Northeastern University

CS 2100: Program Design and Implementation 1

Practice Quiz 3

Instructions

- Please put all of your answers on the answer sheet. Only the answer sheet will be graded.
- Do not begin the quiz until instructed to do so.
- You may use both sides of a sheet of paper up to 8.5"x11" for reference, but no other resources, including phones, computers, AI, headphones, and ear pods.
- You have until the end of the class period to complete the quiz.
- Students may not leave the classroom during the first 10 minutes of the quiz (except in case of emergency).
- Hand your completed answer sheet to an instructor before leaving the room.
- Talk to an instructor if you need to leave the room and reenter.

Please use the following code to answer the questions below:

```
class Course(ABC):
    """Abstract base class representing a course."""

    @abstractmethod
    def hold_lecture(self, attendee_ids: set[str]) -> None:
        """Does the activities involved in holding a lecture
        with the given attendees."""
        pass


class ComputerScienceCourse(Course):
    """A Computer Science course."""
    def __init__(self) -> None:
        self.school = "Computer Science"


class PDI(ComputerScienceCourse):
    """A Program Design and Implementation course."""

    def __init__(self, student_ids: Optional[set[str]] = None):
        """Initializes a PDI course with the given student IDs."""
        super().__init__()
        self.number = "2100"
        self.students = student_ids
        if student_ids is None:
            self.num_lectures_attended: dict[str, int] = {}
        else:
            if "" in student_ids:
                raise ValueError(
                    "Empty string is not a valid student ID")
            self.num_lectures_attended = \
                {student: 0 for student in student_ids}

    def hold_lecture(self, attendee_ids: set[str]) -> None:
        for attendee in attendee_ids:
            if attendee in self.num_lectures_attended:
                self.num_lectures_attended[attendee] += 1


class TestPDI(unittest.TestCase):
    def setUp(self) -> None:
        self.pdi_course = PDI({"s1", "s2", "s3"})

    def test_hold_lecture_with_two_students(self) -> None:
        self.pdi_course.hold_lecture({"s1", "s2"})
        self.assertEqual(
            self.pdi_course.num_lectures_attended,
            {"s1": 1, "s2": 1, "s3": 9999})

    def test_hold_lecture_with_full_attendance(self) -> None:
        self.pdi_course.hold_lecture({"s1", "s2", "s3"})
        self.assertEqual(self.pdi_course.num_lectures_attended["s1"], 1)
        self.assertEqual(self.pdi_course.num_lectures_attended["s2"], 1)
        self.assertEqual(self.pdi_course.num_lectures_attended["s3"], 9999)
```

Using objects (revisited from Quiz 1)

1. The argument to the `PDI` constructor is `student_ids: Optional[set[str]]`. Which of the following is *not* a value that fits the argument's type annotation?
   a. `{"s1", "s2", "s3"}`
   b. `None`
   c. `{"s1", None}`
   d. `{"s1", ""}`
2. Consider the following code:

   ```
   students = {"s1", "s2", "s3"}
   course = PDI(students)
   students.add("s4")
   ```

   Will `course.students` contain `"s4"` after this code?
   a. Yes, because `course.students` is an alias of `student_ids`.
   b. Yes, because `course.students` is a copy of `student_ids`.
   c. No, because `course.students` is an alias of `student_ids`.
   d. No, because `course.students` is a copy of `student_ids`.
3. Consider the following code:

   ```
   course1 = PDI({"s1", "s2", "s3"})
   course2 = PDI({"s3", "s4"})
   course1.hold_lecture({"s3"})
   ```

   After this code, what will be the value of `course2.num_lectures_attended["s3"]`?
   a. 0
   b. 1
   c. 2
   d. `None`
4. Consider the following code:

   ```
   def mystery() -> None:
   ```

```
        x = 5 + 3

    result = mystery()
    print(result is None)
```

What is the output of the print statement?

a. `True`
b. `False`
c. `None`
d. `TypeError`

Unit testing (revisited from Quiz 1)

