

Caché Java バインディングガイド

(Caché Version 5.1 ベース)

V1.0.0

2006 年 4 月

インターシステムズジャパン株式会社

1. はじめに

本ガイドは、Caché と Java の連携を中心に、Java 側から Caché へアクセスするためのプログラミングの基本を解説します。本ガイドでは、ユーザの皆さんが Caché と Java での開発を始めるのに必要な事項を説明していきます。評価版の Caché を実際にインストールして操作しながらお読みいただければ、理解が深まると思います。

また、本ガイドで使用するクラス群は Caché オブジェクト操作ガイドで使用するクラスと同じものも使用します。クラス定義の方法、サーバサイドプログラミングにご興味のある方は Object 編もお読みになることをお勧めいたします。

なお、本ガイドには、Caché のインストールの解説は含まれていません。インストール方法解説については、弊社ダウンロードサイト内、解説書: [Cache51PCKitForWindowsInstallationJPN.pdf](#) をご覧ください。

本ガイドをやり終えれば、Caché と Java を連携して開発をスタートするのに必要な知識が得られます。さらに情報が必要な方は、オンライン・ドキュメントをご参照ください。

このガイドでは、Caché と Java、それぞれのクラスを以下のように呼ぶことにします。

Caché クラス – 主に Caché スタジオ上で定義される Caché のクラス

Java プロキシクラス – Caché の Java プロジェクションによって自動生成される Java クラス

Java クラス – ユーザが作成する Caché にアクセスするためのプログラムを記述した Java クラス

2. Java Binding

2.1 Java バインディングとは

Caché Java バインディングにより、Java アプリケーションは Caché サーバに含まれるオブジェクトと相互運用できます。Caché クラス定義に、**Java プロジェクション**を定義すると Caché は、Caché クラスにアクセスするための Java プロキシクラスを生成します。Java プロキシクラスはサーバ上の Caché メソッドに一致するメソッドと、オブジェクト・プロパティのアクセスするためのメソッドを含みます。Java プログラムでは、これらの Java プロキシクラスは通常の Java クラスと同じものとして認識します。生成された Java プロキシクラスは、インターシステムズが提供する Java ライブラリ(CacheDB.jar)を使用し、クライアントとサーバ間の通信を行います。

詳細は以下のステップで解説します。

2.2 Java バインディングの準備

2.2.1 Java 環境のセットアップ

Java クライアントの実行環境として、Java SDK のバージョン 1.3 以降を使用する必要があります。

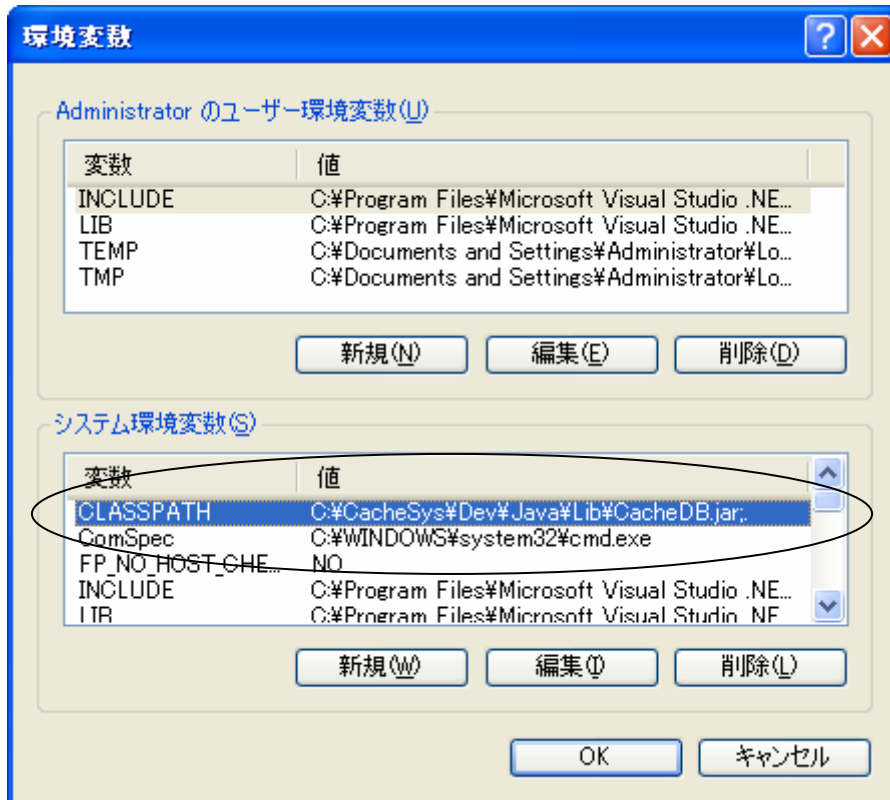
Caché の接続に必要な Java ライブラリは、ファイル CacheDB.jar に含まれます。Windows システムの場合、このファイルは C:\%cachesys%\Dev\java\lib\CacheDB.jar にあります。この jar ファイルに含まれる情報の詳細は C:\%cachesys%\Dev\java\doc\index.html から参照することができます。

