A JAVA-APPLICATION CODE GENERATOR FROM DATA FLOW DIAGRAM

CHIRAWAN RONRAN

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR
THE DEGREE OF MASTER OF SCIENCE
(TECHNOLOGY OF INFORMATION SYSTEM MANAGEMENT)
FACULTY OF GRADUATE STUDIES
MAHIDOL UNIVERSITY
2005

ISBN 974-04

Thesis
Entitled

A JAVA-APPLICATION CODE GENERATOR FROM DATA FLOW DIAGRAM

.....
Miss Chirawan Ronran
Candidate

.....
Lect.Kanat Poonsawad, M.Sc.
Major-Advisor

.....
Lect.Rangsipan Marukatat, Ph.D.
Co-Advisor

.....
Lect. Sritida Kajornpreedanon, M.S.
Co-Advisor

.....
Assoc.Prof.Rassmidara Hoonsawat, Ph.D.
Dean
Faculty of Graduate Studies

.....
Assoc.Prof.Piya Rattanasuwan, B.Eng.(Civil),M.Eng.
Chair
Master of Science Programme in Technology of Information System Management
Faculty of Engineering

Thesis
Entitled

A JAVA-APPLICATION CODE GENERATOR FROM DATA FLOW DIAGRAM

was submitted to the Faculty of Graduate Studies, Mahidol University
For the degree of Master of Science
(Technology of Information System Management)
on
May 26, 2005

.....
Miss Chirawan Ronran
Candidate

.....
Lect.Kanat Poonsawad, M.Sc.
Chair

.....
Lect.Rangsipan Marukatat, Ph.D.
Member

.....
Capt.Aran Namphol, Ph.D.
Member

.....
Lect. Sritida Kajornpreedanon, M.S.
Member

.....
Assoc.Prof.Rassmidara Hoonsawat, Ph.D.
Dean
Faculty of Graduate Studies
Mahidol University

.....
Assoc.Prof.Piya Rattanasuwan, M.Eng.(Civil),M.Eng.
Dean
Faculty of Engineering
Mahidol University

ACKNOWLEDGEMENTS

The success of the thesis can been attributed to the extensive support and assistance from my major advisor, Lect. Kanat Poolsawasd and my co-advisor Lect. Rangdsipan Marukatat, Lect. Srithida Kajornpreedanon. I deeply thank them for their helpful guidance and suggestion though out this study.

I also would like to express my sincere gratitude to external examiner, Lect Aran Namphol, for his recommendations.

I wish to express my sincere thank to all teacher and staffs in the Technology of Information System Management program, Mahidol University and Computer Science department of the Faculty of Science, Maejo University for good teaching from teachers and for good co-operation and generous from all staffs.

Finally, I am grateful to my family for their encouragement, entirely care and love during study in this research and my entire life.

Chirawan Ronran

A JAVA-APPLICATION CODE GENERATOR FROM DATA FLOW DIAGRAM.

CHIRAWAN RONRAN 4637206 EGTI/M

M.Sc.(TECHNOLOGY OF INFORMATION SYSTEM MANAGEMENT)

THESIS ADVISORS: KANAT POOLSAWASD, M.Sc., RANGSIPAN MARUKATAT, Ph.D., SRITHIDA KAJORNPREEDANON, M.S.

ABSTRACT

In this research, Object-Oriented technology was used to create a Java-Application Code Generator from Data Flow Diagram. This application is useful for a system designer or system developer/programmer who wants to create a Java program by using the top-down and bottom-up strategy of structure programming.

The application helps a system designer to draw multi-level and check illegal data flow diagrams. It also helps a system developer/programmer to transform the data flow diagrams into Java-Application code.

The data flow diagrams of this application were created from higher level to lower level data flow diagrams to support the top-down strategy while the generated code of this application was implemented from lower level to higher level data flow diagrams to support bottom-up strategy.

The outcome of this research is a Java-Application Code Generator from Data Flow Diagram. It was evaluated by using it to generate code for a mobile phone game and then comparing the code with the previous code which was developed by J2ME. The result of system evaluation was satisfactory.

KEY WORDS : DATA FLOW DIAGRAM / CODE GENERATOR / TRANSFORMATION DFD

99 P. ISBN 974-04-6151-4

โปรแกรมสร้างโค้ดภาษาจาวาจากแผนภาพกระแสข้อมูล (A JAVA-APPLICATION CODE GENERATOR FROM DATA FLOW DIAGRAM.)

จิราวรรณ รอนราญ 4637206 EGTI/M

วท.ม. (เทคโนโลยีการจัดการระบบสารสนเทศ)

คณะกรรมการควบคุมวิทยานิพนธ์ : ฆนัท พลูสวัสดิ์, M.Sc., รังสิพรรณ มฤคทัต, Ph.D., ศรีธิดา ขจรปรีดานนท์, M.S.

บทคัดย่อ

การวิจัยนี้ใช้เทคโนโลยีการพัฒนาโปรแกรมเชิงวัตถุโปรแกรมเพื่อสร้างโค้ดภาษาจาวาจากแผนภาพกระแสข้อมูล โปรแกรมนี้มีประโยชน์แก่นักออกแบบและนักพัฒนาระบบด้วยโปรแกรมภาษาจาวา โดยใช้หลักการ top-down และ bottom-up ของการเขียนโปรแกรมแบบมีโครงสร้าง

นักออกแบบระบบจะใช้โปรแกรมประยุกต์นี้ในส่วนของการวาดและการตรวจสอบความถูกต้องของแผนภาพกระแสข้อมูล สำหรับนักพัฒนาระบบสามารถใช้โปรแกรมประยุกต์นี้เพื่อแปลงแผนภาพกระแสข้อมูลเป็นโค้ดภาษาจาวา

แผนภาพกระแสข้อมูลของโปรแกรมประยุกต์นี้จะสร้างจากแผนภาพกระแสข้อมูลระดับบนมายังแผนภาพกระแสข้อมูลระดับล่างเพื่อสนับสนุนการทำงานในลักษณะ top-down ขณะที่โค้ดภาษาจาวาจะพัฒนาจากแผนภาพกระแสข้อมูลระดับล่างขึ้นสู่แผนภาพกระแสข้อมูลระดับบนเพื่อสนับสนุนการทำงานในลักษณะ bottom-up

ผลจากงานวิจัยคือโปรแกรมสร้างโค้ดภาษาจาวาจากแผนภาพกระแสข้อมูล จากการเปรียบเทียบผลการสร้างโค้ดภาษาจาวากับโปรแกรมเกมบนมือถือที่ใช้ J2ME ในการพัฒนา ได้ผลการเปรียบเทียบในระดับที่น่าพอใจ

99 หน้า. ISBN 974-04-6151-4

CONTENTS

CONTENTS (Cont.)

LIST OF TABLES

LIST OF FIGURES

CHAPTER I
INTRODUCTION

1.1 Background and Problem Statement

Visual modeling is a way of thinking about problems using model organized around the real-world ideas. Models are useful for understanding problems and communicating with everyone involved with the project.

Models are abstractions that portray the essentials of a complex problem or structure of a complex problem or structure by filtering out nonessential details thus making the problem easier to understand. To build a complex system, a developer must abstract different views of the system, build models using precise notations, verify that the models satisfy the system requirements, and gradually add details to transform the model into an implementation by coding which is tedious, repetitive, and error-prone [9]. To reduce these difficulties in the implementation stage, a number of code generators have been developed.

Code generator is a program that takes a model as input and produces output source code, which is an implementation form of that model. The ability to automate some coding tasks can increase productivity because generating code faster than writing it by hand [14]. Moreover the code is usually consistent because a program that writes code, unlike an inattentive developer, always follows the coding standard of that program [13].

In the last few years, code generators have many benefits and become increasingly common in software development. Most of them work with models that consist of metadata containing information about classes, methods, attributes and parameter names such as Unified Modeling Language (UML). Even though Data Flow Model, which describes the system in the term of Data Flow Diagram or DFD, is as widely used as UML, it is not supported by any code generator. So, *A Java-Application Code Generator from Data Flow Diagram* will be developed to generate

code from Data Flow Diagrams; which helps reduce the time scale and manpower require for implementation.

1.2 Objective

The objectives of this study are as follows:

1.2.1 To analyse algorithms and techniques involved in Java code generation.

1.2.2 To design and implement a prototype of A Java-Application Code Generator from Data Flow Diagrams

1.3 Scope of Study

1.3.1 The study aims to apply the object-objected paradigm (Object Modeling Technique or OMT) for the analysis and design of a Java-Application Code Generator from Data Flow Diagram

1.3.2 A prototype of a Java-Application Code Generator will be implemented, which performs the following modules:

1.3.2.1 Drawing Data Flow Diagram by use Gane and Sarson symbols.

1.3.2.2 Edit the Data Flow Diagram

1.3.2.3 Save the Data Flow Diagram

1.3.2.4 Check illegal Data Flow Diagram

1.3.2.5 Generating skeleton code of Java application from Data Flow Diagrams

1.4 Expected Result

Expected from this study are a prototype of a Java-Application Code Generator from Data Flow Diagrams and general guidelines to develop code generator for other programming languages from the Data Flow Diagrams.

CHAPTER II
LITERATURE REVIEW

The objective of this chapter is collecting and reviewing studies related to Data Flow diagrams, Java language, and code generation techniques. The chapter consists of the following topic:

- Elements of Data Flow Diagrams (DFD)
- Anatomy of a Java Class and Java Syntax
- Code generation technique
- Related Research

2.1 Elements of Data Flow Diagrams (DFD)

A data flow represents incoming of data to a process or outgoing of data (or information) from a process. A data flow is also used to represent the creation, reading, deletion, or updating of data in a file or database. There are four type of symbols in the DFD language (processes, data flows, data stores, and external entities), each of which is represented by a different graphic symbol [1]. There are two commonly used styles of symbols, one is developed by Charis Gane and Trish Sarson and the other is developed by Tom Demarco and Ed Yourdan [10]. Both styles are equally popular; some organizations use Gane and Sarson style and others use DeMarco and Yourdon style.

A process (Figure 2.1 [15]) is an activity or a function that is performed for a specific business reason. Processes can be manual or computerized, and every process has a name that start with a verb and end with a noun. The name should be short but contain enough information so that readers can easily understand exactly what it means. In general, each process performs only one activity. So most system analysts avoid using the word *and* in process names because it suggests that process performs several activities.

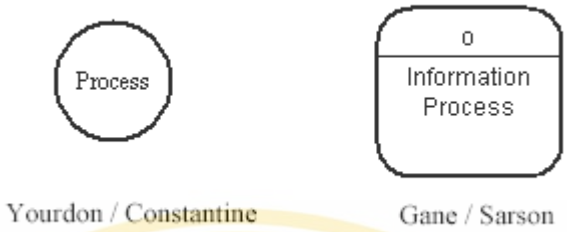


Figure 2.1 Process

Every process has a unique identification number, a name, and a description, all of which are noted in the CASE (computer-aided software engineering) repository. Descriptions clearly and precisely describe the step and detail of the processes; ultimately they are used to guide the developers who need to computerize the processes. More detail will be added to the process description as more information is learned throughout the analysis phase.

A data flow (Figure 2.2 [15]) is a single piece of data, or a logical collection of several pieces of information. Every data flow has a descriptive name that is a noun, and a description. Typically, the description of a data flow will list exactly what element the flow contains. For example, the patient information data flow can list the patient name, address, and phone number as its data elements.

Figure 2.2 Data Flow

Data flows are the glue that holds the processes together. One end of every data flow will always come from or go to a process, with an arrow showing the direction into or out of that process. Data flows show what inputs go into each one-

output data flow because if there is no output, the process does not do anything. Likewise, each process usually has at least one input data flow because it is difficult if not impossible to produce an output without any input.

A data store (Figure 2.3 [15]) is a collection of data that is stored in a file or a database. Similar to processes, every data store has a descriptive name, which is a noun, an identification number, and a description. Data stores are the starting points of the data model, and are the principal link between the process model and the data model.

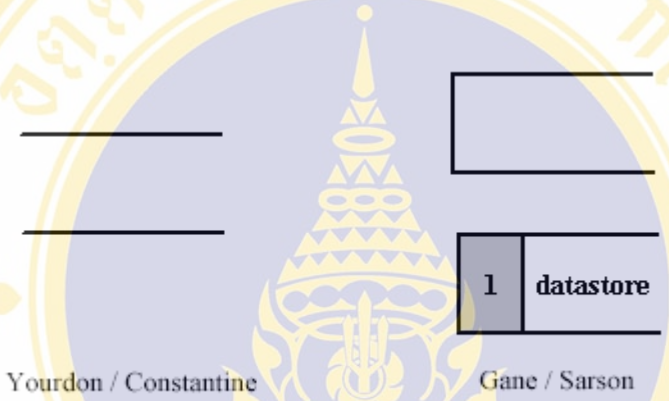


Figure 2.3 Data Store

A Data flow coming out of a data stores indicate that information is retrieved from the data store and data flow going into a data store indicates that information is added to the data store or that information in the data store is changed. Whenever a process update a data store, both the data coming from the data store and the data written back into the data store.

All data stores must have at least one input data flow, unless they are created and maintained by another information system. Likewise, they usually have at least one output data flow. In case that the same process both stores data and retrieves data from a data store, there is a temptation to draw one data flow with two arrows on the both ends. However, for clarity, it is a better practice to draw two separate data flows.

An External Entity (Figure 2.4 [15]) is a person, organization, or system that is external to the system but interact with it. Every external entity has a name and a

description. A key point to remember about an external entity is that it is external to the system but may or may not be part of the organization.

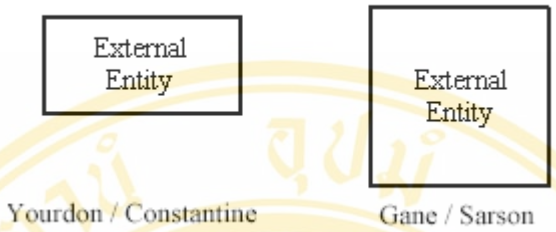


Figure 2.4 External Entity

A common mistake is to include people who are part of the system as external entities; the people who execute processes are part of the processes and are not external to the system. A person who performs a process is often described in the process description but never on the DFD itself. However, a person who uses information from the system to perform other processes or who decide what information goes into the system is documented as an external entity.

2.2 Anatomy of a Java Class and Java Syntax

The formal syntax of a basic Java class definition is shown below; [5] angle brackets <...> indicate optional items, and reserved words are shown in boldface.

<*access*> **class** *classname* <**extends** *classname*> <**implement** *interface*,…, *interface*>

```
{
    // Declare all attributes of the class as follows:
   <access> datatype attribute name;
   // Define all methods as follows:
   <access> return type method name (optional_arugument_list)
   {
     code to implement the method; i.e., the method body
   }
}
```

<*access*> for a class that stands alone in a .java file can be omitted or declared public, and <*access*> for a class feature inside the class can either be omitted or declared public, private or protected.

2.3 Code Generators

Some code generators are discussed by [12]. The details of each tool are described as follow:

2.3.1 Visual CASE 6.0[1]

Visual CASE 6.0 offers a rather good integration with the Microsoft Visual C++ 6.0 development environment. Visual CASE 6.0, developed by the Stingray division of Rogue Wave Software, and runs as an add-in UML modeling tool to the Visual C++ 6.0 user interface and also integrates fully with Visual SourceSafe. It provides support for MFC and ATL and gives roundtrip engineering on class diagrams.

The tool supports the automatic update of class diagrams when changes are made within the C++ code without the code having to be regenerated. The tool also has an ability to generate skeleton code from the class diagrams created by the users. It also allows the users to draw use-case and sequence diagrams but these do not support reverse engineering. One of the most pleasing aspects of Visual CASE is its seamless and unobtrusive integration with the existing Visual C++ developer environment. With Visual CASE active the project workspace window displays a new 'UML' tab, and a further 'UML Workspace' window is available that provides access to status information about the active project and control over the UML model libraries that can be activated.

The tool provides its own UML library for MFC 6.0 classes, and any of these classes can be dragged and dropped onto the UML diagram window. The synchronization between the UML diagrams and the source code is rather good. Adding a class in the project window would create the appropriate UML Workspace entries. It has been stated that future versions will also be able to import and export files generated by Rational Rose.

[1] Visual Case™ is a trademark or registered trademark of Artiso Corp.

Strengths

- Ease of use
- Integration with Visual C++
- Synchronization capabilities
- C++ classes generated direct from UML diagrams

Weaknesses

- Limited reporting facilities
- Depends on SourceSafe for version control

2.3.2 Select Enterprise

Select Enterprise is intended to supply more than a UML modeling tool. It goes somewhat further than UML by specifying 'Select Perspective'. This methodology is used to reduce business requirements to a set of diagrams that supplement the usual class, interaction, state, and use-case models of UML with additional business process models.

The Select Enterprise process was built largely on Rumbaugh's Object Modelling Technique (OMT). This new version still provides a set of notational tools, but Select Enterprise would like the user to adopt rather more than just the product and goes to some lengths to describe the Select Perspective itself.

In this approach, the analyst starts with an architectural model in which collections of local business objects are subsumed under groups of corporate business objects. This is a four layered view of the software system being modeled, comprising local business objects, corporate business objects, interface objects and storage objects. The benefit of this approach is that it provides a fairly intuitive way of adding layers of detail without having to comprehend all the possible interactions in one go.

As it stands, Select Enterprise supports C++ code and SQL schema generation. Round-trip engineering is supported in all target languages, although only through optional add-ons in the case of Visual Basic, Java and ActiveX components (the Component Manager add-on will reverse engineer the COM interface).

A unique feature of Select Enterprise is Object Animator. This tool animates object diagrams, which is a particularly useful feature when verifying design correctness with users and managers.

Reverse engineering has a weakness with regard to Visual Basic, as only code modules and forms are handled. But classes, user control, and recent features such as Data Connection and Data Environment are not supported. This incapacity flies somewhat in the face of the tool's declared intention to support iterative component development, which in Visual Basic entails classes rather than global code modules.

Moreover Select Enterprise lacks overall integration in comparison to Rational Rose. The Visual Basic add-ons, like the C++ generator and the Component Manager, do not share a common interface with the main Select Enterprise user interface.

Strengths

- Excellent reporting facility
- Well-defined software development process
- Support for existing Booch, Rumbaugh and Jacobson methodologies

Weaknesses

- Poor association between diagrams and object model
- Component Manager and VB Code Generator must be purchased separately
- No support for automatic code synchronization. All source code must be reverse-engineered into separate models, and a difference report run manually
- No support for visual Basic classes

2.3.3 Rational Rose 98 Enterprise Edition[2]

Simply by virtue of its pedigree, Rational Rose 98 Enterprise Edition has to be a strong contender for anyone considering a UML modelling tool. The product, and indeed UML itself, is the work of the three leading methodologists - Grady Booch, James Rumbaugh and Ivar Jacobson - who are all now at Rational Software. Booch is probably best known for his cloud diagrams representing classes. Rumbaugh was behind the popular OMT, and Jacobson was the inventor of the widely adopted Object-Oriented Software Engineering (OOSE) as well as the use-ase technology that figures strongly in UML modeling tools.

[2] Rational Rose is a trademark or registered trademark of Rational Software Corp.

Rose also provides much-needed add-ins linking Rose with Oracle and the ERwin database modeling tool, so addressing Rose's deficiency with respect to database modeling and schema generation. Other versions of Rose are available that support Smalltalk, Powerbuilder, Ada and Forté.

Rose is first a developers' tool, and among its more significant strengths is its ability to tie code to the class diagrams. Specify a class and a tabbed dialog box appears giving the facility to define that class in more detail than any other tool reviewed here.

In general Rose 98 provides good code generation, especially when it comes to Visual Basic. Here it does support important features omitted from Select Enterprise, namely classes and global modules, which are represented as class utilities in UML. It also has its own Visual Basic-like scripting language, which permits almost endless customization of the shrink-wrapped product, and makes it possible to address many of the product's inherent weaknesses.

One drawback is the minimal dependency between components in the Rose repository. To a greater extent than Select Enterprise, Rose98 diagrams stand-alone and represent the generated code independent. As a result, project level sharing is not automatically supported unless elements of the software can be subsumed into distinct packages.

However, Rational Rose does integrate with Microsoft Repository, which will support flexible team programming capabilities. The Rose files work well with the source code management tool; however, this Rose's file-centric approach also makes it difficult for multiple developers to work on the same model because only one developer can check out a model file at a time. On the other hand, this approach does make it easier to implement difference tools for comparing different versions of a model.

