

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI[®]

**Comparison Between C++ and Java --- A Case Study Using a
Networked Automated Gas Station Simulation System**

Ying Chun Liu

A Major Report
In
The Department
of
Computer Science

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Computer Science at
Concordia University
Montreal, Quebec, Canada

March 2002

© Ying Chun Liu, 2002



National Library
of Canada

Acquisitions and
Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale
du Canada

Acquisitions et
services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-68461-X

Canada

Abstract

Comparison Between C++ and Java --- A Case Study Using a Networked Automated Gas Station Simulation System

This project aims at the comparison between the features and qualities of the Java and C++ programming languages. Even though they are both object-oriented programming languages, there exists a surprising number of differences between C++ and Java. These differences are intended to be significant improvements in favor of Java. It is by investigating on these differences that we will see why Java and C++ are both such beneficial programming language on different aspects. This project will take us through the important features that distinguish Java from C++.

We proceed by first describing what are the key concepts of object-oriented programming, then we present a case study that consists in the building of a software system for the simulation of a gas station network system. The design of the system is shown, then the system is built in two different versions using both Java and C++. Then the differences between the two languages are shown in light of this experience.

Acknowledgements

Sincerely, I would like to express my deepest respect and gratitude to my major report supervisor Dr. Joey Paquet for his guidance, invaluable suggestions, encouragement and support throughout the course of this work. I am very thankful for the opportunity I had, to work with him, and for everything he taught me during my master's studies.

I wish to thank my colleagues at the graduate laboratory of the Computer Science Department of Concordia University, especially Sherry Wang, ChunLei Ren and Bing Chun Zhang for the fruitful discussions, suggestions and comments on the project and software tools.

I would like to give a special thanks to my wife, my lovely son and my parents. It was their love, patience and continued support that made this major report possible.

Table of Contents

<u>ABSTRACT</u>	III
<u>ACKNOWLEDGEMENTS</u>	IV
<u>1 INTRODUCTION</u>	1
1.1 <u>KEY CONCEPTS OF OBJECT-ORIENTED PROGRAMMING</u>	1
1.2 <u>LANGUAGE COMPARISON CRITERIA USED</u>	2
<u>2 IMPLEMENTATION DESCRIPTION</u>	3
2.1 <u>GENERAL DESCRIPTION</u>	3
2.2 <u>SYSTEM DESCRIPTION</u>	3
2.2.1 <u>Use Case View</u>	4
2.2.2 <u>Conceptual Architecture View</u>	7
2.2.3 <u>Topology View</u>	8
2.2.4 <u>System Architecture View</u>	12
<u>3 IMPLEMENTATION DETAILS IN BOTH LANGUAGES</u>	14
3.1 <u>JAVA IMPLEMENTATION DETAILS</u>	14
3.1.1 <u>Java Implementation Structure</u>	14
3.1.2 <u>Java User Interface Description</u>	17
3.1.3 <u>Use of Packages and Libraries in the Java Implementation</u>	18
3.2 <u>C++ IMPLEMENTATION DETAILS</u>	19
3.2.1 <u>C++ Implementation Structure Description</u>	19
3.2.2 <u>C++ User Interface Description</u>	22
3.2.3 <u>Use of Packages and Libraries in the C++ Implementation</u>	23
3.3 <u>DIFFERENCES IN IMPLEMENTATION TECHNIQUES</u>	24
3.3.1 <u>System Structure Design</u>	24
3.3.2 <u>User Interface Design</u>	24

3.3.3	<u>Database Design</u>	25
3.3.4	<u>Network Communication Design</u>	27
4	<u>COMPARISON OF JAVA AND C++ LANGUAGES</u>	28
4.1	<u>PLATFORM INDEPENDENCE</u>	28
4.2	<u>EFFICIENCY</u>	31
4.3	<u>PORTABILITY</u>	32
4.4	<u>ROBUSTNESS</u>	35
4.5	<u>NETWORK READINESS</u>	37
4.6	<u>MULTITHREADING</u>	38
4.7	<u>INSTANT UPDATES OVER THE INTERNET</u>	41
4.8	<u>SECURITY</u>	43
4.9	<u>DYNAMIC CLASS INTEGRATION</u>	45
4.10	<u>ABSTRACTION</u>	48
4.11	<u>INHERITANCE</u>	51
4.12	<u>POLYMORPHISM</u>	53
4.13	<u>GARBAGE COLLECTION</u>	55
5	<u>CONCLUSION</u>	57
	<u>REFERENCES</u>	63
	<u>APPENDIX OPERATION INSTRUCTIONS</u>	65

Table of Figures

<u>FIGURE 1: USE CASE DIAGRAM FOR THE GAS STATION SYSTEM</u>	4
<u>FIGURE 2: GAS-STATION NETWORK CONCEPTUAL ARCHITECTURE</u>	7
<u>FIGURE 3 TOPOLOGY OF THE GAS STATION NETWORK APPLICATION</u>	8
<u>FIGURE 4: SYSTEM ARCHITECTURE DESCRIPTION DIAGRAM</u>	12
<u>FIGURE 5: CLIENT SIDE IMPLEMENTATION STRUCTURE DIAGRAM IN JAVA</u>	15
<u>FIGURE 6: SERVER SIDE IMPLEMENTATION STRUCTURE DIAGRAM IN JAVA</u>	16
<u>FIGURE 7: USER INTERFACE OF GAS STATION CLIENT IN JAVA</u>	17
<u>FIGURE 8: USER INTERFACE OF GAS STATION SERVER IN JAVA</u>	18
<u>FIGURE 9: CLIENT SIDE IMPLEMENTATION STRUCTURE DESIGN DIAGRAM IN C++</u>	20
<u>FIGURE 10: SERVER SIDE IMPLEMENTATION STRUCTURE DIAGRAM IN C++</u>	21
<u>FIGURE 11: USER INTERFACE OF GAS STATION CLIENT IN C++</u>	22
<u>FIGURE 12 USE INTERFACE OF GAS STATION SERVER IN C++</u>	23
<u>FIGURE 13: JAVA VELCRO EFFECT</u>	32
<u>FIGURE 14: JVM ARCHITECTURE</u>	33
<u>FIGURE 15: SERVER USER INTERFACE OF PROJECT WITH JAVA LANGUAGE</u>	65
<u>FIGURE 16: PERSONAL ACCOUNT REPORT</u>	66
<u>FIGURE 17: SYSTEM NEED A VALID SERVER NAME</u>	66
<u>FIGURE 18: MONITOR PANEL OF A GAS STATION</u>	67
<u>FIGURE 19: SET-UP DIALOG BOX</u>	68
<u>FIGURE 20: GAS PUMP PANEL</u>	68

1 Introduction

This project aims at the comparison between the features and qualities of the Java and C++ programming languages. Even though they are both object-oriented programming languages, there exists a surprising number of differences between C++ and Java. These differences are intended to be significant improvements, and if we understand the differences we'll see why Java and C++ are both such the beneficial programming language on different aspects. This project takes us through the important features that distinguish Java from C++.

Before analyzing differences between C++ and Java, we will start by describing what are the key concepts of object-oriented programming. This will lead the way to a presentation of the criteria on which the two programming languages will be evaluated.

1.1 Key Concepts of Object-Oriented Programming

Object: means the data is quantities into discrete, distinguishable entities. Objects can be concrete, such as a file in a file system, or conceptual, such as a scheduling policy in a multiprocessing operating system.

Class: means that the objects with same data structure and behavior are grouped into a class. A class is an abstraction that describes properties important to an application and ignores the rest. Each object is said to be an instance of its class.

Inheritance: is the sharing of attributes and operations among classes based on a hierarchical relationship. A class can be defined broadly and then redefined into successively finer subclass.

Polymorphism: means that the same operation may behave differently on different classes. In object-oriented methodology, the language can automatically select the correct method to implement an operation based on the name of the operation and the class of the object operated on.

Object-Oriented Methodology: the essence of object-oriented development is the identification and organization of application domain concepts, rather than their final representation in a programming language. Object is the core part of software development to manipulate the inherent complexity of the problems.

1.2 Language Comparison Criteria Used

Comparison on Implementation Techniques

Compare on aspects: System Structure, User Interface, Database Design and Network communication design.

Comparison on Language Feature

Compare on aspects: Platform independence, Efficiency, Portability, Network readiness, Security, Inheritance, Robustness and etc. features. Refer to Section 4.

Comparison on Language Characteristics

Compare on aspects: Data structure, Data Type, Algorithm and Language Design Strategy.

2 Implementation Description

2.1 General Description

Inspired by the gas stations spreading in Montreal, we used the concept of an automated gas station network system as a case study. We do not pretend to strictly follow any software development lifecycle for the development of these two programs, as the problem at hand is quite simple, and the goal of the exercise is to compare the implementation aspects of the two programs.

The system provides the functions of gas station network management and customers behaviors. It simulates the most of gas station network scenarios on both customer and gas stations. The software structure is based on the three layer topology, which are user interface, domain application, and database layer.

The purpose of this case study is to refer to as many aspects of Java as possible through the modeling and implementation of a real-life gas station network. We have developed the Java code using JDK 1.2. The Java features that we have applied into this project include: Java Swing, multithreading, JDBC and Java sockets. Polymorphism, abstract class(interfaces), and inner class are used as the design techniques for JAVA version development.

The C++ implementation has been developed using Borland C++ 3.0. The C++ features we have applied includes: Multithreading, ODBC and C++ sockets (WinSocket), and techniques such as overloading, polymorphism, and virtual functions.

2.2 System Description

Here we describe the system from different viewpoints: a very high level use-case view, a conceptual architectural view, a topology view, and a detailed architectural view. The aggregation of these views aims at the incremental description of the system from

abstract concepts to a physical implementation model. That leads the way to the next section: implementation.

2.2.1 Use Case View

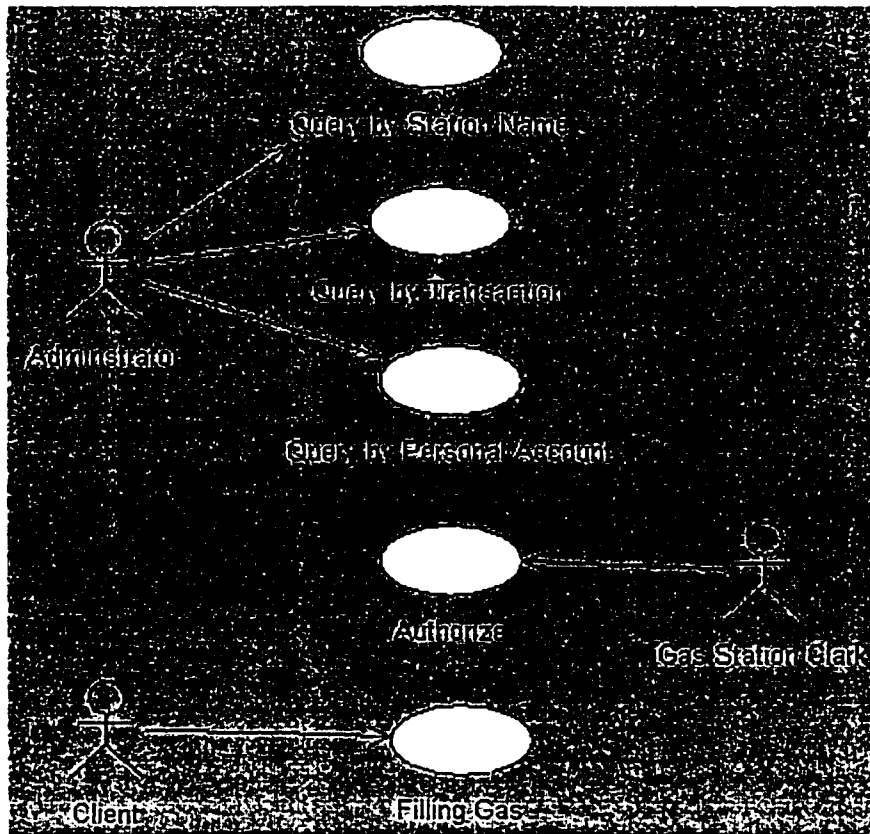


Figure 1: Use Case Diagram for the Gas Station System

Figure 1 describes the system from a very high level use case point of view. It is applicable for the implementation in both Java and C++ .

Actors

Administrator : An authorized user that uses the server application program to administer the application and query the database to manage the gas station network.

Customer : A client that comes to a gas station and performs a gas filling operation using a credit card.

Gas station clerk : Clerk mainly responsible for the activation of the gas pumps after a customer has been granted permission from the server to perform a gas filling operation.

Use Cases

Connect to server : Input the correct server name, client will establish the connection with the server. User can input the server name both on command line and after the system prompt.

Change server : If the server crashes, client can change to connect with another valid server.

Update gas price and gas tank volume : Each gas station can set its own gas price, also it can change the volume of its gas pump (the purpose of changing gas pump volume is for testing the relation between gas level of gas pump and monitored value)

Gas level low alarm : When the gas level of any gas pump is lower than a preset value (5%), the monitor of that gas station will issue an alarm.

Gas tank re-charge : Monitor can re-charge the gas tank when its level is low.

Filling-up auto-stop when gas level low : If the gas level of a tank is lower than a preset limit and a filling-up session is in progress, The system will stop the session automatically.

Authorization : Using a gas pump must get the permission from the monitor of the gas station.

Client card number input : Clients input their card number with the keyboard on the pump.

Client card number authentication : System will check if the card is valid or not.

Client card protection : If a card is being used by someone, it will be invalid for anyone else at the same time.

Balance display : Gas pump will display the balance before and after the filling-up session.

Filling-up pause and resume : Client can pause filling and resume.

Filling-up terminate : At any time, the client can finish the filling-up session.

Send data about the filling-up session to server : Send new balance of the client to the server after gas filling-up session.

Validate client card : Check the card number and send validation information back to the gas station.

Check the balance : Query the balance according to the card number and send balance back to the gas station.

Update balance : Accept the new balance from the gas station and update the database.

Record filling-up session : Record the data of each filling-up session, which includes station name, gas price, gas volume, and card number.

Display transactions by gas station : Given a station number, display all the transactions that happened in this station.

Display transaction by transaction number : Given a transaction number, display all the information about a specific transaction.

Display the client information : Display the information about a client, given a specific account number.

2.2.2 Conceptual Architecture View

The physical structure of this gas station network is shown in Figure 2. The system description is applied to both programming languages C++ and Java. In Figure 2, the *Server* is assumed to be the *Administration Station* of this gas station network. The server application software and management-related database are installed on it. The *Backup Server* is a spare server. If the main server is not available, all gas stations may change to communicate with the backup server. *Gas Station 1..N* are virtual gas stations. General clients can perform a gas-filling-up session through the user interface on these gas stations.

