# NEW ENGLAND INSTITUTE OF TECHNOLOGY

# Information Technology Department

# Quest #1

**Data Structures and Algorithms (SE 394)**

**Due: Week #1**

# 

# Joshua LeGoff

# Developer

# 10/6/2015

# Date

# George Saban

# Instructor

# Requirements:

### *Instructions:*

* According to some research, industry values documentation, and excellent written and oral communication skills. The purpose of this part of the class is to encourage you to gain these skills.
* Backup your work to your USB drive for this material may come out as part of your examination.
* Make a copy of this entire document and add your work into it.
* Submit to Blackboard at the same link where you got this document.
* Points will be a deducted if submitted on the wrong place, or if these instructions are not followed.
* [You can earn a maximum of 40 points for this lab. Refer to the syllabus for late point deductions.]

### *Problem Statement*

|  |  |
| --- | --- |
| **Part I:** | **– this will help you familiarize with Java language and the Eclipse IDE** |

Create your own problem that can be elegantly solved using a class or classes. In other words, come up with a problem that is solvable using class or classes.

* 1. In *200 words* or more, write below the detailed description of the problem.
  2. And then solve the problem by using a class or classes. Implement your solution using the Java language.

– **OR** – Alternatively, if you could not come up with your own problem then you may choose one of the problems below.



Create class **Creature** with class member legs, eyes, hands, height, weight, and skin color. Provide setters and getters. In the main() method instantiate an instance of Creature object and set its members, and then display the members using its getter methods.

– **OR** – another alternative problem you may choose to solve the problem below.



Create class **Employee** with class member name, id, age, salary, phone, and address. Provide setters and getters. In the main() method instantiate an instance of Employ object and set its members, and then display the members using its getter methods.

### *Hints*

* Review the examples discussed by the instructor and are now posted in Blackboard.

# Code Development:

**package** myPackage;

**import** javax.swing.\*;

**public** **class** Creature {

**int** \_legs;

**int** \_eyes;

**int** \_hands;

**float** \_height;

**float** \_weight;

String \_skinColor;

**public** **int** get\_legs() {

**return** \_legs;

}

**public** **void** set\_legs(**int** \_legs) {

**this**.\_legs = \_legs;

}

**public** **int** get\_eyes() {

**return** \_eyes;

}

**public** **void** set\_eyes(**int** \_eyes) {

**this**.\_eyes = \_eyes;

}

**public** **int** get\_hands() {

**return** \_hands;

}

**public** **void** set\_hands(**int** \_hands) {

**this**.\_hands = \_hands;

}

**public** **float** get\_height() {

**return** \_height;

}

**public** **void** set\_height(**float** \_height) {

**this**.\_height = \_height;

}

**public** **float** get\_weight() {

**return** \_weight;

}

**public** **void** set\_weight(**float** \_weight) {

**this**.\_weight = \_weight;

}

**public** String get\_skinColor() {

**return** \_skinColor;

}

**public** **void** set\_skinColor(String \_skinColor) {

**this**.\_skinColor = \_skinColor;

}

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

Creature creature = **new** Creature();

creature.set\_legs(Integer.*parseInt*(JOptionPane.*showInputDialog*("How many legs does the creature have?")));

creature.set\_eyes(Integer.*parseInt*(JOptionPane.*showInputDialog*("How many eyes does the creature have?")));

creature.set\_hands(Integer.*parseInt*(JOptionPane.*showInputDialog*("How many hands does the creature have?")));

creature.set\_height(Float.*parseFloat*(JOptionPane.*showInputDialog*("How tall is the creature in inches?")));

creature.set\_weight(Float.*parseFloat*(JOptionPane.*showInputDialog*("How much does the creature weigh?")));

javax.swing.JOptionPane.*showMessageDialog*(**null**,

"The creature has: " + creature.get\_eyes() + " eyes, "

+ creature.get\_legs() + " legs, "

+ "and " + creature.get\_hands() + " hands."

+ "The weighs " + creature.get\_weight() + "lbs and is "

+ creature.get\_height() + "inches tall.");

}

}

# Testing:

What is testing for? Every program you wrote, you are responsible for unit testing it. The first thing to do, once you have a program that “basically works”, is to try to break it. Try to feed your program input(s) in the hope of getting it to misbehave. By “hope” means that the challenge here is to find as many errors as possible, so that you can fix the errors before anybody else finds them. If you go into this exercise with the attitude that “my program works, and I don’t make errors!”, then you won’t find many bugs, and you will feel bad when you do find one or when someone finds one. You’d be playing head games with yourself! The right attitude when testing is, “I’ll break it! I’m smarter than any program--even my own!”

Feed (or try) a few such “problematic” inputs to your program and try to figure out in how many ways you can get it to misbehave. Can you get the program to crash? Testing is a very important part of game development, and can actually be fun. You may input data that is not “sensible”. A program ideally catches all errors, not just the sensible ones--this will make your program resilient against “strange input.”

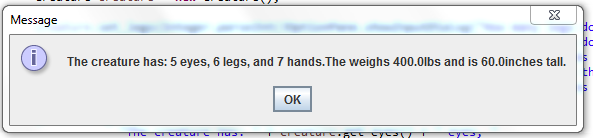
As a goal, you would like the test to exercise every statement in your program, at least once. Test both positive and negative scenario. One example of positive testing is--if you have an input that requires an integer value, would it work if you give it the numeral 7? Moreover, as an example of negative testing: what happens if you give the previous program a string input of “seven” (instead of the numerical 7)? Identify if your test results are “Passed” or “Failed”. Use the table below, and add at least five test cases of your own!

