OOP Exercises

1. TASK 1

   - Write a Person class whose constructor initializes `name` and `age` from arguments.
   - All instances of Person should also initialize with an empty `stomach` array.
   - Give instances of Person the ability to `.eat("someFood")`:
     - When eating an edible, it should be pushed into the `stomach`.
     - The `eat` method should have no effect if there are 10 items in the `stomach`.
   - Give instances of Person the ability to `.poop()`:
     - When an instance poops, its `stomach` should empty.
   - Give instances of Person a method `.toString()`:
     - It should return a string with `name` and `age`. Example: "Mary, 50"

2. TASK 2

   - Write a Car class whose constructor initializes `model` and `milesPerGallon` from arguments.
   - All instances built with Car:
     - should initialize with a `tank` at 0
     - should initialize with an `odometer` at 0
   - Give cars the ability to get fueled with a `.fill(gallons)` method. Add the gallons to `tank`.
   - Give cars ability to `.drive(distance)`. The distance driven:
     - Should cause the `odometer` to go up.
     - Should cause the the `tank` to go down taking `milesPerGallon` into account.
   - A car which runs out of `fuel` while driving can't drive any more distance:
     - The `drive` method should return a string "I ran out of fuel at x miles!" x being `odometer`.

3. TASK 3

   - Write a Lambdasian class.
   - Its constructor takes a single argument - an object with the following keys:
     - name
     - age
     - location
   - Its constructor should initialize `name`, `age` and `location` properties on the instance.
   - Instances of Lambdasian should be able to `.speak()`:
     - Speaking should return a phrase `Hello my name is {name}, I am from {location}`.
     - {name} and {location} of course come from the instance's own properties.

4. TASK 4

   - Write an Instructor class extending Lambdasian.
   - Its constructor takes a single argument - an object with the following keys:
     - All the keys used to initialize instances of Lambdasian.
     - `specialty`: what the instance of Instructor is good at, i.e. 'redux'
     - `favLanguage`: i.e. 'JavaScript, Python, Elm etc.'
     - `catchPhrase`: i.e. `Don't forget the homies`.

- The constructor calls the parent constructor passing it what it needs.
- The constructor should also initialize `specialty`, `favLanguage` and `catchPhrase` properties on the instance.
- Instructor instances have the following methods:
  - `demo` receives a `subject` string as an argument and returns the phrase 'Today we are learning about {subject}' where subject is the param passed in.
  - `grade` receives a `student` object and a `subject` string as arguments and returns '{student.name} receives a perfect score on {subject}'

5. TASK 5

- Write a Student class extending Lambdasian.
- Its constructor takes a single argument - an object with the following keys:
  - All the keys used to initialize instances of Lambdasian.
  - `previousBackground` i.e. what the Student used to do before Lambda School
  - `className` i.e. CS132
  - `favSubjects`. i.e. an array of the student's favorite subjects ['HTML', 'CSS', 'JS']
- The constructor calls the parent constructor passing to it what it needs.
- The constructor should also initialize `previousBackground`, `className` and `favSubjects` properties on the instance.
- Student instances have the following methods:
  - `listSubjects` a method that returns all of the student's favSubjects in a single string: `Loving HTML, CSS, JS!`.
  - `PRAssignment` a method that receives a subject as an argument and returns `student.name has submitted a PR for {subject}`
  - `sprintChallenge` similar to PRAssignment but returns `student.name has begun sprint challenge on {subject}`

6. TASK 6

- Write a ProjectManager class extending Instructor.
- Its constructor takes a single argument - an object with the following keys:
  - All the keys used to initialize instances of Instructor.
  - `gradClassName`: i.e. CS1
  - `favInstructor`: i.e. Sean
- Its constructor calls the parent constructor passing to it what it needs.
- The constructor should also initialize `gradClassName` and `favInstructor` properties on the instance.
- ProjectManager instances have the following methods:
  - `standUp` a method that takes in a slack channel and returns `{name} announces to {channel}, @channel standy times!`
  - `debugsCode` a method that takes in a student object and a subject and returns `{name} debugs {student.name}'s code on {subject}`

7. STRETCH PROBLEM

- Extend the functionality of the Student by adding a prop called grade and setting it equal to a number between 1-100.

- Now that our students have a grade build out a method on the Instructor (this will be used by *BOTH* instructors and PM's) that will randomly add or subtract points to a student's grade. *Math.random* will help.
- Add a graduate method to a student.
    - This method, when called, will check the grade of the student and see if they're ready to graduate from Lambda School
    - If the student's grade is above a 70% let them graduate! Otherwise go back to grading their assignments to increase their score.