Intercomp Ltd.

MineIT™

Automated Extraction of Business Rules from Legacy COBOL Applications to Java Objects and Enterprise Java Beans

White Paper

October 1999

Intercomp Ltd.
6 Maskit Street
Herzliya 46733
Israel

Tel: +972-9-9526777
Fax: +972-9-9526170
E-mail: info@cobol2java.com
Web Site: www.cobol2java.com

November 30, 1999

Contents

About Intercomp Ltd.

Intercomp Ltd. is a subsidiary of Crystal Systems Solutions (NASDAQ: "CRYS") and a Formula Group company (NASDAQ: "FORTY").

Intercomp's line of products is based on a set of compiler-centric software tools that provide a comprehensive automated solution for the migration of legacy mainframe and non-mainframe COBOL applications to Java N-tier client/server e-commerce applications.

The set of products, certified by Sun Microsystems as "100% Pure Java," offers its users the ability to analyze their legacy systems, reengineer and convert them into a new N-tier paradigm.

About Crystal Systems Solutions Ltd.

Crystal Systems Solutions Ltd., and its subsidiaries, provide Fortune 500 companies, and others, with assessments, conversion methodology, software and professional services for all their mainframe and non-mainframe based activities with minimum investment in manpower, resources, and time.

Partial Customer List: FORD Motor Co., Kraft Foods, Farmer Insurance Group, KeyCorp, MCI, BMW, MetLife, United Technologies Pratt & Whitney, Blue Cross / Blue Shield, Phillips Petroleum, Ralston Purina, Alex Laurie Factors (UK), Reynolds Metal, Medical Mutual of Ohio, El Al Israel Airlines, Bank HaMizrahi, Bezeq (Israeli PTT), Bank Leumi–Israel

Crystal Systems Solutions works both independently, and in conjunction with local business partners worldwide who provide project management and system integration activities in conjunction with its products. The list of business partners includes Ernst & Young LLP, Logica Ltd., EDS, SAIC, CGI Informatik GmbH and others.

About the Formula Group

Formula Systems Ltd. (1985) is an information technologies company principally engaged, through its subsidiaries and affiliates, in the development and marketing of proprietary software products and in providing information systems solutions. Formula Systems has a number of subsidiaries which are traded on the Tel-Aviv Stock Exchange (TASE), as well as subsidiaries which are traded on NASDAQ. Additionally, Formula Systems itself is traded on the TASE and has ADR's listed on NASDAQ. Between 1994 and 1998 Formula Systems' revenues grew to over $250 million making Formula Systems the largest publicly-traded software group in Israel. Formula Systems currently employs a staff of over 3,500 employees of which more than 3,000 are computer experts.

Strategic Partnerships of the Formula Group: Formula Systems has developed partnerships with leading players in a broad range of fields. Formula Systems' strategic partners include, among others, BAAN Company, Cincinnati Bell information Systems Inc., Cooper Industries Inc., Ernst & Young LLP and Informix Software Inc.

Key Customers of the Formula Group: ABB, Amdocs, AT&T, Bell Atlantic, Bezeq, Coca-Cola, Comverse, Deutsche Telekom, Dow Chemical, ECI Telecom, Elect. de FranceEskom, FAA, Ford GEC Alsthom, Informix, Israel Electric Corp., Israel Air- Force, Keycorp, Krupp Kraft, Scitex, Telecom Asia, Telstra, Tivoli, Wharf Cable.

1. Legacy System Challenge

The Problem

In the last few years, the message "the Internet changes everything" has become one of the main axioms in the IT world. Indeed, it changed the concept of information and its usage in worldwide business, and its full implications are far from being determined or predictable. Yet, one thing has not changed: information is still a major and fundamental factor in an organization's success. This new era of e-business and Internet communication poses an acute problem to many CIOs: the valuable information that has been accumulated throughout the years, which resides in the organization's IT applications, is deployed in an old, closed and inflexible architecture. Nevertheless, this information is critical to an organization's competitiveness in a fast developing business environment; competitiveness that relies today on developing e-commerce capabilities and an open and flexible IT environment. An example of this situation is clearly seen in the mainframe world: 70% of the software is implemented via obsolescent platforms such as COBOL programs, CICS transactions, VSAM files, etc. Due to their proven stability and reliability, they have supported the core business applications of major organizations; however, they cannot support many of the developments that have already taken place in the IT world, and many that are still unfolding as the Internet, indeed, "changes everything."

The Core Business Has Not Changed

About 70% to 80% of legacy applications s are composed of business rules that are "wrapped" and supported today by obsolescent legacy technology. Hundreds of staff-years have been invested in these legacy applications. Are CIOs really willing to throw it all away? The company's complex business rules are intertwined in the existing application structure. Since these business rules comprise the very core of the organization's competitiveness, their logic should be kept intact and reused in a new technological environment, once they are identified and extracted.

An illustration of this situation, are companies in the financial sector, such as banks and insurance companies. A close look at these companies' business processes reveals that the core business of those companies has not changed because of the Internet and information evolution. Banks still loan money to their customers and therefore still calculate interest rates. They were doing so long before technology was there to support it, and will probably be doing so no matter what technology evolves next. Insurance companies still provide insurance to customers, and therefore still gather, calculate and store policy information according to each customer's specific needs and demands. All these calculations, from the simplest to the most complex, are still useful to these companies. They are these financial organizations' *business rules*.

Unlocking the Power of Existing Systems

As the example in the financial sector, the major problem with these organizations' business rules is that they are "imprisoned" inside obsolescent IT architecture. The following analysis explains where the business rules reside in IT applications, and how can they be extracted and separated from the flow of applications.

A typical application contains several components:

- *Code that handles the data*: Examples of this component are segments of the code that handle data storage, data retrieval, etc. These segments of code are by definition architecture-dependent, since data that is stored on files will be accessed by code that is written in one way, while data that is stored in a relational database would be accessed by code that is written differently.
- *Code that handles the display of information*: Examples of this component are segments of code that handle 3270 screens, or code that handles reports. Much like the case of data access, the code that handles the display of information is dependent on display devices that are used by specific architecture.
- *Code that handles calculations*: These segments of code are the major component of an application where business rules can be found.

