Working with Polymorphic Collections

Purpose: Store Students and Instructors into a HashMap, and apply inheritance and polymorphism correctly.

Learning Objectives:

- Refresher on HashMap;
- Understand downcasting and the advantages of polymorphic collections
- Practice determining the class of an instantiated object.
- Code some data validation routines
- Utilize some String functions

Instructions:

1. Create a class called Person with the following properties:
   - The Person class should have getters and setters for all properties other than id. (Use Eclipse to help you auto-generate these.)
   - Person class should also have a getter for id
   - In addition to these getters and setters, Person should have the following instance methods:
   - toString – returns full name, like "Edgar Allen Poe"
   - getEmailDomain – returns the domain portion of an email (after the @), as in "gmail.com"
   - getLast4SSN – returns the last four digits of the Social Security Number
   - Add some validation to age, email and ssn. You may have done this via the setters in the past. However, this time, let's add class (static) methods to validate these fields as best you can.
     Give some thought to what advantages this approach would have.

Property	Data Type
id	Integer
firstName	String
middleName	String
lastName	String
email	String
ssn	String
age	Integer

2. Create a driver program PersonApp to test the class
   - Prompt user for all Person properties except id. You can hardcode any id you like, but use the id as the HashMap key.
   - Accept any value for the name fields
   - Only accept valid values for age, email and ssn
   - Create a Person with the valid input

3. Notes on validation. Here is a list of validation checks you can perform. Some are easier to implement than others, and there are many ways to correctly validate these aspects of the fields. Implement as many as you can.

   - Age entered by user
     - Must be an integer greater than 16
   - Email address string entered by user
     - String must contain a '@' and at least one '.'
     - A '.' must follow the '@'
     - String must only contain one '@'
   - Social Security Number string entered by user
     - String must contain two '-' characters in the correct position
     - The length of the string must be 11
     - String must contain only digits or numbers.

   If you encounter invalid data, **log it** using the logging API of Java. (Do not use `System.out.println();`)

4. Static variables
   - Keep track of the highest age entered for a Person in a static variable.

5. Create subclasses for Instructor and Student
   - Instructors should have a Department
   - Students should have a Major
   - Generate getters and setters.

6. Add the Person, the Student and the Instructor to a HashMap.
   - In the driver program, create a Student and an Instructor (you don't need to prompt for input)
   - Add these three instances to a HashMap

7. Output the following after creating each person:
   - Loop through the HashMap and display
   - The person's full name and what kind of Person are they
   - The person's email domain
   - The last 4 digits of the person's Social Security Number
   - Whether the person is the oldest in the collection
   - The major for the Student, the department for the Instructor

Here is some sample output, to help you understand the end goal of the exercise

```
Enter person's first name
Julie
Enter person's middle name
Theresa
Enter person's last name
Collins
Enter person's email address
jCollins@acme.com
Enter person's SSN in ###-##-#### format
123-45-6789
Enter person's age
47

Julie Theresa Collins (Person)
acme.com
6789
oldest

Jane Marie Doe (Student)
students.rowan.edu
4444
not oldest
Computer Science

Stephen J. Hartley (Instructor)
rowan.edu
5555
not oldest
Math/Science
```