Strengths

- A complete product, including code generators and database add-ins
- Extensibility through its own scripting engine
- Ease of use
- Oracle and Erwin integration to handles database definition
- Support for existing Booch, Rumbaugh and Jacobson methodologies

Weaknesses

- Errors and omissions in code-generation - although now largely resolved - detract from its role in 'round-trip' engineering
- Limited in-built reporting (but this can be rectified with scripts)

2.3.4 MagicDraw[3]

MagicDraw is a visual UML modelling and CASE tool designed for Business Analysts, Software Analysts, Programmers, QA Engineers, Documentation Writers, or Corporate Executives. The tool allows the developers or the business professionals to draw, design, and view UML diagrams of Object Oriented (OO) systems. Besides UML diagramming, it provides code-engineering mechanism, i.e. full round-trip support for Java, C++, and CORBA IDL programming languages. MagicDraw helps develop robust solutions and deal with the code complexity. It suits a variety of systems including real-time, client/server and distributed n-tier applications. The development tool is oriented towards single-user desktop development, and it integrates with Rational Rose 98 easily- it reads and writes Rose's model (mdl) files. Being written in Java and certified as "100% Pure Java Application", MagicDraw runs on a wide variety of operating systems, such as Windows 98/NT, Solaris, OS/2, Linux, HP-UX, AIX and others.

2.4 Related Research

Many researchers have studied the combination of DFD with objected – oriented methods that transform DFD to Code. A theoretic survey of many topics include:

In [4,8], the transformation of DFD to object-oriented diagrams is explained. Even if the diagrams are no longer available, it is expected to be easier to reverse engineer the program code into DFD. To obtain an object-oriented model of the system, starting from the DFD model, there are 2 approaches.

1. Convert the DFD model directly into an object diagram by transforming, on a one-to-one basis, processes and data stores in the DFD model into objects

[3] MagicDraw™ is a trademark or registered trademark of No Magic, Inc.

2. Follow a more object-oriented view where the focus is on classifying data in the system and detecting class methods that operate over that data

From DFD to objected-bases class diagram

In first approach [4,8] map the DFD model directly into a DFD-like object diagram, by transforming the artifact in the DFD model on a one-to-one basis. [4] Basically, the algorithm consists of transforming each data transformation in the DFD into an object in the object diagram, and each data flows between the data transformations into an association. In addition, the data flows involved in the system become internal attributes of the class (objects), are encapsulated into the objects, and are used now to describe the internal state of the objects. Corresponding methods are added in order to access the data entitled inside objects.

One should note that the way the elements in the object diagram are named is only to help in automating the transformation. Once the transformation process is completed, the names can be change to be more suggestive for the designer. Caution has to be taken that, by changing names, the consistency of the design is not affected. Thus, either the modeling the design tool should provide support for checking the consistency of the design after each transformation step.

After the transformation is completed, [4,8] focus on specifying the behavior of each object. In fact, defining the behavioral aspects of the system is the main purpose of this view. The final object model is very similar to the DFD one, but objects have internal behaviors and provide services (implemented by methods) to the adjacent objects. The newly created objects are classified as being active or reactive based on the process from which they are generated. The difference between them is that the execution of a reactive object is triggered only when of its method is invoked, while an active objects have a state machine (usually implemented by a run () method) executing continuously.

The transformation performs the following steps:

1. Each data transformation in the DFD is transformed into a new class/object in the object model
2. Flows between transformations in DFD into class association

3. The class originating from a process that receives input flows is assigned the set_() method and corresponding attributes
4. Active objects are assigned the run() methods
5. Objects receiving an input data-flow receive send_() method of incoming data flow
6. Each data store is transformed into a class/object, according with read and write methods corresponding to the input/output direction of the flows connected to the data store

From DFD to class diagram

In the second approach to create an object-oriented model of the system, [4] the transformation is based on the classification and encapsulation of data into classes, along with corresponding methods that operate over this data.

Briefly, the transformation is implemented by classifying all the processes, data flows, data stores in the DFD, based on their types. For each identified type in the DFD, a corresponding class is created in the class diagram. In order to add class methods, the transformation looks for three kinds of patterns in the data flow diagram: data flows between two processes (Figure 2.5), data flows communicating with data store elements (Figure 2.6), and data flows communicating with the external environment of the system (Figure 2.7)

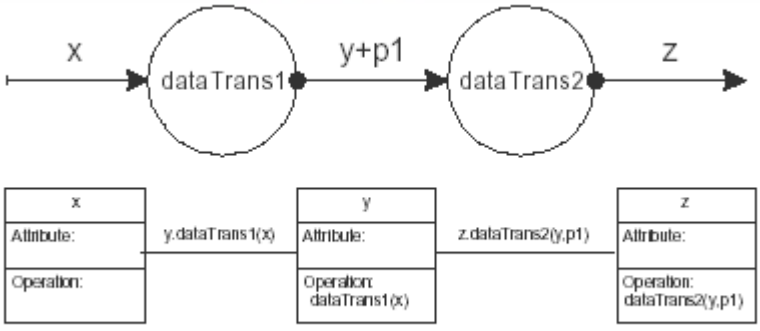


Figure 2.5 Inter-process Communication Pattern

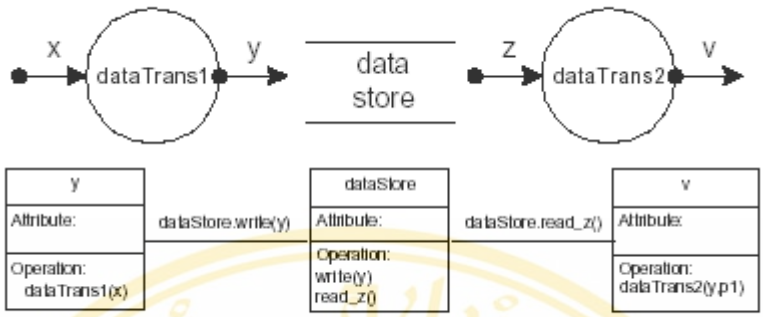


Figure 2.6 Data Store Communication Pattern

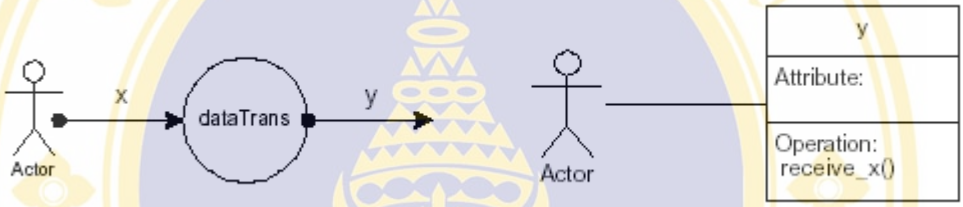


Figure 2.7 Border Process Pattern

The transformation perform the following step

1. Transform distinct data flow and data stores into class.
2. Transform each external entity into an actor
3. Apply the Inter-process Communication Pattern
4. Apply the Border Process Pattern
5. Apply the Data Store Communication Pattern

Alabiso's also proposes the transformation DFD into object [2]. In order to accomplish the transformation, four activities must be performed:

1. Interpret Data, Data processes, Data Stores and Terminals in terms of Object Oriented concept
2. Interpret the DFD hierarchy in terms of design decomposition- it is intuitively obvious that the DFD hierarchy and the decomposition of a complex design into design components must bear some relationship to one another
3. Interpret the significance of control processes for the design model
4. Use data decomposition information to help define object decomposition

In the Object modeling technique (OMT), DFDs can be used to describe the functional model of the system [7]. Furthermore, if the system is modeled using the object model or the dynamic model, the DFDs then specify the meaning of the operations in the object model and the action in the dynamic model. Although there is some attempt at integration, this correspondence is left completely vague and can not be analyzed in any useful way.

Another interesting technique is the transformation a complex DFD, which some processes have more than one output data flows, into programs [18]. The steps of this transformation are:

1. Define a class for each DFD.
2. All of the data stores occurring in the DFD are declared as instance variables (global variables) of the class.
3. Each process of the DFD is defined as a method of the class.
4. The data flow connections between processes can be implemented as variables or method calls in the class.

CHAPTER III
METHODS AND MATERIALS

This chapter describes methods and materials that would be used to design and develop the Java Code Generator from Data Flow Diagram

3.1 Research Methods

This thesis, the research methods are based on System Development Life Cycle (SDLC) that has a similar set of four fundamental phases: planning, analysis, design and implementation [1]. The diagram is shown in figure 3.1.

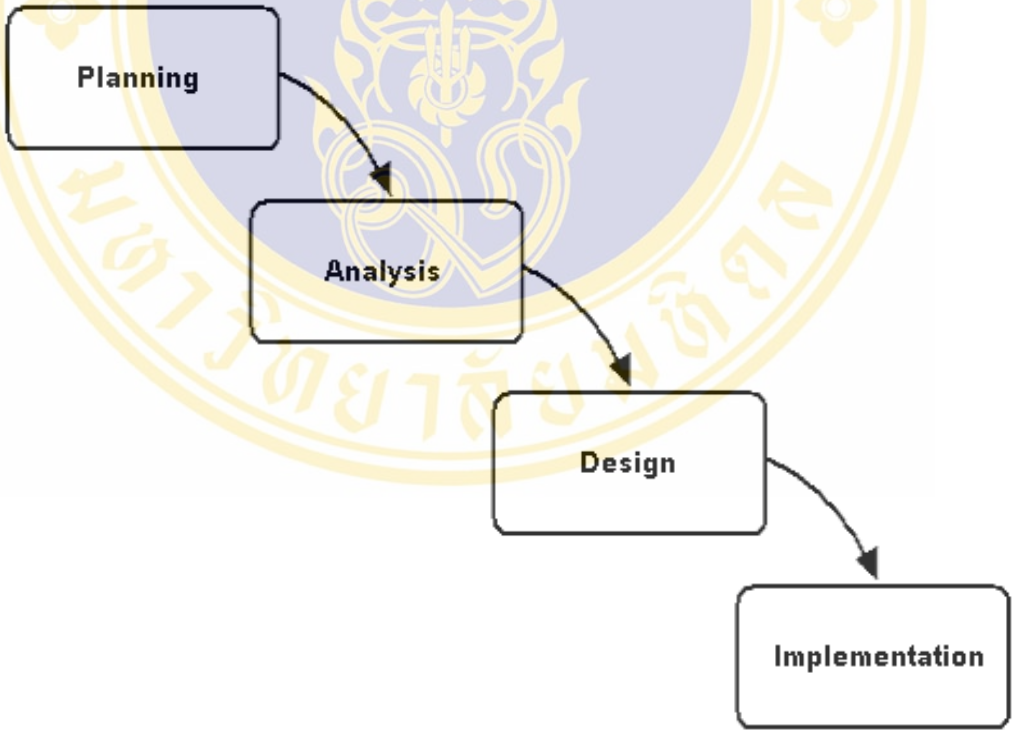


Figure 3.1 Steps of Research Method

3.1.1 Planning Phase

The planning phase is the fundamental process of understanding why a system should be built and determining how to build the project [1]. This phase identify the business value of the system by developing a system request that provide basic information about the proposed system, which is addressed in the chapter1.

3.1.2 Analysis Phase

System analysis is both a business task and an information technology task because its aim to produce a model of the system that is correct, realistic and unambiguous [3].

This analysis begins with answers the questions of who will use the system, what will their specific uses to be, where/when will the system be used [1,12]. The next step is information gathering. In this step, related information will be collected from many sources such as books, journals and electronic data and will be lead to develop idea for analysis use-case model and object model.

3.1.3 Design Phase

The design phase focuses on the specification of a detailed on computer-base solution [7]. In this research, The designing phase consists of two steps:

1. The interface design defines how the users will interact with the system and the nature of the inputs and the outputs that the system accepts and produces. The interface includes two fundamental parts:
    - The input interface : which collect data before it is enter in to the information system
    - The output design: which present information to users.
2. The program design explains how to code each module in detail.

3.1.4 Implement Phase

This is the last step in SDLC phase is the implement phase, during which the system actually built. The implement phase of this research has twos steps:

1. Programming will produce an artifact from the interface and the program specifications.
2. Testing consist of several tests to ensure that the program is functional according to the design. Unit testing examines individual modules or programs within the system; Integration testing examines how well

several modules work together and finally system testing examine the system as a whole.

3.2 Research materials

3.2.1 Hardware

- Microcomputer at least Pentium 500 MHz.
- Hard Disk 40 GB.
- RAM 256 MB.
- Necessary peripheral equipment

3.2.2 Software

- Operating system: Microsoft Window 2000 or Microsoft Window XP
- Borland JBuilder 6.0
- Rational Rose 2000 Enterprise Edition
- Adobe PhotoShop 8.0

3.3 Scheduling

The Java Code Generator from Data Flow Diagram							
Task List	Time (month)						
	1	2	3	4	5	6	7
1. System Analysis							
2. System Design							
2.1 The interface design							
2.2 The program design							
3. Implementation							
3.1 Programming							
3.2 Testing							
4. Conclusion and suggestion							

CHAPTER IV
RESULTS

The objective of this research is to develop A Java-Application Code Generator from Data Flow Diagram, which the results of this application are described as follow.

4.1 Analysis Phase

Form the analysis phase, the questions about who will use the system, what their specific uses will be, and where/when the system was answered as below:

Who will use the system? : Any member of a software development team or an individual who design the program by DFD. Beginning with the system designers wishing to design a system, they can use this to create DFD, which in turn can be passed onto the developer/programmer to be implemented in code.

What will their specific uses be? : There will be two types of users, which can be categorized as follows:

- System designers: The system designers will use this application as a tool to draw DFD and check the correctness of it.
- System developers/programmers: The developers/programmers will get the most use out of this tool. They will be able to generate skeleton code that can be used as a foundation in the implementation.

Where/When will the system be used? : The tool will be used during both the design and implementation phase of a project. The tool will act as a bridge between the two phases.

The next step of the analysis phase is information gathering to select techniques, which able to support the above answers, for design and implement phase.

Firstly, selection techniques to generate code from [4,8,18] research represented in chapter two that explain three methods “form DFD to objected-bases class diagram”, “form DFD class diagram” and “form complex DFD to programs”. The first and last method, the classification all the processes, data flows, data store in

the DFD are avoided. So, benefit of data flow flavor, which is good at describing the flow of data among pieces of functionality, still remain and the object model very similar in DFD. It forces the designer to uses DFD and thus forces the back-end tools to support both DFD and UML. While the second method, assuming that generically DFDs are not an adequate tool for capturing the user's requirements [11], the classification and encapsulation of data into class can be carry on to reduce the system complexity, following the top-down approach.

Both methods are used in the deference purpose. [8] does not believe that it is possible to rely on a "one-size-fits-all" due to the wide range of application covered by software field. This means that in some situations DFD may be adequate model of computation, but that in others UML may be better.

For this research aim to create the code generator for the system designer and system developer/programmer, who design a program by DFD, so the code should to support the DFD as well. Because of this reason the first and last method are selected for the transformation DFD to code.

Secondly, selection a programming language for code generator by considering where the DFD of computation is evidently useful and widely adopted, next choosing a programming language from that application area.

In the view of Gajski and Vahid, a DFD may be most adequate one of transformational system. It is the system that continuously repeat the same data transformation on streams of data for example telecommunication device [Helm and Wess, [1996] and digital signal-processing systems [Benini et. al., 2000], which a Java programming language is ideal for those system application developers. Because of the Java platform provides an application environment that specifically addresses the needs of commodities in the vast and rapidly growing consumer and embedded space, including mobile phones, pagers and personal digital assistants [17]

After answer the questions and select technique for the next phase, the analysis use case model was constructed by performing following steps:

1. Identify and document new actors. The above answers were used to identify actors which a textual definition of that actors according to the users' perspective and using their terms are shown in table 4.1

Table 4.1 Actors of A Java-Application Code Generator from DFD

Term	Synonym	Description
1. System designer		An individual or any member of a software development team who use this application as a tool to draw data flow diagram and check the correctness of it.
2. System developer	System programmer	An individual or any member of a software development team who use this application as a tool to produce skeleton code

2. Identify and document new use cases. The new actor, System designer and system developer, discovered in 1 leads to a new interaction with the system – thus a new use case. In this research, the list of use cases are defined in uses case actor glossary as follows:

Table 4.2 Use cases of A Java-Application Code Generator from DFD

Use case name	Use case description	Participating Actor
Create a workspace	This use case describes the event of creating a workspace by opening the application. On completion a workspace is generated.	- System designer - System designer
Create a drawing canvas	This use case describes the event of creating drawing canvas to draw a data flow diagram by receive information about the data flow diagram from system designer. In this use case, an empty canvas is produced. The canvas is now prom to create data flow diagrams.	- System designer

Table 4.2 Use cases of A Java-Application Code Generator from DFD (Continued)

Use case name	Use case description	Participating Actor
Create a data flow diagram	This use case describes the event of creating a data flow diagram In this use case, a system designer able to insert, delete and update all the relevant features of the data flow diagram. On completion the data flow diagram has been successfully created.	- System designer
Generate Java code	This use case describes the event of code generation from a data flow diagram. On completion, the generated Java code is presented to a system developer.	- System developer
Check illegal data flow diagram	This use case describes the event of ensuring about the correctness of a data flow diagram. On failure, errors are reported and possible solutions are given.	- System designer

3. Construct use case model diagram. Once the use-cases and actors have been identified, a use case model are graphically depicted the system scope and relationship. Below figure (figure 4.1), the Rational Rose 2000 Software are use to create use-case model, which has three relationship line including:
    - Association line contains an arrowhead on the end touch use case to indicate an interaction between an actor and a use-case.
    - Extend line is represented as arrowheaded line beginning at the extension use-case and pointing to the use case it is extending. Each extend line is labeled “<<extends>>”. For the purpose of simplifying the complex use case and making it more easily understand.

- Depend on line is depicted as an arrowheaded line beginning at the one use case and pointing to a use case it is dependent on. Each extend line is labeled "<<depends on>>". In order to determine the sequence in which use-case need to be developed.

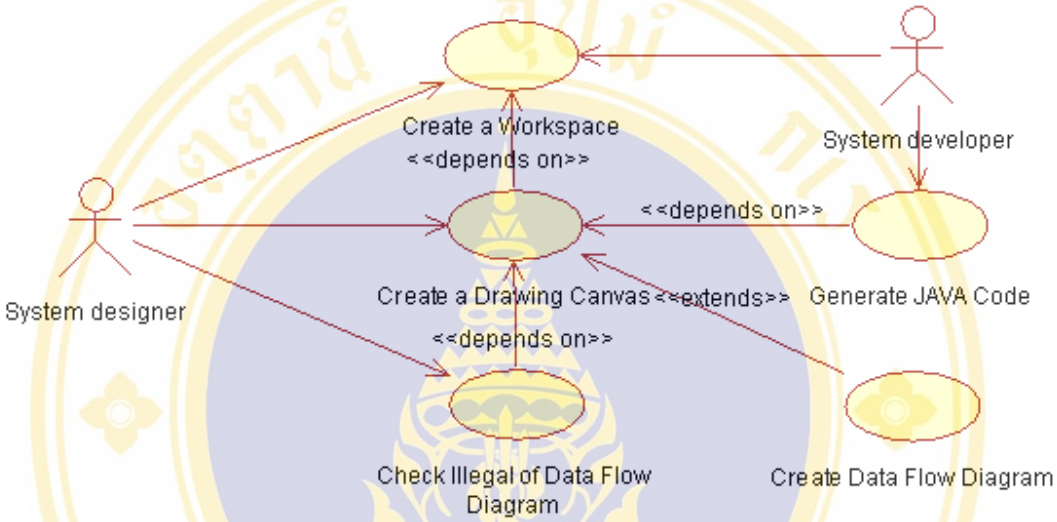


Figure 4.1 The Java-Application Code Generator use case model diagram

4. Document analysis use-case narratives. The narrative document describes the sequence of events of an actor (an external event) using a system to complete a process - [Jacobson92]. A Use Case was described how a system will be used, which the list of narrative documents in appendix A. When the analysis use case model has been finished, nouns of this used case were underlined. Those nouns were selected the essential noun to identify objects for object modeling analysis. In this research able to list the important objects from the use cases including: Code generator, Code generation, Canvas link, Data flow, Data store, Drawing canvas, Entity, Illegal data flow diagram, Line, Process, Rectangle, Status, Symbol link, Symbol of data flow diagram and Workspace.

For the object modeling analysis, those objects were organized and documented any major conceptual relationships between the objects. A class diagram (figure 4.2) was used to graphically depict the objects and their associations. On this diagram compose the associations as follow:

- Association relationship is the most common relationship in a class diagram. The association shows the relationship between instances of classes by arrowhead line.
- Aggregation relationship is used when classes actually comprise other classes. It is often referred to as a whole/path relationship. A solid diamond is placed nearest the class representing the aggregation and lines are drawn from the arrow to connect the classes that serve as its parts.
- Generalization relationship shows that on class (subclass) inherits from another class. The generalization path is shoe with a solid line from the subclass to the superclass and a hollow arrow pointing at the superclass.

