|  |  |
| --- | --- |
| COURSE | BDSE |
| QUALIFICATION NAME | BDSE |
| MODULE NAME | SOFTWARE ENGINEERING |
| ASSIGNMENT | ASSIGNMENT 3 |
| ISSUE DATE | 13-JANUARY-2023 |
| SUBMISSION DATE | 18 JANUARY 2023 |

Table of Contents

TASK 1.13-16

TASK 1.216-28

TASK 229-35

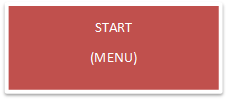
REFERENCE…………………………………………………………………………………………………………………………………………36

Task 1.

Task 1.1

Develop a fully object-oriented solution by providing source code of implementation for the algorithm written in Formative assignment 2 :

-Algorithm :





DISPLAY

NON VALID

AGAIN

NO

VALID

YES



(VALID) YES



YES YES

VALID

NON VALID

NO (DISPLAY AGAIN)

YOU CAN JOIN THE COMPETITION

(ADDITIONAL PRIVATE COACHING)

YES NO

IF ELSE

YOU CANNOT

JOIN THE COMPETITION(FOR BEGINNER)

VALID NON

YES VALID

(DISPLAY AGAIN)

INPUT

HOUR

YOU WANT

ACCUMULATE & CALCULATE TOTAL COSTING

CALCULATE/MONTH

END

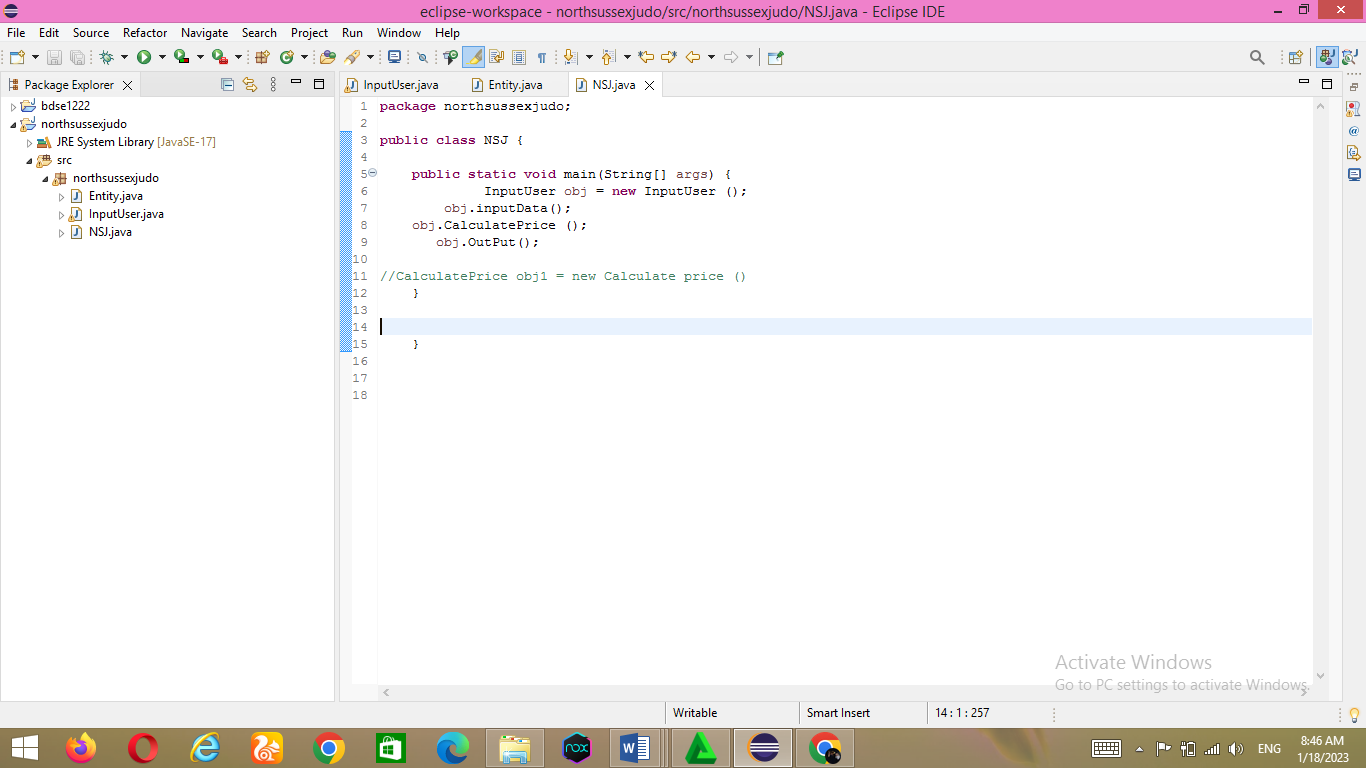
AFTER COMPARE WEIGHT + WEIGHT COMPETITION CATEGORY

CALCULATE/MONTH

* Tool IDE used for Implementation : Eclipse IDE



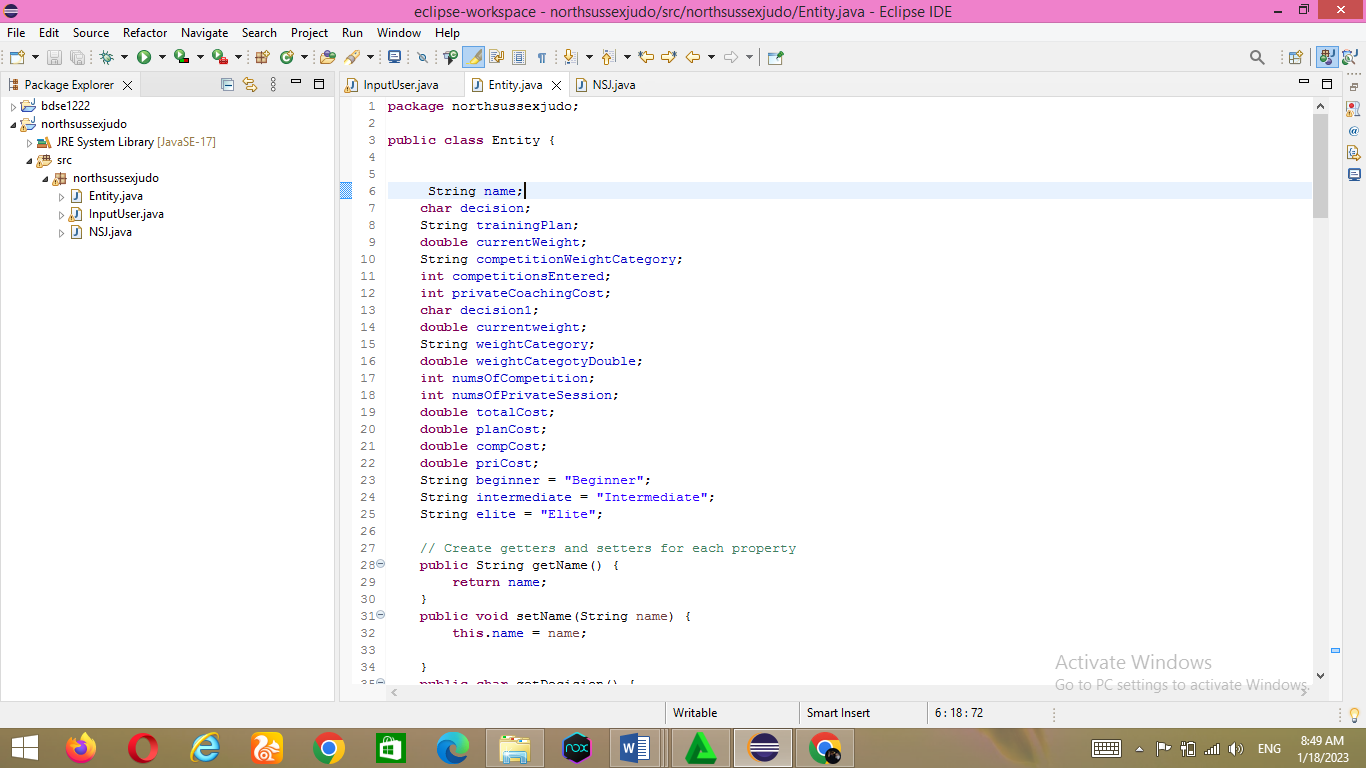
. Java-primarily based totally improvement platform acknowledged for its plugins that permit builders to broaden and check code written in different programming languages. Eclipse is launched below the phrases of the Eclipse Public License

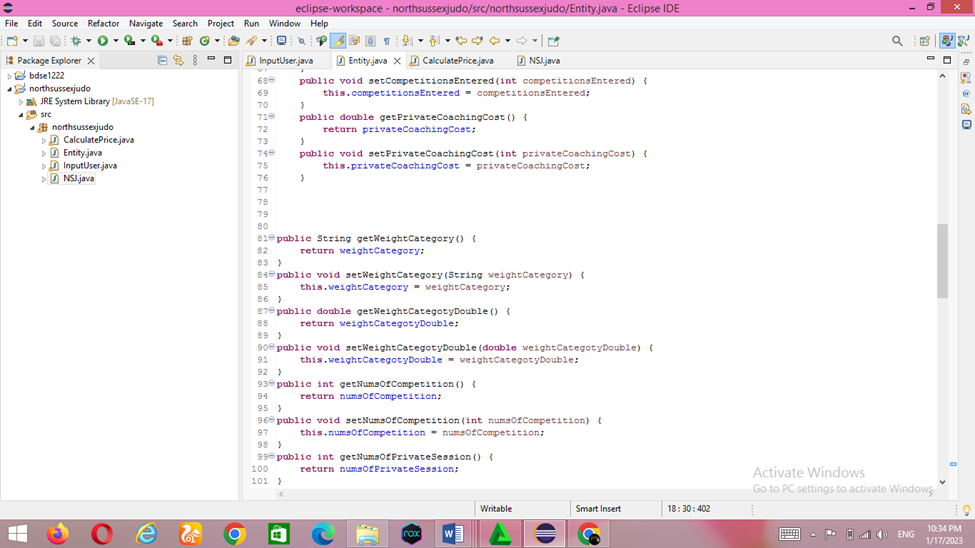
* Java Classes
* NSJ.Java

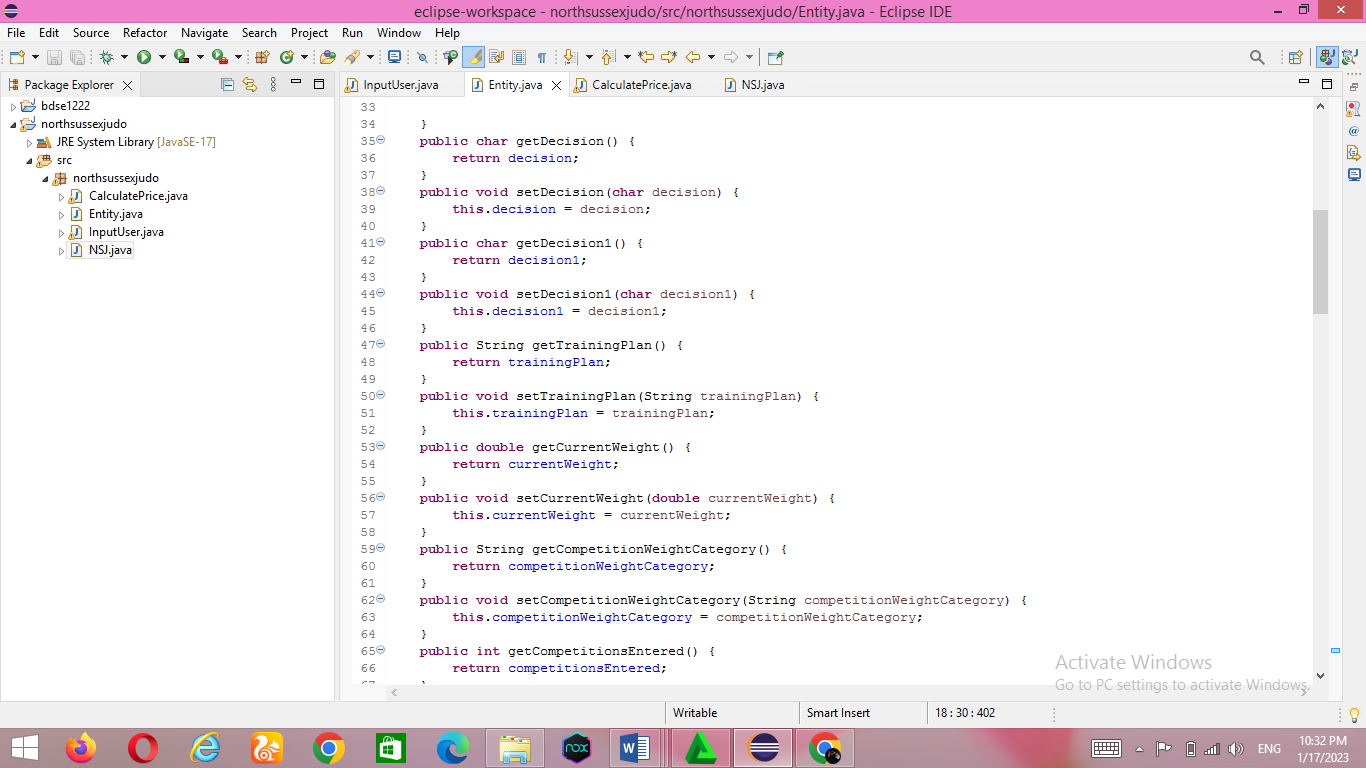
This NSJ Class is in which to run the class that become created earlier. Entity class and InputUser Class will run on this class