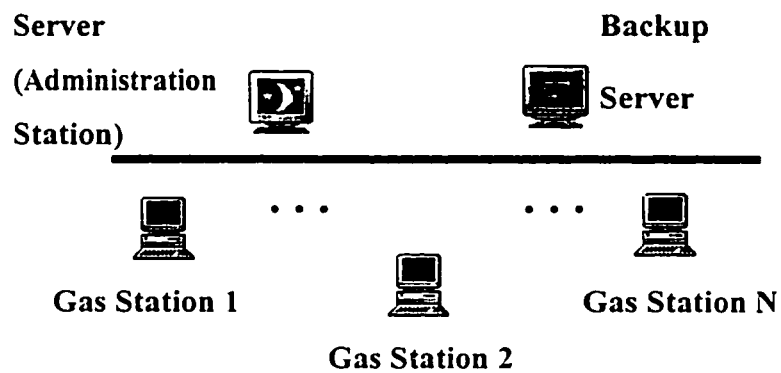


Figure 2: Gas-Station Network Conceptual Architecture

2.2.3 Topology View

The topology of the gas station network system using both Java and C++ is shown in Figure 3. Note that the topology for the Java implementation includes the *Java Virtual Machine* layer. For the C++ implementation, we do not have this layer: the *Application* layer is built directly on the *Operating System* layer. The topology of the system consists of two main parts: the client side and the server side.

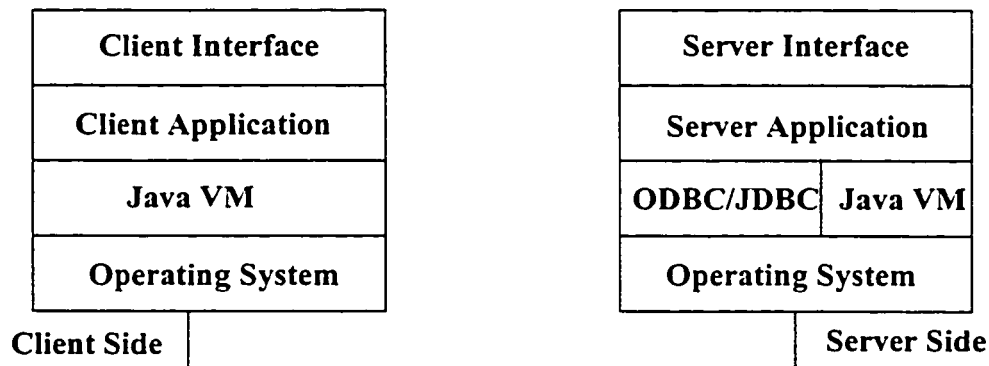


Figure 3 Topology of the Gas Station Network Application

Client Side Topology

The client side is made up of *Client Interface*, *Client Application*, *Java Virtual Machine*, and *Operating System* in Java design. The C++ implementation is made up of *Client Interface*, *Client Application*, and *Operating System* instead. The *Client side* is connected by the *Networking System* to the *Server side*.

Client Interface : Through the Client Interface the client can input the server name both on command line or use the system prompt to connect with the server. Also the user can fill the gas and monitor the gas filling process from the client interface.

Client Application : The client application programs provides functions to enable communication with the server such as setting up network communication, exchanging

information with the server, monitoring the use of the system by the customer, credit card transaction processing, and gas level balance controlling. If a connection failure is occurring with the server application, the customer can change to connect with another valid server. The gas price can be changed independently in each gas station, because different areas can have different gas prices. The gas station can change the volume of its gas pump. The gas station monitor can automatically control the gas level of any pump. If it goes lower than a preset value, the monitor will give an alarm signal to the system. The system will take actions to recharge the tank. Any customer must get permission to access the pump to get gas with control by the monitor. The client application can also verify the customer's credit to decide whether to activate the pump or not. If the card provided by the client is not valid the pump is stopped. The interface will display the balance before and after the filling-up session and after this session the program will send the new balance to the server.

Java Virtual Machine : The Java Virtual machine is very important for any Java program, development of the Java program and runtime environment, to describe software that acts as an interface between compiled Java binary code and the platform that actually performs the program's instructions. Once a Java virtual machine has been provided and installed on a platform, any Java program (which, after compilation, consists of *bytecodes*) can run on that platform. Java was designed to allow application programs to be built that could run on any platform without having to be rewritten or recompiled by the programmer for each separate platform. It is the *Java Virtual Machine* that makes this possible. The Java Virtual Machine defines an abstract rather than a real "machine" (or processor) and specifies an instruction set, a set of registers, a stack, a heap that is garbage collected and a method area.

Client Operating System : The Operating System is there to control and coordinate the use of the hardware among the various applications programs for the various users. The client side operating system for the Client side is Windows 98.

Server Side Topology

The Server side is made up of *Server Interface*, *Server Application*, *Java Virtual Machine*, *ODBC/JDBC* database adapters, and *Operating System*. The server side is connected by the *Networking System* to the *Client side*.

Server Interface : The Server Interface that provides services to up to 100 gas stations and provides a software interface to the client side to interact with the server side. The user can query the information on the gas station by selecting the query options and browsing the query results through the server interface.

Server Application : The server application programs first receives the card ID sent by the client side and checks whether the card is valid or not, and then sends confirmation back to the gas station. When the server application receives the query that requests the balance according to the card ID, the server sends back the credit balance to the client side. After the transaction is finished, the server application updates the balance data in the database. The server application also records the data of each filling up session done by a client. The stored data are station name, gas price, gas volume and card number. The server application provides a functionality that can display all transactions in every station. It also can display the client credit card number and balance.

ODBC/JDBC and Java Virtual Machine : We need the JDBC database adaptor in our project to establish a connection between the server application and the underlying database. This adaptor actually translates our Java database manipulation statements into SQL statements, and submits them to the database management system. The database we chose is a relational database based on SQL (Structured Query Language) for queries. The purpose of using the Java Database Connectivity (JDBC) is to work as a bridge together with ODBC to control the Microsoft Access Database.

Server Operating System : The Operating System is to control and coordinate the use of the hardware among the various applications programs for the various users. The server side operating system for GAS Station is Windows 98.

Network Communication

For the network communication we use the client/server interaction with socket connections. When the gas system starts to work, the server waits for a client connection attempt. After a client application sends a connection request to the server, the server application sends a message indicating that the connection was successful to the client. Then the client application displays this message. When the client or server sends a termination message, the connection between client and server is terminated. Then the server waits for the next client to send a connection request. When the server socket is set up, the application will listen for a connection request from the client on port 8080. After that, a number of connections can wait in a queue to connect to the server (that was set to 100 in our implementation). If the queue is full when a connection is requested, the connection is refused and an appropriate message is sent to the requesting client.

2.2.4 System Architecture View

The following figure describes the system from the package dependency point of view. It is applicable for both Java and C++ implementation.

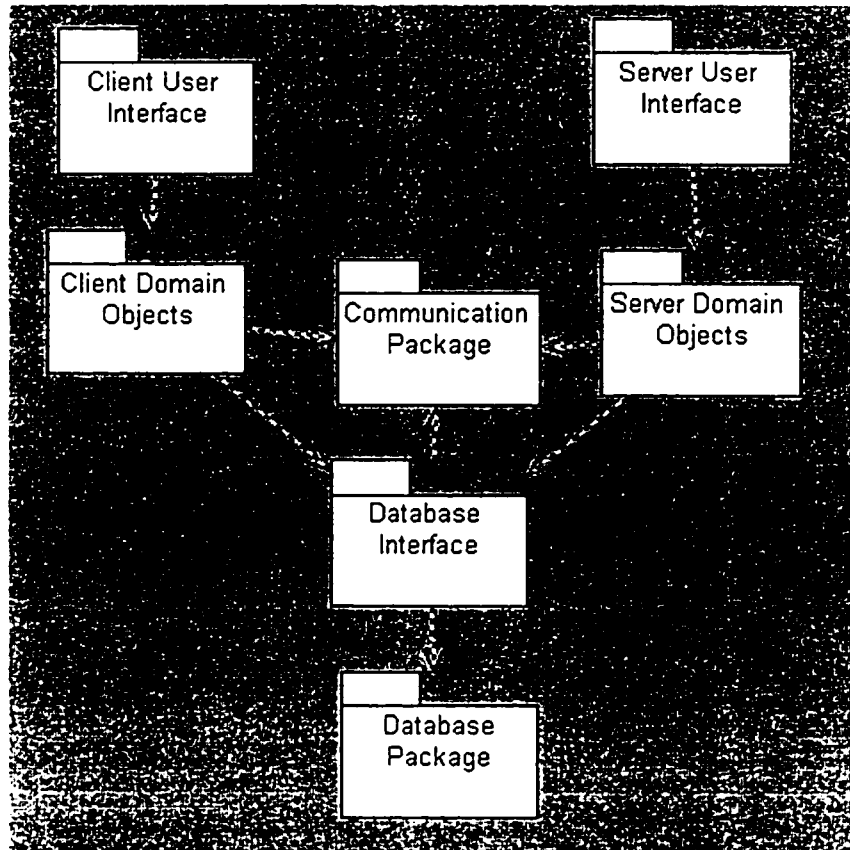


Figure 4: System Architecture Description Diagram

The client side is connected by the networking system to the server side. The server is in state to be connected by the client when it starts. Through the *Client User Interface* package, the user can input information and control the gas station. The *Client Domain Objects* package provides the gas station specific functioning, like filling session simulation and etc. It is also for displaying all information for the client before and after the filling up session. The *Communication* package sets up network communication, which transmits valuable information between server and client. It is also responsible for

checking process and controlling gas level balance. Both C++ and Java version use the socket-based communications, which enable application view networking system as if it were file I/O. The *Database Interface* package provides query information help and SQL language implementation. The interface sets read and write access to the database and query control. The *Server User Interface* package allows the user to select to query on the station, account and transaction. The *Server Domain Objects* package will mainly display the query result to the user. The application program provides the function that can display information for transactions, account and station once the query is submitted. The *Database* package is based on the Access relational database model, which includes the transaction table, personal account table, and station table.

3 Implementation Details in both Languages

This section describes the implementation of the gas station network system, as implemented in Java, and then in C++.

3.1 Java Implementation Details

3.1.1 Java Implementation Structure

The Java version of the system is described in this section. The client side and the server side are depicted separately.

Client side User interface, domain package, and communication package are composed of the classes as depicted in **Error! Reference source not found..** Server side User interface, domain package, communication, and database package are composed of the classes as depicted in Figure 6. Direction arrow represents the dependency among classes; this mean one class can provide service to other class and use other class service as well. The detailed function prototypes are list to show the implementation methods on both Client side and Server side. The functionality will not be detailed here.

Client Side Implementation Structure in Java

See Figure 5 for the Software Structure Description Diagram of Client side in Java design.

See Figure 6 for the Software structure Description Diagram of Server side in Java Design.

.