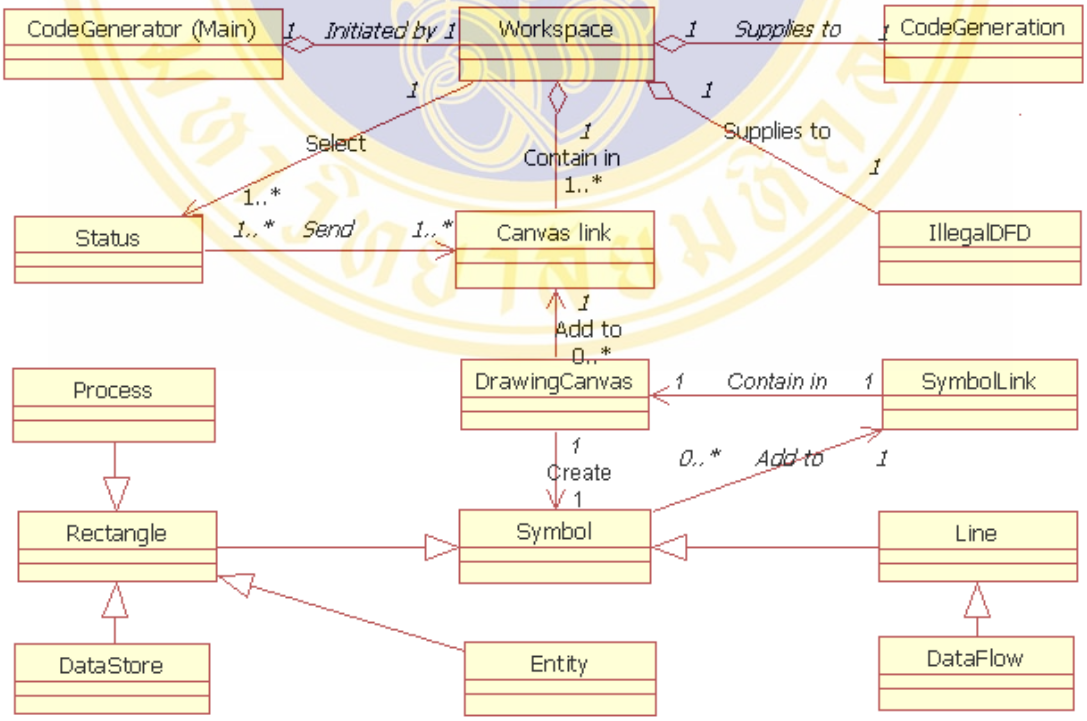


Figure 4.2 The Java-Application Code Generator analysis class diagram

4.2 Design Phase

The design of A Java-Application Code Generator from Data Flow Diagram is divided in interface design and program design. The interface design deals with the exchange information between users and the system whereas the program design deal with the patterns and methods used to implement this code generator.

4.2.1 The Interface Design

A Java Code Generator from Data Flow Diagram has been designed the input and output interface.

1. Input Interface

The design of input Interface is used to enter data into the program. When user opens the application initially they are faced with a blank canvas on which they can draw data flow diagram. The window that the user will illustrated in figure 4.3

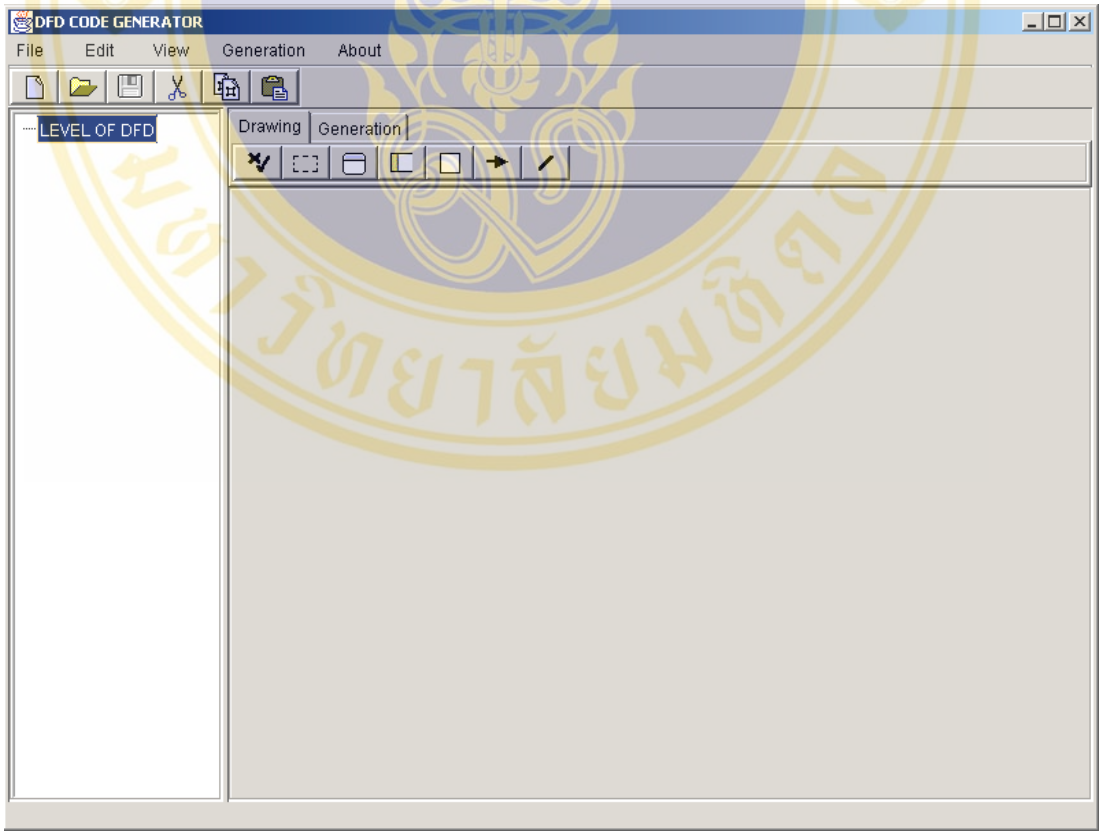


Figure 4.3 The input interface

2. Output Interface

The design of output interface is used to present information to user. In this research can classify two characteristics the data flow diagram output and the Java code output.

- The data flow diagram output is shown in figure 4.4. It uses the Gane and Sarson symbol to illustrate the data flow diagram.

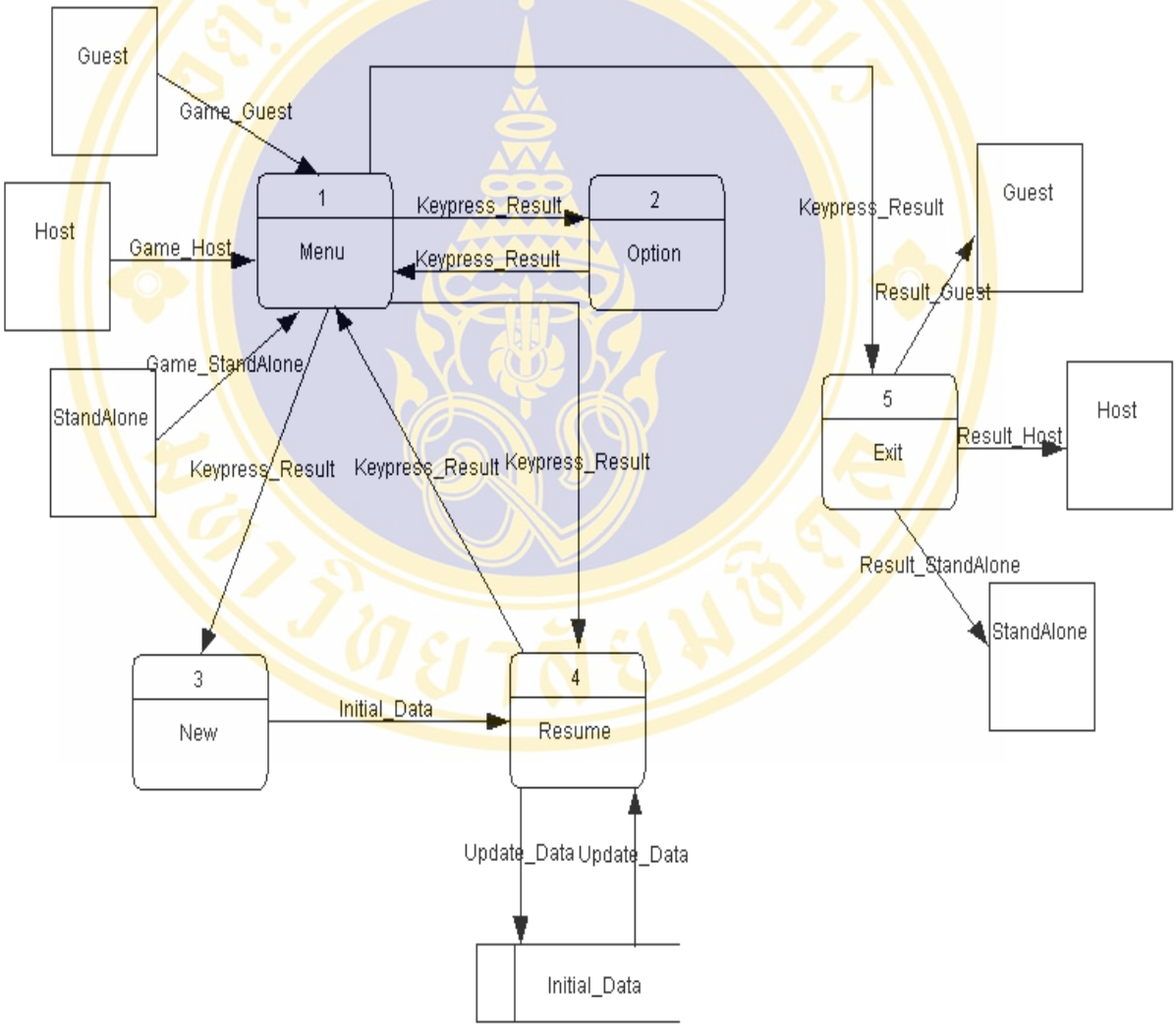


Figure 4.4 The data flow diagram output

- The Java code output is displayed in figure 4.5 and 4.6. The first figure is the output of Java code, which processes are transformed to methods. The steps of this transformation are:
    1. Define a class for each DFD.
    2. The data flow connections between processes can is implemented as variables.
    3. All of the data stores occurring in the DFD are declared as global variables of the class.
    4. Each process of the DFD is defined as a method of the class.

  The second figure is the output of Java code, which processes are transformed to classes. The steps of this transformation are:
    1. Each process of the DFD is defined as a class.
    2. All of input data flows and output data flows are declared as set and send methods of the class, respectively.
    3. Each data store is transformed into a class, according with read and write methods corresponding to the input/output direction of the flows connected to the data store

```
public class ClassName() {
        //SET : TypeName TypeName_1  = new  TypeName();
        //SET : DBName DBName_1 = new = DBName();
        //Main method
        public static void main(String[] args) {
        }
        public void MethodName() {
                Menu  Menu_1 = new Menu();
                //CAN USE THE METHOD :ClassName.MethodName();
                //GET  : TypeName TypeName_1
                //SEND : TypeName TypeName_1
        }

}
```

Figure 4.5 The Java code output – processes transform to method

```
public class ClassName() {
        //SET INPUT TYPE : setMethod (TypeName  Name) { }
        public void setMethod() {
        }

        //SET RETURN TYPE : TypeName  sendMethod { }
        public void sendMethod() {
        }

}

public class DataStoreName() {
        //SET INPUT TYPE : getMethod (TypeName  Name) { }
        public void writeMethod() {
        }

        //SET RETURN TYPE : TypeName  sendMethod { }
        public void readMethod() {
        }

}
```

Figure 4.6 The Java code output – processes transform to classes

4.2.2 The Program Design

The next step of design phase is the program design explains how to code each module in detail. In this program design includes the following activities:

1. Refining the use case to reflect and implement environment. The use cases will be refined to include details of how the actor will actually interface with the system and how the system will respond to that stimulus to process event. Appendix B illustrates the refinement of the use cases that was originally defined during the system analysis.
2. Modeling object behaviors and responsibility that support the use-case scenario. In this activity is examined all verb phases in the use case to identify all behaviors that can be associated with an object type and to identify all collaboration among that object. The document of the behaviors and collaboration of an object in this research are displayed by class responsibility collaboration (CRC) card as follow.

Table 4.3 The class responsibility collaboration (CRC) card of Data flow

Object name: Data flow	
Super Object: Line	
Behavior and Responsibility	**Collaborators**
Copy the selected symbol	Line
Draw data flow	
Paint the new symbol	

Table 4.4 The class responsibility collaboration (CRC) card of Data store

Object name: Data store	
Super Object: Rectangle	
Behavior and Responsibility	**Collaborators**
Copy the selected symbol	Rectangle
Draw data store	
Draw text	
Paint the new symbol	

Table 4.5 The class responsibility collaboration (CRC) card of Drawing canvas

Object name: Drawing canvas	
Super Object:	
Behavior and Responsibility	**Collaborators**
Drag Mouse	Symbol link
Release Mouse	Symbol
Press Mouse	
Paint	

Table 4.6 The class responsibility collaboration (CRC) card of Code generator

Object name: Code generator	
Super Object:	
Behavior and Responsibility	**Collaborators**
Call the new workspace form (main)	Workspace

Table 4.7 The class responsibility collaboration (CRC) card of Code generation

Object name: Code generation	
Super Object:	
Behavior and Responsibility	**Collaborators**
Transform process to methods	Workspace
Transform process to classes	

Table 4.8 The class responsibility collaboration (CRC) card of Canvas link

Object name: Canvas link	
Super Object: Link	
Behavior and Responsibility	**Collaborators**
Link canvas	Canvas
Clear link	

Table 4.9 The class responsibility collaboration (CRC) card of Entity

Object name: Entity	
Super Object: Rectangle	
Behavior and Responsibility	**Collaborators**
Copy the selected symbol	Rectangle
Draw entity	
Draw text	
Paint the new symbol	

Table 4.10 The class responsibility collaboration (CRC) card of Illegal DFD

Object name: Illegal data flow diagram	
Super Object:	
Behavior and Responsibility	**Collaborators**
Checks illegal data flow diagram.	Canvas
	Symbol of data flow diagram

Table 4.11 The class responsibility collaboration (CRC) card of Line

Object name: Line	
Super Object: Symbol of data flow diagram	
Behavior and Responsibility	**Collaborators**
Copy the selected symbol	Symbol
Draw line	
Draw text	
Paint the new symbol	

Table 4.12 The class responsibility collaboration (CRC) card of Process

Object name: Process	
Super Object: Rectangle	
Behavior and Responsibility	**Collaborators**
Copy the selected symbol	Rectangle
Draw process	
Draw text	
Paint the new symbol	

Table 4.13 The class responsibility collaboration (CRC) card of Rectangle

Object name: Rectangle	
Super Object: Symbol of data flow diagram	
Behavior and Responsibility	**Collaborators**
Copy the selected symbol	Symbol
Paint the new symbol	

Table 4.14 The class responsibility collaboration (CRC) card of Status

Object name: Status	
Super Object:	
Behavior and Responsibility	**Collaborators**
Get status	Workspace
Send status	Drawing canvas

Table 4.15 The class responsibility collaboration (CRC) card of Symbol link

Object name: Symbol link	
Super Object: Link	
Behavior and Responsibility	**Collaborators**
Link the symbol	Symbol
Unlink the symbol	
Clear link	

Table 4.16 The class responsibility collaboration (CRC) card of Symbol of DFD

Object name: Symbol of data flow diagram	
Super Object:	
Behavior and Responsibility	**Collaborators**
Add text	Symbol link
Checks the selected coordinate in the bound of a symbol.	
Paint the new symbol	
Select the symbol	
Move the selected symbol	

Table 4.17 The class responsibility collaboration (CRC) card of Workspace

Object name: Workspace	
Super Object:	
Behavior and Responsibility	**Collaborators**
Cut the selected symbol	Canvas link
Copy the selected symbol	Illegal data flow diagram
Paste the selected symbol	Code generation
Create a new canvas	
Display the DFD error position.	
Display the corresponding Java code	
Opens a file	
Save a file	

After determined the object behaviors and responsibilities, it was used to create a detailed model of how the object will interact with each other to provide the functionality specified in each design use case. In this research used sequence diagrams (figure 4.7, 4.8, 4.9, 4.10, and 4.11) to graphically depict the interaction of objects.