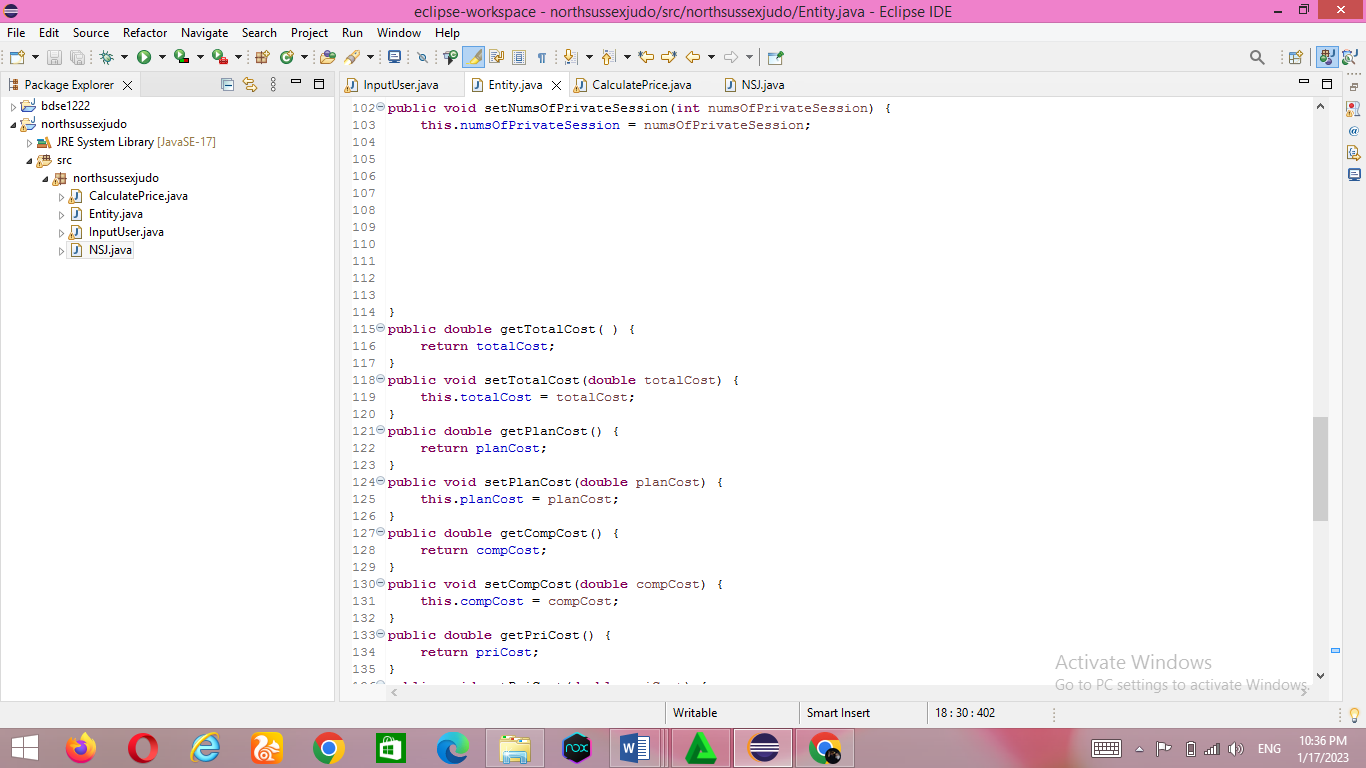
* Entity class

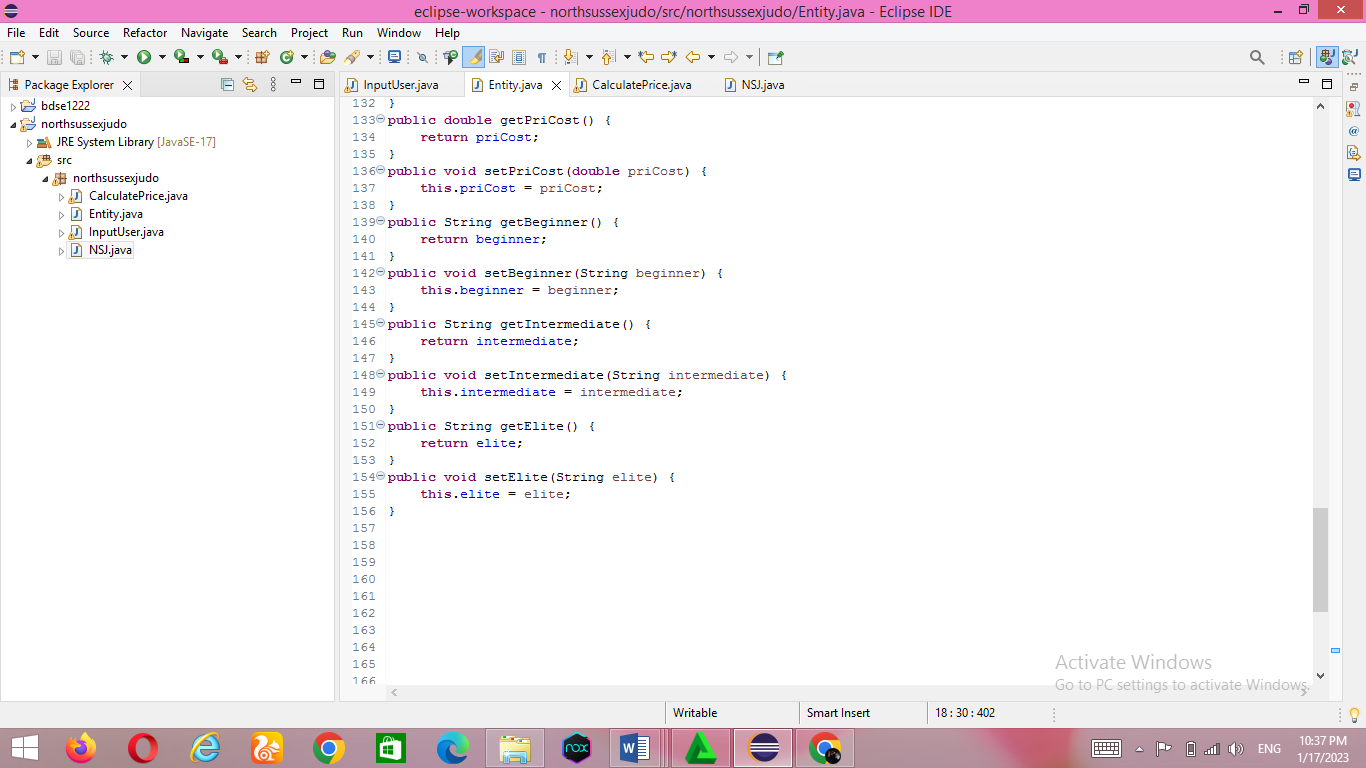
All parameters are defined in this class. All parameters are private to use the oops encapsulation function. Getters and setters are created for each parameter so that child classes can use those parametersity Class







