Sino-japanese: Military Problem Outline

Table of Contents

[PROJECT OVERVIEW: 4](#_Toc175655346)

[Introduction: 6](#_Toc175655347)

[Classes And Objects 7](#_Toc175655348)

[Methods 8](#_Toc175655349)

[Object-Oriented Programming (OOP) 9](#_Toc175655350)

[Enumeration (Enum), Events and Delegates 11](#_Toc175655351)

[References 14](#_Toc175655353)

# 

# PROJECT OVERVIEW:

**Scenario:** The project simulates a situation where China is responding to a surprise military attack by Japan on islands in the East China Sea. China needs to quickly deploy troops and supplies while dealing with challenges like limited infrastructure, potential enemy interference, and resource constraints.

**Objective:** Create a system to manage and optimize the supply chain for delivering military supplies efficiently and cost-effectively under these challenging conditions.

**Core Functionalities:**

1. **Supply Chain Mapping:**
   * **Goal:** Build a digital map of the supply network, showing where supplies are stored and how they are transported.
   * **How:** visualize the network with points (nodes) for supply locations and connections (edges) for transport routes.
2. **Demand Forecasting:**
   * **Goal:** Predict future needs for supplies based on past data and current plans.
   * **How:** Analyze historical data to estimate future supply requirements.
3. **Route Optimization:**
   * **Goal:** Find the best routes for transporting supplies to minimize time and cost, while avoiding risks.
   * **How:** Use algorithms to calculate the shortest and most efficient routes, considering dynamic factors like enemy actions.
4. **Inventory Management:**
   * **Goal:** Keep track of inventory levels at different locations to ensure there’s enough supply and avoid shortages.
   * **How:** Use data structures to monitor and manage stock levels.
5. **Resource Allocation:**
   * **Goal:** Distribute limited resources like vehicles and personnel effectively to meet supply needs.
   * **How:** Develop methods to allocate resources based on priority and demand.

**Simulation Scenarios:**

1. **Scenario 1:** Simulate an attack by Japan and test how well the supply system supports the Chinese military’s response.
2. **Scenario 2:** Examine how weather conditions like storms or fog affect supply operations.
3. **Scenario 3:** Test different strategies for routing supplies to avoid enemy detection and improve efficiency.

**Challenges:**

* **Dynamic Environments:** The system must adapt to rapidly changing situations.
* **Resource Constraints:** Limited resources must be managed carefully.
* **Security:** Protect sensitive information and secure supply routes.
* **Complexity:** Handle a complex network of supply points and routes.

The project aims to develop a system that helps the Chinese military effectively manage its supply chain during a conflict, ensuring timely and efficient delivery of supplies while addressing various challenges and risks.

# Introduction:

The purpose of this topic identification will be to define and looking further into the following learning outcomes:

* Classes and Objects
* Methods
* OOPs
* Enums, Events and Delegates
* Window Forms

The is to explore each of the above outcomes in detail, to understand their purposes and why they are important when trying to create a C# application.

## Classes And Objects

A class is a template that inherits variables and methods once an individual object is created. An object is an instance of a class - meaning it is something we create from a class – and align with real world item. One can think of a class as a blueprint the user has defined, and this is where the object can be created. The class will also join fields and methods into one unit, and supports polymorphism, inheritance and a concept of base and sub-classes; these topics will be covered later in the topic identification.

It is imperative to declare a class. To declare a class, the keyword ‘class’ is used and an ‘identifier’ or name follows. Optional attributes can be used when declaring a class, depending on the requirements of the application the user would like to create. When declaring a class the following components can be utilized:

* Modifiers
* ‘Class’ Keyword
* Identifier
* Base/Super Class
* Interfaces
* Body
* Constructors

Modifiers define whether a class is public or internal – by default a class is internal. The identifier is a variable type class that is provided and begins with a capital first letter. The base or super class is the parent class and there is an interface that is implemented that is proceeded by a colon and can be more than one if the user wishes to implement more. Constructors initialize new objects. The final component (the body) is a class that has curly braces ( {} ).

An object is declared once it has been created and the class is instantiated. Every instance created shares attributes and class behaviours. An object consists of the following components:

* State
* Behavior
* Identity

The state and behavior components are represented by an object’s methods and reflects another object’s properties, while the identity gives the objects a unique name and allows objects to interact with other objects.