Figure 4.7 The sequence diagram of “Create a workspace” use case

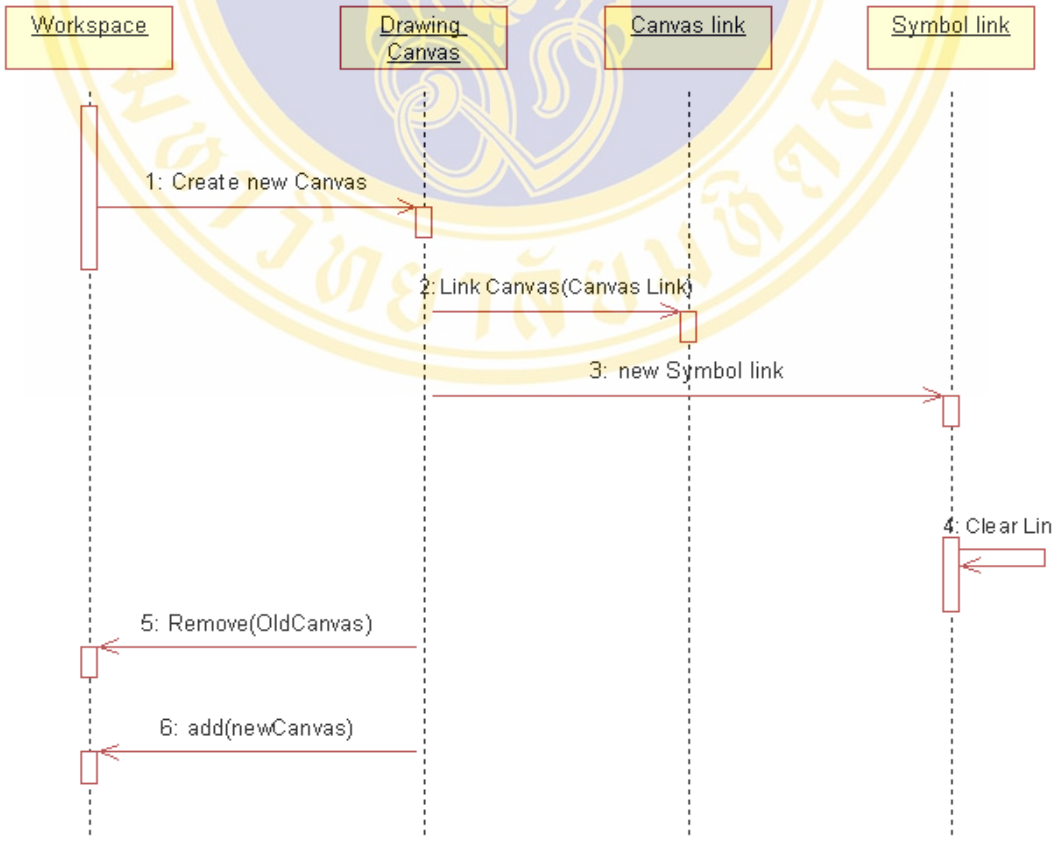


Figure 4.8 The sequence diagram of “Create a drawing canvas” use case

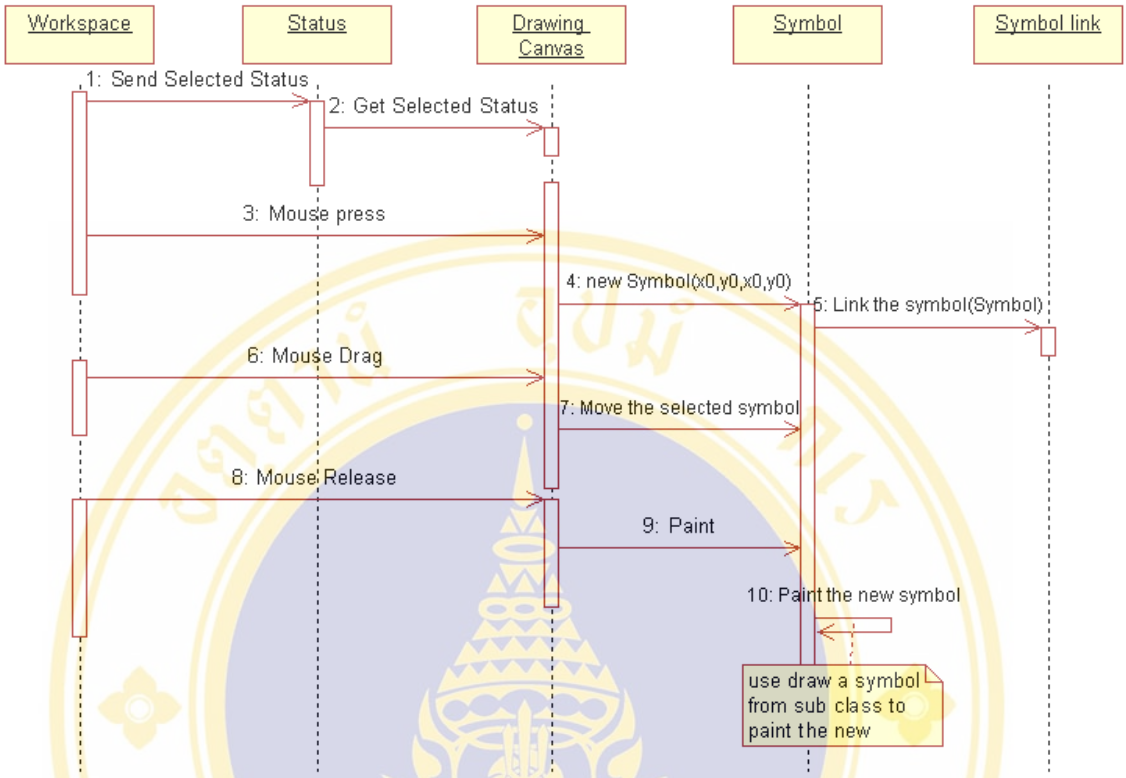


Figure 4.9 The sequence diagram of “Create a data flow diagram” use case

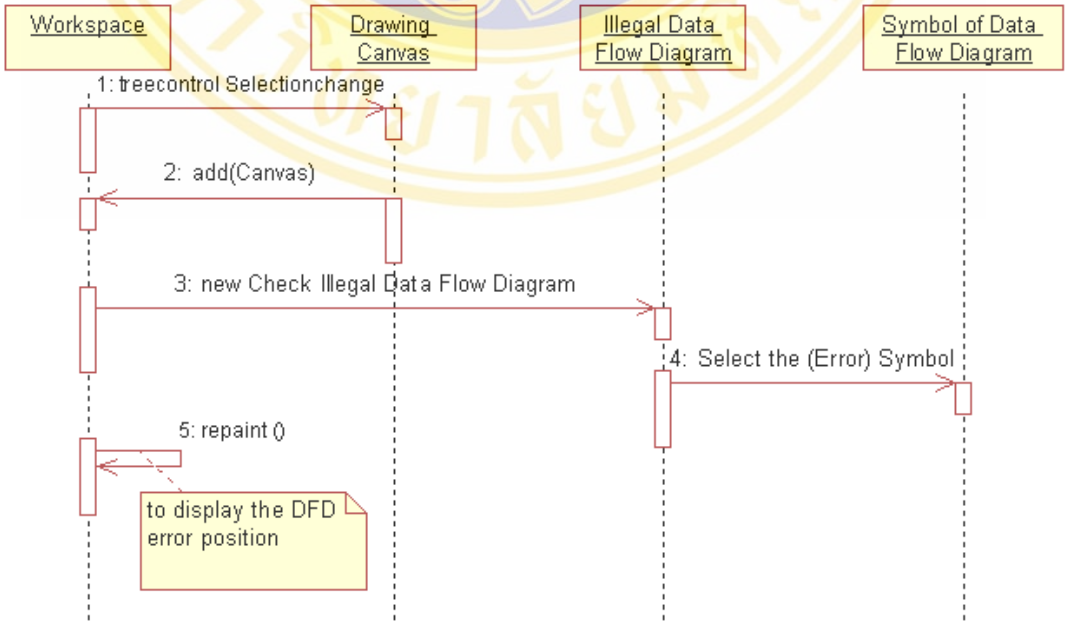


Figure 4.10 The sequence diagram of “Check illegal data flow diagram ” use case

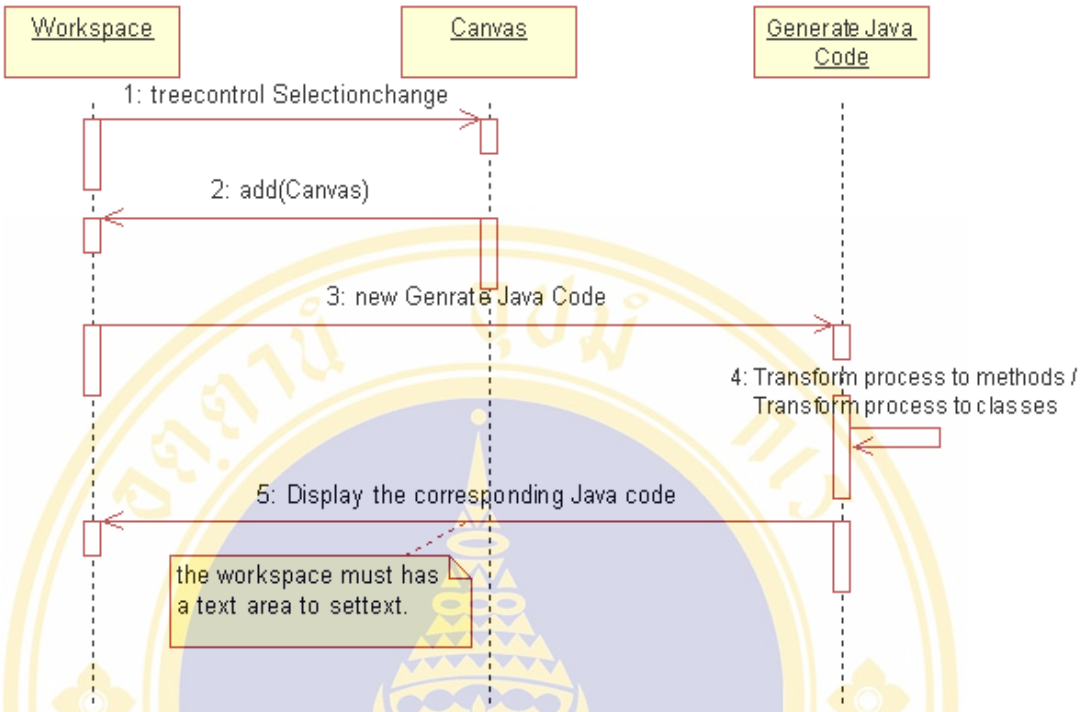


Figure 4.11 The sequence diagram of “Generate Java code” use case

3. Updating the object model to reflect the implementation environment. The following steps are used to transform the class diagram prepared in OOA to a design class diagram.
   - Add attribute and attribute type information to design objects: Examine all nouns in the refined use case to identify attributes and attribute types for each class of A Java-Application Code Generator from Data Flow Diagram.
   - Add attribute visibility: Attributes can be defined as public, protected or private.
   - Add method to design objects: Define the method to implement any previously identified responsibility and behavior.
   - Add method visibility: Methods can be defined as public, protected or private.

   Figure 4.12 is a view of the Java-Application Code Generator design class diagram.

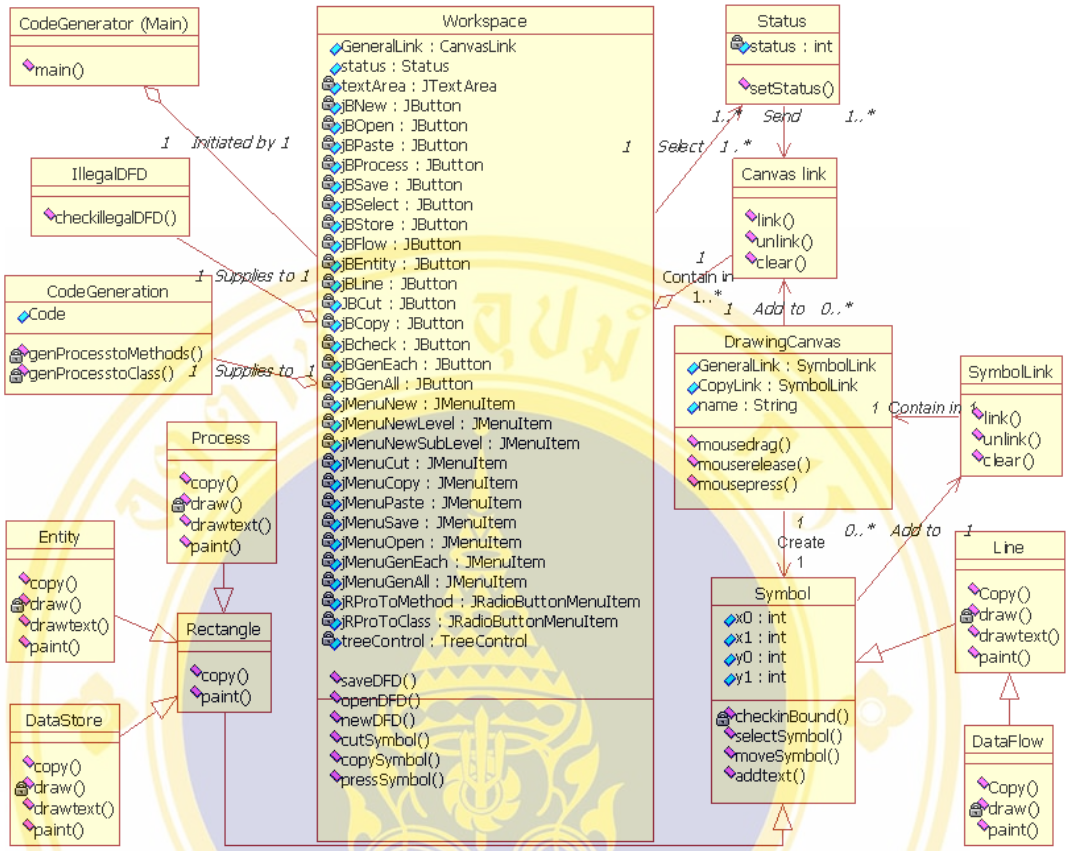


Figure 4.12 The Java-Application Code Generator design class diagram

4.3 Implement Phase

In this phase class diagram that created in analysis and design phase was transformed to Java code. After that it was brought to develop the program meanwhile, it needs to test the program. The Implement Phase consist of there general state of tests:

1. Unit test: This test uses the white-box testing to focus on a program module by looking inside the unit to review the code. The typically defects found in unit test of this thesis include problems with loop termination, simple internal parameter passing, and proper assignment statements. When unit testing was done, all errors are fixed. It verifies that when parameters are passed between routines, the pieces communicate correctly and generate appropriate outputs.

2. Integration test: This test examines how well several modules work together by using scenario testing that done by create test cases. Appendix C illustrates the test cases names of each use case, which this program was passed the test, which testing result verifies that several modules work correctly.
3. System test: This test tests how convenient the system is to use. This research used the mobile phone game – "Plane"[19], supporting J2ME platform to test the Java Code Generator. The DFDs of this game was drawn and generated, using the process specification to decision processes should be transformed to class or method. After that computed the percent of match, not match and missing class or method from generated code. The result of match, not match and missing class or method in this test depicted in the appendix D.

4.4 Evaluation of the Program

This research uses the basic steps for evaluate programs of [21], which the including:

4.4.1 Identify program for comparing

The first step find out the program for comparing, which the DFD Editor was selected to evaluate A Java-Application Code Generator from Data Flow Diagram.

4.4.2 Read existing reviews

From the existing reviews give a DFD Editor detail that developed as part of a project under Dr. Nabendu Chaki at the Department of Computer Science & Engineering, University of Calcutta [20].

It helps the user to draw, edit and validate their DFDs. Users can draw multi-level DFDs and check their flow balance. The tool is extremely easy to use and the user can work with all the basic DFD symbols. The users just have to select them and place them on the drawing canvas. The DFDs can also be saved for future alterations. Besides it can also be used to model software system architecture and for automatic code binding.

4.4.3 Briefly compare the leading programs' function

Once identified the program and read the reviews, in this set begin to briefly the comparing between this Code Generator and DFD Editor. The results of the comparison are shown in table 4.26

Table 4.18 The comparing results

Function	The Java-Application Code Generator	DFD Editor
Cut Symbol	Click on the element to be deleted and click “Cut” button or “Edit/Cut” drop-down menu	Click on the element to be deleted and press “Delete” key from keyboard to delete the element
Copy Symbol	Click on the element to be deleted and click “Copy” button or “Edit/Copy” drop-down menu	Not support this function
Paste Symbol	Click on the element to be deleted and click “Paste” button or “Edit/Paste” drop-down menu	Not support this function
Draw Symbol	Use the Gane and Sarson symbol to illustrate the DFD	Use the DeMarca and Yourdan Symbol to display the DFD
Move symbol	Click on the element to be move and drag mouse to move the element to any position on canvas	Click "Hands Free" button, Select the element to be moved, and drag mouse to change the element position.
Undo any operator	Not support this function	Click “Undo” button to undo other previous operations starting from the last one.

Table 4.18 The comparing results (Continued)

Function	The Java-Application Code Generator	DFD Editor
Save DFD	Click “Save” button on the main toolbar or “File/Save” drop-down menu. Enter a file name into a dialog for identify the saving location The file size of the mobile phone game – “Plane” DFD, drawn by this program is 4.79 KB (see appendix D)	Click “Save” button on the toolbar and Enter a file name into a dialog for identifies the saving location. The file size of the mobile phone game – “Plane” DFD, drawn by this program is 17. 3 KB
Open DFD	Click “Open” button on the main toolbar or “File/Open” drop-down menu. Enter a file name into a dialog for open DFD	Click “Load” button on the main toolbar or “File/Load” drop-down menu. Enter a file name into a dialog for open DFD
Print DFD	Not support this function	Click “Print” button on the main toolbar or “File/Print” drop-down menu to print the DFD shown the canvas
Zoom and Shrink DFD	Not support this function	Click “Zoom in” or “Zoom out” button on the toolbar to zoom and shrink the DFD on canvas respectively

Generate code	Generate Java language by using two technique:
	1. Processes are transformed to Methods
	2. Processes are transformed to Classes
		Generate c language file that code is displayed as follow
		#include "common.h"
		Int main(void) {
		Return 0;
		}

Table 4.18 The comparing results (Continued)

Function	The Java-Application Code Generator	DFD Editor
Validate DFD	Click “Check” button on the draw toolbar to check common illegal DFDs that consist of: 1. A data flow is send from a entity to a entity 2. A data flow is send from a entity to a data stores 3. A data flow is send from a data stores to a entity 4. A data flow is send from a data stores to a data stores	Click “Validation” to see unbalanced flows shown by coloring them red. For the common illegal DFDs. In this program not be allow to draw them.

4.4.4 Perform an analysis

This step, the comparing results are used to evaluate the Java-Application Code Generator on three attributes as follow:

1. Usability: Usability measures the quality of the user friendly interface for its intended user. To be user friendly, software must be [22]:
   - Tolerant of mistakes: The application should allow them to Undo
   - Feedback-rich: The application should always give immediate feedback to the user regarding which actions(s) are being taken.

   Which both of applications have the feedback-rich ability. While only the DFD Editor has the tolerant of mistake ability because the Java-Application Code Generator lacks the Undo any operator function

2. Scalability: In this attribute concern with file size when a user save a data flow diagram. From the comparing results, the scalability of this Code Generator is at a satisfactory level.
3. Functionality: The Java-Application Code Generator specific functions need depend on the scope of this thesis that is addressed in chapter I. However, it needs some functions of the DFD Editor to enhance the efficiency of the program that are the function to check unbalanced flows and to print the output.

CHAPTER V
DISCUSSION

This research uses Object-Oriented technology in Object-Oriented Analysis (OOA) and Object-Oriented Design (OOD) to design and develop A Java-Application Code Generator from Data Flow Diagram. Both OOA and OOD in this research concentrate on use cases regarding to clearly understand the system requirement before it is used to analyze and design class diagram. When finished the OOA and OOD, the class diagram is transformed into a code for implementation, which need three testing to ensure that the program corrects including:

1. Unit test: This test tests by looking inside the unit to review the code. In this step, all unit-errors are fixed.
2. This program tests by following the test case in Appendix C, which testing result verifies that several-modules work correctly.
3. System test: This test tests how convenient the system is to use, which the results of this test indicate that the program is a satisfactory.

The result of this program is a prototype of a Java-Application Code Generator from Data Flow Diagrams, which use as guidelines to develop code generator for other programming languages from the DFDs. The three main modules of this Java code generator consist of:

- Drawing Data Flow Diagram: this module is used by a system designer to insert, update, delete and save symbols of DFD.
- Check illegal Data Flow Diagram: this module is used by a system designer to check common illegal DFDs that consist of:
    1. A data flow is send from a entity to a entity
    2. A data flow is send from a entity to a data stores
    3. A data flow is send from a data stores to a entity
    4. A data flow is send from a data stores to a data stores

- Generating skeleton code of Java: this module is used by a system developer/programmer in order to transform the DFDs to skeleton code of Java.

5.1 Advantage Features

1. The program can generate DFD to Java code: The ability to automate skeleton code helps to reduce the time scale and manpower required for implementation

2. The program is support structure programming: This tool creates a DFD from higher-level DFDs to lower-level DFDs that is the top-down strategy, while a user implement a program from lower-level DFDs to higher-level DFDs that is the bottom-up strategy.

5.2 Excluding Features

1. The program not check levels balanced flows: The program ought to check balanced flows by every element on the higher-level DFDs must appear on lower-level DFDs.

2. The program can not add attribute and method visibility: The attribute and method should able to identify visibility as public, protected, or private.

3. The program limit reporting facilities: This program lacks the function to print a code and DFD.

4. The program is not automatic code synchronization: DFDs should be modified when the codes change.

CHAPTER VI
CONCLUSION

The last chapter describes the conclusion and the recommendation from the study.

6.1 Conclusion

A Java-Application Code Generator from Data Flow Diagram is designed for system designer and system developer. This tool facilitates analysis and design DFD systems. It provides the code engineering mechanism support for Java-application programming languages

A Java-Application Code Generator from Data Flow Diagram has been developed under the object-objected technique. The Rational Rose 2000 Enterprise Edition was used to create all models in analysis and design phase in these phases created three diagram including:

1. The use case diagram presents a collection of use cases and actors and is typically used to specify the attribute, functionality and behavior of a whole application system interacting with external actors.

An actor models a kind of object outside the domain of the system itself that interacts directly with the system. In this Research, system designers and system developers who interact with the Java Code Generator from Data Flow Diagram are the actors.

A use case contains all the events that can occur between an actor and the system. Use cases of this research include Create a workspace, Create a drawing canvas, Create a data flow diagram, Generate Java code, and Check illegal data flow diagram. These contain a set of scenarios that explain various sequences of interaction within the transaction.

Shown in the diagram below is the use case diagram for A Java Code Generator from Data Flow Diagram.

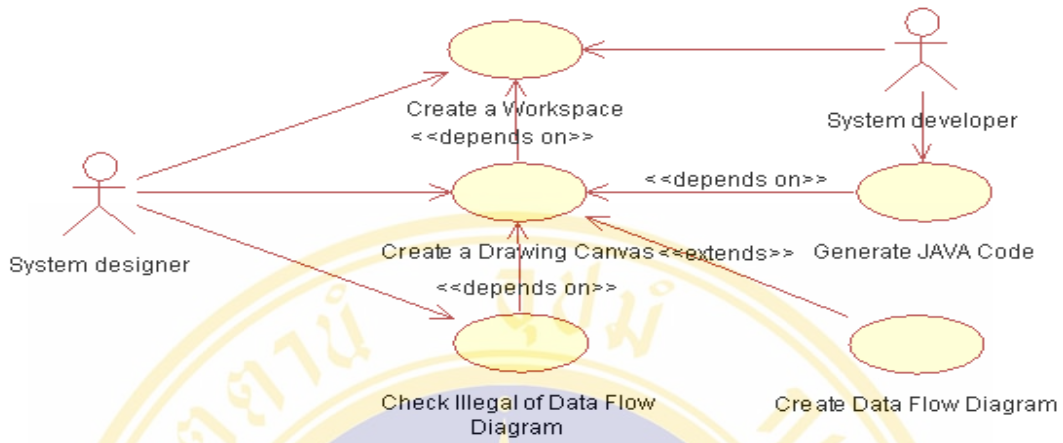


Figure 6.1 The Java-Application Code Generator use case model diagram

2. The sequence diagrams illustrate the operations that an actors requests of a system. These diagrams are obtained from the functionality and behavior of the use cases.

3. The class diagram illustrates meaningful concepts in a problem domain. The technique to specify a class is choosing nouns that are mentioned in each of the expanded essential use cases that was outlined earlier.

In this research has fifteen classes, which are selected from the important noun in the use cases including: Code generator, Code generation, Canvas link, Data flow, Data store, Drawing canvas, Entity, Illegal data flow diagram, Line, Process, Rectangle, Status, Symbol link, Symbol of data flow diagram, and Workspace.

The diagram is also important to identify these concepts association, which shows the relationship between instances of classes. Two special associations, aggregation and generalization, are used when classes comprise other classes or when one subclass inherits properties and behaviors from a superclass, respectively.

After that, the use cases will be refined to include details of how the actor will actually interface with the system and how the system will respond to that stimulus to process event. The refined use cases are use to examine the attribute and method of this code generator.

The entire above is shown in the class diagram below of A Java Code Generator from Data Flow Diagram.

