  - `ContactManagerTest.java`
  - `MobilePhoneTest.java`
  - `TestSuiteHelper.java`
  - `EmailTest.java`
  - `PhoneTest.java`

Importing the code into IntelliJ IDEA

From the "Welcome to IntelliJ IDEA" select "Open" and browse to cloned directory to select it.

Submission

When you have finished, if you want to get feedback you must make sure you have pushed all your changes to GitLab (source code only, not .class files generated by the compiler). You can then use LabTS [1] to test the version of your work that is on GitLab. You must then submit a revision id to CATE so that we know which version of your code you consider to be your final submission.

If you are working in a pair, only one person needs to submit on LabTS - the other member of the pair can declare they are part of the group on CATE.

Note that in CATE, this exercise appears as assessed but has weight zero.

Autotest with LabTS

The LabTS build has a number of stages and checks a number of qualities of your code:

- Compilation: compiles your code - does it compile correctly?
- Test Results: runs all of your test, plus our additional tests - do they pass?
- Style: we run Checkstyle to check a number of things about the style of your code, following the Google Java Style standard conventions for naming, layout etc. If you have the Google Java Format plugin, then the menu option "Code/Reformat code", does most of the reformatting work for you.

Deadline

You are strongly encouraged to work on the exercise during the course timeslots on Tuesday. However, to receive feedback about your code, you must submit before 19hs on Thursday 27th January.

.

[1] `https://teaching.doc.ic.ac.uk/labts`