

CS 272 Midterm Exam - Programming

When you are finished, copy the code from all classes created below into a Word document (named LastNameFirstName_Midterm.docx) and label each part. Then copy the console output from Eclipse and paste it before the code. Finally, create exports (zip files) for each of your Eclipse projects. You should submit **1 Word document and 2 zip files** (LastNameFirstName_CS272_Midterm_Part01.zip and LastNameFirstName_CS272_Midterm_Part02.zip) for full credit on this assignment.

Part 01 – Abstract Class, Interface and Lambda Expressions

[40 points total]

Please download and open the Eclipse project named: **CS272_Midterm_Part01.zip**

Note: the only written code in the package is a file named GamingDemo.java, which you can use to test the final version of your project. Please implement the following components for this assignment:

PerformanceInfo: This interface provides the following method(s):

1. **public abstract double calculateTFlops()** – This method returns an double representing how many Tera FLOPS (Floating Point Operations per Second) a GamingConsole can perform. Tera is the scientific unit meaning trillion. Essentially, it is a measure of how many trillion multiplications/additions can be done in a second and is a proxy for the performance of a computer (in this case GamingConsole). For all consoles, this is calculated as:

$$\text{TeraFLOPs} = (\text{GPU speed in MHz} * \text{number of cores} * 2) / 1,000,000$$

FYI: GPU stands for Graphics Processing Unit (like a CPU specifically for graphics). The number of cores refers to the number of parallel operations that can be done in one clock cycle. We multiply by 2 because a GPU is capable of multiplying and adding concurrently.

2. **public abstract boolean isBackwardCompatible()** – This method returns an boolean representing whether a GamingConsole is backward compatible. For this project, only Xbox is backward compatible (return true). Nintendo and Playstation are NOT backward compatible (return false).
3. **public abstract boolean isDockable()** – This method returns an boolean representing whether a GamingConsole can be docked or undocked (used with a battery instead of power outlet). For this project, only Nintendo is dockable (return true). Playstation and Xbox are NOT dockable (return false).

GamingConsole: This is the abstract parent (base) class of the other three. Here are the specifications:

1. Create member variables for **name** (String), **GPU** (int), **cores** (int) and **price** (double).
2. Create a protected constructor to facilitate code reuse among derived classes
3. Create accessors/mutators for the member variables *as specified in class diagram below*
4. Create an equals() and hashCode() method to compare all member variables for equality.

Nintendo: One of the concrete child (derived) classes of GamingConsole. Here are the specifications:

1. Member variable is **batteryLife** (double). Represents the battery life in hours.
2. When implementing the PerformanceInfo interface:
 - *calculateTFlops* is calculated as indicated above.
 - *isBackwardCompatible* returns false
 - *isDockable* returns true
3. Override the toString() method that displays all fields in the following format:
Be sure that TFLOPS calls the method calculateTFlops() and is formatted to 1 decimal place.
Ensure that Backward Compatible and Dockable states Yes or No (instead of true or false)

Nintendo [Switch, GPU=768MHz, Cores=256, Price=\$299.00, Battery Life=2.5hrs, TFLOPS=0.4, Backward Compatible=No, Dockable=Yes]

4. See class diagram below for other specifications.

Playstation: One of the concrete child (derived) classes of GamingConsole. Here are the specifications:

1. Member variable is **virtualReality** (String). Represents the virtual reality headset name.
2. When implementing the PerformanceInfo interface:
 - *calculateTFlops* is calculated as indicated above.
 - *isBackwardCompatible* returns false
 - *isDockable* returns false
3. Override the toString() method that displays all fields in the following format:
Be sure that TFLOPS calls the method calculateTFlops() and is formatted to 1 decimal place.
Ensure that Backward Compatible and Dockable states Yes or No (instead of true or false)

Playstation [4, GPU=800MHz, Cores=1152, Price=\$399.00, VR Headset=Playstation VR, TFLOPS=1.8, Backward Compatible=No, Dockable=No]

4. See class diagram below for other specifications.

XBox: One of the concrete child (derived) classes of GamingConsole. Here are the specifications:

