UNIVERSITY OF TORONTO
SCARBOROUGH

FALL 2016 EXAMINATIONS

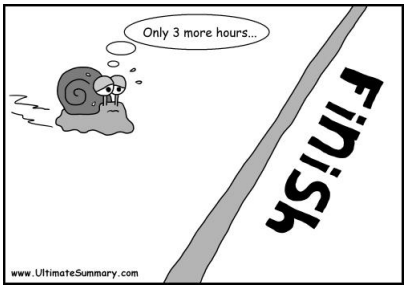
CSC C01H
Duration — 3 hours

No Aids Allowed

Student Number: _____

Last Name: _____

First Name: _____



---

*Do **not** turn this page until you have received the signal to start.*
(In the meantime, please fill out the identification section above, and read the instructions below *carefully*.)

This final examination consists of 7 questions on 19 pages (including this one). When you receive the signal to start, please make sure that your copy of the examination is complete. Answer each question directly on the examination paper, in the space provided.

There are two, clearly marked pages in this exam booklet that you can (carefully!) detach, for your convenience.

Be aware that concise, well thought-out answers will be rewarded over long rambling ones. Also, unreadable answers will be given zero (0) so write legibly.

# 1:	_____/ 18
# 2:	_____/ 6
# 3:	_____/ 10
# 4:	_____/ 18
# 5:	_____/ 15
# 6:	_____/ 15
# 7:	_____/ 18
TOTAL:	_____/100

Good Luck!

Question 1. Object Oriented Software Design I [18 MARKS]

Consider the Java implementation given in Figure 1. Use the space below to provide an implementation, which is functionally equivalent to that in Figure 1, but which follows Object Oriented Design Principles.

Question 1. (CONTINUED)

Question 2. Object Oriented Software Design II [6 MARKS]

Consider the Java implementation given in Figure 2. Use the space below to provide an example usage of classes `Triangle` and `EquilateralTriangle`, which illustrates the problem with the Object Oriented Design of the given implementation.

YOU MAY REMOVE THIS PAGE

```
public class Library {

private Map<String, Book> books;
private Map<String, AudioBook> audioBooks;

public Library() {
  books = new HashMap<>();
  audioBooks = new HashMap<>();
}
public void loanItem(String isbn) {
  if (books.containsKey(isbn)) {
    books.get(isbn).checkOutBook();
  }
  if (audioBooks.containsKey(isbn)) {
    audioBooks.get(isbn).download();
  }
}
public void returnItem(String isbn) {
  if (books.containsKey(isbn)) {
    books.get(isbn).returnBook();
  }
  if (audioBooks.containsKey(isbn)) {
    audioBooks.get(isbn).expire();
  }
}
public void add(List<Object> items) {
  String isbn;
  for (Object item: items) {
    if (item instanceof Book) {
      isbn = ((Book)item).getBookInfo()
          .getIsbn();
      books.put(isbn, (Book) item);
    }
    if (item instanceof AudioBook) {
      isbn = ((AudioBook)item).getBookInfo()
        .getIsbn();
      audioBooks.put(isbn, (AudioBook) item);
    }
}}
```

```
public class BookInfo {

  private String isbn;
  private String title;
  // ... etc ...

  public String getIsbn() {
    return isbn;
  }
  // ... etc ...
}
```

```
public class Book {
  private BookInfo bookInfo;
  private boolean onLoan;

  public Book(BookInfo bookInfo) {
    this.bookInfo = bookInfo;
    onLoan = false;
  }
  public BookInfo getBookInfo() {
    return bookInfo;
  }
  public void checkOutBook() {
    onLoan = true;
  }
  public void returnBook() {
    onLoan = false;
  }
  public boolean isOnLoan() {
    return onLoan;
  }
}
```

```
public class AudioBook {
  private static int downloadLimit;
  private BookInfo bookInfo;

  public AudioBook(BookInfo bookInfo, int limit) {
    downloadLimit = limit;
    this.bookInfo = bookInfo;
  }
  public BookInfo getBookInfo() {
    return bookInfo;
  }
  public void download() {
    downloadLimit --;
  }
  public void expire() {
    downloadLimit ++;
  }
  public boolean limitReached() {
    return downloadLimit == 0;
  }
}
```

Figure 1: Object Oriented Design I

YOU MAY REMOVE THIS PAGE

```
public class Triangle {

  private double sideA;
  private double sideB;
  private double angle;

  public Triangle(double sideA, double sideB, double angle) {
    this.sideA = sideA;
    this.sideB = sideB;
    this.angle = angle;
  }
  public double area() {
    return sideA * sideB * Math.sin(angle) / 2;
  }

  // ... ALL GETTERS AND SETTERS ...

  public double getSideA() {
    return sideA;
  }
  public void setSideA(double sideA) {
    this.sideA = sideA;
  }
  public double getSideB() {
    return sideB;
  }
  public void setSideB(double sideB) {
    this.sideB = sideB;
  }
  public double getAngle() {
    return angle;
  }
  public void setAngle(double angle) {
    this.angle = angle;
  }
}
```

```
public class EquilateralTriangle extends Triangle {

  public EquilateralTriangle(double side) {
    super(side, side, 60);
  }
}
```

