JAVA

Pre Assignment 1

إسلام محمد عطية محمد (20220126)

سكشن 1

Task1

a) Describe the evolution of modern programming languages.

Modern programming languages started from primitive, machine-specific languages like assembly to more human-readable and powerful options.

Early Days (1940s-1950s): Machine language ruled, then assembly language offered a slight improvement.

Rise of High-Level Languages (1950s-1970s): FORTRAN, COBOL, ALGOL, and BASIC brought more readable syntax and purpose-specific features.

Paradigm Shifts (1970s-1990s): C for efficiency, C++ for object-oriented programming (OOP), Pascal for structured programming, and scripting languages like Python for ease of use emerged.

Modern Era (2000s-Present): Java's platform independence and OOP focus, C#'s combination of C and Java, Domain-Specific Languages (DSLs) for specific tasks, scripting/web technologies like JavaScript, and security/concurrency concerns became prominent.

The Future: Readability, maintainability, developer productivity with advanced tools, and exploration of new paradigms like functional programming and AI-focused languages are at the forefront.

b) Define API, JDK, IDE, and other key terms related to Java programming.

API (Application Programming Interface):

It's a collection of prewritten building blocks, like classes and interfaces, that provide specific functionalities. You can leverage these functionalities in your code without reinventing the wheel. Java comes with a rich standard library API, but there are also third-party APIs for various purposes.

JDK (Java Development Kit):

This is your development workbench. It includes the tools you need to write, compile, and run Java programs. The JDK houses the Java compiler that translates your human-readable code into bytecode, a format the Java Virtual Machine (JVM) understands. It also includes a debugger for troubleshooting and other helpful utilities.

IDE (Integrated Development Environment):

An IDE is your command center for Java development. It provides a comprehensive environment with features like code editing, syntax highlighting, code completion, project management, and integration with the JDK tools. Popular IDEs for Java include IntelliJ IDEA, Eclipse, and NetBeans. While you can technically use the JDK tools from the command line, an IDE streamlines the development process significantly.

JVM (Java Virtual Machine):

It is Java's execution environment. It acts like a virtual computer, translating platform-neutral bytecode into instructions the underlying system understands. This enables "Write Once, Run Anywhere" (WORA) for Java programs.

c) Identify common uses for the Java programming language. (Applications)

Web Applications

Java servlets and frameworks like Spring enable developers to create dynamic web applications that run on servers. These web applications can handle complex user interactions and data processing.

Android Development

Though Kotlin is gaining popularity, Java remains a prominent language for developing Android applications. The Android SDK is based on Java, and a vast amount of Android code is written in Java.

Big Data and Machine Learning

Java frameworks like Apache Spark and H2O provide tools for distributed data processing and building machine learning algorithms. These frameworks can handle massive datasets efficiently.

Scientific Computing

Java libraries like SciJava offer functionalities for scientific computing and data analysis. Scientists can leverage these libraries for numerical computations, data visualization, and simulations.

Desktop Applications: While less prominent now, Java can still be used to develop desktop applications with graphical user interfaces (GUIs). These applications can run on various operating systems thanks to Java's platform independence.

d) Define key syntactical elements in Java.

Basic Building Blocks:

Keywords: Reserved words with specific meanings in Java. Examples include `class`, `public`, `static`, `void`, `if`, `else`, `for`, `while`.

Identifiers: Names given to elements like classes, variables, and methods. They follow naming conventions (start with letter/underscore, case-sensitive) and cannot be keywords.

Data Types: Define the kind of data a variable can hold (integers, characters, strings, etc.).

Structure and Organization:

Classes: Blueprints for creating objects. Define the properties (variables) and functionalities (methods) objects of that class will have.

Objects: Instances of classes that hold specific data values in their variables.

Methods: Define the behavior of an object. They can take inputs (parameters) and return outputs.

Control Flow:

Statements: Instructions that tell the program what to do.

Comments: Lines ignored by the compiler, used to explain code for human readers.

Expressions: Combinations of variables, operators, and method calls that evaluate to a value.

Operators: Perform operations on data (arithmetic, logical, comparison, etc.).

Flow Control Statements:

if-else: Control program flow based on conditions.

switch: Multi-way branching based on a value matching different cases.

Looping Statements: Repeat a block of code multiple times (`for`, `while`, `do-while`).

e) Illustrate the difference among syntax, runtime, and logic errors using examples.

Syntax Error

These errors violate the grammatical rules of the programming language. They prevent the program from even running and are usually caught early in the development process.

Ex:`int x = 6 // Missing semicolon`

Runtime Errors

These errors occur while the program is actually running.

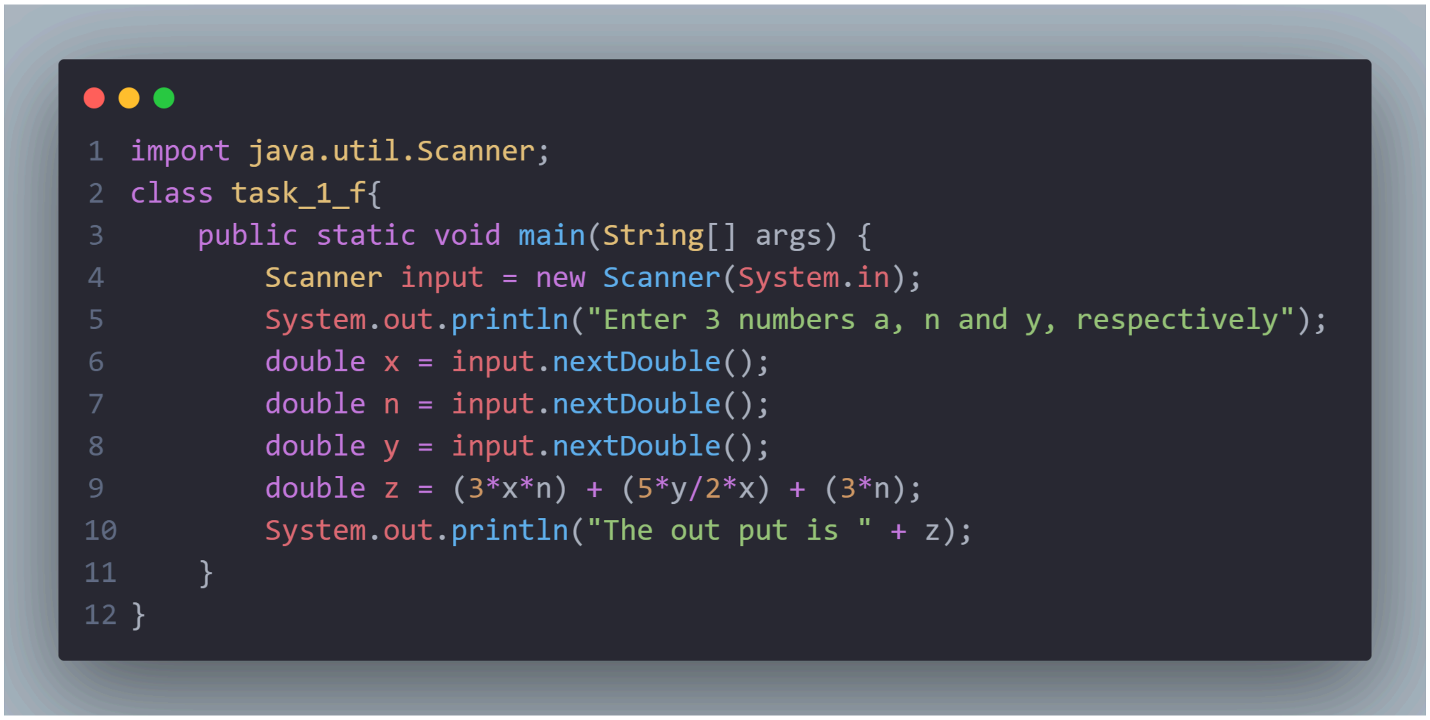
Ex: `while (true) System.out.println("Help!"); // Infinite Loop`

