Computer Sciences 302
Exam 2 Information & Sample Exam

Below you'll find information about the second midterm exam and sample exam questions. This sample is intended to be *similar* in length and difficulty of the actual exam. For a list of topics for exam 2, see the course web pages. Note the following logistics about the exam:

- The second midterm exam will be composed of four parts:
  1. Coding a Complete Program.
  2. Using Pre-built Classes.
  3. Coding an Instantiable Class.
  4. Using Arrays and `ArrayList`s.
- You will be evaluated on the correctness of your Java code *including syntax*. Though we'll forgive forgetting a '`;`' or '`}`', we will make deductions if you habitually make syntax mistakes. In general, the closer your code is to correct Java syntax the higher your score will be.
- The exam is designed to be about 100 minutes in length, and we expect most students will use this time. Like the first exam, we'll allow a total exam time of two hours afterwhich you'll be required to turn in your exam.
- It is ok to use the following abbreviations `S.o.pln()` for `println` and `S.o.p()` for `print`.
- You will *not* need to comment your code.
- For the actual exam we'll provide descriptions of common methods for select classes from the Java API such as the `String` and `ArrayList` classes where needed. For this sample exam you should use the Java API documentation and your textbook to search for useful methods.
- You may assume that non-`null` references are passed to methods. That is, you do not need to check that parameters are `null`.
- You may not use textbooks, notes, neighbors, calculators, or other electronic devices on the exam.
- **Your UW ID is required to take the exam.** Before the exam begins, drop off your UW ID at the front of the room. Sit in columns with an empty seat or an aisle to your right and left. When you're done, turn in your exam, and pick up your UW ID.

We suggest you do the following with this sample exam:

- Do this sample exam on paper as a timed exam. Set aside at least 100 minutes to complete the entire exam and pick a location free of distractions where you can focus on taking the exam.
- After you've completed your exam, verify your answers in Eclipse (i.e., write programs to ensure your answers work as expected). Note, except for question Part II #2, *we will not be posting a solution to this exam* though we will answer questions about particular programming concepts covered in this sample exam.
- Consider forming a study group to compare your answers to other members of your group.

Part I Coding a Complete Program.

Use good programming practices to **write a complete Java program** used to grade eggs. The program is used to grade one egg at a time until the user chooses to quit the program. The user provides for each egg, its weight in ounces and its color (as a character: 'w' for white, 'b' for brown, 'o' for other).The program displays each egg's grade based on these specifications:

- Jumbo grade is for white eggs with a weight greater than or equal to 30 ounces.
- Extra Large grade is for white eggs with a weight greater than or equal to 27 ounces but less than that of Jumbo grade.
- Large grade is for white eggs with a weight greater than or equal to 24 ounces but less than that of Extra Large grade.
- Commercial grade is for all other eggs.

If input of the incorrect type is entered your program should display an error message and allow the user to re-enter input.

Part II Using Pre-built Classes.

1.) A `String` object, named `email`, contains an email address that you may assume is in the standard form: name@domain. **Write a code fragment** that separates out the name and domain into two strings. For example, if the email address is "ernie@sesame_st.pbs" then one string will contain only "ernie" and the other will contain only "sesame_st.pbs".

2.) You are asked to write a code fragment that implements an object-oriented alarm system. The alarm system is composed of one or more sensors that detect intruders, remote controls that allow the user to control the system, and bells that sound the alarm. Three classes have been implemented: `Sensor`, `Remote`, and `Bell` as described below.

`Sensor` class has a zero parameter constructor and methods:

```
boolean sense()      //returns true iff an intruder is detected
void    activate()   //turns the sensor on
void    deactivate() //turns the sensor off
```

`Remote` class constructor and methods:
constructor takes a single integer parameter that is the remote's code
The following methods return either the remote's code or 0

```
int disarm()   //returns the remote's code iff the system is to be disarmed
int panic()    //returns the remote's code iff a panic is signaled
```

`Bell` class has a zero parameter constructor and method:

```
void activate(int volume) //sets the bell to the specified volume
                          //0 is off, 100 is maximum volume
```

Write a code fragment that instantiates and activates three sensors, instantiates one remote using the code 1234, and instantiates one bell. Your program then continually monitors the sensors and the remote until a disarm signal is detected. If an intruder or panic signal is detected your code should set the bell to a volume of 100. If a disarm signal is detected your program should set the bell volume to 0 and deactivate all of the sensors.

Part III Coding an Instantiable Class.

Implement the instantiable class described below using public vs. private, instance vs. class members, and constants vs. variables in a manner that corresponds with good object-oriented programming practices. *You may assume that parameters will be passed valid values.* Write a class, named `ShippingContainer`, that contains:

- The quantity of container objects constructed (`long`)
- A unique container identification number (`long`)
- The weight of the container when empty (`double`)
- The weight of the load in the container (`double`)
- The maximum load weight for the container (`double`)

- A constructor that is passed the maximum load weight and the empty container's weight used to initialize an empty container. The constructor also increments the quantity of containers objects each time it is called.
- A method named `getQuantity` that returns the quantity of container objects.
- A method named `getNumber` that returns the container's unique number.
- A method named `getWeight` that returns the total weight of container and its load.
- A method, named `load`, that is passed a parameter of type `double` and returns a `boolean` value. The load weight of the container is increased by the amount of the parameter and the method returns `true`. However, if the load weight would increased beyond the maximum, the load weight is not changed and the method returns `false`.
- A method named `isEmpty` that accepts no parameters. The method returns the `boolean` value `true` if the container's load weight is 0, and `false` otherwise.
- A method named `isFull` that has no parameters. The method returns the `boolean` value `true` if the container's load weight is the maximum, and `false` otherwise.
- A method, named `unload`, that is passed a parameter of type `double` and returns a `boolean` value. The load weight of the container is decreased by the amount the parameter and the method returns `true`. However, if the load weight would decreased below 0, the load weight is not changed and the method returns `false`.

Part IV Using Arrays and `ArrayLists`.

1.) **Write a method**, named `countNegs`, that is passed an array of `doubles` and returns a count of the values in that array that are negative.

2.) **Write a code fragment** that reverses the contents of an `ArrayList` of `Strings`, named `names`. For example, if `names` initially contains Jim, Bob, Sue, after the code fragment completes it should contain in this order Sue, Bob, Jim.

3.) **Write a method** that is passed an array of `String`s and returns a `String` that is composed of the last letter of each `String` in the array. For example, if the passed array contains "Cat", "TREE", "soda", the method returns the string "tEa".