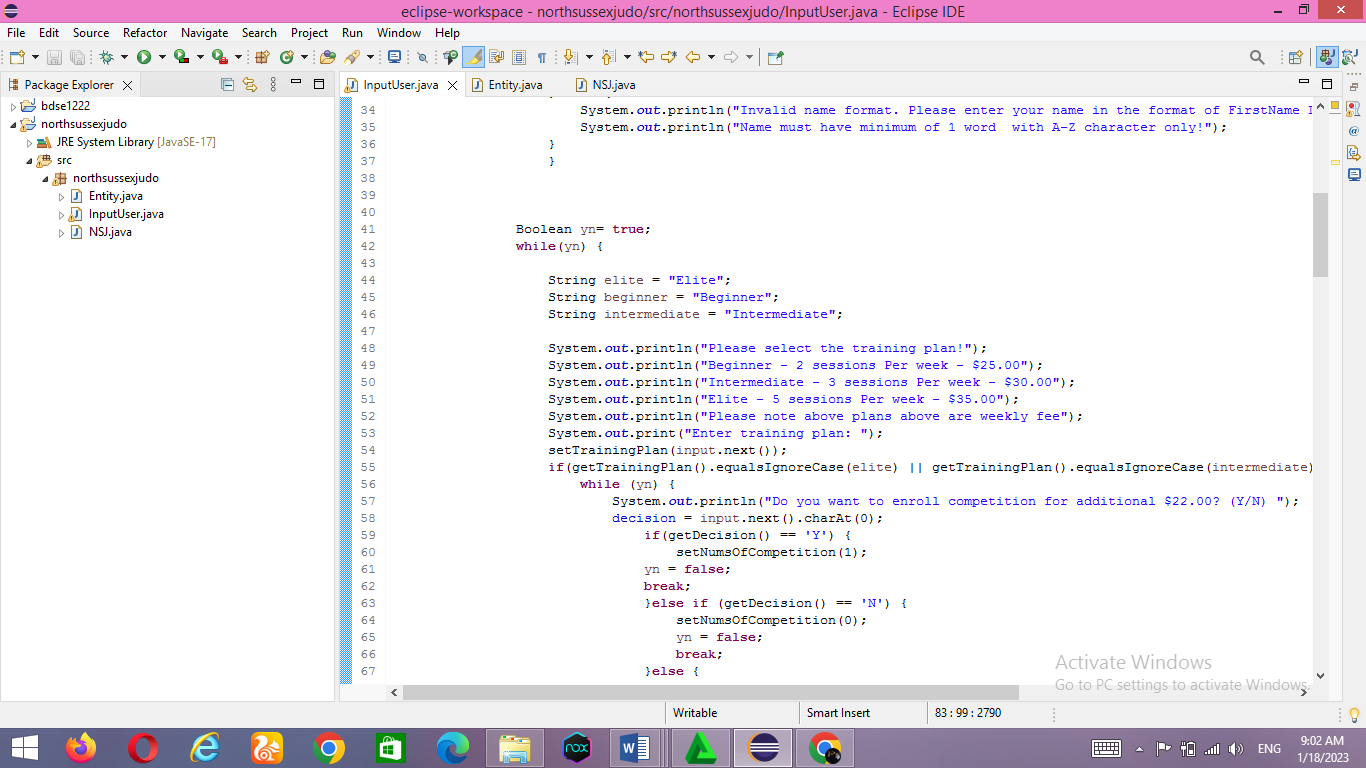


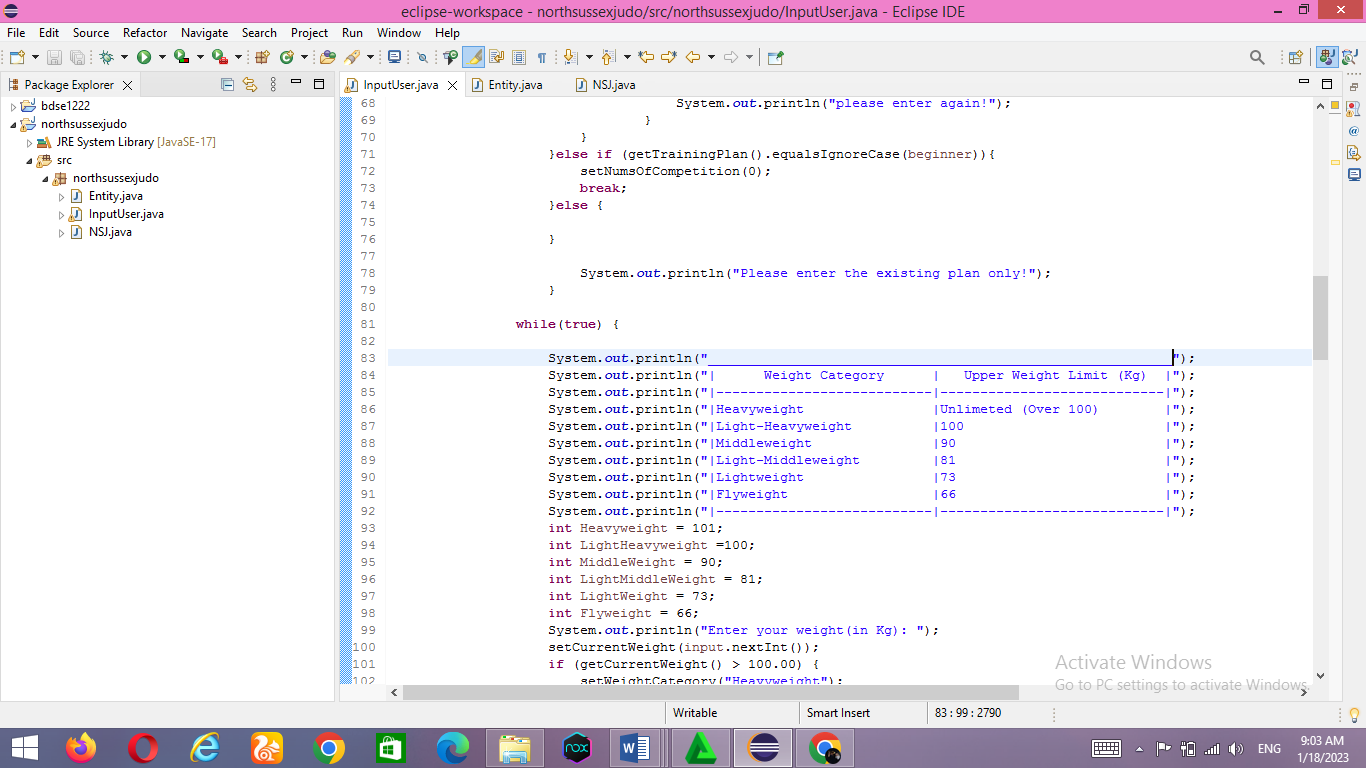


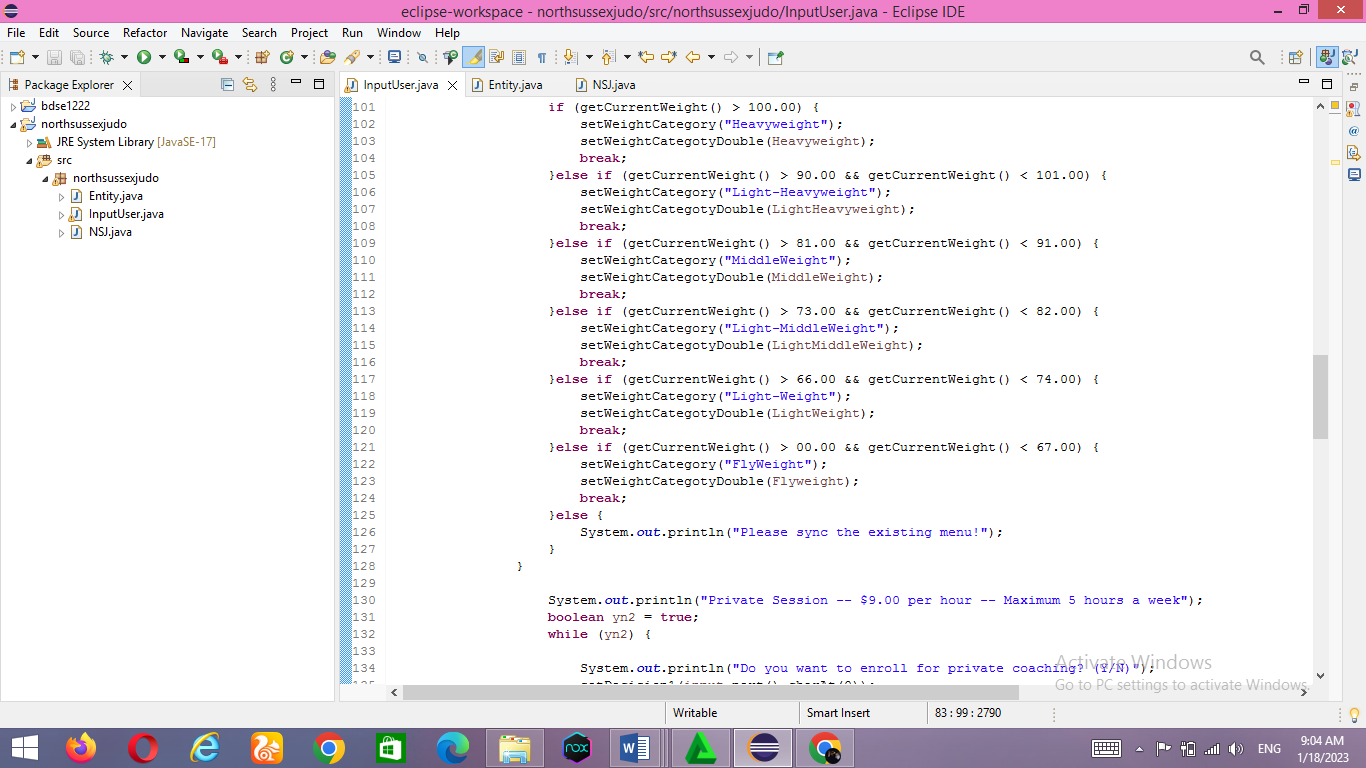
* InputUser class

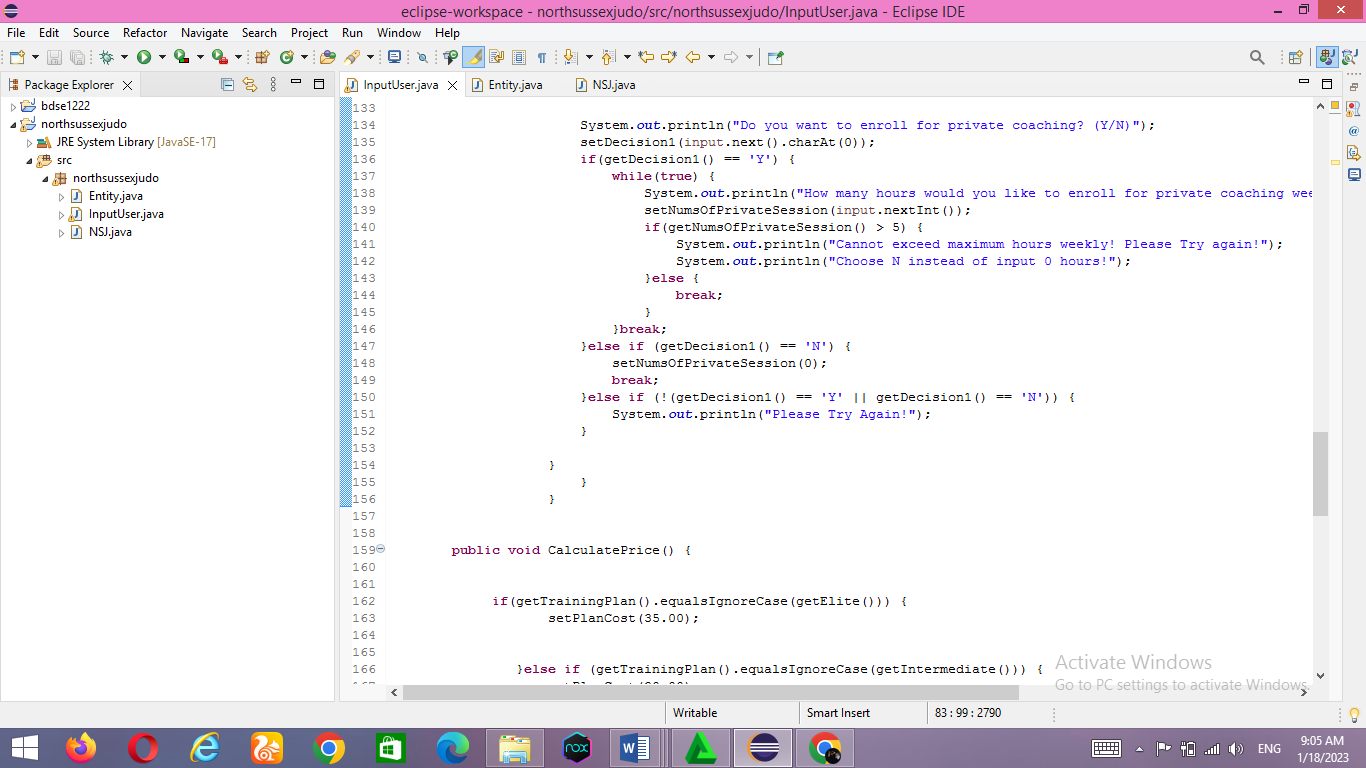
The InputUser class extends the Entity class. This class uses the concept of inheritance so that it can access the members of the object class. This class takes user input and also validates whether the input is valid or not. If the input is incorrect, the application prompts the user for valid input until he or she enters the correct input