Logic Errors:

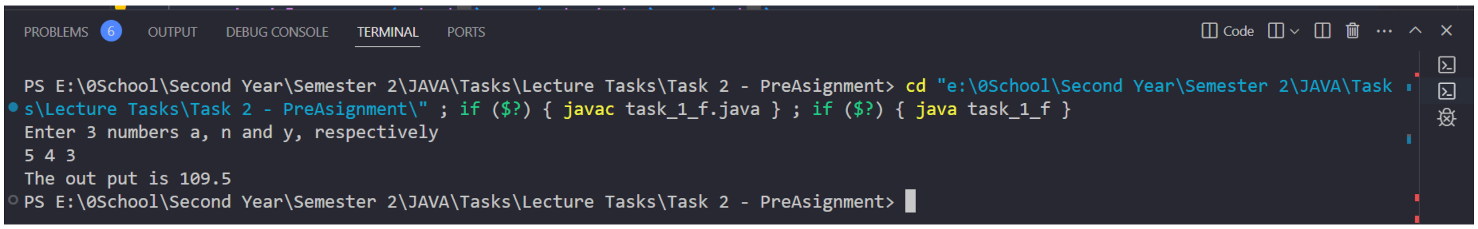
These errors are the trickiest to identify. The code itself might be syntactically correct and run without crashing, but it produces incorrect results due to flaws in the program's logic or intended behavior.

Ex:
```
int num1 = 11; int num2 = 22;

sum = num1 + num1;

// this should calculate the sum of num1 // and num2,
but it calculated the sum of // num1 twice
```

f) Write Java programs that take from the user the values of x, n and y and print z=3x*n+5y/2x+3n.



```java
import java.util.Scanner;
class task_1_f{
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.println("Enter 3 numbers a, n and y, respectively");
        double x = input.nextDouble();
        double n = input.nextDouble();
        double y = input.nextDouble();
        double z = (3*x*n) + (5*y/2*x) + (3*n);
        System.out.println("The out put is " + z);
    }
}
```



```
PS E:\0School\Second Year\Semester 2\JAVA\Tasks\Lecture Tasks\Task 2 - PreAsignment> cd "e:\0School\Second Year\Semester 2\JAVA\Task
s\Lecture Tasks\Task 2 - PreAsignment\" ; if ($?) { javac task_1_f.java } ; if ($?) { java task_1_f }
Enter 3 numbers a, n and y, respectively
5 4 3
The out put is 109.5
PS E:\0School\Second Year\Semester 2\JAVA\Tasks\Lecture Tasks\Task 2 - PreAsignment>
```

g) Describe the syntax of all selections and loops in the Java programming language.

- **If-else statement**

```
// if-else statement
if (condition) {
    // code to execute if condition is true
    }else { // code to execute if condition is
false (optional) }
```

- **Switch statement**

```
switch (expression) {
case value1:
// code to execute if expression equals value1
break;
case value2:
// code to execute if expression equals value2
break;
// ... more cases
default:
// code to execute if expression doesn't match
any case (optional)}
```

- **For Loop**

```
for (initialization; condition;
increment/decrement) {

// code to execute in each iteration

}
```

- **While Loop**

```
while (condition) {

// code to execute as long as condition is true

// (update statement can be placed anywhere within
the loop)

}
```

- **do-while Loop**

```
do {

// code to execute at least once

// (update statement can be placed anywhere within
// the loop)

} while (condition);
```

h) Describe the effect of the Java™ keywords break and continue when used in a loop.

Both `break` and `continue` are keywords used within loops in Java to alter the normal flow of the loop iteration. Here's how each one affects the loop:

1. break:

- **Effect:** Exits the loop entirely.
- **Behavior:**
    - When `break` is encountered within a loop, the loop terminates immediately, and control jumps to the statement following the entire loop construct.
    - It applies to both inner and outer loops if used within nested loops.

2. continue:

- **Effect:** Skips the remaining code in the current iteration and moves to the next iteration.
- **Behavior:**
    - When `continue` is encountered within a loop, the remaining code block for that particular iteration is skipped.
    - Control jumps back to the beginning of the loop to check the loop condition and potentially start the next iteration.

i) Apply Java™ naming conventions to variables, constants, methods, and classes using examples.

Java naming conventions promote code readability and maintainability. Here's how to name different elements following these conventions:

1. Variables:

- Use lowercase letters with camelCase for names with multiple words (first word lowercase, subsequent words uppercase).
- Descriptive names that reflect the variable's purpose.
- **Examples:**
  - `firstName` (stores a person's first name)
  - `isStudentActive` (boolean flag indicating student activity)
  - `totalOrderAmount` (stores the total amount of an order)

2. Constants:

- Use uppercase letters with underscores for word separation.
- Descriptive names that clearly indicate the constant value.
- **Examples:**
  - `MAX_ITEMS_PER_PAGE` (maximum number of items displayed per page)
  - `ACCOUNT_TYPE_ADMIN` (constant representing an admin account type)
  - `PI` (constant for the mathematical value pi)

3. Methods:

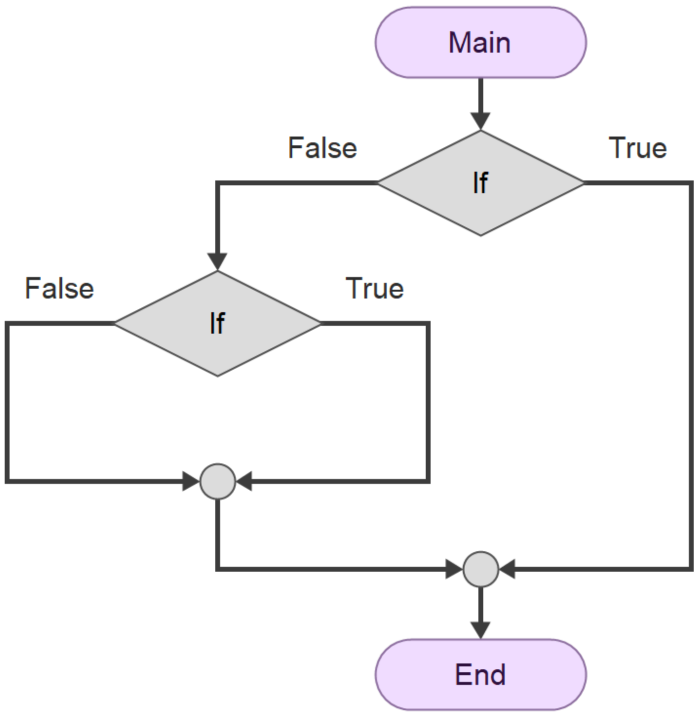
- Use lowercase letters with camelCase for names with multiple words.
- Verb-based names that describe the action the method performs.
- **Examples:**
    - `calculateArea(double length, double width)` (calculates the area)
    - `displayWelcomeMessage(String userName)` (displays a welcome message)
    - `isValidEmail(String email)` (checks if an email is valid)