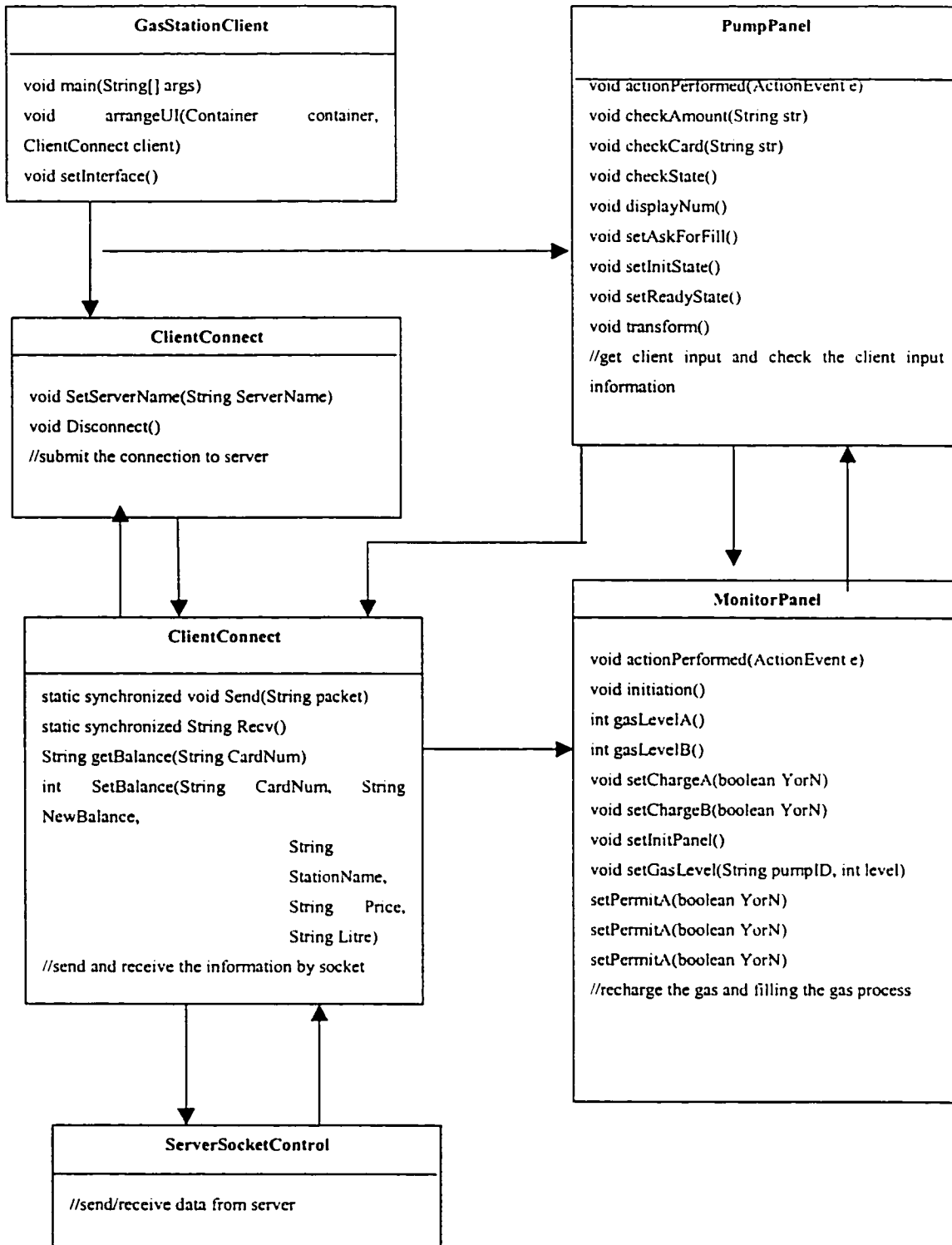


Figure 5: Client Side Implementation Structure Diagram in Java

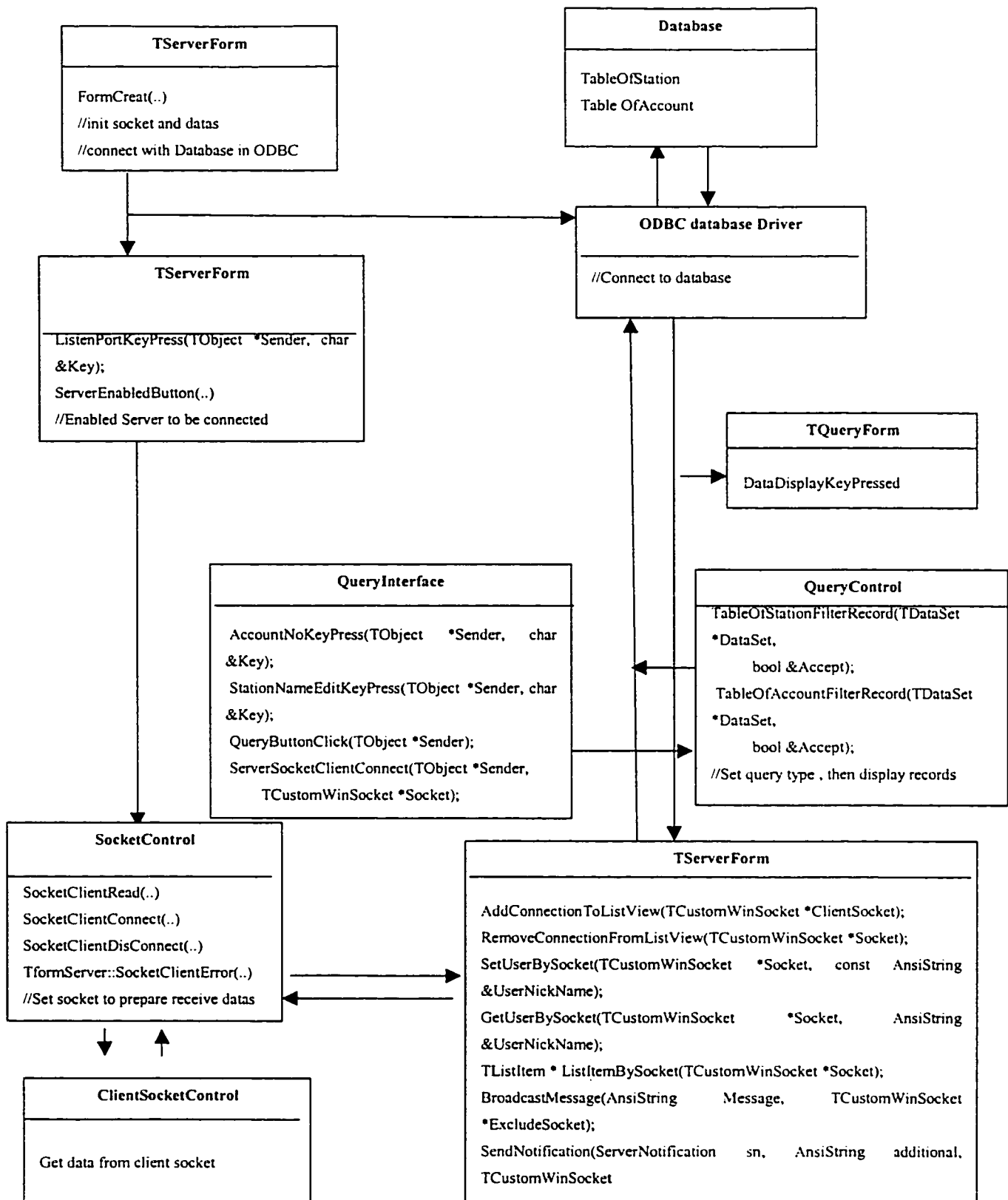


Figure 6: Server Side Implementation Structure Diagram in Java

3.1.2 Java User Interface Description

Client Side User Interface in Java

The gas station client user interface is composed of a monitor and two gas pumps input panels (Pump A and Pump B). The monitor is responsible for giving permission to a filling-up operation, re-charge the gas tanks (we assume that each gas pump equipped with one gas tank), and displaying the amount of gas that has been filled. The two pumps will allow the user to input the card number, display the gas price, and input the amount of gas to be filled.

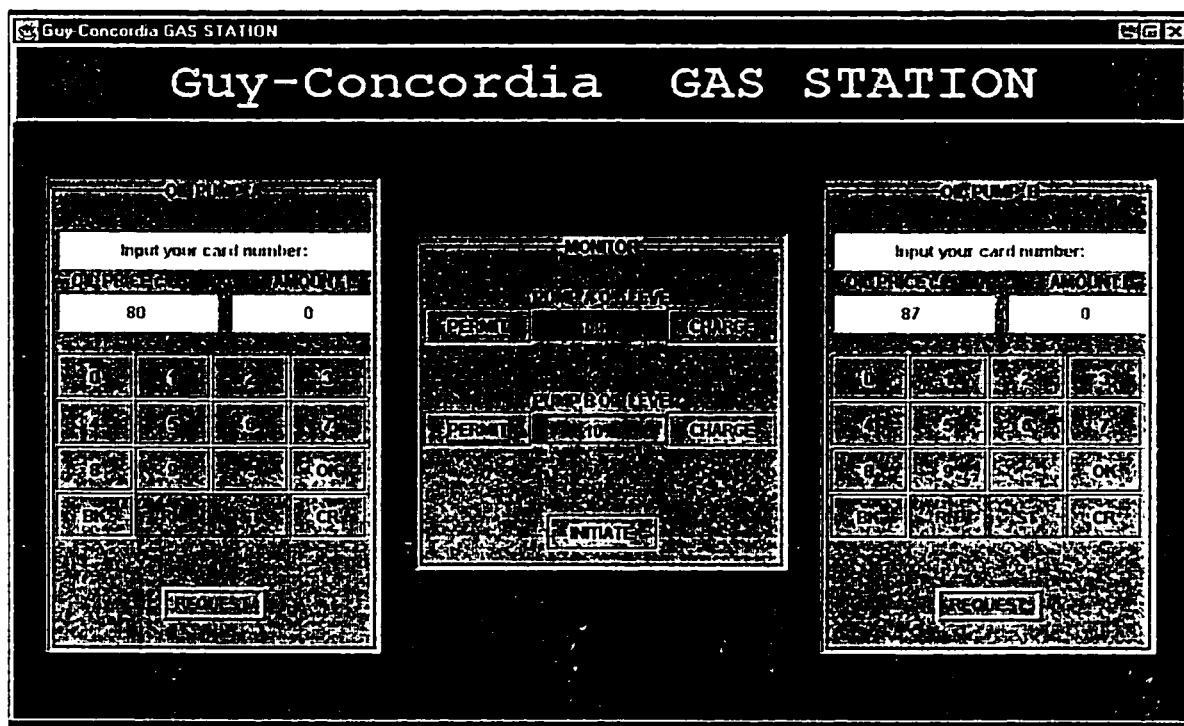


Figure 7: User Interface of Gas Station Client in Java

Server Side User Interface in Java

The gas station server user interface is a menu for the different queries on the database. Query on station name will display all the transactions that happened on this station. Query on a specific transaction will display all the information about this transaction, for

example, the amount of gas being filled, the total money, in which station, etc. Query on personal account will display the account balance for all account numbers.

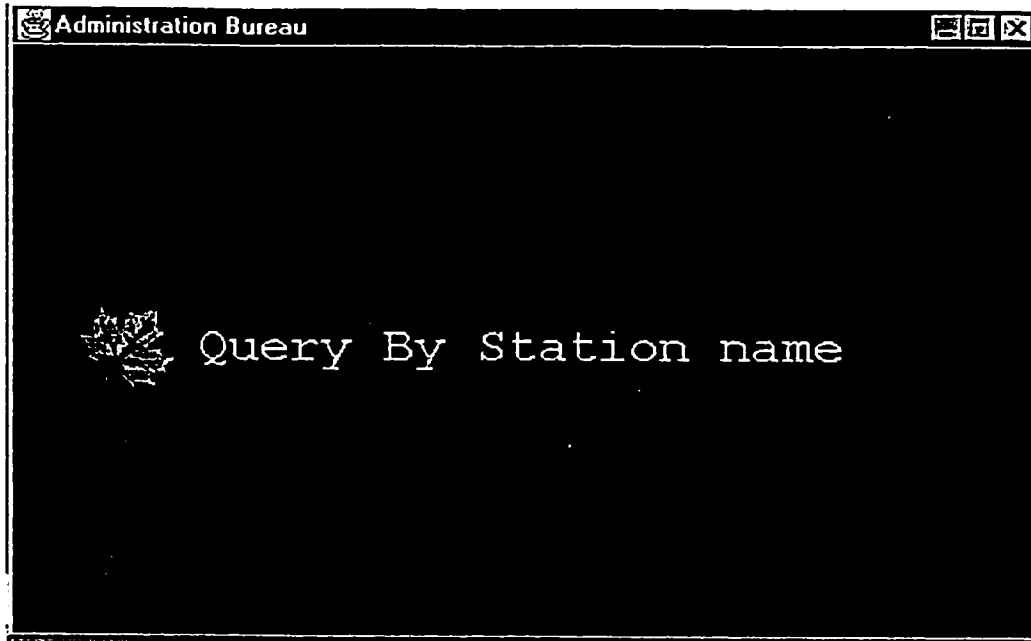


Figure 8: User Interface of Gas Station Server in Java

3.1.3 Use of Packages and Libraries in the Java Implementation

This part describes the package and libraries used in Java implementation.

Client User Interface

```
javax.swing.*  
java.awt.*  
java.awt.event.*  
java.lang.*
```

Network Communication

```
java.net.*  
java.io.*  
java.lang.Thread.*
```

Server User Interface

```
javax.swing.*  
javax.swing.border.*  
javax.swing.event.*  
java.awt.*  
java.awt.event.*  
java.lang.*
```

Server Database

```
java.awt.*;  
java.awt.event.*;  
javax.swing.*;  
javax.swing.event.*  
java.util.*  
java.sql.*
```

3.2 C++ Implementation Details

The C++ version of the system is described in this section. The client side and the server side are depicted separately.

3.2.1 C++ Implementation Structure Description

The C++ version of the system is described in this section. The client side and the server side are depicted separately.

Client side User interface, domain package, and communication package are composed of the classes as depicted in Figure 9 Server side User interface, domain package, communication, and database package are composed of the classes as depicted in Figure 10. Direction arrow represents the dependency among classes; this mean one class can provide service to other class and use other class service as well. The detailed function prototypes are list to show the implementation methods on both Client side and Server side. The functionality will not be detailed here.

Client Side Implementation Structure in C++

See Figure 9 for the Software Structure Description Diagram of Client side in C++ design.

See Figure 10 for the Software Structure Description Diagram of Server side in C++ design.

