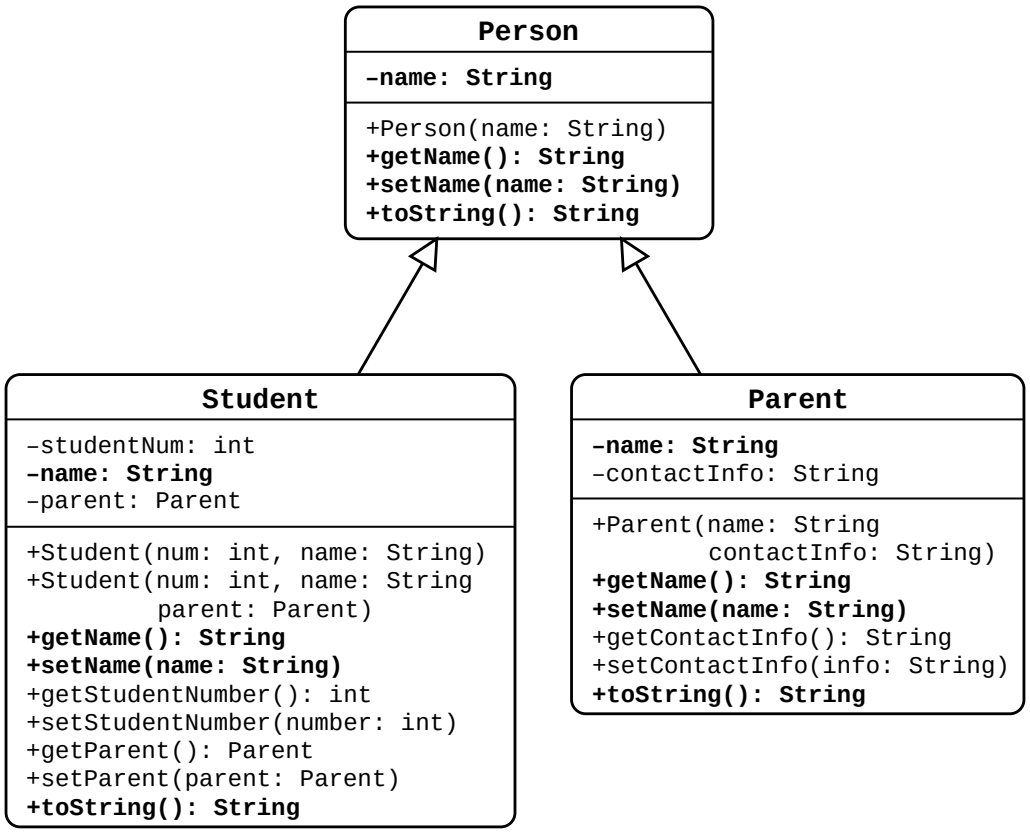
Assignment: Student Roster (part 1)

Follow the steps to implement the following UML class diagram.



1. Implement the `Person` class as described by the UML diagram shown above, and create a separate class to test the basic functionality. For the `toString` method, simply return the field `name`. As the `toString` method is overriding the implementation of `toString` we inherit from the `Object` class, it is best practice to prefix the `toString` method with the `@Override` annotation, like so:

   ```
   @Override
   public String toString() { … }
   ```

   Once you have completed both the code for the `Person` class, and also the code to test the class, show your code to the teacher.

2. Implement the `Parent` class as described by the last UML diagram. Either write a new class to test its functionality, or update the class you used to test the `Person` class. If you update the class that tests the `Person` class, do not remove the code that tests that class, just cleanly add the new test code.

   Override the `toString` method of the `Person` class in the `Parent` class so it returns a string that includes both the `name` field and the `contactInfo` field in a format that you think makes sense for printing to the screen. Again, it is best practice to annotate this method with the `@Override`.

Remember you do not need to write a `setName` or `getName` method inside the `Parent` class because it will be inherited from the `Person` class. Ensure your test code verifies this is true by calling `setName` and/or `getName` on a `Parent` object.

Once you have completed the code for both the `Parent` class, and the code that tests it's functionality, show your code to the teacher.

3. Implement the Student class as described by the last UML diagram. Again, you can write a new class to test its functionality, or add it to your previous test class – but do not remove your previous test code.

   Override the `toString` method of the `Person` class in the `Student` class so it returns a string that includes both the `name` field and the `studentNum` field in a format that you think makes sense for printing to the screen. Include the `@Override` annotation.

   Once you have completed the code for both the `Student` class, and the code that tests it's functionality, show your code to the teacher.

Summary of Inheritance for Code Re-Use

The example we have coded thus far shows how inheritance helps programmers leverage a hierarchy of types in order to avoid duplication of code. In the real world, as well as in our example code, both a `Student` is a type of `Person` and a `Parent` is a type of `Person`. When we define the information (the *fields*) that we need to associate with a `Person` and the functionality (the *methods*) we implement to operate on a `Person`, the other subtypes can ***inherit*** that information and functionality from the super class rather than duplicate it.