Java 環境では、システム環境変数 CLASSPATH を使用して、機能に必要なファイルを見つけます。この変数に CacheDB.jar のロケーションを含む必要があります。

(SDK 1.3.x を使用している場合、*jdbc2_0-stdext.jar* も必要になります。詳細はオンラインドキュメントをご参照ください)

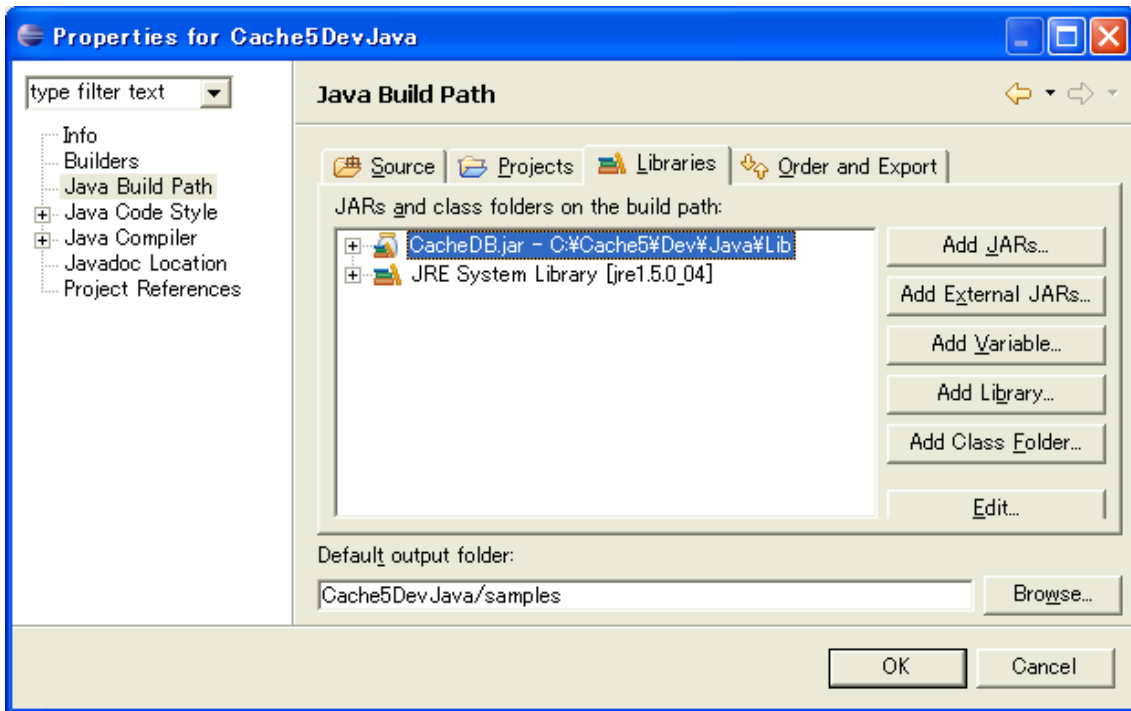
お使いの環境のシステム環境変数にあらかじめ、CLASSPATH を設定しておくことをお勧めします。Windows の場合 コントロールパネル→システム→詳細設定 タブ→環境変数 から指定できます。このとき必ず以下のように値の部分に「.」(ピリオド)も含めるようにしてください。