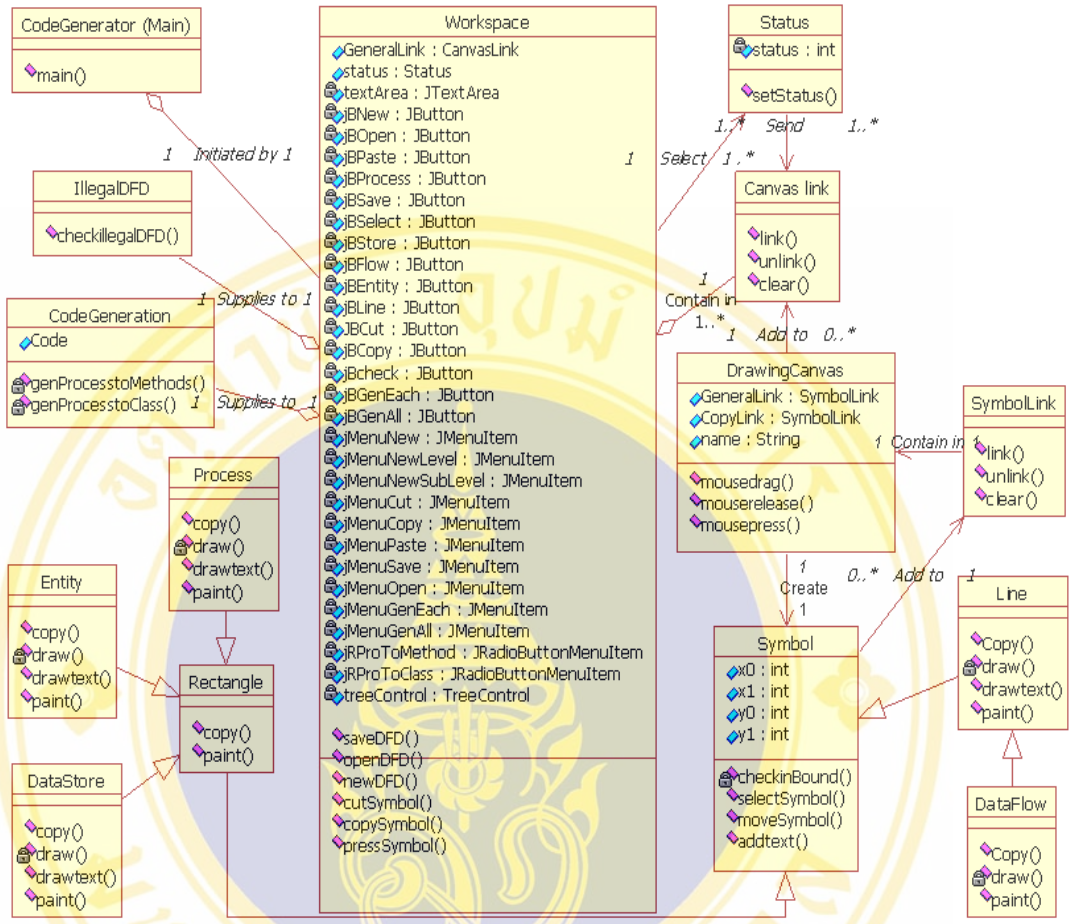


Figure 6.2 The Java-Application Code Generator class diagram

When finished the analysis and design, the class diagram is transformed into a code for implementation. After that bring the code to develop the program meanwhile, it needs to test the program in three parts that are unit test, integration test and system test, which the result of these test indicate the program is at a satisfactory level.

The outcome of this research is a prototype of a Java-Application Code Generator from Data Flow Diagrams, which has three main actives:

1. Drawing data flow diagram: A user able to insert, update and delete all the relevant features of a data flow diagram.

2. Check illegal data flow diagram: A user utilizes this activity to ensure about the correctness of a data flow diagram. On failure, errors are reported.

3. Generating skeleton code of Java application from data flow diagram: The generator transforms a data flow diagram to a Java-code by using two techniques:

The first technique, processes are transformed to methods, which the steps of this transformation are:

1. Define a class for each DFD.
2. The data flow connections between processes can is implemented as variables.
3. All of the data stores occurring in the DFD are declared as global variables of the class.
4. Each process of the DFD is defined as a method of the class.

The second technique, processes are transformed to classes. The steps of this transformation are:

1 Each process of the DFD is defined as a class.

2 All of input data flows and output data flows are declared as set and send methods of the class, respectively.

3 Each data store is transformed into a class, according with read and write methods corresponding to the input/output direction of the flows connected to the data store

6.2 Recommendation

To increase the usefulness of the Java-Application Code Generator from Data Flow Diagram, the following recommendation will guide for future development.

1. Check levels balanced flows: the program should verify the correctness of balanced flows by every element on the higher-level DFDs must appear on lower-level DFDs.
2. Identify attribute and method visibility- public, protected or private: the program should has the function for user to specify an attribute and method visibility.
3. Generate code from other programming languages: this program will be flexible if this program able to generate code form other language

4. Undo and redo any operation: Users are human - they make mistakes. The application should allow them to undo and redo any operation in order to enhance the effective of mistake tolerant of this program.
5. Print outputs: The output of this program consists of the data flow diagram output and Java code output.

REFERENCES

1. Alan Dennis, Barbara Haley wixom. System analysis and design. John Wiley & sons, Inc; 2000.
2. Bruno Alabiso. Transformation of Data Flow Model to Object-Oriented Design. In OOPSA'88, page 335-53.ACM press; 1988.
3. Berno Bruegge and Allen H Dutoit. Object-Oriented Software Engineering conquering Complex and Changing System. Prentice Hall , Inc; 2000.
4. Dragos Truscan, Joao M. Fernandes and Johan M Lilius. Tool support for DFD to UML model-based transformations. Turku Centre for Computer Science, page 1-21.TUCS; 2003.
5. Jacquie Barker. Beginning java objects from concepts to code. Wrox Press LTd; 2000.
6. James Rumbaugh et al. Object-Oriented Modeling and Design. Prentice-Hall International; 1991.
7. Jeffrey L. System analysis and design methods. 6th ed., McGraw-Hill; 2004.
8. Joao Miguel Fernandes. Functional and Object-oriented Modeling of Embedded Software. Technical Report 512, TUCS;2003
9. Terry Quatrani. Visual Model with Rational Rose and UML. 4th ed.,Addison Wesley Longman;1998.
10. Whitten, Jeffery L. System analysis and design method. 6th ed., McGraw-Hill; 2004.
11. Joao Miguel Fernandes. Functional and Object-oriented Views in Embedded Software Modeling. Technical Report 512, TUCS;200
12. http://emhain.wit.ie/~tforan/fypDescription.html
13. http://www.ftponline.com/javapro/2004_03/online/code_gnaccarato_03_03_04/default_pf.aspx
14. http://www.javaworld.com/javaworld/jw-08-2000/jw-0818-javadoc.html
15. http://www.smartdraw.com/resources/centers/software/dfd.htm

16. http://www.andrew.cmu.edu/user/conzalez/Teaching/ISW1/DFD.html
17. http://www.java.sun.com
18. http://cis.k.hosei.ac.jp/~sliu/PDLecture8.ppt
19. http://www.comp.hkbu.edu.hk/~comp2221/ group19/design%20spec.doc
20. http://prdownloads.sourceforge.net/dfdedit
21. http://www.dwheeler.com/oss_fs_eval.html
22. http://developer.kde.org/documentation/standards/kde/style/basics/index.htm

APPENDIX

APPENDIX A
ANALYSIS USE-CASE NARRATIVE DOCUMENTS

Table 1 The analysis “Create a workspace” use-case narrative

Use Case Name:	Create a workspace	
System Actor:	- System designer - System developer	
Description:	This use case describes the event of creating a workspace by opening the application, the data flow diagram code generator. On completion a workspace is generated.	
Precondition:	None	
Trigger:	This use case is initiated when a system designer or system developer opens the application.	
Typical Course of Events:	**Actor Actions**	**System Response**
	Step 1: This use case begins when a system designer or system developer opens the application.	**Step2:** The workspace is successfully created.
Conclusion:	This use case concludes when the workspace is successfully created.	
Postcondition:	The workspace ready to be created new drawing canvas or to be exited the application	

Table 2 The analysis “Create a workspace” use-case narrative

Use Case Name:	Create a drawing canvas
System Actor:	- System designer
Description:	This use case describes the event of creating drawing canvas to draw a data flow diagram by receive information about the data

Table 2 The analysis “Create a workspace” use-case narrative (Continued)

<table>
<tr><td>Description:</td><td colspan="2">flow diagram from a system designer. In this use case, an empty drawing canvas is then produced. The canvas is now prom to create a data flow diagram.</td></tr>
<tr><td>Precondition:</td><td colspan="2">The workspace are created</td></tr>
<tr><td>Trigger:</td><td colspan="2">This use case is initiated when a system designer selects the option to create a new data flow diagram</td></tr>
<tr><td rowspan="4">Typical Course of Events:</td><td>Actor Actions</td><td>System Response</td></tr>
<tr><td>Step 1: This use case begins when a system designer wishes to create a data flow diagram and selects an option to create a new data flow diagram.</td><td>Step 2: The system prompts the user for input a level name of the data flow diagrams.</td></tr>
<tr><td rowspan="2">Step 3: The system designer inserts the level name of the data flow diagram.</td><td>Step 4: An empty drawing canvas is created.</td></tr>
<tr><td>Step 5: the system links canvas to the canvas link.</td></tr>
<tr><td>Alternate Course:</td><td colspan="2">Alt-step 1: The system designer opens a file of this application. All file data are read and added to links. The canvas data are inserted into canvas link and the symbol data are inserted into symbol link. When the system designer selects a canvas, that canvas is created. Terminate use case.
Alt-step 1: The system designer selects an option to create a new level of the data flow diagram.
Alt-step 3: The system designer cancel to create the new level of data flow diagram. Terminate use case.</td></tr>
<tr><td>Conclusion:</td><td colspan="2">This use case concludes when the drawing canvas successfully created.</td></tr>
<tr><td>Postcondition:</td><td colspan="2">The drawing canvas ready to be to created data flow diagram</td></tr>
</table>

Table 3 The analysis “Create a data flow diagram” use-case narrative

Use Case Name:	Create a data flow diagram	
System Actor:	- System designer	
Description:	This use case describes the event of creating a data flow diagram in canvases. In this use case, a system designer able to insert, update, and delete all the relevant features of the data flow diagram. On completion the data flow diagram has been successfully created.	
Precondition:	The drawing canvas are created	
Trigger:	This use case is initiated when a system designer selects the drawing canvas to create a new data flow diagram	
Typical Course of Events:	**Actor Actions**	**System Response**
	Step 1: This use case begins when a system designer wishes to create a data flow diagram and selects a drawing canvas to draw a data flow diagram.	**Step 2:** the selected drawing canvas is displayed in the workspace.
	Step 3: The system designer selects an option to create data flow diagram including drawing a data flow, drawing a data store, drawing a entity, drawing a process, and selecting a symbol.	**Step 4:** The system checks status to know the system user select what status.
	Step 5: The system designer identified a position on the drawing canvas to draw or select a symbol.	**Step 6:** The drawing canvas accepts the position.
	Step 7: The system designer modifies a symbol.	**Step 8:** The drawing canvas inserts, updates or deletes the selected symbol on that position.

Table 3 The analysis “Create a data flow diagram” use-case narrative (Continued)

Typical Course of Events:	**Actor Actions**	**System Response**
		Step 9: the symbol link is modified
Alternate Course:	**Alt-step 6:** If the system designer selects a symbol, the drawing canvas accepts the position and checks what is the selected symbol. **Alt-step 7:** The system designer draws a new symbol. **Alt-step 8:** If the system designer draws a data flow, the drawing canvas displays a line contains an arrowhead, with labels indicating their content **Alt-step 8:** If the system designer draws a data store, the drawing canvas displays an open rectangle (3 sides, open on the right) that has a vertical lines, a label (name) and a process number in it. **Alt-step 8:** If the system designer draws an entity, the drawing canvas displays a rectangle with labels indicating their content. **Alt-step 8:** If the system designer draws a process, the drawing canvas displays a rounded rectangle that has horizontal lines and a label (name) in it. **Alt-step 9:** If the system designer draws a new symbol, the new symbol data is inserted into the symbol link	
Conclusion:	This use case concludes when the data flow diagram successfully created.	
Postcondition:	All canvases and symbols are saved in a file.	

Table 4 The analysis “Generate Java code” use-case narrative

Use Case Name:	Generate Java code
System Actor:	- System developer
Description:	This use case describes the event of code generation from a data flow diagram. On completion, the generated Java code is presented to a system developer.

Table 4 The analysis “Generate Java code” use-case narrative (Continued)

Precondition:	Create a data flow diagram	
Trigger:	This use case is initiated when a system developer selects the drawing canvas to generate a code.	
Typical Course of Events:	**Actor Actions**	**System Response**
	Step 1: This use case begins when a system developer wishes to generate a code and selects a drawing canvas to generate a code.	**Step 2:** the selected drawing canvas are displayed in the workspace
	Step 3: the system developer selects an option, to generate Java code.	**Step 4:** The system will display the corresponding Java code on a text area.
Conclusion:	This use case concludes when code is successfully generated.	
Postcondition:	None	

Table 5 The analysis “Check illegal data flow diagram” use-case narrative

Use Case Name:	Check illegal data flow diagram	
System Actor:	- System designer	
Description:	This use case describes the event of ensuring about the correctness of data flow diagram. On failure, errors are reported and possible solutions are given.	
Precondition:	Create a data flow diagram	
Trigger:	This use case is initiated when a system designer select the drawing canvas to check illegal data flow diagram	
Typical Course of Events:	**Actor Actions**	**System Response**
	Step 1: This use case begins when a system designer wishes to check a data flow diagram and selects a drawing canvas to check illegal data flow diagram.	**Step 2:** the selected drawing canvas are displayed in the workspace

Table 5 The analysis “Check illegal data flow diagram” use-case narrative (Continued)

Typical Course of Events:	**Actor Actions**	**System Response**
	Step 3: the system designer select an option to check illegal data flow diagram	**Step 4:** The system checks illegal data flow diagram.
		Step 5: The system displays the error of illegal data flow diagram.
Alternate Course:	**Alt-step 5:** The system informs the system designer that the diagram is correct.	
Conclusion:	This use case concludes when the errors are reported and possible solutions are given.	
Postcondition:	The system designer corrects illegal data flow.	

APPENDIX B
REFINEMENT USE-CASE DOCUMENTS

Table 1 The refinement of the “Create a workspace” use case

Use Case Name:	Create a workspace	
System Actor:	- System designer - System developer	
Description:	This use case describes the event of creating a workspace by opening the application, data flow diagram code generator. On completion a workspace is generated.	
Precondition:	None	
Trigger:	This use case is initiated when a system designer or system developer opens the application.	
Typical Course of Events:	**Actor Actions**	**System Response**
	Step 1: This use case begins when a system designer or system developer opens the application.	**Step2:** The workspace is successfully created. *By The main program call the new workspace form.*
Conclusion:	This use case concludes when the workspace is successfully created.	
Postcondition:	The workspace ready to be created new drawing canvas or to be exited the application	

Table 2 The refinement of the “Create a drawing canvas” use case

Use Case Name:	Create a drawing canvas
System Actor:	- System designer
Description:	This use case describes the event of creating drawing canvas to draw a data flow diagram by receive information about the data flow diagram from a system designer. In this use case, an empty

Table 2 The refinement of the “Create a drawing canvas” use case (Continued)

<table>
<tr><td>Description:</td><td colspan="2">drawing canvas is then produced. The canvas is now prom to Create a data flow diagram.</td></tr>
<tr><td>Precondition:</td><td colspan="2">The workspace are created</td></tr>
<tr><td>Trigger:</td><td colspan="2">This use case is initiated when a system designer selects the option to create a new data flow diagram</td></tr>
<tr><td rowspan="4">Typical Course of Events:</td><td>Actor Actions</td><td>System Response</td></tr>
<tr><td>Step 1: This use case begins when a system designer wishes to create a data flow diagram and select an option (require a “new” drop-down menu or a “new” button option) to create a new data flow diagram.</td><td>Step 2: The system prompts the user for input a level name of the data flow diagram by the workspace display an input dialog for entering a level name.</td></tr>
<tr><td rowspan="2">Step 3: The system designer inserts the level name of the data flow diagram and confirms to create new drawing canvas by selects a “ok” button from the input dialog.</td><td>Step 4: An empty drawing canvas is created by the input dialog sends an event massage to the workspace for creates a new canvas.
The workspace needs a variable, symbol link.</td></tr>
<tr><td>Step 5: the system links canvas to the canvas link. The workspace needs a canvas link variable to insert level-name and symbol-link (symbols) of the canvas</td></tr>
<tr><td>Alternate Course:</td><td colspan="2">Alt-step 1: The system designer opens a file of this application. All file data are read and added to links. The canvas data are inserted into canvas link and the symbol data are inserted into symbol link by both of links must be cleared before insert data.</td></tr>
</table>

Table 2 The refinement of the “Create a drawing canvas” use case (Continued)

Alternate Course:	When the system designer selects a canvas, that canvas is created. Terminate use case. *Opening file requires “open” button/drop down menu to showing a filechooser dialog, allow the system designer to choose a filename to load from at a given location.* *The workspace needs a variable, canvas link. The canvas link variable inserts level name and symbol-link (symbols) of the canvas.* **Alt-step 1:** The system designer selects an option (*require a “new level” drop-down menu or “new sub level” drop-down menu)* to create a new level of the data flow diagram. **Alt-step 3:** The system designer selects *a “cancel” button from the input dialog* to create the new level of data flow diagram. Terminate use case.
Conclusion:	This use case concludes when the drawing canvas successfully created.
Postcondition:	The drawing canvas ready to be to created data flow diagram

Table 3 The refinement of the “Create a data flow diagram” use case

Use Case Name:	Create a data flow diagram
System Actor:	- System designer
Description:	This use case describes the event of creating a data flow diagram in canvases. In this use case, a system designer able to insert, update and delete all the relevant features of the data flow diagram. On completion the data flow diagram has been successfully created.
Precondition:	The drawing canvas are created
Trigger:	This use case is initiated when a system designer selects the drawing canvas to create a new data flow diagram

Table 3 The refinement of the "Create a data flow diagram" use case (Continued)

Typical Course of Events:	**Actor Actions**	**System Response**
	Step 1: This use case begins when a system designer wishes to create a data flow diagram. *The system designer selects a drawing canvas from an item of treecontrol.*	**Step 2:** the selected drawing canvas is displayed in the workspace.
	Step 3: The system designer select an option *(require "select", "entity", "data flow", "line", "process", and "data store" buttons)* to create data flow diagram including drawing a data flow, drawing a data store, drawing a entity, drawing a process, and selecting a symbol.	**Step 4:** The system checks status to know the system user select what status.
	Step 5: The system designer identified a position on the drawing canvas to draw or select a symbol.	**Step 6:** *the drawing canvas accepts coordinates x and y by a mouse click on the canvas. The modifiability a symbol is started when the system checks the selected coordinate in the bound of a symbol.*
	Step 7: The system designer modifies a symbol *by select "cut", "copy" or "paste" button / drop-down menu. Moreover the system designer able to resize or move the selected symbol by drag mouse.*	**Step 8:** *The drawing canvas is painted.*

Table 3 The refinement of the “Create a data flow diagram” use case (Continued)

Typical Course of Events:	**Actor Actions**	**System Response**
		Step 9: the symbol link is modified *If: the system designer selects the “cut” button/drop-down menu to cut the selected symbol, the symbol is cut by unlink the symbol link.* *Else if: the system designer selects the “copy” button/ drop-down menu to copy the selected symbol, the symbol is linked to a copied symbol link.* *The workspace needs a copied symbol link variable to insert symbols of the canvas* *Else if: the system designer selects the “paste” button/ drop-down menu to paste the new symbol, the symbol link links all symbols in the copied symbol link.*
Alternate Course:	**Alt-step 6:** If the system designer selects a symbol, the drawing canvas accepts the position and checks what is the selected symbol. **Alt-step 7:** The system designer draws a new symbol. **Alt-step 8:** If the system designer draws a data flow, the drawing canvas displays a line contains an arrowhead, with labels indicating their content	

Table 3 The refinement of the “Create a data flow diagram” use case (Continued)

Alternate Course:	**Alt-step 8:** If the system designer draws a data store, the drawing canvas displays an open rectangle (3 sides, open on the right) that has a vertical lines, a label (name) and a process number in it. **Alt-step 8:** If the system designer draws an entity, the drawing canvas displays a rectangle with labels indicating their content. **Alt-step 8:** If the system designer draws a process, the drawing canvas displays a rounded rectangle that has horizontal lines and a label (name) in it. *If a symbol of data flow are drawn, the drawing canvas accepts the coordinate x0, y0 from the event mouse press and the coordinate x1, y1 from the event mouse release. The event mouse drag is used to resize a symbol and the event key press is used to add text into the symbol. After that the new symbol is painted and the text are drawn in the drawing canvas.* **Alt-step 9:** If the system designer draws a new symbol, the new symbol data is inserted into the symbol link.
Conclusion:	This use case concludes when the data flow diagram successfully created.
Postcondition:	All canvases and symbols are saved in a file. *Saving file requires “save” button/drop down menu to showing a filechooser dialog, allow the system designer to choose a filename to save from at a given location.*

Table 4 The refinement of the “Generate Java Code” use case

Use Case Name:	Generate Java code
System Actor:	- System developer
Description:	This use case describes the event of code generation from a data flow diagram. On completion, the generated Java code is presented to a system developer.

