# NEW ENGLAND INSTITUTE OF TECHNOLOGY

# Information Technology Department

# Quest #1

**Design Patterns (GDS 393)**

**Due: Week #1**

# 

# Jesse Golembiesky

# Developer

# 8/5/17

# Date

# George Saban

# Instructor

# Requirements:

### *Instructions:*

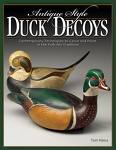
* According to some research, industry values documentation, and excellent written and oral communication skills. The purpose of this part of the class is to encourage you to gain these skills.
* Backup your work to your USB drive for this material may come out as part of your examination.
* Make a copy 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 deducted if submitted on the wrong place, or if these instructions are not followed.
* [You can earn a maximum of 30 points for this lab. Refer to the syllabus for late point deductions.]

### *Problem Statement*

Create your own class using Java. Moreover, add at least 5 member variables and its associated member methods.

– **OR** – Alternatively, if you could not come up with your own class then you may solve the problem below.

Create class Duck with class member variables legs, eyes, wing span, and color. Provide setters and getters and class member methods fly(), display(), and quack(). In the main() method instantiate a duck object and set its member variables, then exercise (meaning call) all its member methods.

[](http://images.google.com/imgres?imgurl=http://www.freefoto.com/images/01/08/01_08_52---Duck_web.jpg&imgrefurl=http://www.freefoto.com/preview/01-08-52?ffid=01-08-52&h=600&w=400&sz=42&hl=en&start=1&usg=__nAl-0ElTLba2sfRTuAKOtjUmJn8=&tbnid=z4oL9BJpidVewM:&tbnh=135&tbnw=90&prev=/images?q=duck&gbv=2&hl=en) [](http://images.google.com/imgres?imgurl=http://www.rubberducks.org.uk/rubber-duck-1.jpg&imgrefurl=http://www.rubberducks.org.uk/about.php&h=1224&w=1632&sz=527&hl=en&start=2&usg=__2f5k0kZTYOCHMcWFAI2_BSAIfCI=&tbnid=ewaOj6pgTWkTgM:&tbnh=113&tbnw=150&prev=/images?q=duck&gbv=2&hl=en) [](http://images.google.com/imgres?imgurl=http://www.freefoto.com/images/01/08/01_08_18---Duck_web.jpg&imgrefurl=http://www.freefoto.com/preview/01-08-18?ffid=01-08-18&h=400&w=600&sz=82&hl=en&start=3&usg=__yl4SydAwnI3m7dJtJBZioAIUk7s=&tbnid=ZuNel9xssnxXZM:&tbnh=90&tbnw=135&prev=/images?q=duck&gbv=2&hl=en) [](http://images.google.com/imgres?imgurl=http://www.foxchapelbooks.com/images/books/high-res/1-56523-298-4.jpg&imgrefurl=http://www.fandfwoodcarving.com/duckand.htm&h=1500&w=1159&sz=493&hl=en&start=35&usg=__LX3rx6EkS_GJW9Dwe2Vz5xwHbtA=&tbnid=9540c_Tx2nPsbM:&tbnh=150&tbnw=116&prev=/images?q=decoy+duck&start=20&gbv=2&ndsp=20&hl=en&sa=N)

Note: There is no cin in Java. The equivalents for it is:

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));  
 *then you can use method like readLine() to read in the input:*  
 int input = Integer.parseInt(br.readLine());

Or, use Scanner:

Scanner inputReader = new Scanner(System.in);

String inputString = "";

inputString = inputReader.next();

# Code Development:

**package** mypackage;

**public** **class** Duck {

**int** legs;

**int** eyes;

**float** wingspan;

String color;

**public** Duck(){}

**public** **int** getLegs() {

**return** legs;

}

**public** **void** setLegs(**int** legs) {

**this**.legs = legs;

}

**public** **int** getEyes() {

**return** eyes;

}

**public** **void** setEyes(**int** eyes) {

**this**.eyes = eyes;

}

**public** **float** getWingspan() {

**return** wingspan;

}

**public** **void** setWingspan(**float** wingspan) {

**this**.wingspan = wingspan;

}

**public** String getColor() {

**return** color;

}

**public** **void** setColor(String color) {

**this**.color = color;

}

**public** **void** fly(){System.***out***.println("Look Mom, I'm flying!");}

**public** **void** display(){System.***out***.println("This is one very dapper duck.");}

**public** **void** quack(){System.***out***.println("Quack! Quack!");}

}

**package** mypackage;

**import** java.util.\*;

**public** **class** runDuck {

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

Scanner keyboard = **new** Scanner(System.***in***);

Duck ducky = **new** Duck();

System.***out***.println("What color is Mr. Ducky?");

ducky.setColor(keyboard.nextLine());

System.***out***.println("How many eyes does Mr. Ducky have?");

ducky.setEyes(keyboard.nextInt());

System.***out***.println("How many legs does Mr. Ducky have?");

ducky.setLegs(keyboard.nextInt());