Some parts of the original application must be discarded when moving to a new architecture, as they cannot be reused. These parts are, for instance, the segments of code that handle data access or display information on the application's screens.

Architecture-dependent code also exists in the calculation component of the application, since it is usually implemented in a language such as COBOL, and may be using external services such as CICS. Both the language must be replaced by a newer language (an object oriented one such as Java) and the external services replaced by equivalent services in the new environment.

In order to unlock the power of existing systems, the business rules of these systems need to be extracted, and then ported to a newer architecture. To achieve this purpose, the code that contains the organization's business rules must be separated from the rest of the application code, and then ported to a new language so it can be used in the new environment.

2. MineIT Solution

Intercomp's **MineIT** provides a comprehensive solution for the mining of business rules. With the **MineIT** tool, you are able to locate business rules originating from various sources (such as COBOL programs and copybooks), extract them out of the legacy code, and port them to Java in an automated way.

Compiler-Centric Approach

The capabilities of **MineIT** to locate, extract and port business rules, are based on an engine which is essentially a "smart compiler." This compiler-centric approach gives **MineIT** an automated capability to analyze and understand the semantics of business rules; without it, business rules mining would be a labor-intensive and tedious job.

The kernel of the **MineIT** product is a large multi-phase compiler package that supports a wide range of COBOL dialects including embedded software packages such as CICS (Customer Information Control System), SQL (Structured Query Language), DDL (Data Definition Language), BMS (Bitmap Screens), etc.

The basic translation scheme is syntax directed and uses mapping techniques in order to achieve maximum clarity and efficiency in the resulting Java source code.

The compiler builds large data structures that semantically describe the application. This description can be saved as an IDE (Integrated Development Environment) file, which enables the user to refine or change the description and perform the translation at a later time.

There are multiple information-collecting phases that take place during compilation: lexical, syntactic and semantic. The final phase performs the actual translation of the application into Java using the results of the previous phases. The original source code is significantly improved by means of the translation schemes used by the translator in conjunction with the information collected.

Lexical Analysis

The input source code file is lexically analyzed. The lexical analyzer partitions the input stream into character strings by matching them with a set of predefined regular expressions. The constructed linked token sequence provides a list of terms that is based on the source file. In addition, each token object contains the exact location of its string; the complete token sequence is used as a base leaf sequence for the parsing tree.

The set of predefined regular expressions is large enough to match a wide range of COBOL terms from different versions of the language including special forms of embedded statements, such as CICS, SQL, etc.

The lexical analyzer does not impose strike rules on the source program format, and relates to it as a virtual space with strings to be matched. However, it does recognize and handle the various areas (sequential number, area A and B, etc.).

Together with the symbol table, the lexical analyzer splits the terms from the input program into syntactic categories. Special internal terms such as white space and comments are also linked into the token sequence but are not passed to the syntactic analysis phase.

At the end of the process this sequence is actually a map of the source code file in the main memory. This sequence is used as the basis for most of the operations that will process this file.

Syntactic Analysis

The purpose of this unit is to build a parse tree of the source code file based on the selected token provided by the lexical analyzer.

The syntactic analysis is conducted by a large set of context free grammars. This collection of grammars is designed to catch a wide range of COBOL versions, COBOL dialects and embedded statements from other supplementary packages such as CICS, SQL, etc.

The parse tree nodes are designed using a special strategy that expresses grammatical equivalence and reduces and simplifies the semantic analysis and therefore the translation process and results.

The parse tree is fully object oriented and implements smart inheritance relations between its nodes. The main principle of the strategy is to design the inheritance relation in such a way that properties with major semantics will be at the bases of the inheritance structures, while nodes that add minor semantics will be derived from these bases. This strategy also helps to split the different parts of the code between different nodes thus helping to build good structured software, where each node is an object that knows its own semantics.

As in the previous phase, this phase also creates a data structure, i.e., a parse tree for later use. Splitting the code into different objects simplifies the translation scheme and improves code readability and reusability.

Semantic Analysis

Semantics analysis captures the semantics of the different object structures, the meaning of their position in the source code file and the way they relate to each other. An additional purpose is to understand the ideas that hide behind the code. These ideas are expressed by the parse tree and the token sequence.

Understanding the programming techniques that were used by the original COBOL programmers and implemented in the code can help later in the translation phase by producing much more readable and efficient Java code.

There are three main reasons why it is useful for the compiler to understand the programming techniques that were used in the source program:

- *To facilitate the change in the software architecture*. Legacy COBOL source programs were programmed using concepts of structured procedural programming style while Java source code is based upon object-oriented programming and event-driven programming style.
- *To retain the ideas behind the code*. This information helps in generating new code that keeps these ideas and by doing so makes it easier for the programmers to understand the differences between the old and the new parts of code. In addition it helps them to find their way through the many newly generated lines and modules.
- *To improve the methods used by the original programmers*. Not all programs were written clearly and efficiently. Furthermore, during years of maintenance, COBOL code frequently becomes a patchwork of fixes and original written in different styles. However, grouping the

knowledge of programming techniques in sophisticated software can help to transform inconsistent techniques into consistent and efficient ones.

Refining the Parsing Results

This process reviews and analyzes the repository in order to fine tune **MineIT** decisions.

The legacy system analysis provides a conceptual description of the system that can be enhanced by adding new functionality or making changes that improve efficiency. These enhancements may be achieved by using the compiler's Decisions Mechanism.

Creating Java Objects

The translation phase completes the compilation, and translates the data structures built during the previous phases.

The translation is directed by a predefined collection of schemes that specify the base structures of the resulting new Java objects.

3. Business Rules Extraction

The Business Rules mining process consists of two phases:

- Locating candidates for business rules
- Porting them to Java

Locating Candidates for Business Rules

In general, a business rule can be regarded as specific functionality that is related to a piece of data. This definition of a business rule is similar to an object, which has methods (i.e., functionality) and data. The business rule may have input data with which it will operate, as well as output data, which serves for returning its functionality results. The input data may come through data files or input screens, where the data is input manually. The output data may be data files, reports, screens and/or other output methods.

