CSC207/B07H1 F 2013 Midterm Test
Duration — 50 minutes
Aids allowed: none

Student Number: |___|___|___|___|___|___|___|___|___|___|

Last Name: _____ **First Name:** _____

Lecture Section: Daytime

---

*Do **not** turn this page until you have received the signal to start.*
(Please fill out the identification section above, **write your name on the
back of the test**, and read the instructions below.)
Good Luck!

This midterm consists of 3 questions on 8 pages (including this one). *When
you receive the signal to start, please make sure that your copy is complete.*

- Legibly write your name and student number on this page. Legibly
  write your name on the back page of this exam.

- If you use any space for rough work, indicate clearly what you want
  marked.

- In all programming questions you may assume all input is valid.

- You do not need to write Javadocs.

- You may use a pencil; however, work written in pencil will not be
  considered for remarking.

1: _____/10

2: _____/ 8

3: _____/23

TOTAL: _____/41

Question 1.   [10 MARKS]

Alice and Bob are interested in poetry, and they put together a collection of poems with one word per line by L. Eigner. They share a subversion repository located at http://poetry.utoronto. ca/eigner. They begin with two files in the repository, eigner.txt and 14march1983.txt, with the following contents:

eigner.txt
```
poems
with
but
one
word
per
line
```

14mar1983.txt
```
sky
clouds
far
snowy
corners
```

Q: Bob just bought a new laptop and wants to get a local copy of the repository on it. What command does he need to execute?

svn checkout http://poetry.utoronto.ca/eigner

Suppose now that Alice creates a new file in her local copy:

7nov1989.txt
```
tree
phonepole
bird
```

Q: What sequence of command(s) do Alice and Bob need to issue so that Bob also has a copy of this file? For each command, make sure to state who needs to execute it, Bob or Alice.

Alice: svn add 7nov1989.txt
Alice: svn commit -m "Adding another poem"
Bob: svn update

Now that Alice and Bob share all three files, they both decide to edit the file 7nov1989.txt as follows:

7nov1989.txt
Alice:
```
tree
Phonepoles
bird
```

7nov1989.txt
Bob:
```
tree
phone pole
bird
```

Suppose Alice now executes this command:

```
svn commit -m "fixed a typo"
```

Q: What is the second line of the file `7nov1989.txt` **in**
- Alice's copy   Phonepoles

- Bob's copy   phone pole

- the latest version in the repository `Phonepoles`

Suppose Bob now executes this command:

```
svn commit -m "revised line 2"
```

Q: What is the second line of the file `7nov1989.txt` **in**
- Alice's copy   Phonepoles

- Bob's copy   phone pole

- the latest version in the repository `Phonepoles`

Q: Give the sequence of steps that Alice and Bob should take to make sure they both have the same version of all three files, with `"phone pole"` **as the second line of** `7nov1989.txt.`

[These can be described in English, rather than by listing commands.]

Bob: `svn update`

Bob: resolve conflict so that second line is phone pole

Bob: `svn commit -m "changed second line to phone pole"`

Alice: `svn update`

Question 2.   [8 MARKS]