変数	値	
CLASSPATH	C:%CacheSys%Dev%Java%Lib%CacheDB.jar;. ←セミコロンの上にピリオド追加	



2.2.2 Java IDE 環境のセットアップ

開発に Java IDE 環境を利用している場合、外部 JAR として CacheDB.jar を追加する必要があります。Eclipse3.1 の場合は以下ようになります。



2.2.3 Sample プログラムを動かしてみよう。

Caché をインストールすると、いくつかのサンプル Java プログラムもインストールされます。(デフォルトインストールの場合は、C:\%CacheSys%\Dev\Java\samples にあります。)

このプログラムは Caché の SAMPLES ネームスペースに接続してデータをやり取りします。環境の確認もかねて、このサンプルを動かしてみましよう。

(以下、インストールディレクトリを C:\%Cachesys として記述しています。)

コマンドプロンプトを起動し、C:\%CacheSys%\Dev\Java\samples に移動します。

Person クラスのデータを Caché から取得し、画面に出力する CJTest3.java を実行してみましよう。

:実行結果

```
C:¥CacheSys¥Dev¥Java¥samples>java CJTest3
```

```
Connected.
```

```
ID: 1
```

```
Name: Waterman,Rob Z.
```

```
SSN: 859-67-7992
```

```
DOB: 1973-03-25
```

```
Age: 33
```

```
Street: 8468 Ash Court
```

```
City: Reston
```

```
State: NC
```

```
Zip: 87865
```

2.2.4 Java Projection の設定

Java Binding のための Java プロキシファイルを出力するためには、二つの方法があります。ひとつは、スタジオで Projection ウィザードを使用して Java Projection を設定することです。

「クラス」→「追加」→「新規プロジェクション」 から 「新規プロジェクションウィザード」を使用します。

Projection 名を決定し、次ページにて Java ファイルを出力するディレクトリ(ROOTDIR パラメータ)を指定します。それ以外のパラメータについては Caché Class Reference を参照してください。

もうひとつは、\$SYSTEM.OBJ.ExportJava()関数を使用する方法です。

ターミナルから

```
USER>Do $SYSTEM.OBJ.ExportJava("Shop.Customer","C:¥Temp")
```

を実行します。対象クラス、出力ディレクトリは任意に指定できます。

どちらの場合も、そのオブジェクトと関連があるクラス(参照、継承元など)はすべて出力されません。

2.3 Java バインディングアクセス方法

この章では Java からどのようにして Caché のオブジェクトにアクセスするのか、順に説明してい

きます。

2.3.1 パッケージのインポート

Java Binding を行うには、CacheDB.jar で提供されている com.intersys.objects パッケージをインポートする必要があります。

```
import com.intersys.objects.*;
```

2.3.2 Java から Caché への接続

Java から Cache'へアクセスするためには、Caché と接続する必要があります。

接続のためには、Database オブジェクトを用意し、CacheDatabase クラスの getDatabase メソッドを実行します。

```
Database      dbconnection = null;  
String        url="jdbc:Cache://localhost:1972/SAMPLES";  
  
dbconnection = CacheDatabase.getDatabase (url, username, password);
```

より軽量で高速なアクセス (Java Light Binding) を提供する getLightDatabase メソッドも存在します。

```
dbconnection = CacheDatabase.getLightDatabase (url, username, password);
```

ただしこの方法では、Caché サーバプロセス側にインスタンスが生成されないため、Caché クラスに定義したインスタンスメソッドが呼び出せないなど、いくつかの制限があります。

2.3.3 Java から Caché のデータを取得するには

ここからは Sample の CJTest*.java クラスから各プロパティへのアクセス方法を、ピックアップして解説します。皆さんの環境でも実際に動かしながら確認してみてください。