Table 4 The refinement of the “Generate Java Code” use case (Continued)

Precondition:	Create a data flow diagram	
Trigger:	This use case is initiated when a system developer selects the drawing canvas to generate a code.	
Typical Course of Events:	**Actor Actions**	**System Response**
	Step 1: This use case begins when a system developer wishes to generate a code and selects a drawing canvas to generate a code. *The system designer selects a drawing canvas from an item of treecontrol.*	**Step 2:** the selected drawing canvas are displayed in the workspace
	Step 3: the system developer selects an option (*require “Generate Select Level” and “Generate All Level” drop-down menus/buttons* to generate Java code. Moreover the system developer able to select style of code by select a style option (*require “Processes Transform to Methods” and “Processes Transform to Classes” Radio button menus option*) to identify the coding style.	**Step 4*:*** *The system sends the symbol link data to generation for generate code by transform processes to methods or transform processes to classes.* The workspace will display the corresponding Java code on a text area.
Conclusion:	This use case concludes when code is successfully generated.	
Postcondition:	None	

Table 5 The refinement of the “Check illegal data flow diagram” use case

<table>
<tr><td>Use Case Name:</td><td colspan="2">Check illegal data flow diagram</td></tr>
<tr><td>System Actor:</td><td colspan="2">- System designer</td></tr>
<tr><td>Description:</td><td colspan="2">This use case describes the event of ensuring about the correctness of data flow diagram. On failure, errors are reported and possible solutions are given.</td></tr>
<tr><td>Precondition:</td><td colspan="2">Create a data flow diagram</td></tr>
<tr><td>Trigger:</td><td colspan="2">This use case is initiated when a system designer selects the drawing canvas to check illegal data flow diagram</td></tr>
<tr><td rowspan="4">Typical Course of Events:</td><td>Actor Actions</td><td>System Response</td></tr>
<tr><td>Step 1: This use case begins when a system designer wishes to check a data flow diagram and selects a drawing canvas to check illegal data flow diagram.</td><td>Step 2: the selected drawing canvas are displayed in the workspace</td></tr>
<tr><td rowspan="2">Step 3: the system designer selects an option (require a “check” button) to check illegal data flow diagram</td><td>Step 4: The system checks illegal data flow diagram.</td></tr>
<tr><td>Step 5: The system displays the data flow diagram error position by the select error symbols.</td></tr>
<tr><td>Alternate Course:</td><td colspan="2">Alt-step 5: The system informs the system designer that the diagram is correct.</td></tr>
<tr><td>Conclusion:</td><td colspan="2">This use case concludes when the errors are reported and possible solutions are given.</td></tr>
<tr><td>Postcondition:</td><td colspan="2">The system designer corrects illegal data flow diagram.</td></tr>
</table>

APPENDIX C
TEST CASES

Table 1 The test cases

Test case	CodeGenerator	CodeGeneration	CanvasLink	Data flow	Data store	DrawinCcanvas	Entity	IllegalDFD	Line	Link	Process	Rectangle	Status	SymbolLink	Symbol	workspace
Create a workspace																
Open the application.	x		x							x						x
Close the application.	x		x							x						x
Create a drawing canvas																
Select new canvas button						x										x
Select new level menu						x										x
Select new sub level menu						x										x
Create input context dialog						x										x
Create input level dialog						x										x
Create a new DFD canvas			x			x				x				x		x
Create a new level			x			x				x				x		x
Create a new sub level			x			x				x				x		x
Delete a level			x			x				x				x		x
Delete a sub level			x			x				x				x		x
Select a canvas			x			x				x				x		x
Save files			x			x				x				x		x
Open files			x			x				x				x		x
Create a data flow diagram																
Select a drawing canvas			x			x								x	x	x
Select data flow button						x							x			x

Table 1 The test cases (Continued)

Test case	CodeGenerator	CodeGeneration	CanvasLink	Data flow	Data store	DrawinCcanvas	Entity	IllegalDFD	Line	Link	Process	Rectangle	Status	SymbolLink	Symbol	workspace
Create a data flow diagram																
Draw data flows				x		x							x	x	x	x
Select data flows				x		x							x	x	x	x
Cut data flows				x		x							x	x	x	x
Copy data flows				x		x							x	x	x	x
Paste data flows				x		x							x	x	x	x
Move data flows				x		x							x	x	x	x
Edit data flows				x		x							x	x	x	x
Add text to data flows				x		x							x	x	x	x
Change text of data flows				x		x							x	x	x	x
Select data store button						x							x			x
Draw data stores					x	x							x	x	x	x
Select data stores					x	x							x	x	x	x
Cut data stores					x	x							x	x	x	x
Copy data stores					x	x							x	x	x	x
Paste data stores					x	x							x	x	x	x
Move data stores					x	x							x	x	x	x
Edit data stores					x	x							x	x	x	x
Add text to data stores					x	x							x	x	x	x
Change text of data stores					x	x							x	x	x	x
Add text No. to data stores					x	x							x	x	x	x
Change text No. of data stores					x	x							x	x	x	x
Select entity button						x							x			x
Draw entities						x	x						x	x	x	x
Select entities						x	x						x	x	x	x
Cut entities						x	x						x	x	x	x

Table 1 The test cases (Continued)

Test case	CodeGenerator	CodeGeneration	CanvasLink	Data flow	Data store	DrawinCcanvas	Entity	IllegalDFD	Line	Link	Process	Rectangle	Status	SymbolLink	Symbol	workspace
Create a data flow diagram																
Copy entities						x	x						x	x	x	x
Paste entities						x	x						x	x	x	x
Move entities						x	x						x	x	x	x
Edit entities						x	x						x	x	x	x
Add text to entities						x	x						x	x	x	x
Change text of entities						x	x						x	x	x	x
Select line button						x							x			x
Draw lines						x			x				x	x	x	x
Select lines						x			x				x	x	x	x
Cut lines						x			x				x	x	x	x
Copy lines						x			x				x	x	x	x
Paste lines						x			x				x	x	x	x
Move lines						x			x				x	x	x	x
Edit lines						x			x				x	x	x	x
Add text to lines						x			x				x	x	x	x
Change text of lines						x			x				x	x	x	x
Select process button						x							x			x
Draw processes						x					x		x	x	x	x
Select processes						x					x		x	x	x	x
Cut processes						x					x		x	x	x	x
Copy processes						x					x		x	x	x	x
Paste processes						x					x		x	x	x	x
Move processes						x					x		x	x	x	x
Edit processes						x					x		x	x	x	x
Add text to processes						x					x		x	x	x	x

Table 1 The test cases (Continued)

Test case	CodeGenerator	CodeGeneration	CanvasLink	Data flow	Data store	DrawinCcanvas	Entity	IllegalDFD	Line	Link	Process	Rectangle	Status	SymbolLink	Symbol	workspace
Create a data flow diagram																
Change text of processes						x					x		x	x	x	x
Add text No. to data stores						x					x		x	x	x	x
Change text No. of processes						x					x		x	x	x	x
Select all symbols			x			x								x		x
Unselect all symbols			x			x								x		x
Save files			x			x								x		x
Open files			x			x								x		x
Check illegal DFD																
Select a drawing canvas			x			x								x	x	x
Select check illegal button						x										x
Check entity to entity						x		x							x	x
Check entity to data store						x		x							x	x
Check data store to entity						x		x							x	x
Check data store to data store						x		x							x	x
Show DFD Complete dialog						x		x							x	x
Show illegal DFD dialog						x		x							x	x
Show entity to entity						x		x							x	x
Show entity to data store						x		x							x	x
Show data store to entity						x		x							x	x
Show data store to data store						x		x							x	x
Write entity to entity						x		x							x	x
Write text entity to data store						x		x							x	x
Write text data store to entity						x		x							x	x
Write data store to data store						x		x							x	x

Table 1 The test cases (Continued)

Test case	CodeGenerator	CodeGeneration	CanvasLink	Data flow	Data store	DrawinCcanvas	Entity	IllegalDFD	Line	Link	Process	Rectangle	Status	SymbolLink	Symbol	workspace
Generate Java code																
Select a drawing canvas			x			x								x	x	x
Select Generation code tab						x										x
Generate process to method			x											x		x
Generate process to class			x											x		x
Identify the coding style			x											x		x

APPENDIX D

SYSTEM TEST

The DFD of the Mobile Phone Game – “Plane” from the Java code Generator

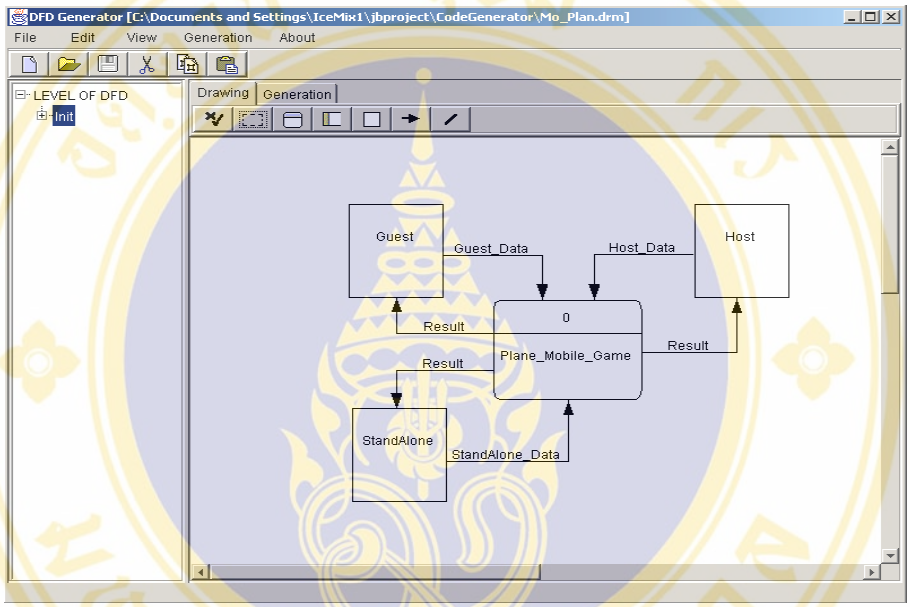


Figure1 The context diagram of the Mobile Phone Game – “Plane”

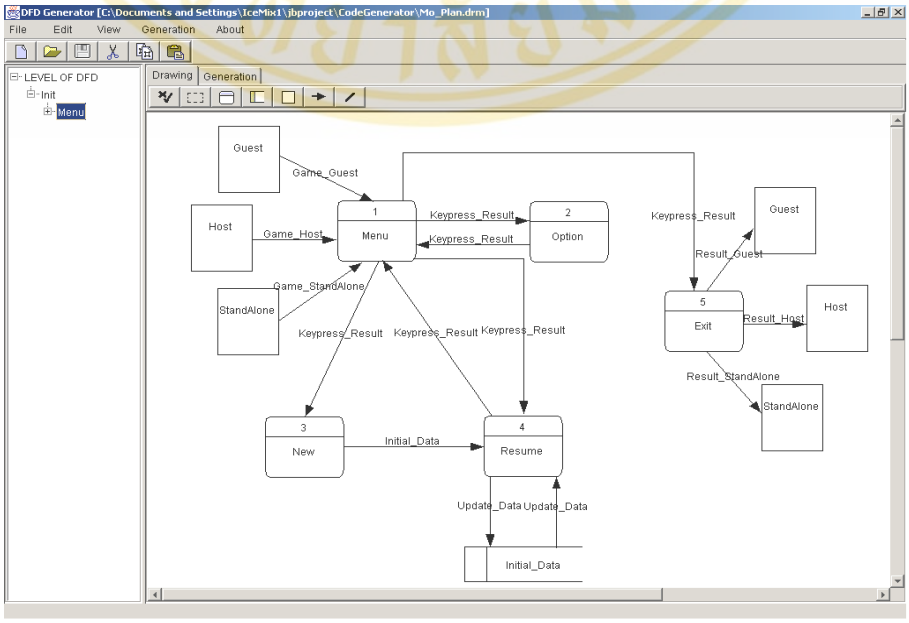


Figure2 The Level1-“Menu” of the Mobile Phone Game – “Plane”

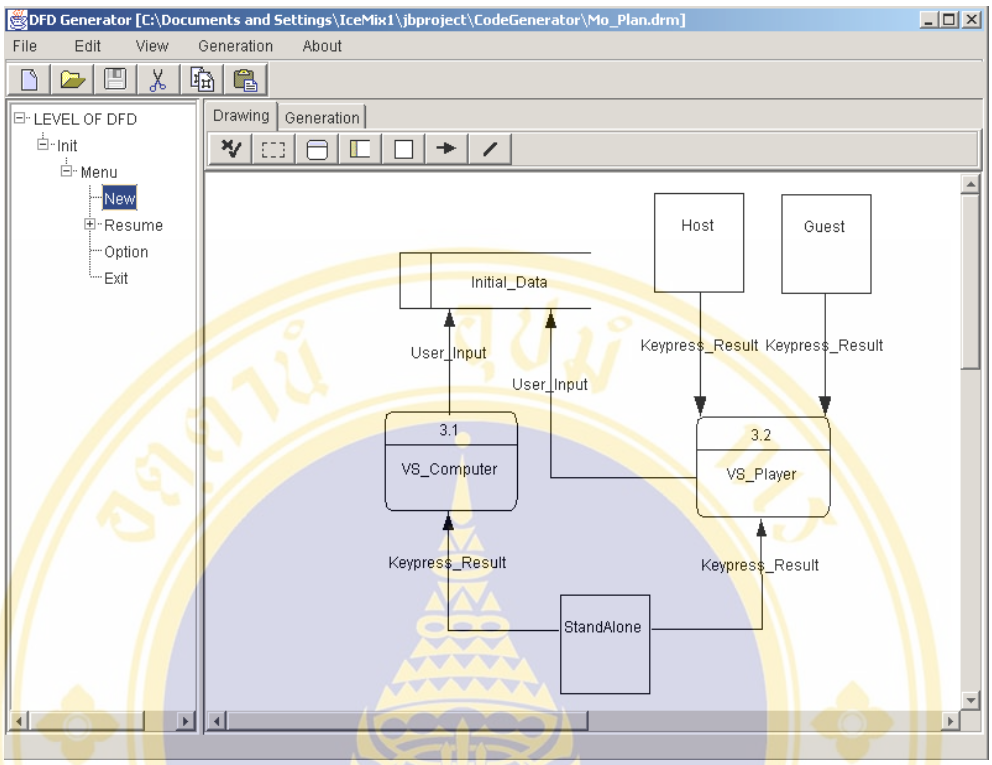


Figure 3 The Level2- “New” of the Mobile Phone Game – “Plane”

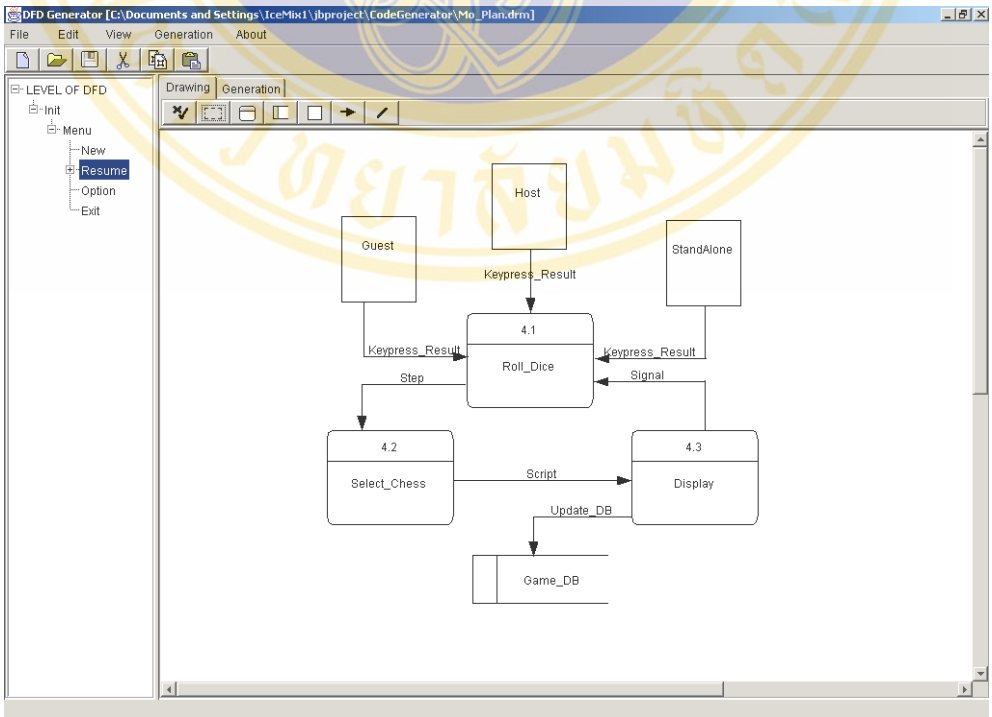


Figure 4 The Level2- “Resume” of the Mobile Phone Game – “Plane”

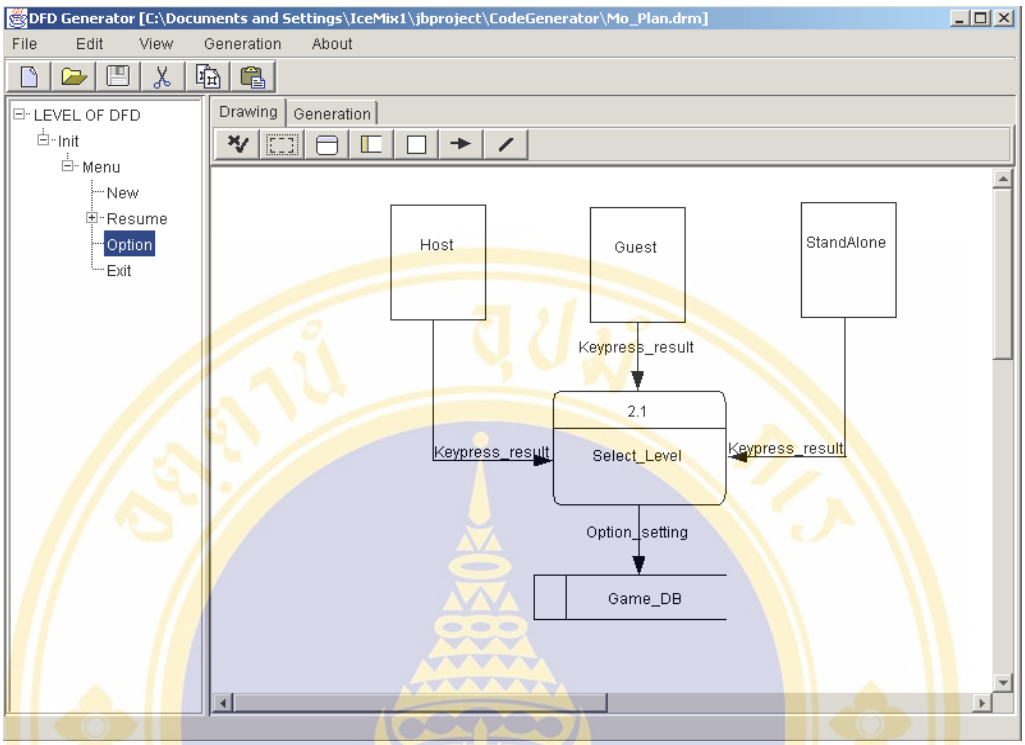


Figure 5 The Level2-“Option” of the Mobile Phone Game – “Plane”