4. Classes:

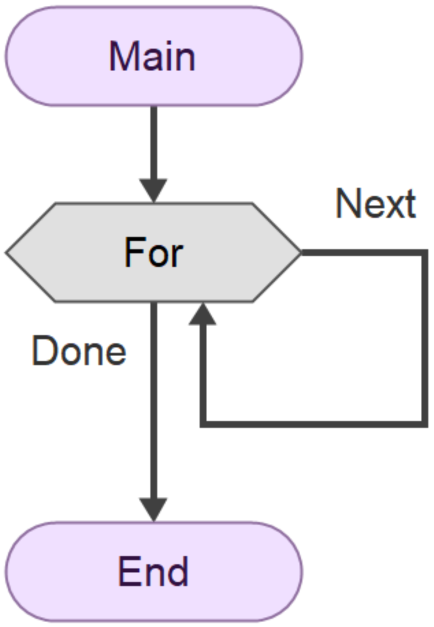
- Use uppercase letters with PascalCase (all words start with uppercase letters).
- Noun-based names that describe the object the class represents.
- **Examples:**
    - `Customer` class represents a customer object
    - `OrderProcessor` class processes orders
    - `FileHandler` class handles file operations

j) Predict the behavior of selections and loops using the flowcharts.

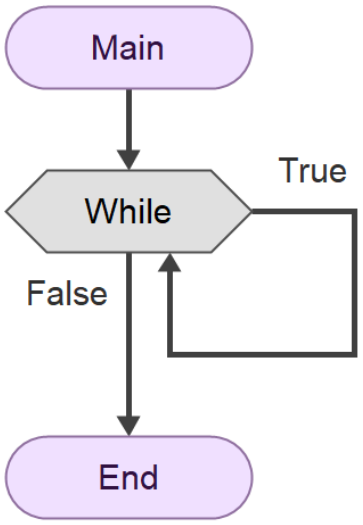
- **If-else statement**



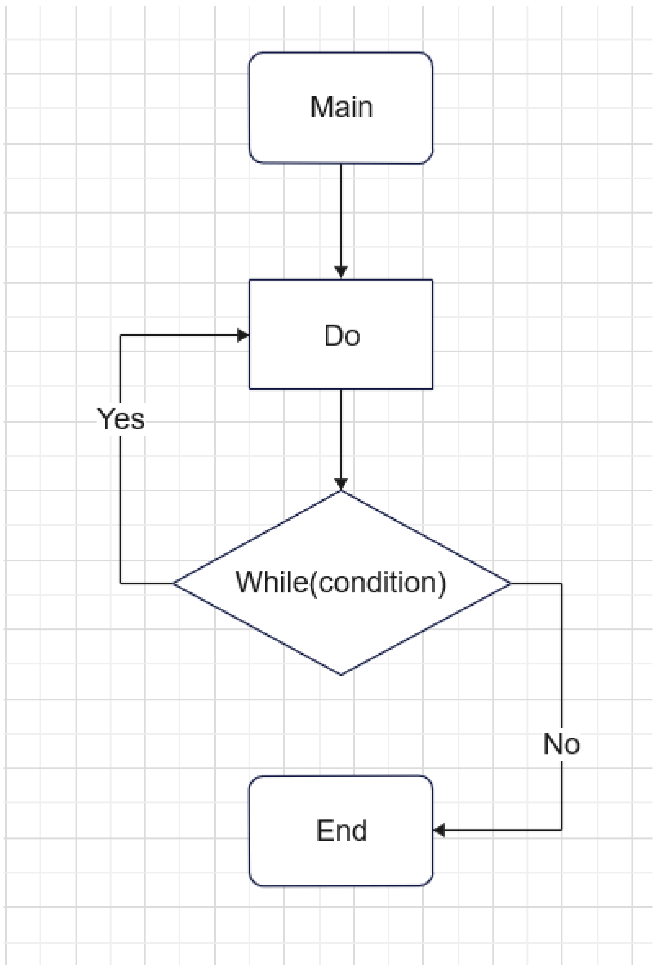
- **For Loop**



- **While Loop**



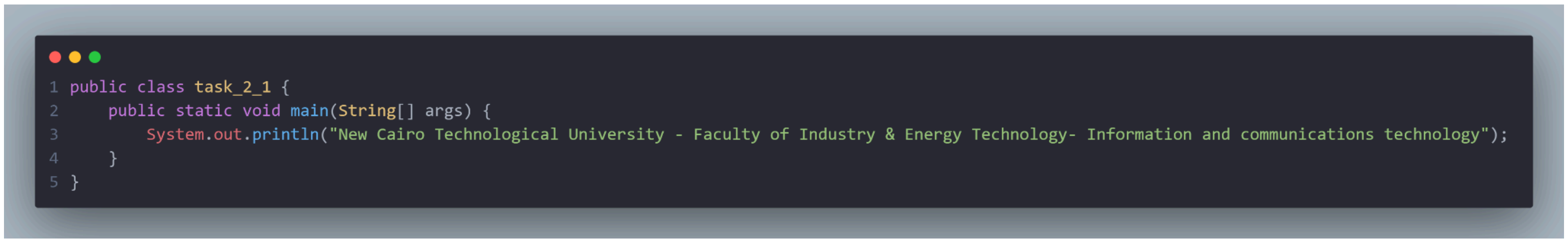
- **do-while Loop**



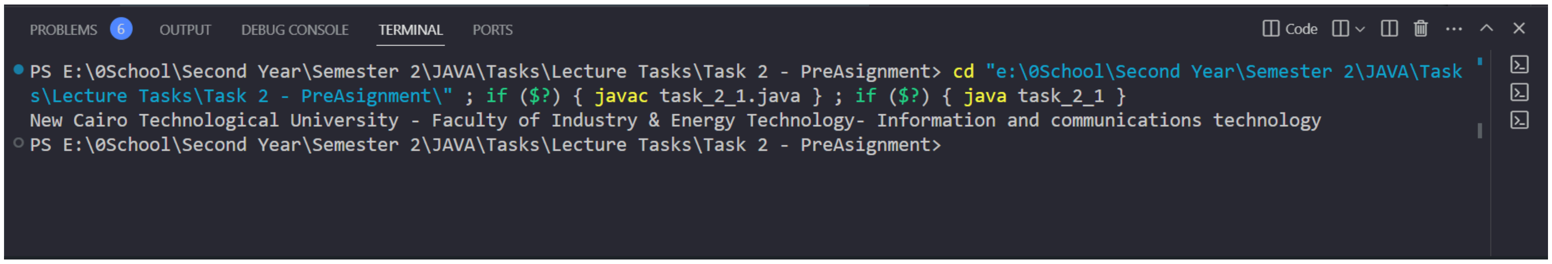
Task 2

Write, compile, and execute a simple Java program that prints the string " New Cairo Technological University - Faculty of Industry & Energy Technology- Information and communications technology".

a) Create a programming environment using an IDE and the Java Development Kit. Then, implement the above program in this environment.