2.3.3.1 通常のプロパティ

通常のプロパティはそのままドットシンタックスを使用してデータ取得が可能です。ただし、get メソッド、set メソッドを使用します。

```
//データの取得 (CJTest3.java)
/* Fetch some properties */
System.out.println( "ID: " + person.getId() );
System.out.println( "Name: " + person.getName() );
System.out.println( "SSN:  " + person.getSSN() );

//データの設定 (CJTest2.java)
/* Set some properties */
person.setName("Doe, Joe A");
person.setSSN(generateSSN ());
```

2.3.3.2 埋め込みオブジェクト

埋め込みオブジェクトのデータを取得する場合には、まず埋め込みオブジェクトのインスタンスを生成し、通常のプロパティと同様に get メソッドを使用します。また、ドットシンタックスを使用して直接参照することも可能です。

```
//インスタンス生成後、参照する場合 (CJTest3.java)
/* Attempt to bring in an embedded object */
addr = person.getHome();
System.out.println( "Street: " + addr.getStreet() );
System.out.println( "City:  " + addr.getCity() );
System.out.println( "State: " + addr.getState() );
System.out.println( "Zip:   " + addr.getZip() );

//次のように直接参照も可能です。
System.out.println( "State: " + person.getHome().getState() );
System.out.println( "Zip:   " + person.getHome().getZip() );
```

変更も同様です。

```
//インスタンス生成後、変更する場合 (CJTest4.java)
```

```
/* Modify some values */
```

```
addr.setStreet( "One Memorial Drive" );
```

```
addr.setCity( "Cambridge" );
```

```
addr.setState( "MA" );
```

```
addr.setZip( "02142" );
```

```
//次のように直接変更も可能です。
```

```
person.getHome().setStreet( "One Memorial Drive" );
```

```
person.getHome().setCity( "Cambridge" );
```

2.3.3.3 Collection プロパティ 1 (List/Array)

List/Array のプロパティにアクセスするには以下の 2 種類の方法があります。

1. Java で一般的な `java.util.List` / `java.util.Map` を利用する。
2. Caché が提供する `ListOfObjects` / `ArrayOfObjects` を利用する。

Java Projection の `NEWCOLLECTION` パラメータが 1 の場合は 1. の形式で出力され、0 の場合は 2. の形式で出力されます。現在の `NEWCOLLECTION` パラメータのデフォルトは 1 になっており、ここではそちらを利用したアクセス方法を紹介します。

参照は以下のように行います。

```
//リスト数を取得してループ (CJTest5.java)
```

```
/* Iterate over the FavoriteColors collection */
```

```
colors = person.getFavoriteColors();
```

```
for (r = 0; r < colors.size(); r++) {
```

```
    System.out.println(" Element #" + r + " -> " + colors.get(r));
```

```
}
```

次は変更のサンプルです。

```
//最初のデータを削除して新しいデータ(Red)を追加 ///(CJTest5.java)
/* Remove the first element */
if (colors.size() > 0)
    colors.remove(0);

/* Insert a new element */
colors.add("Red");
```

Iterator を使用して以下のようにアクセスすることも可能です。

```
//Iteratorオブジェクトを使用してアクセス
/* Iterate over the FavoriteColors collection */
colors = person.getFavoriteColors();
Iterator ci = colors.iterator();
r=0;
while(ci.hasNext()){
    r++;
    System.out.println(" Element #" + r + " -> " + ci.next().toString());
}
```

2.3.3.4 Collection プロパティ 2 (Relationship)

Sample には Relationship を実装したクラスはないので、ここでは Caché オブジェクト操作ガイドで使用したクラスにアクセスする方法を示します。

(ここでは親:子の関係を使って説明しますが、一:多の関連の場合も同様です。)

< Caché オブジェクト操作ガイドのクラス定義をインポートする場合は、弊社 Web サイトの「技術ガイド」のページより、サンプルをダウンロードしインポートしてください >

Caché 技術ガイド : <http://www.intersystems.co.jp/support/guide.html>

子から親のデータを参照、更新する場合、子から見て親は一意に決定できるので、通常のプロパティと同じようにアクセス可能です。

```
//直接親のデータを参照
customer = po.getCustomer();
customer.getName();
//子から親を設定
po.setCustomer(cust);
```