5. The two provided tests each increment the lecture counts for "s1" and "s2". If we were to add a third test, would the lectures attended in the first two tests be counted in the third test? I.e., will the lecture counts in `self.pdi_course` *not* be reset to zero at the beginning of the new test?
   a. The lecture counts would not be reset to zero at the beginning of the new test because `self.pdi_course` is an attribute, which is shared between methods.
   b. The lecture counts would not be reset to zero at the beginning of the new test, but they would if we had replaced `setUp()` with `setUpClass()`.
   c. The lecture counts would be reset to zero at the beginning of the new test because `self.pdi_course` is local to each method, and not shared between methods.
   d. The lecture counts would be reset to zero at the beginning of the new test because `setUp()` would overwrite the existing `self.pdi_course` with a new instance of `PDI`.
6. How could we test that the `PDI` constructor raises a `ValueError` if `student_ids` contains an empty string?
   ```
   a. with self.assertRaises(ValueError):
          PDI({"s1"})
   b. self.assertEqual(PDI({"s1"}), ValueError)
   c. with self.assertRaises(ValueError):
          PDI({""})
   d. self.assertEqual(PDI({""}), ValueError)
   ```
7. Will `test_hold_lecture_with_full_attendance()` pass?
   a. Yes, because it is impossible to write a test that fails.
   b. Yes, because the first two `assertEqual()` statements pass.
   c. No, because the third `assertEqual()` statement does not pass.
   d. No, because the first test does not pass, so it will not run the second test.
8. Select the statement which is true:
   a. If all tests pass, then it is impossible for the source to contain bugs.

b. If we write tests with full coverage (i.e., every line of code in the source gets run at some point in a test), then it is impossible for the source to contain bugs.
c. If a test fails, it is possible that the bug is in the test, not the source code.
d. If the source code contains bugs, then it is impossible for all tests to pass.

Sets and dictionaries (revisited from Quiz 2)

9. What happens if the `attendee_ids` passed to `PDI`'s `hold_lecture()` contain an attendee who is not registered for the course?
   a. It will add the attendee to `self.num_lectures_attended` with a lecture count of `1`.
   b. It will add the attendee to `self.num_lectures_attended` with a lecture count of `0`.
   c. It will run, but that attendee's lecture count will not be affected.
   d. It will raise a `KeyError`.
10. What happens if there is a `student_id` in the `PDI` instance who is never in the set of `attendee_ids` passed to `hold_lecture()`, but they are in the original `student_ids` passed to the constructor?
    a. That student will not be in `self.num_lectures_attended`.
    b. That student will be in `self.num_lectures_attended`, mapped to a value of `0`.
    c. That student will be in `self.num_lectures_attended`, mapped to a value of `None`.
    d. `hold_lecture()` will raise a `KeyError`.
11. Is it possible for `hold_lecture()` to increment a student's lecture count twice in the same method call (i.e., without calling `hold_lecture()` a second time)?
    a. No, because `self.num_lectures_attended` is a dictionary, which cannot hold the same key (student ID) twice.
    b. Yes, because `self.num_lectures_attended` can have the same value twice, just not the same key.
    c. No, because every time a lecture is held, the values in `self.num_lectures_attended` are all overwritten with ones (or kept at zero).
    d. No, because the `student_ids` argument is a set, which cannot hold the same student twice.
12. Is it possible for two students to have the same lecture count?
    a. No, because `self.num_lectures_attended` is a dictionary, which cannot hold the same key (student ID) twice.
    b. Yes, because `self.num_lectures_attended` can have the same value twice, just not the same key.
    c. Yes, because the values in `self.num_lectures_attended` are all always zero.
    d. Yes, because the values in `self.num_lectures_attended` are all always either zero or one.

Abstract methods

13. Can we instantiate a `ComputerScienceCourse`?
    a. No, because it has an abstract method.
    b. Yes, because it has a constructor.
    c. Yes, because it has no abstract methods.
    d. Yes, because it has no concrete methods.
14. Can we instantiate a `PDI`?
    a. No, because it has an abstract method.
    b. No, because its superclass has an abstract method.
    c. Yes, because it has no abstract methods.
    d. Yes, because it has a concrete method.
15. Is it possible to have a fully concrete subclass of `Course` which does not have an implementation for `hold_lecture()`?
    a. Yes
    b. No, but it would be possible if we added some concrete methods to `Course`.
    c. No, but it would be possible if we removed the `@abstractmethod` annotation.
    d. No, but it would be possible if we changed `hold_lecture()` to return a value other than `None`.
16. Is this legal:

```
course: Course = PDI({"s1", "s2", "s3"})
course.hold_lecture({"s1", "s2"})
```

    a. No, because `PDI` inherits an abstract method, so it cannot be instantiated.
    b. No, because `PDI`'s `hold_lecture()` is abstract, so it cannot be called.
    c. No, because `Course`'s `hold_lecture()` is abstract, so it cannot be called.
    d. Yes

