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

# 1: _____/ 8  
# 2: _____/13  
# 3: _____/16  
TOTAL: _____/37
Question 1.  [8 marks]
Paola and Rohit are on the CSSU executive and are running a hackathon. They share a subversion repository located at http://markus.cdf.toronto.edu/extra-curricular/hackathon. Paola has put files registered.txt and scores.txt into the master repository.

Part (a)  [2 marks]
Rohit has logged in to CDF for the first time. What commands must he execute in order to create a directory called cssu and check out a local copy of the repository into it?

```bash
mkdir cssu
cd cssu
svn checkout http://markus.cdf.toronto.edu/extra-curricular/hackathon
```

Part (b)  [2 marks]
After a few days of work on the project, Rohit and Paola each have the latest revision of the master repository in their local copies and there are no local changes. Rohit now does some work on his local copy, making changes. He runs `svn status` and sees:

```
rohit$ svn status
?
M scores.txt
```

Rohit wants to share his changes with Paola, so he executes this command:

```
rohit$ svn commit -m "entered round 1 scores and added a budget file"
```

But when Paola tries to see the new file budget.txt, she runs into trouble:

```
paola$ svn update
Updating ".":
U scores.txt
Updated to revision 5.

paola$ cat budget.txt
no such file or directory
```

Modify Rohit’s and/or Paola’s commands above so that Paola will be able to cat the file budget.txt. If you add a new command, be sure it is clear who is issuing it.
Insert an "svn add budget.txt" command before the commit

---

**Part (c) [2 marks]**

Suppose more work happens, and that again Rohit and Paola each have the latest revision of the master repository in their local copies and there are no local changes. Here is the current content of scores.txt:

<table>
<thead>
<tr>
<th>Paola: scores.txt</th>
<th>Rohit: scores.txt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Team D 510</td>
<td>Team D 510</td>
</tr>
<tr>
<td>Mighty Mice 600</td>
<td>Mighty Mice 600</td>
</tr>
<tr>
<td>Snafu 107</td>
<td>Snafu 107</td>
</tr>
</tbody>
</table>

Now, they both edit their local copies of scores.txt as follows:

<table>
<thead>
<tr>
<th>Paola: scores.txt</th>
<th>Rohit: scores.txt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Team D 510</td>
<td>Team D 510</td>
</tr>
<tr>
<td>Mighty Mice 600</td>
<td>Mighty Mice 600</td>
</tr>
<tr>
<td>Weasleys 440</td>
<td>Snafu 107</td>
</tr>
<tr>
<td>Snafu 107</td>
<td>ABC 22</td>
</tr>
</tbody>
</table>

Give a sequence of svn commands that Paola and Rohit can issue so that their changes become part of the repository and so that they both have the most recent version of scores.txt. For each command, state who needs to execute it, Paola or Rohit.

Note: These changes do not create a conflict in subversion. They are not incompatible.

Paola: svn commit -m 'added Weasleys team'
Paola: svn update
Rohit: svn commit -m 'added ABC team'
Rohit: svn update

---

**Part (d) [2 marks]**

What is the contents of scores.txt in the master repository now?

Team D 510
Mighty Mice 600
Weasleys 440
Snafu 107
ABC 22

Total Pages = 8 Page 3 OVER...
Question 2. [13 marks]

The following code compiles without errors:

```java
public class Car implements Drivable {
    public void drive() {
        System.out.println("pbrr");
    }
    public void honk() {
        System.out.println("awooga");
    }
}
public interface Drivable {
    public void drive();
}
public class Porsche extends Car {
    public void honk() {
        System.out.println("beep beep");
    }
}
public class Ford extends Car {
    public void drive() {
        System.out.println("sputter");
    }
}
```

Let’s consider some segments of code that might go in a `main` method somewhere in the same package as our code above.

(a) [2 marks] Will the following code compile and run without error? □ yes □ no

```java
Car c = new Drivable();
c.drive();
```

If yes, show the output. If no, explain:

NO, because you cannot instantiate an interface.