Figure 2: Object Oriented Design II

YOU MAY REMOVE THIS PAGE

```
public class Person {

  private String firstName;
  private String lastName;
  private int neededHoursOfSleep;

  public Person(String firstName, String lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
    neededHoursOfSleep = 8;
  }
  public String firstName() {
    return firstName;
  }
  public String lastName() {
    return lastName;
  }
  public void setNeededHoursOfSleep(int hours) {
    neededHoursOfSleep = hours;
  }
  public boolean hadEnoughSleep(int hours) {
    return hours >= neededHoursOfSleep;
  }
  public String greet(int hoursOfSleep) {
    if (hadEnoughSleep(hoursOfSleep)) {
      return String.format("Hello, I am %s %s. How are you?", firstName, lastName);
    } else {
      return "Grumble grumble grumble";
    }
  }
}
```

```
public class Student extends Person {

  private String studentNumber;

  public Student(String firstName, String lastName, String studentNumber) {
    super(firstName, lastName);
    this.studentNumber = studentNumber;
  }
  public String studentNumber() {
    return studentNumber;
  }
  @Override
  public boolean hadEnoughSleep(int hours) {
    return false;
  }
```

Figure 3: Verification II

YOU MAY REMOVE THIS PAGE

```
public class Gossip {

  private List<Secret> secrets;

  public Gossip() {
    secrets = new ArrayList<>();
  }

  public void hear(Secret secret) {
    secrets.add(secret);
  }

  public List<Secret> tell() {
    return new ArrayList<>(secrets);
  }
}
```

```
public class Secret {
  private String secret;
  private int importance;

  public Secret(String secret,
      int importance) {
    this.secret = secret;
    this.importance = importance;
  }

  public String getSecret() {
    return secret;
  }

  public int getImportance() {
    return importance;
  }
}
```

```
public class Liar {

  private List<Secret> secrets;

  public Liar() {
    secrets = new ArrayList<>();
  }

  public void hear(Secret secret) {
    secrets.add(secret);
  }

  public List<Secret> tell() {
    List<Secret> lies = new ArrayList<>();
    for (Secret secret: secrets) {
      lies.add(new Secret(secret.getSecret()
                          + "\nAnd she told him about it!!!",
               secret.getImportance() * 2));
    }
    lies.add(new Secret("Complete nonsense", 100));
    return lies;
  }
}
```

```
public class BrokenPhone {

  private Gossip gossip;
  private Liar liar;

  public BrokenPhone(Gossip gossip, Liar liar) {
    this.gossip = gossip;
    this.liar = liar;
  }

  public void discoverSecret(String secret, int importance) {
    Secret wrongSecret = new Secret(secret, importance + 5);
    gossip.hear(wrongSecret);
    liar.hear(wrongSecret);
  }
}
```

Figure 4: Verification III

Question 3. Project Management [10 MARKS]

You are managing a team of three developers — Alice, Bob, and Chi. They all hate working together on a task, and you decided that it is not worth the effort trying to make them do so.

Your sprints are 5 days in length (Mon – Fri), and working hours are 10 a.m. to 6 p.m. Each developer "burns" 6 dev-hours per day (there's always coffee and Facebook). You have the following tasks (on the right) in your backlog, sorted in non-increasing priority (i.e., highest priority first). Of course, things don't go as planned. Bob suddenly decided to spend Wed and Thu at a hackathon in Seattle. To make matters worse, it took 6 dev-hours to complete task #1.

Task	Cost (dev-hours)
3	8
2	12
1	2
6	9
4	2
5	11

Show a complete burn-down chart for this sprint. You can use the table below to work out your plan(s). The table will not be graded.

Task							

Question 4. Verification and Validation I [18 MARKS]

Consider the following simple Java method, with its description.

```
/**
 * Returns the average of all elements in the given list.
 * @param posInts the list of positive integers, to take the average of
 * @return the average of all elements in ints, if it is not empty
 * @throws NoElementsException if ints is empty
 * @throws InvalidElementException if ints contains negative elements
 */
public static double average(List<Integer> posInts) {
  if (posInts.isEmpty()) {
    throw new NoElementsException();
  }
  double sum = 0;
  for (Integer integer: posInts) {
    if (integer < 0) {
      throw new InvalidElementException(integer.toString());
    }
    sum += integer;
  }
  return sum / posInts.size();
}
```

Part (a) [3 MARKS]

Does the above implementation follow Design by Contract or Defensive Design? Explain.