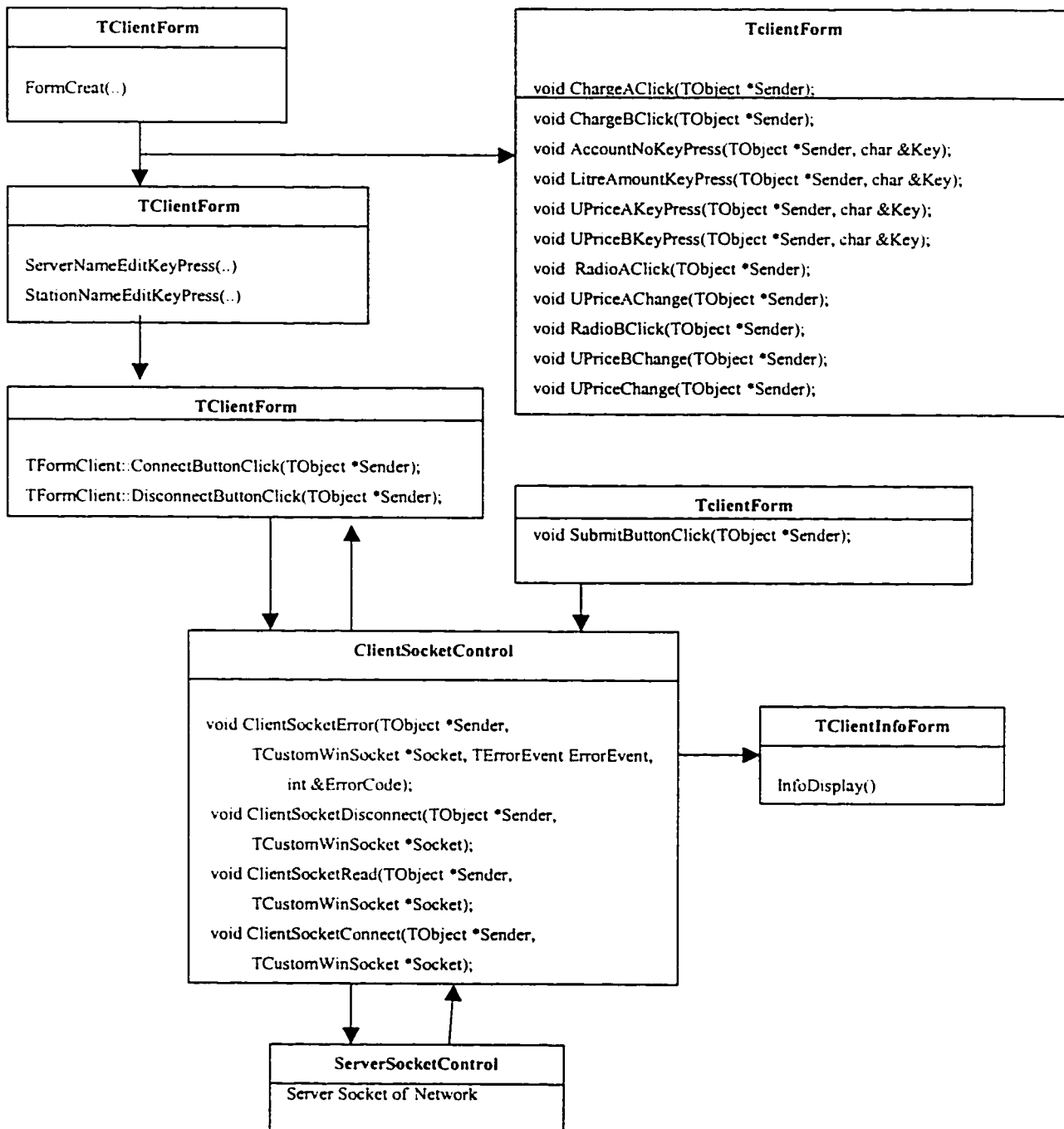


Figure 9: Client Side Implementation Structure Design Diagram in C++

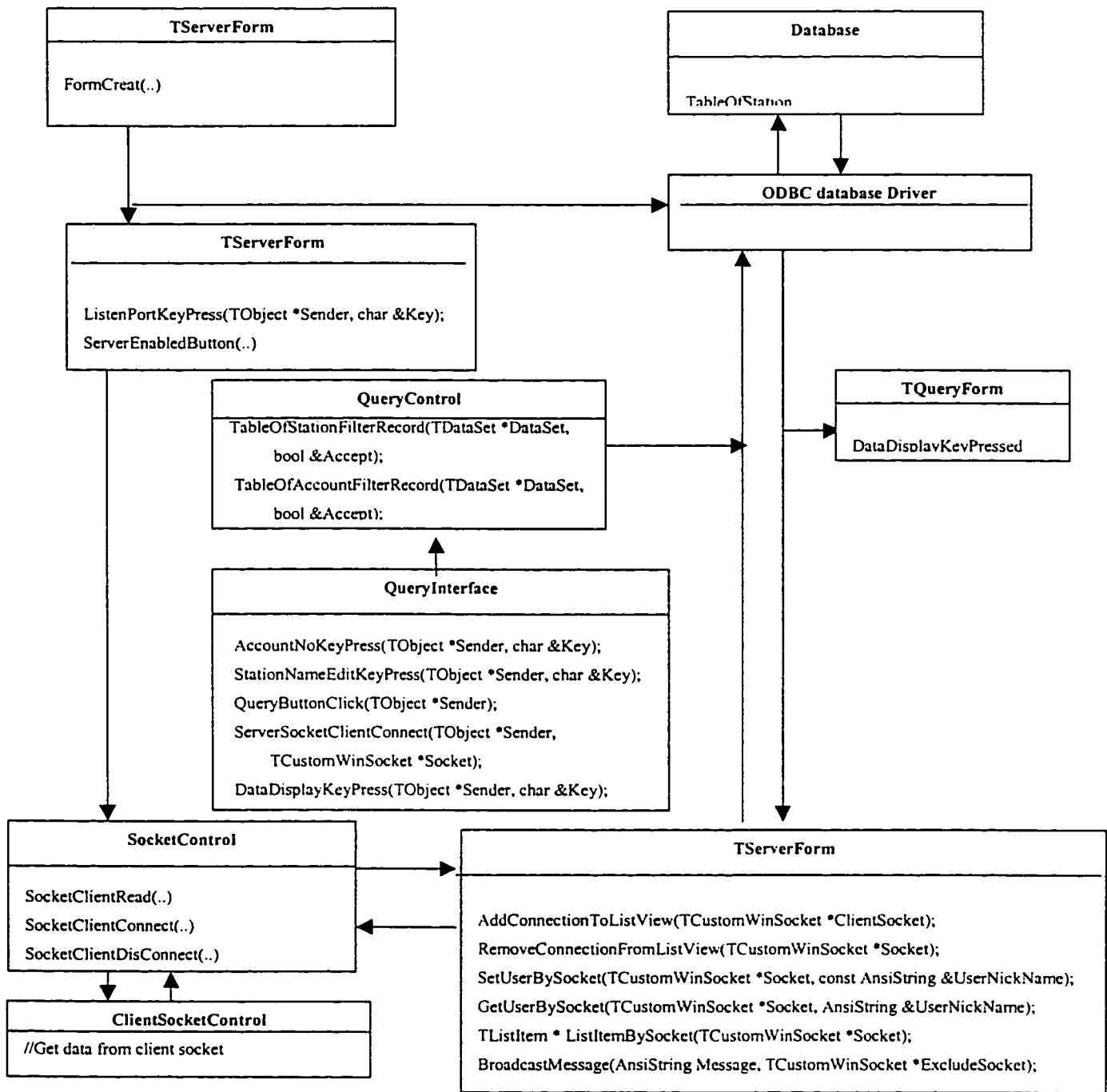


Figure 10: Server Side Implementation Structure Diagram in C++

3.2.2 C++ User Interface Description

Client Side User Interface in C++

The gas station client user interface in C++ is composed of a monitor for pump A and pump B and one user information input panel. The monitor is responsible for giving permission to a gas filling operation, filling-up the gas tanks (we assume that each gas pump equipped with one gas tank), display the gas level that has been filled by a customer, and display the total value of a filling-up operation. The user information input panel will allow the user to input the card number, unit gas price and selection for different pumps.

The screenshot shows a window titled "Gas Station" with a header "Gas Station Management". The interface is divided into several sections:

- Welcome Panel:** A box containing the text "Welcome to use GAS Management!!".
- User Input Panel:** Fields for "Account Number", "Litres Amount (L)" (set to 100), "Unit Price (c/L)" (set to 99.00), and "Sum (\$)" (set to 99). A "Submit" button is at the bottom.
- Gas Selection:** Radio buttons for "Gas A" (selected) and "Gas B".
- Connection Panel:** Fields for "Station Name" (Station001), "Server Name" (localhost), and "Link port" (5790). The "Link Status" is "Disconnected". There are "Connect" and "Disconnect" buttons.
- Pump Monitors:** Two panels for "Oil A" and "Oil B". Each panel shows "Unit Price (c/L)" (99.00), "Value" (20000), a progress bar, and a "Charge" button.

Figure 11: User Interface of Gas Station Client in C++

Server Side User Interface in C++

A gas station server user interface in C++ is composed of a data display panel and the input field for user to query. The data display panel is responsible for display the query results from the database. The user input field will allow the user to input the station name and account number to query on them.

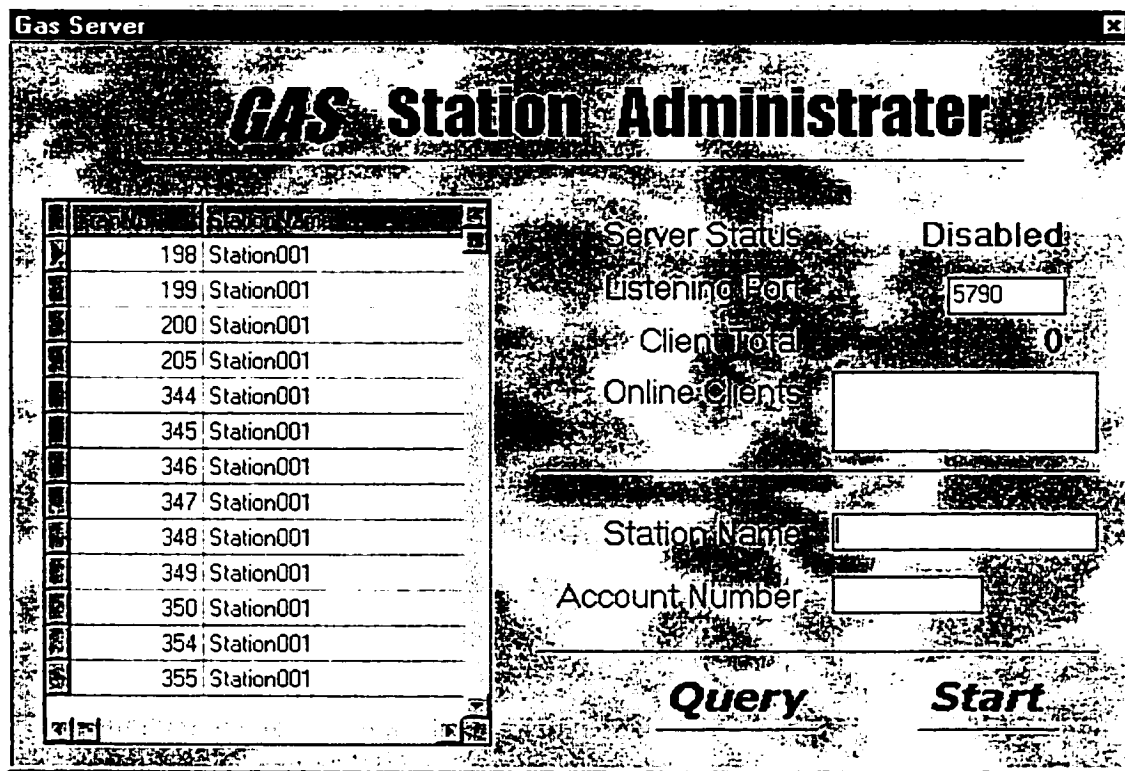


Figure 12 Use Interface of Gas Station Server in C++

3.2.3 Use of Packages and Libraries in the C++ Implementation

Client User Interface	Server User Interface	Network Communication
Controls.hpp	Buttons.hpp	ScktComp.hpp
StdCtrls.hpp	Controls.hpp	Server Database
Forms.hpp	StdCtrls.hpp	DBGrids.hpp
ExtCtrls.hpp	Forms.hpp	DBTables.hpp
ComCtrls.hpp	ExtCtrls.hpp	Db.hpp
Mask.hpp	ComCtrls.hpp	Grids.hpp
vcl.h	vcl.h	

3.3 Differences in Implementation Techniques

In order to improve the quality of the system and provide a friendly user interface, we applied some special techniques on different parts of the system during the implementation of this Gas Station Network. Here are described some technique used in Java or C++. Note that each part will be compared to C++ implementation only if the case is applicable.

3.3.1 System Structure Design

Basically, this system is composed of three parts: user interface, database, and network communication. Both Java and C++ are object-oriented language. We fully took advantage of this characteristic and designed two sets of APIs (one is for the network communication and another is for database). These APIs not only make the system integration smoother, but also make it possible to improve system functionalities separately without affecting other parts.

3.3.2 User Interface Design

In the Java user interface, the system provides to the user friendly input scheme for the account number and filling volume. It looks like a calculator panel, so the user feels more comfortable with this input style. The concrete design and implementation is in `public class GasStationClient` and described as following.

Definition of the button array and button label array:

```
 JButton digit[]=new JButton[16];  
 String keys[]={ "0","1","2","3","4","5","6","7","8","9",  
                 ".","OK","BK","RN","ST","CR"};
```

Add the action listener to each of buttons:

```
 for (int i=0; i<16; i++) {  
     digit[i]=new JButton(keys[i]);  
     digit[i].addActionListener(this);  
     keyboard.add(digit[i]) }
```

Set the button state accordingly by calling functions:

```
public void setInitState()
public void setReadyState()
public void setWaitingState(int part)
public void setWaitAmountState()
public void setWrongState()
```

In C++ implementation, it is not applicable to define an array of buttons, the input for account and gas volume is implemented using text fields.

3.3.3 Database Design

In the Java implementation, connection to database is done by JDBC to ODBC and to database. The connection method is in `public class Record` and in `public class Query`, as shown in the following.

Define the database URL

```
String url="jdbc:odbc:gasStation2"
```

Connect to the database

```
try{
    Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
    connection=DriverManager.getConnection (url);
}
catch(...) {
}
```

In C++, the database can be connected through ODBC directly. Then the tables are connected with database, whenever the table is set to *enabled* the database is connected. Dataset is the subset of the table. Whenever the data need to be displayed, the datagrid component will response for displaying the data. Dataset and datagrid are database components in Borland C++.

Both in Java and C++, the system provides data protection. To ensure data integrity, we set up a data validation mechanism to identify the current status of each record. If queried on a specific field, all the records have to be locked in case of updating by others, and unlock the records that have been updated already. In Java, the code below demonstrates how to set the validation bit to true in the database table in `public class Record`.

```
private void lockAccount(String a) {
    query="UPDATE Account SET Lock ='Y'
          WHERE AccountNumber='" + a + "'";
    try{
        statement = connection.createStatement ();
        int intRS = statement.executeUpdate (query);
        ...
    }
    catch {...}
}

private void unLockAccount(String a){
    query="UPDATE Account SET Lock='N'
          WHERE AccountNumber =' " + a + "'";
    try{
        statement=connection.createStatement ();
        int intRS=statement.executeUpdate (query);
        if (intRS==1){
            JOptionPane.showMessageDialog(this,
                                         "opensuccessfully!");
        }
        statement.close();
    }
    catch(...) {
    }
}
```

In C++ implementation, the lock and unlock task is accomplished by setting the following statements:

```
TableOfStation->Filtered=true;//Lock the dataset of table  
TableOfStation->Filtered=false;//Unlock the dataset of table
```

3.3.4 Network Communication Design

In the Java implementation, the network communication function on server side is based on Java sockets and multithreading. The main communication function is achieved by a server thread. The server thread keeps listening on a specific port 5000 and accept connect request issued by client hosts. When it accepts a new connection request, the server thread dynamically assigns a unique port number to the client host, and creates a new thread. The new thread uses the assigned port number to build a connection with the client host, to communicate with the client host, and to query the database when the client requests it. The server can make the communication more efficient by assigning a unique port number to each client to avoid competition among multiple clients.

On the client side, the client application issues a connection request initially through port 5000. When the server assigns a new port number to the client, it builds connection by the new port number and communicates with the server. The client can connect to a backup server if the main server is not available.

In C++ implementation, we also use the TCP/IP protocol to implement communication programming. The difference between client and server is important in design because each uses the socket interface differently at certain step in the communication. On the client side, we create the socket object when a connection to the server is initiated. The socket's default port number is 5790. On server side, the communication is based on multithreading, which allow multiple clients to connect to one server. We set the server socket to the multiple clients mode in order to service to multiple clients. The socket's default port number is also 5790. We use the `open()` and `close()` functions to operate on the socket. Also, `sendText()` and `receiveText()` is for passing information through the socket.

4 Comparison of Java and C++ Languages

This part will focus on describing the main differences between the C++ and Java programming language. The aspects that are applicable on the gas station case study will be mentioned on each related sub-sections.