## Methods

A method is a block or statement of code run when it is called. It performs certain tasks – also known as functions – and returns a result once called. It can be reused to save memory, time and provides the code with better readability. Methods contain the following component: a parameter. A parameter is data that can be passed into the method and is specified after the name of the method name, inside parentheses. There are also default parameters that use the equal sign, which are methods that are called, but have no argument.

Methods are declared through constructing it. They contain certain components that make it up: the first being the modifier. A modifier defines the access type of the method – private, public or protected. The second component is the method signature. It consists of two parameters: a method name and a parameter list. The method refers to the name the user will use to call the method, while the parameter list is a list of input parameters defined and preceded with their data types. Next is the body, which is the line of code of tasks that the method will perform while executing. The method is then declared, and the user can decide whether to implement it or not.

## Object-Oriented Programming (OOP)

Involves creating objects that have both data and methods within them. OOP allows programs to be faster, execution to be easier and provides a clear programmable structure. It makes it possible for applications with less code to be created with fully reusable applications. Within OOP are four components that make up the fundamentals of OOP:

* Inheritance
* Polymorphism
* Abstraction
* Interface

#### Inheritance

Inheritance allows new classes to be defined using existing classes. It looks at the base class (the parents) and the derived class (the child class). Properties and methods from the existing class – the base class – can be used in addition to brand new methods and properties of the new class – the derived class. It assists in promoting reusing code, makes code maintenance simpler and improves the organization of the code. There are four main types of inheritance are the following:

* + **Single inheritance:** the derived class inherits fields and methods from a single base class.
  + **Multi-level inheritance:** the derived class inherits fields and methods from a base class, and then the derived class itself becomes a base class to another derived class.
  + **Hierarchical inheritance:** the base class is the parent class pf two or more derived classes.
  + **Multiple inheritance:** the derived class inherits fields and methods from two or more base classes.

#### Polymorphism

Polymorphism refers to ‘many forms’ and occurs when there are multiple classes related to each other by inheritance. Many methods are used to perform many different classes, which then allows the user to perform a single action in a multitude of ways.

Polymorphism comes in either compile time (static) or run-time (static). Compile time identifies which method is being called. This can be done by firstly, method overloading by creating methods with the same name, but different numbers and types of parameters, and secondly, operator overloading which performs numeric and string concatenation. Run-time is achieved through overloading, which occurs when a derived class overrides a base class with the same method.

#### Abstraction

This involves hiding details and showing the user only what they need to see. There can be an abstract class – which is a class that cannot be used to create objects due to it being restricted – and an abstract – which can only be used in an abstract class. Abstract classes are declared through the ‘abstract’ keyword, but new operators cannot be used to directly in this class. There is a base class that contains all the methods associated with the class.

Abstract classes allow the base class to be defines, without a complete implementation of all the methods. It provides a general form that all other derived classes will utilize. Abstraction has a slight difference from encapsulation which hides the information the user doesn’t see. Encapsulation also groups data and methods together. Abstraction is good for reducing the complexity of viewing the application, it avoids duplicating code and helps better application security.

#### Interface

This is another way to achieve abstraction, and this is done through implementation in the body. Properties, methods, events and indexers can be done within an interface, but it only declares the members. It specifies what the class should do and how it should be done. It can’t have any private members, so by default the interface is public. Interfaces are good for loose coupling and total abstraction. They can achieve component based programming and it’s possible to inherit multiple inheritance and abstraction.

## Enumeration (Enum), Events and Delegates

An enum represents a groups of read-only constants. It is a value data type that also acts as a special class used to define the user’s own data types. Its main use is to assign name or string values to integral constants to make the program easy to read and maintain.

Events (and event handlers) are used when referring to the Publisher’s (which is an event that invokes a conditions of sorts) and the Subscriber’s (which is notified based on the event’s subscription) design pattern, where a certain event happens in a certain class.

A delegate is a method that is called once an event is triggered. Delegates are a library class in System namespace and are a type-safe pointer of any method. They are a good way to encapsulate methods and two or more of these methods can be called on a single event. The main function of delegates is to implement call-back methods and events. EventHandlers are also delegates. They have a void return type method and two input parameters: the object – the source or publisher that invoked the event – and the EventArgs – an object class that contains information on the event.

