



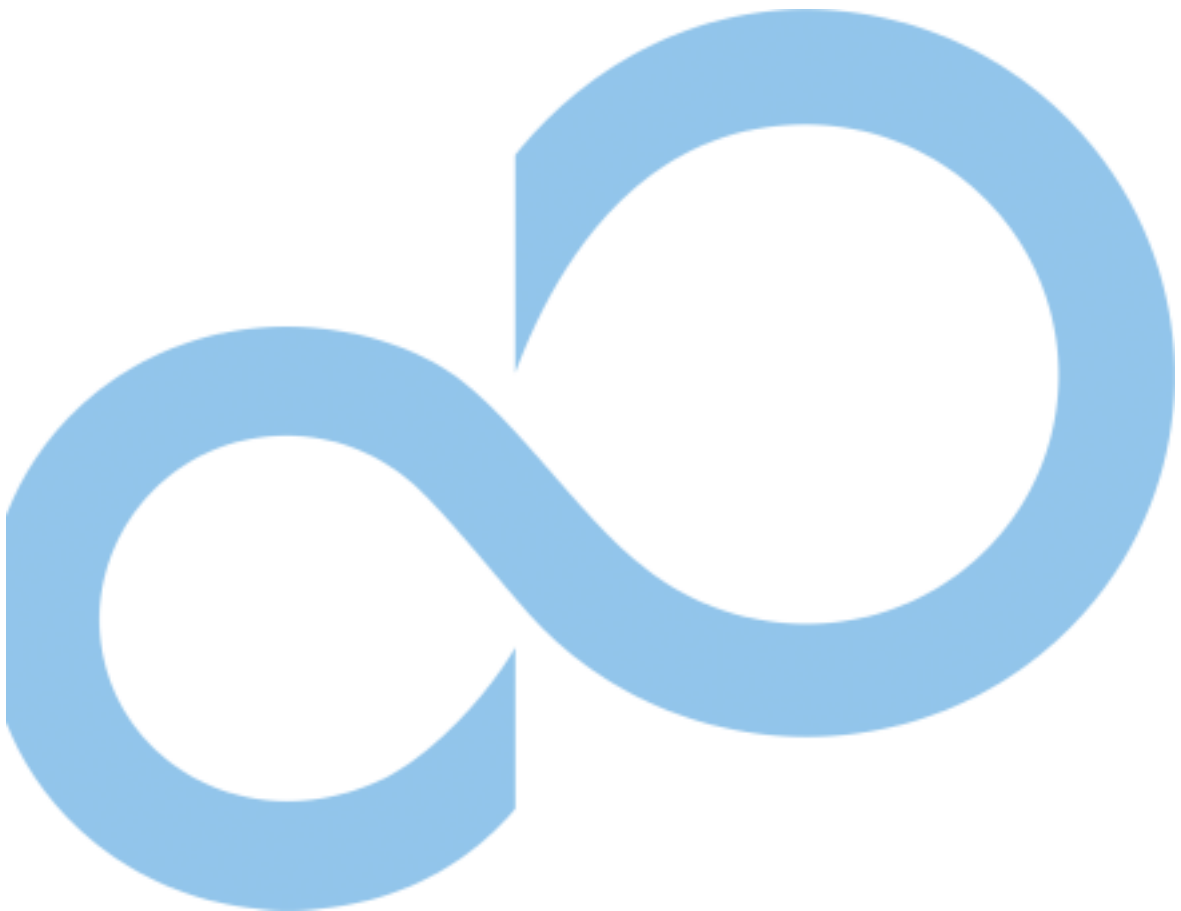
Microsoft® Windows® 95  
Microsoft® Windows® 98  
Microsoft® Windows® Me

Microsoft® Windows NT®  
Microsoft® Windows® 2000  
Microsoft® Windows® XP

B1JW-5521-01Z2

# Ｊアダプタクラスジェネレータ V7.0

## 使用手引書



Net  OBOL

 FUJITSU



---

# まえがき

“Jアダプタクラスジェネレータ”は、Java(TM)クラスを呼び出すCOBOLクラス（アダプタクラス）を生成するツールです。生成したアダプタクラスを使用することにより、COBOLからJavaのクラスライブラリを利用できるようになります。

なお、ジェネレータおよび生成したアダプタクラスを実行するためには、Javaの実行環境がインストールされている必要があります。必要な製品については“[準備するもの](#)”を参照してください。

## 製品の呼び名について

本書に記載されている製品の名称を、以下のように略して表記します。

- 「Microsoft(R) Windows(R) 95 operating system」  
→ 「Windows(R) 95」
- 「Microsoft(R) Windows(R) 98 operating system」  
→ 「Windows(R) 98」
- 「Microsoft(R) Windows(R) Millennium Edition」  
→ 「Windows(R) Me」
- 「Microsoft(R) Windows NT(R) Workstation operating system Version 4.0」  
→ 「Windows NT(R)」または、「Windows NT(R) 4.0」
- 「Microsoft(R) Windows NT(R) Server Network operating system Version 4.0」  
→ 「Windows NT(R)」または 「Windows NT(R) 4.0」
- 「Microsoft(R) Windows NT(R) Server Network operating system, Enterprise Edition Version 4.0」  
→ 「Windows NT(R)」、「Windows NT(R) 4.0」、「Windows NT(R) E.E.」または、「Windows NT(R) Server」
- 「Microsoft(R) Windows(R) 2000 Professional operating system」  
→ 「Windows(R) 2000」または、「Windows(R) 2000 Professional」
- 「Microsoft(R) Windows(R) 2000 Server operating system」  
→ 「Windows(R) 2000」または、「Windows(R) 2000 Server」
- 「Microsoft(R) Windows(R) 2000 Advanced Server operating system」  
→ 「Windows(R) 2000」または、「Windows(R) 2000 Advanced Server」
- 「Windows(R) 95」、「Windows(R) 98」、「Windows(R) Me」、「Windows NT(R)」および「Windows(R) 2000」  
→ 「Windows(R)」

## 本書の目的

本書は、COBOLプログラムからJavaクラスを利用するための、アダプタクラスの作成方法、プログラムの書き方およびその実行方法について説明しています。

COBOLの文法規則については“COBOL 文法書”を参照してください。また、NetCOBOLを使ったプログラム開発方法については“NetCOBOL 使用手引書”を参照してください。

## 本書の対象読者

本書は、Javaクラスを利用するCOBOLプログラムを開発される方を対象としています。

## 前提知識

本書を読むにあたって、以下の知識が必要です。

- COBOLの文法に関する基本的な知識
- COBOLのオブジェクト指向プログラミングに関する基本的な知識
- Javaに関する基本的な知識

## 本書の構成

本書の構成と内容は、以下のとおりです。

---

## **第1章 Jアダプタクラスジェネレータの概要**

Jアダプタクラスジェネレータの機能および動作環境について説明しています。

## **第2章 Jアダプタクラスジェネレータの仕組み**

Jアダプタクラスジェネレータの仕組みについて説明しています。

## **第3章 開発方法**

Javaクラスを使用するプログラムの開発方法について説明しています。

## **第4章 ジェネレータの使い方**

ジェネレータコマンド (java2cob) の使い方について説明しています。

## **第5章 アダプタクラスリファレンス**

Jアダプタクラスジェネレータで提供する、FJ-JAVA-BASEクラス、FJ-JAVA-CONTROLクラス、FJ-JAVA-ERRORクラスおよび生成するアダプタクラスについて、詳細を説明しています。

## **付録A メッセージ一覧**

Jアダプタクラスジェネレータが出力するメッセージの内容および対処方法について説明しています。

## **付録B 例外種別一覧**

Jアダプタクラスジェネレータが発生させる例外の種別およびその対処方法について説明しています。

## **付録C 例題プログラム一覧**

Jアダプタクラスジェネレータを使用したプログラムの例を説明しています。

## **本書の読み方**

Jアダプタクラスジェネレータを初めて利用する場合は、第1章からお読みください。第1章ではJアダプタクラスジェネレータの概要、第2章では仕組み、第3章では開発から実行までの手順を説明しています。

第4章および第5章では、コマンドおよびクラスの使い方を詳細に説明しています。プログラム開発の際に参照してください。

付録AではJアダプタクラスジェネレータが出力するメッセージについて、付録BではJアダプタクラスジェネレータが設定する例外種別について説明しています。必要に応じてお読みください。付録Cでは、Jアダプタクラスジェネレータを使用したプログラムの例を説明しています。必要に応じてお読みください。

## **本書の位置付け**

NetCOBOLシリーズでの本書の位置付け、および関連マニュアルについては、“NetCOBOL解説書”を参照してください。

## **登録商標について**

本書に記載されている登録商標を、以下に示します。

Microsoft、Windows、Windows NTは、米国Microsoft Corporationの米国およびその他の国における登録商標です。

Java およびその他のJavaを含む商標は、米国 Sun Microsystems, Inc. の米国およびその他の国における商標です。

2002年7月

All Rights Reserved, Copyright(C) 富士通株式会社 1999-2002

---

# 目次

<b>第1章</b>	<b>Jアダプタクラスジェネレータの概要</b>	<b>1</b>
1.1	Jアダプタクラスジェネレータとは	2
1.2	できること	4
1.3	できないこと	5
1.4	準備するもの	6
1.5	エンハンス機能	7
<b>第2章</b>	<b>Jアダプタクラスジェネレータの仕組み</b>	<b>9</b>
2.1	アダプタクラス	10
2.2	アダプタオブジェクト	11
<b>第3章</b>	<b>開発方法</b>	<b>13</b>
3.1	アダプタクラスの作成	14
3.1.1	Javaクラスの調査	14
3.1.2	アダプタクラスのソース生成	14
3.1.3	アダプタクラスの構築	15
3.1.4	クラスファイルがない場合の生成方法	15
3.1.5	アダプタクラスのサイズ縮小	16
3.2	アダプタクラスを利用するアプリケーションの開発	18
3.2.1	プログラムの書き方	18
3.2.2	プログラムの構築	25
3.3	プログラムの実行	26
3.4	JAVA2への移行	27
<b>第4章</b>	<b>ジェネレータの使い方</b>	<b>29</b>
4.1	起動方法	30
4.2	オプションファイル	33
4.3	出力	38
4.3.1	アダプタクラスのソースファイル	38
4.3.2	生成名管理ファイル	38
4.3.3	メソッド名対応表ファイル	39
<b>第5章</b>	<b>アダプタクラスリファレンス</b>	<b>41</b>
5.1	クラス構成	42
5.2	FJ-JAVA-BASEクラス	43
5.2.1	J-NARROWメソッド (ファクトリメソッド)	43
5.2.2	J-DUPLICATEメソッド (オブジェクトメソッド)	43
5.2.3	J-EQUALSメソッド (オブジェクトメソッド)	43
5.3	FJ-JAVA-CONTROLクラス	45
5.3.1	JVM-INITメソッド (ファクトリメソッド)	45
5.3.2	JVM-TERMINATEメソッド (ファクトリメソッド)	45
5.3.3	JVM-ATTACHメソッド (ファクトリメソッド)	46
5.3.4	JVM-DETACHメソッド (ファクトリメソッド)	46
5.4	FJ-JAVA-ERRORクラス	47
5.4.1	GET-MESSAGEメソッド (オブジェクトメソッド)	47
5.4.2	GET-CODEメソッド (オブジェクトメソッド)	47
5.4.3	GET-EXCEPTIONメソッド (オブジェクトメソッド)	47
5.5	クラス/インタフェースのアダプタクラス	49
5.5.1	データ型	49

---

5.5.2	クラス／インタフェース.....	50
5.5.3	コンストラクタ.....	51
5.5.4	クラス変数.....	52
5.5.5	クラスメソッド.....	54
5.5.6	インスタンス変数.....	55
5.5.7	インスタンスメソッド.....	56
5.6	JAVA-LANG-STRINGクラス.....	58
5.6.1	NEW-STRING-Xメソッド (ファクトリメソッド) .....	58
5.6.2	NEW-STRING-Nメソッド (ファクトリメソッド) .....	59
5.6.3	GET-STRING-Xメソッド (オブジェクトメソッド) .....	59
5.6.4	GET-STRING-Nメソッド (オブジェクトメソッド) .....	60
5.6.5	GET-STRING-LENGTH-Xメソッド (オブジェクトメソッド) .....	60
5.6.6	GET-STRING-LENGTH-Nメソッド (オブジェクトメソッド) .....	60
5.7	配列のアダプタクラス.....	61
5.7.1	配列クラス.....	61
5.7.2	NEW-ARRAYメソッド (ファクトリメソッド) .....	62
5.7.3	GET-ARRAY-LENGTHメソッド (オブジェクトメソッド) .....	63
5.7.4	GET-ARRAY-ELEMENTメソッド (オブジェクトメソッド) .....	63
5.7.5	SET-ARRAY-ELEMENTメソッド (オブジェクトメソッド) .....	63
5.8	名前の番号付け.....	65
5.8.1	名前の有効範囲.....	65
5.8.2	生成規則により必ず一意になる名前.....	65
5.8.3	スーパークラス・サブクラス間で一意にする名前.....	65
5.8.4	実行単位全体で一意にする名前.....	66
付録A	メッセージ一覧.....	69
A.1	JAVA2COBコマンドのメッセージ.....	69
A.2	生成時のメッセージ.....	69
A.3	実行時のメッセージ.....	72
付録B	例外種別一覧.....	77
付録C	例題プログラム一覧.....	81

---

# 第1章 Jアダプタクラスジェネレータの概要

---

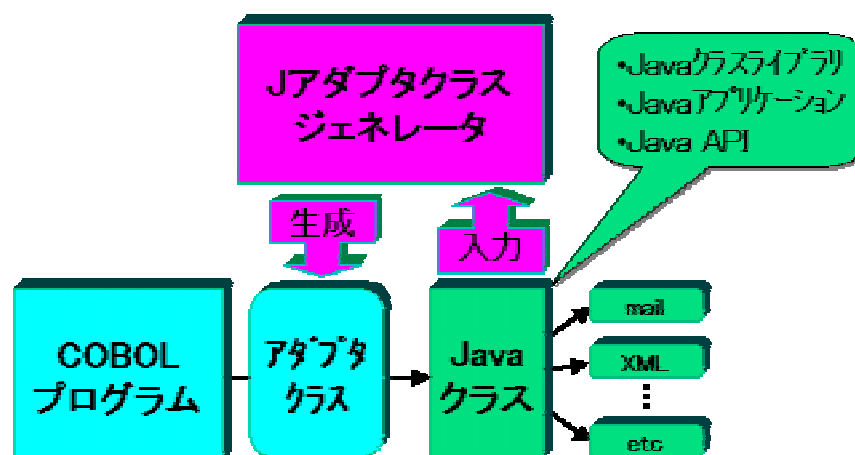
この章では、Jアダプタクラスジェネレータの機能および動作環境について説明します。

## 1.1 Jアダプタクラスジェネレータとは

NetCOBOLでは、オブジェクト指向機能により、さまざまな機能のクラスライブラリを利用したプログラミングが可能です。NetCOBOLシリーズでも、ファウンデーションクラスとして、多くの有用なクラスを提供しています。その一方で、最近では、Javaの普及に伴いJavaのクラスライブラリも多く提供されています。しかし、クラスの構造は言語ごとに異なるため、COBOLからJavaのクラスライブラリは利用できません。

Jアダプタクラスジェネレータは、COBOLからJavaのクラスを利用する機構を提供します。

Jアダプタクラスジェネレータの概要を以下に示します。



COBOLプログラムからJavaのクラスを利用するためには、JavaクラスのインタフェースをCOBOLインタフェースに変換する必要があります。Jアダプタクラスジェネレータは、JavaインタフェースをCOBOLインタフェースに変換するアダプタクラス（COBOLソース）を生成します。

Jアダプタクラスジェネレータを使用することにより、以下が可能となります。

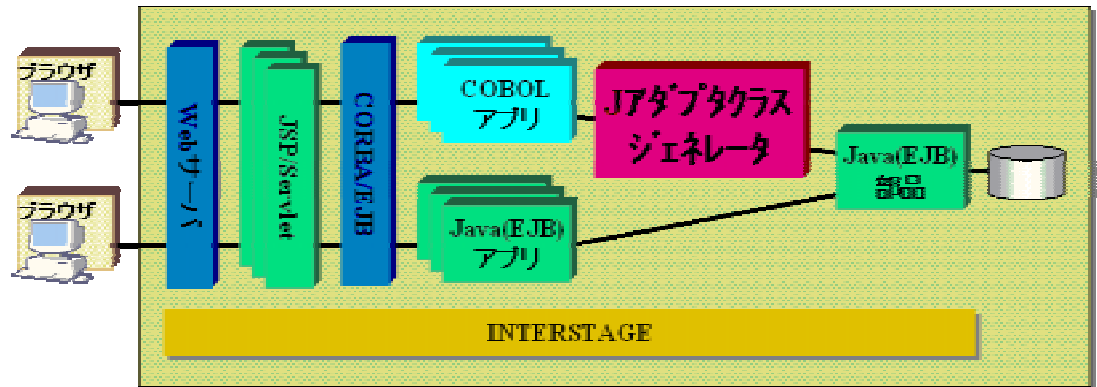
- COBOLプログラムから、Javaのクラスライブラリを利用する
- COBOLプログラムから、Javaアプリケーションを呼び出す
- COBOLプログラムから、Java用に提供されたAPI（Application Program Interface）を利用する

これにより、これまでJavaでしか実現できなかったシステムにも、COBOLを適用できるようになります。

Jアダプタクラスジェネレータは、以下のような場面での使用をお勧めします（下図参照）。

- EJBコンポーネント等のJava共通部品を利用したCOBOLシステムを構築する
- Javaのクラスライブラリ、Javaアプリケーション、およびJava用に提供されたAPIを活用したCOBOLシステムを構築する





## 1.2 できること

Jアダプタクラスジェネレータで生成したアダプタクラスを使うことにより、Javaに対する以下の操作が可能になります。

COBOLプログラムからはJavaオブジェクトはCOBOLオブジェクトのように見えます。そのため、一般のCOBOLオブジェクトを扱う場合と同じ方法でJavaオブジェクトを操作できます。

- クラス変数へのアクセス  
Javaのクラスで宣言した、パブリックなクラス変数（スタティックフィールド）にアクセスできます。COBOLからは、ファクトリのプロパティとして扱います。
- クラスメソッドの呼出し  
Javaのクラスで宣言した、パブリックなクラスメソッド（スタティックメソッド）を呼び出せます。COBOLからは、ファクトリメソッドとして扱います。
- インスタンスオブジェクトの生成（コンストラクタの呼出し）  
コンストラクタを呼び出すことにより、Javaのインスタンスオブジェクトを生成できます。COBOLからは、オブジェクトを返すファクトリメソッドとして扱います。
- インスタンス変数へのアクセス  
Javaインスタンスオブジェクトの、パブリックなインスタンス変数（スタティックでないフィールド）にアクセスできます。COBOLからは、オブジェクトのプロパティとして扱います。
- インスタンスメソッドの呼出し  
Javaインスタンスオブジェクトの、パブリックなインスタンスメソッド（スタティックでないメソッド）を呼び出せます。COBOLからは、オブジェクトメソッドとして扱います。
- 例外の受取り  
クラスメソッド、コンストラクタおよびインスタンスメソッドを呼び出した際に発生する例外を受け取り、エラー処理を行うことができます。COBOLでは、USE文で例外オブジェクトを受け取ります。