親から子のデータを取得する場合、子の集合の Java データタイプ RelationshipObject を使用します。このクラスは java.util.Map インターフェースを実装しているので、通常の Map オブジェクトとしてもアクセスが可能です。以下は Iterator を使って子供の要素を次々に取得する方法です。

```
//Iteratorを使用してLineItemを取得
Iterator oi = po.getItems().values().iterator();
while (oi.hasNext()) {
    Shop.LineItem lineitem = (Shop.LineItem) oi.next();
    lineitem.getUnitPrice();
    ...
}
```

2.3.3.5 Caché のメソッドを実行する

Caché 内で定義されたメソッドを実行することもできます。

クラスメソッドを実行する場合、引数に Database オブジェクトが必要です。

```
//クラスメソッドの例：POOrderクラスのcreateメソッドを呼び出す
Shop.POrder po
= (Shop.POrder) Shop.POrder.create(dbconnection,cust,shipto);
```

インスタンスメソッドの場合は、接続先は明らかですので、Database オブジェクトは必要あり

ません。

```
//インスタンスメソッドの例:POrder クラスの place メソッドを呼び出す。  
//この場合は dbconnection は必要ありません。  
po.place();
```

2.3.3.6 Caché のクエリを実行する。

Caché クラスに定義されたクエリを呼び出し実行することができます。

CacheQuery オブジェクト、java.sql.ResultSet オブジェクトを用意し、実行します。

この例では Sample.Person クラスの ByName クエリを実行しています。

```
CacheQuery          cq = null;  
java.sql.ResultSet  rs = null;
```

```
//Cache'クエリを作成し、実行結果を ResultSet に格納する  
// (CJTest6.java)  
/* Create a ResultSet */  
System.out.println( "Creating a ResultSet" );  
/* Create a CacheQuery */  
cq = new CacheQuery( server, "Sample.Person", "ByName" );  
/* Execute the query and loop across the returned rows */  
rs = cq.execute(query);  
while (rs.next()) {  
    /* Dump the columns in each row */  
    String s = "";  
    for (int i = 1; i <= rs.getMetaData().getColumnCount(); i++) {  
        if (s.length() > 0) {  
            s += " ";  
        }  
        s += rs.getString( i );  
    }  
    System.out.println( s );  
}
```

動的にクエリを生成して実行することもできます。

SQL 文を作成する。

```
/* Create the SQL statement */  
sql = "SELECT ID, Name, DOB, SSN FROM Sample.Person WHERE  
      Name %STARTSWITH ?";
```

```
//同様に Cache'クエリを作成し、実行結果を ResultSet に格納する  
// (CJTest7.java)  
/* Create a CacheQuery */  
cq = new CacheQuery( server, sql );  
/* Execute the query and loop across the returned rows */  
rs = cq.execute(query);  
while (rs.next()) {  
    /* Dump the columns in each row */  
    String s = "";  
    for (int i = 1; i <= rs.getMetaData().getColumnCount(); i++) {  
        if (s.length() > 0) {  
            s += " ";  
        }  
        s += rs.getString( i );  
    }  
    System.out.println( s );  
}
```

2.4 Java Method

Caché クラス中に Java 言語を用いたメソッドを書くことができます。これは Java メソッドと呼ばれ、Projection された Java クラス内に出力されます。Caché 上ではこのメソッドを単独で実行することはできません。

Java メソッド内では Java 言語を用いて、ロジックを記述しなければいけません。

次章にて Java メソッドを使用したサンプルプログラムを作成します。

```
Method JavaPrintName(tab As %Integer) [ Language = java ]
{
    try {
        System.out.println( "名前" + this.getName() );
    } catch(Exception ex) {
        System.out.println( "Caught exception: "
            + ex.getClass().getName() + ": " + ex.getMessage() );
        ex.printStackTrace();
    }
}
```

3. Java アプリケーションを作成する

それでは、実際に Java アプリケーションを作成してみましょう。

ここでは

Caché オブジェクト操作ガイド のオブジェクトを利用したアプリケーション
デザインパターンとして有名な Composite パターンを実装したアプリケーション