4.1 Platform independence

Platform independence is useful in a networked environment. This means that the programs built using a programming language can run unchanged on multiple platforms. This is important in a networked environment because networks usually interconnect many different kinds of computers and devices. In a typical enterprise environment, for example, a network might connect Macs in the art department, UNIX workstations in engineering, and PCs running Windows everywhere else. Although this arrangement enables various kinds of computers and devices within the company to share data, it requires a great deal of administration. Such a network presents a system administrator with the task of keeping different platform-specific editions of programs up to date on many different kinds of computers. Programs that can run without change on any networked computer, regardless of the computer's type, make the system administrator's job simpler, especially if those programs can actually be delivered across the network. [1]

Java

The obvious advantage of Java using byte-code is platform independence. Write a program in Java, and it will run in any environment in which the Java interpreter has been implemented. A more subtle advantage of byte-code is its potential to express more instructions in fewer units. If you're loading the code across a relatively slow transport medium such as the Internet, loading and performing a final compile on a smaller amount of byte-code makes more sense than transporting and executing larger native code. [10]

Java class files and byte-code (machine independent) for tailoring a program to a specific platform is desirable in a stand-alone, homogeneous environment. But the computer world is no longer a place of separate islands. The Internet in general and the World-Wide-Web in particular are different landscapes altogether. By definition, the Internet is a network of networks, consisting of a variety of hardware and software platforms. The trick is that all these platforms speak the same protocol so that they can communicate seamlessly. Java is interpreted. That simplifies the development phase of an application by eliminating the compile (although Java needs to be compiled to byte-code), link cycles of a compiled language. The interpreter translates source code directly into byte-code. This byte-code in turn is executed on the local machine. A Java program can be executed on any machine that has an interpreter resident. The Java interpreter ensures less chance to download any viruses. When a Java program is downloaded over the Web, it is placed in a restricted region of memory. [15] The interpreter looks over all of the byte-code for the program and makes sure that they obey the laws and constraints of the language. If the byte-code is deemed safe, the interpreter lets the program run, but it still limits access to certain system resources.

C++

C++ programs are platform dependent. For example, to add a programming element to the Internet that can permeate the Web, we can not use C or C++ programs, because there's the problem of security. Most importantly for some purpose, a C/C++ program compiled for a Sun work station running Solaris will not, for instance, run on MacOS or Windows, nor will a MacOS program run on Solaris. What we need, especially for the web, is an interpreted language.

C and C++ machine code (machine dependent) is built for speed and flexibility, and is geared primarily for creating the most efficient machine code possible. Unfortunately, since the program is inevitably optimized to a particular chip set, this creates machine dependence, requiring you to recompile your code for different processors.

Verdict

The Java and C++ programming languages are both powerful high-performance programming languages. However both languages were designed for different purposes. The C++ programming language was designed to be a highly efficient general purpose programming language with object-oriented features.

On terms of platform independence, the C/C++ and Java languages are geared toward solving different problems. For Java, the issues are simplicity object orientation, innovation, and platform independence. For C/C++, the issues are speed, power, and programmer flexibility. The Java programming language was designed to be a secure, portable, completely object oriented programming language. Therefore, Java can be used by network programmers to develop secure client-server based applications that can be executed on any platform. But interpreted languages can be slow, however, if the interpreter must parse the source code into a machine language, though not one specific to any particular chip. C++ is more suitable for speed concern system although it is platform dependent.

Gas Station Implementation Result

This issue is not of any concern for the gas station system because it is not a real networked system, but only a simulation system. We did not need to consider the multi platform issue.

Conclusion

An obvious advantage of Java compared to C++ is platform independence.

4.2 Efficiency

Java

Java using byte-code and an interpreter means code executes more slowly, roughly on the order of 10 times more slowly than native C++. That is also a drawback of any interpreted language such as Java. Compiled programming languages usually have longer development cycles, however their applications will run at a much faster rate than interpreted program applications. The new execution schemer for Java is more efficient now.

C++

C++ compiler transfers the program inevitably optimized to a particular chip set, this provides faster execution speed. The machine code generated by the C++ compiler are machine dependent and optimized for a specific architecture.

Verdict

Compiled programming languages like C++ usually have longer development cycles. However their applications will run at a much faster rate than interpreted program applications. Unless Java runs on a chip supporting the op-codes of the JVM, native C/C++ code will almost certainly be faster.

Gas Station Implementation Result

The gas station software gets the obvious network performance in C++.

Conclusion

Efficiency is an obvious advantage of C++ compared to interpreted Java.

4.3 Portability

Portability is an important aspect of software development. It allows flexibility in the choice of hardware platform for each system component. Coupled to this is the need to be able to benefit from the continuous and rapid progress being made in hardware technology. [6]

Java

Java components portability can be paraphrased using an analogy to Velcro [11]. For instance, if we want to temporarily mount a pair of speakers onto our computer monitor, we could attach strips of opposing pieces of Velcro to our speakers and computer, and then put the devices together. Using Velcro instead of, say, glue, gives the ability to move the speakers at will, without any adverse effects.

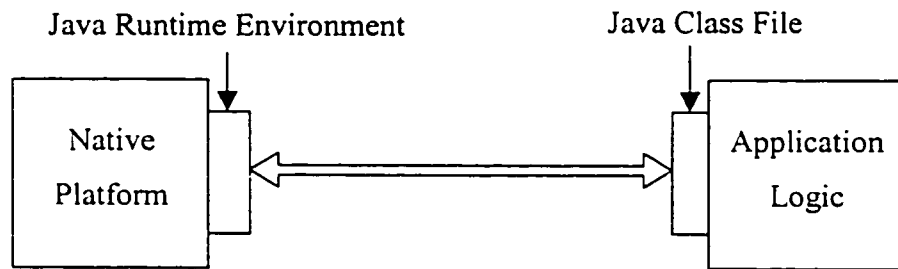


Figure 13: Java Velcro Effect

As shown in Figure 13, a Java executable, or class file, is the first piece, and the Java runtime environment is the second. We can think of the Java runtime environment as the receptor to which the Java executable bonds. The Java environment binds itself to the specific native platform and provides a shield against any platform-specific issues by eliminating machine-specific issues. This shielding is known as a layer of indirection for the Java executable.

At the time the executable is handed to the Java runtime environment for execution by the Java Virtual Machine (JVM), the application can assume a certain set of predefined functionalities. It is up to the JVM to shield the application from any platform-specific dependencies. As long as the platform has a JVM installed, all Java executables are able

to run. In this way, the Java application binds to the JVM and the JVM, in turn, binds to the specific platform.[12]

The JVM is probably one of the most misunderstood technologies of the Java suite. Some say that the JVM is an interpreter, interpreting Java-compiled byte-code into actual native machine calls. This statement is not completely true. In fact, the JVM is not an interpreter, but rather an emulator.[15] Virtual machine technology has historically been distinct from interpreter technology. The reason lies in how and at what level the interpretation takes place. Interpreters directly map a proprietary byte-code into system calls. Some map directly into machine instructions, but no attempt is made by any interpreter to emulate an intermediate machine architecture.

The JVM functions like an interpreter in some areas, but this comparison is not a direct one. If anything, we can generally refer to interpreter as scaled-down, simplified, and altogether less-complicated forms of a virtual machine. The JVM is an entire framework for a machine architecture, not unlike Intel or Motorola. The only difference is that the JVM does not require a chip or silicon-based implementation to operate.[18] As shown in Figure 14, the JVM is broken into three distinct units of functionality, which all work in harmony to achieve the primary task of executing class files.

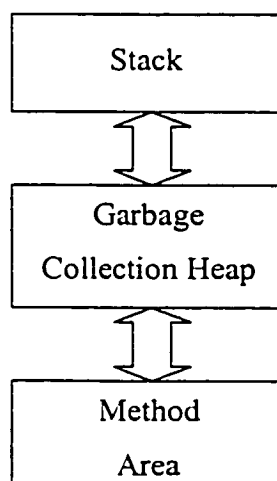


Figure 14: JVM Architecture

To summarize the task of the class loader, it finds, verifies, and loads the class into the JVM for execution. At this point, the JVM method region is loaded with all of the classes' methods, the JVM stack is loaded with class initialization, and the registers are set to the initial class execution state. All that is left is for the class loader to signal the JVM to begin execution.[21] The operating differences between a register-based architecture and JVM's stack-based model, for the point of class execution, the JVM follows the exact model specified in that for popping operands off the stack and pushing onto it for execution.

Java has other features to ensure portability. First of all, Java uses the same Application Programmer Interface (API) and Application Binary Interface (ABI) on all platforms. API is the interface used for writing source code and ABI is the executable file's format. When an API is on a range of different processors, it helps achieve program portability. Using the same ABI for all platforms is beneficial because a program that conforms to a system's ABI is able to run on any processor that complies with the ABI. Finally, Java uses the Unicode character set, which is an ASCII superset that represents the symbols of the majority of alphabets in the world.

C++

If we use C++ appropriately it has the potential for producing highly portable applications. That potential is more likely to be realized if we design for portability, select sensible programming paradigms and use the right tools (including compilers).

The importance of porting is not just supporting multiple platforms at the same time but recognizing that if we have a good product we might need to port it to a new platform at a later stage. If we are among the majority of programmers who use Microsoft Visual C++ we already know just how a bad choice of tools can badly effect our productivity. Moving from 16-bits to 32-bits will always require some work but when that is compounded by substantial changes in the compiler your task is made much more difficult.[9]

For C++, the ideas of architecture neutrality and portability go hand-in-hand. But there is more to portability. One notable difference between Java and C++ is that, in Java, the size of the bitwise representation of data types are all uniformly defined. For example, `int` always means a signed two's complement 32 bit integer, and `float` always means a 32-bit IEEE 754 floating point number. This is just one example of what C++ calls "implementation specific" details. Java makes abstraction of such details.

Verdict

A Java program can run on all computer systems. This is the case because when Java source code is compiled, this results in Java byte-code, which must be run by a Java Virtual Machine (JVM). JVM is generally an interpreter that executes the byte-code instructions. Therefore, any platform on which JVM is installed can run a compiled Java program, while identical program behavior on different systems is ensured.

Gas Station Implementation Result

This issue is not of great concern for the Gas Station System because the system is not a real network system, it is only a simulation of gas station system.

Conclusion

Portability is somehow Java's "single best feature", because it is a very obvious and strong point of Java.

4.4 Robustness

Robustness is the ability of an object to fulfill its intended function notwithstanding a changing environment and despite the violation of prerequisites. Robustness is meant to overcome our deficiencies due to the searchlight effect and tunnel vision, linear cause-effect thinking and the tendency to plausible thinking instead of deductive thinking.

Java

Java is considered to be simple because it is modeled after C++, making it easy for programmers to learn the language. Java is also simple because it removes the complex features of C++. Some of the removed features of C++ are pointers, structures, free functions, operator overloading, multiple inheritance, and preprocessor directives. [18]

C++

C++ is a more complicated programming language. Its features are redundant, offering several ways of accomplishing the same task. For example, most simple `#define` statements can be replaced by simply declaring constants, free functions can be avoided by simply defining class methods, and structures can be eliminated and replaced with classes.

The following list is just some specific C++ features needed to be concerned. Be watchful of states that can be altered inadvertently or accidentally:

1. Don't pass references and pointers to local variable.
2. Erect and strengthen barriers. Use wrapper classes. Apply maximum restrictions to the validity range of variables.
3. Use `const` declarations whenever possible.
4. Adhere to defensive programming.
5. Avoid full path names following an `#include` statement.
6. Don't rely on internal representations of data or particularities of a specific character set.
7. Use `while(i<n) ...` instead of `while(i!=n) ...` (if appropriate).
8. Set pointers to released memory to 0.

Verdict

Generally, Java is more robust :

1. Object handles initialized to null
2. Handles are always checked and exceptions are thrown for failures

3. All array accesses are checked for boundary violations
4. Automatic garbage collection prevents memory leaks
5. Clean, relatively fool-proof exception handling
6. Simple language support for multithreading
7. Byte-code verification of network applets

Gas Station Implementation Result

The Java version is easier to maintain and update than the C++ version. The Java version is more robust than the C++ version because of the aforementioned reasons.

Conclusion

From the programming aspect, Java provides more robustness capacities.

4.5 Network readiness

This feature represents the various aspects of a language that makes it simple for the programmer to build applications for the Internet, or any other networked environment.

Java

Java's main dynamic capabilities are Java's applets. They demonstrate the dynamic capabilities when it used together with Java Servlet and other applications. Java is interpreted, which means it is largely platform independent, and easily migrates programs and objects from location to location. Java has left the door open to efficient compilation in terms of current compiler technology. [17]

C++

Network programming in C++ is much more efficient than Java, but on the other hand, to consider the aspect of programming through the Internet, Java provides more safety than C++. On the other hand, C++ provides more flexible network features to make programming in C++ to be much easier.

Verdict

Usually, we choose Java as a suitable language for a browser because of Java's extensibility. A simple Java browser can be written in a fraction of the time the same browser written in C++. Java seriously cuts down the amount of coding time on large applications. This is because of the simple features of Java compared to C++.

Gas Station Implementation Result

This issue is not of great concern for the gas station system because the system is not a real networked system, it is only a simulation of gas station system. However, Java solution proves that it is faster way to program on network applications.

Conclusion

Java is more suitable for writing Web based application by using the applet and servlet packages.

4.6 Multithreading

Concurrency is important in our lives. Ironically, though, most programming languages do not enable programmers to specify concurrent activities. Rather, programming languages generally provide only a simple set of control structures that enable programmers to perform one action at a time and then proceed to the next action after the previous one is finished. The concurrency ability of Java and C++ can significantly improve the system performance.[14]

Java

Java is a popular general-purpose programming language that makes concurrency primitives available to the application programmer. Multithreading is a built-in feature of Java language. The programmer specifies that applications contain threads of execution, each thread designating a portion of a program that may execute concurrently with other

threads. This capability, called multithreading, gives the Java programmer powerful capabilities.

We can think about many applications of concurrency programming. When programs download large files such as audio clips from the World Wide Web, we do not want to wait until an entire clip is downloaded before starting the playback. So we can put multiple threads to work: one that downloads a clip, and another that plays the clip so that these activities, or tasks, may proceed concurrently. To avoid choppy playback, we will coordinate the threads so that the player thread does not begin until there is a sufficient amount of the clip in memory to keep the player thread busy.[14]

Java makes multithreading easy. It does so in two ways. First, Java provides classes that can execute as separate threads of control. Second, Java makes coordination among asynchronous threads part of the language itself, thus easing the programmer's burden.

Java has built-in multithreading support. There's a `Thread` class that inherits to create a new thread (it overrides the `run()` method). Mutual exclusion occurs at the level of objects using the `synchronized` keyword as a type qualifier for methods. Only one thread may use a synchronized method of a particular object at any one time. In other words, when a synchronized method is entered, it first "locks" the object against any other synchronized method using that object and "unlocks" the object only upon exiting the method. There are no explicit locks; they happen automatically. You are still responsible for implementing more sophisticated synchronization between threads by creating your own "monitor" class. Recursive synchronized methods work correctly. Time slicing is not guaranteed between equal priority threads.