## 1.3 できないこと

Jアダプタクラスジェネレータでは、以下のような使い方はできません。

- Javaクラスの継承  
Javaクラスを継承したCOBOLクラスを定義することはできません。アダプタクラスを継承したCOBOLクラスを定義しても、Javaクラスの機能を上書きできません。
- COBOLオブジェクトの受け渡し  
メソッドを呼び出す際のパラメタとしてCOBOLオブジェクトを渡すことはできません。また、Javaのフィールドに対し、COBOLオブジェクトを設定することはできません。Javaとやり取りできるオブジェクトは、Javaオブジェクトをラッピングしたアダプタオブジェクトだけです。  
そのため、以下のような使い方はできません。
  - リスナー  
Javaでは、イベント発生時の処理を記述したリスナーオブジェクトを、イベントを発生するオブジェクトに登録します。しかし、COBOLオブジェクトをJavaオブジェクトに登録できないため、COBOLでリスナーを記述することはできません。
  - コレクションクラス  
Javaのコレクションクラスには、COBOLオブジェクトに登録できません。COBOLオブジェクトを集合として扱う場合は、COBOLのコレクションクラスを使用してください。
- 日本語名標を持つクラス  
クラス名、フィールド名またはメソッド名に日本語を含むクラスは使用できません。
- JavaからCOBOLの呼出し  
JavaからCOBOLプログラムを呼び出すことはできません。また、Javaから呼び出されたCOBOLプログラムからは、アダプタクラスを使用できません。
- クラスリテラル  
クラスリテラルは、“クラス名.class”、“インタフェース.class”のようなプリミティブ型から構成される式です。これはClassという型のオブジェクト、名前の付いた型のクラスオブジェクトを評価します。  
Jアダプタクラスジェネレータでは、クラスリテラルを直接使用できません。このため以下のどちらかの方法で回避してください。
  - java.lang.ClassLoaderクラスのloadClassメソッドを使用して、クラスリテラルに対応したクラスオブジェクトを取得する。
  - クラスリテラルを使用するJavaのクラスを作成し、このJavaクラスのアダプタクラスを生成する。

## 1.4 準備するもの

本製品の利用に際して、開発環境および実行環境として以下の製品が必要となります。

### **NetCOBOL Standard EditionまたはNetCOBOL Standard Editionサーバ運用パッケージ**

本製品を使用してプログラムを開発する場合は、NetCOBOL Standard EditionまたはNetCOBOL Professional Editionが必要です。また、本製品で開発したアプリケーションを実行する場合は、NetCOBOL Standard Editionサーバ運用パッケージが必要です。

### **Java開発キットまたは Javaランタイム環境**

Java(TM) Development Kit (以降、JDKと略す) は、Sun Microsystems社が提供するJava開発キットです。JDKは、Javaによるプログラム開発に必要な基本的なはん用クラスライブラリを提供しています。

Java(TM) 2 SDK, Standard Edition (以降、J2SDKと略す) は、Sun Microsystems社が提供するJava 2対応アプリケーションの構築に使用可能なソフトウェア開発キットです。J2SDKは、Java 2によるプログラム開発に必要な基本的なはん用クラスライブラリを提供しています。

Java(TM) Runtime Environment (以降、JREと略す) は、Sun Microsystems社が提供するJDKのランタイム環境です。Javaで開発したプログラムの実行に必要です。

Java(TM) 2 Runtime Environment, Standard Edition (以降、J2REと略す) は、Sun Microsystems社が提供するJava 2対応アプリケーションのランタイム環境です。Java 2で開発したプログラムの実行に必要です。

本製品を使用してプログラムを開発する場合は、JDK 1.1.8以降またはJ2SDK 1.2.2以降が必要です。また、本製品で開発したアプリケーションを実行する場合は、JDK 1.1.8以降、J2SDK 1.2.2以降、JRE 1.1.8以降、またはJ2RE 1.2.2以降が必要です。

## 1.5 エンハンス機能

Jアダプタクラスジェネレータのエンハンス機能は以下で参照できます。

- [V6. 0L10からV6. 1L10へのエンハンス機能](#)
- [V5. 0L10からV6. 0L10へのエンハンス機能](#)

### V5. 0L10からV6. 0L10へのエンハンス機能

- 従来のシフトJISに加えて、Unicode（英数字項目にUTF-8、日本語項目にUCS-2）を利用できるようになりました。  
[起動方法](#)  
[オプションファイル](#)
- 従来のJava1.1系のクラスに加えて、Java2対応のJavaクラスをCOBOLから利用できるようになりました。  
[準備するもの](#)  
[Java2への移行](#)
- アダプタクラスを使用したアプリケーション（WebアプリおよびEJBアプリを除く）がマルチスレッドで動作できるようになりました。  
[マルチスレッドアプリケーションの開発](#)
- アダプタクラスの生成速度が約20倍速くなりました。また、オプションを指定することで、アダプタクラスの生成サイズを約半分に削減できるようになりました。  
[-omオプションまたは“Option ReduceClass”パラメタを指定する](#)

### V6. 0L10からV6. 1L10へのエンハンス機能

- アプリケーションで使用するJavaのメソッド名等を指定することで、アダプタクラス生成サイズを削減できるようになりました。  
[コンストラクタ／メソッド／フィールドを指定する](#)
- アダプタクラスを使用したWebアプリケーションおよびEJBアプリケーションがマルチスレッドで動作できるようになりました。  
[マルチスレッドアプリケーションの開発](#)
- COBOLからJavaへ文字列を受け渡す方法として、従来のStringオブジェクトを生成して受け渡す方法に加えて、COBOLの英数字項目を直接受け渡すことができるようになりました。  
[COBOLの英数字項目を直接受け渡す](#)
- 従来、String型をパラメタとするメソッドおよびString型のフィールドに対し、データ項目長より短い英数字項目および日本語項目の文字列を渡す場合、ユーザアプリケーションで文字列の終端(X"00")を意識して設定する必要がありました。文字列の終端制御により、アダプタクラスで文字列の終端を設定するため、ユーザアプリケーションでは文字列の終端を意識しないでデータ項目長より短い文字列をメソッドに渡せるようになりました。  
[文字列の終端制御](#)
- Jアダプタクラスジェネレータのランタイムでエラーが発生した場合、FJ-JAVA-ERRORクラスの例外オブジェクトを発生するようになりました。  
[エラー処理](#)
- コマンドラインに直接オプションを指定する従来の方法に加えて、オプションをファイルに記述できるようになりました。  
[オプションファイル](#)



---

## 第2章 Jアダプタクラスジェネレータの仕組み

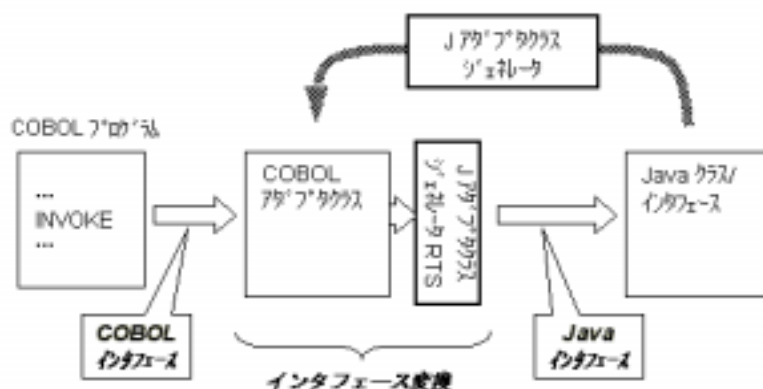
---

この章では、Jアダプタクラスジェネレータの仕組みについて説明します。

## 2.1 アダプタクラス

COBOLからJavaクラスを使用可能にするためには、JavaクラスのインタフェースをCOBOLのインタフェースに変換する機構が必要です。Jアダプタクラスジェネレータは、インタフェース変換機構として、Javaクラスに対応したアダプタクラスを生成します。COBOLプログラムからJavaクラスを使用する際は、生成したアダプタクラスを呼び出します。アダプタクラスはCOBOLで書かれたクラスなので、COBOLのクラスを呼び出すのと同じように呼び出せます。

Javaクラス／インタフェースとアダプタクラスの関係を示します。



Javaクラス／インタフェースとアダプタクラス

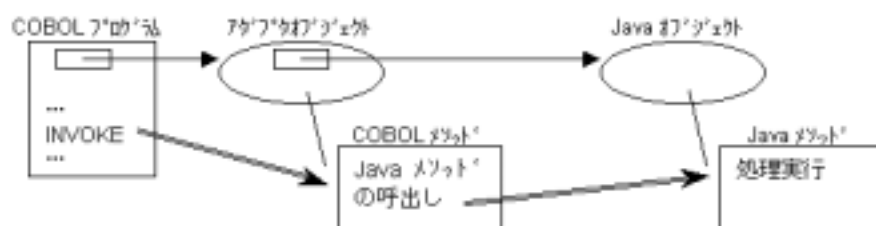


## 2.2 アダプタオブジェクト

アダプタクラスは、実行時にJavaインスタンスオブジェクトに対応するアダプタオブジェクトを生成します。アダプタオブジェクトは、以下の役割を持ちます。

- 対応するJavaインスタンスオブジェクトへのポインタを保持する。
- アダプタオブジェクトのメソッドが呼ばれると、対応するJavaインスタンスオブジェクトの対応するJavaメソッドを呼び出す。

COBOLプログラムから見えるのはアダプタオブジェクトだけです。アダプタオブジェクトへの操作はすべて対応するJavaオブジェクトに伝わります。そのため、COBOLプログラムからは、アダプタオブジェクトがあたかもJavaオブジェクトであるように見えます。アダプタオブジェクトは、Javaオブジェクトの代理の役割を持つので、プロキシ（代理）オブジェクトとも呼びます。Javaオブジェクトとアダプタオブジェクトの関係を以下に示します。



Javaオブジェクトとアダプタオブジェクト



---

## 第3章 開発方法

---

この章では、Javaクラスを使用するプログラムの開発方法について説明します。

## 3.1 アダプタクラスの作成

ここでは、Javaクラスからアダプタクラスを生成する方法について説明します。  
アダプタクラスの生成は、以下の手順で行います。

1. [Javaクラスの調査](#)
2. [アダプタクラスのソース生成](#)
3. [アダプタクラスの構築](#)

また、以下の特殊な生成方法があります。

- [クラスファイルがない場合の生成方法](#)
- [アダプタクラスのサイズ縮小](#)

### 3.1.1 Javaクラスの調査

はじめに、使用したいJavaクラスおよびインタフェースの仕様（クラス名、パッケージ名、使用方法など）を調査します。そして、Jアダプタクラスジェネレータで利用できるクラス／インタフェースであるか調べます。どのようなクラス／インタフェースが利用できるかは、“[できること](#)” および “[できないこと](#)” を参照してください。

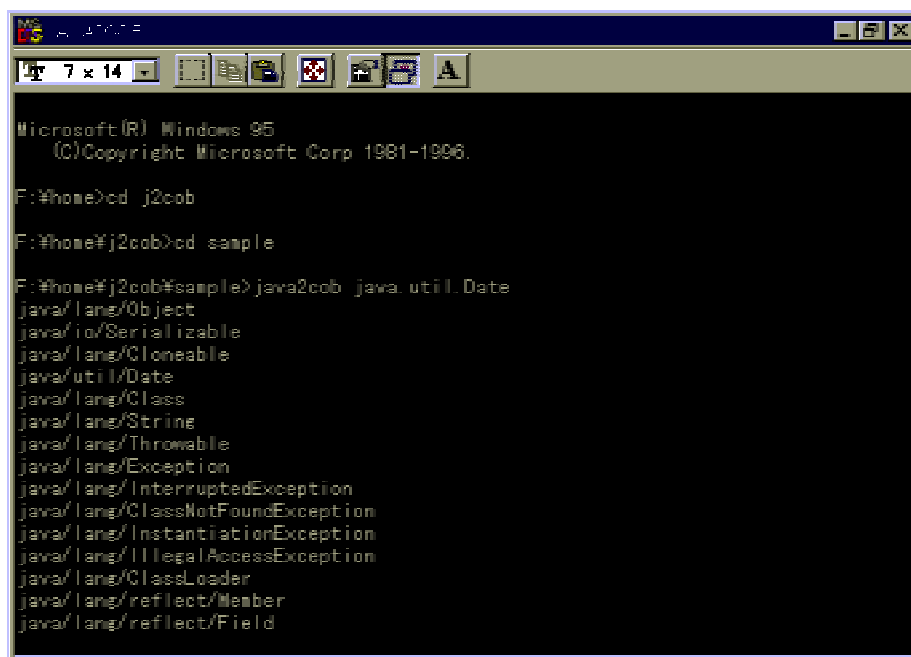
また、アプリケーションで使用するコンストラクタ／メソッド／フィールドが分かる場合で、生成するアダプタクラスのサイズを小さくしたい場合は、使用するコンストラクタ／メソッド／フィールドの仕様（名前、パラメタの型など）を調査します。

### 3.1.2 アダプタクラスのソース生成

目的のJavaクラス／インタフェースが使用可能なら、次にアダプタクラスのソースを生成します。アダプタクラスの生成には、[java2cobコマンド](#)を使用します。java2cobコマンドは、Javaクラス／インタフェースのクラスファイル（拡張子:.class）を読み込み、対応するアダプタクラスのソースを生成します。

[java2cobコマンド](#)は、オプションで指定したクラスだけでなく、そのクラスを使用する際に必要となるすべてのクラス／インタフェースのアダプタクラスを生成します。

java.util.Dateクラスからアダプタクラスのソースを生成する例を下図に示します。



```
Microsoft(R) Windows 95
(C) Copyright Microsoft Corp 1981-1996.

F:\home>cd j2cob
F:\home\j2cob>cd sample
F:\home\j2cob\sample>java2cob java.util.Date
java/lang/Object
java/io/Serializable
java/lang/Cloneable
java/util/Date
java/lang/Class
java/lang/String
java/lang/Throwable
java/lang/Exception
java/lang/InterruptedException
java/lang/ClassNotFoundException
java/lang/InstantiationException
java/lang/IllegalAccessException
java/lang/ClassLoader
java/lang/reflect/Member
java/lang/reflect/Field
```

**補足**

実行環境によっては、クラス／インタフェースから参照するクラス／インタフェースのクラスファイルがすべてそろっていない場合があります。参照するクラスファイルが存在しない場合のアダプタクラス生成方法については、“[クラスファイルがない場合の生成方法](#)”を参照してください。

**3.1.3 アダプタクラスの構築**

最後に、生成したアダプタクラスソースを翻訳・リンクし、アダプタクラスライブラリ（DLL）を作成します。これには、COBOLプロジェクトマネージャを使用します。COBOLプロジェクトマネージャの使い方については、“NetCOBOL 使用手引書”を参照してください。

アダプタクラスの構築は、以下の手順で行います。

1. プロジェクトマネージャで新規プロジェクトを作成します。
2. ターゲットとなるDLLを登録します。
3. COBOLソースファイルフォルダを作成し、生成したアダプタクラスソースを登録します。
4. 翻訳オプションを指定します。
  - REPINには、XXX¥REP（XXXはJアダプタクラスジェネレータのインストールフォルダ）を指定します。
  - ALPHAL（WORD）を指定します。
  - Unicodeで動作するアダプタクラスを作成する場合、RCS（UCS2）を指定します。
  - マルチスレッドアプリケーションを作成する場合、THREAD（MULTI）を指定します。
5. 翻訳順序を決めるために、[編集] メニューの [リポジトリファイル検索] — [すべて] を実行します。
6. ライブラリファイルフォルダを作成し、Jアダプタクラスジェネレータの実行時ライブラリ F3BIJART.LIBを登録します。F3BIJART.LIBは、インストールフォルダのLIBフォルダにあります。
7. ビルドします。

この結果作成される以下のファイルは、アダプタクラスを使用する際に必要になります。

- アダプタクラスのDLLファイル（実行時に必要）
- アダプタクラスのLIBファイル（リンク時に必要）
- アダプタクラスのリポジトリファイル（翻訳時に必要）

**3.1.4 クラスファイルがない場合の生成方法**

Jアダプタクラスジェネレータでは、アダプタクラス生成時に、以下のクラスの情報を参照します。

- スーパークラス／インタフェース
- パブリックメソッド／コンストラクタのパラメタ／復帰値に指定されたクラス／インタフェース
- パブリック変数に指定されたクラス／インタフェース

したがって、これらのクラスファイルが存在しないと、アダプタクラスを正しく生成できません。ただし、環境によってはこれらのクラスファイルが存在しない場合があります。この場合、アダプタクラスを正しく生成するためには、同名のダミーのクラスファイルを用意しておく必要があります。

ダミーのクラスファイルは、以下の手順で作成します。

1. パッケージ名に対応するフォルダを作成します。たとえば、aaa.bbb.cccというパッケージ名なら、aaa¥bbb¥cccというフォルダを作成します。
2. 以下の内容のJavaソースを作成します。ファイル名は“クラス名.java”とします。

```
package パッケージ名;
public class クラス名 { }
```

3. Javaコンパイラで翻訳します。

```
C:\¥> javac フォルダ名¥クラス名.java
```

4. java2cobコマンドを実行します。

```
C:\¥> java2cob パッケージ名.クラス名
```

## 補足

参照するクラスファイルが存在しなかった場合、以下のようになります。

- メッセージ “[クラス情報が見つかりません。生成処理を中止します。](#)” を出力し、処理を中止する

### 3.1.5 アダプタクラスのサイズ縮小

Jアダプタクラスジェネレータが生成したアダプタクラスソースは、そのまま翻訳・リンクするだけで利用可能です。通常は、そのまま運用可能です。

ただし、アダプタクラスソースには、アプリケーションで使用しないものも含まれるため、アダプタクラスのDLLファイルが大きくなる場合があります。このような場合、以下の方法により、アダプタクラスのサイズを小さくすることができます。

- [コンストラクタ／メソッド／フィールドを指定する](#)
- [-omオプションまたは“Option ReduceClass”パラメタを指定する](#)

#### 3.1.5.1 コンストラクタ／メソッド／フィールドを指定する

アプリケーションで使用するコンストラクタ／メソッド／フィールドが分かる場合、アダプタクラス生成時にこれらを指定することにより、生成するアダプタクラス数を少なくすることができます。

Jアダプタクラスジェネレータは、指定されたコンストラクタ／メソッド／フィールドに必要なアダプタクラスだけを生成します。

コンストラクタ／メソッド／フィールドの指定方法の詳細については、[-rオプション](#)、[-gcオプション](#)、[-gmオプション](#)、[-gfオプション](#)および“[Class クラス名／インタフェース名](#)”パラメタを参照してください。

## 例

アプリケーションでjava.io.PrintStreamクラスのprintln(Object)メソッドだけを使用する場合は、以下のように指定します。

```
C:\¥> java2cob -r java.io.PrintStream -gm "println(java.lang.Object)"
```

