Assignment 9 Instructions

1. Assignment 9: **37 - 40 points total**

   Project points: 30 points

   Reflection points: 5 points

   Extra Credit points:

   2 E.C. points (for on-time work)

   3 E.C. points (for Ask and Answer forum participation),

   [35pts + 2 E.C. for on time work = 37 points] or [35pts + 2 e.c pts for being on time and 3 e.c pts for Ask and Answer = 40 points] or [35pts + 3 e.c points for Ask and Answer = 38 pts]

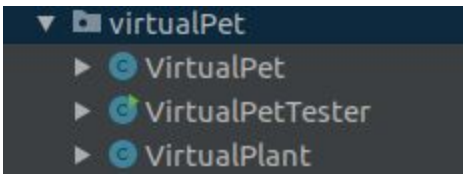
2. Due Date & Time: **11/30/2020 at 11:59 PM**

WHAT TO SUBMIT

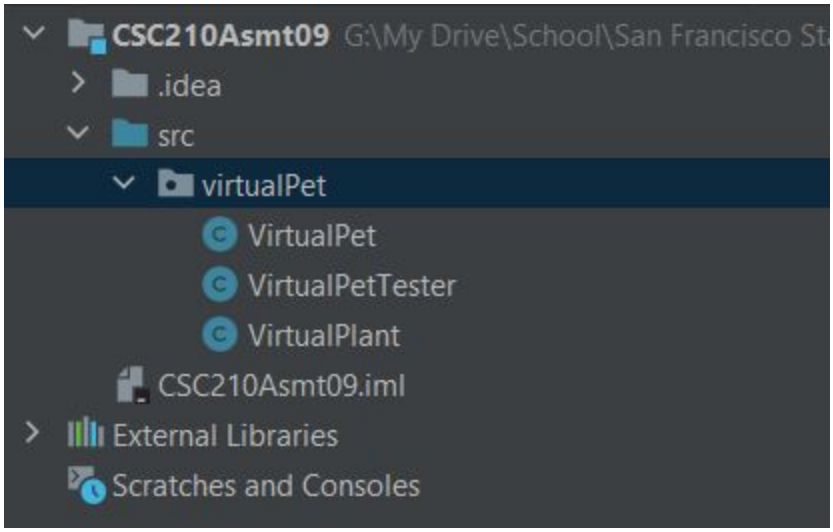
Submit 4 files to iLearn by the deadline and post a reflection on iLearn:

1. 3 Java File: Please submit All 3 Java Files to iLearn: **VirtualPetTester.java, VirtualPet.java, VirtualPlant.java[30 points]**
2. You do not need to submit your java package but instead submit a screenshot of your Java Package **"virtualPet"**, with all Java files in one package here:

   Example:



   Paste your package screenshot here:



3. 1 PDF File: Submit 1 Word/PDF file which is a filled-out, downloaded local copy of this Google page on your local computer, named "firstname-lastname-assignment-9-report.pdf". Fill this out with screenshots then save it as Word or PDF

Then go to iLearn forum to submit:

4. 1 Reflection **[5 points]**

Then go to iLearn to ask a question:

5. 1 Optional Ask and Answer forum **[3 points]**

HOW TO SUBMIT

Please upload all files **separately** via iLearn Assignments Submission, unzipped

---

GUIDELINES FOR ALL ASSIGNMENTS:

1. Each assignment includes a code portion and a non-code portion. Please submit both 2 portions.
   a. Code portion: Your source code files, only the files which you create and edit.
   b. Non-code portion: Your assignment report, only 1 **Word** or **PDF** file.

2. Please submit all required files separately, un-zipped, via iLearn Assignments Submission

3. Always **read through the entire assignment before starting and submitting any of it. Missing files or missing requirements will result in deducted points**
4. **a. Include a proper header at the top of every Java file. Figure 1**

Header Format
/* * Assignment **<assignment number>** * Description: **<program description>** * Name: **<your name>** * ID: **<your SFSU ID number>** * Class: CSC 210-**<section number>** * Semester: **<current semester>** */

Replace each tag (such as **<assignment number>**) with the appropriate text.

