**Name**: Steve Hommy

**Pair: -**

**Amount of completed tasks: 7**

**Which tasks were left undone or incomplete: 0**

Self-assessment:

## Test report

Write the test report yourself to each coding task (task number, input/action, desired output and then the testing evidence (actual output)). Add rows if necessary. Include answers to theoretical questions and pseudocode to this return document as well in addition to code screen captures. Actual output can be a screen capture of the terminal showing the output.

|  |  |  |  |
| --- | --- | --- | --- |
| **Task** | **Input / action** | **Desired output** | **Actual output (use red color if desired output != actual output)** |
| **3** | <Run Program> |  |  |
| **5** |  |  |  |
| **6** |  |  |  |
| **7** |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

1. Answer the following questions.

a. What does polymorphism (in object-oriented programming) mean? Also give a short (coding) example, e.g. google for examples).

**Polymorphism is the method in an object-oriented programming language that performs different things as per the object’s class, which calls it. With Polymorphism, a message is sent to multiple class objects, and every object responds appropriately according to the properties of the class.**

**class** Bird:

**def** intro(self):

**print**("There are many types of birds.")

**def** flight(self):

**print**("Most of the birds can fly but some cannot.")

**class** sparrow(Bird):

**def** flight(self):

    print("Sparrows can fly.")

**class** ostrich(Bird):

**def** flight(self):

**print**("Ostriches cannot fly.")

obj\_bird **=** Bird()

obj\_spr **=** sparrow()

obj\_ost **=** ostrich()

obj\_bird.intro()

obj\_bird.flight()

obj\_spr.intro()

obj\_spr.flight()

obj\_ost.intro()

obj\_ost.flight()

**Output:**

**There are many types of birds.**

**Most of the birds can fly but some cannot.**

**There are many types of birds.**

**Sparrows can fly.**

**There are many types of birds.**

**Ostriches cannot fly.**

b. What is a class variable and how are they used?

**A class variable defines a specific attribute or property for a class and may be referred to as a member variable or static member variable. They are associated with the class, rather than with any object. Every instance of the class shares a class variable, which is in one fixed location in memory. Any object can change the value of a class variable, but class variables can also be manipulated without creating an instance of the class**

c. What is an instance variable and how is it different from the class variable?

**Instance variables are owned by instances of the class. This means that for each object or instance of a class, the instance variables are different. Unlike class variables, instance variables are defined within methods.**

d. What is a UML sequence diagram used for?

**A sequence diagram is a type of interaction diagram because it describes how—and in what order—a group of objects works together. These diagrams are used by software developers and business professionals to understand requirements for a new system or to document an existing process.**

e. What is a lifeline in UML sequence diagrams?

**A lifeline represents an individual participant in a sequence diagram. A lifeline will usually have a rectangle containing its object name. If its name is "self", that indicates that the lifeline represents the classifier which owns the sequence diagram.**

2. More theory tasks

a. Multiple choice

i. In an inheritance relationship, the \_\_\_\_\_\_ is the general class.

1. Child class

2. Subclass

3. Superclass

4. Specialized class

ii. In an inheritance relationship, the \_\_\_\_\_\_ is the specialized class:

1. Superclass

2. Master class

3. Parent class

4. Subclass

iii. Let’s say we have two classes in our program: BankAccount and SavingsAccount. Which one of them would most likely be the subclass?

1. BankAccount

2. SavingsAccount

3. Neither of them

4. Both of them.

iv. Which one of the options you will use if you want to check whether an object is an instance of a class?

1. The instance operator

2. The is\_object\_of function

3. The isinstance function

4. There is not a way to check that at all.

v. Which one of the UML diagrams is a behavioral diagram?

1. Class diagram

2. Sequence diagram

3. Object diagram

4. Deployment diagram

vi. Which one of the UML diagrams is a structural diagram?

1. Use case diagram

2. State machine diagram

3. Activity diagram

4. Composite structure diagram

vii. In UML class diagrams, what does the notation \* mean.

1. Multiplication operation

2. Power of operation

3. Multiplicity 0..n

4. Multiplicity 0..1

b. True or false?

i. It is not possible to call a superclass’s \_\_init\_\_ method from a subclass’s \_\_init\_\_ method.

False

ii. A subclass never inherits any methods or attributes from the superclass.

False

iii. A superclass can inherit methods from subclass, if they have been denoted with pass\_to\_super function.

False

iv. In a subclass it is possible to have methods and attributes in addition to those that the subclass inherits from superclass.

True

v. In Python, multiple inheritance does not exist.

False

vi. Aggregation and composition shall never be used in UML class diagrams.

False

vii. Aggregation and composition mean exactly the same thing in UML class diagrams.

False

# File name: pet.py

# Author: Steve Hommy

# Description: Create a Pet Class

class Pet:

    def \_\_init\_\_(self, species, name, owner):

        self.\_\_species = species

        self.\_\_name = name

        self.\_\_owner = owner

    def set\_species(self):

        self.\_\_species = input("Give species for pet: ")

    def get\_species(self):

        return self.\_\_species

    def set\_name(self):

        self.\_\_name = input("Give name for pet: ")

    def get\_name(self):

        return self.\_\_name

    def set\_owner(self, owner):

        self.\_\_owner = owner

    def get\_owner(self):

        return self.\_\_owner

    def \_\_str\_\_(self):

        return f"""

        Species of the pet: {self.\_\_species}

        Name of the pet: {self.\_\_name}

        Owner of the pet: {self.\_\_owner}"""

# File name: student.py

# Author: Steve Hommy

# Description: Create a Student Class

from pet import Pet

class Student:

    def \_\_init\_\_(self, first\_name, last\_name, student\_ID):

        self.\_\_first\_name = first\_name

        self.\_\_last\_name = last\_name

        self.\_\_student\_ID = student\_ID

        self.\_\_pets = []

    def set\_first\_name(self):

        self.\_\_first\_name = input("Student first name: ")

    def get\_first\_name(self):

        return self.\_\_first\_name

    def set\_last\_name(self):

        self.\_\_last\_name = input("Student last name: ")

    def get\_last\_name(self):

        return self.\_\_last\_name

    def set\_student\_ID(self):

        self.\_\_student\_ID = input("Student ID: ")

    def get\_student\_ID(self):

        return self.\_\_student\_ID

    def add\_pets(self):