#### 3.1.5.2 -omオプションまたは“Option ReduceClass”パラメタを指定する

アプリケーションで使用するコンストラクタ／メソッド／フィールドが分からない場合、[-omオプション](#)または“[Option ReduceClass](#)”パラメタを指定することにより、生成するアダプタクラス数を少なくすることができます。

Jアダプタクラスジェネレータでは、メソッド呼出し時のパラメタ妥当性チェックのために、各パラメタに対応するアダプタクラスを生成します。そのため、ひとつのクラスに対し、多くのアダプタクラスが生成されます。[-omオプション](#)または“[Option ReduceClass](#)”パラメタを指定すると、オブジェクト型のメソッドパラメタはすべてjava.lang.Objectクラスにマッピングされます。これにより、メソッドパラメタに対応するアダプタクラスの生成を抑制し、生成アダプタクラス数を減らすことができます。

## 注意事項

- [-omオプション](#)または“[Option ReduceClass](#)”パラメタを指定して生成したアダプタクラスのメソッドを呼び出す場合、オブジェクト参照のパラメタには、BY CONTENT 指定が必要となります。

- [-omオプション](#)または[“Option ReduceClass” パラメタ](#)を指定した場合、メソッドのパラメタのうち復帰値を除くパラメタのオブジェクト参照の型がjava-lang-Objectになるため、元のパラメタの型が分からなくなります。このため、オブジェクトの参照型がjava-lang-Objectになるパラメタについては、パラメタ名の中に元の型情報を含めるように、以下の規則によりパラメタ名を生成します。

<b>P<math>n</math>-クラス名</b>
-----------------------------

- “P”の後に、パラメタの通し番号(1～99)を振る
- ハイフン(-)の後に、パッケージ名を除いた外部クラス名を大文字に変換して付加
- 30文字を超えた場合は、31文字目以降を切り捨てる

### 例

[-omオプション](#)および[“Option ReduceClass” パラメタ](#)を指定しない場合、java.io.PrintStreamクラスのアダプタクラスソース java-io-PrintStream.cob は以下のようになり、java-io-OutputStreamクラスが必要となります。

```
...
REPOSITORY.
    CLASS J-OUTPUTSTREAM AS "java-io-OutputStream"
...
LINKAGE SECTION.
01 PARA-1 OBJECT REFERENCE J-OUTPUTSTREAM.
...
```

[-omオプション](#)または[“Option ReduceClass” パラメタ](#)を指定した場合、以下のようにパラメタは、java-lang-Object型となります。

したがって、java-io-OutputStreamクラスは不要となり、生成しません。

```
...
REPOSITORY.
    CLASS J-OBJECT AS "java-lang-Object"
...
LINKAGE SECTION.
01 P1-OUTPUTSTREAM OBJECT REFERENCE J-OBJECT.
...
```

## 3.2 アダプタクラスを利用するアプリケーションの開発

ここでは、アダプタクラスを使用するプログラムの開発方法について説明します。

### 3.2.1 プログラムの書き方

ここでは、アダプタクラスを使用するプログラムの書き方について説明します。アダプタクラスを使用するプログラムの処理の流れは、以下のようになります。

- シングルスレッドアプリケーションの場合
  1. [Java VMの初期化 \(JVM-INITメソッドの呼出し\)](#)
  2. [オブジェクトの生成](#)
  3. [メソッドの呼出し](#)
  4. [Java VMの終了 \(JVM-TERMINATEメソッドの呼出し\)](#)
- マルチスレッドアプリケーションの場合
  1. [Java VMの初期化およびカレントスレッドのJava VMへの接続 \(JVM-ATTACHメソッドの呼出し\)](#)
  2. [オブジェクトの生成](#)
  3. [メソッドの呼出し](#)
  4. [カレントスレッドのJava VMからの分離 \(JVM-DETACHメソッドの呼出し\)](#)

また、以下の処理を行う場合は、注意が必要です。

- [変数の操作](#)
- [オブジェクト参照の比較](#)
- [サブクラスへの代入](#)
- [文字列の受け渡し](#)
- [文字列の終端制御](#)
- [エラー処理](#)

#### 3.2.1.1 Java VMの初期化

アダプタクラスを使用する場合、最初にJava VM (Virtual Machine) を初期化する必要があります。Java VMの初期化は、[FJ-JAVA-CONTROLクラス](#)の[JVM-INITメソッド](#)または[JVM-ATTACHメソッド](#)で行います。

以下にコーディング例を示します。

```
...
REPOSITORY.
  CLASS FJ-JAVA-CONTROL
...
PROCEDURE DIVISION.
...
  INVOKE FJ-JAVA-CONTROL "JVM-INIT".
...
```

#### 3.2.1.2 Java VMの終了

アダプタクラスを使用しなくなった場合、Java VM (Virtual Machine) を終了させます。Java VMの終了は、[FJ-JAVA-CONTROLクラス](#)の[JVM-TERMINATEメソッド](#)で行います。

以下にコーディング例を示します。



```

...
REPOSITORY.
  CLASS FJ-JAVA-CONTROL
...
PROCEDURE DIVISION.
...
  INVOKE FJ-JAVA-CONTROL "JVM-TERMINATE".
...

```

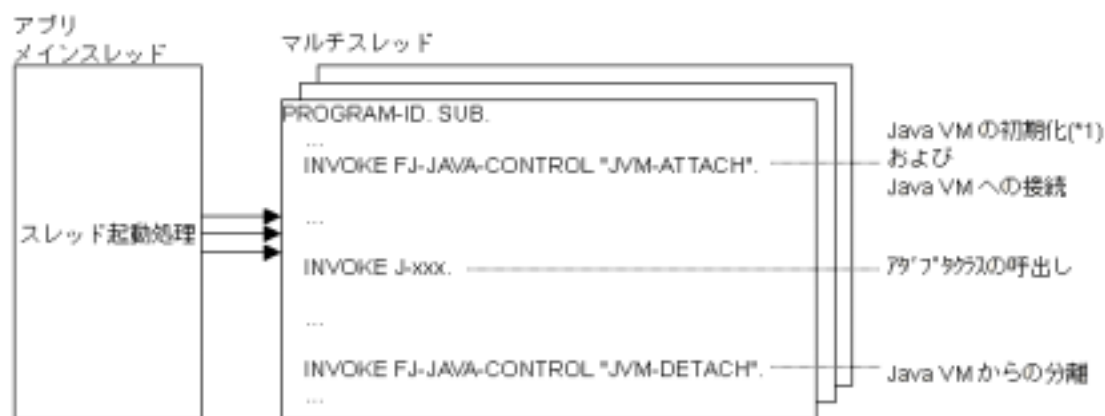
### 3.2.1.3 マルチスレッドアプリケーションの開発

アダプタクラスを使用したマルチスレッドのアプリケーションでは、スレッドごとにカレントスレッドをJava VM (Virtual Machine) に接続する必要があります。また、スレッドを終了する前に、カレントスレッドをJava VMから分離する必要があります。

カレントスレッドのJava VMへの接続は、[FJ-JAVA-CONTROLクラス](#)の[JVM-ATTACHメソッド](#)で行います。Java VMへの接続は、スレッド内でアダプタクラスを使用する前に必ず実行しなければなりません。

また、カレントスレッドのJava VMからの分離は、[FJ-JAVA-CONTROLクラス](#)の[JVM-DETACHメソッド](#)で行います。Java VMからの分離は、スレッド内でアダプタクラスを使わなくなった場合に必ず実行しなければなりません。

以下に、マルチスレッドアプリケーションの例を示します。



\*1: Java VMの初期化はJVM-ATTACHの最初の呼出しだけで行われます。

### 3.2.1.4 オブジェクトの生成

オブジェクトの生成は、アダプタクラスの、コンストラクタに対応するファクトリメソッドを呼び出すことにより行います。ファクトリメソッドは、以下の名前で生成します（“[コンストラクタ](#)”参照）。

Create-*Java*クラス名-*nn* (*nn*は01～99の番号)

以下にDateクラスのオブジェクトを生成する例を示します。

```

...
REPOSITORY.
    CLASS J-Date AS "java-util-Date"
...
WORKING-STORAGE SECTION.
01  anDate OBJECT REFERENCE J-Date.
...
PROCEDURE DIVISION.
...
    INVOKE J-Date "Create-Date-01" RETURNING anDate.
...

```

## 補足

アダプタクラスの名前は長いので、REPOSITORY段落でASを指定して別名をつけると便利です。

### 3.2.1.5 メソッド呼出し

インスタンスメソッドの呼出しは、アダプタクラスの対応するオブジェクトメソッドを呼び出すことにより行います。メソッド名は、Javaのメソッド名と同じです。ただし、同名のメソッドを複数定義している場合、区別のために後ろに番号を付加します（“[クラスメソッド](#)” および “[インスタンスメソッド](#)” 参照）。

以下にDateクラスのgetTimeメソッドを呼び出す例を示します。

```

...
REPOSITORY.
    CLASS J-Date AS "java-util-Date"
...
WORKING-STORAGE SECTION.
01  anDate OBJECT REFERENCE J-Date.
01  currentTime PIC S9(9) COMP-5.
...
PROCEDURE DIVISION.
...
    INVOKE anDate "getTime" RETURNING currentTime.
...

```

### 3.2.1.6 変数の操作

変数の操作は、アダプタクラスの対応するプロパティを通して行います。プロパティ名はJavaのフィールド名に“JF-”をつけた名前です。ただし、同名のフィールドを複数定義している場合は、区別のために後ろに番号を付加します（“[クラス変数](#)” および “[インスタンス変数](#)” 参照）。以下にDateFormatクラスのクラス変数AM\_PM\_FIELD（スタティックフィールド）を参照する例を示します。

```

...
REPOSITORY.
    CLASS J-DateFormat AS "java-text-DateFormat"
...
WORKING-STORAGE SECTION.
01  FMT-Type PIC S9(9) COMP-5.
...
PROCEDURE DIVISION.
...
    MOVE JF-AM_PM_FIELD OF J-DateFormat TO FMT-Type.
...

```

### 3.2.1.7 オブジェクト参照の比較

COBOLでは、複数のオブジェクト参照が同じオブジェクトを指しているか検査する場合、“=”を

使います。しかし、アダプタオブジェクトが指すJavaオブジェクトが同じであるか検査する場合は、“=”の代わりにアダプタクラスの[J-EQUALSメソッド](#)を使用します。アダプタクラスは、必ずJ-EQUALSメソッドを持っています。

以下に、二つのDateオブジェクトを比較する例を示します。Date-1とDate-2が同じJavaオブジェクトを指している場合に条件が成立します。

```
...
REPOSITORY.
    CLASS J-Date AS "java-util-Date"
...
WORKING-STORAGE SECTION.
01 Date-1 OBJECT REFERENCE J-Date.
01 Date-2 OBJECT REFERENCE J-Date.
01 rst PIC 1.
...
PROCEDURE DIVISION.
...
    INVOKE Date-1 "J-EQUALS" USING CONTENT Date-2 RETURNING rst.
    IF rst = B"1" THEN
        条件成立
...

```

### 3.2.1.8 サブクラスへの代入

COBOLでは、オブジェクトをサブクラスに代入する場合、ASを使用します。しかし、アダプタオブジェクトの場合は、ASの代わりに[J-NARROWメソッド](#)を使用します。アダプタクラスは、必ずJ-NARROWメソッドを持っています。

以下に、Objectクラスのデータで参照していたオブジェクトをDateクラスのデータに代入する例を示します。

```
...
REPOSITORY.
    CLASS J-Object AS "java-lang-Object"
    CLASS J-Date AS "java-util-Date"
...
WORKING-STORAGE SECTION.
01 anDate OBJECT REFERENCE J-Date.
01 anObject OBJECT REFERENCE J-Object.
...
PROCEDURE DIVISION.
...
    INVOKE J-Date "J-NARROW" USING CONTENT anObject RETURNING anDate.
...

```

### 3.2.1.9 文字列の受け渡し

アダプタクラスに文字列を受け渡すには、以下の方法があります。

- [Stringオブジェクトで受け渡す](#)
- [COBOLの英数字項目を直接受け渡す](#)

#### 3.2.1.9.1 Stringオブジェクトで受け渡す

通常のアダプタクラスでの文字列受け渡し方法です。

JavaのString型はアダプタクラスのjava-lang-Stringにマッピングされるため、ユーザアプリケーションでは[java-lang-Stringクラス](#)のメソッド（NEW-STRING-X、GET-STRING-Xなど）を使用してStringオブジェクトとCOBOLデータ項目間の変換を行う必要があります。このため、ユーザアプリケーションの作成が繁雑になります。

以下に、Stringオブジェクトで受け渡す場合の例を示します。NEW-STRING-Xメソッドを使用して

文字列からStringオブジェクトを生成した後、parseメソッドに渡しています。また、GET-STRING-Xメソッドを使用して、toStringメソッドで受け取ったStringオブジェクトから文字列を取り出しています。

```
...
REPOSITORY.
  CLASS J-Date AS "java-util-Date"
  CLASS J-String AS "java-lang-String"
  CLASS J-DateFormat AS "java-text-DateFormat"
...
WORKING-STORAGE SECTION.
01 aDateFormat OBJECT REFERENCE J-DateFormat.
01 aDate OBJECT REFERENCE J-Date.
01 dateString OBJECT REFERENCE J-String.
01 dateValue PIC X(30).
...
PROCEDURE DIVISION.
...
  MOVE "2000/01/01" & X"00" TO dateValue.
  INVOKE J-String "NEW-STRING-X" USING dateValue RETURNING dateString.
  INVOKE aDateFormat "parse" USING dateString RETURNING aDate.
...
  INVOKE aDate "toString" RETURNING dateString.
  INVOKE dateString "GET-STRING-X" RETURNING dateValue.
...
```

### 3.2.1.9.2 COBOLの英数字項目を直接受け渡す

[-sオプション](#)または[“Option String”パラメタ](#)を指定してアダプタクラスを生成することにより可能となる文字列受け渡し方法です。

JavaのString型はPIC X ANY LENGTHにマッピングされるため、ユーザアプリケーションでは英数字項目を直接受け渡すことができるようになります。このため、ユーザアプリケーションの作成が簡単になります。

英数字項目として直接受け渡すことができるのは、以下の項目です。

- メソッドでのjava.lang.String型の復帰値
- コンストラクタおよびメソッドでのjava.lang.String型の引数
- java.lang.String型のフィールド ([クラス変数](#)および[インスタンス変数](#))

以下に、英数字項目を直接受け渡す場合の例を示します。String型の引数および復帰値に英数字項目を指定できるため、StringオブジェクトとCOBOLデータ項目間の変換を行う必要がありません。

```
...
REPOSITORY.
  CLASS J-Date AS "java-util-Date"
  CLASS J-DateFormat AS "java-text-DateFormat"
...
WORKING-STORAGE SECTION.
01 aDateFormat OBJECT REFERENCE J-DateFormat.
01 aDate OBJECT REFERENCE J-Date.
01 dateValue PIC X(30).
...
PROCEDURE DIVISION.
...
  MOVE "2000/01/01" & X"00" TO dateValue.
  INVOKE aDateFormat "parse" USING dateValue RETURNING aDate.
...
  INVOKE aDate "toString" RETURNING dateValue.
...
```

**注意事項**

[-sオプション](#)または[“Option String” パラメタ](#)を指定してアダプタクラスを生成する場合の注意事項を以下に示します。

- java.lang.Stringクラスのコンストラクタの復帰値はjava.lang.String型です。
- Stringオブジェクトのメソッドを使用する場合は、オブジェクトの生成（Stringコンストラクタの呼び出しまたはNEW-STRING-Xメソッドの呼び出し）が必要です。
- java.lang.String型のフィールド（[クラス変数](#)および[インスタンス変数](#)）を参照／設定する場合は、[-sオプション](#)または[“Option String” パラメタ](#)でサイズを指定します。
- String型NULLオブジェクトを扱いたい場合は、[-sオプション](#)および[“Option String” パラメタ](#)を使用しないでください。

**3.2.1.10 文字列の終端制御**

String型をパラメタとするメソッドおよびString型のフィールドに対し、データ項目長より短い英数字項目および日本語項目の文字列を渡す場合、通常のアプリケーション開発では、Java側のデータと整合性を取るために、文字列の終端（X” 00”）を意識して設定する必要があります。以下に、長さ50文字の英数字項目initialValueに” ABC” を転記してNEW-STRING-Xメソッドに渡す例を示します。

```
...
REPOSITORY.
  CLASS J-String AS " java-lang-String"
  ...
WORKING-STORAGE SECTION.
01  initialValue  PIC X(50).
01  aString       OBJECT REFERENCE J-String.
  ...
PROCEDURE DIVISION.
  ...
  MOVE " ABC" & X" 00" TO initialValue.
  INVOKE J-String " NEW-STRING-X" USING initialValue RETURNING aString.
```

これに対し、[-tオプション](#)または[“Option Terminal” パラメタ](#)を指定してアダプタクラスを生成することにより、アダプタクラスで文字列の終端を設定するため、ユーザアプリケーションでは文字列の終端を意識しないでデータ項目長より短い文字列をメソッドに渡せるようになります。

以下に、長さ50文字の英数字項目initialValueに” ABC” を転記してNEW-STRING-Xメソッドに渡す例を示します。

```
...
REPOSITORY.
  CLASS J-String AS " java-lang-String"
  ...
WORKING-STORAGE SECTION.
01  initialValue  PIC X(50).
01  aString       OBJECT REFERENCE J-String.
  ...
PROCEDURE DIVISION.
  ...
  MOVE " ABC" TO initialValue.
  INVOKE J-String " NEW-STRING-X" USING initialValue RETURNING aString.
```

**注意事項**

- [-tオプション](#)または[“Option Terminal” パラメタ](#)を指定してアダプタクラスを生成した場合、文字列の終端の空白文字列は削除されてメソッドに渡されます。
- [-tオプション](#)または[“Option Terminal” パラメタ](#)を指定してアダプタクラスを生成した場合、空白だけの文字列をメソッドに渡せません。

### 3.2.1.11 エラー処理

アダプタクラスは、実行時にエラーを検出した場合、以下の例外オブジェクトを発生させます。

- Javaの例外クラスに対応する例外オブジェクト（Javaクラスで例外が発生した場合）
- [FJ-JAVA-ERRORクラス](#)の例外オブジェクト（Jアダプタクラスジェネレータのランタイムでエラーが発生した場合）

アプリケーションがアダプタクラスで発生した例外オブジェクトを検出するには、手続き部の宣言節部分にUSE文を使った例外処理を記述する必要があります。USE文を使った例外処理の詳細については“NetCOBOL 使用手引書”の“例外オブジェクト”を参照してください。