You should adhere to this format as closely as possible. You do not need to include the **<>** symbols in your header fields.

b. Only if you work with a Study Buddy, include your Buddy's name in your header at the top of every Java file. Figure 1

Header Format
/* * Assignment **<assignment number>** * Description: **<program description>** * Name: **<your name>** * Teammate: **<Study Buddy name>** * ID: **<your SFSU ID number>** * Class: CSC 210-**<section number>** * Semester: **<current semester>**

```
*/
```

Assignment 9

PRACTICE WITH OBJECTS & INHERITANCE

❒Part 1: VirtualPet and VirtualPetTester Classes [20 points]

Guidelines for this Assignment:

1. The required name for the package is "virtualPet" package
2. The required name for the main Java file is VirtualPetTester.Java
- It should have a main method
- It should have at most 20 lines of code

3. The required name for the 2nd Java file is VirtualPet.Java
4. The required name for the 3rd Java file is VirtualPlant.Java that inherits from the VirtualPet.java

Triple check to make sure these requirements are included. If they are not, your submission will be considered invalid and will need to be corrected before being graded.

1. **VirtualPetTester.java**

What should your VirtualPetTester.java file look like?

- **It should have a main method**
- **It should have at most 20 lines of code**

This is a requirement and I will reduce points if there are more than 20 lines of code in the main.

Use loops, arrays, or methods we have learned in class. I emphasize practicing this important skill so you understand what is efficient and inefficient code.

As a developer this will help your manager and team enjoy working with you more and as a non-developer you can help design/spot inefficient code so that maintaining the program takes less time in the long run.

The VirtualPetTester.java Program class contains the main method, and contains the user experience for your program.

The user experience should be the following:

1. The program should ask the user how many pets they want to create and name them. Think of a method to take care of this first setup.

- a. With this number you will create an array to hold the VirtualPet objects. Meaning if the user enters 3, you will use a loop to create an array type VirtualPet that will be initialized with empty VirtualPet objects. Hint: you will need to use an empty constructor in your VirtualPet class to do this.
- b. Validate the user entry, it should be a number larger than 0. If the user enters a smaller number then loop until the user enters a number larger than 0
- c. Then ask the user to enter a name for each of their pets. You will need to loop through your Array of VirtualPet Objects and for each element in the Array, set the name for the VirtualPet object. Hint: you can either create VirtualPet object with empty array and set name using setName() setter method in your VirtualPet class, or you can directly use the VirtualPet constructor with parameter name to initialize the objects in the array with names. See below instructions for the VirtualPet class.

2. And then provide the following menu options:

1. Check statuses
2. Feed your virtual pet
3. Play with your virtual pet
4. End program

3. Have the user enter their choice as an integer. Depending on their choice, the following should happen.

(1. Check statuses) --- Create a method that will display the current values of happiness and energy in regards to their virtual pet.

This method will call the printStatus method from VirtualPet.java.

In VirtualPet.java class:

- a.Create a **plainStatus** method in your VirtualPet.java class that builds and returns the string to print Energy and Happiness levels. Use getters for this.
- b. Create a **printStatus** method in your VirtualPet.java class that does not return anything but prints the plain status method.

In VirtualPlant.java class:

- Create a **printStatus** method that overrides printStatus in your VirtualPet.java class that does not return anything but uses super to call the parent plainStatus method and adds to it Plant Type. i.e. printStatus should print Energy: 35 Happiness: 35 Plant Type: orchid

(2. Feed your virtual pet) --- Create a method that will simulate the experience of feeding the virtual pet and do the following

- a. Call the method feed. Luckily for you, should already be built into the VirtualPet class as an instance method.
- b. Print status for each object. You can also reuse the same method from check status to print status for each object.
- Since you can create various pets in the array, whenever you call the method feed(), it will feed all the pets in the array

(3. Play with your virtual pet) --- Create a method that will simulate the experience of playing with the virtual pet and do the following:

- a. Call the method play. Same thing as the 2nd option: You already have this built into the VirtualPet class!
- b. Print status for each object. You can also reuse the same method from check status to print status for each object.
- Since you can create various pets in the array, whenever you call the method play(), it will play all the pets in the array

(4. End program) --- This will discontinue any other interactions with the virtual pet. Before the program ends, you should display a short summary of how well the user took care of their pet depending on the happiness status. You should user selections statements (if-else) to determine how to summarize this experience. For example, if happiness is high, then you might want to say that the user did a good job!

Or maybe happiness is low, so you might want to tell them that they should take care of their pet better.

4. If the user did not choose to end the program (Option 4), then they should be taken back to Step 2, where they will again be provided with the menu of possible actions to choose from. This cycle should repeat until the user decided to exit the program.

HINT: Use loops to make this happen!