Consider the following definitions.

```java
public class Animal {                    public class Mammal extends Animal {
    public Animal() {                        public Mammal() {
    }                                            super();
}                                            }
                                         }
public interface Winged {                public interface FourLegged {
    public void fly();                       public void walk();
}                                        }
public class Horse extends Mammal implements FourLegged {
    public Horse() {
        super();
    }
    public void walk() {
        System.out.println("One, two, three, four.");
    }
}
```
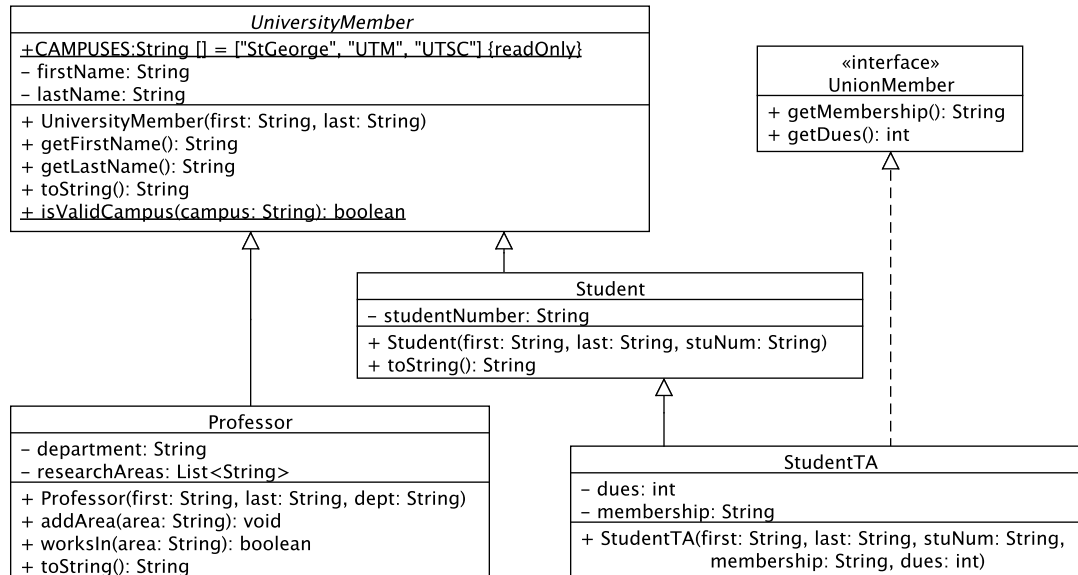
Q: Which of the following pieces of Java code will compile without errors (in addition to the above definitions)? <u>Circle all that apply.</u>

<u>*1.*</u> Mammal mammal = new Mammal();

<u>*2.*</u> Animal animal = new Mammal();

  3. Mammal mammal = new Animal();

  4. Winged winged = new Winged();

<u>*5.*</u> FourLegged fourLegged = new Horse();

  6. FourLegged fourLegged = new Mammal();

  7. public class Bat extends Mammal implements Winged {

```java
        public Bat() {
            super();
        }
        public void talk() {
            System.out.println("I'm a flying mammal.");
        }
    }
```

<u>*8.*</u> public class Bird extends Animal implements Winged {

```java
        public Bird() {
            super();
        }
        public void fly() {
            System.out.println("I'm flying!");
        }
    }
```

Question 3. [23 MARKS]

```
┌───────────────────────────────────────────────────┐
│                 UniversityMember                    │
├───────────────────────────────────────────────────┤
│ +CAMPUSES:String [] = ["StGeorge", "UTM", "UTSC"] {readOnly} │
│ − firstName: String                                 │
│ − lastName: String                                  │
├───────────────────────────────────────────────────┤
│ + UniversityMember(first: String, last: String)     │
│ + getFirstName(): String                            │
│ + getLastName(): String                             │
│ + toString(): String                                │
│ + isValidCampus(campus: String): boolean            │
└───────────────────────────────────────────────────┘
```

```
┌─────────────────────────────┐
│        «interface»          │
│        UnionMember          │
├─────────────────────────────┤
│ + getMembership(): String   │
│ + getDues(): int            │
└─────────────────────────────┘
```

```
┌───────────────────────────────────────────────────┐
│                     Student                         │
├───────────────────────────────────────────────────┤
│ − studentNumber: String                             │
├───────────────────────────────────────────────────┤
│ + Student(first: String, last: String, stuNum: String) │
│ + toString(): String                                │
└───────────────────────────────────────────────────┘
```

```
┌─────────────────────────────────────────────┐
│                  Professor                    │
├─────────────────────────────────────────────┤
│ − department: String                          │
│ − researchAreas: List<String>                 │
├─────────────────────────────────────────────┤
│ + Professor(first: String, last: String, dept: String) │
│ + addArea(area: String): void                 │
│ + worksIn(area: String): boolean              │
│ + toString(): String                          │
└─────────────────────────────────────────────┘
```

```
┌───────────────────────────────────────────────────┐
│                    StudentTA                        │
├───────────────────────────────────────────────────┤
│ − dues: int                                         │
│ − membership: String                                │
├───────────────────────────────────────────────────┤
│ + StudentTA(first: String, last: String, stuNum: String, │
│             membership: String, dues: int)          │
└───────────────────────────────────────────────────┘
```