(b) [2 marks] Will the following code compile and run without error? □ yes □ no

```java
Ford f = new Car();
f.drive();
((Car) f).drive();
```

If yes, show the output. If no, explain:

NO, because variable f must refer to some kind of Ford object, i.e., a Ford or some descendant of class Ford.
(c) [2 marks] Will the following code compile and run without error?  
☐ yes  ☐ no

```
Car c = new Porsche();
c.honk();
((Porsche) c).honk();
```

If yes, show the output. If no, explain:

Yes. The output is:
beep beep
beep beep

(d) [2 marks] What code could you add to class Car that would make the following line not even compile?
```
Car c = new Car();
```

You may either describe this new code or write it.

If we add any constructor that has arguments, we will no longer have the no-argument constructor that Java defines for us. So the "new" above will fail.

(e) [1 mark] Is any method overloaded? If so, name one that is. If not, say "None".

None

(f) [2 marks] On the code we provided, change class Car to express that every Car must have a service method that takes no arguments and returns nothing, but that class Car itself can’t provide a method body for it.

Add this line to class Car:
```
    public abstract void service();
```
Make class car abstract:
```
    public abstract class Car:
```

(g) [2 marks] Describe any changes that must be made to the other three class/interface definitions in order to make them compile, now that the changes in part (f) have been made. Write your answer here:

We must implement method service() within class Porsche and class Ford.
Question 3. [16 marks]

A Bunch has a capacity and holds items in it. Method add is used to put items into the Bunch, in order. If the Bunch is full when add is called, one of the existing items in the Bunch is chosen at random, and is replaced by the new one. Method get returns one of the items, randomly chosen, but does not modify the Bunch. The following main method demonstrates how to use a Bunch:

```java
public static void main(String[] args) {
    Bunch<String> b = new Bunch<String>(3); // This Bunch has capacity to hold 3 items.
    b.add("coding");
    b.add("is");
    System.out.println(b); // Prints "coding is"
    b.add("great");
    System.out.println(b); // Prints "coding is great"
    String sample = b.get();
    System.out.println(sample); // Prints "coding", or "is", or "great"
    System.out.println(b); // Prints "coding is great" as before.
    b.add("fun");
    System.out.println(b); // Prints "fun is great", or "coding fun great", or
    // "coding is fun"
}
```

Implement class Bunch, using an array to store its contents. You may define additional instance variables as required. Ensure that the main method above compiles, runs, and produces the output shown. You should assume the main method is in class Bunch.

Reminder: You must use an array. You may not use an ArrayList, any kind of List, or any class in the Java Collections Framework.
Solution:

```java
public class Bunch<T> {

    private T[] contents;
    private int capacity;
    private int size;
    private Random r;

    public Bunch(int capacity) {
        this.contents = (T[]) new Object[capacity];
        this.capacity = capacity;
        this.size = 0;
        this.r = new Random();
    }

    public void add(T item) {
        if (size < capacity) {
            this.contents[size] = item;
            this.size += 1;
        } else {
            int where = this.r.nextInt(capacity-1);
            this.contents[where] = item;
        }
    }

    public T get() {
        if (this.size == 0) {
            return null;
        } else {
            int where = this.r.nextInt(size-1);
            return this.contents[where];
        }
    }

    @Override
    public String toString() {
        String answer = "";
        for (int i = 0; i < this.size; i++) {
            answer += this.contents[i].toString() + " ";
        }
        return answer;
    }
}
```
Short Java built-in class descriptions

class ArrayList<E> is not used on this test!

class Random:
    // An instance of this class is used to generate a stream of pseudorandom numbers.
    Random()
    Creates a new random number generator.
    nextInt(int bound)
    Returns an int value between 0 (inclusive) and the specified value (exclusive).

SVN commands

svn add [PATH]
svn checkout URL
svn commit [PATH] -m MESSAGE
svn delete [PATH]
svn list [PATH]
svn status [PATH]
    Meaning of output:
    A: item is scheduled for addition.
    D: item is scheduled for deletion.
    M: item has been modified.
    ?: item is not under version control.
svn update [PATH]
svn resolved

Simplified description of Unix commands

mkdir NAME
    make a new directory called NAME
ls
    list contents of the current directory
cd SUBDIRECTORY
    change directories to SUBDIRECTORY
cat FILE
    display contents of FILE