Part (b) [15 MARKS]

We will now perform **Black box** JUnit testing of the above method `average`. In the Java code below, fill in **as much as you think is necessary** to provide a thorough test suite. Take a note of the `private` helper at the end of the file.

```
public class BlackboxTest {
  public static final double EPSILON = 0.001;









  @Before
  public void setUp() {









  @Test
  public void test









  @Test
  public void test
```

```
@Test
public void test
```

```
@Test
public void test
```

```
@Test
public void test
```

```
  @Test
  public void test
```

```
  @Test
  public void test
```

```
  @Test
  public void test
```

```
private String errorMessage(double expected, double actual, List<Integer> lst) {
    return String.format("Expected %.2f on list %s, but got %.2f", expected, lst, actual);
}
```

Question 5. Verification and Validation II [15 MARKS]

Consider the Java implementation of classes `Person` and `Student` in Figure 3. Below we provided complete testing of class `Person`.

```
public class PersonTest {
  Person alice, bob;

  @Before
  public void setUp() throws Exception {
    alice = new Person("Alice", "Anderson");
    bob = new Person("Bob B", "Bobson");
  }
  @Test
  public void testPerson() {
    // full test of constructor
  }
  @Test
  public void testFirstNameOne() {
    String actual = alice.firstName();
    assertEquals(errorMessage("Alice", actual), "Alice", actual);
  }
  @Test
  public void testFirstNameMany() {
    String actual = bob.firstName();
    assertEquals(errorMessage("Bob B", actual), "Bob B", actual);
  }
  @Test
  public void testLastName() {
    String actual = bob.lastName();
    assertEquals(errorMessage("Bobson", actual), "Bobson", actual);
  }
  @Test
  public void testHoursOfSleep() {
    alice.setNeededHoursOfSleep(5);
    String sleepMessage = " be enough hours of sleep.";
    assertTrue("5 should" + sleepMessage, alice.hadEnoughSleep(5));
    assertTrue("6 should" + sleepMessage, alice.hadEnoughSleep(6));
    assertFalse("4 should not" + sleepMessage, alice.hadEnoughSleep(4));
  }
  @Test
  public void testGreetEnoughSleep() {
    String actual = alice.greet(10);
    String expected = "Hello, I am Alice Anderson. How are you?";
    assertEquals(errorMessage(expected, actual), expected, actual);
  }
  @Test
  public void testGreetNotEnoughSleep() {
    String actual = alice.greet(7);
    String expected = "Grumble grumble grumble";
    assertEquals(errorMessage(expected, actual), expected, actual);
  }
  String errorMessage(String expected, String actual) {
    return String.format("Expected %s, but got %s", expected, actual);
  }}
```

Question 5. (CONTINUED)

Your task is to test class `Student`. In the Java code below, fill in **as much as you think is necessary** to provide a thorough test suite. You do not need to provide tests for the constructor.

Question 6. Verification and Validation III [15 MARKS]

Consider Java implementation that describes the spreading of rumours, given in Figure 4. Your task is to thoroughly test the method `discoverSecret` of class `BrokenPhone`.

```
public class BrokenPhoneTest {









  @Before
  public void setUp() {
```

Question 6. (CONTINUED)

Question 7. Source Control [18 MARKS]

Part (a) [10 MARKS]

You are managing a team of five developers: Aki, Bo, Cai, Datta, and Elsa. It is December 7, and currently Aki and Bo are working on developing FeatureX, and Cai and Datta are working on developing FeatureY, and Elsa is updating a user guide.

In your current backlog, FeatureX has a higher priority than FeatureY, and the user guide has the highest priority — you have a client waiting for it. According to the plan, both features should be finished by the end of the day on December 11 and the user guide should be done by the end of December 10.

Provide a careful sketch of an effective use of version control software to aid in development in the above scenario. Clearly indicate all dates, branch points, synchronizations, commits, merge points.

On the morning of December 9, you find out that a third party software package that you have been using in your development has just released a security patch, and you need to incorporate this change into your code. This task takes up half of a work-day of two developers. Provide an updated sketch.

Part (b) [8 MARKS]

Recall your Exercise 2, in which you submitted two versions of some Python code: one for Python 2 and another for Python 3.

Suppose you use two branches, `python2` and `python3`, to maintain the two corresponding versions of a file named `submission.py`. The file is in your GitHub repository called `yourID-Exercise2`, in the directory `e2`.

In order to grade your exercise, I created a tester file called `test.py`. To grade your Python 2 version of the code, I need to run the command `python2 test.py` and, similarly, to grade the Python 3 version, I need to run `python3 test.py`. I need to run this command in a directory that contains both your submission and the tester file, and the run creates a file `result.txt` in that same directory.

Suppose I want to place two files, both called `result.txt`, that contain the outputs of my two test runs, in the two corresponding branches in your repository. Assume that the file `test.py` is currently in the directory `/home/anya/c01/e2/marking` on my computer, and that I do not currently have your repo on it.

Give a sequence of **git** commands that I need to issue in order to grade your exercise as described above. At the end of the process, the two result files should be in your GitHub repository, in directory `e2`, on the two corresponding branches.

Total Marks = 100