以下に例外処理のコーディング例を示します。

```
...
REPOSITORY.
    CLASS FJ-JAVA-ERROR
    CLASS J-String AS "java-lang-String"
    CLASS J-Exception AS "java-lang-Exception"
    ...
WORKING-STORAGE SECTION.
01 msgString      OBJECT REFERENCE J-String.
01 errMessage     PIC X(256).
01 expMessage     PIC X(1024).
01 expClass       PIC X(256).
01 errCode        PIC S9(9) COMP-5.
01 errMessageLen  PIC S9(9) COMP-5.
01 expMessageLen  PIC S9(9) COMP-5.
01 expClassLen    PIC S9(9) COMP-5.
01 rc             PIC S9(9) COMP-5.
    ...
PROCEDURE DIVISION.
DECLARATIVES.
JAVA-ERR SECTION.
    USE AFTER EXCEPTION J-Exception.
    INVOKE EXCEPTION-OBJECT "GET-MESSAGE" RETURNING msgString.
    ...
JADP-ERR SECTION.
    USE AFTER EXCEPTION FJ-JAVA-ERROR.
    INVOKE EXCEPTION-OBJECT "GET-CODE" RETURNING errCode.
    INVOKE EXCEPTION-OBJECT "GET-MESSAGE"
        USING errMessage RETURNING errMessageLen.
    INVOKE EXCEPTION-OBJECT "GET-EXCEPTION"
        USING expMessage expMessageLen expClass expClassLen
        RETURNING rc.
    DISPLAY "Error Code: " errCode.
    DISPLAY "Error Message: " errMessage(1:errMessageLen).
    IF rc NOT = -1 THEN
        DISPLAY "Java Class Name: " expClass(1:expClassLen)
        DISPLAY "Java Exception Message: " expMessage(1:expMessageLen)
    END-IF.
    ...
END DECLARATIVES.
...
```

#### 注意事項

プログラムで例外処理を記述しなかった場合は、例外オブジェクトの発生によりプログラムで実行時エラー（JMP0104I-U）が発生します。

### 3.2.2 プログラムの構築

ここでは、アダプタクラスを使用するプログラムを、COBOLプロジェクトマネージャを用いて構築する方法について説明します。

プログラムの構築には、アダプタクラスから作成した以下のファイルが必要になります。

- アダプタクラスのLIBファイル（リンク時）
- アダプタクラスのリポジトリファイル（翻訳時）

プログラムの構築は、以下の手順で行います。

1. プロジェクトマネージャで新規プロジェクトを作成します。
2. ターゲットとなる実行形式プログラム（EXE）を登録します。
3. COBOLソースファイルフォルダを作成し、プログラムのソースを登録します。
4. ライブラリファイルフォルダを作成し、アダプタクラスのLIBファイルとJアダプタクラスジェネレータの実行時ライブラリF3BIJART.LIBを登録します。F3BIJART.LIBは、インストールフォルダのLIBフォルダにあります。
5. 翻訳オプション・リンクオプションを指定します。
  - REPINには、XXX¥REP（XXXはJアダプタクラスジェネレータのインストールフォルダ）と、アダプタクラスのリポジトリを格納したフォルダを指定します。
  - ALPHAL（WORD）またはNOALPHALを指定します。
  - Unicodeで動作するプログラムを作成する場合、RCS（UCS2）を指定します。
  - マルチスレッドアプリケーションを作成する場合、THREAD（MULTI）を指定します。
  - その他のオプションについては、“NetCOBOL 使用手引書”を参照してください。
6. ビルドします。

## 3.3 プログラムの実行

プログラムの実行には、アダプタクラスから作成した以下のファイルが必要になります。

- アダプタクラスのDLLファイル

プログラムを実行する前に、環境変数PATHにアダプタクラスのDLLファイルを格納したフォルダを追加します。

プログラムの実行方法は、一般のCOBOLアプリケーションと同じです。詳細は“NetCOBOL 使用手引書”を参照してください。

### 補足

[環境変数](#)を指定することにより、Java VMの実行環境をカスタマイズできます。



## 3.4 Java2への移行

JDK 1.1.xで作成した資産（実行形式プログラム（EXE）、DLL、アダプタクラスのソース、およびアダプタクラスを使用するプログラムのソース）を、Java2（J2SDK 1.2.2以降またはJ2RE 1.2.2以降）で使用する場合は、以下の点に注意してください。

- JDK 1.1.xで作成したアダプタクラスのソースおよびDLLは、Java2の環境でそのまま利用できます。
- Java2でアダプタクラスを作成しなおすと、JDK1.1.xで作成した場合と異なるメソッド名が生成される場合があります（“[名前の番号付け](#)”参照）。このような場合、Javaクラスを使用していたCOBOLプログラムを修正する必要があります。これを防ぐために、Java2で作業する際に、JDK1.1.xで作業した際に生成された[生成名管理ファイル](#)を使用してアダプタクラスを生成してください。



---

## 第4章 ジェネレータの使い方

---

この章では、ジェネレータコマンド（java2cob）の起動方法、オプションファイルおよび出力結果について説明します。

## 4.1 起動方法

### コマンドの形式

#### コンストラクタ／メソッド／フィールドを指定しない場合

```
java2cob [-classpath クラスパス] [-d 出力先フォルダ] [-ov] [-om] [-oi] [-c{s|u}] [-s{n}] [-t] クラス名／インタフェース名 ...
```

#### コンストラクタ／メソッド／フィールドを指定する場合

```
java2cob [-classpath クラスパス] [-d 出力先フォルダ] [-ov] [-oi] [-c{s|u}] [-s{n}] [-t] -r クラス名／インタフェース名 [-gc [" コンストラクタ名[(パラメタ型)][, コンストラクタ名...]"] ] [-gm [" メソッド名[(パラメタ型)][, メソッド名...]"] ] [-gf [フィールド名[, フィールド名...]]] [-r...]
```

#### オプションファイルを指定する場合

```
java2cob -i オプションファイル
```

### コマンド記述上の留意事項

[]で囲まれた字句は省略可能です。

{ }は|で区切られた字句の選択を示します。

...は繰り返し指定できることを示します。

イタリック体は可変文字列を示します。

### オプションの意味

#### -classpath クラスパス

Javaクラス／インタフェースの検索パスを指定します。複数のフォルダを指定する場合は、フォルダの間をセミコロン (;) で区切ります。

このオプションを指定した場合、環境変数CLASSPATHを無視します。

このオプションはjava2cobコマンドの直後に指定します。

#### -c{s|u}

実行時のコード系を指定します。Javaクラスを使用するCOBOLプログラムと同じコード系を指定してください。省略した場合は、シフトJISが指定されたものとみなします。

-cs: 実行時のコード系がシフトJISの場合に指定します。

-cu: 実行時のコード系がUnicodeの場合に指定します。

#### -d 出力先フォルダ

アダプタクラスのソースを出力するフォルダを指定します。省略した場合はカレントフォルダに生成します。

#### -gc [" コンストラクタ名[(パラメタ型)][,...]" ]

このオプションより前に指定した一番近いクラス名／インタフェース名に対し、アダプタクラスとして生成するコンストラクタ名を指定します。コンストラクタ名を省略した場合、対応するクラス／インタフェース内のすべてのコンストラクタを生成します。複数のコンストラクタを指定する場合、カンマ (,) または空白で区切って指定します。

同名のコンストラクタが複数存在する場合、パラメタ型を指定することで、パラメタ型が一致するコンストラクタだけを生成します。パラメタ型を省略した場合、同名コンストラクタをすべて生成します。パラメタ型を指定する場合、パラメタ型を括弧 ((および)) で囲み、オプション全体を二重引用符 (") で囲みます。複数のパラメタがある場合は、カンマ (,) で区切って指定します。パラメタ型は、パッケージ名で修飾したクラス名またはJavaのデータ型名で指定します。パラメタ型に内部クラス (インナークラスともいう) 名を指定する場合は、内部クラス名の直前のピリオド (.) をドル (\$) に置き換えて指定します (例: java.lang.Character\$Subset)。

このオプションは-rオプションが指定された場合に有効です。

このオプションを指定した場合は、[-omオプション](#)を指定できません。

#### **-gf [フィールド名[,...]]**

このオプションより前に指定した一番近いクラス名／インタフェース名に対し、アダプタクラスとして生成するフィールド名を指定します。フィールド名を省略した場合、対応するクラス／インタフェース内のすべてのフィールドを生成します。複数のフィールドを指定する場合、カンマ (,) または空白で区切って指定します。

このオプションは[-rオプション](#)が指定された場合に有効です。

このオプションを指定した場合は、[-omオプション](#)を指定できません。

#### **-gm [”メソッド名[(パラメタ型)][,...]”]**

このオプションより前に指定した一番近いクラス名／インタフェース名に対し、アダプタクラスとして生成するメソッド名を指定します。メソッド名を省略した場合、対応するクラス／インタフェース内のすべてのメソッドを生成します。複数のメソッドを指定する場合、カンマ (,) または空白で区切って指定します。

同名のメソッドが複数存在する場合、パラメタ型を指定することで、パラメタ型が一致するメソッドだけを生成します。パラメタ型を省略した場合、同名メソッドをすべて生成します。パラメタ型を指定する場合、パラメタ型を括弧 ((および)) で囲み、オプション全体を二重引用符 (”) で囲みます。複数のパラメタがある場合は、カンマ (,) で区切って指定します。パラメタ型は、パッケージ名で修飾したクラス名または[Javaのデータ型名](#)で指定します。パラメタ型に内部クラス名を指定する場合は、内部クラス名の直前のピリオド (.) をドル (\$) に置き換えて指定します (例: java.lang.Character\$Subset)。

このオプションは[-rオプション](#)が指定された場合に有効です。

このオプションを指定した場合は、[-omオプション](#)を指定できません。

#### **-i オプションファイル**

[オプションファイル](#)を指定します。

このオプションを指定した場合、コマンドライン中の他のオプションを無視します。

#### **-oi**

[メソッド名対応表ファイル](#) (Javaクラスのメソッドに対応するアダプタクラスのメソッドの一覧) を出力する場合に指定します。メソッド名対応表ファイルは、Javaクラスごとに “アダプタクラスのソースファイル名.txt” で出力されます。

#### **-om**

アダプタクラス数を減らす場合に指定します。指定した場合、RETURNINGを除くパラメタのオブジェクト参照の型がjava-lang-Objectになります。そして、その代わりに元の型情報を含むようにパラメタ名を生成します (“[-omオプションまたは “Option ReduceClass” パラメタを指定する](#)” 参照)。

このため、ほとんどの場合クラスブラウザでメソッドを区別できます。パラメタ名が長いなどの理由によりクラスブラウザでメソッドを区別できない場合は、[メソッド名対応表ファイル](#)を使用して区別できます。

このオプションを指定した場合は、[-rオプション](#)、[-gcオプション](#)、[-gmオプション](#)、[-gfオプション](#)および[-sオプション](#)を指定できません。

#### **-ov**

同名のアダプタクラスが既に存在するときに、上書きする場合に指定します。省略した場合は上書きしません。

#### **-r**

このオプションの直後に指定したクラス名／インタフェース名に対し、アダプタクラスとして生成するコンストラクタ／メソッド／フィールドを指定する場合に指定します。これによりアダプタクラスのサイズを小さくすることができます (“[コンストラクタ／メソッド／フィールドを指定する](#)” 参照)。

コンストラクタの指定方法については[-gcオプション](#)を、メソッドの指定方法については[-gmオプション](#)を、フィールドの指定方法については[-gfオプション](#)を、それぞれ参照してください。

このオプションの直後には[クラス名／インタフェース名](#)を指定します。

複数のクラス名／インタフェース名に対してコンストラクタ／メソッド／フィールドを指定する場合は、クラス名／インタフェース名ごとに-rオプションを指定します。

このオプションに対応する[-gcオプション](#)、[-gmオプション](#)および[-gfオプション](#)が一つもない場合、コンストラクタ／メソッド／フィールドなしのアダプタクラスを生成します。このオプションを指定した場合は、[-omオプション](#)を指定できません。

このオプションを使用する場合、一回のjava2cobコマンドですべてのアダプタクラスを作成してください。複数回のjava2cobコマンドに分けて生成した場合、正しいアダプタクラスが生成されない場合があります。

一回のjava2cobコマンドで、コンストラクタ／メソッド／フィールドを指定する場合と指定しない場合とを同時に指定できません。

#### -s[n]

java.lang.String型を英数字項目 (PIC X) にマッピングしたアダプタクラスを生成する場合に指定します (“[文字列の受け渡し](#)” 参照)。

nは、String型フィールドに対応するプロパティメソッドのパラメタサイズ (PIC X(n)) を指定します。省略した場合は256が指定されたものとみなします。

このオプションを指定した場合は、[-omオプション](#)を指定できません。

#### -t

String型をパラメタとするメソッドおよびString型のフィールドに対し、終端文字列の設定を行うようにしたアダプタクラスを生成する場合に指定します (“[文字列の終端制御](#)” 参照)。

#### クラス名／インタフェース名

アダプタクラスを生成するJavaのクラス名またはインタフェース名を、パッケージ名で修飾して指定します。内部クラス名を指定する場合は、内部クラス名の直前のピリオド(.)をドル(\$)に置き換えて指定します (例: java.lang.Character\$Subset)。クラス名／インタフェース名は複数指定できます。

### 環境変数

#### CLASSPATH

Javaクラス／インタフェースの検索パスを指定します。-classpathを指定した場合、この環境変数を無視します。

### 注意事項

- コンストラクタ／メソッド／フィールドを指定する場合、[-rオプション](#)、[クラス名／インタフェース名](#)、[-gcオプション](#)、[-gmオプション](#)および[-gfオプション](#)を続けて指定してください。これらのオプションの間にその他のオプションを指定すると指定エラーになります。
- クラス名およびインタフェース名には、日本語は使用できません。
- 実行時のコード系にUnicodeを指定した場合、翻訳オプションに“RCS(UCS2)”を指定してください (“[アダプタクラスの構築](#)” 参照)。

### 使用例

java2cobコマンドの使用例を通して、その使い方を説明します (“¥” は継続行を示します)。

- java.io.PrintStreamクラスおよびjava.util.Dateクラスに関連するすべてのアダプタクラスを生成します。

```
c:¥> java2cob java.io.PrintStream java.util.Date
```

- java.io.PrintStreamクラスのすべてのprintlnメソッドおよびjava.util.Dateクラスのすべてのコンストラクタに関連するアダプタクラスを生成します。

```
c:¥> java2cob -r java.io.PrintStream -gm println -r java.util.Date -gc
```

- java.io.PrintStreamクラスのprintln(Object)メソッドおよびjava.util.DateクラスのDate()コンストラクタに関連するアダプタクラスだけを生成します。

```
c:¥> java2cob -r java.io.PrintStream -gm "println(java.lang.Object)" ¥
-r java.util.Date -gc "Date()"
```

## 4.2 オプションファイル

オプションファイルは、java2cobコマンドのコマンドラインで指定できるオプションをファイルに定義するためのテキスト形式ファイルです。

たとえばコンストラクタ/メソッド/フィールド指定で多くのメソッド名などを毎回コマンドラインに指定するのが面倒な場合に、オプションファイルを作成することで、簡単に実行できるようになります（コマンド形式の“[オプションファイルを指定する場合](#)”参照）。

### 形式

オプションファイルに指定可能なパラメタを以下に示します。

パラメタ名	概要	備考
<a href="#">Class クラス名/インタフェース名</a>	アダプタクラスを生成するJavaのクラス名またはインタフェース名を指定します。 アダプタクラスとして生成するコンストラクタ/メソッド/フィールドを指定します。	省略不可。 複数指定可能。
<a href="#">Option ClassPath</a>	Javaクラス/インタフェースの検索パスを指定します。	省略可能。 複数指定不可。
<a href="#">Option Code</a>	実行時のコード系を指定します。	省略可能。 複数指定不可。
<a href="#">Option CommandOptions</a>	Java2cobコマンドのオプションを直接指定します。	省略可能。 複数指定不可。
<a href="#">Option GenOnlyUsed</a>	アダプタクラスとして生成するコンストラクタ/メソッド/フィールドを指定するかどうかを指定します。	省略可能。 複数指定不可。
<a href="#">Option MethodTable</a>	<a href="#">メソッド名対応表ファイル</a> を出力するかどうかを指定します。	省略可能。 複数指定不可。
<a href="#">Option OutPutPath</a>	アダプタクラスのソースを出力するフォルダを指定します。	省略可能。 複数指定不可。
<a href="#">Option OverWrite</a>	同名のアダプタクラスが既に存在するときに、上書きするかどうかを指定します。	省略可能。 複数指定不可。
<a href="#">Option ReduceClass</a>	アダプタクラス数を減らすかどうかを指定します。	省略可能。 複数指定不可。
<a href="#">Option String</a>	java.lang.String型を英数字項目にマッピングしたアダプタクラスを生成するかどうかを指定します。	省略可能。 複数指定不可。
<a href="#">Option Terminal</a>	String型をパラメタとするメソッドおよびString型のフィールドに対し、終端文字列の設定を行うようにしたアダプタクラスを生成するかどうかを指定します。	省略可能。 複数指定不可。

### オプションファイル記述上の留意事項

[]で囲まれた字句は省略可能です。

{ }は|で区切られた字句の選択を示します。

イタリック体は可変文字列を示します。

### Class クラス名/インタフェース名

#### 指定形式

```
Class クラス名/インタフェース名 [= コンストラクタ/メソッド/フィールド指定オプション]
```

**指定内容**

アダプタクラスを生成するJavaのクラス名またはインタフェース名を指定します。

また、そのクラス名／インタフェース名に対し、アダプタクラスとして生成するコンストラクタ／メソッド／フィールドを指定する場合は、コンストラクタ／メソッド／フィールド指定オプションを指定します。

**パラメタの意味****クラス名／インタフェース名**

クラス名またはインタフェース名を、パッケージ名で修飾して指定します。内部クラス名を指定する場合は、内部クラス名の直前のピリオド（.）をドル（\$）に置き換えて指定します（例：java.lang.Character\$Subset）。

**コンストラクタ／メソッド／フィールド指定オプション**

コンストラクタ／メソッド／フィールドを、java2cobコマンドの[-gcオプション](#)、[-gmオプション](#)および[-gfオプション](#)の形式で指定します。

コンストラクタ／メソッド／フィールド指定オプションは、[“Option GenOnlyUsed” パラメタ](#)でYESを指定した場合に有効です。

[“Option GenOnlyUsed” パラメタ](#)でYESを指定し、かつコンストラクタ／メソッド／フィールド指定オプションを指定しなかった場合、コンストラクタ／メソッド／フィールドなしのアダプタクラスを生成します。

**Option ClassPath****指定形式**

```
Option ClassPath = クラスパス
```

**指定内容**

Javaクラス／インタフェースの検索パスを指定します。複数のフォルダを指定する場合は、フォルダの間をセミコロン（;）で区切ります。

このオプションを指定した場合、環境変数CLASSPATHを無視します。

**Option Code****指定形式**

```
Option Code = { SJIS | UNICODE }
```

**指定内容**

実行時のコード系を指定します。Javaクラスを使用するCOBOLプログラムと同じコード系を指定してください。

省略した場合は、SJISが指定されたものとみなします。

**パラメタの意味****SJIS**

実行時のコード系がシフトJISの場合に指定します。

**UNICODE**

実行時のコード系がUnicodeの場合に指定します。

**Option CommandOptions****指定形式**

```
Option CommandOptions = コマンドオプション
```

**指定内容**

java2cobコマンドの[オプション](#)を直接指定します。ただし、[-iオプション](#)は指定できません。コマンドオプションと同種のパラメタをオプションファイルに指定した場合、コマンドオプションの指定値を有効にします。