C++

A good way to prevent our C++ programs from locking up is to use multithreading, which simply means you write a program that enables multiple execution streams to occur simultaneously within the same program. Each stream processes a different

transaction or message. For multithreading to be useful, we must run a multithreaded program in a multitasking or multiprocessing environment, allowing multiple operations to take place. For example, if you've started a lengthy, single-threaded calculation with errant parameters, the only way to stop the calculation is to terminate the application entirely. The program can't respond to our input because it's busy performing some other operation. To prevent this problem, use multithreading. [3]

In C++, synchronizing provides integrity within a program. Threads share state, address space, and resources provided by the operating system, allowing multithreading to work with shared memory and making the CPU more efficient. However, a thread's state might impact another. Programmers use synchronization to prevent this. Thread synchronization doesn't mean the threads work together in lockstep like synchronized swimmers. Rather, the threads agree about data access and modification, so one thread doesn't corrupt the data that another thread is using. The CPU scheduler doesn't understand program flow and can't anticipate when one thread might be inopportunistically interrupted. You'll need to specify explicitly such situations to prevent problems.[8]

Verdict

It is better using Java to expect multiple threads. Fortunately, for a Java programmer, implementing multiple threads means taking advantage of the language and class libraries. Java itself handles the coordination and scheduling of threads. The Java runtime environment chooses which thread gets access to the CPU. It does this by scheduling events based on priorities. All threads are not created equal: the thread with the highest priority wins. If there's more than one thread with the highest priority, the Java runtime alternates between them, or all three of them, or however many there are.

If using C++ threading technique, the programmer has to take care by himself how the threads will interact with each other, not the thread itself. It is more complicated than in Java.

Gas Station Implementation Result

The gas station need the multithread to set up the multiple connection between the client side and server side. In Java versions, the server subsystem has class inherit from thread to be able to connect with 100 clients simultaneously. Java is achieved this by define the **Threadgroup** object. In C++, the server uses the **List** structure to hold the client connection with the server. Each time the client request to connect with, the socket and client name will be added to this list. So, the server can be connected with the multiple clients simultaneously.

Conclusion

Java threading is more easier to use than C++, as multithreading feature in Java language simplify the usage of synchronizing designing.

4.7 Instant updates over the Internet

Some of the communication domains supported by the socket mechanism provide access to network protocols. These protocols are implemented as a separate software layer logically below the socket software. The socket provides many ancillary services, such as buffer management, message routing, standardized interfaces to the protocols, and interfaces to the network interface drivers for the use of the various network protocols.

Java

Java's socket-based communication system, which enables applications to view networking as if it were file I/O. A program can read from a socket or write to a socket as simple as reading from a file or writing to a file.

Just like C++, Java provides stream sockets and datagram sockets. With a stream socket a process establishes a connection to another process. While the connection is in place, data flows between the processes is in continuous streams. Streams sockets are said to provide

a connection-oriented service. The protocol used for transmission is the popular TCP (Transmission Control Protocol).

Java's TCP/IP socket support is provided by a set of two classes, `java.net.socket` and `java.net.serversocket`. The difference between the two classes lies in the role of a client and server application. Client applications open a communication channel with a server chosen by the user. Server applications maintain communication channels with multiple clients at once. The `java.net.serversocket` package facilitates a server application by spawning socket connections for each contextual client session.

C++

C++ libraries provide amazing facilities for the creation of TCP/IP client/server applications. A socket is a private communication channel between a client and server. However, one of the applications needs to initiate the socket connection. The responsibility of establishing the socket falls on the client, and the responsibility of accepting and waiting for the client request falls on the server applications.

In C++, the primary role of any TCP/IP socket server application is to accept new client connections and fulfill the requests. However, there are two distinct development tasks that can be taken in creating such an application. The first is called a blocking server, or a server that can only process a single request at a time. The second, and more common of the two, is called a concurrent server, or a server that can fulfill multiple requests simultaneously. The difference between the two is how the `accept()` call is handled.[8]

Verdict

The network communication protocol for Java and C++ are basically same. They are both TCP/IP and UDP. The byte-code for Java applets on web pages are always download from web servers which is the latest.

Gas Station Implementation Result

The gas station in both C++ and Java version use the TCP/IP protocol and socket technique to set up the connection between client and server. The only difference of network communication between the C++ and Java is they have some different functions. In principle, they are same.

Conclusion

The client side in Java can update instantly over the Internet.

4.8 Security

Security is very important for programming to avoid the potential bugs of the system. Java and C++ has their own concerns for promise the security as following.

Java

Java offers a multilevel security model. The Java Virtual Machine (JVM) architecture assesses the safety of code before the code is run. Also, the Java class loader, which is the byte-code loading mechanism of the Java interpreter, builds a wall around potentially unsafe classes. In this way, Java sets a foundation for high-level security policies that control what kinds of activities are allowed for each application. This setup of several layers of protection guards against dangerously flawed code, and viruses.

Java is more secure in the way that it does not include certain potentially unsafe C++ features. One of them is pointers. Java provides references instead of pointers, which have been called “a safe kind of pointer”. A reference is a strongly typed handle for an object, and in Java it can point only to the instances of classes.

The security manager controls the access to critical system resources. This allows the writer of a Web browser to implement a specific security policy by subclassing the security manager and overriding certain methods, and then installing the new version as

the system security manager. Since the subclassed security manager implements the security policy, it is critical that the Web browser's version of the security manager is implemented correctly. In the extreme, if a Java enabled Web browser did not install a system security manager, an applet would have the same access as a local Java application. [12]

The Java programming language provides a memory allocation model that is deferred until run time. In C++ the memory layout is declared at compile time. Since Java defers its memory allocation until run time, the memory can be mapped out according to the hardware architecture the program is run on. This memory layout is used by compiled Java code. The symbolic references in the compiled Java code are resolved to physical memory. Therefore, programmers are unable to use traditional C and C++ pointer arithmetic all memory allocation and layout is deferred until run time. The memory allocation model of Java enables applications to be more secure and reliable, and also reduces the risk of error prone source code.

C++

C++ is a strictly typed language; therefore it is not possible for a C++ application to safely import new data types at runtime. C++ does static binding in compiling time and dynamic binding as run time. It is not possible for C++ to safely determine the types and relations of objects during execution.

Verdict

Security, an important advantage of Java is that it is more secure than C++. Not only that it does not include the unsafe features of C++, but also includes some more specific security features. Java is also more type safe than C++. This makes it possible for Java to safely determine the types and relationships of objects during execution, as well as to use new kinds of dynamically loaded objects at runtime with a level of type safety.

Java attempts to offer something no other language has introduced yet and that is the idea of security built into the language and the implementation. Give the language some time and the general population will be sure to agree that its the best choice for an application that requires a secure environment. [7]

Gas Station Implementation Result

In Gas Station implementation, the Java Version avoids the use of the pointer and memory allocation that are not safe. The C++ version has to take care of type binding at the compile time and running time.

Conclusion

Java makes a lot of effort on software security.

4.9 Dynamic class integration

If the programming language is portable and interpreted, it allows programs to be more flexible and dynamically extensible. The interpreter runs the source programs directly, allowing classes to be linked on a need-by-need basis. These classes can reside on a local machine or elsewhere across the network.

Java

When the class is linked, the Java interpreter resolves the references into numeric offsets. Therefore, the storage of class objects is not done at compile time, but rather at run time. This way, when an existing class definition is modified, it will not affect any classes that refer to it. Java is based on C++, but has been simplified and cleaned up in many ways; it has been extended in other ways. All program code is in classes and types, such as arrays and strings. The programmer has less explicit control, such as control over :

1. Parameter passing modes
2. Pointer manipulations are not allowed

3. Single inheritance with a single-rooted hierarchy
4. Packages are a convenient way to provide some of the capabilities of multiple inheritance
5. Parallel programming is via threads

The role of classes in Java:

1. In Java, everything must be in a class
2. There are no global functions or data; in Java use static methods or static data within a class to simulate this
3. Forward declarations are not needed in Java
4. Source code is in `.java` files which compile to `.class` files
5. Typically there is a separate source file for every class, but even when several classes are in the same source file, the compiler produces separate `.class` files

Some issues about Java classes vs C++ classes:

Primitive types :

1. In Java, only boolean, char, byte, short, int, long, float, and double are primitive; all other types are classes
2. Even primitive types have wrapper class representations in Java so they can be treated like other classes
3. There is no automatic conversion from int or char into Boolean, and char uses the 16 bit Unicode character set

Strings :

1. Strings are a separate class, not just an array of characters
2. Any literal enclosed in double quotes is automatically of type String

Arrays :

1. As in C++, arrays have a zero index base
2. There is a read-only length member
3. Arrays are created on the heap
4. Subscript bounds are enforced; out of range errors are handles with exceptions
5. Vectors are similar to arrays, but they can grow dynamically in size as needed

Parameter Passing :

1. The programmer has no direct control over parameter passing modes in Java
2. Primitive types are passed by value, unless wrapper classes are used, in which case types are passed by reference
3. All non-primitive types are passed by reference
4. If you need a copy of an argument passed by reference, use the clone() operation
5. There are no default arguments

C++

In C++, when a class is modified by adding methods or instance variables, any classes that refer to the containing class must be recompiled. This can be a cumbersome problem, especially for large programs involving numerous classes and subclasses. [2]

C++ can be dynamically binding at the running time by using the inheritance and polymorphism that provide the flexibility to design the more dynamically extensibility. This feature is also applicable for Java.

There is no `virtual` keyword in Java because all non-static methods always use dynamic binding. In Java, the programmer doesn't have to decide whether to use dynamic binding. The reason `virtual` exists in C++ is so we can leave it off for a slight increase in efficiency when we are tuning for performance, which often results in confusion and unpleasant surprises.

Verdict

Because of some more features not present in Java, so we prefer Java is a simplification of C++ in following aspects:

1. There are no structs, enumerations, or unions
2. There is no scope resolution with `::`, Java uses the dot notation
3. There is no `goto` in Java, use `break` or `continue` to a label

4. Pointer arithmetic is not allowed
5. There are no templates in Java
6. There is no preprocessor or macro definitions
7. There are no nested classes (but “inner” classes provide similar capability)
8. There are no inline functions
9. There is no specification of “virtual” in Java

Gas Station Implementation Result

The system takes advantage of Java’s simplicity. The system also take advantage of C++’s dynamic binding and multiple data structure.

Conclusion

Java’s principle is simpler than C++ without low-level programming constructs that make for error-prone programs, but C++ gives more flexibility to the programmer.

4.10 Abstraction

Data abstraction and object-oriented programming together represent a style of programming that offers opportunities for improved software productivity. While other modern programming techniques like modular programming are similarly motivated, they often are used in concern with conventional procedural programming. They tend to emphasize ways to overcome particular problems with widely used programming practices, and thus offer incremental improvements to the art of computer programming. Because of its close association with object-oriented programming, we have to think about data abstraction and how to use data abstraction. The data abstraction similarly offers substantial benefits when used with conventional programming style. More important is the value of data abstraction as a necessary foundation, differs greatly from other programming styles and methodologies in that it requires a different way of thinking, a different approach to problem solving using computers. [2]

Java

Java is completely object oriented. Since Java does not support functional programming, every procedure must be defined as a class method. Java provides several predefined class methods that implement data types, network interfaces, GUI (Graphical User Interface) toolkits, and more. Java provides the interface keyword, which creates the equivalent of an abstract base class filled with abstract methods and with no data members. This makes a clear distinction between something designed to be just an interface and an extension of existing functionality via the `extends` keyword. It is worth noting that an abstract keyword produces a similar effect in that you cannot create an object of that class. An abstract class may contain abstract methods (although it isn't required to contain any), but it is also able to contain implementations, so it is restricted to single inheritance. Together with interfaces, this scheme prevents the need for some mechanism like virtual base classes in C++.

C++

Programmers have long recognized the value of organizing related data items in program constructs like C's `struct`, and then treating the resulting data structures as unit. Data abstraction extends this organization to encompass a set of operations that can be performed on a particular instance of the structure. Usually, the data elements and the implementation of the operations that can be performed on them are held private or encapsulated to prevent unwanted alteration. Instead of accessing data elements directly, user code, often called client programs, must invoke the permissible operations to achieve results. To do this, clients have access to a client interface or specification by which they can know how to invoke the operations.

Programming languages that support data abstraction provide a language construct-called a class in C++ and in some other languages-which we can use to encapsulate the abstract data type's data elements and operations. Although they are not precisely the same, we often use the terms "class" and "abstract data type" interchangeable. [2]

In C++, nesting a class is an aid to name hiding and code organization. Java packaging provides the equivalence of namespaces, so that isn't an issue. Java 1.1 has inner classes that look just like nested classes. However, an object of an inner class secretly keeps a handle to the object of the outer class that was involved in the creation of the inner class object. This means that the inner class object may access members of the outer class object without qualification, as if those members belonged directly to the inner class object. This provides a much more elegant solution to the problem of callbacks, solved with pointers to members in C++.

C++ is semi object oriented. Therefore functional programming is allowed in the language. This makes it easier to convert functional based programs, such as C, to the C++ language.

Verdict

Traditionally, code and data have been kept apart. For example, in the C language, units of code are called functions, while units of data are called structures. Functions and structures are not formally connected in C. A C function can operate on more than one type of structure, and more than one function can operate on the same structure.

C++ is an object-oriented version of C. However, C++ uses compile-time binding, which means that the programmer must specify the specific class of an object, or at the very least, the most general class that an object can belong to. Java is the latest object-oriented language and it has the same feature as C++ on terms of abstraction as following:

Encapsulation : implements information hiding and modularity (abstraction)

Polymorphism : the same message sent to different objects results in behavior that's dependent on the nature of the object receiving the message.

Inheritance : you define new classes and behavior based on existing classes to obtain code re-use and code organization.

Dynamic binding : objects could come from anywhere, possibly across the network. You need to be able to send messages to objects without having to know their specific type at

the time you write your code. The type of variable is going to be decided at the specific run time. Dynamic binding provides a maximum flexibility while a program is executing. [18]

Exceptionally, in Java we have abstract class instead of virtual function as in C++. Also, Java encapsulates the data and function implementation in one class, whereas C++ puts the data and function declaration in the `.h` file and implementation in `.cpp` file.