1. Member variable is **physicalMedia** (String). Represents the drive included with Xbox.
2. When implementing the PerformanceInfo interface:
 - *calculateTFlops* is calculated as indicated above.
 - *isBackwardCompatible* returns true
 - *isDockable* returns false
3. Override the toString() method that displays all fields in the following format:
Be sure that TFLOPS calls the method calculateTFlops() and is formatted to 1 decimal place.
Ensure that Backward Compatible and Dockable states Yes or No (instead of true or false)

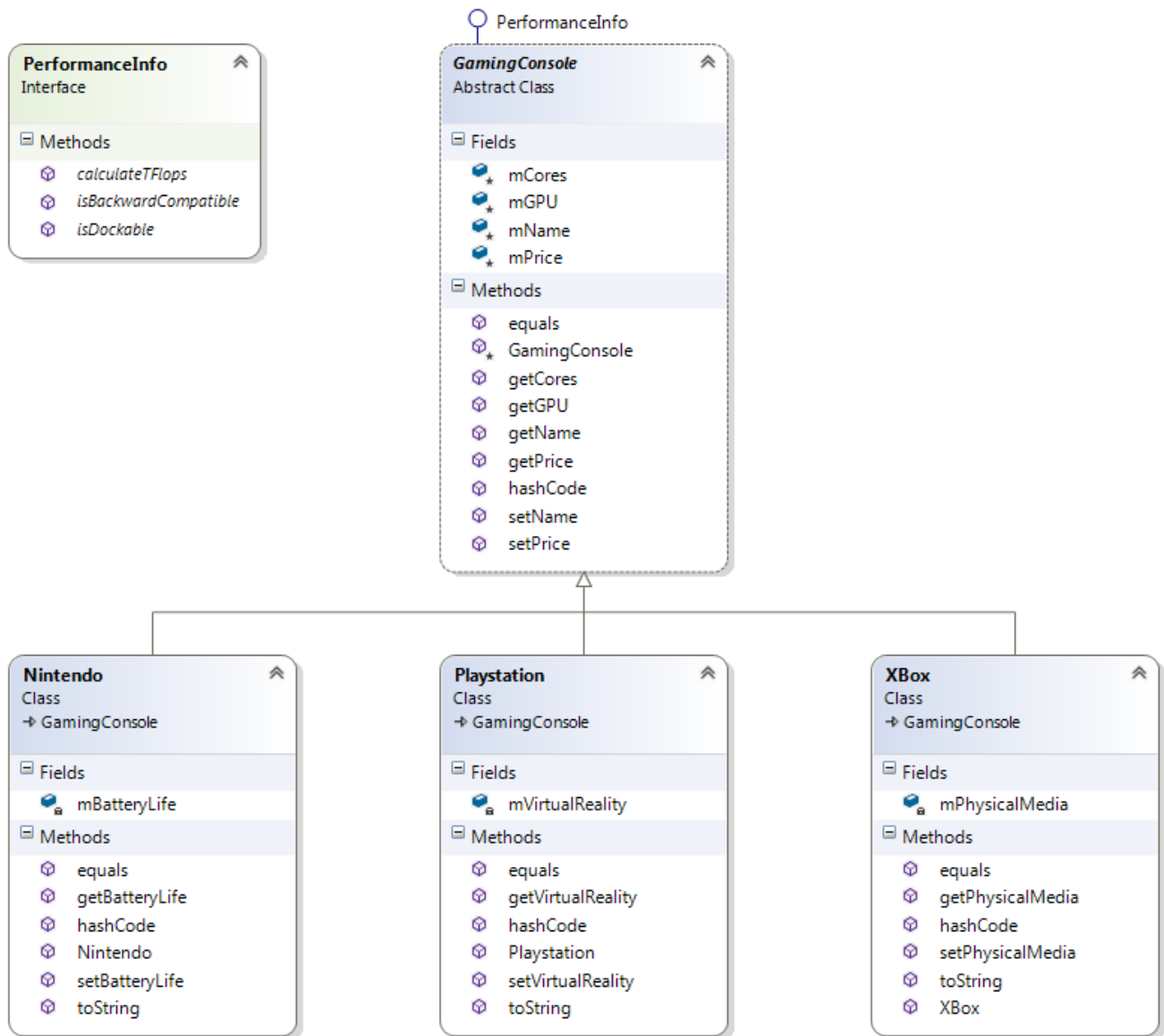
XBox [One, GPU=853MHz, Cores=768, Price=\$499.00, Physical Media=Blu Ray, TFLOPS=1.3, Backward Compatible=Yes, Dockable=No]