## Option GenOnlyUsed

### 指定形式

Option GenOnlyUsed = { YES   NO }
-----------------------------------

### 指定内容

アダプタクラスとして生成するコンストラクタ/メソッド/フィールドを指定するかどうかを指定します。コンストラクタ/メソッド/フィールドを指定することにより、アダプタクラスのサイズを小さくすることができます（“[コンストラクタ/メソッド/フィールドを指定する](#)” 参照）。

省略した場合は、NOが指定されたものとみなします。

“Option GenOnlyUsed” パラメタの指定値は、オプションファイル内のすべての“[Class クラス名/インタフェース名](#)”に影響します。

“Option GenOnlyUsed” パラメタを指定した場合は、“[Option ReduceClass](#)” パラメタを指定できません。

### パラメタの意味

YES

コンストラクタ/メソッド/フィールドを指定します。

NO

コンストラクタ/メソッド/フィールドを指定しません。

## Option MethodTable

### 指定形式

Option MethodTable = { YES   NO }
-----------------------------------

### 指定内容

[メソッド名対応表ファイル](#)（Javaクラスのメソッドに対応するアダプタクラスのメソッドの一覧）を出力するかどうかを指定します。

省略した場合は、NOが指定されたものとみなします。

メソッド名対応表ファイルは、Javaクラスごとにアダプタクラスのソースファイル名.txtで出力されます。

### パラメタの意味

YES

メソッド名対応表ファイルを出力します。

NO

メソッド名対応表ファイルを出力しません。

## Option OutPutPath

### 指定形式

Option OutPutPath = <i>出力先フォルダ</i>
------------------------------------

### 指定内容

アダプタクラスのソースを出力するフォルダを指定します。

省略した場合はカレントフォルダに生成します。

## Option OverWrite

### 指定形式

Option OverWrite = { YES   NO }
---------------------------------

### 指定内容

同名のアダプタクラスが既に存在するときに、上書きするかどうかを指定します。

省略した場合は、NOが指定されたものとみなします。

**パラメタの意味**

YES

上書きします。

NO

上書きしません。

**Option ReduceClass****指定形式**

Option ReduceClass = { YES   NO }
-----------------------------------

**指定内容**

アダプタクラス数を減らすかどうかを指定します。YESを指定した場合、RETURNINGを除くパラメタのオブジェクト参照の型がjava-lang-Objectになります。そして、その代わりに元の型情報を含むようにパラメタ名を生成します（“[-omオプションまたは“Option ReduceClass”パラメタを指定する](#)”参照）。

このため、ほとんどの場合クラスブラウザでメソッドを区別できます。パラメタ名が長いなどの理由によりクラスブラウザでメソッドを区別できない場合は、[メソッド名対応表ファイル](#)を使用して区別できます。

省略した場合は、NOが指定されたものとみなします。

“Option ReduceClass”パラメタを指定した場合は、[“Option GenOnlyUsed”パラメタ](#)および[“Option String”パラメタ](#)を指定できません。

**パラメタの意味**

YES

アダプタクラス数を減らします。

NO

アダプタクラス数を減らしません。

**Option String****指定形式**

Option String = { YES [ <i>n</i> ]   NO }
---

**指定内容**

java.lang.String型を英数字項目（PIC X）にマッピングしたアダプタクラスを生成するかどうかを指定します（“[文字列の受け渡し](#)”参照）。

省略した場合は、NOが指定されたものとみなします。

“Option String”パラメタを指定した場合は、[“Option ReduceClass”パラメタ](#)を指定できません。

**パラメタの意味**YES [*n*]

java.lang.String型を英数字項目にマッピングします。

*n*は、String型フィールドに対応するプロパティメソッドのパラメタサイズ（PIC X(*n*））を指定します。省略した場合は256が指定されたものとみなします。

NO

java.lang.String型を英数字項目にマッピングしません。

**Option Terminal****指定形式**

Option Terminal = { YES   NO }
--------------------------------

**指定内容**

String型をパラメタとするメソッドおよびString型のフィールドに対し、終端文字列の設定を行

うようにしたアダプタクラスを生成するかどうか指定します（“[文字列の終端制御](#)”参照）。省略した場合は、NOが指定されたものとみなします。

### パラメタの意味

#### YES

終端文字列の設定を行うようにしたアダプタクラスを生成します。

#### NO

終端文字列の設定を行うようにしたアダプタクラスを生成しません。

### 例

java.io.PrintStreamクラスのprintln(Object) メソッドおよびjava.util.DateクラスのDate() コンストラクタに関連するアダプタクラスだけを生成する場合、オプションファイルに以下のよう記述します。

```
Option GenOnlyUsed = YES
Class java.io.PrintStream = -gm " println(java.lang.Object)"
Class java.util.Date = -gc " Date()"
```

### 注意事項

- 同種のパラメタを複数個指定した場合、最後に指定したパラメタが有効になります。ただし、“[Class クラス名/インタフェース名](#)”パラメタはすべてが有効になります。
- 1カラム目に“#”を記述した行は、コメント行として無視します。
- 行の終端に“¥”を記述した行は、次行に継続します。

## 4.3 出力

ジェネレータは以下のファイルを出力します。

- [アダプタクラスのソースファイル](#)
- [生成名管理ファイル](#)
- [メソッド名対応表ファイル](#)

### 4.3.1 アダプタクラスのソースファイル

指定したJavaクラス／インタフェースに対応するアダプタクラスのソースファイルを生成します。また、指定したクラス／インタフェースが他のクラス／インタフェースを参照している場合、そのクラス／インタフェースのアダプタクラスも同時に生成します。

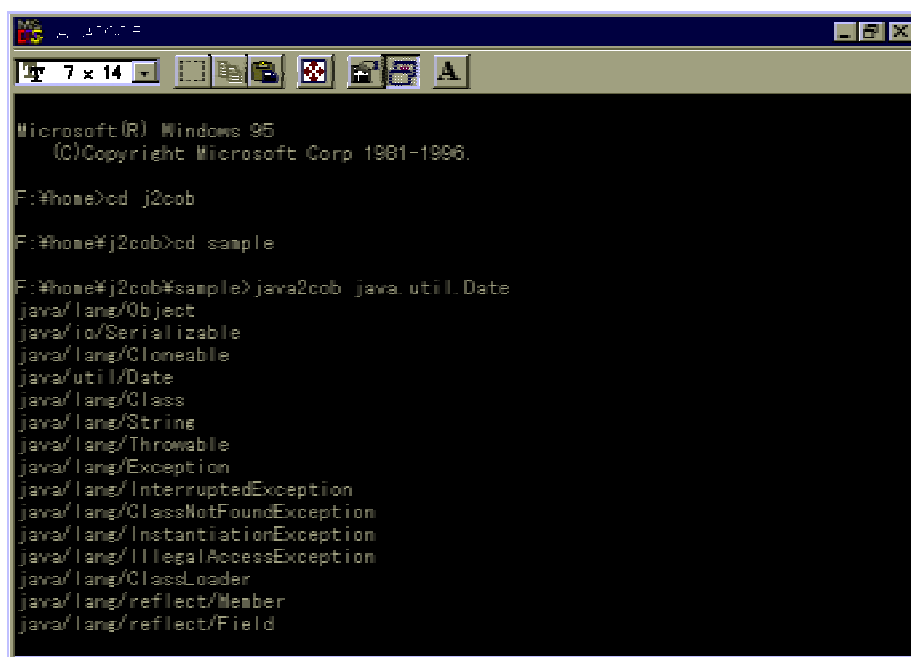
これらの処理を、以下の条件が成立するまで、再帰的に繰り返します。

- 他のクラス／インタフェースを参照していない場合
- 参照しているクラス／インタフェースに対するアダプタクラスを生成済みの場合
- 出力先に同じ名前のソースファイルが既にあり、上書き指定をしなかった場合

生成するソースファイルの名前は、パッケージ名により修飾したクラス名／インタフェース名から、以下の規則により作成します。

- ピリオド (.) およびドル (\$) をハイフン (-) に変換
- 拡張子は “.cob” 固定

java2cobコマンドを実行すると、アダプタクラス生成中のクラス名が表示されます。



```
Microsoft(R) Windows 95
(C)Copyright Microsoft Corp 1981-1996.

F:\home>cd j2cob
F:\home\j2cob>cd sample
F:\home\j2cob\sample>java2cob java.util.Date
java/lang/Object
java/io/Serializable
java/lang/Cloneable
java/util/Date
java/lang/Class
java/lang/String
java/lang/Throwable
java/lang/Exception
java/lang/InterruptedException
java/lang/ClassNotFoundException
java/lang/InstantiationException
java/lang/IllegalAccessException
java/lang/ClassLoader
java/lang/reflect/Member
java/lang/reflect/Field
```

### 4.3.2 生成名管理ファイル

Jアダプタクラスジェネレータは、Javaクラスとアダプタクラスで使う名前の対応関係を管理するために、生成名管理ファイルを使います。生成名管理ファイルは、アダプタクラスの[出力先フォルダ](#)に以下の名前で生成します。

```
java2cob.mgt
```

生成名管理ファイルの使い方については、“[名前の番号付け](#)”を参照してください。

### 4.3.3 メソッド名対応表ファイル

Javaクラスのメソッドに対応するアダプタクラスのメソッドの対応関係を参照可能にするため、メソッド名対応表ファイルを出力します。メソッド名対応表ファイルは、[-oiオプション](#)または[“Option MethodTable”パラメタ](#)が指定された場合に、アダプタクラスの[出力先フォルダ](#)に以下の規則により作成します。

- パッケージ名により修飾したクラス名／インタフェース名のピリオド (.) およびドル (\$) をハイフン (-) に変換
- 拡張子は “.txt” 固定

メソッド名対応表ファイルは以下の形式で出力されます。

```
[Java]   Javaクラス名
[COBOL]  COBOL外部クラス名
(1) [Java] 型 Javaメソッド名 (引数の型)
      [COBOL] COBOL外部メソッド名
(2) [Java] 型 Javaメソッド名 (引数の型)  OVERRIDE
      [COBOL] -None-
...
(n) [Java] 型 Javaメソッド名 ( )
      [COBOL] COBOL外部メソッド名
```

Javaクラス名	パッケージ名で修飾されたクラス名またはインタフェース名
COBOL外部クラス名	Javaクラスに対応するアダプタクラスの外部クラス名
Javaメソッド名	Javaクラスのメソッド
型	Javaメソッドの復帰型
引数の型	Javaメソッドの引数の型
OVERRIDE	スーパークラスのメソッドをオーバーライドする場合に付加
COBOLメソッド名	Javaメソッドに対応するアダプタクラスの外部メソッド名
-None-	Javaメソッドに対応するアダプタクラスのメソッドが存在しない場合

なお、以下のアダプタクラスのメソッド名対応表は出力されません。

- 配列のアダプタクラス

また、java-lang-Stringクラスの以下のメソッドは、メソッド名対応表ファイルに出力されません。

- NEW-STRING-X
- NEW-STRING-N
- GET-STRING-X
- GET-STRING-N
- GET-STRING-LENGTH-X
- GET-STRING-LENGTH-N



---

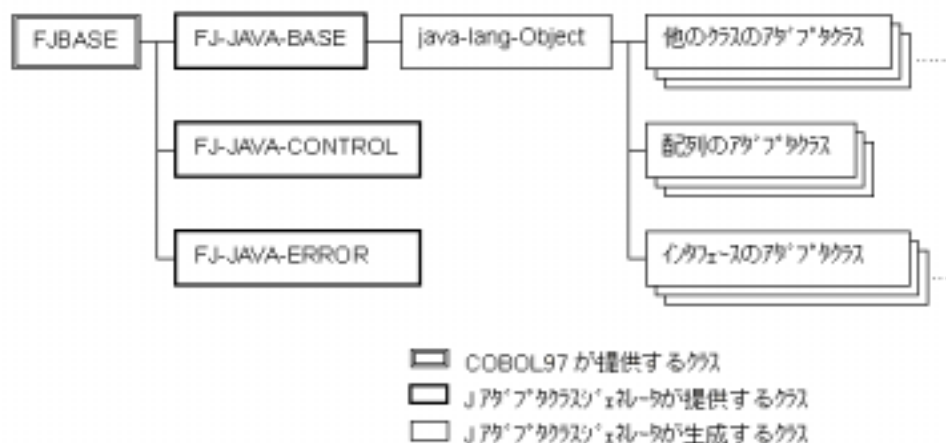
## 第5章      アダプタクラスリファレンス

---

この章では、Jアダプタクラスジェネレータで提供する、[FJ-JAVA-BASEクラス](#)、[FJ-JAVA-CONTROLクラス](#)、[FJ-JAVA-ERRORクラス](#)および生成する[アダプタクラス](#)について、詳細を説明します。

## 5.1 クラス構成

Jアダプタクラスジェネレータで提供するFJ-JAVA-BASEクラス、FJ-JAVA-CONTROLクラス、FJ-JAVA-ERRORクラスおよび生成するアダプタクラスの継承関係を以下に示します。



クラス階層

- [FJ-JAVA-BASE](#) : すべてのアダプタクラスのスーパークラスです。
- [FJ-JAVA-CONTROL](#) : Java VMの初期化・終了処理、また、スレッドのJava VMへ接続・分離処理を行います。
- [FJ-JAVA-ERROR](#) : アダプタクラスで発生する例外オブジェクトのクラスです。
- [クラスのアダプタクラス](#) : java.lang.Objectクラスのアダプタクラス (java-lang-Object) は、[FJ-JAVA-BASE](#)のサブクラスとして生成します。以下、Javaクラスと同じ継承関係を持つアダプタクラスを生成します。Javaインタフェースを実装するクラスのアダプタクラスは、インタフェースのアダプタクラスも継承します。
- [インタフェースのアダプタクラス](#) : 他のインタフェースを継承しないインタフェースのアダプタクラスは、java-lang-Objectのサブクラスとして生成します。他のインタフェースを継承するインタフェースのアダプタクラスは、Javaインタフェースと同じ継承関係を持つように生成します。
- [配列のアダプタクラス](#) : 配列のアダプタクラスは、すべてjava-lang-Objectのサブクラスとして生成します。



## 5.2 FJ-JAVA-BASEクラス

FJ-JAVA-BASEクラスは、すべてのアダプタクラスのスーパークラスです。

FJ-JAVA-BASEクラスは、以下のメソッドを持ちます。

### オブジェクトを操作するメソッド

メソッド名	種別	機能
<a href="#">J-NARROW</a>	ファクトリ	オブジェクトをサブクラスへ代入
<a href="#">J-DUPLICATE</a>	オブジェクト	アダプタオブジェクトの複製を作成
<a href="#">J-EQUALS</a>	オブジェクト	アダプタオブジェクトが指すJavaオブジェクト が一致しているか検査する

### 5.2.1 J-NARROWメソッド（ファクトリメソッド）

#### 説明

アダプタオブジェクトをサブクラスに代入します。

#### 書き方

```
INVOKE クラス名 "J-NARROW" USING object-1 RETURNING object-2
```

#### パラメタ・復帰値

クラス名

代入先データのクラス名を指定します。

*object-1*（属性： OBJECT REFERENCE FJ-JAVA-BASE）

代入するオブジェクトを指定します。

*object-2*（属性： OBJECT REFERENCE SELF）

代入先のクラスに変換したオブジェクトを返します。

### 5.2.2 J-DUPLICATEメソッド（オブジェクトメソッド）

#### 説明

アダプタオブジェクトの複製を生成します。Javaオブジェクトは複製しません。複製したアダプタオブジェクトは、元のアダプタオブジェクトと同じJavaオブジェクトを指します。

#### 書き方

```
INVOKE anObject "J-DUPLICATE" RETURNING duplicatedObject
```

#### パラメタ・復帰値

*anObject*（属性： OBJECT REFERENCE アダプタクラス）

複製もとのオブジェクトを指定します。

*duplicatedObject*（属性： OBJECT REFERENCE CLASS OF SELF）

複製したオブジェクトを返します。

### 5.2.3 J-EQUALSメソッド（オブジェクトメソッド）

#### 説明

二つのアダプタオブジェクトが同じJavaオブジェクトを指しているか検査します。

#### 書き方

```
INVOKE object-1 "J-EQUALS" USING object-2 RETURNING result
```

### パラメタ・復帰値

object-1、object-2（属性： OBJECT REFERENCE FJ-JAVA-BASE）

比較するオブジェクトを指定します。

result（属性： PIC 1）

一致する場合はB"1"、一致しない場合はB"0"を返します。

## 5.3 FJ-JAVA-CONTROL クラス

FJ-JAVA-CONTROL クラスは、Java VMの制御処理を行います。

FJ-JAVA-CONTROL クラスは、以下のメソッドを持ちます。

### Java VMを制御するメソッド

メソッド名	種別	機能
<a href="#">JVM-INIT</a>	ファクトリ	<ul style="list-style-type: none"> <li>● Java VMの初期化</li> </ul>
<a href="#">JVM-TERMINATE</a>	ファクトリ	<ul style="list-style-type: none"> <li>● Java VMの終了</li> </ul>
<a href="#">JVM-ATTACH</a>	ファクトリ	<ul style="list-style-type: none"> <li>● Java VMの初期化（初期化していない場合）</li> <li>● カレントスレッドをJava VMに接続</li> </ul>
<a href="#">JVM-DETACH</a>	ファクトリ	<ul style="list-style-type: none"> <li>● カレントスレッドをJava VMから分離</li> </ul>

### 5.3.1 JVM-INITメソッド（ファクトリメソッド）

#### 説明

Java VMを初期化します。

シングルスレッドのアプリケーションは、アダプタクラスを使用する前に必ず実行しなければなりません。

#### 書き方

```
INVOKE FJ-JAVA-CONTROL "JVM-INIT"
```

#### 環境変数

以下の環境変数を指定することにより、Java VMの実行環境をカスタマイズできます。環境変数の指定方法については、“NetCOBOL使用手引書”を参照してください。

環境変数名	機能
COBJNI_MAX_NSTACK	ネイティブコードが使用するスタックサイズの最大値をバイト単位で指定します。デフォルトは128KBです。
COBJNI_JAVA_STACK	Java コードが使用するスタックサイズの最大値を指定します。デフォルトは400KBです。
COBJNI_MIN_HEAP	メモリアロケーションプールのスタートアップサイズをバイト単位で指定します。デフォルトは、1MBです。
COBJNI_MAX_HEAP	メモリアロケーションプールの最大サイズをバイト単位で指定します。デフォルトは、16MBです。
COBJNI_CLASSPATH	実行時のクラスの検索パスを指定します。 環境変数CLASSPATHは、生成時のクラスの検索パスを指定します。実行時は意味を持ちません。