Using this definition of business rules, with this terminology, two basic methods can be used in order to locate them:

- Start with data, either input data or output data, and then locate the business rule functionality. Examples of this type of process are:
    - ◊ Looking for different parts of the code, which handles the same data, since different parts of code that are referencing the same data may be related to the same business rule.
    - ◊ Selecting a part of the data that serves as a result at a specific point of the code and searching backwards for all influencing code.
    - ◊ Selecting a part of the data that serves as input at a specific point of the code, and searching forward for all the references to that part.
- Start from the opposite direction—with the functionality—and then locate the data. An example of this type of process is:
    - ◊ Looking for different parts of the code that have a similar flow. This similarity may indicate belonging to the same business rule.

MineIT provides several tools for applying locating methods:

- *High Level Flow Graph* – describes the flow and interrelations between the objects composing the application, such as programs, data files, screens, etc.
- *Low Level Flow Graph* – describes the inner flow of a specific program and the interrelations of its paragraphs with other programs, data files, screens, etc.
- *Pattern Matching* – enables the user to indicate a pattern of flow, access or any other phenomenon in the program and search for matching occurrences.
- *Data Locating* –locates usage of a specific data item (taking into consideration the different variable names used for the same data).

Porting Business Rules to Java

Once identified, the business rules can be ported to Java using the migration capabilities of **MineIT**.

During the porting process, the **MineIT** tool generates Java classes, methods and variables from the COBOL application, based on information gathered in the previous phase of locating the object candidates.

There are several porting rules regarding that are given as suggestions by the **MineIT** mechanism, and may be selected by the user:

- *External references from the mined section* – In cases where the selected section includes a GO TO or a PERFORM statement, the user should decide whether or not to generate that statement in the Java Object. Another related decision is whether or not the referenced part should be ported as well.
- *External resource references* – If the section that was selected uses resources such as files or screens, a decision must be made either to generate that reference in the Java code or to pass that information to and from the object as parameters.
- *Variable references* – Variables that are found in the selected section can become local variables of the object or parameters to the object. The user can accept or override the automatic suggestion of the tool, which is based on the usage of those variables in other parts of the code where they were found.
- *Naming conventions* – During the porting of the code, the user may keep or override names of variables and external references. This becomes especially useful and important when porting code from different programs into the same object, as the same data can be referenced differently in each of the programs. An example is naming Social Security Number differently (SSN, SSNO, SSNUM, etc.) when in fact all these names refer to the same piece of information.

4. Advantages of Using Intercomp's MineIT

There are several solutions for renewing applications:

- Face-lifting the application
- Rewriting the application
- Reusing parts of the application

While face lifting is a suitable solution in some cases, the core of the application is left as it was before: obsolete, hard to maintain, not flexible to changes, and not naturally integrated with the new e-commerce environment.

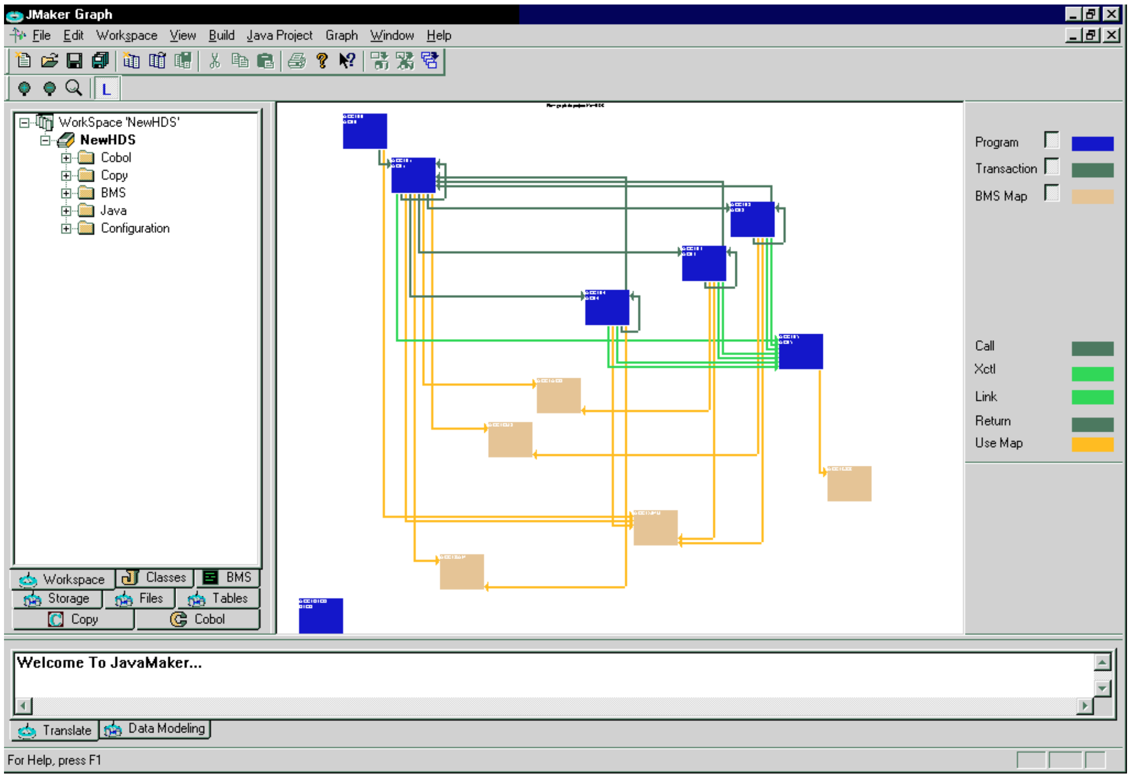
A comparison between rewriting an application and mining for the purpose of reusing the application components, reveals several differences:

- *Time to market* – The time it takes to rewrite an application is significantly longer than mining and reusing parts of the application. As time to market is a very crucial aspect of any company's existence and competitiveness in the fast-paced IT world, the difference between these two solutions may be crucial.
- *Cost* – The cost of rewriting an application is significantly greater than mining and reusing parts of it, since rewriting involves much more effort and time.
- *Risk* –Rewriting an application poses a much greater risk to the organization than mining and reusing. The risk factor involves different aspects of a rewriting project :
    - ◊ The chances that such a project will be completed successfully is estimated to be a 20% probability.
    - ◊ When porting business rules, using automated compiler and compilation techniques, instead of rewriting an application manually, the chance of err is significantly lower. Testing of the automatically ported business rules also is easier and safer.
- "*Reinventing the wheel*" – When rewriting an application, it is in fact being reinvented, including parts that do not need to be rethought since their functionality has not changed (i.e., business rules). Using MineIT enables benefiting from both worlds: on the one hand, designing the new application according to an object oriented methodology, and on the other hand reusing low-level objects from the old application.