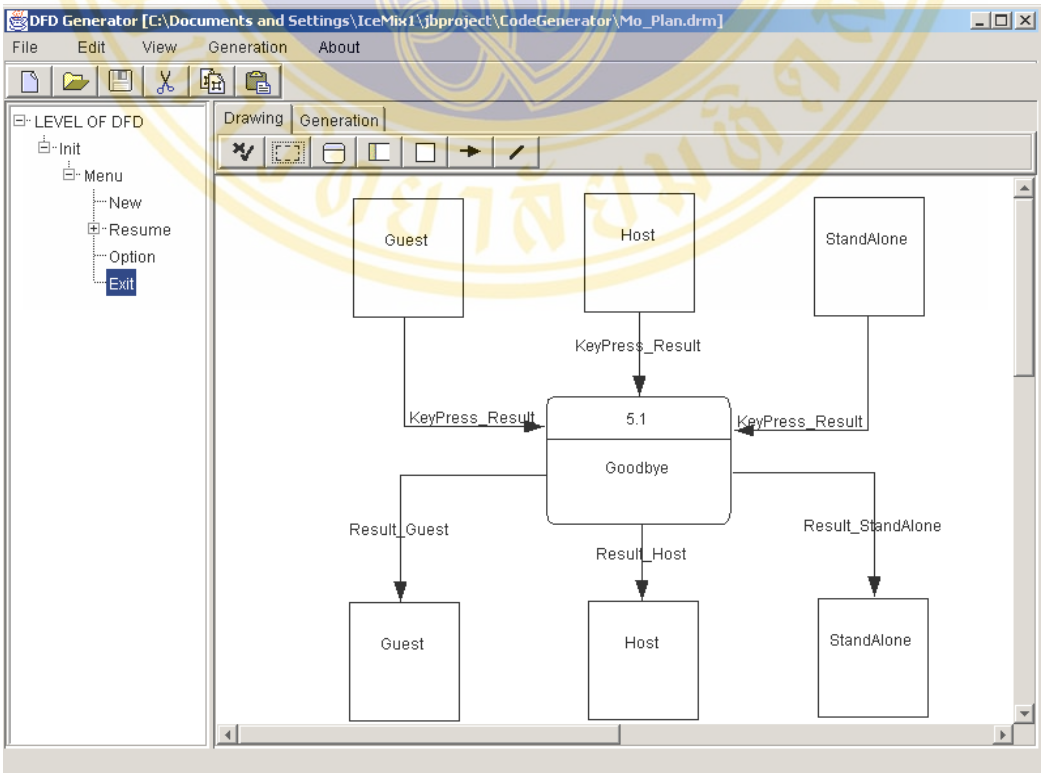


Figure 6 The Level2-“Exit” of the Mobile Phone Game – “Plane”

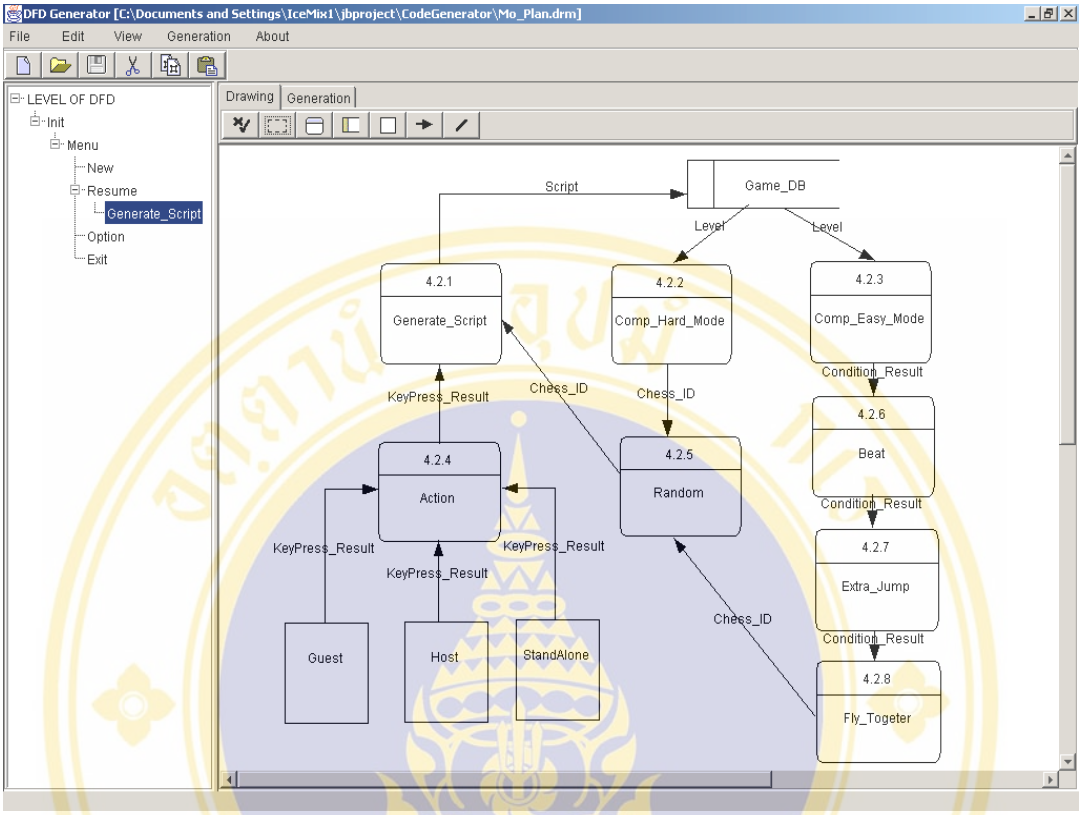


Figure 7 The Level3-“Generate Script” of the Mobile Phone Game – “Plane”

A Number of match, not match and missing methods

All processes of mobile phone “Plane” game DFD are transformed to methods for counting a number of following methods:

- Match methods: The code generator generates these methods, which their function appears in the process specification of the Mobile Phone “Plane” Game
- Not match methods: The code generator generates these methods but the process specification of the Mobile Phone “Plane” Game not identify the their function.
- Missing methods: The code generator not generates these methods whereas the function of them are displayed in the process specification of the Mobile Phone “Plane” Game.

Table 1 Process initial

The Mobile Phone "Plane" Game process specification	The Generated Code, Processes Transform to Methods

	Public class Init() {
	public static void main(String[] args) {
	}
	public void Plane_Mobile_Game() {
	}
	}

	Name of method - Main : the main function of this program - Plane_Mobile_Game() = Initial.initial() in process new
- Match method: 2 - Not match method: 0 - Missing method: -	

Table 2 Process menu

The Mobile Phone "Plane" Game process specification	The Generated Code, Processes Transform to Methods

Input: User Keypress.
Output: terminated or next function
Start process
Display choice;
Get_Keypress;
Case keypress() of
New : new()
Resume: resume()
Option: option()
Exit: quit()
	Public class Menu() {
	public void Menu() {
	}
	public void Option() {
	}
	public void New() {
	New New_1 = new New();
	}
	public void Resume() {
	Resume Resume_1 = new Resume();

Table 2 Process menu (Continued)

The Mobile Phone "Plane" Game process specification	The Generated Code, Processes Transform to Methods

End process	public void Exit() {
	Exit Exit_1 = new Exit();
	}
	}

Name of method - New.new() - Resume.resume() - Option.option() - Exit.quit()	Name of method - Menu() = Display choice - New() = Call New() - Resume() = Call Resume() - Option() = Call Option () - Exit() = Call Exit()
- Match method: 5 (Menu, New, Resume, Option, Exit) - Not match method: - - Missing method: -	

Table 3 Process option

The Mobile Phone "Plane" Game process specification	The Generated Code, Processes Transform to Methods

Input: User Keypress.
Output: Change of Level,
Back to menu()
Start process
Do Display option menu
If keypress()= level
If level =>easy then
level=hard
Else level=hard
End if
End if
	public class Option() {
	public void Select_Level() {
	}
	}

Table 3 Process option (Continued)

The Mobile Phone "Plane" Game process specification	The Generated Code, Processes Transform to Methods
While keypress != back End do Process menu() End process	

	Name of method
	Select_Level(): For implementation a path of code as follow
	If keypress()= level
	If level =>easy then
	level=hard
	Else level=hard
	End if

- Match method: 1 - Not match method: - - Missing method: -	

Table 4 Process new

The Mobile Phone "Plane" Game process specification	The Generated Code, Processes Transform to Methods

Input: User Keypress.
Output: initialed database
Start process
Display menu : VS player
Or VS computer
Case of keypress
VS player:player()
VS computer:computer()
Back : break function
	public class New() {
	public void VS_Computer() {
	}
	public void VS_Player() {
	}
	}

Table 4 Process new (Continued)

The Mobile Phone "Plane" Game process specification	The Generated Code, Processes Transform to Methods
End case of Process initial() Process resume() End process	
Name of method - Player.player() - VS_computer.computer() - Initial.initial() - Resume.resume()	Name of method - VS_Computer() = VS_Player.player() - VS_Player() = VS_Computer.computer() * Initial.initial(): Called by the Constructor * Resume.resume() : Called by the Constructor
- Match method: 2 (VS_Computer, VS_Player) - Not match method: - - Missing method: -	

Table 5 Process exit

The Mobile Phone "Plane" Game process specification	The Generated Code, Processes Transform to Methods

Input: User Keypress.
Output: Goodbye message
Start process
Display goodbye message
Delay for 3 second
Free all memory
End process
	Public class Exit() {
	public void Goodbye() {
	}
	}

	Name of method - Goodbye() = Display goodbye message
- Match method: 1 (Goodbye) - Not match method: - - Missing method: -	

Table 6 Process resume

The Mobile Phone "Plane" Game process specification	The Generated Code, Processes Transform to Methods

Input: User Keypress.
Output: update database
Start process
Do
Rolldice()
Selectchess()
Generatescript()
Do
Wait()
Move()
Display()
While (not player turn)
End do while
While (not end game) or (exit)
End do while
End process
	Public class Resume() {
	public void Roll_Dice() {
	}
	public void Select_Chess() {
	}
	public void Display() {
	}
	}

Name of method - Rolldice.rolldice() - Selectchess.selectchess() - Generatescript.generate script() - Display.display() - Wait.wait() - Move.move()	Name of method - Roll_Dice() = Rolldice.rolldice() - Select_Chess() = Selectchess.selectchess() - Display() = Display.display() * Generatescript(): Called by the Constructor
- Match method: 3 (Roll_Dice, Select_Chess, Display) - Not match method: - - Missing method: 2 (Wait.wait, Move.move)	

Table 7 Process rolldice

The Mobile Phone “Plane” Game process specification	The Generated Code, Processes Transform to Methods
Input: User Keypress. Output: update variable step Start process Display message If Keypress()= back Break process; Else Return Step=(random ()%6)+1 End process	Method in Resume
	Name of method - Resume.Roll_Dice: For implementation code from Algorithm
- Match method: 1 identified (Resume.Roll_Dice) - Not match method: - - Missing method: -	

Table 8 Process display

The Mobile Phone “Plane” Game process specification	The Generated Code, Processes Transform to Methods
Input: Database Output: N.A. Start process Display board For all chesses Display chess in there Location End process	Method in Resume

Table 8 Process display (Continued)

The Mobile Phone “Plane” Game process specification	The Generated Code, Processes Transform to Methods
	Name of method - Resume.Display(): For implementation code from Algorithm
- Match method: 1 identified (Resume.Display) - Not match method: - - Missing method: -	

Table 9 Process selectchess

The Mobile Phone “Plane” Game process specification	The Generated Code, Processes Transform to Methods
Input: User Keypress. Output: return chessID Start process Display board; Case of keypress() Chess1: Return chess 1 chessID Chess2: Return chess 2 ChessID Chess3: Return chess 3 chessID Chess4: Return chess 4 chessID Back : break process; End case of End process	Method in Resume

Table 9 Process selectchess (Continued)

The Mobile Phone “Plane” Game process specification	The Generated Code, Processes Transform to Methods
	Name of method - Resume.Select_Chess(): For implementation code from Algorithm
- Match method: 1 identified (Resume.Select_Chess) - Not match method: - - Missing method: -	

Table 10 Process generatescrtip

The Mobile Phone “Plane” Game process specification	The Generated Code, Processes Transform to Methods

Input: PlayerID, ChessID, step
Output: script
Start process
Script=Step*100+ChessID*10 +PlayerID
Case of PlayerID [ChessID][Location]+=step
Same color: script+=400
Flying position: script+=1200
Return script
End process
	Public class Generate_Script() {
	Public void Random() {
	}
	public void Generate_Script() {
	}
	public void Comp_Hard_Mode() {
	}
	public void Action() {
	}
	public void Comp_Easy_Mode() {
	}
	public void Beat() {
	}
	public void Extra_Jump() {
	}
	public void Fly_Togeter() {
	}
	}

Table 10 Process generatescrtip (Continued)

The Mobile Phone "Plane" Game process specification	The Generated Code, Processes Transform to Methods
	Name of method - Random() - Generate_Script() = Script - Comp_Hard_Mode() = Step - Action() - Comp_Easy_Mode() = Step - Beat() - Extra_Jump() - Fly_Togeter() = Flying position
- Match method: 4 (Generate_Script, Comp_Hard_Mode, Comp_Easy_Mode, Fly_Togeter) - Not match method: 4 (Random, Action, Beat, Extra_Jump) - Missing method: 0	

Table 11 Process player

The Mobile Phone "Plane" Game process specification	The Generated Code, Processes Transform to Methods
Input: User Keypress. Output: update database Start process Display menu; Case of keypress() Wireless: wireless() Stand alone: Standalone(); Back: Break process; End case of End process	Method in New()

Table 11 Process player (Continued)

The Mobile Phone "Plane" Game process specification	The Generated Code, Processes Transform to Methods
Name of method - Wireless.wireless() - Standalone.standalone();	Name of method New.VS_Player(): For implementation code from Algorithm
- Match method: 1 identified (New.VS_Player) - Not match method: - - Missing method: 2	

Table 12 Process computer

The Mobile Phone "Plane" Game process specification	The Generated Code, Processes Transform to Methods

Input: User Keypress.
Output: update database
Start process
Do Display menu;
If keypres()=Computer
If computer=3
Computer=1
Else Computer++
End if
End if
While (keypress()=start)
Or (keypress()=back)
End do while
If keypress()=start Initial()
Else Break process;
End if
End process
	Method in New()

Table 12 Process computer (Continued)

The Mobile Phone "Plane" Game process specification	The Generated Code, Processes Transform to Methods
Name of method - Initial.initial()	Name of method - New.VS_Computer(): For implementation code from Algorithm
- Match method: 1 identified (New.VS_Computer) - Not match method: - - Missing method: -	

Table 13 Process move

The Mobile Phone "Plane" Game process specification	The Generated Code, Processes Transform to Methods
Informal description: function to decode script Input: script Output: database Source/Sink: script, database Events: database updated Start process PlayerID=script % 10 ChessID=script/10 %10 Step=script/100 PlayerID[ChessID] [Location]+=step Update database End process	
- Match method: - - Not match method: - - Missing method: 1 identified (Move.move)	

Table 14 Process wait

The Mobile Phone “Plane” Game process specification	The Generated Code, Processes Transform to Methods
Input: N.A. Output: script Start process Do Open receiver While (not receive script) Return script End process	
- Match method: - - Not match method: - - Missing method: 1 identified (Wait.wait)	

Table 15 Result

Conclusion : The percent of match, not match and missing methods
- Match method: 18/22 = 81.81% - Not match method: 4/22 = 18.18% - Missing method: 4/22 = 18.18%

APPENDIX E
USER MANUAL

คู่มือผู้ใช้
โปรแกรมสร้างโค้ดภาษาจาวาจากแผนภาพกระแสข้อมูล

1 ความต้องการของระบบ

1. CPU Pentium 500 MHz ขึ้นไป
2. ระบบปฏิบัติการ Windows 95/98/2000/NT/XP
3. หน่วยความจำอย่างน้อย 32 MB ขึ้นไป
4. พื้นที่ว่างในฮาร์ดดิสต์อย่างน้อย 3 MB

2 การติดตั้ง Java Development Kit

ตรวจสอบโปรแกรมในเครื่องว่ามีการติดตั้ง JVM (Java Virtual Machine) และ JDK (Java Development Kit) หรือไม่โดยเข้าไปที่ Control Panel -> Add/Remove Program

หากในเครื่องได้มีการลง JVM และ JDK อยู่แล้ว จะมีข้อความดังรูปที่ 1 คือ Java 2 Runtime Environment Standard Edition v1.3 หมายถึง JVM และ Java 2 SDK Standard Edition v1.3 หมายถึง JDK

รูปที่ 1 ตรวจสอบ JVM และ JDK

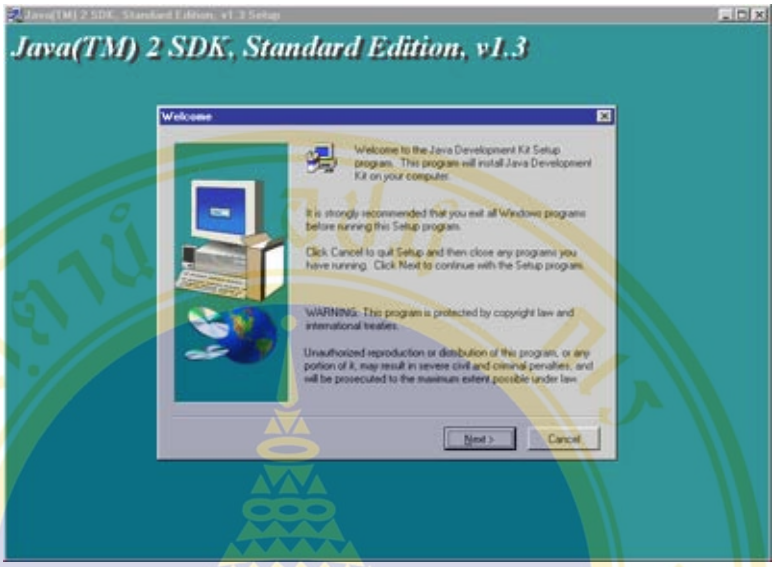
ถ้าไม่มีข้อความดังกล่าว ให้ทำการลง JVM และ JDK ที่มาพร้อมกับแผ่น CD ซึ่งอยู่ในไดเรกทอรี่ JDK ชื่อไฟล์ j2sdk-1_3_0_02-win.exe โดยมีขั้นตอนดังนี้

1 ทำการ Unpacking Java 2 SDK

2 เตรียมการติดตั้ง Java 2 SDK

รูปที่ 2 แสดงหน้าจอเตรียมการติดตั้ง Java 2 SDK

3 อ่านรายละเอียดและข้อตกลงของโปรแกรม Java 2 SDK แล้วกดปุ่ม Next >

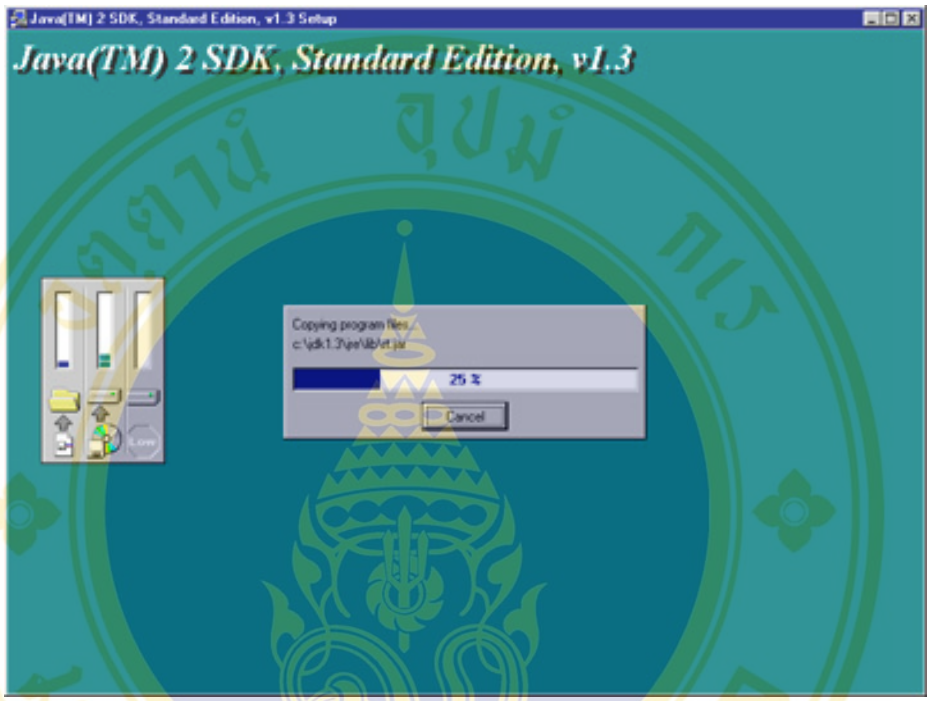


รูปที่ 3 แสดงรายละเอียดและข้อตกลงของโปรแกรม Java 2 SDK

4 เลือก Directory ที่จะติดตั้ง Java 2 SDK ซึ่งโปรแกรมจะใช้ Directory นี้ เพื่อกำหนด Path ที่ใช้ในการ Run โปรแกรม โดยปกติจะกำหนดอัตโนมัติที่ c:\jdk1.3

รูปที่ 4 แสดงรายละเอียดและข้อตกลงของโปรแกรม Java 2 SDK

5 โปรแกรมทำการติดตั้งโปรแกรม Java 2 SDK เมื่อติดตั้งเสร็จเรียบร้อย ควร Restart เครื่องคอมพิวเตอร์



รูปที่ 5 แสดงการติดตั้งโปรแกรม Java 2 SDK

3 การติดตั้งโปรแกรมสร้างโค้ดภาษาจาวาจากแผนภาพกระแสข้อมูล

ในการติดตั้งโปรแกรมสร้างโค้ดภาษาจาวาจากแผนภาพกระแสข้อมูล จะประกอบด้วยแผ่น CD ที่ใช้ในการติดตั้งจำนวน 1 แผ่น ซึ่งมีลำดับขั้นตอนการติดตั้งดังนี้