Gas Station Implementation Result

In the Java implementation, the abstract class and the interface (pure abstract class) is used as super abstract class. In C++, the base class and virtual function is used to achieve class abstraction.

Conclusion

Both Java and C++ has the feature of abstraction by using classes.

4.11 Inheritance

In object-oriented languages, we can derive one class from another class. The derived class (also called the descendant class) inherits the data members and member functions of its parent and ancestor classes. The attributes and new operators have been appended. The derived class typically declares new data members and new member functions. In addition, when the operations of these functions are not suitable for the derived class, the functions can be override by the derived class.

Java

Java uses a singly-rooted hierarchy, so all objects are ultimately inherited from the root class `Object`. Inheritance in Java has the same effect as in C++, but the syntax is different. Java uses the `extends` keyword to indicate inheritance from a base class and the `super` keyword to specify methods to be called in the base class that have the same name

as the method you're in. (However, the `super` keyword in Java allows us to access methods only in the parent class, one level up in the hierarchy.). The base-class constructor is also called using the `super` keyword. In Java, all classes are ultimately automatically inherited from the `Object` class. There's no explicit constructor initializer list like in C++, but the compiler forces you to perform all base-class initialization at the beginning of the constructor body and it won't let you perform these later in the body. Member initialization is guaranteed through a combination of automatic initialization and exceptions for uninitialized object handles. [12]

Inheritance in Java doesn't change the protection level of the members in the base class. We cannot specify `public`, `private`, or `protected` inheritance in Java, as we can in C++. Also, overridden methods in a derived class cannot reduce the access of the method in the base class. The **`final`** keyword provides some latitude for efficiency tuning it tells the compiler that this method cannot be overridden, and thus that it may be statically bound (and made inline, thus using the equivalent of a C++ non-virtual call).

C++

In C++, we can start a new inheritance tree anywhere, so we end up with a forest of trees. C++ can get multiple hierarchy. C++ appears to be the only object-oriented language that does not impose a singly rooted hierarchy. Base-class scoping in C++ allows us to access methods that are deeper in the hierarchy. These optimizations are up to the compiler.[20]

Verdict

Multiple-inheritance is a feature in C++, but Java doesn't provide multiple inheritance, at least not in the same sense that C++ does. Like `protected`, multiple inheritance seems like a good idea but we know we need it only when we are face to face with a certain design problem. Since Java uses a singly-rooted hierarchy, we'll probably run into fewer situations in which multiple inheritance is necessary. The **`interface`** keyword takes care of combining multiple interfaces in Java.

The multiple inheritance, in which a derived class may have than one base class. The syntax for expressing multiple inheritance is simple to help practice modular programming techniques in C++. Modular programming is not directly related to the major themes of project. In Data abstraction and object-oriented programming, it is nonetheless a useful programming technique. Because it provides a way to eliminate global variable and function names, multiple inheritance thus reduces the chances that you will have difficulties with duplicate names when we try to use several libraries in a single program. For example, you can use one class to have features of multiple classes instead of creating several classes.

Gas Station Implementation Result

The Gas Station System uses class inheritance in Java, and implements the interface to achieve the multiple inheritance in Java. In C++, we design the pure virtual class to work as the Java's interface.

Conclusion

Java doesn't support the multiple inheritance apparently, any class can only inherit from one class; however, by implements interface it also support multiple inheritance. C++ support multiple inheritance apparently.

4.12 Polymorphism

Polymorphism is an object-oriented programming feature that allows the instances of different classes to react in a particular way to a message (or function invocation, in C++ terms). It is ability of different objects to respond to the same message in different ways.
[7]

Java

In Java, polymorphism depends on the run-time dynamic binding as in C++. It captures similarities of objects, thus simplify maintainability of code. Java reference variables of a

class type can be used to hold references of objects of classes derived directly or indirectly. When the method is called on the reference variable, it is the version of available in the current object is invoked, not the one available in the type of reference variable. The polymorphism only happens for methods appearing in base classes.

C++

In C++, polymorphism depends on the inheritance between the base class and derived class. The base class defines the type of derived class object. When the function is called on the object, the correct version of the function will be invoked depend on the object version (base or derived). By defining a virtual function, the derived function has to implement the virtual function. When the function is called, the different version of functions will respond accordingly. The principle of virtual function calling is implemented by a C++ virtual function table. The function pointer can point to the right function when the related function is called.

Verdict

Java does not have virtual functions. Also, it does not have the virtual function table (a table structure contains pointers to virtual function) .The polymorphism is performed more efficiently in Java than C++ because save the time to an indirection to the function table. Java uses interface classes as the same function of C++ pure virtual class.

Gas Station Implementation Result

The gas station software applied polymorphism in Java implementation, which provides the extendibility of the system for future update and for easy maintainability too.

Conclusion

Java's polymorphism is performed more efficiently than C++ from the practical learning experience. This issue may need more theoretical support later.

4.13 Garbage collection

By garbage collection, the system can avoid the memory leaking. Java and C++ use different way to perform the garbage collection.

Java

Java's garbage collection means memory leaks are impossible to cause, as Java use automatically garbage collection. If you make native method calls that allocate storage, these are typically not tracked by the garbage collector. However, many memory leaks and resource leaks can be tracked to a badly written `finalize()` or to not releasing a resource leaks at the end of the block where it is allocated. The garbage collector is a huge improvement over C++, and makes a lot of programming problems simply vanish. On other hand, it might make Java unsuitable for solving some problems (sometime need control memory by designer) that cannot tolerate a garbage collector, but the advantage of a garbage collector seems to greatly outweigh this potential drawback.[18]

C++

C++ has to use destructor to delete the object form the system explicitly. In order to release the resource of the system, the destructor will be coded explicitly but called automatically. C++ also use `new` and `delete` as method to explicitly allocate and deallocate the memory. Both steps is very important to avoid memory leak error. Once the object goes out the scope of its function, the destructor will be called implicitly.

Verdict

For some cases, Java provides an easy way to release the resource and avoid the memory leaking. But for some special cases, the software needs to control the destroying the object intentionally not automatically. C++ destructors makes more sense to achieve this goal.

Gas Station Implementation Result

The gas station software in C++ has to take care of writing the destructor. The Java code has no destructor at all, the system use the feature that frees the programmer from explicitly allocating and de-allocating memory.

Conclusion

Java provide automatic garbage collection; C++ use destructor to do the garbage collection.

5 Conclusion

This part is mainly for concluding the differences between Java and C++ based on the experience got from the project practice and academic research as well.

Language Complexity

Java is smaller than C++. It has fewer facilities. This is not necessarily a bad thing, in a different context, 'make smaller to make better'. Most of the features of high-level programming languages that are not absolutely vital have been left out of Java. The advantages, generally, of a smaller language are:

1. Easier to learn
2. Easier to understand, test, debug and maintain a program
3. Concentration on essentials.

Arguably, C++, which was derived from C, is large and complex. So Java is perhaps a welcome return to basics. The bottom line is that C++ has a number of features that are not present in Java. Examples are: operator overloading, header files, a preprocessor, pointers, structures, unions, templates and implicit type conversion. Some of these are arguably useful and others are superfluous.

Object-oriented programming

Both Java and C++ support object-oriented programming. However, Java is object-oriented from the ground up – it was explicitly designed from the start to be object-oriented. A full object-oriented language is called an object language. C++ was derived from C (not an object-oriented language). It is easy to write a non-object-oriented program in C++, but not possible in Java.

Inheritance

Java supports only single inheritance. This means that a class can only inherit from one other class. There are major arguments in object-oriented circles about inheritance – what is useful and what is desirable. Some languages, including C++, support multiple inheritance, where a class can inherit from more than one other class. The designers of Java thought that this was complicated and unnecessary.

Compilation and interpretation

Both Java and C++ are compiled, but whereas C++ is usually compiled to machine code, Java is compiled to a machine-independent low-level code called byte-code. This byte-code is then interpreted by the Java Virtual Machine running on the particular machine that is apparently running the Java program. This gives the Java code platform independence – the same type code can be run on any of a huge variety of machines, with different operating systems.

Machine independence

Because of Java byte-code, Java programs can be run on any of a variety of machines. This machine independence goes further, however. In C++, for example, there is uncertainty about how big a variable declared as an integer can be. Depending on the architecture of the target machine, a C++ integer can be 16, 32 or 64 bits long. Thus the way that a program behaves will be different from one machine to another. Such ambiguity is removed in Java, because an integer is defined to be 32 bits long and a long integer is defined to be 64 bits long – whatever the architecture of the machine it is running on. Java is so machine-independent that porting a Java program to another machine does not require recompilation.

Robustness and security

Java programs run on a virtual machine, rather than a real machine (like C++). This means that the Java Virtual Machine can carry out a number of checks that a program is running properly. For example, references to arrays are checked to ensure that the subscript is within the array bounds. When something goes wrong – the program does not crash (as with C++), but instead the virtual machine performs an orderly handing of the situation. Similarly, the Virtual Machine can check for viruses within Java byte-code.

Concurrency

Concurrency is sometimes called multithreading or parallelism. It is the ability to do two or more things at once. Now, while it is true that a single-processor computer can only execute one instruction at once, a computer is sufficiently fast to switch between a number of task that are apparently being carried out simultaneously. This switching is called multithreading, multitasking or concurrency. Java was designed from the start to provide concurrency (as part of the libraries). C++ does not support concurrency in such an integrated way.

Garbage collection

When a C++ program requests memory to use as workspace, it must keep track of it and return it to the operating system when it ceases to use it. This requires extra programming and extra care. It is a common source of errors in C++ programs. This task of garbage collection is carried out automatically in Java – an object that is no longer used is automatically destroyed and the memory released. On the other hand, the explicit release of resources (as is carried out in a C++ destructor function) is useful for releasing such resource as files and windows.

Pointers

C++ has pointers but Java does not. Pointers are a way of referring to the actual memory address of a variable or a function. Programming with pointers has long been a touchstone of the real hard-bitten programmer. But programming with pointers is extremely

error-prone and, because of the dangerous nature of machine addresses, can lead to subtle bugs and sometimes errors that defy removal.

In C++, parameters can be passed to functions either as values or references (pointers). In Java, any parameter that is a built-in type is passed to methods by value. This creates a degree of safety, because a method cannot change the value of the parameter back in the invoking method. If a value is to be returned to the user of a method, it is returned as the value of the method (function).

Libraries

Like C++, Java programs make extensive use of functions (methods) from libraries. This tends to make the language itself smaller as many actions are carried out by library functions rather than the language itself. An example is input-output facilities.

Strings

In C++, strings are represented and manipulated as arrays of characters. Although there are plenty of library functions to help, it is not always convenient, it is not conceptually simple and it can be error-prone. In Java, a string is not the same as an array of characters; it is simpler in concept and is more easily manipulated. For example, rather than terminating a string of character with a special character (as in C++), strings in Java have their length stored alongside them. Both C++ and Java provide a comprehensive set of library functions to manipulate strings.

Data structures

Both Java and C++ support one-dimensional arrays. C++ directly supports multidimensional arrays, but this is done in Java by using the concept that a component of an array can itself be an array. The effect is therefore very much as if multidimensional arrays are directly supported. Java does not support pointers that can be manipulated by the program. At first sight this is disappointing for programmers who like to build linked lists. But Java does not use pointers – though they are invisible and inaccessible to the

programmer. In a real sense, Java provides a higher level view of data structures than C++, a view in which pointers are at a lower level, invisible and managed by the system. To implement this in C++ requires the explicit use of pointers. In Java however, the way that it can be implemented is to regard every item in the lists (an Object) as consisting of the data item accompanied by another object – the remainder of the list. The Java code looks like this. A particular example is where the data item in each item of the list is a single integer.

Exception handing

Exception are unusual situations that arise as a program is running. Some can be anticipated by the programmer; others are less easily foreseen. Some programming languages, including C++ and Java, provide language mechanisms, called exception handlers, for dealing with exceptions.

Templates

One of the more popular object-oriented features of C++ is templates. Templates allow the programmer to create a set of functions (methods) that act on a type that is not defined. The type of the data is defined when the class is used. This powerful facility allows the programmer to write a reusable, general-purpose classes. Java does not support templates. But all the Java object derived form the `Object` class can be cast to any type. This provides a functionality similar to C++ templates.

Names and packages

Problem with large C++ programs is that there can be a clash between different classes with the same name. This can be a particular problem with large software, written by a number of people and perhaps using library classes from number of sources.

Java helps to solve this problem by providing a package concept. A number of classes can be grouped into a package. Two classes with the same name but in different packages

are regarded as unique. C++ has a facility called `namespaces` which solves this same problem.

General Conclusion

Java and C++ have similar features which are applicable to solve the common problems. However, the difference between Java and C++ has been accepted and applied by the different domain requirements. Especially, Java is used as a rapid GUI generate tool; meanwhile, C++ takes main role on developing the low level application for its compatibility with the C language. Comparing on them give us the clear and meaningful solutions on selecting language decision. From GAS Station system development, I have got the practical experiences on how to utilize Java and C++ effectively and efficiently.

References

- [1] Dov Bulka and David Mayhew, *Efficient C++: Performance Programming Techniques*, Addison-Wesley, 2000, pp. 30-41.
- [2] Frank M. Carrano, Paul Helman, and Robert Veroff, *Data Abstraction and Problem Solving in C++: Walls and Mirrors (Second Edition)*, Addison-Wesley, 1998, pp. 102-121.
- [3] Tracey Hughes, *Object-Oriented Multithreading Using C++*, John Wiley & Sons, 1997, pp. 44-65.
- [4] Personallogic URL: <http://www.personallogic.com/>
- [5] Firefly URL: <http://www.firefly.com/>
- [6] Scott Meyers, *Effective C++: 50 Specific Ways to Improve Your Programs and Designs (Second Edition)*, Addison-Wesley, 1998, pp. 33-54.
- [7] Scott Meyers, *More Effective C++: 35 New Ways to Improve Your Programs and Designs*, Addison-Wesley, 1996.
- [8] Jeff Prosise, *Programming Windows with MFC (Second Edition)*, Microsoft Press, 1999, pp. 76-95.
- [9] James T. Smith, *C++ Toolkit for Scientists and Engineers (Second Edition)*, Springer, 1999, pp. 23-108.
- [10] Daniel J. Berg and J. Steven Fritzinger, *Advanced Techniques for Java Developers (Revised Edition)*, John Wiley and Sons, 1999, pp. 211-154.
- [11] Thomas W. Christopher and George K. Thiruvathukal, *High Performance Java Platform Computing*, Prentice Hall, 2000.
- [12] Li Gong, *Inside Java 2 Platform Security: Architecture, API Design, and Implementation*, Addison-Wesley, 1999, pp. 154-234.
- [13] Elliotte Rusty Harold, *Java Network Programming (Second Edition)*, O'Reilly & Associates, 2000.
- [14] Bill Lewis and Daniel J. Berg, *Multithreaded Programming with Java Technology*, Prentice-Hall, 2000, pp. 35-103.