2. VirtualPet.java

Your base class needs the following:

I. Three instance variables:

name --- A string variable representing the pet's name.

happiness --- An integer variable representing the pet's happiness. The value should range from 1-100.

energy --- An integer variable representing the pet's energy. This value should range from 1-100.

II. Constructors:

It needs two public constructors. They should initialize happiness and energy at 25.

One constructor should have a single parameter: A String parameter to set the name instance variable. Remember, you have to make sure the name has a length of less than 30 characters. How can you do this without rewriting your validation code? Hint: You can create additional methods.

One constructor should have no parameters. This constructor should simply set the name as DEFAULT, but you should use the this keyword to make sure you are not writing duplicate code. If you are not sure what this means, see the slides near the end of Chapter 9.

II. Public setter and getter methods:

a. VirtualPet needs to have setter and getter methods for the ***name*** instance variable. The setter method should validate input, meaning it should only set the name if the new name is less than 30 characters. If the validation fails, you don't have to set a default value for the name, but you can if you want to.

b. There needs to be a setter and getter for ***happiness***. You do not need to provide validation in the setter method for happiness.

c. There needs to be a setter and getter for ***energy***. You do not need to provide validation in the setter method for energy.

III. Additional public instance methods so that the user can manage their Virtual Pet

Additional public instance methods so that the user can interact with their virtual pet. These methods should not return a value, and they shouldn't have any parameters. Both methods below will have an effect on happiness and/or energy. Additionally, both methods will have restrictions. If the virtual pet is restricted from performing the task, then there should be no effect on the pet's happiness or energy.

feed --- A method that simulates the experience of feeding the pet. When called, this method should use setter methods to

- increase happiness by 5.

- increase energy by 30.

RESTRICTION: You shouldn't be able to feed your pet if energy is 80 or higher. This means that if energy is greater than or equal to 80, then calling feed() should have no effect on happiness or energy.

play --- A method that simulates the experience of playing with the pet. When called, this method should use setter methods to

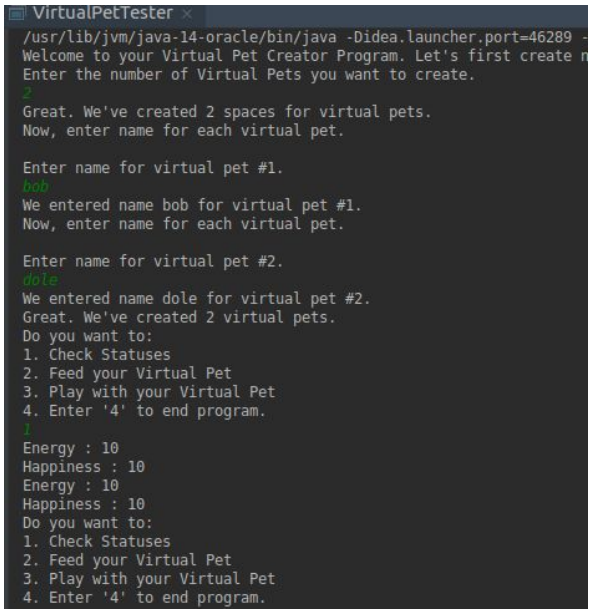
- increase happiness by 20.
- decrease energy by 15.

RESTRICTION: You shouldn't be able to play with the pet if their energy is 30 or lower. This means that if energy is lower than or equal to 30, then calling play() should have no effect on happiness or energy.

Sample output:

Enter number of pets and name output

Enter # 1 output



Enter #2, #3 output

```
2
Energy : 40
Happiness : 15
Energy : 40
Happiness : 15
Do you want to:
1. Check Statuses
2. Feed your Virtual Pet
3. Play with your Virtual Pet
4. Enter '4' to end program.
3
Energy : 55
Happiness : 35
Energy : 55
Happiness : 35
Do you want to:
1. Check Statuses
2. Feed your Virtual Pet
3. Play with your Virtual Pet
4. Enter '4' to end program.
1
Energy : 55
Happiness : 35
Energy : 55
Happiness : 35
Do you want to:
1. Check Statuses
2. Feed your Virtual Pet
3. Play with your Virtual Pet
4. Enter '4' to end program.
```

Exit output

```
Do you want to:
1. Check Statuses
2. Feed your Virtual Pet
3. Play with your Virtual Pet
4. Enter '4' to end program.
4

Process finished with exit code 0
```

In VirtualPet.java class:

- a.Create a **plainStatus** method in your VirtualPet.java class that builds and returns the string to print Energy and Happiness levels. Use getters for this.
- b. Create a **printStatus** method in your VirtualPet.java class that does not return anything but prints the plain status method.