Part (a) [5 MARKS]

Add a class StudentTA (a *student* who is a *teaching assistant*) to the UML diagram above. Note that all teaching assistants are UnionMembers. In StudentTA, based on the UML diagram, include any necessary instance variables and methods.

Part (b) [5 MARKS]

Implement the class StudentTA.

```java
class StudentTA extends Student implements UnionMember {

    private int dues;
    private String membership;

    public StudentTA(String first, String last,
            String stuNum, String membership, int dues) {
        super(first, last, stuNum);
        this.membership = membership;
        this.dues = dues;
    }
    public String getMembership() {
        return membership;
    }
    public int getDues() {
        return dues;
    }
}
```

Part (c) [5 MARKS]

Complete the implementation of the class `UniversityMember` below, according to the UML diagram. Recall that the UML keyword `readOnly` indicates that the field is a constant.

```java
public abstract class UniversityMember {
    private String firstName;
    private String lastName;
```

```java
    public static final String [] CAMPUSES = {"StGeorge", "UTM", "UTSC"};
```

```java
    public UniversityMember(String first, String last) {
        firstName = first;
        lastName = last;
    }

    public String getFirstName() {
        return firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public String toString() {
        return lastName + ", " + firstName;
    }
```

```java
    public static boolean isValidCampus(String campus) {
        return Arrays.asList(CAMPUSES).contains(campus);
    }
```

```java
}
```

Part (d)   [2 MARKS]

Implement `UnionMember` according to the UML diagram.

```java
public interface UnionMember {
    public String getMembership();
    public int getDues();
}
```

Part (e)   [6 MARKS]

Implement `Professor` according to the UML diagram. The method `worksIn` returns whether the given `area` is one of this `Professor`'s research areas. The method `toString` should, in addition to the last and first name, include the department this `Professor` works at.

```java
publi class Professor extends UniversityMember {
    private String department;
    private List<String> researchAreas;

    public Professor(String first, String last, String department) {
        super(first, last);
        this.researchAreas = new ArrayList<String>();
        this.department = department;
    }

    public void addArea(String area) {
        researchAreas.add(area);
    }

    public boolean worksIn(String area) {
        return researchAreas.contains(area);
    }

    public String toString() {
        return super.toString() + ", " + department;
    }
}
```

Last Name: _____   First Name: _____

```
public class Arrays:
  static <T> List<T> asList(T... a):
    Returns a fixed-size list backed by the specified array.

public class ArrayList<E> implements List<E>:
  boolean add(E e):
    Appends the specified element to the end of this list.
  void add(int index, E element):
    Inserts the specified element at the specified position in this list.
  boolean contains(Object o):
    Returns true if and only if this list contains the specified element.
  E get(int index):
    Returns the element at the specified position in this list.
  int indexOf(Object o):
    Returns the index of the first occurrence of the specified element
    in this list, or -1 if this list does not contain the element.
  boolean isEmpty():
    Returns true if and only if this list contains no elements.
  E remove(int index):
    Removes the element at the specified position in this list.
  E set(int index, E element):
    Replaces the element at the specified position in this list with
    the specified element.
  int size():
    Returns the number of elements in this list.
```

SVN commands

```
svn add [PATH]
svn checkout URL
svn commit [PATH] -m MESSAGE
svn list [PATH]
svn status [PATH]
svn update [PATH]
```

Total Marks = 41

End of Solutions