- [15] Tim Lindholm and Frank Yellin, *The Java Virtual Machine Specification* (Second Edition), Addison-Wesley, 1999, pp. 233-302.
- [16] Scott Oaks, *Java Security* (Second Edition), 2001.
- [17] Robert Orfali and Dan Harkey, *Client/Server Programming with Java and CORBA* (Second Edition), John Wiley & Sons, 1998.
- [18] Govind K. Seshadri, *Enterprise Java Computing: Applications and Architecture*, Cambridge University Press, 1999.
- [19] Dick Steflik and Prashant Sridharan, *Advanced Java Networking* (Second Edition), Prentice-Hall, 2000, pp. 265-305.
- [20] Nigel Warren and Phil Bishop, *Java in Practice: Design Styles and Idioms for Effective Java*, Addison-Wesley, 1999, pp. 24-56.
- [21] Steve Wilson and Jeff Kesselman, *Java Platform Performance: Strategies and Tactics*, Addison-Wesley, 2000, pp. 211-256.

Appendix Operation Instructions

Server side

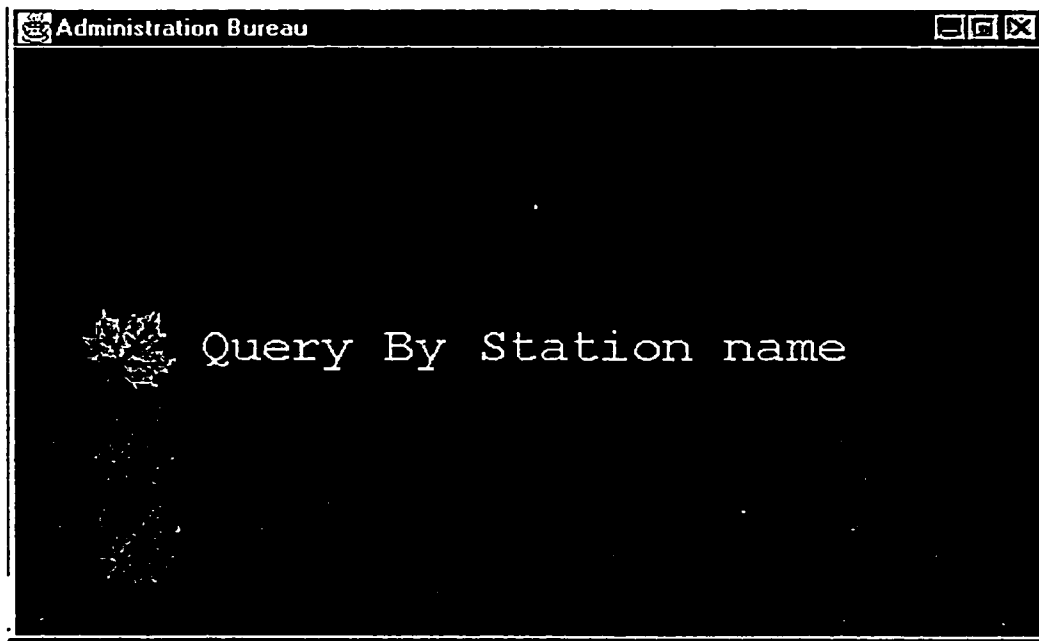
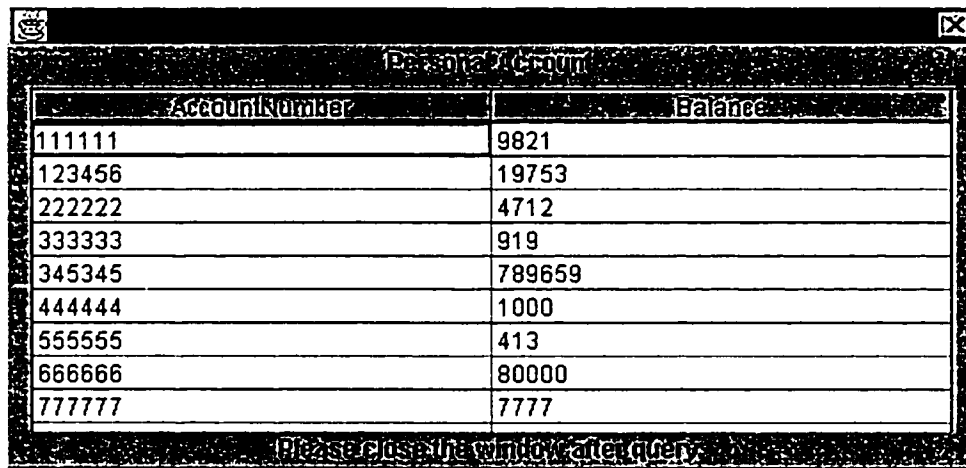


Figure 15: Server user interface of project with Java language

There are 3 functions offered to general user:

- Query transactions by station name,
- Query transactions by transaction number,
- And Query personal account.

When mouse moves over certain item, the item will be highlighted. Then, click the highlighted item will display the related report form. Fig.16 is an example of personal account report.



Account Number	Balance
111111	9821
123456	19753
222222	4712
333333	919
345345	789659
444444	1000
555555	413
666666	80000
777777	7777

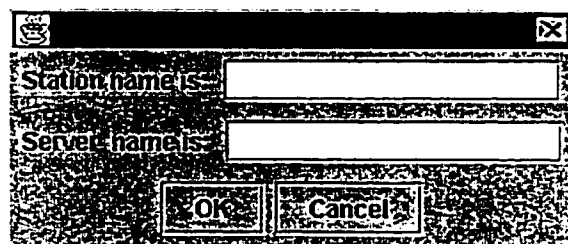
Please close the window after query

Figure 16: Personal account report

Client side

Start

To start a gas station (client application), you have to input the **gas station name** and the **server name**. You can input them on command line, like '`java ClientProject Gas-Station-Name Server-Name`'. If not, system will display a dialog box (fig. 17) asking you to input them. Note that the connection with the server will not be established with a wrong server name. Press 'Cancel' will quit the system.



Station name is:

Server name is:

OK Cancel

Figure 17: System need a valid server name

Monitor

The whole appearance of a gas station is shown as fig15. The panel of monitor (fig.18) is the heart of the gas station.

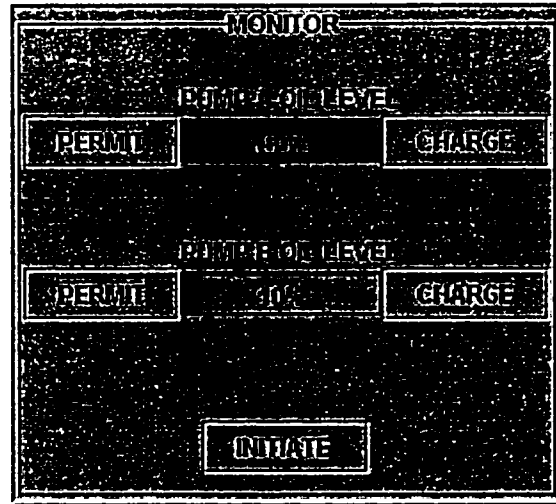


Figure 18: Monitor panel of a gas station

The following is the introductions of each part of the monitor:

- Each combination of a PERMIT button, a progress bar and a CHARGE button manages related pump. The **Progress Bar** displays the gas level of related pump.
- When a user press 'Request' button on a pump panel, the related **PERMIT** button will turn to red and flick. That means that someone is asking for filling gas. Press the **PERMIT** button will give that pump authority and enable that pump.
- When the gas level of a pump is lower than a limit (5%), the related **CHARGE** button will turn to red and flick. If an gas pump is working, it will be stopped automatically. Press **CHARGE** button will re-charge that gas pump to level 100%.
- The **Progress Bar** indicates the current gas level of related pump.
- Press **INITIATE** button invokes a set-up dialog box (fig.19).

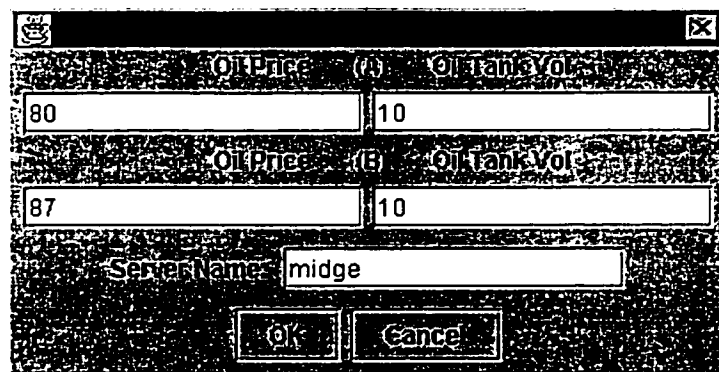


Figure 19: Set-up dialog box

In this box, user can change the gas price of each pump, the gas tank volume of each gas pump, and change the server. Note that if any pump is working, INITIATE button is disabled. If the set-up dialog box is on, any pump operation is prohibited.

Pump

There are 2 pumps in each gas station, Pump A and Pump B. They work independently. Their operation procedures are same. Fig.20 is the picture of a gas pump panel.

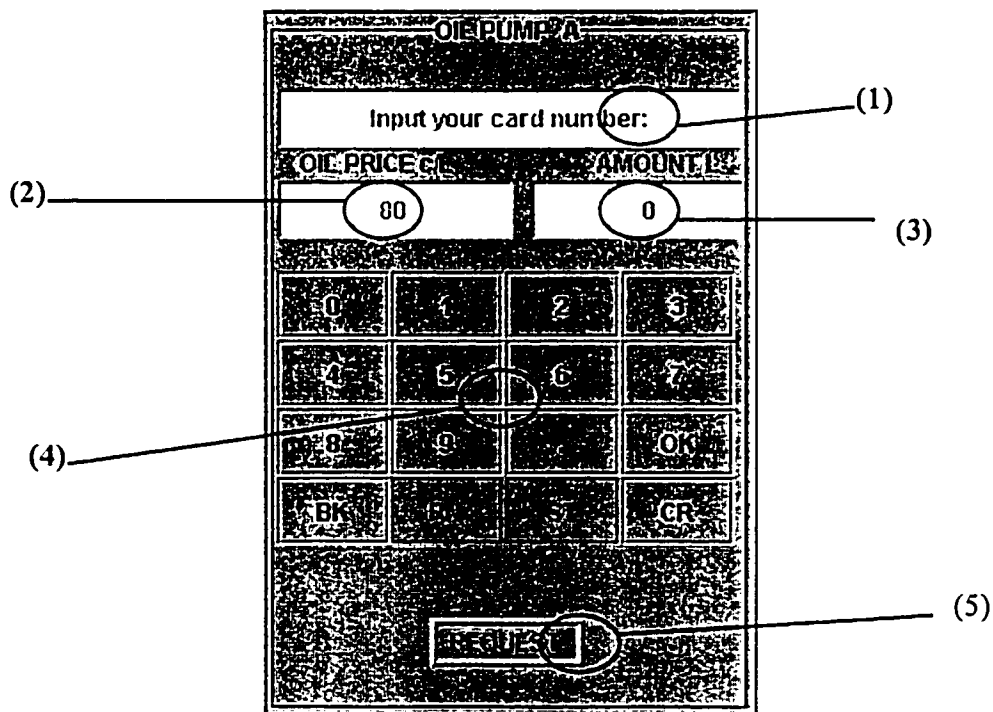


Figure 20: Gas pump panel

Display area (1) : Display various prompt information, such as balance, wrong input, etc.

Gas price area (2) : Display the current gas price. It will be changed automatically after gas price is updated on monitor panel.

Amount area (3) : Display how many liters of gas a client want to fill.

Keyboard area (4) : Client input by clicking this keyboard with mouse. OK button is like an enter key. BK button will back and erase the last input number. RN (run) button will start or resume filling gas. ST (stop) button will stop filling gas. And CR (clear) button will quit a transaction at any time.

Request area (5) : There is only one button inside this area – REQUEST button. Whenever a client wants to begin a filling-up session, he or she must press REQUEST button first. Monitor will receive this request and give a feedback to client. Then the pump panel will be enabled and the client may start the session.

A Filling-up Session

The filling-up session is mainly for describe the filling process. The following procedure is a complete filling-up session, suppose that both server and client sides (Administration station and gas station) are in normal states:

- Click **REQUEST** button on pump A panel, a filling-up session start,
- **PERMIT** button (related to pump A) on monitor panel turns to red and flicks,
- Click **PERMIT** button means that the request has been authorized, **PERMIT** button then turns back to normal (gray color), and the panel of pump A is enabled,
- **Display area** displays ‘Input your card number:’,
- Click digit buttons with mouse to input the card number (maximum length is 6), click **BK** button to erase the last input digit, click **OK** to acknowledge,
- If the card number is not valid or if somebody is using this number, **Display area** will show ‘Invalid number, input again’, then back to previous step,
- If the card number is valid, **Display area** will show the **balance** of the client and how many liters of oil he/she can fill according to the oil price,

- After reading the balance information, click **OK** button,
- Then **Display area** shows 'How much do you want?' to ask client to input the quantity of oil he/she wants to fill,
- Click the digit button to input figure, click **BK** button to update and click **OK** button to acknowledge,
- **Display area** displays 'Press **RN** to start filling', and **Amount area** display the amount of oil the client wants to fill,
- Click **RN** button will start filling, **Display area** display 'You've charged xx liter' continuously,
- Now, if click **ST** button (stop) the filling-up will be paused and **Display area** will display 'Press **RN** to start filling'; after click **ST** button, click **RN** button will resume filling-up,
- During the filling-up session, if the oil level of pump A is lower than 5%, filling-up will be stopped and the **CHARGE** button on monitor panel will turn red and flick, **Display area** displays 'Oil level low, please wait'; click the red **CHARGE** button will re-charge the oil tank of pump A, and then, **Display area** displays 'Oil level OK, press **RN** continue',
- When filled oil reaches to the target of the client, the filling-up will be finished and display area shows 'Card number xxxxxx, new balance xxx \$', this information will displays for 5 seconds,
- 5 seconds later, the filling-up session is completed, and the pump panel bake to initial state.

Note: Anytime click **finish** button (clear) will terminate the transaction immediately.