Appendix: MineIT Process Work Flow

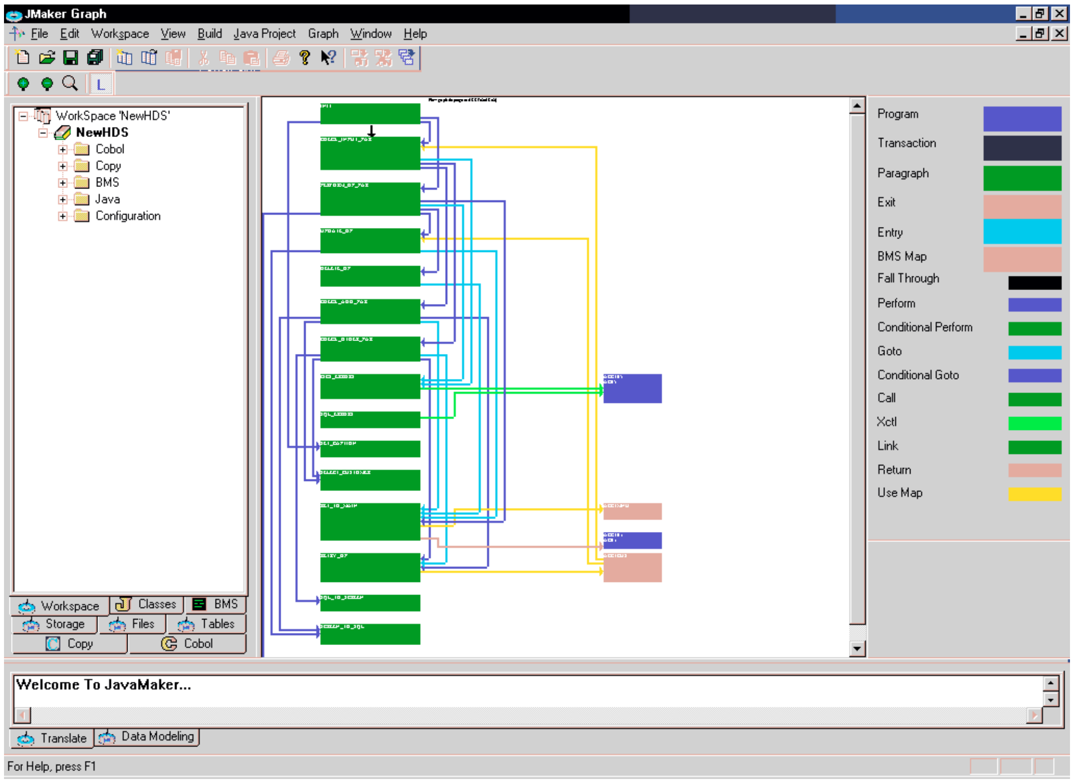
The following sequence of screens demonstrates the work flow for the **MineIT** process:

High Level Graph

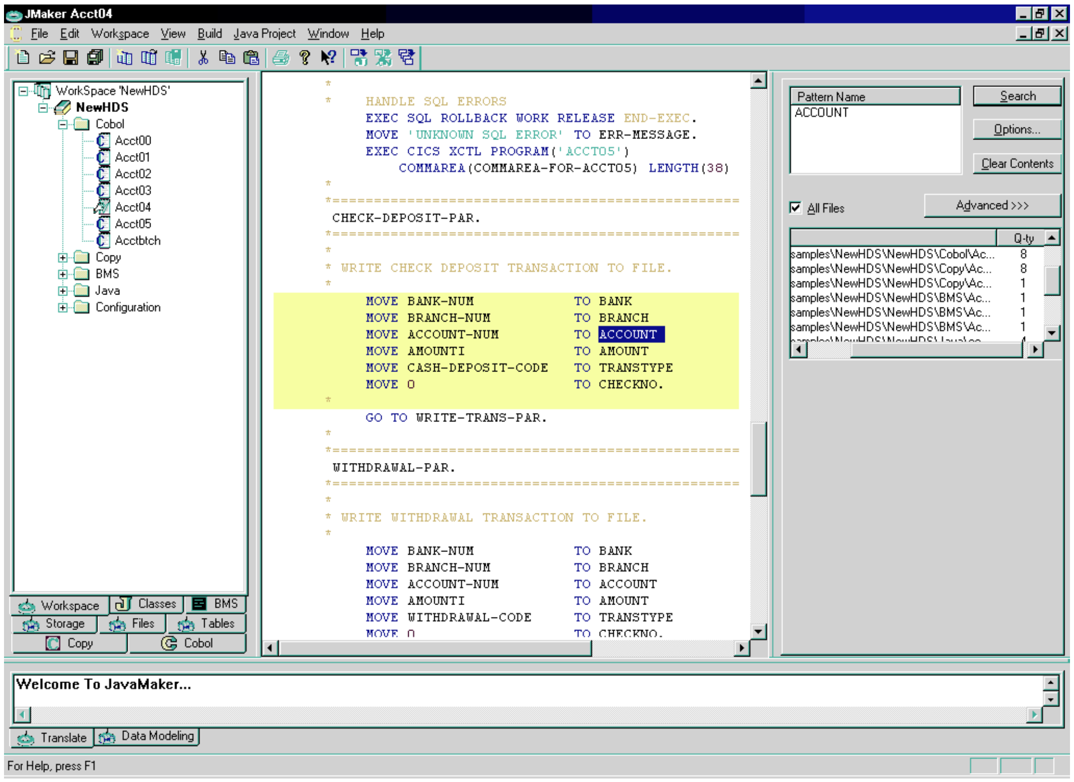


Welcome To JavaMaker...