# PSUEDOCODE:

START

INITIALIZE SupplyChainSystem as system

CALL system.Initialize()

CALL system.StartRealTimeUpdates()

WHILE TRUE

PRINT "Choose an action:"

PRINT "1. Simulate Scenario"

PRINT "2. Optimize Routes"

PRINT "3. Allocate Resources"

PRINT "4. Forecast Demand"

PRINT "5. Update Weather"

PRINT "6. View System Status"

PRINT "7. Exit"

READ choice

SWITCH choice

CASE "1":

CALL system.SimulateScenario()

CASE "2":

CALL system.OptimizeRoutes()

CASE "3":

CALL system.AllocateResources()

CASE "4":

CALL system.ForecastDemand()

CASE "5":

CALL system.UpdateWeather()

CASE "6":

CALL system.ViewSystemStatus()

CASE "7":

PRINT "Exiting..."

EXIT

DEFAULT:

PRINT "Invalid choice. Please try again."

END SWITCH

END WHILE

END

FUNCTION Initialize()

PRINT "Initializing supply chain system..."

// Initialize supply points

number\_of\_supply\_points = GET\_VALID\_INTEGER("Enter number of supply points:")

FOR i FROM 1 TO number\_of\_supply\_points

PRINT "Enter latitude and longitude for Supply Point i (comma separated):"

location = GET\_VALID\_LOCATION()

latitude = PARSE\_FLOAT(location[0])

longitude = PARSE\_FLOAT(location[1])

ADD new SupplyPoint(i, new Location(latitude, longitude)) TO supplyPoints

END FOR

// Initialize transportation units

number\_of\_transport\_units = GET\_VALID\_INTEGER("Enter number of transportation units:")

FOR i FROM 1 TO number\_of\_transport\_units

PRINT "Enter type, capacity, and location (latitude, longitude) for Transport Unit i (comma separated):"

input = GET\_VALID\_TRANSPORT\_INPUT()

type = input[0]

capacity = PARSE\_INT(input[1])

latitude = PARSE\_FLOAT(input[2])

longitude = PARSE\_FLOAT(input[3])

ADD new TransportUnit(i, type, capacity, new Location(latitude, longitude)) TO transportationUnits

END FOR

// Initialize routes

number\_of\_routes = GET\_VALID\_INTEGER("Enter number of routes:")

FOR i FROM 1 TO number\_of\_routes

WHILE TRUE

TRY

PRINT "Enter start point ID, end point ID, and distance for Route i (comma separated):"

input = READ\_LINE().SPLIT(',')

IF LENGTH(input) != 3 THEN

THROW FormatException("Invalid input format. Please enter three comma-separated values.")

END IF

start\_point\_id = PARSE\_INT(input[0])

end\_point\_id = PARSE\_INT(input[1])

distance = PARSE\_FLOAT(input[2])

CALL VALIDATE\_ROUTE\_INPUT(start\_point\_id, end\_point\_id, distance)

start\_point = FIND\_SUPPLY\_POINT\_BY\_ID(start\_point\_id)

end\_point = FIND\_SUPPLY\_POINT\_BY\_ID(end\_point\_id)

IF start\_point IS NULL OR end\_point IS NULL THEN

THROW ArgumentException("Invalid Supply Point ID.")

END IF

IF ROUTE\_EXISTS(start\_point\_id, end\_point\_id) THEN

THROW ArgumentException("Route already exists.")

END IF

ADD new Route(i, start\_point, end\_point, distance) TO routes

BREAK

CATCH FormatException

PRINT "Error: FORMAT\_EXCEPTION\_MESSAGE. Please try again."

CATCH ArgumentException

PRINT "Error: ARGUMENT\_EXCEPTION\_MESSAGE. Please try again."

CATCH Exception

PRINT "An unexpected error occurred: EXCEPTION\_MESSAGE. Please try again."