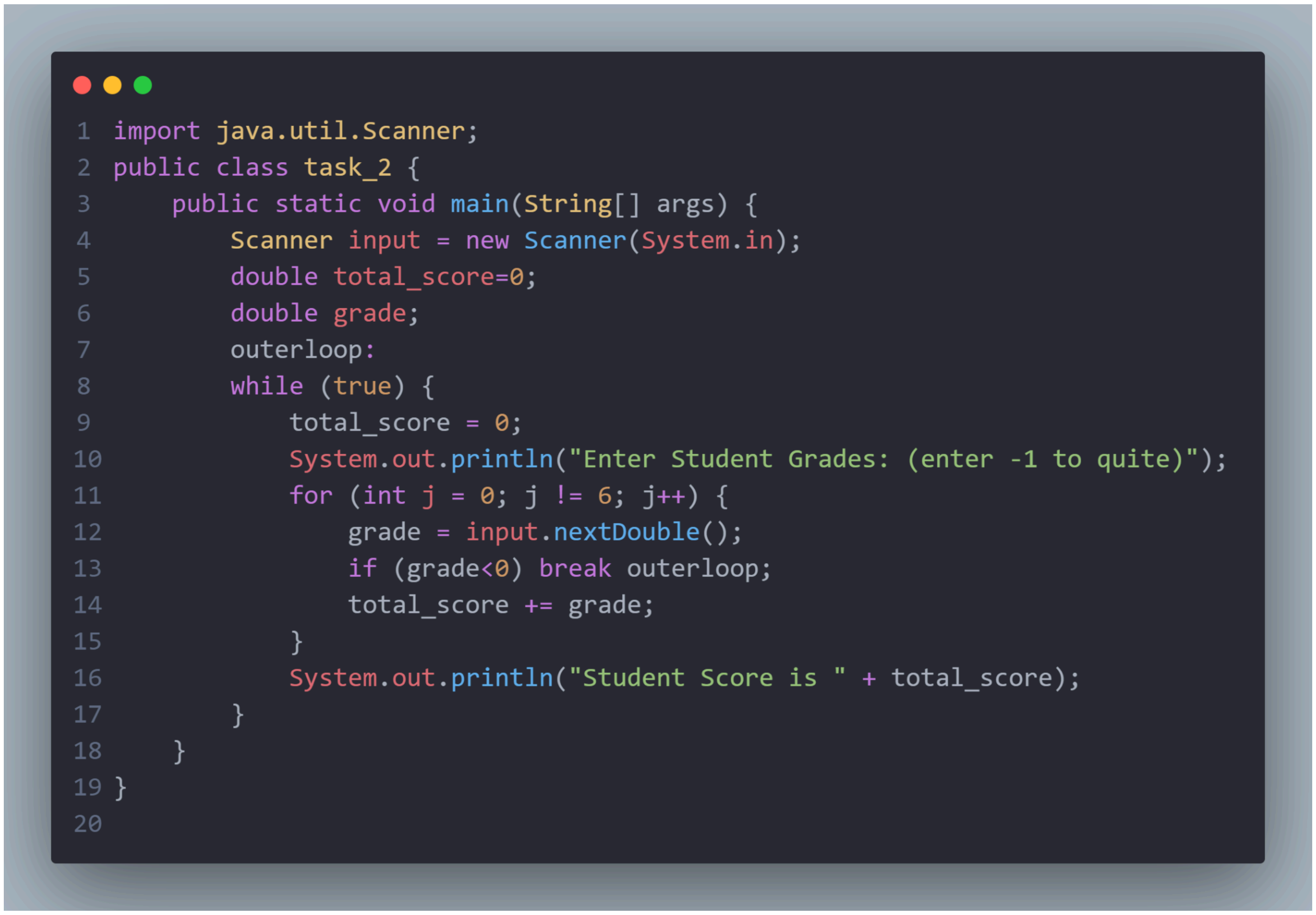
```java
public class task_2_1 {
    public static void main(String[] args) {
        System.out.println("New Cairo Technological University - Faculty of Industry & Energy Technology- Information and communications technology");
    }
}
```



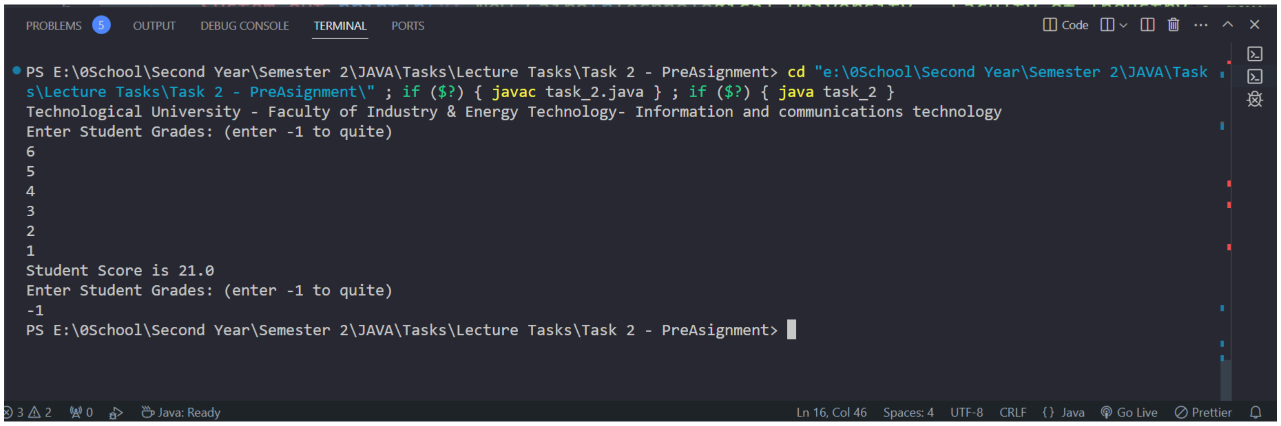
```
PS E:\0School\Second Year\Semester 2\JAVA\Tasks\Lecture Tasks\Task 2 - PreAsignment> cd "e:\0School\Second Year\Semester 2\JAVA\Tasks\Lecture Tasks\Task 2 - PreAsignment\" ; if ($?) { javac task_2_1.java } ; if ($?) { java task_2_1 }
New Cairo Technological University - Faculty of Industry & Energy Technology- Information and communications technology
PS E:\0School\Second Year\Semester 2\JAVA\Tasks\Lecture Tasks\Task 2 - PreAsignment>
```

b) Implement a java program in 5 ways. This program takes from a user the grades of students in 6 courses to compute the sum of grades for all students (hint the number of students is unknown).