Low Level Graph



Find Pattern—Using "ACCOUNT" as Hook to Find Business Rule

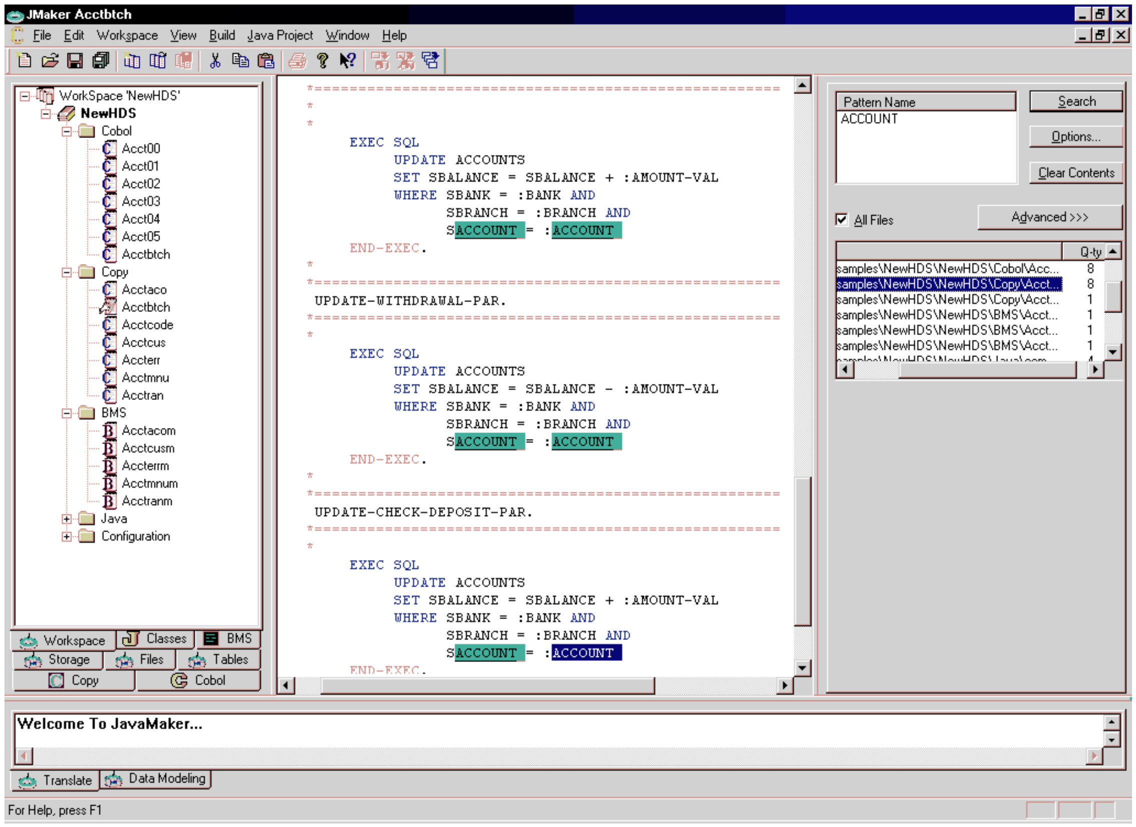


```
      *
      *    HANDLE SQL ERRORS
           EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
           MOVE 'UNKNOWN SQL ERROR' TO ERR-MESSAGE.
           EXEC CICS XCTL PROGRAM('ACCT05')
               COMMAREA(COMMAREA-FOR-ACCT05) LENGTH(38)
      *
      *=====================================================
       CHECK-DEPOSIT-PAR.
      *=====================================================
      *
      * WRITE CHECK DEPOSIT TRANSACTION TO FILE.
      *
           MOVE BANK-NUM             TO BANK
           MOVE BRANCH-NUM           TO BRANCH
           MOVE ACCOUNT-NUM          TO ACCOUNT
           MOVE AMOUNTI              TO AMOUNT
           MOVE CASH-DEPOSIT-CODE    TO TRANSTYPE
           MOVE 0                    TO CHECKNO.
      *
           GO TO WRITE-TRANS-PAR.
      *
      *=====================================================
       WITHDRAWAL-PAR.
      *=====================================================
      *
      * WRITE WITHDRAWAL TRANSACTION TO FILE.
      *
           MOVE BANK-NUM             TO BANK
           MOVE BRANCH-NUM           TO BRANCH
           MOVE ACCOUNT-NUM          TO ACCOUNT
           MOVE AMOUNTI              TO AMOUNT
           MOVE WITHDRAWAL-CODE      TO TRANSTYPE
           MOVE 0                    TO CHECKNO.
```

Pattern Name: ACCOUNT

```
samples\NewHDS\NewHDS\Cobol\Ac...   8
samples\NewHDS\NewHDS\Copy\Ac...    8
samples\NewHDS\NewHDS\Copy\Ac...    1
samples\NewHDS\NewHDS\BMS\Ac...     1
samples\NewHDS\NewHDS\BMS\Ac...     1
samples\NewHDS\NewHDS\BMS\Ac...     1
```

Find Pattern—List of Occurrences Throughout the Application; Select Item From List to Show Source



```
*===========================================================
*
*
     EXEC SQL
          UPDATE ACCOUNTS
          SET SBALANCE = SBALANCE + :AMOUNT-VAL
          WHERE SBANK = :BANK AND
                SBRANCH = :BRANCH AND
                SACCOUNT = :ACCOUNT
     END-EXEC.
*
*===========================================================
 UPDATE-WITHDRAWAL-PAR.
*===========================================================
*
     EXEC SQL
          UPDATE ACCOUNTS
          SET SBALANCE = SBALANCE - :AMOUNT-VAL
          WHERE SBANK = :BANK AND
                SBRANCH = :BRANCH AND
                SACCOUNT = :ACCOUNT
     END-EXEC.
*
*===========================================================
 UPDATE-CHECK-DEPOSIT-PAR.
*===========================================================
*
     EXEC SQL
          UPDATE ACCOUNTS
          SET SBALANCE = SBALANCE + :AMOUNT-VAL
          WHERE SBANK = :BANK AND
                SBRANCH = :BRANCH AND
                SACCOUNT = :ACCOUNT
     END-EXEC.
```