❒Part 2: Child Class [10 points]

File Name: VirtualPlant.Java

1. The VirtualPlant class inherits from VirtualPet. This class contains:

a. A string data field plantType for plant's type, i.e. "orchid", or "succulent", etc.

b. A constructor that inherits from the parent class with the parameter name. Inherit from this parent class constructor, use "super" to initialize the name, and add a new parameter plantType that is initialized in this constructor

for ex:

```
VirtualPlant(String name, String plantType){

super(name);

this.plantType = plantType

}
```

c. Create getters and setters for the unique child variable: plantType

d. **Override** (see Chapter 11 notes for reminder of syntax) the parent method **printStatus**. In this method, use super.plainStatus to inherit the method that returns the plainStatus string and concatenate it to form another string with Plant Type string and value. Then use this string to print out Energy, Happiness, and the Plant type. Use getters for plantType value.

2. Implement the above class.

Refactor your **VirtualPetTester.java** file to include an if/else statement. This control statement will ask the user to enter the number "1" to create new Virtual Pets or "2" to create new Virtual Plants.

a. **Use overloading** to simplify your code, we covered this in lectures on Chapter 6 Methods, and on the last lecture Chapter 11 Inheritance. Here's an example of what my methods look like in VirtualPetTester, note that there are duplicates of methods, with different parameter types == overloading.

```
public static void menu(VirtualPet[] virtualPets){...}
public static void menu(VirtualPlant[] virtualPlants){...}
public static void handleUserInput(int input, VirtualPet[] virtualPets) {...}
public static void handleUserInput(int input, VirtualPlant[] virtualPlants) {...}
public static VirtualPet[] firstSetup(VirtualPet[] virtualPets) {...}
public static VirtualPlant[] firstSetup(VirtualPlant[] virtualPlants) {...}
public static void checkStatus(VirtualPet[] virtualPets){...}
public static void activateFeed(VirtualPet[] virtualPets){...}
public static void activatePlay(VirtualPet[] virtualPets){...}
public static void checkStatus(VirtualPlant[] virtualPlants){...}
public static void activateFeed(VirtualPlant[] virtualPlants){...}
public static void activatePlay(VirtualPlant[] virtualPlants){...}
```

b. If the user chooses “1” use the existing menu you’ve created for Virtual Pet. Hint: You might want to create a method “menu” with different parameters, one for type VirtualPet[] and one for type VirtualPlant[], see overloading tip in a.

Note on overloading

It turns out that overloading a lot of the methods in the test class were not needed. The only significant change that was needed was the invocation of the different constructors depending on the type of pet the user chooses to create.

Since VirtualPlant is a child of VirtualPet, instances of either class can be stored in the same VirtualPlant array. This means that overloading the methods in the test class that relies on the array as a parameter was not necessary. Note that the only difference between the two classes is the addition of the field “plantType” in the child, VirtualPlant, and the plainStatus method in the child class has already been overloaded.

