CSC207H Winter 2017 Worksheet: Python to Java

Welcome to the first lab for CSC207! If you don't manage to finish by the end of your lab, please complete it on your own time.

Icebreaker

Your TA will introduce themselves, and then help you introduce yourself to your neighbours. The course will go much better if we create a learning community, and part of that involves getting to know people. Find out:

1. Three (3) neighbours' names
2. Why they are enrolled in CSC207
3. Something fun about their home town

After that, choose up to three of your neighbours to do the remainder of this lab with. Make sure everyone has read through each step before you attempt it!

Setting up IntelliJ

Linux is case sensitive. Throughout the course, use exactly the capitalization we use.

To start IntelliJ, go to the command line and type "`idea`".

1. When you first run IntelliJ, it will ask you in which folder you want to store your projects. Use the default, `/IdeaProjects/`.
2. You will usually create a new project for each lab and assignment. Do so now: go to *File* → *New* → *Project*. You may have to tell IntelliJ where Java is located. If so, select Java 1.8 from the Project SDK drop-down list.

   If it is not available, click new JDK and go to /local/packages/jdk1.8.0. Once the project SDK has been set, click next. Then click next again. Set the project name to Lab1 and press Finish. It may take a few seconds for the project to appear in the menu on the left side of your window.

CSC207H Winter 2017 Worksheet: Python to Java

Using IntelliJ, translate the following code fragments from Python to their closest equivalents in Java. (There is no direct equivalent to doctest, so you don't need to translate those. You'll learn how to unit test in Java later in the course.)

Translate the first three in a single class called `Fib`. In IntelliJ, create a new class by right-clicking on `src` and selecting *New → Java Class.*

Remmber that there are no top-level functions in Java, and that the most direct equivalent is a `static` method.

1.
```
if __name__ == "__main__":
    print('This is a string')
```

2.
```
def fib_iterative(n: int) -> int:
    """Returns the nth Fibonacci number.

    >>> print(fib_iterative(6))
    8
    """
    answer = 0
    current = 1
    for i in range(n):
        next = answer + current
        answer = current
        current = next
    return answer
```
To verify you did it correctly, modify your main method to call `fib_iterative` and print the results.

3.
```
def fib_recursive(n: int) -> int:
    """Returns the nth Fibonacci number.

    >>> print(fib_recursive(4))
    3
    """

    if n == 0:
        return 0
    if n == 1:
        return 1
    else:
        return fib_recursive(n - 1) + fib_recursive(n - 2)
```
To verify you did it correctly, modify your main method to call `fib_recursive` and print the results.

4.
```
class Student:
    """A Student with a name and an id."""

    def __init__(self: Student, id: int, name: str=None) -> None:
        """Create a student named name with UTORid ID.
        """

        self.name = name
        self.id = id


    def __eq__(self: Student, other: object) -> bool:
        """Returns True iff other is a Student and has an identical id to this student.

        >>> uncle = Student("Bob", 111)
        >>> bob = Student("Bob", 111)
        >>> print(bob == uncle)
        True
        """

        return isinstance(other, Student) and other.id == self.id
```

To properly translate method `__eq__`, you should have this as your method header:

```
public boolean equals(Object other) {
```

Variable `other` may refer to an object of any type. To check whether it refers to a `Student`, use the `instanceof` operator, which produces a Boolean. Use this expression in an `if` statement:

```
other instanceof Student
```

You will also need to tell Java that `other` is actually a `Student`. This is done with a *typecast*:

```
Student otherStudent = (Student) other;
```

Once you have done the typecast, you can look at the contents of `otherStudent` and do the necessary comparison.

CSC207H Winter 2017 Worksheet: Python to Java

5.
```
class Cohort:
    """A Cohort is a list of Students.
    """

    def __init__(self: Cohort, year: int = 0) -> None:
        """Initializes a Cohort of Students for a year.
        """

        self.year = year
        self.students = []

    def add(self: Cohort, student: Student) -> None:
        """Adds the student to the Cohort.
        """

        self.students.append(student)


    def contains(self: Cohort, student: Student) -> bool:
        """Returns True if student is in this Cohort.
        """

        return student in self.students
```

Here, create an array of `Students`. You'll need to specify a maximum size; pick any number you like. (Notice that this means you have a maximum number of students in a Cohort; later in the course, you'll learn how Java deals with variable-length sequences.)

Also create a main method that creates a couple of students and compares them.

CSC207H Winter 2017 Worksheet: Python to Java

6.
```
from typing import List

class Employee:
    #implementation not given

class Department:
    """A Department consists of a list of Employees as well as a list of Cohorts.
    """

    def __init__(self: Department,
                 staff: List[Employee] = None,
                 cohorts: List[Cohort] = None) -> None:
        """A department with a staff and a list of cohorts.
        """

        if staff is None:
            self.staff = []
        else:
            self.staff = staff

        if first is None:
            self.cohorts = []
        else:
            self.cohorts = cohorts

    def contains_student(self: Department, cohort_index: int, student: Student) -> bool:
        """Returns whether student is in the cohort found at cohort_index in this department.

        >>> cohort = Cohort(2017)
        >>> cohort.add(Student("Bob", 111))
        >>> dept = Department([], [cohort])

        >>> print(dept.contains_student(Student("Bob", 111)))
        True
        """

        return self.cohorts[cohort_index].contains(student)
```

Java doesn't have default parameter values. Instead, either multiple versions of a constructor method are created (with different parameters), or instance variables are given default values when they are declared. We suggest that you start by assuming that they'll be supplied, and only after you get that working should you add any extra constructors to explore this.