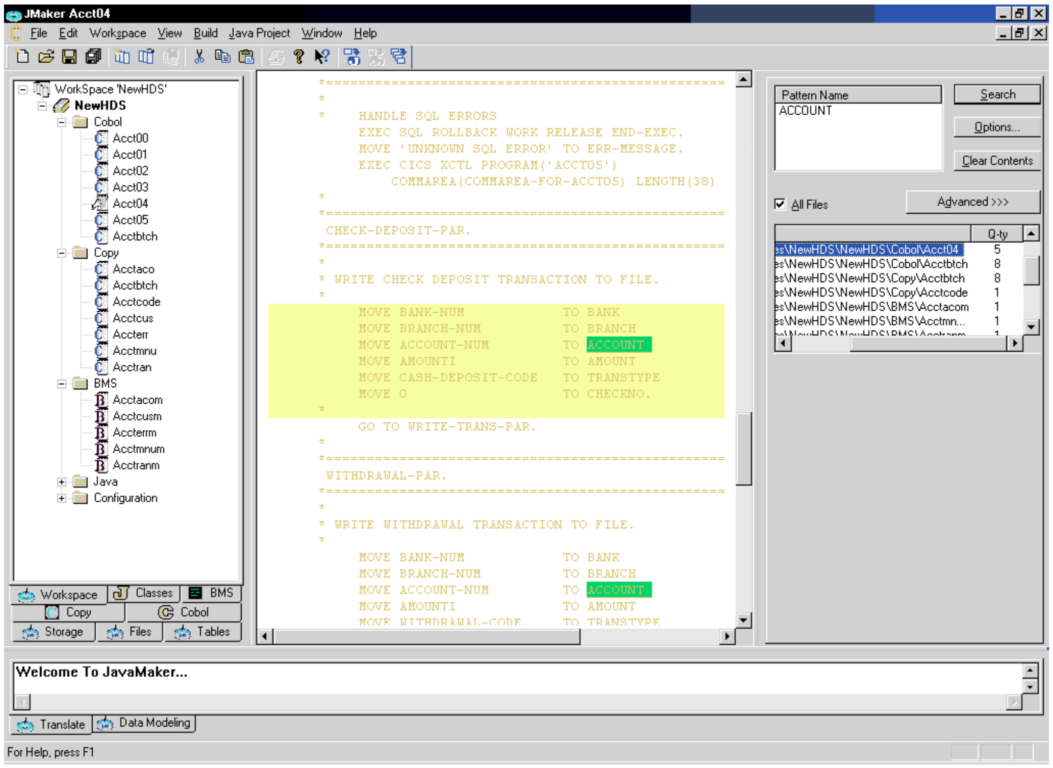
Pattern Name: ACCOUNT

[✓] All Files

	Q-ty
samples\NewHDS\NewHDS\Cobol\Acc...	8
samples\NewHDS\NewHDS\Copy\Acct...	8
samples\NewHDS\NewHDS\Copy\Acct...	1
samples\NewHDS\NewHDS\BMS\Acct...	1
samples\NewHDS\NewHDS\BMS\Acct...	1
samples\NewHDS\NewHDS\BMS\Acct...	1

Welcome To JavaMaker...

Find Pattern—Selection for Creating New Method



```
      *=========================================================
      *
      *    HANDLE SQL ERRORS
           EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
           MOVE 'UNKNOWN SQL ERROR' TO ERR-MESSAGE.
           EXEC CICS XCTL PROGRAM('ACCT05')
               COMMAREA(COMMAREA-FOR-ACCT05) LENGTH(38)
      *
      *=========================================================
       CHECK-DEPOSIT-PAR.
      *=========================================================
      *
      * WRITE CHECK DEPOSIT TRANSACTION TO FILE.
      *
           MOVE BANK-NUM              TO BANK
           MOVE BRANCH-NUM            TO BRANCH
           MOVE ACCOUNT-NUM           TO ACCOUNT
           MOVE AMOUNTI               TO AMOUNT
           MOVE CASH-DEPOSIT-CODE     TO TRANSTYPE
           MOVE 0                     TO CHECKNO.
      *
           GO TO WRITE-TRANS-PAR.
      *
      *=========================================================
       WITHDRAWAL-PAR.
      *=========================================================
      *
      * WRITE WITHDRAWAL TRANSACTION TO FILE.
      *
           MOVE BANK-NUM              TO BANK
           MOVE BRANCH-NUM            TO BRANCH
           MOVE ACCOUNT-NUM           TO ACCOUNT
           MOVE AMOUNTI               TO AMOUNT
           MOVE WITHDRAWAL-CODE       TO TRANSTYPE
```