を作成します。

3.1 Caché クラスのインポート / Java プロキシクラスを出力しよう

ここで使用するクラス群は Caché オブジェクト操作ガイドで使用するクラスと同じものです。すでにこのガイドを終了されている方は、インポートの必要はありません。

まだの方は、弊社 Web サイトから Caché オブジェクト操作ガイド用サンプルをダウンロードしてください。(<http://www.intersystems.co.jp/support/guide.html>)

注意) 以下作成する Java ファイルは **ネームスペース USER に接続する** として作成しています。別のネームスペースへサンプルをインポートした方は Java ファイル内 Caché への接続部分を対象のネームスペースに修正し、お試しください。

2.2.4 で説明したように、Java Binding のために Java プロキシクラスを生成する必要があります。今回はすべてのクラスに Java Projection を追加し、コンパイルを実行してください。

```
Projection JavaProjection As %Projection.Java;
```

ROOTDIR パラメータを明示的に指定しない場合、Java Binding ファイルは C:\%CacheSys%\Devuser\Java\<ネームスペース名>\ディレクトリに出力されます。

3.2 Java クライアントプログラムを作成しよう 1

このクライアントプログラムは Caché クラス POrder のインスタンス一覧を表示するものです。

(Caché クラスの Shop.POrder printOrderByDate メソッドと同じような内容です)

それでは

```
public class PrintOrder {  
  
    public static void main(String[] args) {  
    }  
}
```

ここから 1 ステップずつプログラムを作成していきましょう

- ・ com.intersys.object のインポート
インターシステムズの提供しているパッケージをインポートする必要があります。その他にも後で必要となるパッケージをここでインポートしておきます。

```
import com.intersys.objects.*;  
  
import java.util.*;  
import java.sql.SQLException;  
  
public class PrintOrder {
```

- ・ 変数の宣言

接続に必要な Database オブジェクト、および url,user,password を設定します。

また、結果を格納する Shop.POrder, クエリに必要な CacheQuery,ResultSet もここで作成しておきましょう。

(接続ユーザ、パスワードはインストール時用意される事前定義のユーザである_system を利用しています。ユーザ名などはお使いの環境に合わせて適宜変更してお試ください。なお、ユーザ管理については以下ドキュメントより詳細をご参照いただけます。

http://127.0.0.1:8972/csp/docbook/DocBook.UI.Page.cls?KEY=GCAS_users)

```
public static void main(String[] args) {  
  
    Database          dbconnection = null;  
    String            url="jdbc:Cache://localhost:1972/USER";  
    String            username="_SYSTEM";  
    String            password="SYS";  
  
    Shop.POrder po = null;  
    CacheQuery        cq=null;  
    java.sql.ResultSet rs=null;  
}
```

- ・ Database コネクションの作成、クエリの実行

CacheDatabase.getDatabase メソッドを実行してコネクションを確立します。

また、Shop.POrder の Extent クエリを作成し、実行した結果を ResultSet に格納します。

(Extent クエリについて: Extent クエリは Cache クラスで自動的に生成されるすべてのインスタンスを返すクエリです。)

```
try {  
  
    dbconnection = CacheDatabase.getDatabase (url, username, password);  
  
    cq = new CacheQuery(dbconnection, "Shop.POrder", "Extent");  
    rs = cq.execute();  
}
```

- ・ 結果の取得

ResultSet を while ループでまわし、ID フィールドの値を取得します。Shop.POrder オブジェクトをオープンし、データを取得します。

(オブジェクトをオープンしなくても、クエリの結果に各フィールドは含まれていますが、ここではあえてオブジェクトを使用しています。)

```
while (rs.next()){
    Id id = new Id( rs.getInt("ID"));
    po = (Shop.POrder)Shop.POrder._open( dbconnection, id );

    System.out.println( "【注文番号】:" + po.getOrderNumber() );
    System.out.println( "【顧客名】:" + po.getCustomer().getName() );
    System.out.println( "【合計金額】:" + po.getTotalPrice() );
    System.out.println();
}
```

- ・ Relationship の取得(親から子)