END WHILE

END FOR

// Initialize weather

CALL weatherConditions.UpdateForecast()

END FUNCTION

FUNCTION StartRealTimeUpdates()

CREATE weather\_thread TO CALL UpdateWeatherPeriodically()

CREATE status\_thread TO CALL UpdateStatusPeriodically()

START weather\_thread

START status\_thread

END FUNCTION

FUNCTION UpdateWeatherPeriodically()

WHILE TRUE

CALL UpdateWeather()

SLEEP 60000 // 1 minute

END WHILE

END FUNCTION

FUNCTION UpdateStatusPeriodically()

WHILE TRUE

CALL UpdateAllStatuses()

SLEEP 60000 // 1 minute

END WHILE

END FUNCTION

FUNCTION SimulateScenario()

PRINT "Simulating supply chain scenario..."

FOR EACH supply\_point IN supplyPoints

CALL supply\_point.UpdateStatus()

END FOR

FOR EACH transport\_unit IN transportationUnits

CALL transport\_unit.UpdateStatus()

END FOR

FOR EACH route IN routes

CALL route.UpdateStatus()

END FOR

CALL weatherConditions.SimulateImpactOnRoutes()

END FUNCTION

FUNCTION OptimizeRoutes()

PRINT "Optimizing supply routes..."

FOR EACH route IN routes

CALL route.CalculateShortestPath()

CALL route.EvaluateRisk()

END FOR

END FUNCTION

FUNCTION AllocateResources()

PRINT "Allocating resources across supply points..."

FOR EACH supply\_point IN supplyPoints

CALL supply\_point.CalculateDemand()

CALL supply\_point.AllocateResources()

END FOR

END FUNCTION

FUNCTION ForecastDemand()

PRINT "Forecasting future demand..."

FOR EACH supply\_point IN supplyPoints

PRINT "Supply Point supply\_point.Id:"

CALL supply\_point.Demand.Forecast()

CALL supply\_point.Demand.AdjustForScenario()

END FOR

END FUNCTION

FUNCTION UpdateWeather()

PRINT "Updating weather conditions..."

CALL weatherConditions.UpdateForecast()

CALL weatherConditions.SimulateImpactOnRoutes()

IF weatherConditions.Severity = "Severe" THEN

CALL OnCriticalSituationOccurred("Severe weather conditions detected. Routes may be affected.")

END IF

END FUNCTION

FUNCTION ViewSystemStatus()

PRINT "System Status:"

PRINT "---------------"

PRINT "Supply Points:"

FOR EACH sp IN supplyPoints

PRINT "ID: sp.Id, Location: (sp.Location.Latitude, sp.Location.Longitude)"

inventory = CALL sp.Inventory.CheckStockLevels()

PRINT "Inventory: Fuel: inventory["Fuel"], Ammo: inventory["Ammunition"], Food: inventory["FoodSupplies"], Medical: inventory["MedicalSupplies"]"

demand = CALL sp.Demand.GetDemandLevels()

PRINT "Demand: Fuel: demand["Fuel"], Ammo: demand["Ammunition"], Food: demand["FoodSupplies"], Medical: demand["MedicalSupplies"]"

END FOR

PRINT "Transportation Units:"

FOR EACH tu IN transportationUnits

PRINT "ID: tu.Id, Type: tu.Type, Capacity: tu.Capacity, Status: tu.Status"

PRINT "Location: (tu.Location.Latitude, tu.Location.Longitude)"

END FOR

PRINT "Routes:"

FOR EACH route IN routes

PRINT "ID: route.Id, Start: route.StartPoint.Id, End: route.EndPoint.Id, Distance: route.Distance, Risk: route.RiskLevel"

END FOR

PRINT "Weather: weatherConditions.Type, Severity: weatherConditions.Severity"

PRINT "Forecast: weatherConditions.Forecast"

END FUNCTION

FUNCTION GET\_VALID\_INTEGER(prompt)

WHILE TRUE

PRINT prompt

READ input

IF TRY\_PARSE\_INT(input) IS VALID THEN

RETURN PARSE\_INT(input)

ELSE

PRINT "Invalid input. Please enter a positive integer."