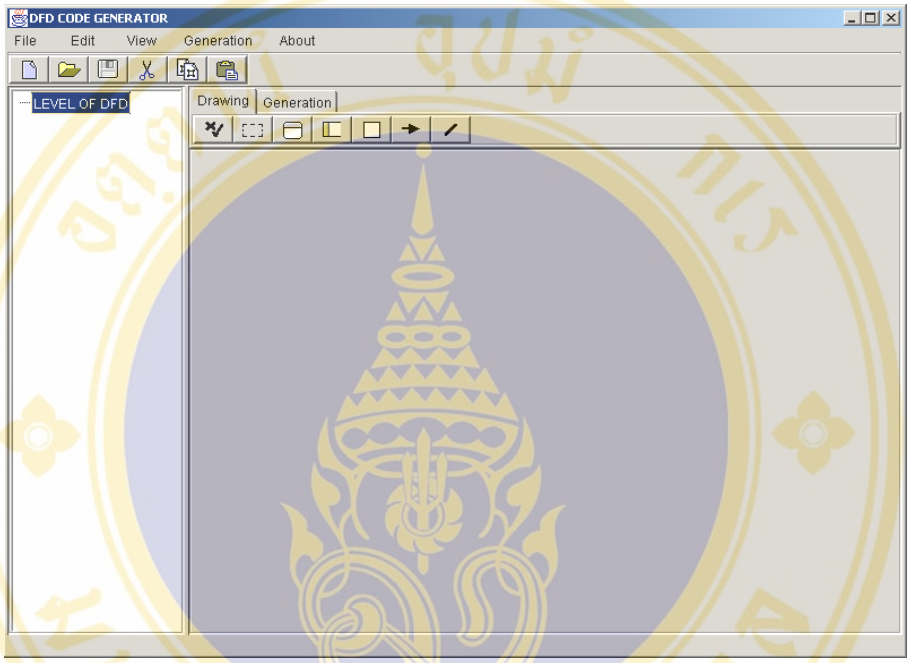
1 ใส่แผ่น CD ลงใน CD-ROM Drive

2 คัดลอกไฟล์ CodeGenerator.jar ในแผ่น CD ที่อยู่ในไดเรกทอรี่ CodeGenerator ไปไว้ ณ. ตำแหน่งใด ๆ ในฮาร์ดดิสต์ตามความต้องการ

3 ทำการ Run โปรแกรมโดยดับเบิ้ลคลิกที่ไฟล์ CodeGenerator.jar

4 อธิบายส่วนประกอบต่าง ๆ ในโปรแกรม

เมื่อโปรแกรมสร้างโค้ดภาษาจาวาจากแผนภาพกระแสข้อมูลเริ่มทำงาน จะปรากฏหน้าจอหลักของโปรแกรมดังรูปที่ 6



รูปที่ 6 หน้าจอหลักโปรแกรมสร้างโค้ดภาษาจาวาจากแผนภาพกระแสข้อมูล

หน้าจอหลักประกอบด้วยเมนูต่าง ๆ ซึ่งมีรายละเอียดการทำงานดังต่อไปนี้

1 เมนูแฟ้มข้อมูล (File) ประกอบด้วยเมนูย่อย ๆ ดังนี้

- เมนูสร้างแผนภาพกระแสข้อมูล (New)
- เมนูเปิดแผนภาพกระแสข้อมูล (Open)
- เมนูบันทึกแผนภาพกระแสข้อมูล (Save)
- เมนูสร้าง level ของแผนภาพกระแสข้อมูลใหม่ (New Level)
- เมนูสร้าง sub level ของแผนภาพกระแสข้อมูลใหม่ (New Sublevel)
- เมนูออกจากโปรแกรม (Exit)

2 เมนูแก้ไข (Edit) ประกอบด้วยเมนูย่อย ๆ ดังนี้

- เมนูลบสัญลักษณ์ใด ๆ ในแผนภาพกระแสข้อมูล (Cut)
- เมนูคัดลอกสัญลักษณ์ใด ๆ ในแผนภาพกระแสข้อมูล (Copy)
- เมนูวางสัญลักษณ์ใด ๆ ในแผนภาพกระแสข้อมูล (Paste)
- เมนูเลือกสัญลักษณ์ทั้งหมดในแผนภาพกระแสข้อมูล (Select All)
- เมนูยกเลิกการเลือกสัญลักษณ์ทั้งหมดในแผนภาพกระแสข้อมูล (Unselect All)

3 เมนูมุมมอง (View) ประกอบด้วยเมนูย่อย ๆ ดังนี้

- เมนูมุมมอง Windows (Look and Feel Windows)
- เมนูมุมมอง Motif (Look and Feel Motif)
- เมนูมุมมอง Metal (Look and Feel Metal)
- เค้าโครงแนวนอน (Windows Orientation Horizontal)
- เค้าโครงแนวตั้ง (Windows Orientation Vertical)

4 เมนูสร้างโค้ด (Generation) ประกอบด้วยเมนูย่อย ๆ ดังนี้

- เมนูการสร้างโค้ดจาก level ใด level หนึ่งของแผนภาพกระแสข้อมูล (Generate Select Level)
- เมนูการสร้างโค้ดจากแผนภาพกระแสข้อมูล (Generate All Level)
- เมนูรูปแบบการสร้างโค้ดโดยแปลง Process เป็น Method (Coding style/Processes to Method)
- เมนูรูปแบบการสร้างโค้ดโดยแปลง Process เป็น Class (Coding style/Processes to Classes)

5 เมนูแสดงข้อมูลของโปรแกรม (About) ประกอบด้วยเมนูย่อย ๆ ดังนี้

- เมนูแสดงข้อมูลของโปรแกรม (About)

หน้าจอหลักประกอบด้วย Toobar ต่าง ๆ ได้แก่ Main toolbar, Drawing toolbar และ Generation toolbar ซึ่งในแต่ละ toolbar จะประกอบด้วยปุ่มการทำงานต่าง ๆ ดังนี้

1 Main toolbar ประกอบด้วยปุ่มการทำงานมาตรฐาน ดังรูปที่ 7

รูปที่ 7 แสดง Main toolbar

คำอธิบายสัญลักษณ์

- หมายเลข 1 คือ ปุ่มสร้างแผนภาพกระแสข้อมูล (New)
- หมายเลข 2 คือ ปุ่มเปิดแผนภาพกระแสข้อมูล (Open)
- หมายเลข 3 คือ ปุ่มบันทึกแผนภาพกระแสข้อมูล (Save)
- หมายเลข 4 คือ ปุ่มลบสัญลักษณ์ใด ๆ ในแผนภาพกระแสข้อมูล (Cut)
- หมายเลข 5 คือ ปุ่มคัดลอกสัญลักษณ์ใด ๆ ในแผนภาพกระแสข้อมูล (Copy)
- หมายเลข 6 คือ ปุ่มวางสัญลักษณ์ใด ๆ ในแผนภาพกระแสข้อมูล (Paste)

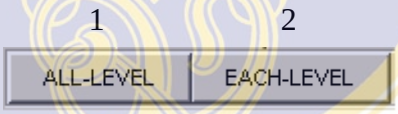
2 Drawing toolbar ประกอบด้วยปุ่มสร้างและตรวจสอบความถูกต้องของสัญลักษณ์แผนภาพกระแสข้อมูล ดังรูปที่ 8

รูปที่ 8 แสดง Drawing toolbar

คำอธิบายสัญลักษณ์

- หมายเลข 1 คือ ปุ่มตรวจสอบความถูกต้อง (Check)
- หมายเลข 2 คือ ปุ่มเลือกสัญลักษณ์ใด ๆ ในแผนภาพกระแสข้อมูล (Select)
- หมายเลข 3 คือ ปุ่มสร้าง Process (Process)
- หมายเลข 4 คือ ปุ่มสร้าง Data store (Data store)
- หมายเลข 5 คือ ปุ่มสร้าง Entity (Entity)
- หมายเลข 6 คือ ปุ่มสร้าง Data flow (Data flow)
- หมายเลข 7 คือ ปุ่มสร้างเส้นตรง (Line)

3 Generation toolbar ประกอบด้วยปุ่มสร้างโค้ดภาษาจาวาจากแผนภาพกระแสข้อมูล ดังรูปที่ 9



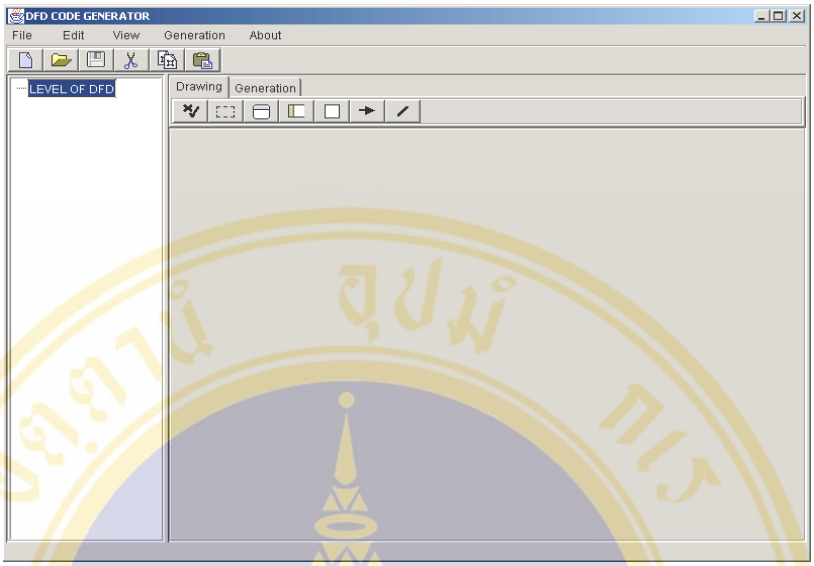
รูปที่ 9 แสดง Generation toolbar

คำอธิบายสัญลักษณ์

- หมายเลข 1 คือ เมนูการสร้างโค้ดจากแผนภาพกระแสข้อมูล (All-Level)
- หมายเลข 2 คือ ปุ่มเมนูการสร้างโค้ดจาก level ใด level หนึ่งของแผนภาพกระแสข้อมูล (Each-Level)

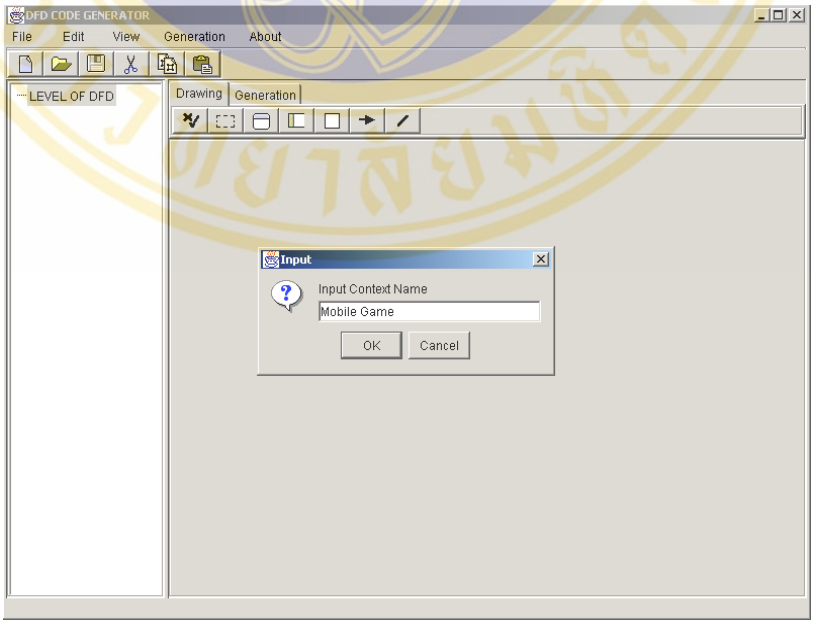
5 วิธีการสร้างโค้ดภาษาจาวาจากแผนภาพกระแสข้อมูล

1 เปิดโปรแกรมสร้างโค้ดภาษาจาวาจากแผนภาพกระแสข้อมูล เมื่อโปรแกรมพร้อมทำงานจะปรากฏหน้าจอหลักของโปรแกรม ดังรูปที่ 10



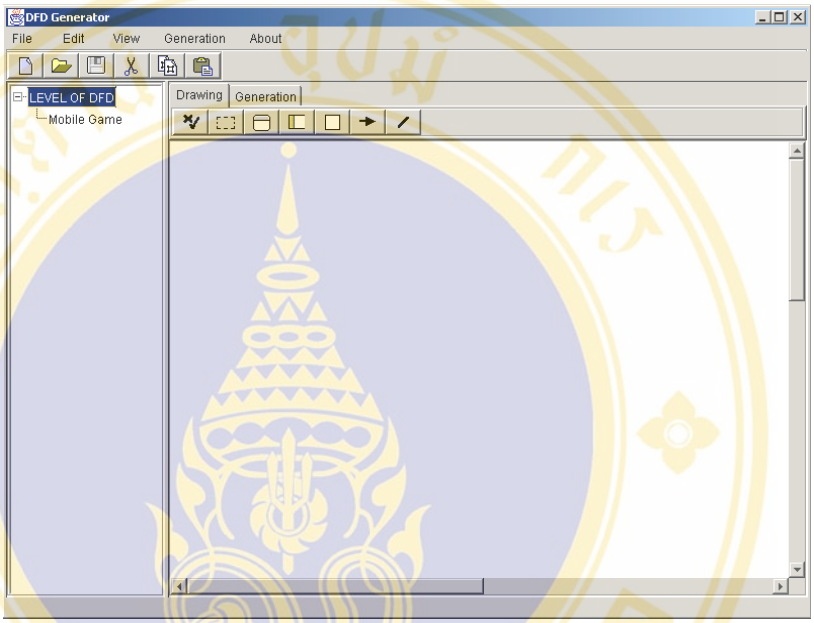
รูปที่ 10 หน้าจอหลักของโปรแกรม

2 กดปุ่มสร้างแผนภาพกระแสข้อมูลใหม่ โปรแกรมจะให้ผู้ใช้ระบุชื่อ Context Diagram ของระบบงานที่ต้องการสร้างแผนภาพกระแสข้อมูล ดังรูปที่ 11



รูปที่ 11 แสดงการระบุชื่อ Context Diagram

3 เมื่อการระบุชื่อ Context Diagram เสร็จสิ้นจะปรากฏหน้าจอ ดังรูปที่ 12 ซึ่งผู้ใช้สามารถทำการสร้างและตรวจสอบความถูกต้องของสัญลักษณ์ในแผนภาพกระแสข้อมูลได้โดยใช้เครื่องมือต่าง ๆ ในกลุ่ม Drawing toolbar

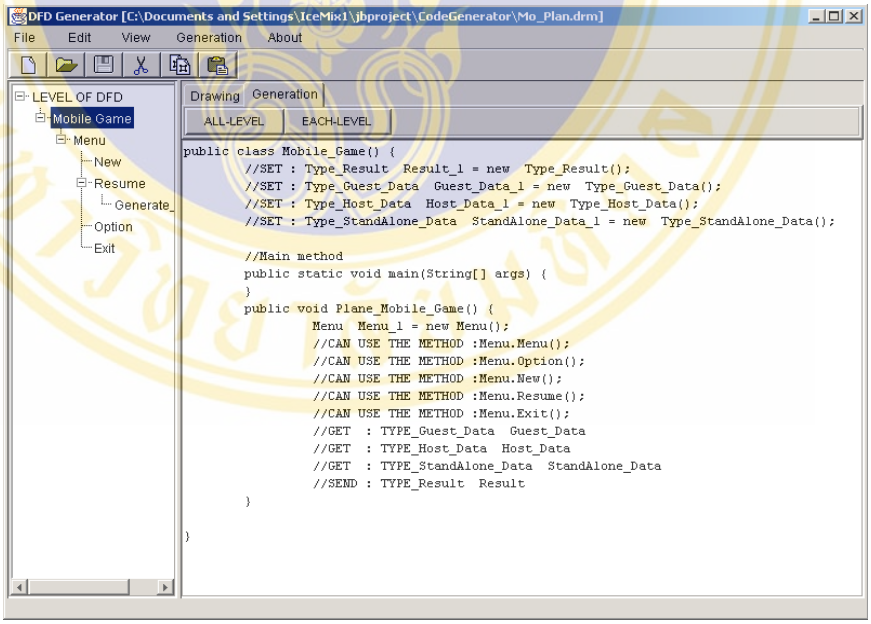


รูปที่ 12 หน้าจอสำหรับวาดแผนภาพกระแสข้อมูล

สำหรับการย่อ-ขยายหรือใส่ข้อความลงในสัญลักษณ์ใด ๆ ในแผนภาพกระแสข้อมูล สามารถทำได้โดยกดปุ่มเลือกสัญลักษณ์ (Select) ก่อน แล้วทำการเลือกที่สัญลักษณ์นั้นเพื่อทำการย่อ-ขยาย หรือใส่ข้อความที่ต้องการ

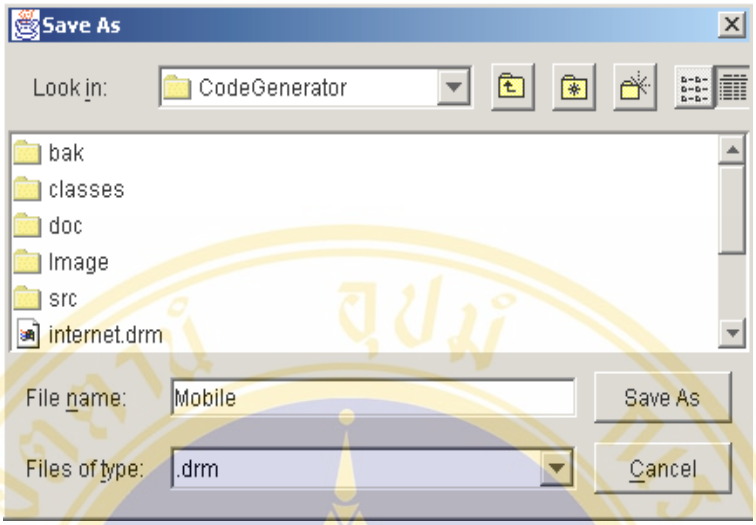
4 ในกรณีที่ต้องการจะสร้างแผนภาพกระแสข้อมูลใน level หรือ sublevel ถัดไปให้เลือกเมนูสร้าง level ของแผนภาพกระแสข้อมูลใหม่ (File->New Level) หรือเมนูสร้าง sub level ของแผนภาพกระแสข้อมูลใหม่ (File->New Sublevel) แล้วระบุชื่อแผนภาพกระแสข้อมูลใน level นั้น ๆ เมื่อการระบุชื่อเสร็จสิ้นโปรแกรมจะสร้างพื้นที่สำหรับสร้างแผนภาพกระแสข้อมูลใหม่ให้แก่ผู้ใช้

5 หลังจากสร้างแผนภาพกระแสข้อมูลแล้ว ผู้ใช้สามารถทำการแปลงแผนภาพเป็นโค้ดภาษาจาวาได้ โดยเลือกรูปแบบของโค้ดจากเมนูรูปแบบการสร้างโค้ดโดยแปลง Process เป็น Method (Generation ->Coding style/Processes Transform to Method) หรือ เมนูรูปแบบการสร้างโค้ดโดยแปลง Process เป็น Class (Generation->Coding style/Processes Transform to Classes) แล้วจึงเลือกแท็บ Generation ซึ่งจะปรากฏหน้าจอแสดงรายละเอียดของโค้ดที่สร้างจากแผนภาพกระแสข้อมูล ดังรูปที่ 13 นอกจากนี้ผู้ใช้สามารถใช้เครื่องมือต่าง ๆ ในกลุ่ม Generation toolbar เพื่อเลือกดูโค้ดที่สร้างจากแผนภาพกระแสข้อมูลเพียง level เดียวหรือดูจากทุก level



รูปที่ 13 หน้าจอแสดงรายละเอียดของโค้ดที่สร้างจากแผนภาพกระแสข้อมูล

6 เมื่อต้องการบันทึก สามารถทำได้โดยกดปุ่มบันทึกแผนภาพกระแสข้อมูล โปรแกรมจะแสดง Dialog Box ให้ผู้ใช้ระบุชื่อไฟล์ที่ต้องการบันทึก ดังรูปที่ 14 แล้วกดปุ่ม Save



รูปที่ 14 Dialog Box สำหรับระบุชื่อไฟล์ที่ต้องการบันทึก

7 หากต้องการออกจากโปรแกรม สามารถทำได้โดยการเลือกเมนูออกจากโปรแกรม (File->Exit) หรือ กดปุ่มปิดที่อยู่ด้านบนขวาของตัวโปรแกรม

BIOGRAPHY

NAME	Miss Chirawan Ronran
DATE OF BIRTH	30 October 1979
PLACE OF BIRTH	Lamphang, Thailand
INSTITUTIONS ATTENED	Maejo University, 1997 – 2001: Bachelor of Science (Computer Science) Mahidol University, 2003-2005: Master of Science (Technology of Information System Management)
HOME ADDRESS	257 Soi 11 Tambon Pichai Amphoe Maung Lamphang Thailand.