- **Way 1**

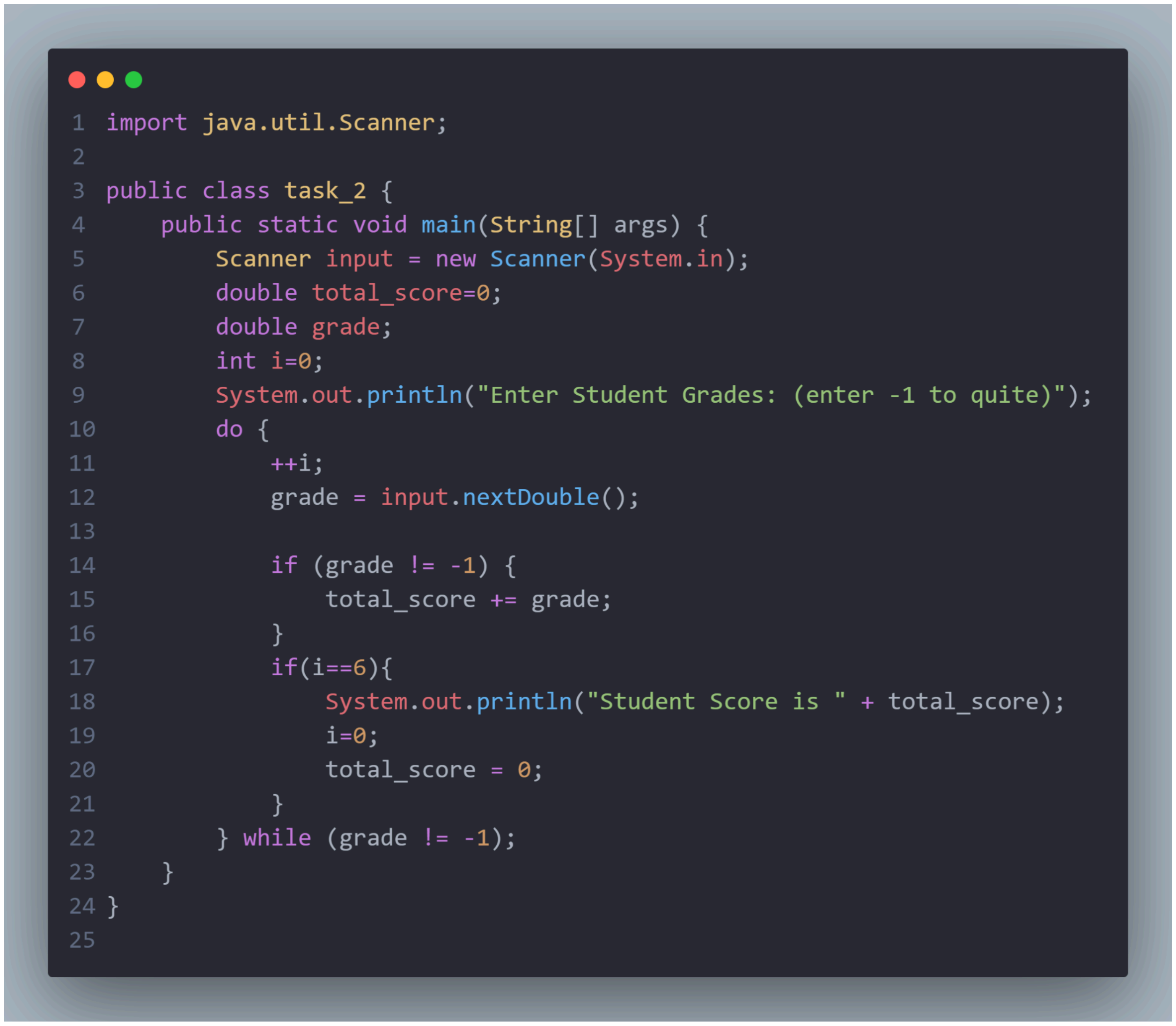


```java
import java.util.Scanner;
public class task_2 {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        double total_score=0;
        double grade;
        outerloop:
        while (true) {
            total_score = 0;
            System.out.println("Enter Student Grades: (enter -1 to quite)");
            for (int j = 0; j != 6; j++) {
                grade = input.nextDouble();
                if (grade<0) break outerloop;
                total_score += grade;
            }
            System.out.println("Student Score is " + total_score);
        }
    }
}
```

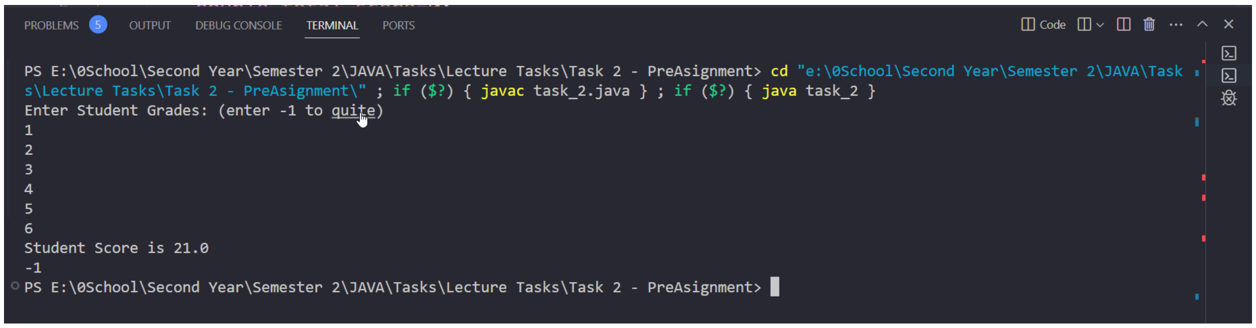


```
PS E:\0School\Second Year\Semester 2\JAVA\Tasks\Lecture Tasks\Task 2 - PreAsignment> cd "e:\0School\Second Year\Semester 2\JAVA\Task
s\Lecture Tasks\Task 2 - PreAsignment\" ; if ($?) { javac task_2.java } ; if ($?) { java task_2 }
Technological University - Faculty of Industry & Energy Technology- Information and communications technology
Enter Student Grades: (enter -1 to quite)
6
5
4
3
2
1
Student Score is 21.0
Enter Student Grades: (enter -1 to quite)
-1
PS E:\0School\Second Year\Semester 2\JAVA\Tasks\Lecture Tasks\Task 2 - PreAsignment>
```