### 5.3.2 JVM-TERMINATEメソッド（ファクトリメソッド）

#### 説明

Java VMを終了します。

プロセスでアダプタクラスを使わなくなった場合に使用します。

#### 書き方

```
INVOKE FJ-JAVA-CONTROL "JVM-TERMINATE"
```

### 5.3.3 JVM-ATTACHメソッド（ファクトリメソッド）

#### 説明

Java VMが初期化されていない場合、Java VMを初期化します。また、カレントスレッドをJava VMに接続します。

マルチスレッドのアプリケーションは、アダプタクラスを使用する前にスレッドごとに必ず実行しなければなりません。

#### 書き方

```
INVOKE FJ-JAVA-CONTROL "JVM-ATTACH"
```

### 5.3.4 JVM-DETACHメソッド（ファクトリメソッド）

#### 説明

カレントスレッドをJava VMから分離します。

マルチスレッドのアプリケーションは、スレッドを終了する前にスレッドごとに必ず実行しなければなりません。

#### 書き方

```
INVOKE FJ-JAVA-CONTROL "JVM-DETACH"
```

## 5.4 FJ-JAVA-ERRORクラス

アダプタクラスは、実行時にエラーを検出した場合、例外オブジェクトを発生させます。FJ-JAVA-ERRORクラスは、その例外オブジェクトのクラスです。FJ-JAVA-ERRORクラスのメソッドを使用することにより、例外メッセージ、例外種別およびJavaの例外情報を取り出すことができます。

例外オブジェクトおよび例外オブジェクトが発生した場合の例外処理の詳細については、“NetCOBOL使用手引書”を参照してください。

FJ-JAVA-ERRORクラスは、以下のメソッドを持ちます。

### 例外情報を取得するメソッド

メソッド名	種別	機能
<a href="#">GET-MESSAGE</a>	オブジェクト	例外メッセージを返す
<a href="#">GET-CODE</a>	オブジェクト	例外種別を返す
<a href="#">GET-EXCEPTION</a>	オブジェクト	Javaの例外情報を返す

### 5.4.1 GET-MESSAGEメソッド（オブジェクトメソッド）

#### 説明

例外メッセージを返します。エラーの内容を表示するために使用します。

#### 書き方

```
INVOKE EXCEPTION-OBJECT "GET-MESSAGE" USING message RETURNING messageLength
```

#### パラメタ・復帰値

*message*（属性：PIC X ANY LENGTH）

例外メッセージを格納するデータ項目を指定します

*messageLength*（属性：PIC S9(9) COMP-5）

例外メッセージの長さ（バイト数）を返します。

### 5.4.2 GET-CODEメソッド（オブジェクトメソッド）

#### 説明

例外種別を返します。

#### 書き方

```
INVOKE EXCEPTION-OBJECT "GET-CODE" RETURNING code
```

#### 復帰値

*code*（属性：PIC S9(9) COMP-5）

[例外種別](#)が返却されます。

### 5.4.3 GET-EXCEPTIONメソッド（オブジェクトメソッド）

#### 説明

Javaの例外情報を返します。

#### 書き方

```
INVOKE EXCEPTION-OBJECT "GET-EXCEPTION" USING message messageLength class classLength
RETURNING result
```

### パラメタ・復帰値

**message** (属性: PIC X ANY LENGTH)

Javaの例外メッセージを格納するデータ項目を指定します

**messageLength** (属性: PIC S9(9) COMP-5)

Javaの例外メッセージの長さ (バイト数) を返します。

**class** (属性: PIC X ANY LENGTH)

Javaの例外クラスを格納するデータ項目を指定します

**classLength** (属性: PIC S9(9) COMP-5)

Javaの例外クラスの長さ (バイト数) を返します。

**result** (属性: PIC S9(9) COMP-5)

Java例外情報がある場合は“0”、ない場合は“-1”を返します。

### 注意事項

Javaの例外情報を取得できるのは、[例外種別](#)が“1”の場合だけです。

## 5.5 クラス／インタフェースのアダプタクラス

Javaのクラスおよびインタフェースに対応して、COBOLのクラス（アダプタクラス）を生成します。ここでは、Javaクラスおよびインタフェースを、COBOLのクラスにどのようにマッピングするか説明します。

Javaの各言語要素は、COBOLの言語要素に以下のようにマッピングします。

Java	COBOL
<a href="#">クラス</a>	クラス
<a href="#">インタフェース</a>	クラス
<a href="#">コンストラクタ</a>	ファクトリメソッド
<a href="#">クラス変数（スタティックフィールド）</a>	ファクトリのプロパティ
<a href="#">クラスメソッド（スタティックメソッド）</a>	ファクトリメソッド
<a href="#">インスタンス変数（非スタティックフィールド）</a>	オブジェクトのプロパティ
<a href="#">インスタンスメソッド（非スタティックメソッド）</a>	オブジェクトメソッド
<a href="#">Javaの基本データ型</a>	COBOLの基本データ型

なお、COBOLにマッピングするのはパブリックな要素だけです。パブリックでないクラス、インタフェース、コンストラクタ、フィールドおよびメソッドは、COBOLにはマッピングしません。

### 5.5.1 データ型

Javaのデータ型は、COBOLのデータ型に以下のようにマッピングします。

Javaのデータ型	COBOLのデータ型
boolean	PIC 1
byte	PIC X
char	PIC X(2) <ul style="list-style-type: none"> <li>● ANK文字を格納する場合、1バイト目を使用し、2バイト目はX"00"になります。</li> <li>● 日本語文字を格納する場合、2バイト使用します。</li> <li>● コード指定のオプションで-cs（シフトJIS）が指定された場合にマッピングします。コード指定のオプションが指定されていない場合も同様です。</li> </ul>
	PIC N <ul style="list-style-type: none"> <li>● ANK文字を格納する場合、1バイト目を使用し、2バイト目はX"00"になります。</li> <li>● 日本語文字を格納する場合、2バイト使用します。</li> <li>● コード指定のオプションで-cu（Unicode用）が指定された場合にマッピングします。</li> </ul>
short	PIC S9(4) COMP-5
int	PIC S9(9) COMP-5
long	PIC S9(18) COMP-5
float	COMP-1
double	COMP-2

Javaのデータ型	COBOLのデータ型
<a href="#">配列</a>	OBJECT REFERENCE 配列アダプタクラス
<a href="#">オブジェクト</a>	OBJECT REFERENCE アダプタクラス
java.lang.Stringクラス	PIC X ANY LENGTH <ul style="list-style-type: none"> <li>● <a href="#">-s オプション</a>または“<a href="#">Option String</a>” <a href="#">パラメタ</a>を指定してアダプタクラスを生成した場合にマッピングします。</li> </ul>

## 5.5.2 クラス／インタフェース

### 説明

パブリックなクラスおよびインタフェースは、COBOLのクラスにマッピングします。

アダプタクラスの継承関係は、対応するJavaクラスの継承関係と同じです。ただし、以下のクラスは、他のクラス／インタフェースを継承しません。しかし、生成するアダプタクラスは[FJ-JAVA-BASE](#)を継承します。

- java.lang.Objectクラス

また、以下のインタフェースは、他のクラス／インタフェースを継承しません。しかし、生成するアダプタクラスはjava.lang.Objectを継承します。

- 他のインタフェースを継承しないインタフェース

### 展開形式

```

CLASS-ID. 内部クラス名-1 AS “外部クラス名” INHERITS 内部クラス名-2.
...
FACTORY.
...
  《コンストラクタに対応するファクトリメソッド》
  《クラス変数に対応するプロパティメソッド》
  《クラスメソッドに対応するファクトリメソッド》
END FACTORY.
OBJECT.
...
  《インスタンス変数に対応するプロパティメソッド》
  《インスタンスメソッドに対応するオブジェクトメソッド》
END OBJECT.
END CLASS 内部クラス名-1.

```

### 生成規則

1. 内部クラス名-1および内部クラス名-2は、Jアダプタクラスジェネレータが内部的に使う名前前で、クラス利用者からは見えません。
2. 外部クラス名は、このクラスを一意に識別するための名前です。クラス利用者は、外部クラス名によりクラスを識別します。
3. 外部クラス名は、以下の規則により生成します。

[[パッケージ名](#)-[[パッケージ名](#)- ... ]] [クラス名／インタフェース名](#)

- パッケージ名で完全修飾したクラス名／インタフェース名のピリオド(.)をハイフン(-)に置換
  - 160文字を超えた場合は、161文字目以降を切り捨てる
4. ファクトリ定義には、以下の3種類のメソッドを生成します。
    - [コンストラクタに対応するファクトリメソッド](#)
    - [クラス変数に対応するプロパティメソッド](#)
    - [クラスメソッドに対応するファクトリメソッド](#)
  5. オブジェクト定義には、以下の2種類のメソッドを生成します。
    - [インスタンス変数に対応するプロパティメソッド](#)



- [インスタンスメソッドに対応するオブジェクトメソッド](#)

### 生成例

java.util.Dateクラスのアダプタクラスは、以下のように生成します。

```
CLASS-ID. J-DATE AS "java-util-Date" INHERITS J-OBJECT.      [1]
...
REPOSITORY.
  CLASS J-OBJECT AS "java-lang-Object".                      [2]
  ...
END CLASS J-DATE.
```

1. java.util.Dateクラスのアダプタクラスの名前は、java-util-Dateになります。
2. INHERITS句のJ-OBJECTの外部名は、java-lang-Objectです。つまり、java-util-Dateは、java-lang-Objectを継承しています。

### 注意事項

NetCOBOLでは、クラス名の太文字・小文字は区別しません。したがって、外部クラス名が太文字・小文字の違いしかないクラスは、同時に扱えません。

### 補足

java.lang.Stringのアダプタクラスを生成する場合、java.lang.Stringクラスで定義されたパブリックメソッドを生成します。そしてさらに、文字列データの参照・設定用メソッドを生成します（["java-lang-Stringクラス"](#) 参照）。

## 5.5.3 コンストラクタ

### 説明

パブリックなコンストラクタは、COBOLのファクトリメソッドにマッピングします。

### 展開形式

```
METHOD-ID. 内部メソッド名 AS "外部メソッド名".
...
DATA DIVISION.
LINKAGE SECTION.
01 生成オブジェクト OBJECT REFERENCE SELF.
   [パラメタ ... ]
PROCEDURE DIVISION [USING パラメタ ...] RETURNING 生成オブジェクト
   RAISING FJ-JAVA-ERROR [例外クラス名].
END METHOD 内部メソッド名.
```

### 生成規則

1. 内部メソッド名は、Jアダプタクラスジェネレータが内部的に使う名前です。クラス利用者からは見えません。
2. 外部メソッド名は、このメソッドを一意に識別するための名前です。クラス利用者は、外部メソッド名によりメソッドを識別します。
3. 外部メソッド名は、以下の規則により生成します。

**Create-Javaクラス名-nn**

- “Create-” の後に、Javaクラス名と、ハイフン (-) に続く2けたの番号 (nn) を付加
  - Javaクラス名は、パッケージ名を含まない
  - nnは、同一のJavaクラス名を持つメソッドに対し、01から順に振った番号 (01～99)
  - スーパークラスに同じJavaクラス名を持つクラスがある場合、名前の重なりを避けるために、スーパークラスから順に連続した番号を振る（["名前の番号付け"](#) 参照）
  - 160文字を超えた場合は、161文字目以降を切り捨てる
4. コンストラクタにパラメタが宣言されている場合、対応するパラメタを生成します。パラメタのデータ型の対応については、["データ型"](#) を参照してください。

5. 生成オブジェクトは、生成したアダプタオブジェクトへのオブジェクト参照を格納するオブジェクト参照一意名です。
6. コンストラクタで宣言された例外およびFJ-JAVA-ERRORを指定したRAISING指定を生成します。

### 生成例

java.util.DateクラスのコンストラクタDate()に対応するファクトリメソッドは、以下のように生成します。

```
METHOD-ID. CREATE-01 AS "Create-Date-01". [1]
...
LINKAGE SECTION.
01 CREATED-OBJECT OBJECT REFERENCE SELF.
PROCEDURE DIVISION RETURNING CREATED-OBJECT RAISING FJ-JAVA-ERROR.
...
END METHOD CREATE-01.
```

1. java-util-Dateのファクトリメソッド名は、“Create-Date”に番号を振って生成します。java.sql.Dateクラス（java.util.Dateのサブクラス）のコンストラクタDate(long)に対応するファクトリメソッドは、以下のように生成します。

```
METHOD-ID. CREATE-08 AS "Create-Date-08". [1]
...
LINKAGE SECTION.
01 CREATED-OBJECT OBJECT REFERENCE SELF.
01 PARA-1 PIC S9(18) COMP-5.
PROCEDURE DIVISION USING PARA-1 RETURNING CREATED-OBJECT RAISING FJ-JAVA-ERROR.
...
END METHOD CREATE-08.
```

1. java-sql-Dateのファクトリメソッド名も、“Create-Date”に番号を振って生成します。ただし、名前の重なりを避けるために、java-util-Dateクラスと通しで番号を振ります。

### 補足

コンストラクタからファクトリメソッド名を生成する際に、名前の一意性を保つために番号を付加します。どのコンストラクタがどのファクトリメソッドに対応するかは、クラスブラウザまたは[メソッド名対応表ファイル](#)で確認できます。クラスブラウザの場合、メソッド選択時に表示されるパラメタから識別できます。メソッド名対応表ファイルの場合、コンストラクタの引数の型から識別できます。

## 5.5.4 クラス変数

### 説明

パブリックなクラス変数（スタティックフィールド）は、COBOLのプロパティメソッド（ファクトリ）にマッピングします。

## 展開形式

```

METHOD-ID. GET PROPERTY プロパティ名.
...
LINKAGE SECTION.
01 プロパティ値 データ記述項.
PROCEDURE DIVISION RETURNING プロパティ値 RAISING FJ-JAVA-ERROR.
...
END METHOD プロパティ名.
METHOD-ID. SET PROPERTY プロパティ名.
...
LINKAGE SECTION.
01 プロパティ値 データ記述項.
PROCEDURE DIVISION USING プロパティ値 RAISING FJ-JAVA-ERROR.
...
END METHOD プロパティ名.

```

## 生成規則

1. プロパティ名は、このプロパティを一意に識別するための名前です。クラス利用者は、プロパティ名によりプロパティを識別します。
2. プロパティ名は、以下の規則により生成します。

**JF-Javaフィールド名[-nn]**

- “JF-” の後に、Javaのフィールド名を大文字に変換して付加
  - すでに同名のプロパティ名が割り当てられている場合、名前の重なりを避けるために、2番目以降のプロパティ名に対し、ハイフン (-) に続く2けたの番号 (01~99) を振る (“[名前の番号付け](#)” 参照)
  - 30文字を超えた場合は、31文字目以降を切り捨てる
3. ファイナルが指定されている場合、SET指定のプロパティメソッドは生成しません。
  4. プロパティ値は、プロパティの値の受け渡しに使うパラメタです。データ記述項は、Javaのフィールドの属性に対応するCOBOLの記述項を展開します。データ型の対応については、“[データ型](#)” を参照してください。
  5. FJ-JAVA-ERRORを指定したRAISING指定を生成します。

## 生成例

java.lang.Systemクラスのクラス変数out (スタティック・ファイナル フィールド) に対応するプロパティメソッドは、以下のように生成します。

```

METHOD-ID. GET PROPERTY JF-OUT. [1]
...
LINKAGE SECTION.
01 GET-VALUE OBJECT REFERENCE J-PRINTSTREAM. [2]
PROCEDURE DIVISION RETURNING GET-VALUE RAISING FJ-JAVA-ERROR.
...
END METHOD JF-OUT.

```

1. プロパティ名は、“JF-” にJavaのフィールド名を大文字化した “OUT” を付加して生成します。
2. outの属性はjava.io.PrintStreamなので、J-PRINTSTREAM (java-io-PrintStreamの内部クラス名) にマッピングします。

## 補足

フィールド名からプロパティ名を生成する際に、名前の一意性を保つために番号を付加します。どのフィールドがどのプロパティに対応するかは、クラスブラウザで確認できます。

## 5.5.5 クラスメソッド

### 説明

パブリックなクラスメソッド（スタティックメソッド）は、COBOLのファクトリメソッドにマッピングします。

### 展開形式

```
METHOD-ID. 内部メソッド名 AS “外部メソッド名” [OVERRIDE].
...
DATA DIVISION.
LINKAGE SECTION.
[パラメタ ... ]
[復帰値 ... ]
PROCEDURE DIVISION [USING パラメタ ...] [RETURNING 復帰値]
RAISING FJ-JAVA-ERROR [例外クラス名].
END METHOD 内部メソッド名.
```

### 生成規則

1. 内部メソッド名は、Jアダプタクラスジェネレータが内部的に使う名前です。クラス利用者からは見えません。
2. 外部メソッド名は、このメソッドを一意に識別するための名前です。クラス利用者は、外部メソッド名によりメソッドを識別します。
3. 外部メソッド名は、以下の規則により生成します。

**Javaメソッド名[-nn]**

- Javaのメソッド名をそのままCOBOLのメソッド名とする
  - すでに同名のメソッドが割り当てられている場合、名前の重なりを避けるために、2番目以降のメソッド名に対し、ハイフン（-）に続く2けたの番号（01～99）を振る（“[名前の番号付け](#)”参照）
  - 160文字を超えた場合は、161文字目以降を切り捨てる
4. メソッドにパラメタが宣言されている場合、対応するパラメタを生成します。パラメタのデータ型の対応については、“[データ型](#)”を参照してください。
  5. メソッドに復帰値が宣言されている場合、対応する復帰値を生成します。復帰値のデータ型の対応については、“[データ型](#)”を参照してください。
  6. メソッドで宣言された例外およびFJ-JAVA-ERRORを指定したRAISING指定を生成します。

### 生成例

java.lang.Mathクラスのクラスメソッドabs(long)に対応するファクトリメソッドは、以下のよう生成します。

```
METHOD-ID. JM-ABS-01 AS “abs-01”. [1]
...
LINKAGE SECTION.
01 RTN-VALUE PIC S9(18) COMP-5.
01 PARA-1 PIC S9(18) COMP-5.
PROCEDURE DIVISION USING PARA-1 RETURNING RTN-VALUE RAISING FJ-JAVA-ERROR.
...
END METHOD JM-ABS-01.
```

1. “abs”という名前のメソッドは複数宣言されているので、2番目に宣言されているabs(long)に対しては、“abs-01”という名前のメソッドを生成します。

### 補足

Javaメソッド名からCOBOLメソッド名を生成する際に、名前の一意性を保つために番号を付加します。どのJavaメソッドがどのCOBOLメソッドに対応するかは、クラスブラウザまたは[メソッド名対応表ファイル](#)で確認できます。クラスブラウザの場合、メソッド選択時に表示されるパラメ

タから識別できます。メソッド名対応表ファイルの場合、Javaメソッドの引数の型から識別できます。

## 5.5.6 インスタンス変数

### 説明

パブリックなインスタンス変数（スタティックでないフィールド）は、COBOLのプロパティメソッド（オブジェクト）にマッピングします。

### 展開形式

```
METHOD-ID. GET PROPERTY プロパティ名.
...
LINKAGE SECTION.
01 プロパティ値 データ記述項.
PROCEDURE DIVISION RETURNING プロパティ値 RAISING FJ-JAVA-ERROR.
...
END METHOD プロパティ名.
METHOD-ID. SET PROPERTY プロパティ名.
...
LINKAGE SECTION.
01 プロパティ値 データ記述項.
PROCEDURE DIVISION USING プロパティ値 RAISING FJ-JAVA-ERROR.
...
END METHOD プロパティ名.
```

### 生成規則

1. プロパティ名は、このプロパティを一意に識別するための名前です。クラス利用者は、プロパティ名によりプロパティを識別します。
2. プロパティ名は、以下の規則により生成します。

**JF-Javaフィールド名[-nn]**

- “JF-” の後に、Javaのフィールド名を大文字に変換して付加
  - すでに同名のプロパティ名が割り当てられている場合、名前の重なりを避けるために、2番目以降のプロパティ名に対し、ハイフン（-）に続く2けたの番号（01～99）を振る（“[名前の番号付け](#)” 参照）
  - 30文字を超えた場合は、31文字目以降を切り捨てる
3. ファイナルが指定されている場合、SET指定のプロパティメソッドは生成しません。
  4. プロパティ値は、プロパティの値の受け渡しに使うパラメタです。データ記述項は、Javaのフィールドの属性に対応するCOBOLの記述項を展開します。データ型の対応については、“[データ型](#)” を参照してください。
  5. FJ-JAVA-ERRORを指定したRAISING指定を生成します。

### 生成例

java.io.StreamTokenizerクラスのインスタンス変数nvalに対応するプロパティメソッドは、以下のように生成します。

```
METHOD-ID. GET PROPERTY JF-NVAL. [1]
...
LINKAGE SECTION.
01 GET-VALUE COMP-2. [2]
PROCEDURE DIVISION RETURNING GET-VALUE RAISING FJ-JAVA-ERROR.
...
END METHOD JF-NVAL.
```

```

METHOD-ID. SET PROPERTY JF-NVAL.                                [3]
...
LINKAGE SECTION.
01 SET-VALUE COMP-2.
PROCEDURE DIVISION USING SET-VALUE RAISING FJ-JAVA-ERROR.
...
END METHOD JF-NVAL.