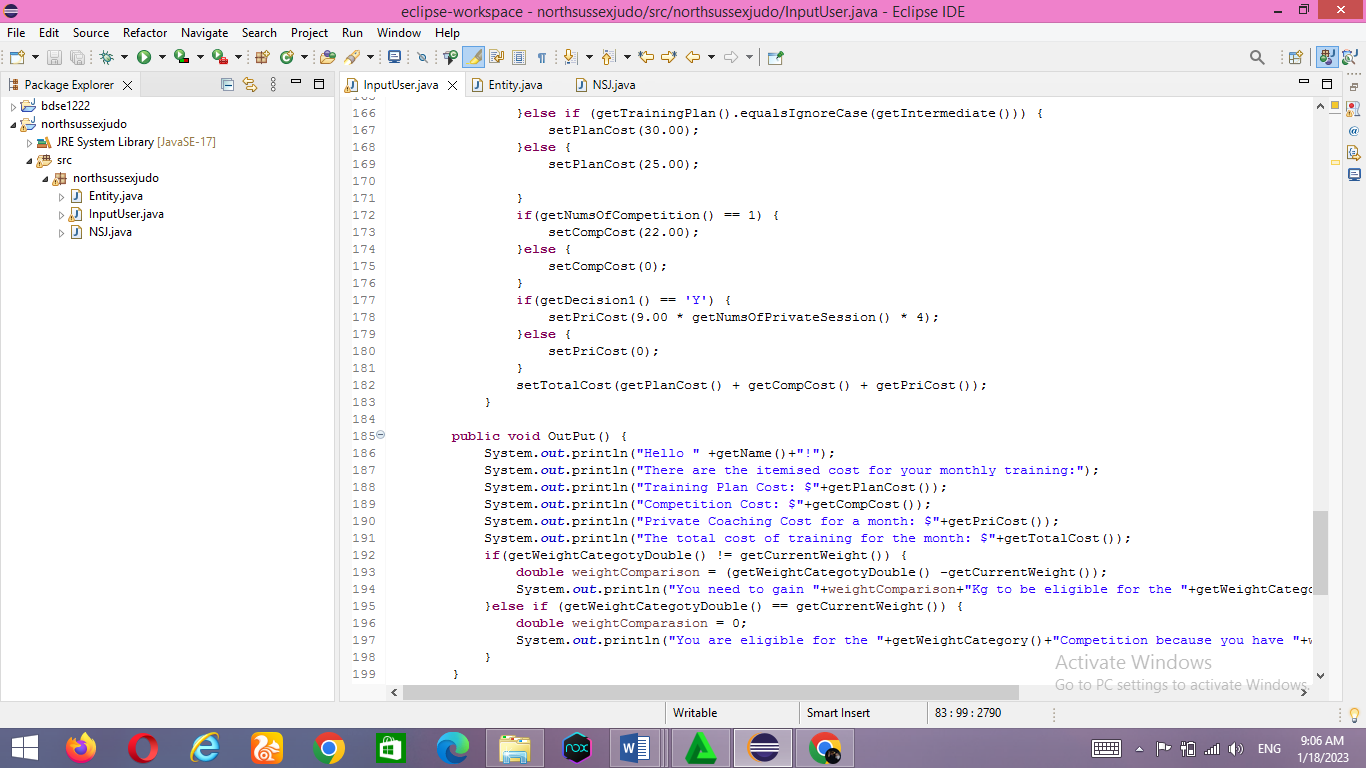




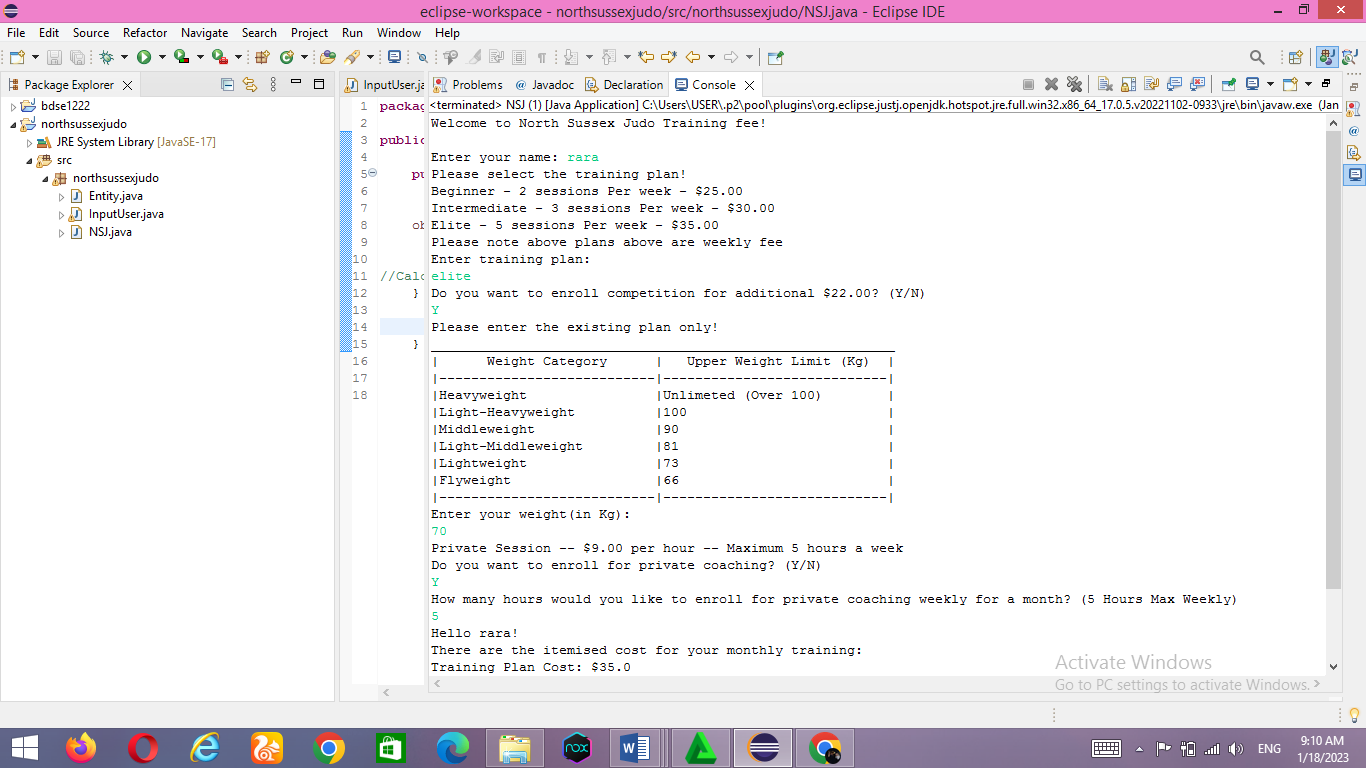


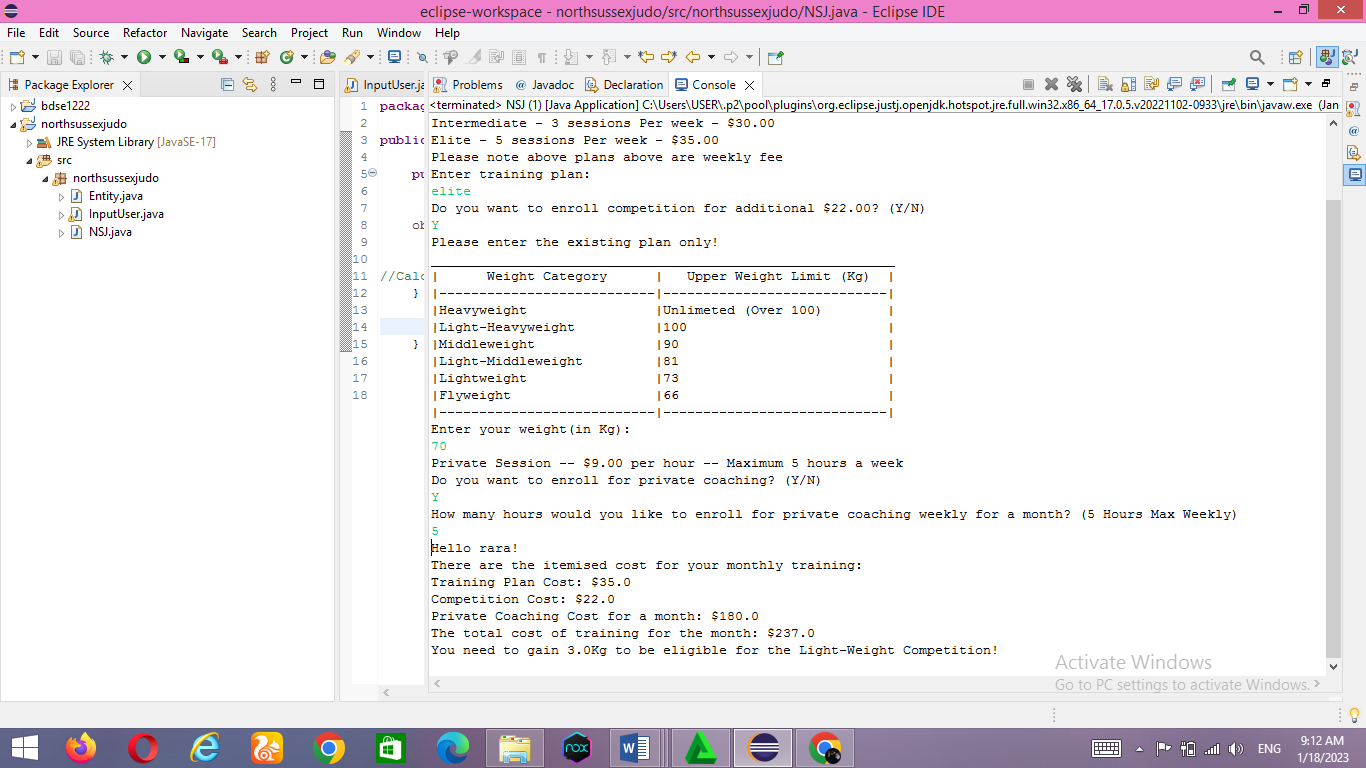




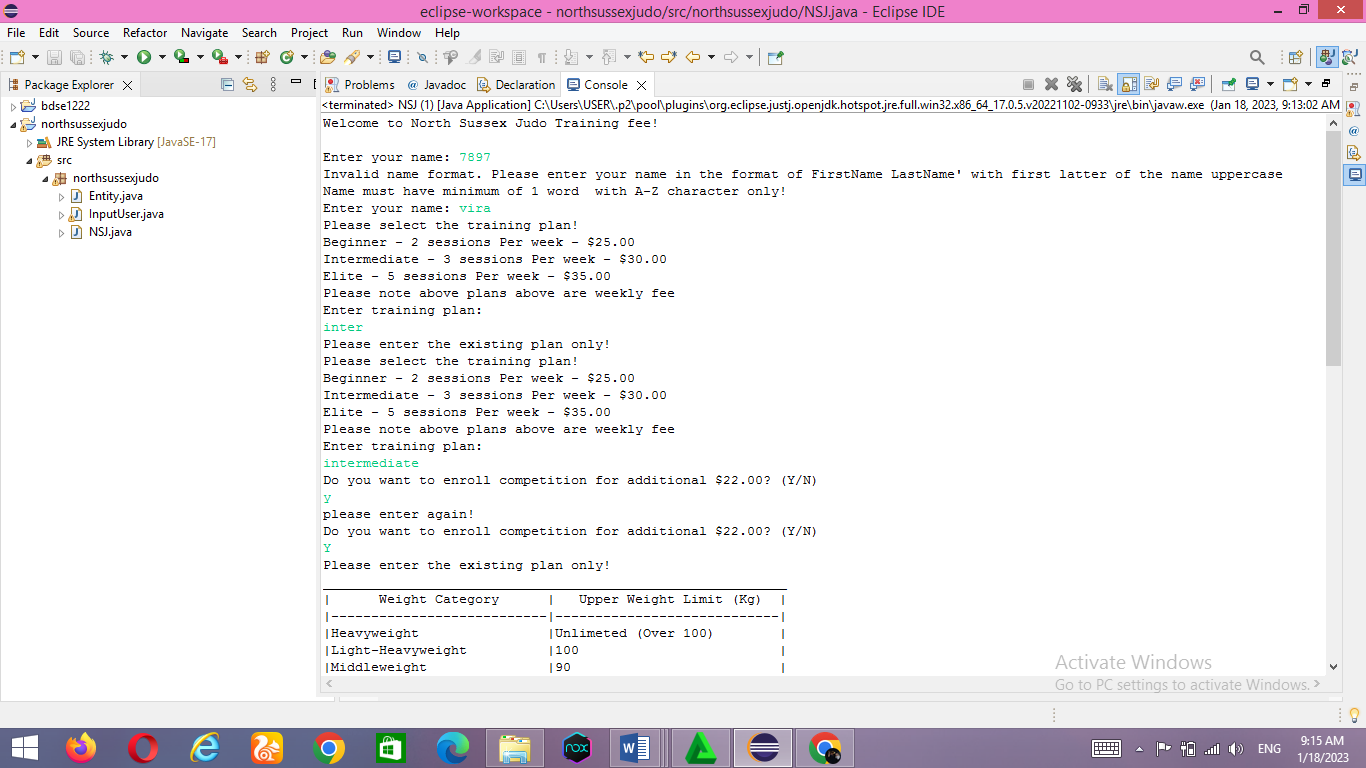


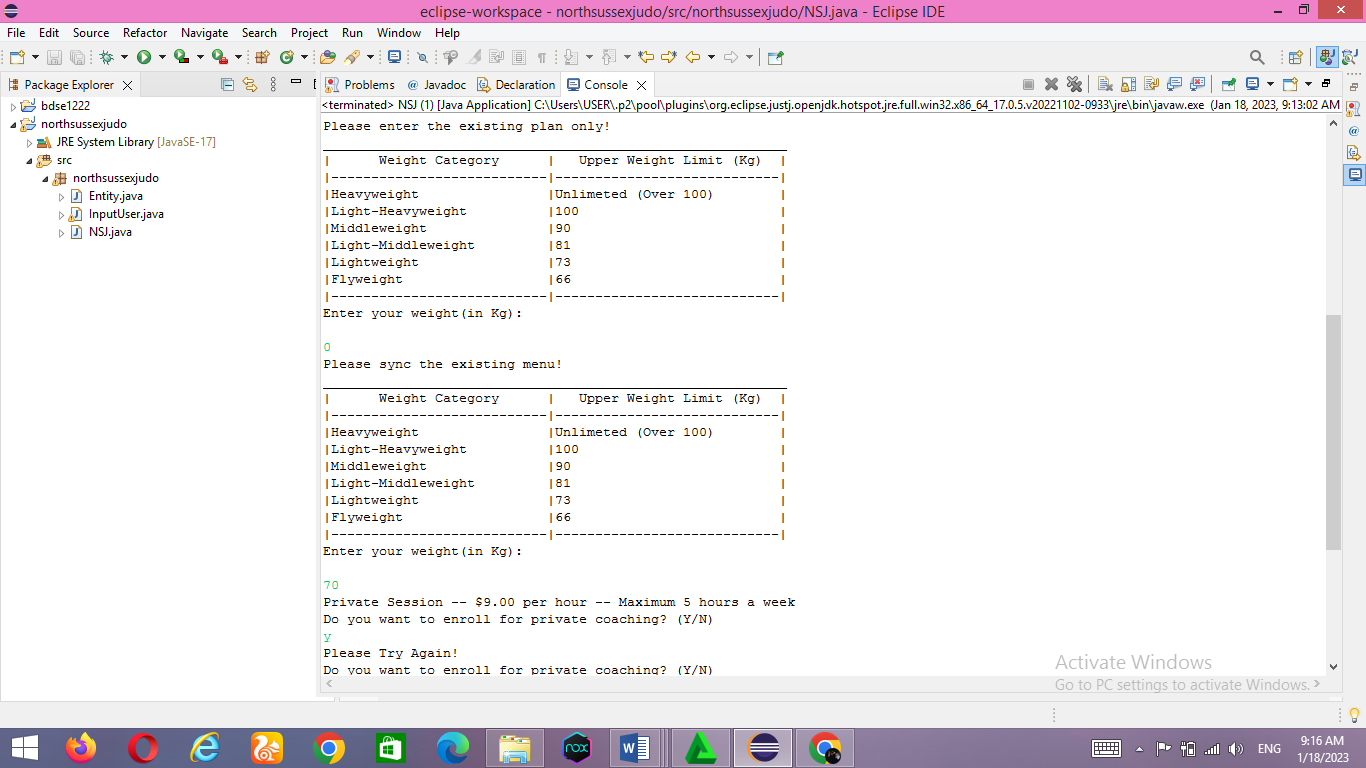
* Project output screen captures

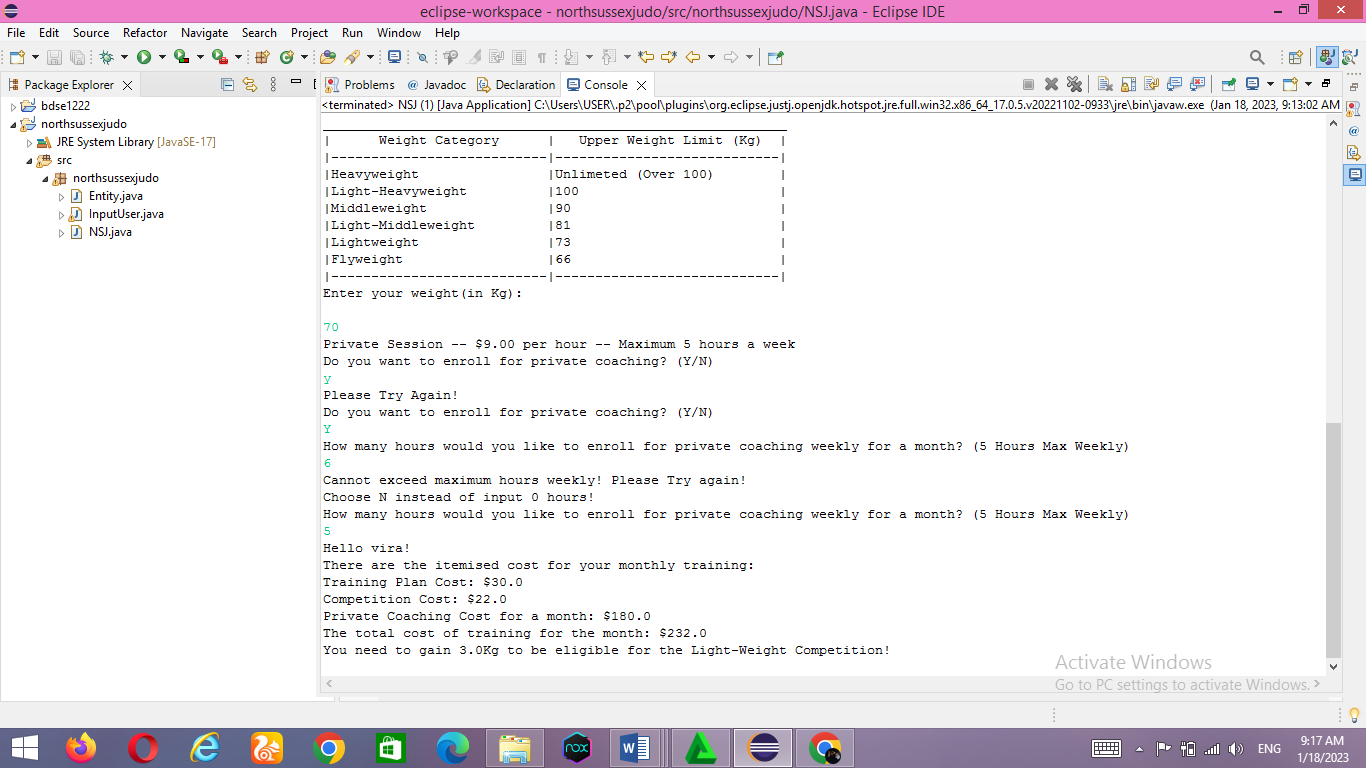
-Athlete 1 (rara) with including correct output.



-Athlete 2, with including incorrect output and execution handling.







* 1. Provide a brief note on how the algorithm is translated to equivalent code.

Programming Languange : POP & OOPS

Programming Language : POP & OOPS

POP : A procedural language follows a sequence of statements or commands to achieve a desired result. Each set of steps is called a procedure, and a program written in one of these languages ​​contains one or more procedures.

OOPS : This type of language treats a program as a collection of objects consisting of data and program elements known as attributes and methods. Objects can be reused within an application or in other applications. This makes it a popular type of language for complex applications, as the code is easier to reuse and scale. For implementation we use Java.

programming paradigms are a way of classifying programming languages ​​based on their characteristics. Languages ​​can be divided into several paradigms. Some paradigms mainly deal with the influence of the language implementation model, such as B. the possibility of side effects, or whether the sequence of actions is determined by the implementation model.

Other paradigms mainly deal with the way code is organized, for example: B. Grouping code into units together with the state modified by the code. Others deal mainly with syntax and grammar. We will use the imperative language as a procedural paradigm and an object-oriented paradigm.

* Procedural programming is a programming paradigm based on the concept of procedure calls from imperative programming. Procedures (a type of routine or subroutine) simply contain a series of computer steps that must be performed. Any procedure can be called at any time during program execution, including by another procedure or by itself. Procedural programming languages ​​are also imperative languages ​​because they are directly related to the state of the execution environment. This can be anything from variables (which can correspond to processor registers) to the location of a "turtle" in the Logo programming language.
* COMPARISON WITH OTHER PARADIGMS

Often, the terms "procedural programming" and "imperative programming" are used synonymously. However, procedural programming relies heavily on blocks and scope, whereas imperative programming as a whole may or may not have such features. As such, procedural languages generally use reserved words that act on blocks, such as if, while, and for, to implement control flow, whereas non-structured imperative languages use goto statements and branch tables for the same purpose.

* Object-oriented programming The focus of procedural programming is to break down a programming task into a collection of variables, data structures, and subroutines, whereas in object-oriented programming it is to break down a programming task into objects that expose behavior (methods) and data (members or attributes) using interfaces. The most important distinction is that while procedural programming uses procedures to operate on data structures, object-oriented programming bundles the two together, so an "object", which is an instance of a class, operates on its "own" data structure.

Nomenclature varies between the two, although they have similar semantics:

|  |  |
| --- | --- |
| **Procedural** | **Object-oriented** |
| Procedure | [Method](https://en.wikipedia.org/wiki/Method_(computer_science)) |
| [Record](https://en.wikipedia.org/wiki/Record_(computer_science)) | Object |
| Module | Class |
| Procedure call | Message |

Languange Key Components :

⎫ java.lang package is automatically imported in every Java class. we can import all classes and sub-packages from a package by using the wildcard \* e.g. import northsussexjudo\* will import all classes on package northsussexjudo as well as classes on any subpackage. package must be the first statement, even before import statement in Java source file

⎫ Programmers should use hierarchical structure to define packages and sub-packages, as used in JDK itself e.g. java.util, then java.util.concurrent, then java.util.concurrent.atomic or java.util.concurrent.lock etc. Just like keeping the first letter as Capital case of a Java class, packages usually use all lowercase letters, you can see it through Java libraries e.g. java.lang, java.util, java.io they all use small letters. One example of creating a package for a product in a company to define some features can be com.prod.features. If the product or company name contains a hyphen, replace it with an underscore. packages and classes should not be cyclically dependent on each other because cyclical dependencies make it difficult to understand and modify the software system.

⎫ Support for explicit imports by defining each imported class from an implicit import with a wildcard, using the \* operator to include all classes.

from a package. An explicit import is better than an implicit import because it cannot be misinterpreted. Java's import preferences (properties, closures, wildcards) are not well known and are often misused, leading to problems that are difficult to find and understand. Also, using explicit imports results in a clear listing of all dependencies. This makes it easier to understand how dependent a class is on other classes (the higher the dependency, the more complex the class is).

* Overview of the Eclipse Java Development Tools (JDT)

The JDT project provides a set of plugins that add full Java IDE functionality to the Eclipse platform. JDT plugins provide APIs so they can even be extended by other tool manufacturers. JDT plugins are classified as follows: JDT APT

JDT APT adds support for handling annotations in Java 5 projects in Eclipse. It provides the following features:

• Support for running an annotation processor written for a compatible Sun command-line tool

• Contribution of annotation build artifacts during incremental builds

• Contribution of numeric symbols for annotation-based issues

JDT Core

JDT Core defines non-UI infrastructure.

It contains:

• an incremental Java constructor

• a Java model that provides an API for navigating a tree of Java elements. The Java element tree defines the Java-centric view of the project. It shows components such as package fragments, compilation units, binary classes, types, methods, and fields.

• Support code help and code selection.

• Indexed search infrastructure is used for search, code support, type hierarchy computation, and refactoring. • Evaluation support The JDT Core infrastructure has no built-in JDK version dependencies.

JDT Debug

JDT Debug implements Java debugging support and works with any JDPA-compliant target Java VM. It is implemented on top of the language independent "debug model" provided by the platform debugger.

JDT debug provides the following debugging features:

• Launching of a Java VM in either run or debug mode

• Attaching to a running Java VM • Expression evaluation in the context of a stack frame

• Scrapbook pages for interactive Java code snippet evaluation

• Dynamic class reloading where supported by Java virtual machine.