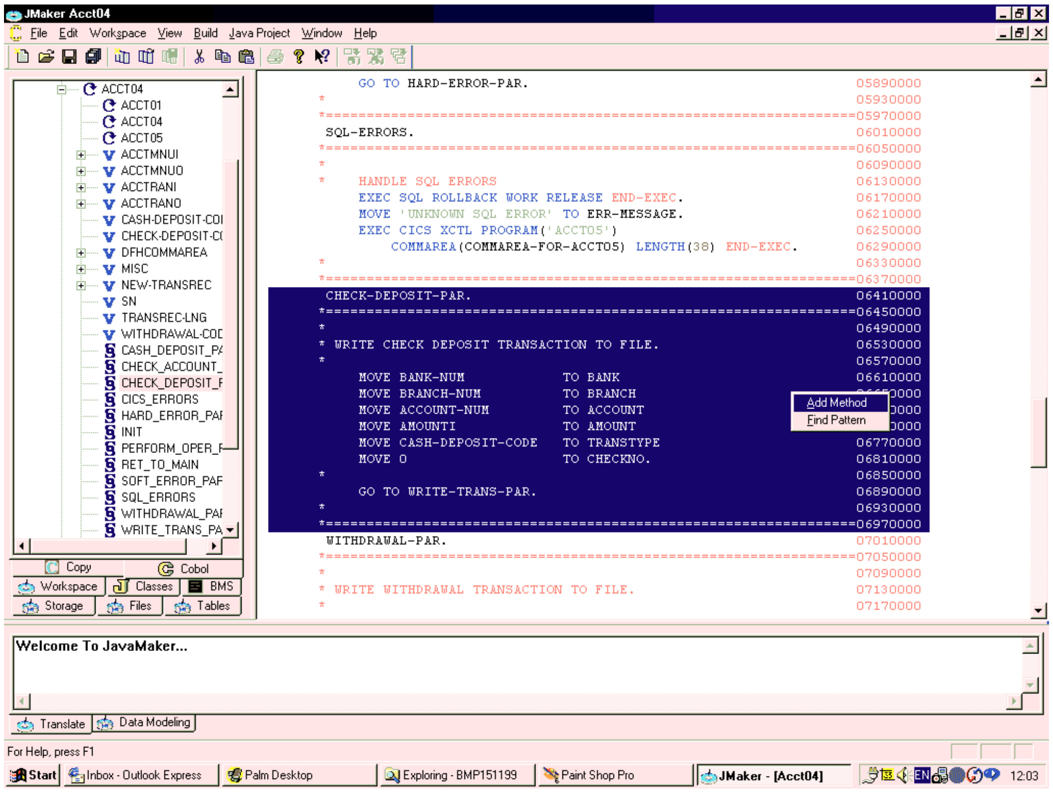
Pattern Name: ACCOUNT

[✓] All Files

	Q-ty
es\NewHDS\NewHDS\Cobol\Acct04	5
es\NewHDS\NewHDS\Cobol\Acctbtch	8
es\NewHDS\NewHDS\Copy\Acctbtch	8
es\NewHDS\NewHDS\Copy\Acctcode	1
es\NewHDS\NewHDS\BMS\Acctacom	1
es\NewHDS\NewHDS\BMS\Acctmn...	1

Welcome To JavaMaker...

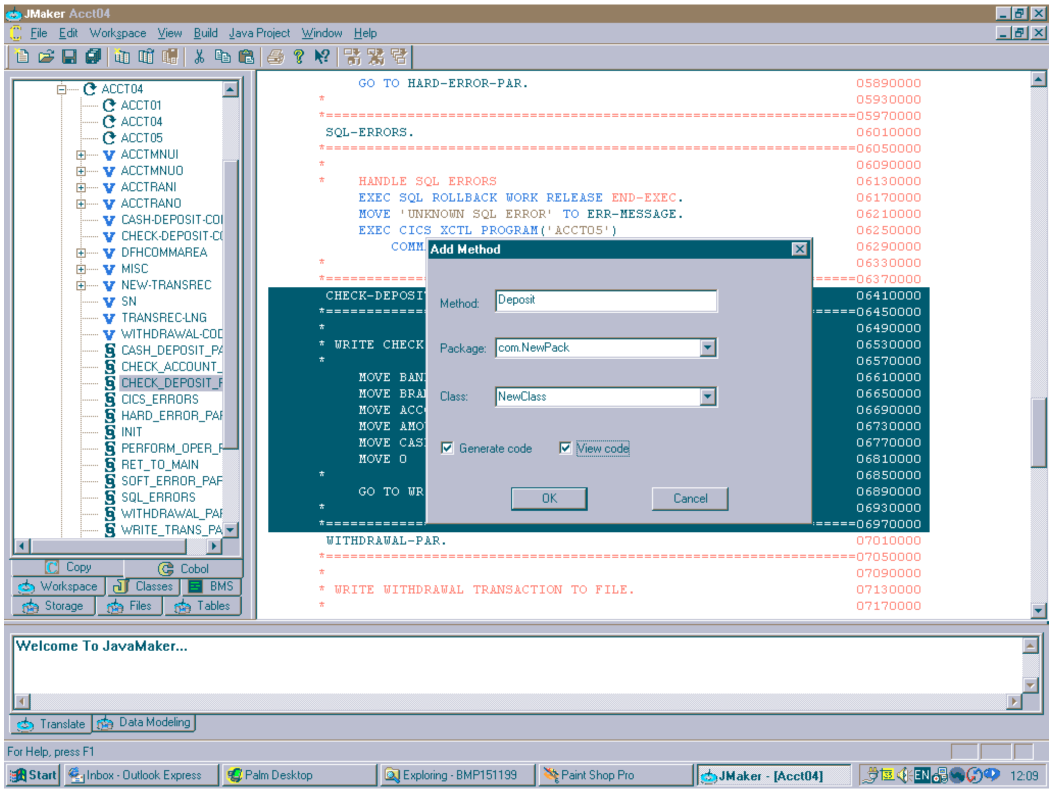
Create New Object—Adding New Method



```
           GO TO HARD-ERROR-PAR.
      *
      *==============================================================
       SQL-ERRORS.
      *==============================================================
      *
      *    HANDLE SQL ERRORS
           EXEC SQL ROLLBACK WORK RELEASE END-EXEC.
           MOVE 'UNKNOWN SQL ERROR' TO ERR-MESSAGE.
           EXEC CICS XCTL PROGRAM('ACCT05')
               COMMAREA(COMMAREA-FOR-ACCT05) LENGTH(38) END-EXEC.
      *
      *==============================================================
       CHECK-DEPOSIT-PAR.
      *==============================================================
      *
      * WRITE CHECK DEPOSIT TRANSACTION TO FILE.
      *
           MOVE BANK-NUM              TO BANK
           MOVE BRANCH-NUM            TO BRANCH
           MOVE ACCOUNT-NUM           TO ACCOUNT
           MOVE AMOUNTI               TO AMOUNT
           MOVE CASH-DEPOSIT-CODE     TO TRANSTYPE
           MOVE 0                     TO CHECKNO.
      *
           GO TO WRITE-TRANS-PAR.
      *
      *==============================================================
       WITHDRAWAL-PAR.
      *==============================================================
      *
      * WRITE WITHDRAWAL TRANSACTION TO FILE.
      *
```

Welcome To JavaMaker...