```

1. プロパティ名は、“JF-”にJavaのフィールド名を大文字化した“NVAL”を付加して生成します。
2. nvalの属性はdoubleなので、COMP-2にマッピングします。
3. ファイナルではないので、SET指定のプロパティメソッドも生成します。

### 補足

フィールドからプロパティ名を生成する際に、名前の一意性を保つために番号を付加します。どのフィールドがどのプロパティに対応するかは、クラスブラウザで確認できます。

## 5.5.7 インスタンスメソッド

### 説明

パブリックでないインスタンスメソッド（スタティックでないメソッド）は、COBOLのオブジェクトメソッドにマッピングします。

### 展開形式

```

METHOD-ID. 内部メソッド名 AS “外部メソッド名” [OVERRIDE].
...
DATA DIVISION.
LINKAGE SECTION.
[パラメタ ... ]
[復帰値 ... ]
PROCEDURE DIVISION [USING パラメタ ...] [RETURNING 復帰値]
RAISING FJ-JAVA-ERROR [例外クラス名].
END METHOD 内部メソッド名.

```

### 生成規則

1. 内部メソッド名は、Jアダプタクラスジェネレータが内部的に使う名前です。クラス利用者からは見えません。
2. 外部メソッド名は、このメソッドを一意に識別するための名前です。クラス利用者は、外部メソッド名によりメソッドを識別します。
3. 外部メソッド名は、以下の規則により生成します。

**Javaメソッド名[-nn]**

- Javaのメソッド名をそのままCOBOLのメソッド名とする
  - すでに同名のメソッドが割り当てられている場合、名前の重なりを避けるために、2番目以降のメソッド名に対し、ハイフン（-）に続く2けたの番号（01～99）を振る（“[名前の番号付け](#)”参照）
  - 160文字を超えた場合は、161文字目以降を切り捨てる
4. メソッドにパラメタが宣言されている場合、対応するパラメタを生成します。パラメタのデータ型の対応については、“[データ型](#)”を参照してください。
  5. メソッドに復帰値が宣言されている場合、対応する復帰値を生成します。復帰値のデータ型の対応については、“[データ型](#)”を参照してください。
  6. メソッドで宣言された例外およびFJ-JAVA-ERRORを指定したRAISING指定を生成します。

### 生成例

java.util.DateクラスのインスタンスメソッドgetTime()に対応するオブジェクトメソッドは、以下のように生成します。

```
METHOD-ID. JM-GETTIME AS "getTime". [1]
...
LINKAGE SECTION.
01 RTN-VALUE PIC S9(18) COMP-5.
PROCEDURE DIVISION RETURNING RTN-VALUE RAISING FJ-JAVA-ERROR.
...
END METHOD JM-GETTIME.
```

1. “getTime”という名前のメソッドはひとつだけなので、番号をつけずに“getTime”という名前のメソッドを生成します。

## 補足

Javaメソッド名からCOBOLメソッド名を生成する際に、名前の一意性を保つために番号を付加します。どのJavaメソッドがどのCOBOLメソッドに対応するかは、クラスブラウザまたは[メソッド名対応表ファイル](#)で確認できます。クラスブラウザの場合、メソッド選択時に表示されるパラメタから識別できます。メソッド名対応表ファイルの場合、Javaメソッドの引数の型から識別できます。

## 5.6 java-lang-Stringクラス

java.lang.Stringクラスは、他のクラスと同様に、java-lang-Stringクラスにマッピングします。ただし、java-lang-Stringクラスは、java.lang.Stringクラスで定義されたメソッドに加えて、文字列データの参照・設定用に、以下のメソッドを持ちます。

メソッド名	種別	機能
<a href="#">NEW-STRING-X</a>	ファクトリ	英数字項目を初期値に、Stringオブジェクトを生成する
<a href="#">NEW-STRING-N</a>	ファクトリ	日本語項目を初期値に、Stringオブジェクトを生成する
<a href="#">GET-STRING-X</a>	オブジェクト	文字列を英数字項目として取り出す
<a href="#">GET-STRING-N</a>	オブジェクト	文字列を日本語項目として取り出す
<a href="#">GET-STRING-LENGTH-X</a>	オブジェクト	英数字項目としての長さを求める
<a href="#">GET-STRING-LENGTH-N</a>	オブジェクト	日本語項目としての長さを求める

### 5.6.1 NEW-STRING-Xメソッド（ファクトリメソッド）

#### 説明

英数字項目で指定した文字列を値として持つStringオブジェクトを生成します。

#### 書き方

```
INVOKE クラス名 "NEW-STRING-X" USING initialValue RETURNING createdObject
```

#### パラメタ・復帰値

##### クラス名

リポジトリ段落で宣言した、java-lang-Stringクラスの内部クラス名を指定します。

**initialValue** (属性: PIC X ANY LENGTH)

Stringオブジェクトの初期値を英数字項目で指定します。

**createdObject** (属性: OBJECT REFERENCE SELF)

作成したオブジェクトを返します。

#### 補足

initialValueにデータ名を指定した場合、データ項目長分のStringオブジェクトを生成します。ただし、途中にX" 00" を挿入することにより、データ項目長より短いStringオブジェクトを生成できます。

```
...
REPOSITORY.
  CLASS J-String AS "java-lang-String"
  ...
WORKING-STORAGE SECTION.
01  initialValue  PIC X(50).
01  aString       OBJECT REFERENCE J-String.
  ...
PROCEDURE DIVISION.
  ...
  MOVE "ABC" TO initialValue.
  INVOKE J-String "NEW-STRING-X" USING initialValue RETURNING aString. [1]
  ...
  MOVE "ABC" & X" 00" TO initialValue.
  INVOKE J-String "NEW-STRING-X" USING initialValue RETURNING aString. [2]
```

1. 後ろに空白を詰めた50文字のStringオブジェクトを生成します。
2. 3文字のStringオブジェクトを生成します。



## 5.6.2 NEW-STRING-Nメソッド（ファクトリメソッド）

### 説明

日本語項目で指定した文字列を値として持つStringオブジェクトを生成します。

### 書き方

```
INVOKE クラス名 "NEW-STRING-N" USING initialValue RETURNING createdObject
```

### パラメタ・復帰値

#### クラス名

リポジトリ段落で宣言した、java-lang-Stringクラスの内部クラス名を指定します。

*initialValue*（属性：PIC N ANY LENGTH）

Stringオブジェクトの初期値を日本語項目で指定します。

*createdObject*（属性：OBJECT REFERENCE SELF）

作成したオブジェクトを返します。

### 補足

*initialValue*にデータ名を指定した場合、データ項目長分のStringオブジェクトを生成します。ただし、途中にX" 0000" を挿入することにより、データ項目長より短いStringオブジェクトを生成できます。

```
...
REPOSITORY.
  CLASS J-String AS "java-lang-String"
  ...
WORKING-STORAGE SECTION.
01  initialValue PIC N(50).
01  aString      OBJECT REFERENCE J-String.
  ...
PROCEDURE DIVISION.
  ...
  MOVE NC "日本語" TO initialValue.
  INVOKE J-String "NEW-STRING-N" USING initialValue RETURNING aString. [1]
  ...
  MOVE NC "日本語" & X" 0000" TO initialValue.
  INVOKE J-String "NEW-STRING-N" USING initialValue RETURNING aString. [2]
```

1. 後ろに空白を詰めた50文字のStringオブジェクトを生成します。
2. 3文字のStringオブジェクトを生成します。

## 5.6.3 GET-STRING-Xメソッド（オブジェクトメソッド）

### 説明

Stringオブジェクトが持つ文字列を、英数字項目として取り出します。

### 書き方

```
INVOKE anObject "GET-STRING-X" RETURNING stringValue
```

### パラメタ・復帰値

#### *anObject*

文字列を取り出す、Stringオブジェクトを指定します。

*stringValue*（属性：PIC X ANY LENGTH）

Stringオブジェクトから取り出した文字列を設定します。指定したデータ項目が文字列より短い場合は、後ろを切り捨てます。指定したデータ項目が文字列より長い場合は、後ろに空白を詰めます。

### 5.6.4 GET-STRING-Nメソッド（オブジェクトメソッド）

#### 説明

Stringオブジェクトが持つ文字列を、日本語項目として取り出します。

#### 書き方

```
INVOKE anObject "GET-STRING-N" RETURNING stringValue
```

#### パラメタ・復帰値

*anObject*

文字列を取り出す、Stringオブジェクトを指定します。

*stringValue*（属性：PIC N ANY LENGTH）

Stringオブジェクトから取り出した文字列を設定します。指定したデータ項目が文字列より短い場合は、後ろを切り捨てます。指定したデータ項目が文字列より長い場合は、後ろに空白を詰めます。

### 5.6.5 GET-STRING-LENGTH-Xメソッド（オブジェクトメソッド）

#### 説明

Stringオブジェクトが持つ文字列の長さを、英数字項目の長さ（文字数）として返します。

#### 書き方

```
INVOKE anObject "GET-STRING-LENGTH-X" RETURNING stringLength
```

#### パラメタ・復帰値

*anObject*

文字列の長さを調べる、Stringオブジェクトを指定します。

*stringLength*（属性：PIC S9(9) COMP-5）

文字列の長さを格納します。

### 5.6.6 GET-STRING-LENGTH-Nメソッド（オブジェクトメソッド）

#### 説明

Stringオブジェクトが持つ文字列の長さを、日本語項目の長さ（文字数）として返します。

#### 書き方

```
INVOKE anObject "GET-STRING-LENGTH-N" RETURNING stringLength
```

#### パラメタ・復帰値

*anObject*

文字列の長さを調べる、Stringオブジェクトを指定します。

*stringLength*（属性：PIC S9(9) COMP-5）

文字列の長さを格納します。

## 5.7 配列のアダプタクラス

Javaの配列は、COBOLではオブジェクトとして扱います。そのために、配列の属性（要素の型、次元数）ごとにアダプタクラスを生成します。ここでは、Javaの配列を、COBOLのクラスにどのようにマッピングするか説明します。

### 5.7.1 配列クラス

#### 説明

以下の場合に、配列に対応するアダプタクラスを作成します。

- パブリックなコンストラクタのパラメタとして配列を使用している場合
- パブリックなフィールド（スタティク／非スタティク）の型が配列の場合
- パブリックなメソッド（スタティク／非スタティク）のパラメタおよび復帰値として配列を使用している場合

ただし、以下の属性が一致している配列は、同一のアダプタクラスに対応付けます。

- 配列要素の型
- 次元数

#### 展開形式

```
CLASS-ID. 内部クラス名-1 AS "外部クラス名" INHERITS 内部クラス名-2.
...
FACTORY.
...
《NEW-ARRAYメソッド》
END FACTORY.
OBJECT.
...
《GET-ARRAY-LENGTHメソッド》
《GET-ARRAY-ELEMENTメソッド》
《SET-ARRAY-ELEMENTメソッド》
END OBJECT.
END CLASS 内部クラス名-1.
```

#### 生成規則

1. 内部クラス名-1および内部クラス名-2は、Jアダプタクラスジェネレータが内部的に使う名前、クラス利用者からは見えません。
2. 外部クラス名は、このクラスを一意に識別するための名前です。クラス利用者は、外部クラス名によりクラスを識別します。
3. 外部クラス名は、以下の規則により生成します。

#### JA-次元数-要素型名

- “JA-”の後に、次元数（1～9）と要素型名をハイフン（-）でつないで付加する
  - 要素型名は、型に応じて以下の形式で生成する
    - オブジェクトの場合は、Javaクラスに対応するアダプタクラスの外部クラス名（“[クラス/インタフェース](#)”参照）
    - 基本型の場合は、型を表すJavaのキーワード（boolean、byte、char、short、int、long、float、double）
  - 160文字を超えた場合は、161文字目以降を切り捨てる
4. ファクトリ定義およびメソッド定義に、以下のメソッドを生成します。

メソッド名	種別	機能
<a href="#">NEW-ARRAY</a>	ファクトリ	配列オブジェクトを生成する
<a href="#">GET-ARRAY-LENGTH</a>	オブジェクト	配列の要素数を求める

メソッド名	種別	機能
<a href="#">GET-ARRAY-ELEMENT</a>	オブジェクト	配列要素から値を取り出す
<a href="#">SET-ARRAY-ELEMENT</a>	オブジェクト	配列要素に値を設定する

5. n次元の配列は、n-1次元の配列を要素として持つ1次元の配列として扱います。したがって、n次元の配列の場合、n個のアダプタクラス（JA-n-XYZ、JA-(n-1)-XYZ、…、JA-1-XYZ）を生成します。JA-n-XYZは、JA-(n-1)-XYZを要素として持つ1次元配列です。

### 生成例

java.lang.String[]のアダプタクラスは、以下のように生成します。

```
CLASS-ID. JA-1-STRING AS "JA-1-java-lang-String" INHERITS J-OBJECT.
...
END CLASS JA-1-STRING.
```

int[][][]のアダプタクラスは、以下のように生成します。

```
CLASS-ID. JA-3-INT AS "JA-3-int" INHERITS J-OBJECT. [1]
...
END CLASS JA-3-INT.
CLASS-ID. JA-2-INT AS "JA-2-int" INHERITS J-OBJECT. [2]
...
END CLASS JA-2-INT.
CLASS-ID. JA-1-INT AS "JA-1-int" INHERITS J-OBJECT. [3]
...
END CLASS JA-1-INT.
```

1. JA-2-intクラスのオブジェクトを要素として持つ1次元配列。
2. JA-1-intクラスのオブジェクトを要素として持つ1次元配列。
3. intを要素として持つ1次元配列。

### 注意事項

9次元を超えた配列は扱えません。

## 5.7.2 NEW-ARRAYメソッド（ファクトリメソッド）

### 説明

配列オブジェクトを生成します。

### 書き方

```
INVOKE クラス名 "NEW-ARRAY" USING elmNum RETURNING createdObject
```

### パラメタ・復帰値

#### クラス名

リポジトリ段落で宣言した、配列クラスの内部クラス名を指定します。

*elmNum*（属性：PIC S9(9) COMP-5）

生成する配列の要素数を指定します。

*createdObject*（属性：OBJECT REFERENCE SELF）

作成した配列オブジェクトを返します。

### 補足

多次元配列を生成する場合、n次元の配列の各要素には、n-1次元の配列を生成して格納します。たとえば、n×mの2次元配列の生成は、以下の手順で行います。

```

...
REPOSITORY.
  CLASS JA-2-INT AS "JA-2-int"
  CLASS JA-1-INT AS "JA-1-int"
  ...
01 anArray OBJECT REFERENCE JA-2-INT.
01 wArray OBJECT REFERENCE JA-1-INT.
...
  INVOKE JA-2-INT "NEW-ARRAY" USING n RETURNING anArray.          [1]
  PERFORM VARYING I FROM 0 BY 1 UNTIL I >= n
    INVOKE JA-1-INT "NEW-ARRAY" USING m RETURNING wArray          [2]
    INVOKE anArray "SET-ARRAY-ELEMENT" USING I wArray             [3]
  END-PERFORM.
  SET wArray TO NULL.