4. See class diagram below for other specifications.

GamingDemo: in this demo, implement the public static void main(String[] args) method to perform the following:

1. Create a List of GamingConsole to store all objects.
2. Add all 6 objects (pre-created in the demo) to the List
3. In a loop, prompt the user with 4 options to enter Nintendo (option 1), Playstation (option 2), or Xbox (option 3) in the inventory. Option 4 is to exit.
4. If the user enters option 1, prompt for all Nintendo info, create a new Nintendo object and add it to the List. *****Use the following info when you run the demo: Name=Wii U, GPU=550, Cores=320, Price=299, BatteryLife=0*****
5. If the user enters option 2, prompt for all Playstation info, create a new Playstation object and add it to the List. *****Use the following info when you run the demo: Name=3, GPU=500, Cores=400, Price=499, VirtualReality=N/A*****
6. If the user enters option 3, prompt for all Xbox data, then create a new Xbox object and add it to the List. *****Use the following info when you run the demo: Name=360, GPU=500, Cores=240, Price=399, PhysicalMedia=N/A*****
7. If the user enters option 4 (exit), your program should:
 - Print a header "~~~~~All Gaming Consoles~~~~~" the loop through the List and print GamingConsole objects to the console.
 - Print a header "~~~~~Gaming Consoles Less than \$300~~~~~" then display the filtered list showing only consoles priced less than \$300.
 - Print a header "~~~~~Backward Compatible Consoles~~~~~" then display the filtered list showing only backward compatible consoles.
 - Print a header "~~~~~Consoles with at least 1 TFLOP ~~~~~" then display the filtered list showing GamingConsoles with >= 1.0 Tera FLOP
8. Create a method: **public static List<GamingConsole> filter(List<GamingConsole> allConsolesList, Predicate<GamingConsole> predicate)** returning a List of GamingConsoles matching the predicate.
9. Create a method: **public static Predicate< GamingConsole> consolesLessThan(double price)** that will use a lambda expression to return a Predicate of GamingConsole less than a specified price.
10. Create a method: **public static Predicate< GamingConsole> backwardCompatibleConsoles()** that will use a lambda expression to return a Predicate of **GamingConsole** that are backward compatible.
11. Create a method: **public static Predicate< GamingConsole> minimumTFLOPS(double value)** that will use a lambda expression to return a Predicate of GamingConsole that have a minimum TFLOPS of the specified value.

Below is a UML class diagram showing the design:



When finished

1. Copy all output from the Eclipse console showing one new Nintendo, one new Playstation and one new Xbox object added by the user. Be sure to use the specific values indicated previously. Paste the Eclipse output into your Word document.
2. Copy all code from each Java file into your Word document. Label each file.
3. Finally, please export your project as a zip file (LastNameFirstName_CS272_Midterm_Part01_Complete.zip) and upload it to Canvas
4. JavaDocs are NOT required for this project.

Part 02 – Java FX and Event Handling

[20 points total]

Please create a new Eclipse project named: **CS272_Midterm_Part02**

Note: You will be creating this Eclipse project from scratch (no code provided).

1. Create the following JavaFX user interface, making use of a GridPane layout within the Scene. Ensure that all nodes (e.g. Labels, TextFields and Buttons) are present and aligned with appropriate padding, margins and horizontal alignment.
2. Ensure that Hgap and Vgap for the entire GridPane is set to 5.
3. Ensure that future value text field is not editable.
4. Format the future value output as currency (with \$ symbol and commas between 3 numbers)
5. The future value can be calculated using the following formula:

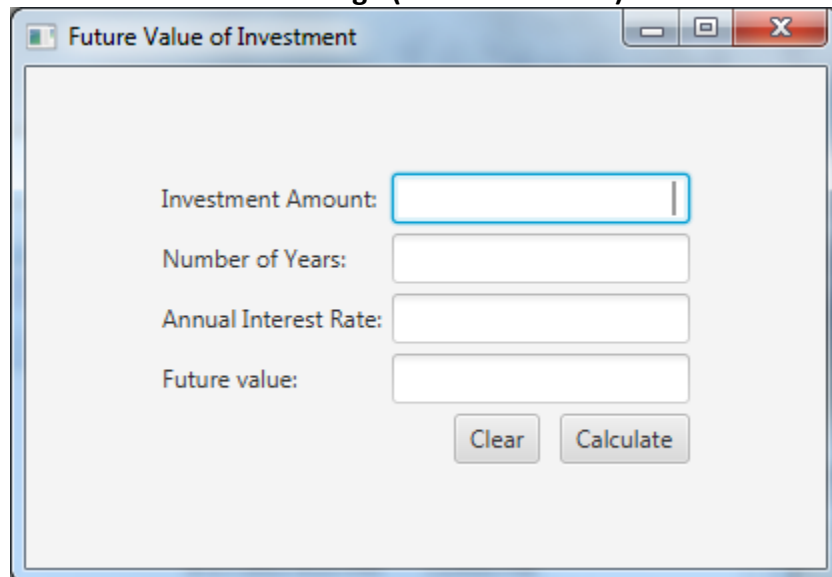
$$\text{Future Value} = A * (1 + i)^n$$

A = investment amount

i = interest rate per period (per month)

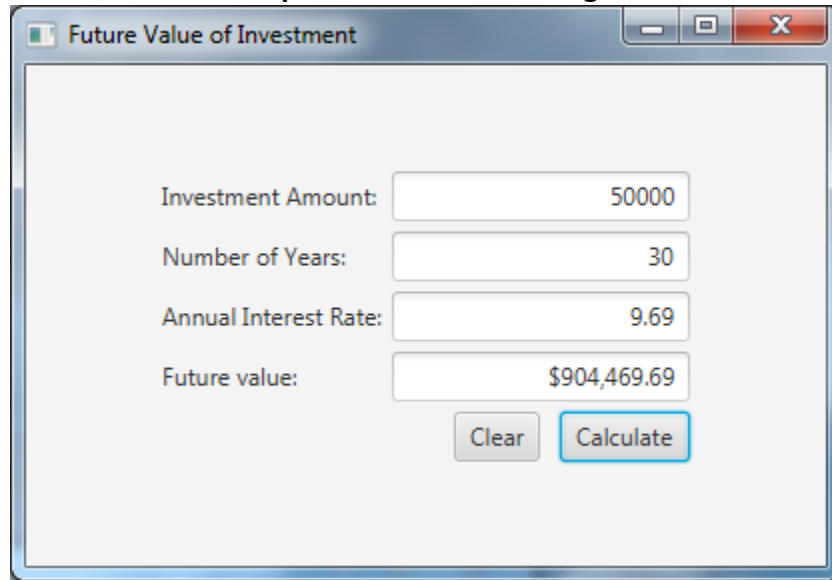
n = total number of periods (total months)

Initial Stage (no data entered)



The screenshot shows a JavaFX window titled "Future Value of Investment". Inside the window, there is a light gray background. On the left side, there are four labels: "Investment Amount:", "Number of Years:", "Annual Interest Rate:", and "Future value:". To the right of each label is a text input field. The "Investment Amount" field is highlighted with a blue border. Below the input fields, there are two buttons: "Clear" and "Calculate". The window has a standard title bar with minimize, maximize, and close buttons.

Data Entered – Output shown after clicking Calculate button



Future Value of Investment

Investment Amount: 50000

Number of Years: 30

Annual Interest Rate: 9.69

Future value: \$904,469.69

Clear Calculate

6. The Clear button should clear all text fields and return focus back to the Investment Amount text field.
7. Implement the EventHandlers for both the Calculate and Clear buttons. You can use lambda expressions or inner classes to handle the events.

When finished

5. Take a screen shot of your completed JavaFX application after clicking the Calculate button with the input above. Paste the screen shot into your Word document.
6. Copy all code from Eclipse into your Word document.
7. Finally, please export your project as a zip file (LastNameFirstName_CS272_Midterm_Part02_Complete.zip) and upload it to Canvas.
8. JavaDocs are NOT required for this project.

Congratulations! Have a great spring break (you've earned it).