JDT Text JDT Text provides the Java editor with the following features:

• Keyword and syntax coloring

• Context specific (Java, Javadoc) code assist and code select • Method level edit • Margin annotations for problems, break points, or search matches • Outliner updating as editing takes place

• API help shows Javadoc specification for selected Java element in a pop-up window • Import assistance automatically creates and organizes import declarations • Code formatting

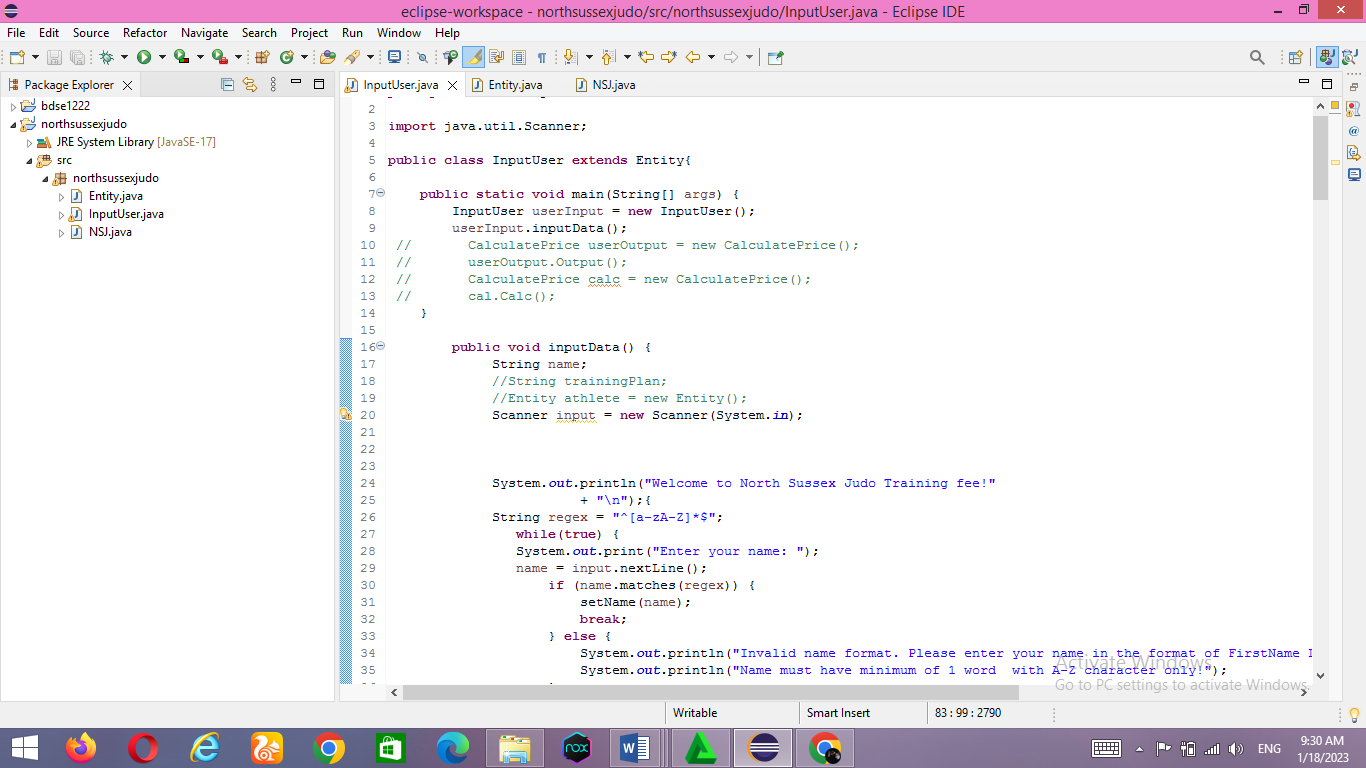
JDT User Interface J

DT UI implements Java-specific workbench contributions: 1087 • Package Explorer 1087 • 1087 Type Hierarchy View • Java 1087 Overview • Generating Wizards Java Section 1087 User Interface JDT provides refactoring support such as: fetch or safe method renaming for Java elements that also update references. Users can preview (and veto) individual changes caused by refactoring. Search Support JDT implements detailed searches such as finding declarations and/or references to packages, types, methods, and fields, within the scope of the current workspace, workspace, or selection. . Comparative Support JDT performs a structured comparison of Java compilation units showing changes to individual Java methods. It supports replacing individual Java entries with their version in the local history.

Applied principles OOPS: inheritance / encapsulation Inheritance

Meaning: a concept that inherits properties from one class to another class example, the relationship between father and son. Inheritance in Java is the process of inheriting all the behavior of a parent object. The concept of inheritance in Java is that new classes can be built on top of older classes. We can use the methods and properties of the parent class if they inherit from the existing class. We can also add additional fields and methods to your existing class. A parent-child relationship, also called an IS-A relationship, is represented by inheritance. The idea of ​​inheritance means that new classes can be built on top of existing classes. If you derive from an existing class, you can use its methods and properties. You can also add new fields and methods to your current class.

Inheritance syntax in Java: extends the inheritance meaning that the InputUser class inherits from the Entity class. is used to inherit the properties of a subclass/parent class to its child subclass



if you need a category with a purpose to inherit from every other magnificence, then you may use "extends" kind extends after the class call after which upload the call of the inheriting magnificence. Then anything is withinside the heir magnificence may be at once accessed through the inheriting magnificence.

Extends key-word in Java The prolonged key-word extends a category and is a hallmark that a category is being inherited through every other magnificence. When you assert InputUser Class extends a Entity class, it method that UserInput class is inheriting the properties(methods, attributes) from Entity class.

Here, Entity Class is the superclass or figure magnificence and InputUser Class is the subclass or infant magnificence.

* Is-A courting IS-A Relationship is finished associated with inheritance.

For example – a kiwi is a fruit; a bulb is a tool. This courting may be done through:

• Using extend Keyword

• To keep away from code redundancy

• IS-A courting is unidirectional, this means that we will say that a bulb is a tool, however vice versa; a tool is a bulb isn't viable seeing that all of the gadgets aren't bulbs

• The IS-A courting is a good couple, which means that converting one entity will have an effect on different en How to obtain IS-A courting

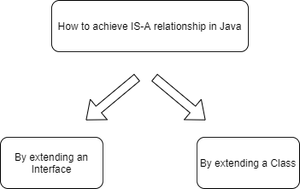
• IS-A courting can surely be done through extending an interface or magnificence through the use of increase key-word entities.

Advantage of IS-A courting :

Code Reusability.

Reduce redundancy.

How to obtain IS-A courting : IS-A courting can surely be done through extending an interface or magnificence through the use of increase key-word.



ENTITY CLASS (parent class)

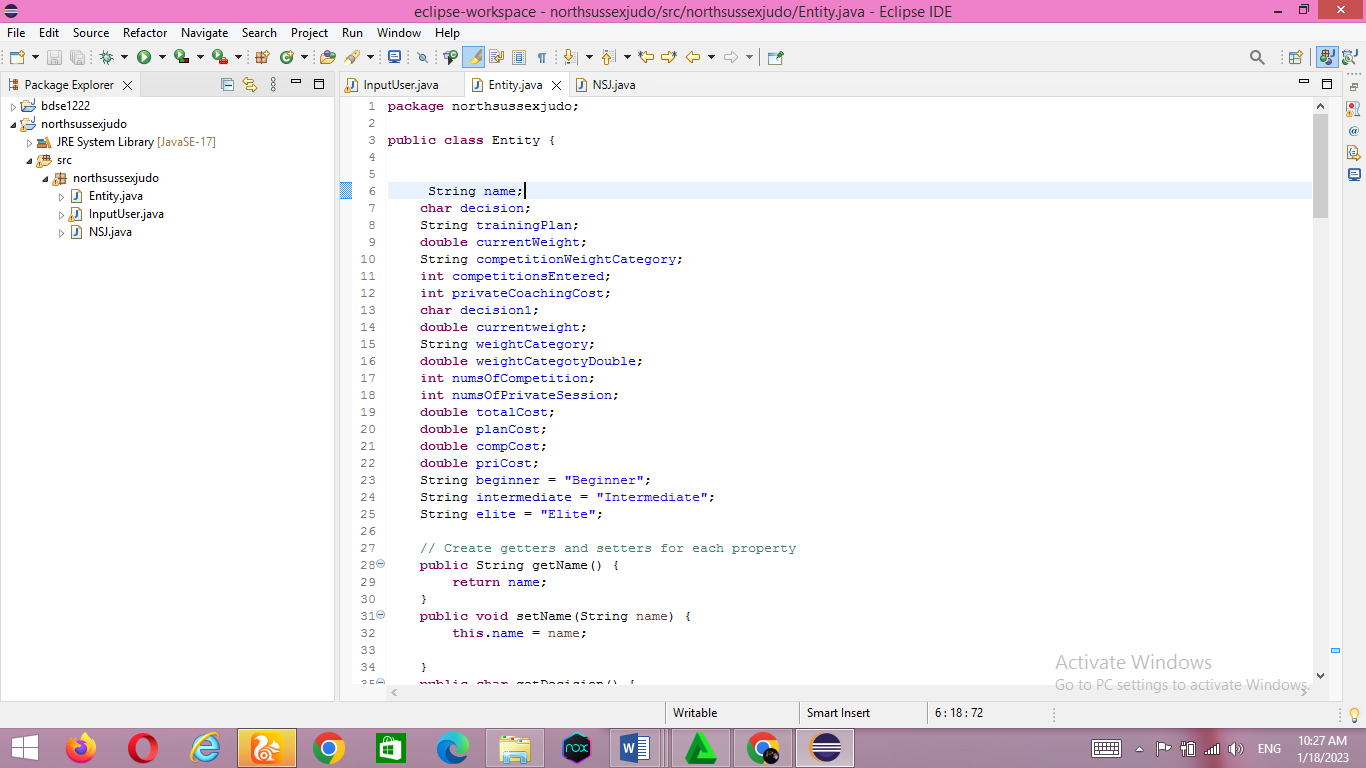
**NSJ (grand Father class)**

InputUser (child class)

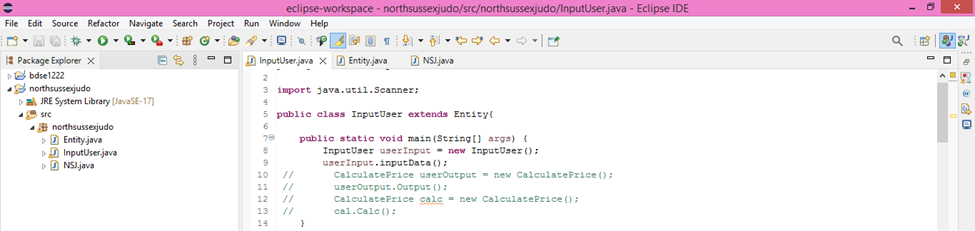
In the diagram above, InputUser extends the Object class, which means that Entity is the parent class of InputUser, and InputUser is intended to have an Is-A relationship.

* WORKING EXAMPLE IN THE PROJECT:

This is an object class that inherits its content from the InputUser class



And here is the InputUser class which inherits from the entity class by typing extends



* Java keywords**:** package, static, import, public, class, void, extends, new, while, if, else, break, boolean, int, private, char, double, return, this.

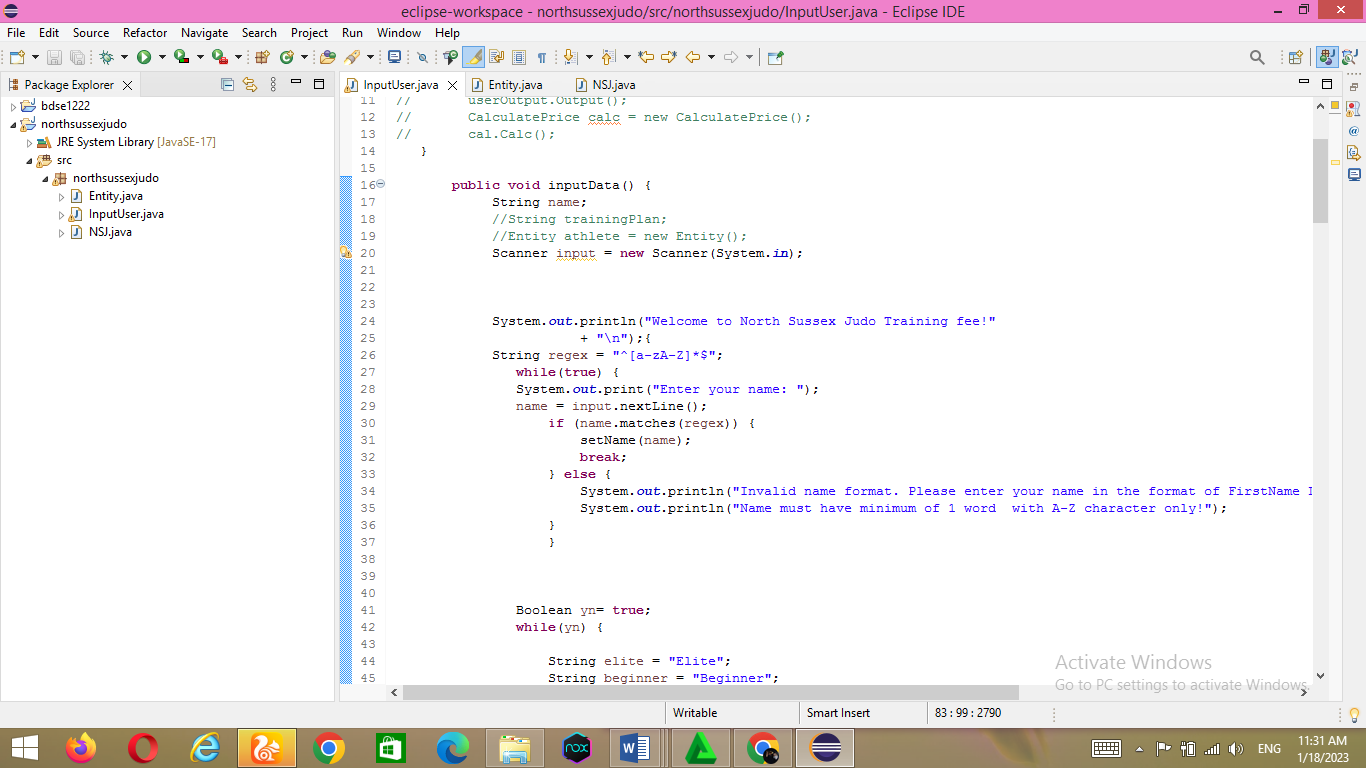
|  |  |
| --- | --- |
| **Keyword** | **Fungsion** |
| package | To define the class Package name |
| static | One of the properties for variables and methods, so that we can call them directly without having to create an object from the class |
| import | Importing packages |
| public | Access rights that allow attributes/data to be accessed by any class, in the same or different packages |
| class | Used to define and implement objects |
| void | Void is a method that does not have a return value, usually it is not used to look up values ​​in an operation |
| extends | Used to inherit the properties of a subclass/parent class on its child subclass/class |
| new | New is used to create an instance of a class or array |
| while | The while keyword is used to loop, as long as the condition is true until the expression evaluates to false |
| if | To make a conditional statement |
| else | A keyword that is used in combination with an if to make a statement if the condition in the if is not met |
| break | Break is used to execute the program to the next command. So, this break keyword skips the statement the programmer doesn't want, and continues on to the next statement |
| boolean | Boolean is used to declare a variable that can store values ​​of type boolean, i.e. true or false |
| int | The data type is an integer, the data type Int has a size of 32 bits and a Range of -2147483648 to.d. 2147483647 |
| private | Access Rights so that the properties of a class cannot be inherited and can only be accessed by the class itself |
| char | Declare a variable that can store a 16 bit character value |
| double | To store fractional values, has a size of 8 bytes and 64 bits and a length of Range +/- 1.8 x 10308 with a total precision/digit of 15 |
| return | To return the value of a variable |
| this | Used to represent an instance of the class in which it appears, or used if a variable name is the same as a parameter |

* JAVA CONTROL STRUCTURES

Decision-Making statements:

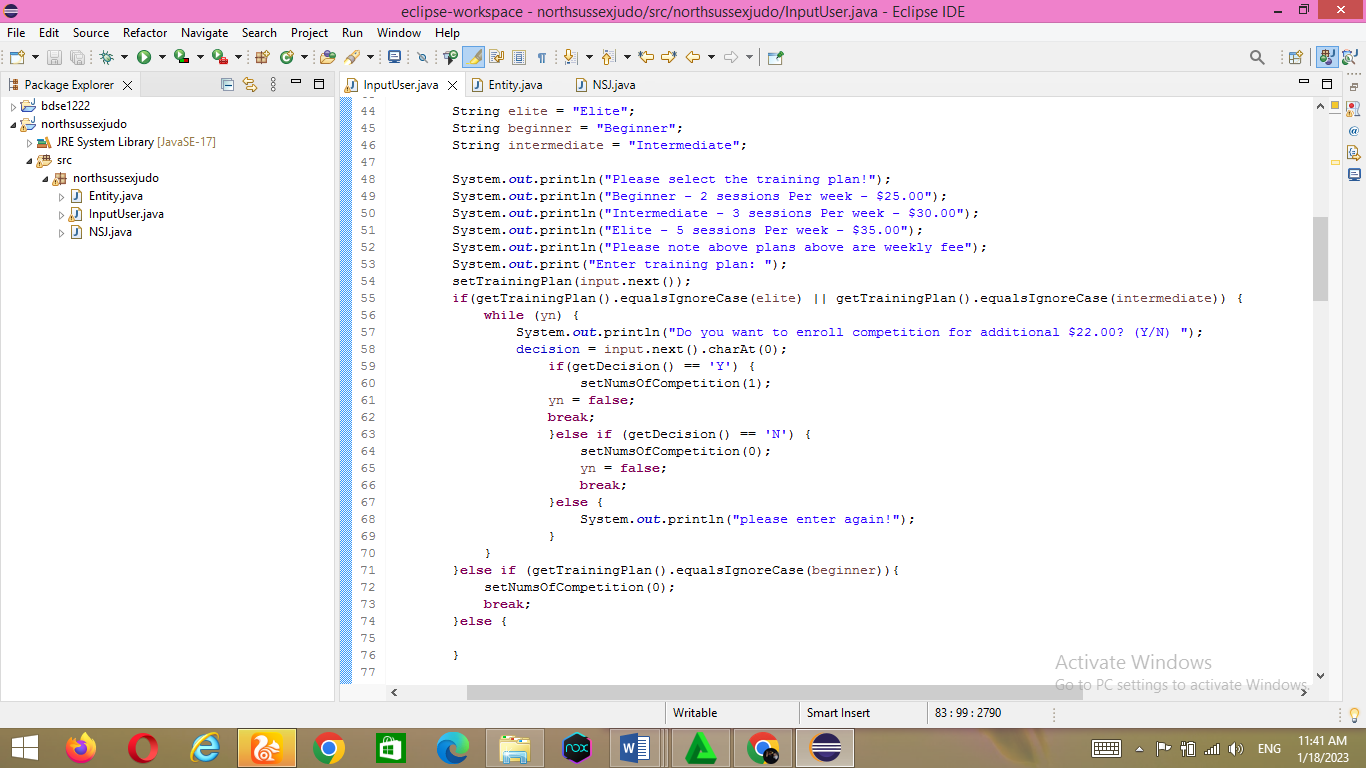
As the name suggests, decision statements specify which statements are executed and when. Decision statements evaluate the logical expression and flow of a control program depending on the outcome of a given condition. There are two types of decision statements in Java i.e. if statement and switch statement. In this project I used an if statement.

¬ If Statement: In Java, the "if" statement is used to evaluate a condition. The control of the program is directed depending on the specific situation. The condition of an if statement evaluates to a logical value of true or false.



2) Nested if-statement

In nested if-statements, the if statement can contain a **if** or **if-else** statement inside another if or else-if statement.



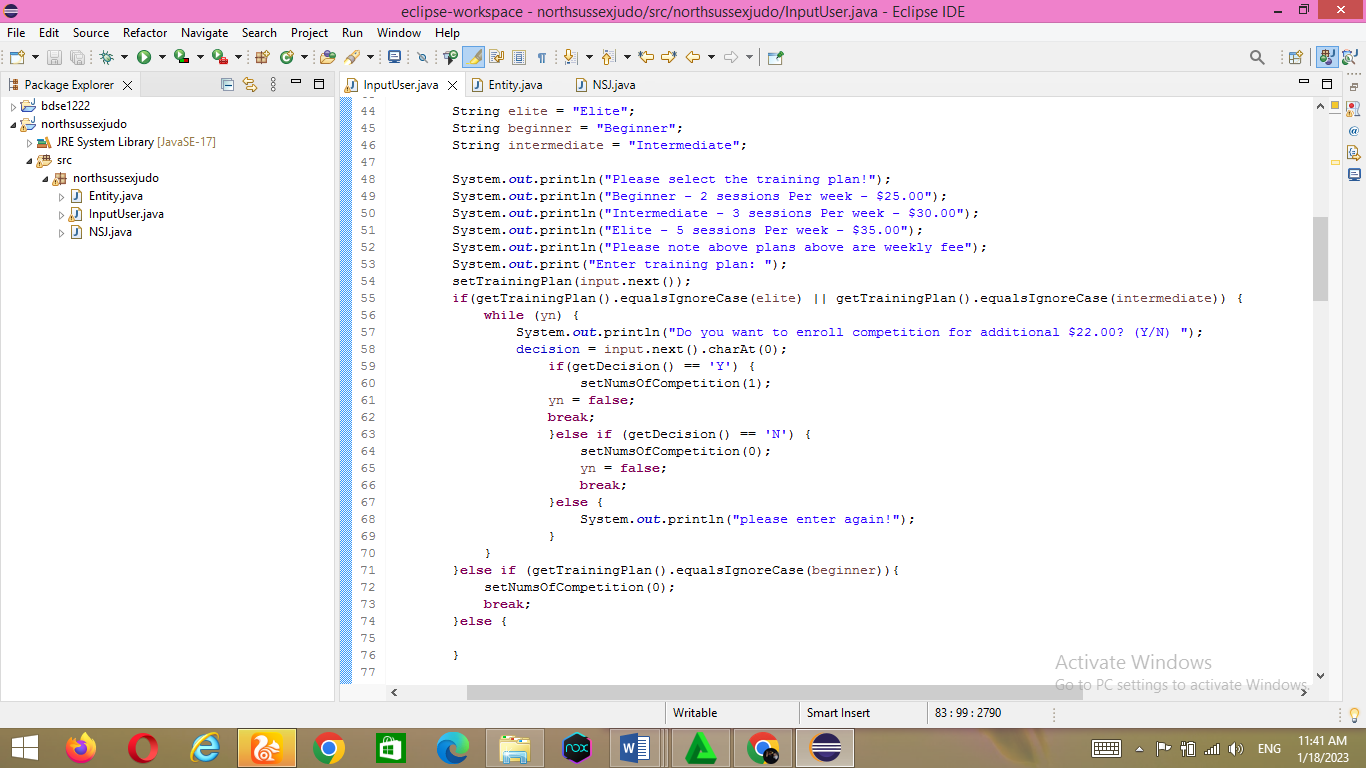
* (JUMP STATEMENT)

Java break statement

As the name suggests, the break statement is used to interrupt the current program flow and transfer control to the next statement outside a loop or switch statement. However, with nested loops, only the inner loop is broken.

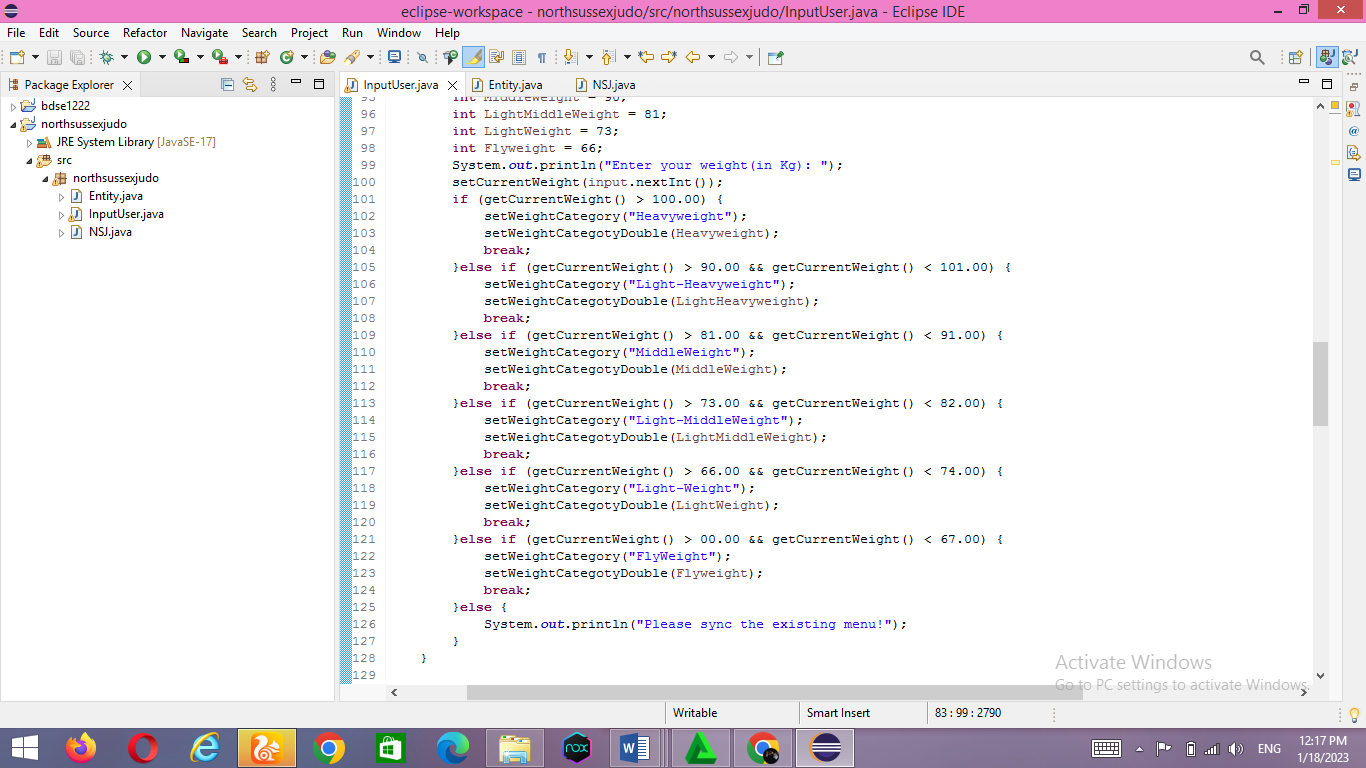
Java program cannot use break statement by itself. H. Can only appear inside a loop or switch statement

.



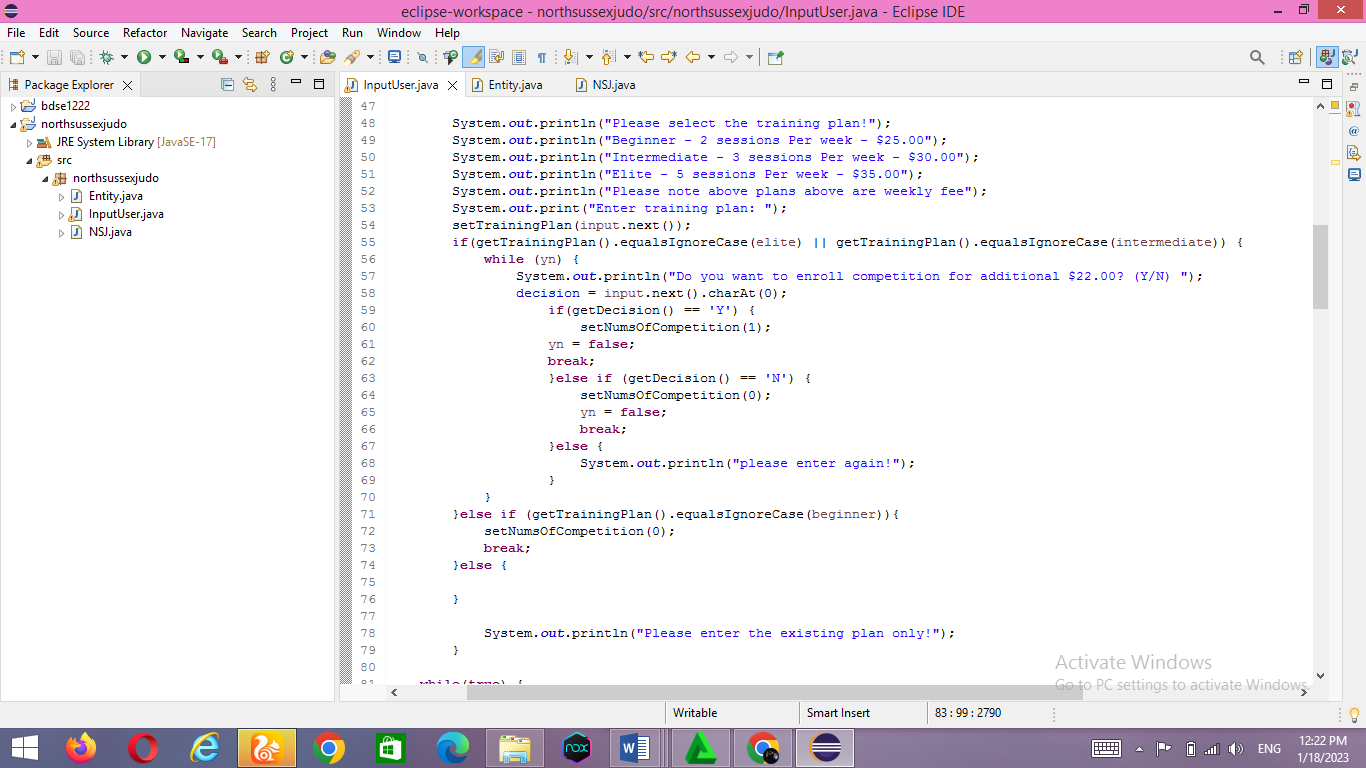
* && (Logical AND)

This operator is used to compare AND conditions. Returns true if both conditions are true. Operators are executed from left to right. Also note that && is a short-circuit operator. The short-circuiting operator here means that if the condition on the left is false, the operation on the right is never executed because the result is definitely false.



* equalsIgnoreCase()

The equalsIgnoreCase() method is used to compare a string to a specific object, regardless of case.



* Coding editor and JDK version installed.

Eclipse IDE for Java Developers - 2022-09 and jdk-11.0.17

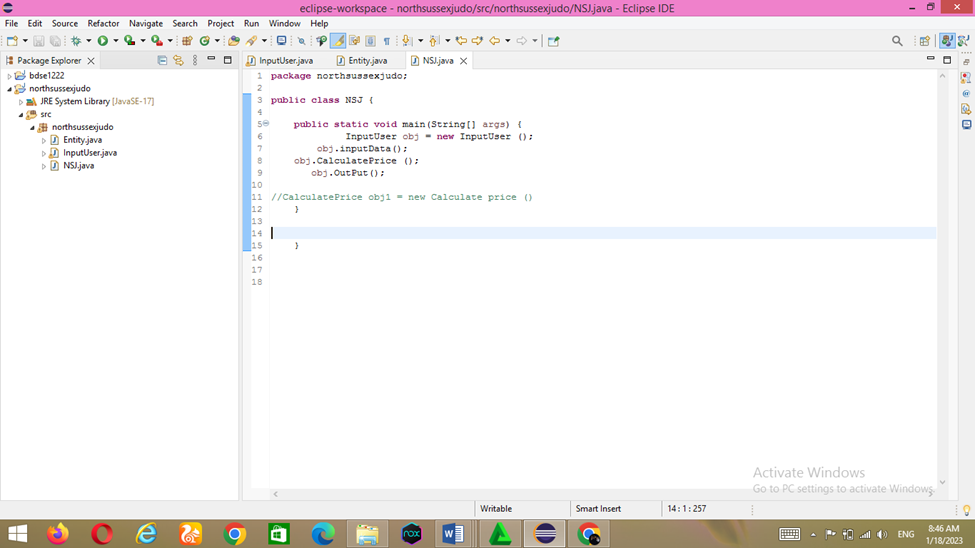
* Prepare your project structure.