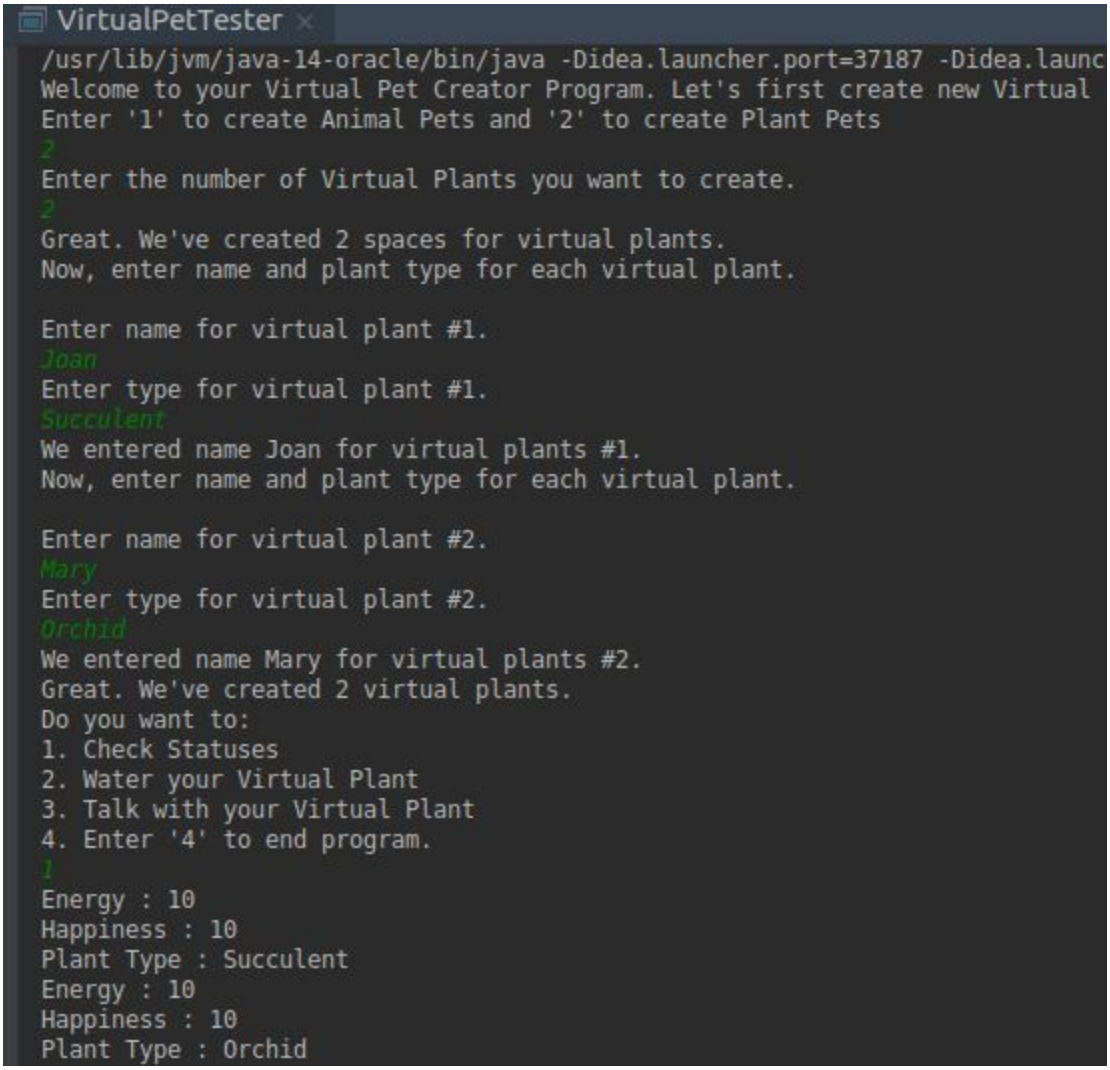
c. If the use chooses “2” use the overloaded menu you should create for Virtual Plant and will contain VirtualPlant[] array which will contain VirtualPlant objects

d. Test your code for both. You should be able to see the difference in the print outs for Pets vs Plants, i.e. the Plants will have plant types.

Example output

Output of

- if/else 1 or 2 for Virtual Pet and Virtual Plants
- quantity of Plants and naming Plants
- Plant Menu display
- option #1



Output for Options #2,#3, and #4

```
Do you want to:
1. Check Statuses
2. Water your Virtual Plant
3. Talk with your Virtual Plant
4. Enter '4' to end program.
2
Energy : 40
Happiness : 15
Plant Type : Succulent
Energy : 40
Happiness : 15
Plant Type : Orchid
Do you want to:
1. Check Statuses
2. Water your Virtual Plant
3. Talk with your Virtual Plant
4. Enter '4' to end program.
3
Energy : 25
Happiness : 35
Plant Type : Succulent
Energy : 25
Happiness : 35
Plant Type : Orchid
Do you want to:
1. Check Statuses
2. Water your Virtual Plant
3. Talk with your Virtual Plant
4. Enter '4' to end program.
4
```

❒Part 4: Reflect 50 words [5 points]

Let's reflect on what we've learned.

The following reflection is identical to the one posted on iLearn.

This assignment was a little bit confusing at the start because the instructions were a little bit unclear, but the professor cleared up a lot of the questions in the lecture. As I completed the assignment, it turns out that I did not need to overload a lot of the methods in the main class which I thought was quite interesting.

👋 **Assignment 9 Part 4: Your own Reflection**

❒Part 5 Optional Extra Credit : Ask and Answer [3 e.c points]

👋 [Assignment 9 Part 5: Ask and Answer]