[Type at least 5 test cases and what was the result? Did the result pass or fail? Do a negative test case too!]

|  |  |  |
| --- | --- | --- |
| **Test #** | **Test Description** | **Result**  **(Passed/Failed)** |
| 1 | Example **positive** testing: Typed the number 7 as input for health field. | Passed |
| 2 | Example **negative** testing: Type the word “seven” as input for health field. | Failed |
|  |  |  |
| 1 | Typed six for integer input for number of legs | Failed |
| 2 | Typed four hundred for float input for weight | Failed |
| 3 | Typed sixty for float input for height | Failed |
| 4 | Typed number 6 for input for number of legs | Passed |
| 5 | Typed number 60 for float input for height | Passed |
|  | *Add as many rows as needed.* |  |

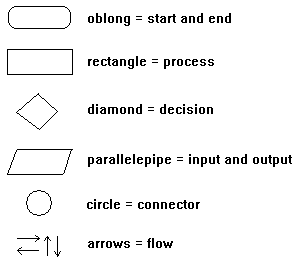
# Production Deployment:

[Paste all your own final screens in this section.] Make sure your output screen shot is readable, **magnify** if necessary so the instructor can easily read it. A sample magnified output is shown below; replace this with your own.



|  |  |
| --- | --- |
| **Part II:** | **– this will help you understand logic, problem solving, and algorithm.** |

**Draw a flowchart** of your algorithm (step-by-step solution) to sort inputted three numbers A, B, and C and then display the sorted numbers as your output. **Then code in Java** your solution. Your flowchart should use the most basic flowchart symbols below:



### *Hints*

* Review the examples discussed by the instructor and are now posted in Blackboard.

# Diagram:

# C:\Users\Joanie\Dropbox\SortAlgorithmFlowChart.jpgCode Development:

**package** myPackage;

**import** javax.swing.JOptionPane;

**public** **class** MySort {

**public** **int**[] sortInt(**int**[] ar){

**for** (**int** i = (ar.length - 1); i >= 0; i--)

{

**for** (**int** j = 1; j <= i; j++)

{

**if** (ar[j-1] > ar[j])

{

**int** temp = ar[j-1];

ar[j-1] = ar[j];

ar[j] = temp;

}

}

}

**return** ar;

}

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

**int** arrayCount = Integer.*parseInt*(JOptionPane.*showInputDialog*("How many numbers will you be entering?"));

**int**[] arrayToSort = **new** **int**[arrayCount];

**for**(**int** i = 0; i < arrayCount; i++)

{

arrayToSort[i] = Integer.*parseInt*(JOptionPane.*showInputDialog*("Enter number."));

}

MySort sorter = **new** MySort();

arrayToSort = sorter.sortInt(arrayToSort);

**for**(**int** i = 0; i < arrayToSort.length; i++)

{

System.***out***.println(arrayToSort[i]);

}

}

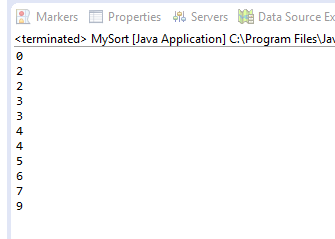
}

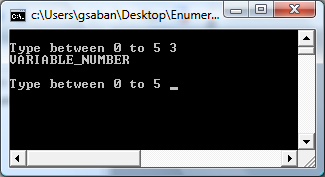
# Testing:

[Type your test cases and what was the result? Did the result passed or failed? Do a negative test case too!]

|  |  |  |
| --- | --- | --- |
| **Test #** | **Test Description** | **Result**  **(Passed/Failed)** |
| 1 | Entered letters for first input box. | Failed |
| 2 | Entered numbers for first input box. | Passed |
| 4 | Entered letters for second input boxes. | Failed |
| 5 | Entered numbers for second input boxes. | Passed |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  | *Add as many rows as needed.* |  |

# Production Deployment:

sure your output screen shot is readable, **magnify** if necessary so the instructor can easily read it. A sample magnified output is shown below; replace this with your own.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| GRADING RUBRIC | | | | |
| Grading Criteria | 3  **Exceeds**  *Excellent*  Epic Wow | 2  **Meets**  *Satisfactory*  O.K. | 1  **Partially Meets**  *Below Expectations*  Not Yet | 0  **Does Not Meet**  *Unacceptable*  Fail |
| **Coding** | +10- Code is excellent, comments are added, and different techniques were used. | +7-Code is O.K., and program works. | +4-Code works, but still needs improvement. | Unfortunately, no coding. |
| **Diagram** | +10-Excellent diagram. | +7-Diagram meets requirements. | +4-Diagram does not make sense, or needs improvement. | 0-Unfortunately, no table relationships. |
| **Testing** | +10-Test cases were excellent, and provided more test cases than required. | +7-Provided valid tests, and meets minimum test case requirements. | +4-Test needs improvement, did not make sense, and did not meet minimum test case requirements. | Unfortunately, no testing. |
| **Completeness** | +10-Completed all the required work and added more examples. | +7-Completed all the work required. | +4-Partially completed the work required. | Unfortunately, did not complete the work required. |
| **Late** | Excellent, you submitted it before the deadline. | -5, for submitting after the deadline. | -10, for submitting several weeks after the deadline. | -20, unfortunately, for submitting very late. |