Inheritance

17. Is this legal:

```
course = PDI({"s1", "s2"})
print(course.school)
```

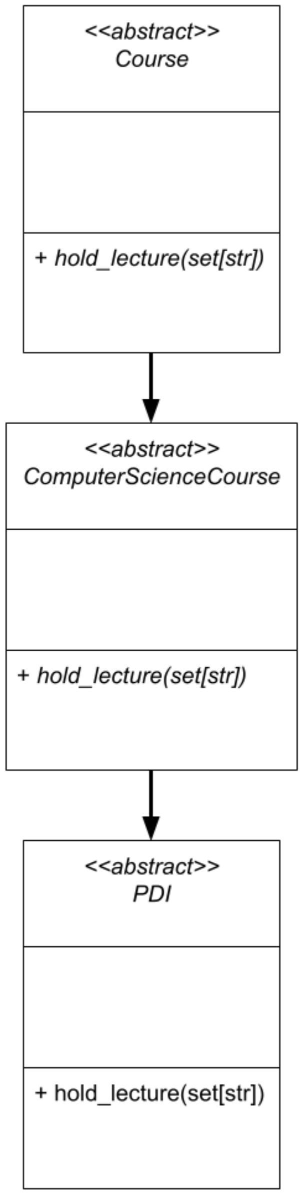
    a. Yes, because `PDI` inherits `self.school` from `ComputerScienceCourse`.
    b. Yes, because `PDI` inherits `self.school` from `Course`.
    c. No, because `PDI` does not have an attribute or property called `self.school`.
    d. No, because `PDI` overwrites `self.school` to be `None` by default.
18. Let's say there is a variable called `course` that holds an instance of a subclass of `ComputerScienceCourse`, but its type annotation is `ComputerScienceCourse`, not the subclass. Can I access `course.number` without any `MyPy` errors?

   a. No, because `ComputerScienceCourse` does not have an attribute or property called `self.number`.
   b. Yes, because `ComputerScienceCourse` inherits `self.number` from `PDI`.
   c. Yes, but only if that subclass has an attribute or property called `self.number`.
   d. Yes, but `course.number` will be None.

19. If we modify `ComputerScienceCourse`'s constructor, will `PDI`'s constructor also be automatically updated?
    a. No, because `PDI` overwrites the constructor inherited from `ComputerScienceCourse`.
    b. Yes, because `PDI` inherits the constructor from `ComputerScienceCourse`.
    c. Yes, because `PDI`'s constructor calls `ComputerScienceCourse`'s constructor.
    d. No, because `PDI`'s constructor calls `ComputerScienceCourse`'s constructor.
20. If we add an implementation of `hold_lecture()` to `ComputerScienceCourse`, will `PDI`'s `hold_lecture()` also be automatically updated?
    a. No, because `PDI` overwrites the `hold_lecture()` inherited from `ComputerScienceCourse`.
    b. Yes, because `PDI` inherits `hold_lecture()` from `ComputerScienceCourse`.
    c. Yes, because `PDI`'s `hold_lecture()` calls `ComputerScienceCourse`'s `hold_lecture()`.
    d. No, because `PDI`'s `hold_lecture()` calls `ComputerScienceCourse`'s `hold_lecture()`.

Interpreting UML diagrams

21. Consider this UML diagram:



What part of it does *not* match the provided code?

a. It says that `Course` is abstract.
b. It says that `ComputerScienceCourse` is abstract.
c. It says that `PDI` is abstract.
d. It says that `Course` has no attributes (other than those inherited from the `object` class).

22. What else in the UML diagram does *not* match the provided code?

a. It says that `ComputerScienceCourse` is a subclass of `Course`.
b. It says that `PDI` is a subclass of `ComputerScienceCourse`.
c. It says that `PDI` is a subclass of `Course`.

    d. It says that `ComputerScienceCourse` overwrites the `hold_lecture()` inherited from Course.

23. What is wrong with the UML diagram (other than that it doesn't match the provided code)?
    a. The arrows are pointing the wrong way.
    b. It has no attributes in any of the classes.
    c. It needs a concrete class in order to be valid.
    d. It has both italicized (abstract) and un-italicized (concrete) versions of `hold_lecture()`.
24. Which of the following statements is true?
    a. UML diagrams are visual depictions of classes and their relationships.
    b. UML diagrams specify method signatures and the method implementation details.
    c. UML diagrams cannot depict abstract and concrete classes in the same diagram.
    d. UML diagrams cannot depict abstract and concrete methods in the same class.

Identifying privacy issues

25. Consider an in-class polling software (like Poll Everywhere). What is *not* a piece of information that is being shared?
    a. Student homework grades
    b. Student id numbers (or a similar identifier)
    c. Whether the student participated each day
    d. What the student selected as their answers
26. Considering the in-class polling software, who is the subject of the information?
    a. The user interface
    b. The student
    c. The course material
    d. The polling software engineers
27. Considering the in-class polling software, who is *not* a likely recipient of the information?
    a. The instructor
    b. The student's family members
    c. Other students looking over their shoulder
    d. TAs and other staff who manage grades
28. Which of the following statements is true?
    a. We must always minimize the number of recipients of information at all costs.
    b. We must always minimize the number of unintended recipients of information at all costs.
    c. We should only share pieces of information that are necessary.
    d. We should only share pieces of information that are optional.