- **Way 2**

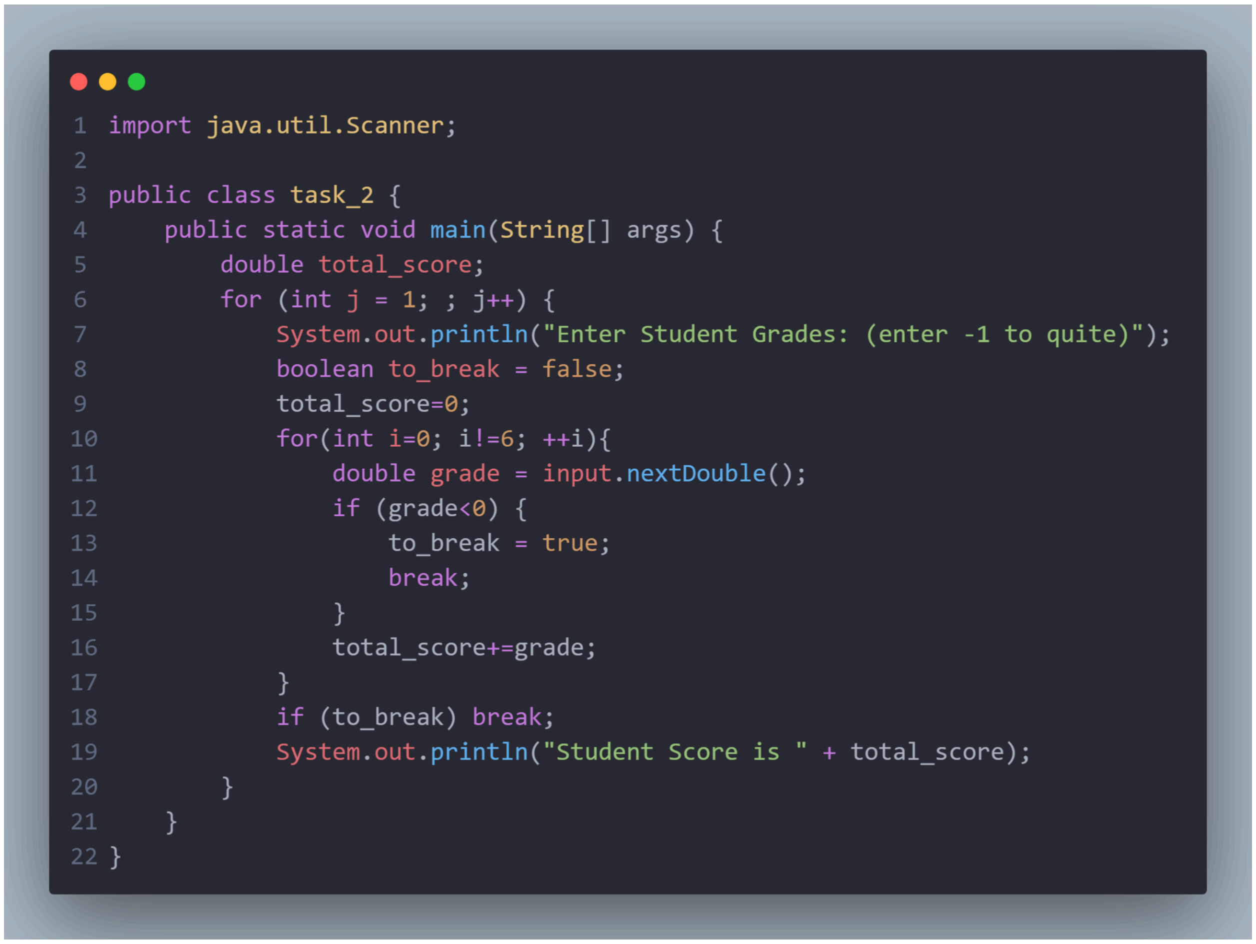


```java
import java.util.Scanner;

public class task_2 {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        double total_score=0;
        double grade;
        int i=0;
        System.out.println("Enter Student Grades: (enter -1 to quite)");
        do {
            ++i;
            grade = input.nextDouble();

            if (grade != -1) {
                total_score += grade;
            }
            if(i==6){
                System.out.println("Student Score is " + total_score);
                i=0;
                total_score = 0;
            }
        } while (grade != -1);
    }
}

```

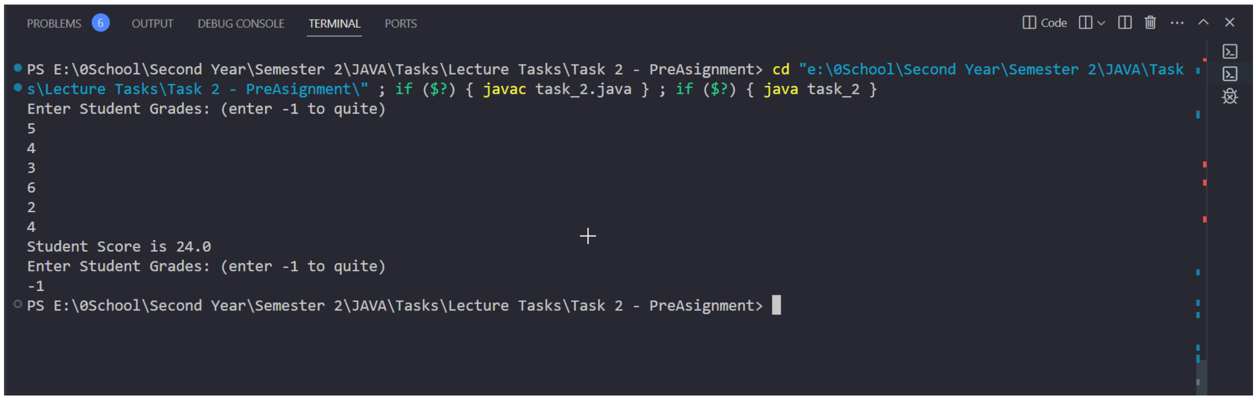


```
PS E:\0School\Second Year\Semester 2\JAVA\Tasks\Lecture Tasks\Task 2 - PreAsignment> cd "e:\0School\Second Year\Semester 2\JAVA\Task
s\Lecture Tasks\Task 2 - PreAsignment\" ; if ($?) { javac task_2.java } ; if ($?) { java task_2 }
Enter Student Grades: (enter -1 to quite)
1
2
3
4
5
6
Student Score is 21.0
-1
PS E:\0School\Second Year\Semester 2\JAVA\Tasks\Lecture Tasks\Task 2 - PreAsignment>
```