Iterator を使用して、Shop.POrder オブジェクトの子、LineItem オブジェクトを取得します。

```
//Relationship(LineItem:One) のデータ取得(Map)
Collection ci = po.getItems().values();
Iterator oi = ci.iterator();

while (oi.hasNext()) {
    Shop.LineItem lineitem = (Shop.LineItem) oi.next();
    System.out.println( "【商品名】:" +
        lineitem.getProduct().getName() );
    System.out.println( "【数量】:" + lineitem.getAmount() );
}
}
```

- ・ オブジェクトの Close

オブジェクトをクローズします。ループが繰り返され、変数 po には次の Shop.POrder オブジェクトが格納されます。

```
dbconnection.closeObject(po.getOref());
po = null;

System.out.println("-----");
```

実行結果は以下のようになります。

```

【注文番号】： 0000000001
【顧客名】： 日本 太郎
【合計金額】： 118000

【商品名】： デスクトップPC Pentium IV 2.6G
【数量】： 1

-----

【注文番号】： 0000000004
【顧客名】： インターシステムズジャパン
【合計金額】： 944000

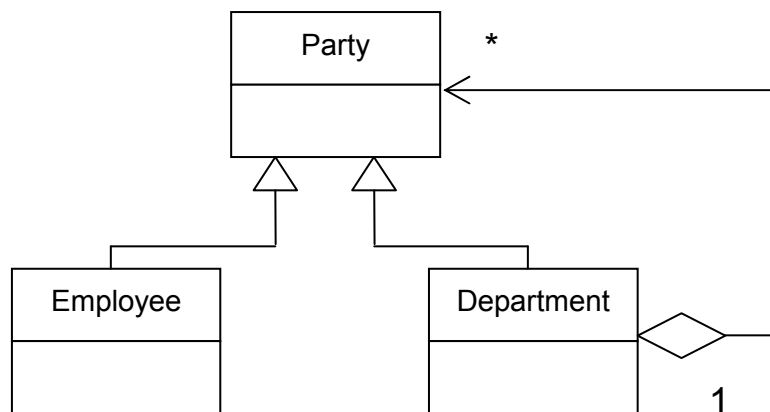
【商品名】： デスクトップPC Pentium IV 2.6G
【数量】： 10

-----
  
```

<ここまでの Java サンプルプログラムは Step1 フォルダ以下にあります。(PrintOrder.java) >

3.3 Java クライアントプログラムを作成しよう 2

次のプログラムでは、代表的なデザインパターンであるコンポジットパターンを実装します。オブジェクト指向言語 Java と、オブジェクトデータベース Caché の相性の良さがご理解いただけます。



まず Caché クラスを定義します。

上位の Party クラスを以下のように定義します。

- Java プロジェクション
- Name プロパティ
- JavaPrintName メソッド (Java メソッド)

のみ含みます。JavaPrintName メソッドの実装は継承先で行うので、ここでは何も記述しません。

```
Class Sample.Party Extends %Persistent
{
  Projection JavaProjection1 As %Projection.Java;
  Property Name As %String;
  Method JavaPrintName(tab As %Integer) [ Language = java ]
  {
  }
}
```

次にこの Party クラスを継承した Employee クラスを定義します。JavaPrintName メソッドを実装します。

```
Class Sample.Employee Extends Sample.Party
{
Method JavaPrintName(tab As %Integer) [ Language = java ]
{
    try {
        //表示位置の調整
        for (int i=1;i<tab.intValue();i++) {
            System.out.print(" ");
        }
        System.out.println( "Emp: " + this.getName() );
    } catch(Exception ex) {
        System.out.println( "Caught exception: " + ex.getClass().getName() +
": " + ex.getMessage() );
        ex.printStackTrace();
    }
}
}
```