END IF

END WHILE

END FUNCTION

FUNCTION GET\_VALID\_LOCATION()

WHILE TRUE

location = READ\_LINE().SPLIT(',')

IF LENGTH(location) = 2 AND TRY\_PARSE\_FLOAT(location[0]) AND TRY\_PARSE\_FLOAT(location[1]) THEN

RETURN location

ELSE

PRINT "Invalid location format. Please enter latitude and longitude separated by a comma."

END IF

END WHILE

END FUNCTION

FUNCTION GET\_VALID\_TRANSPORT\_INPUT()

WHILE TRUE

input = READ\_LINE().SPLIT(',')

IF LENGTH(input) = 4 AND NOT EMPTY(input[0]) AND TRY\_PARSE\_INT(input[1]) AND TRY\_PARSE\_FLOAT(input[2]) AND TRY\_PARSE\_FLOAT(input[3]) THEN

RETURN input

ELSE

PRINT "Invalid input format. Please enter type, capacity, latitude, and longitude separated by commas."

END IF

END WHILE

END FUNCTION

FUNCTION VALIDATE\_ROUTE\_INPUT(start\_point\_id, end\_point\_id, distance)

IF distance <= 0 THEN

THROW ArgumentException("Distance must be greater than zero.")

END IF

END FUNCTION

FUNCTION UpdateAllStatuses()

FOR EACH updatable IN supplyPoints + transportationUnits + routes

CALL updatable.UpdateStatus()

END FOR

END FUNCTION

FUNCTION OnCriticalSituationOccurred(message)

CALL CriticalSituationOccurred(message)

END FUNCTION

END

## Class Diagram: A black screen with white text Description automatically generated

# References

Anon., n.d. *W3Schools.* [Online]   
Available at: https://www.w3schools.com/cs/cs\_enums.php

geeksforgeeks, 2020. *C# Interface.* [Online]   
Available at: https://www.geeksforgeeks.org/c-sharp-interface/?ref=gcse\_ind

geeksforgeeks, 2021. *C# Delegates.* [Online]   
Available at: https://www.geeksforgeeks.org/c-sharp-delegates/?ref=gcse\_ind

geeksforgeeks, 2021. *C# Methods.* [Online]   
Available at: https://www.geeksforgeeks.org/c-sharp-methods/?ref=gcse\_ind

geeksforgeeks, 2022. *C# Class and Oject.* [Online]   
Available at: https://www.geeksforgeeks.org/c-sharp-class-and-object/?ref=gcse\_ind

geeksforgeeks, 2023. *C# Abstraction.* [Online]   
Available at: https://www.geeksforgeeks.org/c-sharp-abstraction/?ref=gcse\_ind

geeksforgeeks, 2024. *Enumeration (or enum).* [Online]   
Available at: https://www.geeksforgeeks.org/c-sharp-enumeration-or-enum/?ref=gcse\_ind

geeksforgeeks, n.d. *C# Inheritance.* [Online]   
Available at: https://www.w3schools.com/cs/cs\_inheritance.php

Programiz, n.d. *C#Polymorphism.* [Online]   
Available at: https://www.programiz.com/csharp-programming/polymorphism

W3School, n.d. *C# OOP.* [Online]   
Available at: https://www.w3schools.com/cs/cs\_oop.php

W3Schools, 2023. *C# Inheritance.* [Online]   
Available at: https://www.geeksforgeeks.org/c-sharp-inheritance/?ref=gcse\_ind

W3Schools, n.d. *C# Abstraction.* [Online]   
Available at: https://www.w3schools.com/cs/cs\_abstract.php

W3Schools, n.d. *C# Default Parameter Value.* [Online]   
Available at: https://www.w3schools.com/cs/cs\_method\_parameters\_default.php

W3Schools, n.d. *C# Interface.* [Online]   
Available at: https://www.w3schools.com/cs/cs\_interface.php

W3Schools, n.d. *C# Methods.* [Online]   
Available at: https://www.w3schools.com/cs/cs\_methods.php

W3Schools, n.d. *C# OOP.* [Online]   
Available at: https://www.w3schools.com/cs/cs\_oop.php

W3Schools, n.d. *C# Polymorphism.* [Online]   
Available at: https://www.w3schools.com/cs/cs\_polymorphism.php

W3Schools, n.d. *Parameters and Arguments.* [Online]   
Available at: https://www.w3schools.com/cs/cs\_method\_parameters.php