Different classes structured in their respective packages

Just click new-->package and Eclipse will ask you for the name of the package. Enter the name of the package and you're done.

If you want to create a Java class for this package, select the package in the Package Explorer and create a new -->Java class.

Using packages in Java is very easy. To declare a package in Java, simply use the package keyword and package name at the top of your Java source file. The package declaration must be the first line in the Java source file, before any import statements.



This project has a northsussexjudo directory with packaged NSJ classes. When the Java compiler compiles this class, it will check if this particular class is stored as /north/sussex/judo/NSJ.java relative to the classpath.

All Java classes are structured in separate packages based on their functionality. For example, the java.lang package contains essential classes for the Java programming language. B. Threads, exceptions, errors, objects, etc.

Packages like java.util, on the other hand, contain all the helper classes. Collection classes, scanners and other utilities. The java.io package contains Java classes related to input/output functions. Packages provide encapsulation for Java programs. The default access modifier for variables or classes is package-private. They are visible only in the package in which they are declared.

Use packages to encapsulate all functionality. This allows you to change functionality, including new functionality, or change the implementation without breaking the entire application.

Packages are not the highest level of Java encapsulation that the private keyword can achieve, but they are still the best option and should encapsulate all functionality, not just classes.

Packages are also the primary tool for organizing your code. You have to put a lot of effort into determining package naming, which classes belong to which packages, etc. Also, for better architecture, try following the packaging best practices from Uncle Bob (Robert C. Martin):

1) CCP (Common Closure Principle) - advises merging classes that are likely to change together. Obviously it should be part of the same functionality.

2) CRP (Common Reuse Principle) - advice to set classes based on reusability aspects. Huge packages containing many classes that cannot be reused together are discouraged. The idea is to keep the package small and cohesive.

Most importantly, name your packages appropriately based on their functionality.

.

TASK 2

2.1 An explanation of the debugging features available in your chosen IDE.

* Debugging:

Debugging is the process of finding and fixing bugs and errors in software source code. When software doesn't work as expected, computer programmers examine the code to determine the reason for the failure. Debugging can also mean a system that shows the flow of a program, sometimes referred to as a flowchart/process of a running program.

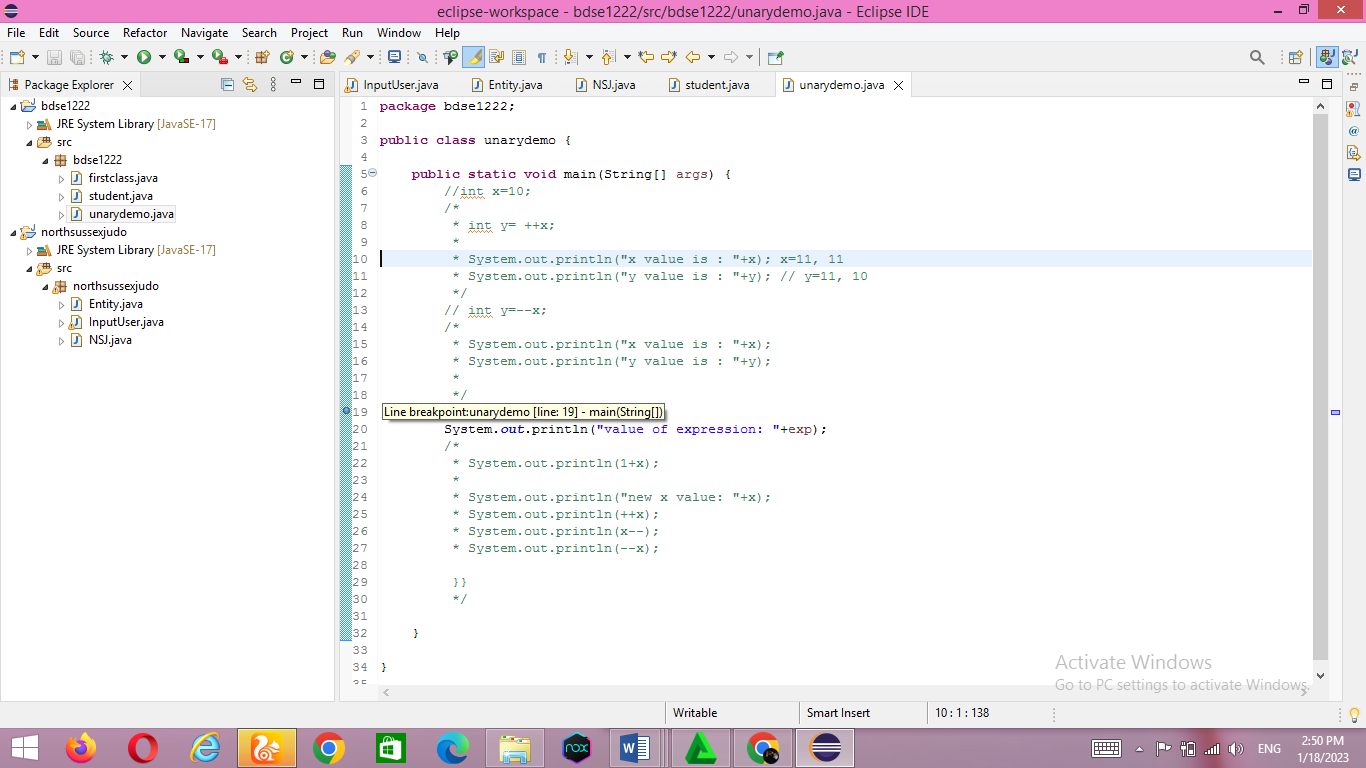
Debugging is the process of finding and eliminating program or system errors. A bug is an error or defect that prevents a program or system from working properly. Programs are usually written with very complicated and complicated code, so a small mistake can have a big impact on the whole program. Therefore, pre- and post-release debugging of your program is very important.

The word debug comes from the word bug, which means insect. The term comes from a rather unique incident. On September 9, 1947, an American computer scientist named Grace Brewster Murray Hopper discovered an insect trapped in his computer's relay. This will prevent your computer from working. After this incident, the word debugging came into general use. In 1945 the word debug was used in aviation. This can be interpreted as testing aircraft engines. Then, in 1963, the term debugging came into common use among programmers to refer to the process of finding and correcting program or system errors. The

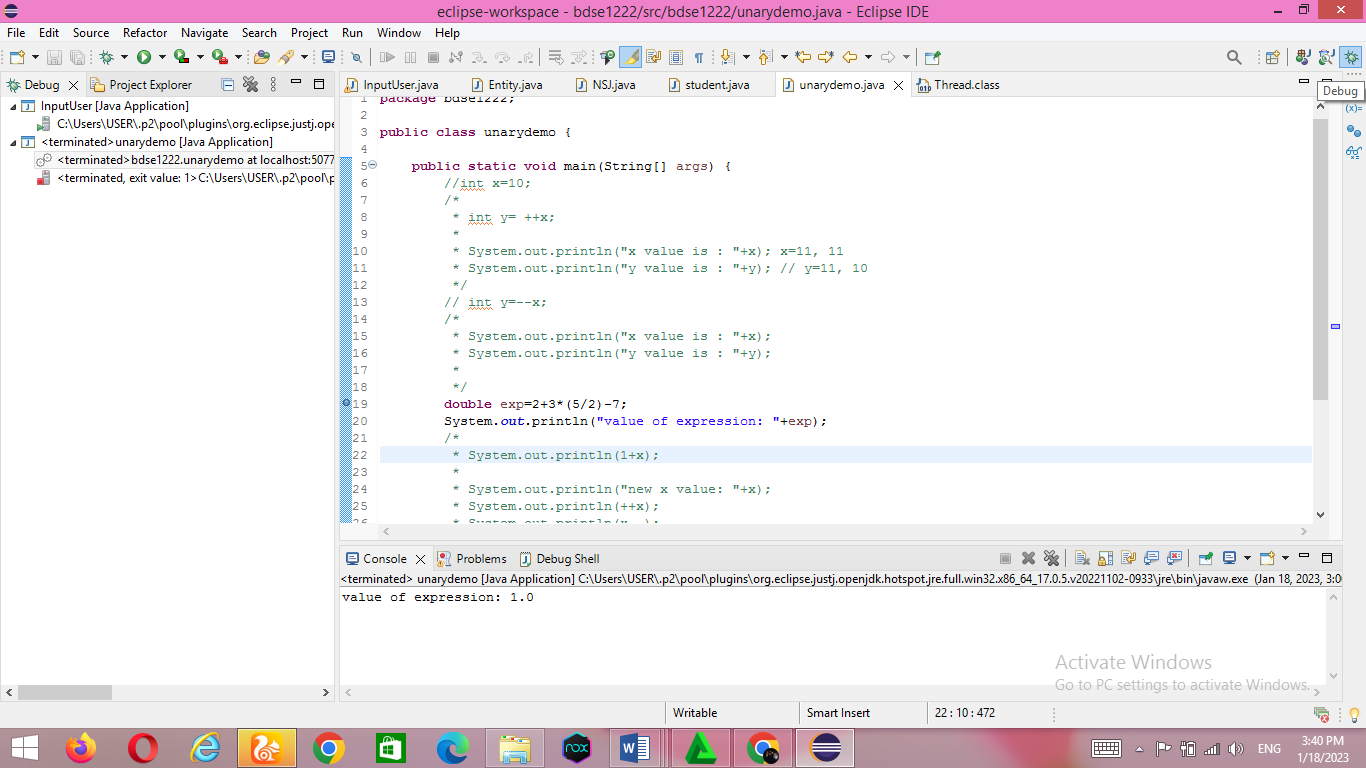
* breakpoint is useful for interrupting program execution to which it is applied. A breakpoint is a place in executable code where the operating system stops execution and switches to the debugger. This allows you to analyze your target and issue debugger commands.
* . Watch options become available when you right-click the watch code. Click the Watch button to display the Expression tab. Variables listed in the Expression tab are in scope of the tested method. You are free to create expressions based on the variables contained within the method's scope. If the variable does not exist in the scope of the method, an error will be displayed on the Expressions tab. So when you press the Step Into button that takes you to another method, you get an error because the variables in the formula you created were not recognized.
* Inspect Press Ctrl + Shift + I to see the output of the statement in a popup window. You don't need to save the file to see the output of a particular expression.
* step keys

There are three buttons that I use all the time in the Eclipse IDE. That is, the Step Over, Step Into, and Step Return buttons.

* Step over = easily row by row. However, the step over does not enter the body of the function.
* Step into = entry into a function.
* step return = step into function, opposite of step out
  1. Explain briefly the steps involved in debugging the developed application, along with screen captures.

.First, we need to create a breakpoint. Breakpoints here are useful to set where debugging starts/starts (not where the application starts, but where the programming process starts reading occurs. Debugging can be placed in the middle of the application ). To make it very easy, click twice (twice) to the left of the line number to display a dot as shown below.

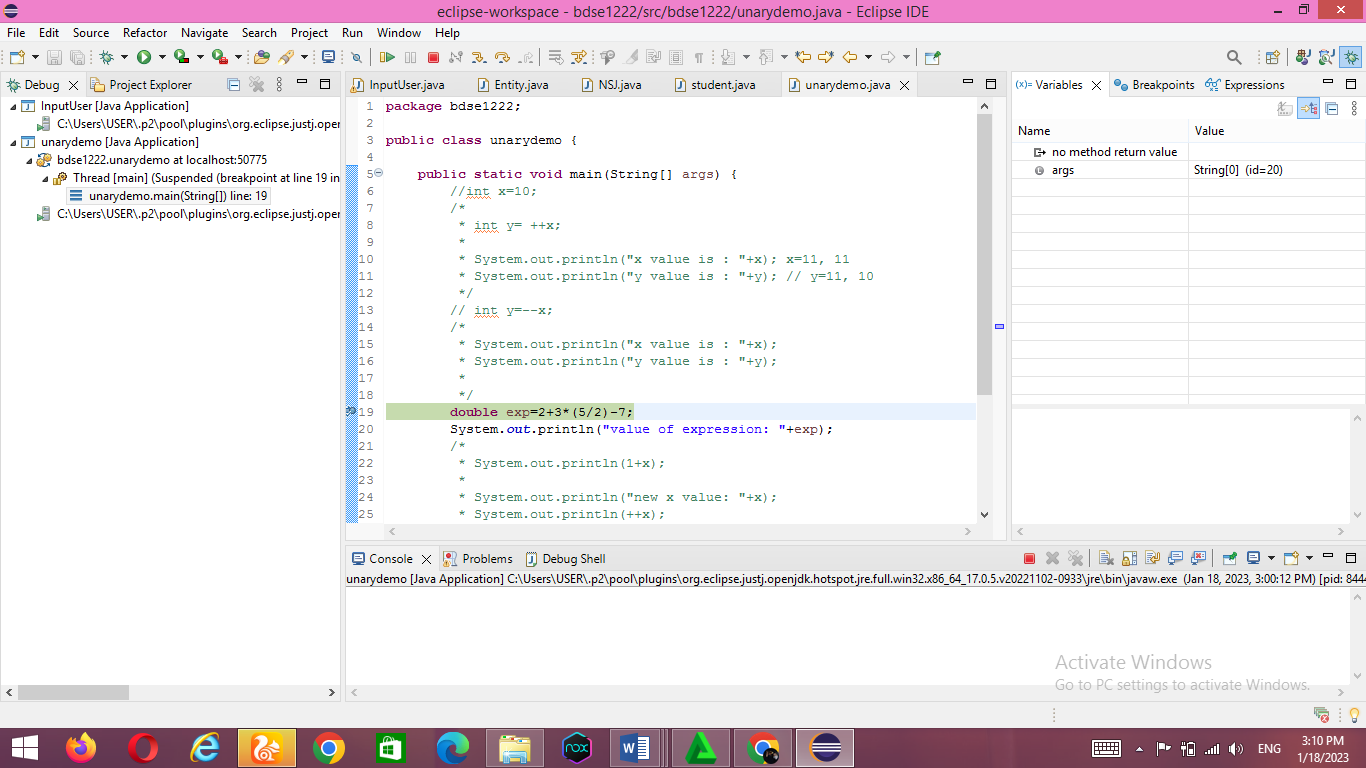
Second, open the debug properties to display the debug functions window so that we can find out what is actually running when we run the application we created