最後に、Party クラスを継承した Department クラスを作成します。

このクラスには

- Parties プロパティ

を追加します。このプロパティは Collection タイプで、「Department」か「Employee」のインスタンスを複数保持します。

```
Class Sample.Department Extends Sample.Party
{
Property Parties As Party [ Collection = list ];

Method JavaPrintName(tab As %Integer) [ Language = java ]
{
    try {
        //表示位置の調整
        for (int i=1;i<tab.intValue();i++) {
            System.out.print(" ");
        }
        System.out.println( "Dept: " + this.getName() );
        java.util.Iterator pi = this.getParties().iterator();
        while (pi.hasNext()) {
            Sample.Party party = (Sample.Party) pi.next();
            //下位のコンポーネントの表示位置 +2
            int a = tab.intValue()+2;
            Integer newtab = new Integer(a);
            party.JavaPrintName(newtab);
        }
    } catch(Exception ex) {
        System.out.println( "Caught exception: " + ex.getClass().getName()
+ ": " + ex.getMessage() );
        ex.printStackTrace();
    }
}
}
```

これで Caché クラスの定義は完了です。コンパイルを実行し、Java クラスを取得します。

<ここまでのサンプルクラス定義は、Step1 フォルダ 以下にあります。

(JavaBinding_practice2.xml) >

注意 これから作成する Java ファイルは、**ネームスペース USER を接続対象として**作成していません。別のネームスペースにサンプルクラス定義をインポートする場合には、Caché への接続部分を対象のネームスペースに修正してお試しください。

ターミナルを使用して以下のようにデータを作成します。

Department クラスの Parties プロパティには Insert メソッドを使用してデータを追加します。

サンプルクラス定義を(Step1¥JavaBinding_practice2.xml)インポートされた方は、Sample.Party クラス内クラスメソッド DataPopulate()を実行し作成することもできます。

```
USER> Set top=##class(Sample.Department).%New()

USER> Set top.Name="ABC 株式会社"

USER> Do top.%Save()

USER>

USER> Set dep1=##class(Sample.Department).%New()

USER> Set dep1.Name="営業部"

USER> Do top.Parties.Insert(dep1)

USER> Do dep1.%Save()

USER> Set sp=##class(Sample.Employee).%New()

USER> Set sp.Name="中村主任"

USER> D dep1.Parties.Insert(sp)

USER> Do sp.%Save()

USER>
```

クライアントのコードは以下のようになります。

```
import com.intersys.objects.*;

public class Partymain {

    public static void main( String[] args )
        throws Exception
    {
        Database          dbconnection = null;
        String            url="jdbc:Cache://localhost:1972/USER";
        String            username="_SYSTEM";
        String            password="SYS";
        Sample.Party     party = null;
        Id                id = null;

        try {
            dbconnection = CacheDatabase.getDatabase (url, username, password);

            for (int i = 0; i < args.length; i++)
                if (args[i].equals("-id"))
                    id = new Id (args[++i]);

            if (id == null)
                id = new Id( 1 );

            if (!(Sample.Party.exists (dbconnection, id)))
            {
                System.out.println ("There is no Party with id " +
                    id.toString() + " in the database.");
                dbconnection.close();
                return;
            }
        }
    }
}
```

```
//指定されたIDのインスタンスをオープン
party = (Sample.Party) Sample.Party._open( dbconnection, id );

Integer tab = new Integer(2);

//JavaPrintNameメソッドを呼び出し。
//子要素がある場合は、メソッド内で再帰的に呼び出される
party.JavaPrintName(tab);

dbconnection.closeObject(party.getOref());
party = null;

dbconnection.close();
System.out.println( "Disconnected." );

} catch (CacheException ex) {
    System.out.println( "Caught exception0: " + ex.getClass().getName() + ": " +
ex.getMessage() );
    ex.printFullTrace(System.out);
} catch (Exception ex) {
    System.out.println( "Caught exception0: " + ex.getClass().getName() + ": " +
ex.getMessage() );
    ex.printStackTrace();
}
}
}
```

<ここまでの Java プログラムは Step1 フォルダ以下にあります。(Partymain.java)>

実行結果は以下のようになります。

```
Dept: ABC株式会社
  Dept: 営業部
    Emp: 中村主任
      Dept: 営業 1 課
        Emp: 山田太郎
          Emp: 田中花子
            Dept: 営業2課
              Emp: 島耕作
```

補足: Step1 フォルダにある残りの Java ファイルは、Cache スタジオにてクラス定義コンパイル時に生成される Java ソースコードのサンプルです。