- Way 3

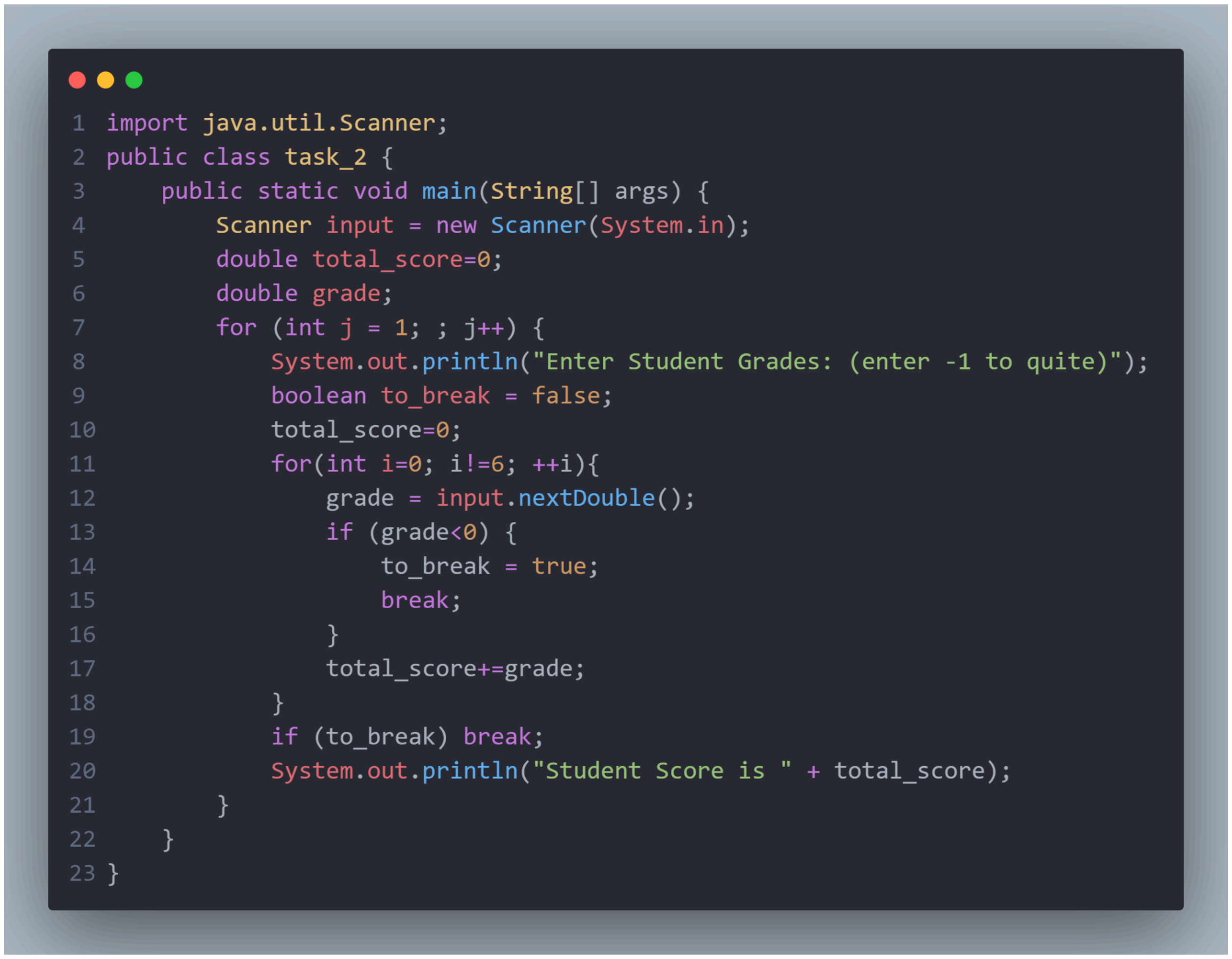


```java
import java.util.Scanner;

public class task_2 {
    public static void main(String[] args) {
        double total_score;
        for (int j = 1; ; j++) {
            System.out.println("Enter Student Grades: (enter -1 to quite)");
            boolean to_break = false;
            total_score=0;
            for(int i=0; i!=6; ++i){
                double grade = input.nextDouble();
                if (grade<0) {
                    to_break = true;
                    break;
                }
                total_score+=grade;
            }
            if (to_break) break;
            System.out.println("Student Score is " + total_score);
        }
    }
}
```

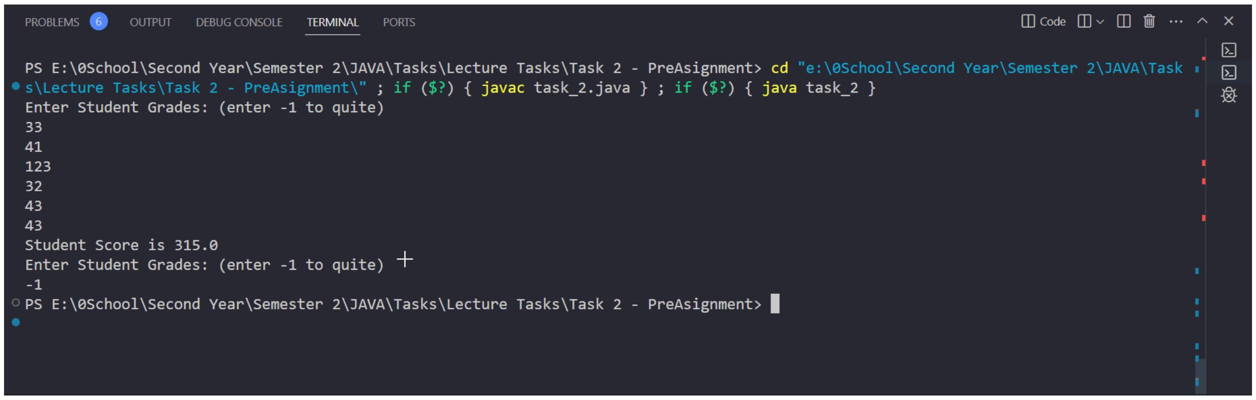


```
PS E:\0School\Second Year\Semester 2\JAVA\Tasks\Lecture Tasks\Task 2 - PreAsignment> cd "e:\0School\Second Year\Semester 2\JAVA\Task
s\Lecture Tasks\Task 2 - PreAsignment\" ; if ($?) { javac task_2.java } ; if ($?) { java task_2 }
Enter Student Grades: (enter -1 to quite)
5
4
3
6
2
4
Student Score is 24.0
Enter Student Grades: (enter -1 to quite)
-1
PS E:\0School\Second Year\Semester 2\JAVA\Tasks\Lecture Tasks\Task 2 - PreAsignment>
```