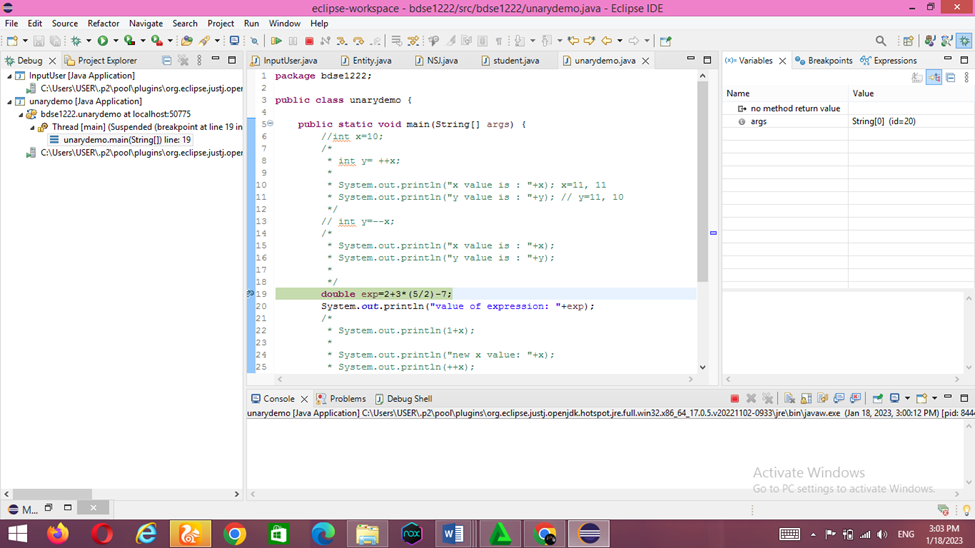
Select Debug Properties and you will get a layout similar to the image below. To return to normal, you can repeat the second step

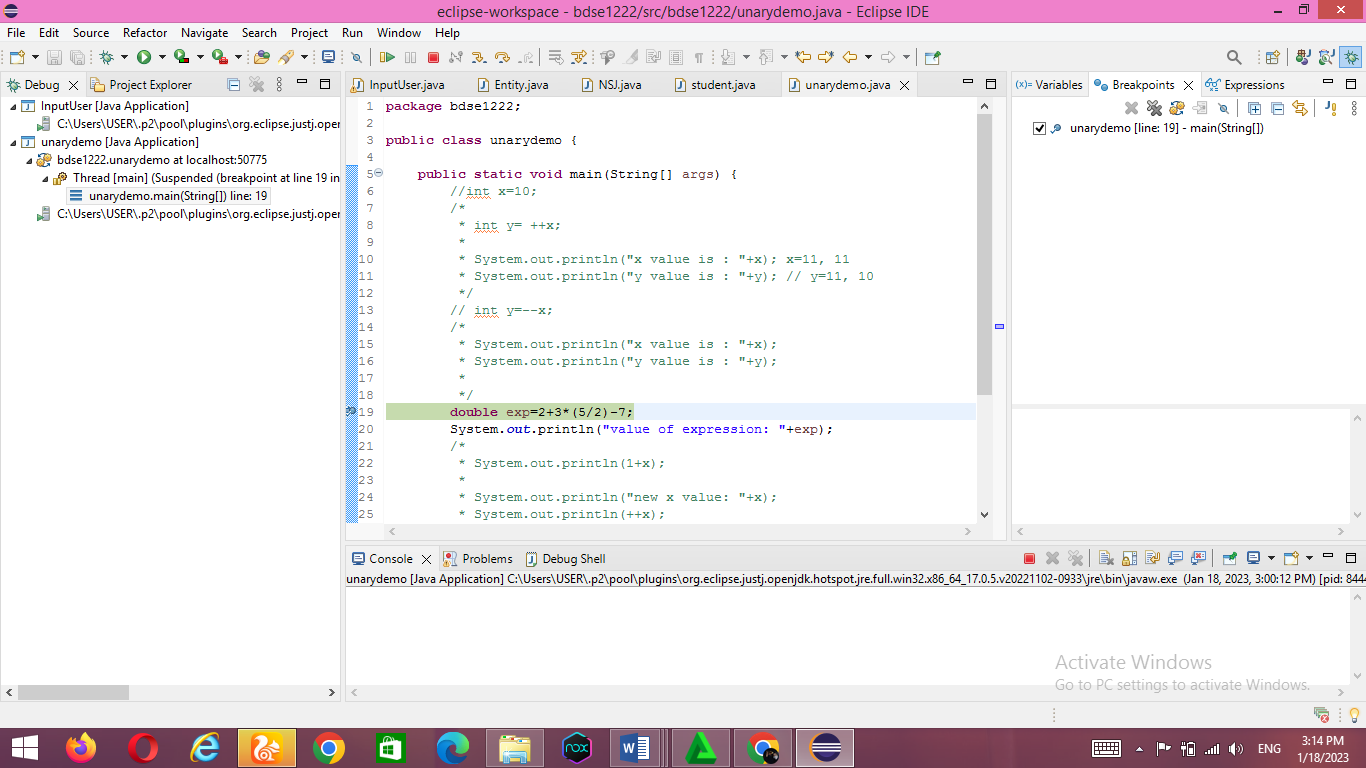


Third, start debugging by pressing the button as shown below



After that a debug will appear as below

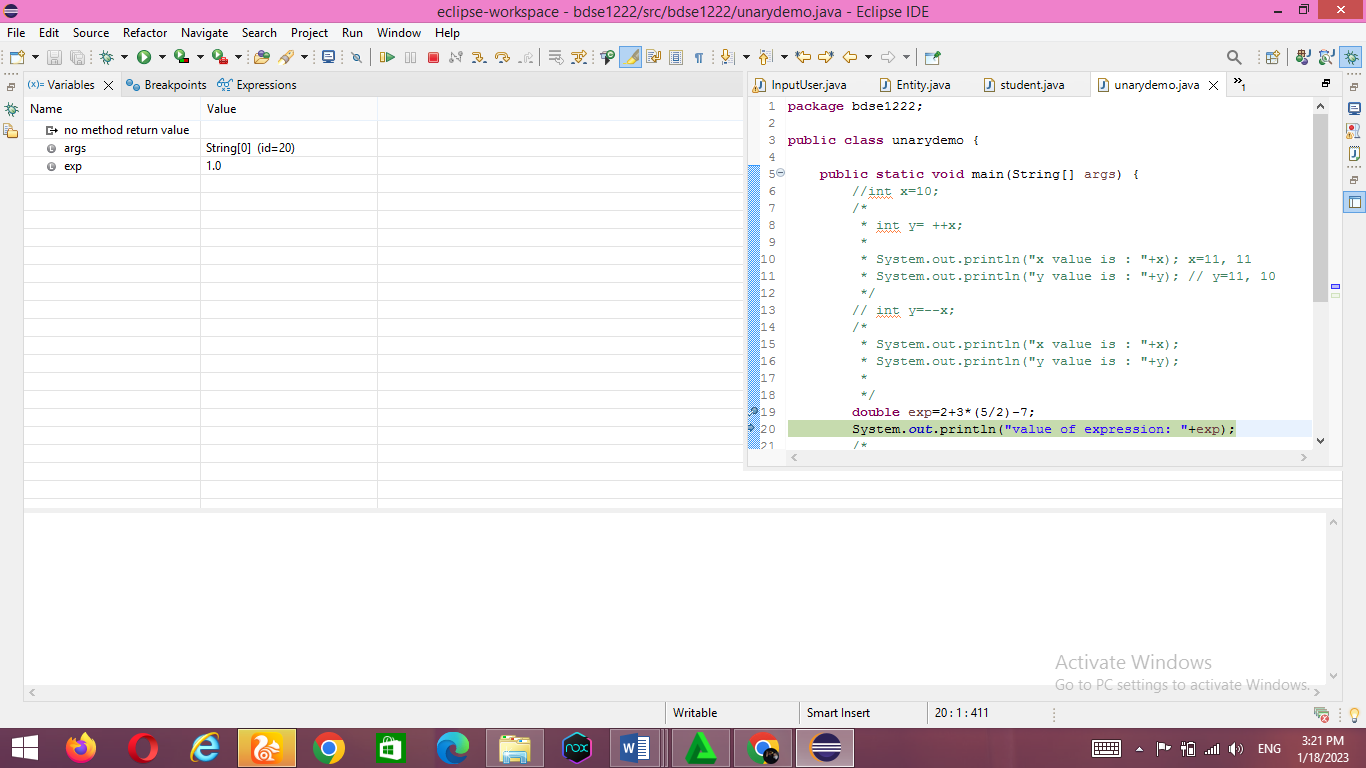


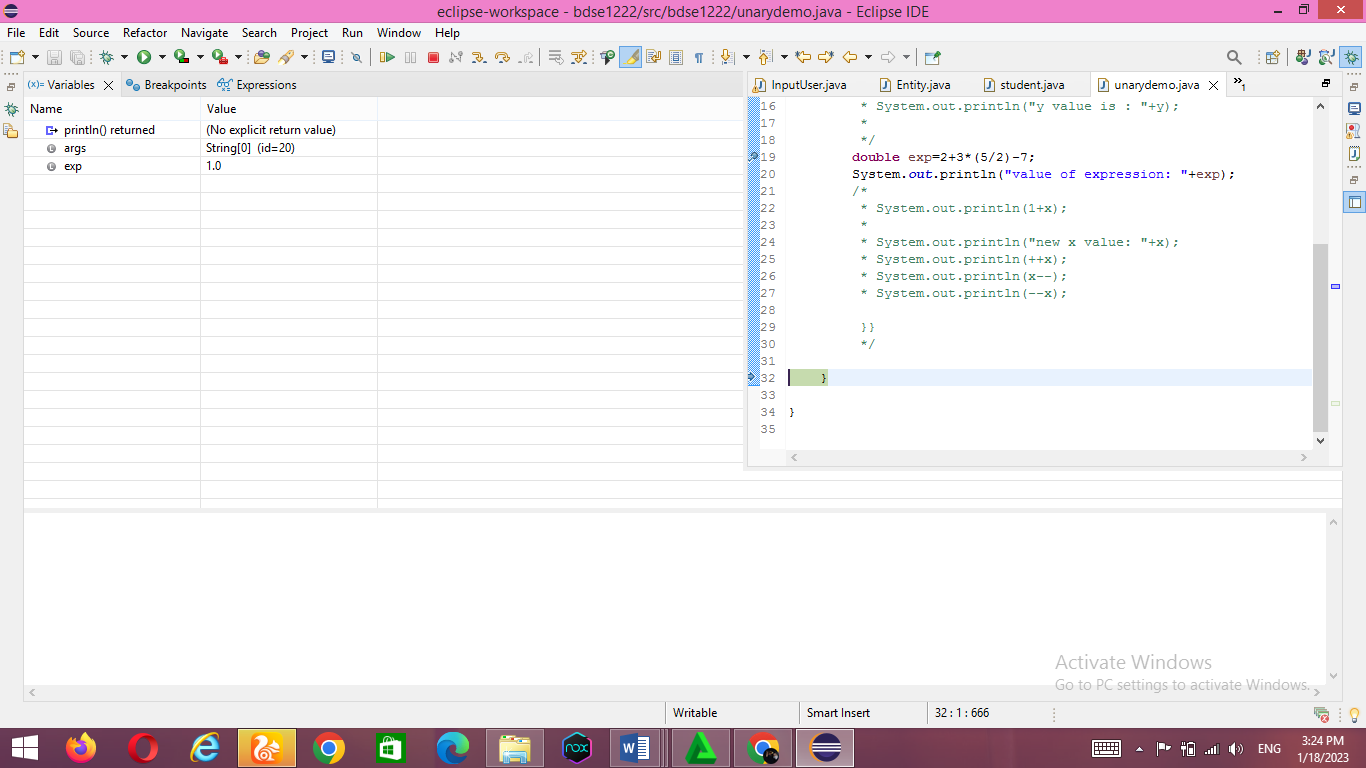


Fourth, click Step Over. simply line by line. However, the step over does not enter the body of the function. The Step Over button provides another way to step through your app's code at runtime. This code moves execution to the next line of code and speeds up the debugger



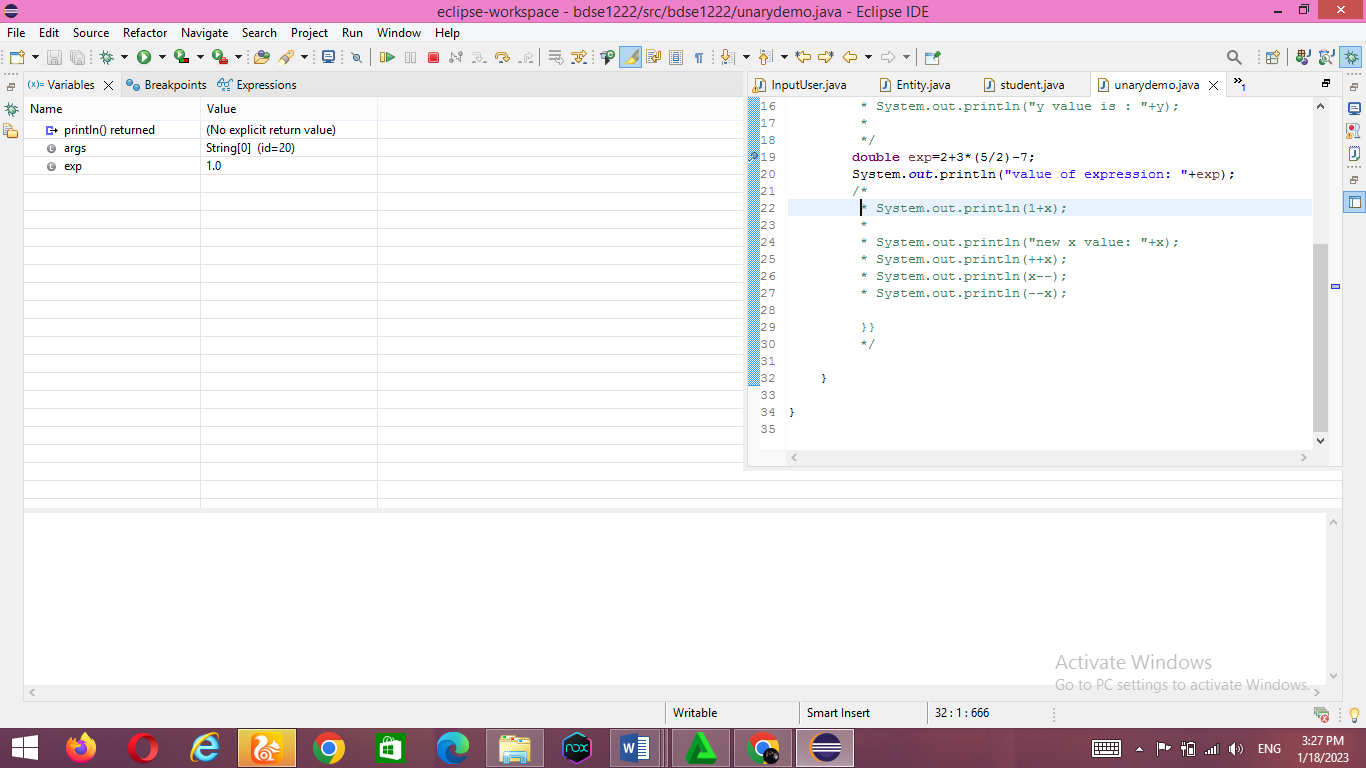
The result text below will come out, then click step over again.





After clicking again, the next result will appear

Click step into then it will enter into the function as shown below :



Click stop to end it



References :

<https://www.coursera.org/articles/types-programming-language>

<https://www.mygreatlearning.com/blog/inheritance-in-java/>

<https://www.geeksforgeeks.org/what-is-is-a-relationship-in-java/>