Create New Object—Defining New Method

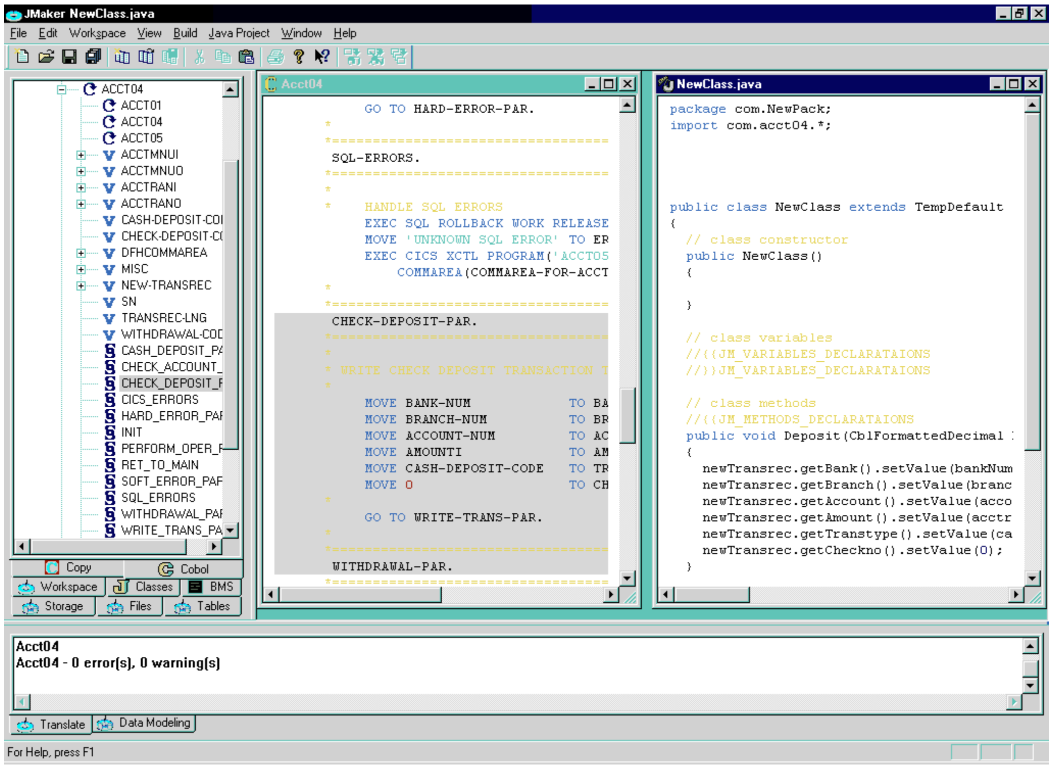


```
         GO TO HARD-ERROR-PAR.                                  05890000
      *                                                          05930000
      *=========================================================05970000
       SQL-ERRORS.                                               06010000
      *=========================================================06050000
      *                                                          06090000
      *      HANDLE SQL ERRORS                                   06130000
         EXEC SQL ROLLBACK WORK RELEASE END-EXEC.                06170000
         MOVE 'UNKNOWN SQL ERROR' TO ERR-MESSAGE.                06210000
         EXEC CICS XCTL PROGRAM('ACCT05')                        06250000
             COMM                                                06290000
      *                                                          06330000
      *===========                                        =====06370000
       CHECK-DEPOSI                                              06410000
      *===========                                        =====06450000
      *                                                          06490000
      * WRITE CHECK                                              06530000
      *                                                          06570000
         MOVE BAN                                                06610000
         MOVE BRA                                                06650000
         MOVE ACC                                                06690000
         MOVE AMO                                                06730000
         MOVE CAS                                                06770000
         MOVE 0                                                  06810000
      *                                                          06850000
         GO TO WR                                                06890000
      *                                                          06930000
      *===========                                        =====06970000
       WITHDRAWAL-PAR.                                           07010000
      *=========================================================07050000
      *                                                          07090000
      * WRITE WITHDRAWAL TRANSACTION TO FILE.                    07130000
      *                                                          07170000
```

Method: Deposit
Package: com.NewPack
Class: NewClass
[✓] Generate code [✓] View code

Welcome To JavaMaker...

Create New Object—COBOL Code vs. New Method in Java



```
         GO TO HARD-ERROR-PAR.
      *
      *=================================
       SQL-ERRORS.
      *=================================
      *
      *      HANDLE SQL ERRORS
         EXEC SQL ROLLBACK WORK RELEASE
         MOVE 'UNKNOWN SQL ERROR' TO ER
         EXEC CICS XCTL PROGRAM('ACCT05
             COMMAREA(COMMAREA-FOR-ACCT
      *
      *=================================
       CHECK-DEPOSIT-PAR.
      *=================================
      *
      * WRITE CHECK DEPOSIT TRANSACTION T
      *
         MOVE BANK-NUM                TO BA
         MOVE BRANCH-NUM              TO BR
         MOVE ACCOUNT-NUM             TO AC
         MOVE AMOUNTI                 TO AM
         MOVE CASH-DEPOSIT-CODE       TO TR
         MOVE 0                       TO CH
      *
         GO TO WRITE-TRANS-PAR.
      *
      *=================================
       WITHDRAWAL-PAR.
      *=================================
```

```java
package com.NewPack;
import com.acct04.*;



public class NewClass extends TempDefault
{
  // class constructor
  public NewClass()
  {

  }

  // class variables
  //{{JM_VARIABLES_DECLARATAIONS
  //}}JM_VARIABLES_DECLARATAIONS

  // class methods
  //{{JM_METHODS_DECLARATAIONS
  public void Deposit(CblFormattedDecimal :
  {
    newTransrec.getBank().setValue(bankNum
    newTransrec.getBranch().setValue(branc
    newTransrec.getAccount().setValue(acco
    newTransrec.getAmount().setValue(acctr
    newTransrec.getTranstype().setValue(ca
    newTransrec.getCheckno().setValue(0);
  }
```

Acct04
Acct04 - 0 error(s), 0 warning(s)

Intercomp Ltd.

6 Maskit Street
Herzliya 46733
Israel

Tel: +972-9-9526777
Fax: +972-9-9526170
E-mail: info@cobol2java.com
Web Site: www.cobol2java.com