- **Way 4**

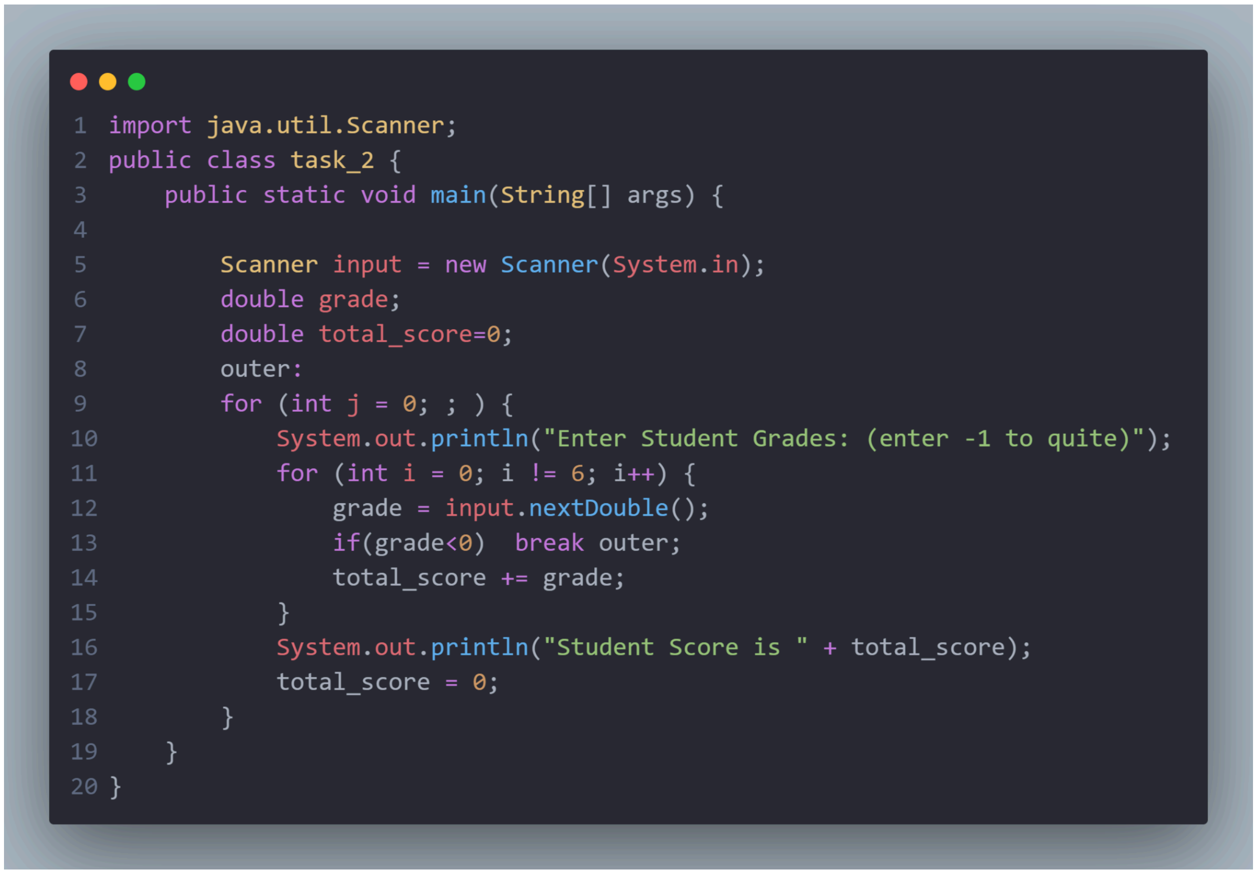


```java
import java.util.Scanner;
public class task_2 {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        double total_score=0;
        double grade;
        for (int j = 1; ; j++) {
            System.out.println("Enter Student Grades: (enter -1 to quite)");
            boolean to_break = false;
            total_score=0;
            for(int i=0; i!=6; ++i){
                grade = input.nextDouble();
                if (grade<0) {
                    to_break = true;
                    break;
                }
                total_score+=grade;
            }
            if (to_break) break;
            System.out.println("Student Score is " + total_score);
        }
    }
}
```

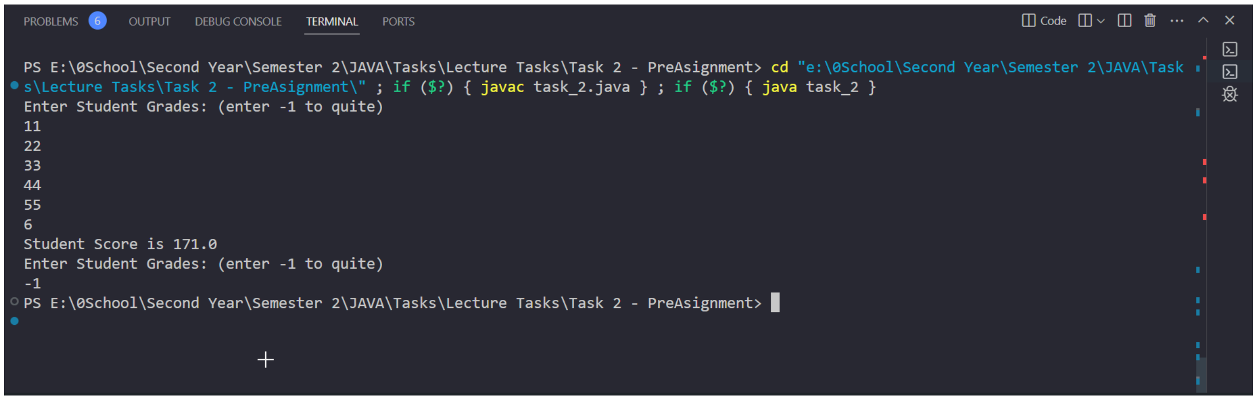


```
PS E:\0School\Second Year\Semester 2\JAVA\Tasks\Lecture Tasks\Task 2 - PreAsignment> cd "e:\0School\Second Year\Semester 2\JAVA\Task
s\Lecture Tasks\Task 2 - PreAsignment\" ; if ($?) { javac task_2.java } ; if ($?) { java task_2 }
Enter Student Grades: (enter -1 to quite)
33
41
123
32
43
43
Student Score is 315.0
Enter Student Grades: (enter -1 to quite)
-1
PS E:\0School\Second Year\Semester 2\JAVA\Tasks\Lecture Tasks\Task 2 - PreAsignment>
```

- **Way 5**



```java
import java.util.Scanner;
public class task_2 {
    public static void main(String[] args) {

        Scanner input = new Scanner(System.in);
        double grade;
        double total_score=0;
        outer:
        for (int j = 0; ; ) {
            System.out.println("Enter Student Grades: (enter -1 to quite)");
            for (int i = 0; i != 6; i++) {
                grade = input.nextDouble();
                if(grade<0)  break outer;
                total_score += grade;
            }
            System.out.println("Student Score is " + total_score);
            total_score = 0;
        }
    }
}
```



```
PS E:\0School\Second Year\Semester 2\JAVA\Tasks\Lecture Tasks\Task 2 - PreAsignment> cd "e:\0School\Second Year\Semester 2\JAVA\Task
s\Lecture Tasks\Task 2 - PreAsignment\" ; if ($?) { javac task_2.java } ; if ($?) { java task_2 }
Enter Student Grades: (enter -1 to quite)
11
22
33
44
55
6
Student Score is 171.0
Enter Student Grades: (enter -1 to quite)
-1
PS E:\0School\Second Year\Semester 2\JAVA\Tasks\Lecture Tasks\Task 2 - PreAsignment>
```

Task 3

Explain the programming style with an Example of Documentation, Vertical Alignment, Comments, Indentation, Meaningful Identifier Names Consistently Typed, and Appropriate use of Typedef with an Example

Documentation: Clear explanations about the code's purpose, usage, and non-obvious logic (e.g., comments at the beginning of functions or classes).

Vertical Alignment: Aligning code elements visually improves readability (e.g., proper indentation for `if` statements and code blocks).

Comments: Explain complex logic or non-obvious sections for better understanding.

Indentation: Creates hierarchy, indicating code blocks and control flow (e.g., consistent indentation using spaces or tabs).

Meaningful Identifier Names: Use descriptive variable/function names that reflect their purpose and data type (e.g., `customerCount` instead of `c`).

Consistent Typing: Use consistent data types for variables and function arguments (e.g., always use `int` for whole numbers).

`typedef` (not directly applicable in some languages):

While not present in all languages (like Java), `typedef` allows creating aliases for complex data types, making code more concise (e.g., `typedef std::vector<int> IntVector` in C++).