System.***out***.println("What is Mr. Ducky's wingspan?");

ducky.setWingspan(keyboard.nextFloat());

ducky.fly();

ducky.display();

ducky.quack();

}

}

# UML Diagram:

# 

# 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.”

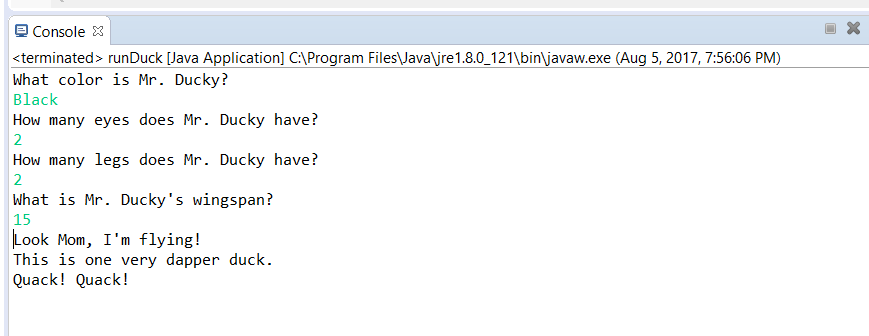
As a goal, you would like the test to exercise every statement in your program, at least once. Test both positive and negative scenario. One example of positive testing is--if you have an input that requires an integer value, would it work if you give it the numeral 7? Moreover, as an example of negative 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!

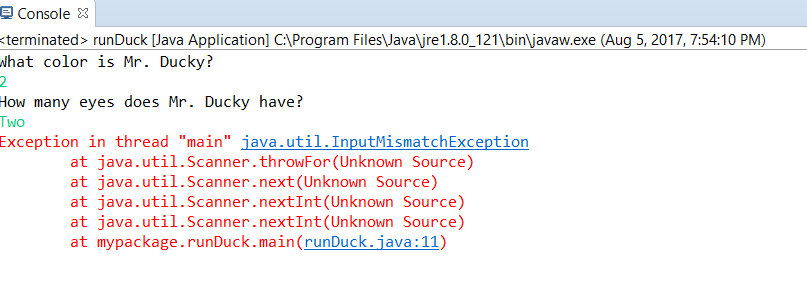
[Type at least 5 test cases and what was the result? Did the result pass or fail? Do a negative test case too!]

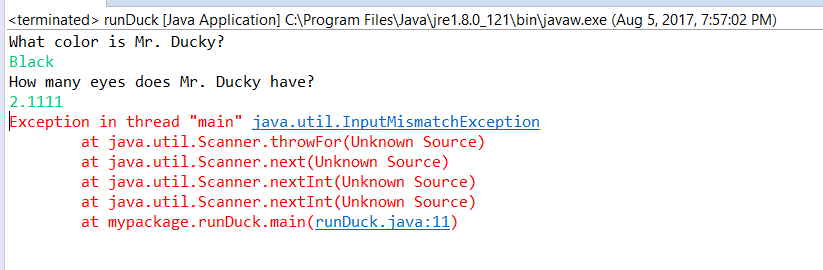
|  |  |  |
| --- | --- | --- |
| **Test #** | **Test Description** | **Result**  **(Passed/Failed)** |
| 1 | Positive: Typed in a string for color | Passed |
| 2 | Positive: Typed in an integer for legs | Passed |
| 3 | Positive: Typed in an integer for eyes | Passed |
| 4 | Positive: Typed in an integer for wingspan | Passed |
| 5 | Negative: Typed in an integer for color | Passed |
| 6 | Negative: Typed in a string for eyes | Failed |
| 7 | Negative: Typed in a decimal for eyes | Failed |
|  |  |  |
|  | *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.







|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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. | +8-Code is O.K., and program works. | +5-Code works, but still needs improvement. | Unfortunately, no coding. |
| **Output** | +5-Outputs are correct, and provided additional output cases. | +3-Output meets requirement and is readable. | +1-There is output, but not readable, and/or needs improvement. | Unfortunately, no output. |
| **Testing** | +5-Test cases were excellent, and provided more test cases than what is required. | +3-Provided valid tests, and meets minimum test case requirements. | +1-Test needs improvement, did not make sense, and did not meet minimum test case requirements. | Unfortunately, no testing. |
| **Documentation** | +5-Excellent documentation. No misspelling, well formatted, and correct syntax. | +3-Documentation meets requirements. | +1-Documentation has misspelling, or syntax issues, or not clear, or needs improvement. | Unfortunately, no documentation. |
| **Diagram** | +5-Excellent diagram. Correct symbols were used and labeled correctly. | +3-Diagram meets requirements. | +1-Diagram does not make sense, or needs improvement. | Unfortunately, no diagram. |
| **Late** | Excellent, you submitted it before the deadline. | -5, unfortunately for submitting after the deadline. | -10, unfortunately for submitting several weeks after the deadline. | -15, unfortunately, for submitting very late. |