```

1. intの2次元配列オブジェクトを生成します（要素数 n）。
2. intの1次元配列オブジェクトを生成します（要素数 m）。
3. anArrayの1次元目の各要素に対し、[2]で生成した1次元配列を設定します。

### 5.7.3 GET-ARRAY-LENGTHメソッド（オブジェクトメソッド）

#### 説明

配列オブジェクトの要素数を返します。

#### 書き方

```
INVOKE anObject "GET-ARRAY-LENGTH" RETURNING elmNum
```

#### パラメタ・復帰値

*anObject*

要素数を求める配列オブジェクトを指定します。

*elmNum*（属性：PIC S9(9) COMP-5）

配列の要素数を返します。

### 5.7.4 GET-ARRAY-ELEMENTメソッド（オブジェクトメソッド）

#### 説明

配列オブジェクトの要素を取り出します。

#### 書き方

```
INVOKE anObject "GET-ARRAY-ELEMENT" USING inx RETURNING elemValue
```

#### パラメタ・復帰値

*anObject*

要素を取り出す配列オブジェクトを指定します。

*inx*（属性：PIC S9(5) COMP-5）

取り出す要素の添字を指定します。添字は0から始まります。

*elemValue*（属性：配列要素の型）

取り出した配列要素の値を格納します。

### 5.7.5 SET-ARRAY-ELEMENTメソッド（オブジェクトメソッド）

#### 説明

配列オブジェクトの要素を設定します。

## 書き方

INVOKE <i>anObject</i> "SET-ARRAY-ELEMENT" USING <i>inx elemValue</i>
---

## パラメタ・復帰値

*anObject*

要素を設定する配列オブジェクトを指定します。

*inx* (属性: PIC S9(5) COMP-5)

設定する要素の添字を指定します。添字は0から始まります。

*elemValue* (属性: 配列要素の型)

配列要素に設定する値を指定します。

## 5.8 名前の番号付け

Jアダプタクラスジェネレータでは、アダプタクラスで使用する名前を、Javaクラス／インタフェースで使用している名前から生成します。しかし、JavaとCOBOLの文法規則の違いから、単純に名前を1対1に対応付けることはできません。たとえば、Javaでは、同じクラス内に同名でパラメタの異なるメソッドを複数定義できます。しかし、COBOLではそのような定義はできません。このような場合、Javaでは同じ名前であっても、COBOLでは異なる名前に対応付けなければなりません。Jアダプタクラスジェネレータでは、Javaの名前から生成した名前にハイフン（-）に続く番号を付加し、一意な名前を生成します。ここでは、アダプタクラスで使用する名前の番号付けの規則について説明します。

### 5.8.1 名前の有効範囲

Jアダプタクラスジェネレータで扱う名前は、有効範囲により以下の三つに分類できます。

[生成規則により必ず一意になる名前](#)

[クラス名](#)

[配列クラス名](#)

[スーパークラス・サブクラス間で一意にする名前](#)

[コンストラクタに対応するファクトリメソッド名](#)

[実行単位全体で一意にする名前](#)

メソッド名 ([クラス](#)/[インスタンス](#))

フィールド (変数) 名 ([クラス](#)/[インスタンス](#))

### 5.8.2 生成規則により必ず一意になる名前

[クラス名](#)は、Javaのパッケージ名で修飾したクラス名／インタフェース名から生成します。修飾した名前は一意なので、そこから生成したCOBOL名も必ず一意になります。[配列クラス名](#)についても同様です。

クラス名および配列クラス名は、番号付けの対象になりません。

### 5.8.3 スーパークラス・サブクラス間で一意にする名前

[コンストラクタ](#)に対応するファクトリメソッド名は、パッケージ名で修飾しないクラス名から生成します。ファクトリメソッド名は、スーパークラスから継承したメソッドも含めて一意でなければなりません。しかし、以下の場合に名前が衝突します。

- スーパークラスが、パッケージの異なる同名のクラスである場合
- ひとつのクラスに、コンストラクタが複数定義されている場合

このような場合、同名のメソッドに対し、以下の規則で番号をつけます。

1. 最初に現れた名前に01をつけ、以下昇順に番号をつける。
2. 上記の規則を、スーパークラスで定義されたコンストラクタから順に適用する。ひとつのクラスに複数のコンストラクタがある場合、定義順に適用する。

#### 例

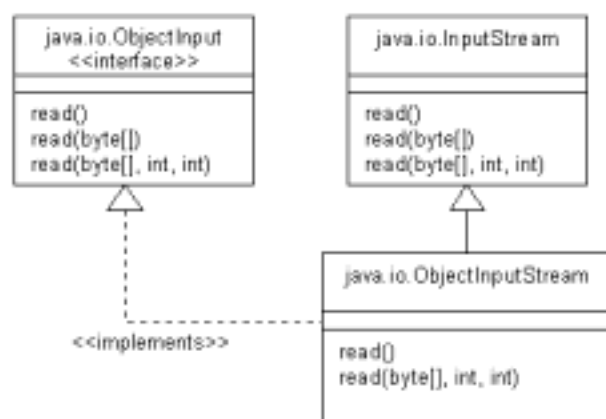
java.util.Dateクラスおよびjava.sql.Dateクラス（java.util.Dateクラスのサブクラス）のコンストラクタに対応するファクトリメソッドは、以下の名前になります。

Javaのコンストラクタ		COBOLのファクトリメソッド	
java.util.Date クラス	Date()	java-util-Date クラス	Create-Date-01
	Date(int, int, int)		Create-Date-02

Javaのコンストラクタ		COBOLのファクトリメソッド	
	Date(int, int, int, int, int)		Create-Date-03
	Date(int, int, int, int, int)		Create-Date-04
	Date(long)		Create-Date-05
	Date(String)		Create-Date-06
java.sql.Date クラス	Date(int, int, int)	java-sql-Date クラス	Create-Date-07
	Date(long)		Create-Date-08

#### 5.8.4 実行単位全体で一意にする名前

[メソッド名](#)には、Javaのメソッド名をそのまま使います。しかし、Javaでは同名でかつパラメタの異なるメソッドを複数定義できるので、COBOLのメソッド名に番号をつけて区別します。ただし、オブジェクト指向の特徴である多態性（多様性）を生かすために、同名かつ同じパラメタを持つメソッドには、常に同じ名前をつけます。この対応関係は、Jアダプタクラスジェネレータを使うCOBOLプログラムの実行単位内で、一貫していなければなりません。たとえば、java.io.ObjectInputStream クラスは、java.io.ObjectInput インタフェースと java.io.InputStream クラスを継承しています（下図参照）。



java.io.ObjectInput インタフェース、java.io.InputStream クラスおよび java.io.ObjectInputStream クラスの関係

これらに対応するアダプタクラスは、同じパラメタを持つreadメソッドに対して、同じ名前のメソッドを生成する必要があります。

Jアダプタクラスジェネレータは、クラス/インタフェースをまたがってメソッド名の対応関係を管理するために、[生成名管理ファイル](#)を使用します。そして、同名のメソッドに対し、以下の規則で番号をつけます。

1. 生成名管理ファイルに同名・同パラメタのメソッドが登録されていないか探します。
2. 見つかった場合、対応するCOBOLメソッド名を使います。
3. 見つからなかった場合、新しいCOBOLメソッド名を生成し、生成名管理ファイルに登録します。
4. COBOLメソッド名は、以下の規則で生成します。
  - 最初に登録するCOBOL名は、Javaのメソッド名と同じ
  - 2番目以降は、01から順に昇順に番号をつける

#### 例

java.io.ObjectInput インタフェース、java.io.InputStream クラスおよび



java.io.ObjectInputStreamクラスのreadメソッドに対応するオブジェクトメソッドは、以下の名前になります。

Javaのメソッド		COBOLのメソッド	
java.io.ObjectInput インタフェース	read()	java-io-ObjectInput クラス	read
	read(byte[])		read-01
	read(byte[], int, int)		read-02
java.io.InputStream クラス	read()	Java-io-InputStream クラス	read
	read(byte[])		read-01
	read(byte[], int, int)		read-02
java.io.ObjectInput Streamクラス	read()	Java-io-ObjectInput Streamクラス	read
	read(byte[], int, int)		read-02

## ■ 注意事項

アダプタクラスを生成する順番により、番号のつけ方が変わることがあります。同じ環境で使うアダプタクラスの生成には、同じ生成名管理ファイルを使用してください。

## ■ 補足

- これらの規則は、[クラスメソッド](#)にも適用します。
- Javaのフィールド（変数）（[クラス](#)/[インスタンス](#)）は、COBOLのプロパティメソッドに対応付けます。プロパティメソッドの場合も、メソッドと同じ規則を適用します。
- アダプタクラス生成時に使用するJDKおよびJ2SDKのバージョンにより、番号のつけ方が変わることがあります。



---

## 付録A メッセージ一覧

この章では、Jアダプタクラスジェネレータが出力するメッセージの内容および対処方法について説明します。

### A.1 java2cobコマンドのメッセージ

ここでは、java2cobコマンドが出力するメッセージの内容および対処方法について説明します。

**アダプタクラス生成時のメモリ不足のため実行できません。必要のないアプリケーションを終了してから、実行してください。**

#### 対処

メモリ不足です。必要のないアプリケーションを終了してから、再度java2cobコマンドを実行してください。

**Java VMの起動に失敗しました。JDKの環境定義 (PATH, CLASSPATH)、JDKのインストールが正しく行われていることを確認してください。**

#### 対処

JDKまたはJ2SDKの環境に誤りがあります。環境変数PATHおよびCLASSPATHが正しいか、また、JDKまたはJ2SDKのインストールが正しく行われているか、確認してください。

**Java2cobクラスのロードに失敗しました。Jアダプタクラスジェネレータを再インストールしてから、実行してください。**

#### 対処

Jアダプタクラスジェネレータが正しくインストールされていません。Jアダプタクラスジェネレータを再インストールしてください。

**オプションファイルのオープンに失敗しました。ファイルの状態を確認してください。生成処理を中止します。**

#### 対処

オプションファイルの状態 ([-iオプション](#)で指定したファイル名は正しいか、読み取り禁止になっていないかなど)を確認して、再度実行してください。

**オプションファイル名が指定されていません。生成処理を中止します。**

#### 対処

[-iオプション](#)にオプションファイル名を指定して、再度実行してください。

### A.2 生成時のメッセージ

ここでは、アダプタクラス生成時に出力するメッセージの内容および対処方法について説明します。

メッセージの形式を以下に示します。

メッセージ種別 : メッセージ本文
-------------------

メッセージ種別の意味を以下に示します。

メッセージ種別	レベル	意味
Error	エラー	アダプタクラスは生成されません。
Warning	警告	アダプタクラスは生成されます。ただし、利用者の意図どおりか確認する必要があります。

**Warning :不正なオプション 「xxx」 が指定されました。オプションを無視して処理を続行します。**

**対処**

正しい[オプション](#)を指定して、再度実行してください。

**Warning: クラス「xxx」に指定された「xxx」は、生成されませんでした。指定した名前(及びパラメタ)を確認してください。**

**対処**

[コンストラクタ/メソッド/フィールドの指定](#)は正しいか確認して、再度実行してください。

**Warning :ディレクトリ名が指定されていません。オプションを無視して処理を続行します。**

**対処**

正しい[オプション](#)を指定して、再度実行してください。

**Warning: 生成名管理ファイルの作成に失敗しました。**

**対処**

[-dオプション](#)または[“Option OutPutPath” パラメタ](#)に指定したフォルダ名は正しいか、生成名管理ファイル（java2cob.mgt）が書き込み禁止になっていないかなどを確認して、再度実行してください。

**Warning: クラス「xxx」に対するクラス名が160文字を超えました。160文字を超える部分を削除します。**

**対処**

160文字を超える部分を削除して問題がないか、確認してください。

**Warning: フィールド「xxx」に対するプロパティ名が30文字を超えました。30文字を超える部分を削除します。**

**対処**

30文字を超える部分を削除して問題がないか、確認してください。

**Warning: メソッド「xxx」に対するメソッド名が160文字を超えました。160文字を超える部分を削除します。**

**対処**

160文字を超える部分を削除して問題がないか、確認してください。

Warning: コンストラクタ「xxx」に対するメソッド名が160文字を超えました。160文字を超える部分を削除します。

対処

160文字を超える部分を削除して問題がないか、確認してください。

Error : クラス名またはインタフェース名が指定されていません。生成処理を中止します。

対処

正しい[クラス名](#)または[インタフェース名](#)を指定して、再度実行してください。

Error: クラス名「xxx」が正しくありません。Javaのクラス名またはインタフェース名をパッケージ名で修飾して指定してください。

対処

正しい[クラス名](#)または[インタフェース名](#)を指定して、再度実行してください。

Error : ディレクトリ名「xxx」に誤りがあります。生成処理を中止します。

対処

正しいディレクトリ名を指定して、再度実行してください。

Error: 連番が99を超えました。生成処理を中止します。

対処

[-rオプション](#)、[-gcオプション](#)、[-gmオプション](#)、[-gfオプション](#)、[-omオプション](#)、[“Class クラス名/インタフェース名”](#) パラメタの[コンストラクタ/メソッド/フィールド指定オプション](#)、または[“Option ReduceClass”](#) パラメタを指定して、再度実行してください。

Error: ファイル「xxx」が作成できない状態にあります。ファイルの状態を確認してください。

対処

ファイルの状態 ([-dオプション](#)または[“Option OutPutPath”](#) パラメタに指定したフォルダ名は正しいか、書き込み禁止になっていないかなど)を確認して、再度実行してください。

Error: ファイル「xxx」の書き込み中にエラーが発生しました。

対処

出力先に容量などの問題がないか確認して、再度実行してください。

Error : [-omオプション](#)([Option ReduceClass](#))と[-rオプション](#)([Option GenOnlyUsed](#))を同時に指定することはできません。生成処理を中止します。

対処

[-omオプション](#) (または[“Option ReduceClass”](#) パラメタ) か [-rオプション](#) (または[“Option GenOnlyUsed”](#) パラメタ) のどちらか一方を指定して、再度実行してください。[-rオプション](#) (または[“Option GenOnlyUsed”](#) パラメタ) の指定を推奨します。

Error : **-omオプション**(Option ReduceClass) と **-sオプション**(Option String) を同時に指定することはできません。生成処理を中止します。

**対処**

[-omオプション](#) (または[“Option ReduceClass”パラメタ](#)) か [-sオプション](#) (または[“Option String”パラメタ](#)) のどちらか一方を指定して、再度実行してください。

Error : オプションファイルのオープンに失敗しました。ファイルの状態を確認してください。  
生成処理を中止します。

**対処**

オプションファイルの状態 ([-iオプション](#)で指定したファイル名は正しいか、読み取り禁止になっていないかなど) を確認して、再度実行してください。

Error : オプションファイル中に不正な定義「xxx」が指定されました。定義を無視して処理を続行します。

**対処**

正しい[定義](#)を指定して、再度実行してください。

Error: クラス情報が見つかりません。生成処理を中止します。

**対処**

クラス/インタフェースから参照するクラス/インタフェースのクラスファイルがあるか確認してください ([“クラスファイルがない場合の生成方法”](#) 参照)。

Error : メモリ不足が発生しました。生成処理を中止します。

**対処**

必要のないアプリケーションを終了してから、再度実行してください。

Error: システムエラーが発生しました。生成処理を中止します。

**対処**

Jアダプタクラスジェネレータの障害です。資料を採取して技術員 (SE) に連絡してください。

## A. 3 実行時のメッセージ

ここでは、アダプタクラスを使用したプログラムの実行時に出力するメッセージの内容および対処方法について説明します。

実行時のメッセージは、COBOLのUPON SYSERR指定のDISPLAY文と同じ出力先に出力されます。UPON SYSERR指定のDISPLAY文の出力先については、“NetCOBOL 使用手引書”を参照してください。

アダプタクラスはメッセージを出力すると同時に例外オブジェクトを発生させます。アプリケーションで[例外処理](#)を記述しているなどの理由で実行時のメッセージを抑止したい場合は、環境変数COBJNI\_NOMESSAGEに“YES”を設定することで、抑止することができます。

メッセージの形式を以下に示します。

**クラス名情報 : メッセージ本文**

クラス名情報、エラーが発生したアダプタクラスを示します。クラス名情報の形式は、アダプタクラスの種別によって異なります。

アダプタクラス種別		形式	補足
クラス／インタフェース		パッケージ名/…/クラス名	パッケージ名で完全修飾したクラス名です。パッケージ名、クラス名の間は “/” で区切ります。
基本データ型の配列	boolean	[Z	“[” の数は次元数を表します。たとえば2次元の配列の場合は “[Z” になります。
	byte	[B	“[” の数は次元数を表します。
	char	[C	“[” の数は次元数を表します。
	short	[S	“[” の数は次元数を表します。
	int	[I	“[” の数は次元数を表します。
	long	[J	“[” の数は次元数を表します。
	float	[F	“[” の数は次元数を表します。
	double	[D	“[” の数は次元数を表します。
クラス／インタフェースの配列		[Lパッケージ名/…/クラス名;	“[” の数は次元数を表します。パッケージ名で完全修飾したクラス名を、“L” と “;” ではさみます。

**Java VMの初期化に失敗しました。環境変数 (PATH、COBJNI\_CLASSPATH) の値、JDKまたはJREのインストール環境を確認してください。**

#### 対処

JDK、J2SDK、JREまたはJ2REの環境に誤りがあります。環境変数PATHおよび[COBJNI\\_CLASSPATH](#)が正しいか、JDK、J2SDK、JREまたはJ2REのインストールが正しく行われているか、確認してください。

または、同一プロセス内の異なるスレッドで、[JVM-INITメソッド](#)が複数回呼び出されました。JVM-INITメソッドの代わりに[JVM-ATTACHメソッド](#)を使用してください。

**カレントスレッドをJava VMへ接続できませんでした。**

#### 対処

プログラムで[JVM-INITメソッド](#)または[JVM-ATTACHメソッド](#)を正しく呼び出しているか確認してください。

**カレントスレッドをJava VMから分離できませんでした。**

#### 対処

プログラムで[JVM-TERMINATEメソッド](#)または[JVM-DETACHメソッド](#)を正しく呼び出しているか確認してください。

**ジェネレータが生成したアダプタクラス中のJavaクラス名形式に誤りがありました。Jアダプタクラスジェネレータの提供元に連絡してください。**

#### 対処

Jアダプタクラスジェネレータの障害です。資料を採取して技術員 (SE) に連絡してください。

Javaクラス／インタフェース定義の親と子の名前が重複しています。重複しない名前に変更してください。

**対処**

Javaのクラス／インタフェース定義に誤りがあります。Javaのクラス／インタフェースを見直してください。

Javaクラス／インタフェース定義が見つかりません。環境変数 (COBJNI\_CLASSPATH) の値を確認してください。

**対処**

検索パス上で、Javaクラス／インタフェースが見つかりません。環境変数[COBJNI\\_CLASSPATH](#)の値が正しいか確認してください。

メモリ不足が発生しました。環境変数 (COBJNI\_MAX\_NSTACK、COBJNI\_JAVA\_STACK、COBJNI\_MIN\_HEAP、COBJNI\_MAX\_HEAP) の値を大きくして、実行してください。

**対処**

Java VMでメモリ不足が発生しました。環境変数 ([COBJNI\\_MAX\\_NSTACK](#)、[COBJNI\\_JAVA\\_STACK](#)、[COBJNI\\_MIN\\_HEAP](#)、[COBJNI\\_MAX\\_HEAP](#)) の値を変更して、Java VMに割り当てるメモリを増やしてください。

Javaインタフェース／抽象クラスのインスタンスは作れません。Jアダプタクラスジェネレータ実行後に、Javaクラス／インタフェースを変更していないか確認してください。

**対処**

抽象クラス上でコンストラクタを実行しました。アダプタクラス生成後にJavaクラス／インタフェースを変更した可能性があります。Javaのクラス／インタフェースを見直してください。

型変換できません。J-NARROWメソッドに渡したパラメータを確認してください。

**対処**

パラメータに指定したオブジェクトが、そのクラスまたはサブクラスのオブジェクトではありません。[J-NARROWメソッド](#)のパラメータを確認してください。

Javaフィールドが見つかりません。Jアダプタクラスジェネレータ実行後に、Javaクラス／インタフェースを変更していないか確認してください。

**対処**

アダプタクラス生成後にJavaクラス／インタフェースを変更した可能性があります。Javaのクラス／インタフェースを見直してください。

Javaクラスの初期化に失敗しました。Jアダプタクラスジェネレータの提供元に連絡してください。

**対処**

Jアダプタクラスジェネレータの障害です。資料を採取して技術員 (SE) に連絡してください。



**Javaメソッドが見つかりません。Jアダプタクラスジェネレータ実行後に、Javaクラス／インタフェースを変更していないか確認してください。**

**対処**

アダプタクラス生成後にJavaクラス／インタフェースを変更した可能性があります。Javaのクラス／インタフェースを見直してください。

**Stringオブジェクトが持つ文字列の取り出しに失敗しました。Jアダプタクラスジェネレータの提供元に連絡してください。**

**対処**

Jアダプタクラスジェネレータの障害です。資料を採取して技術員（SE）に連絡してください。

**配列オブジェクトの添字が誤っています。配列範囲内の添字を指定してください。**

**対処**

添字の値が 0 ～（要素数-1）の範囲にありません。正しい添字を指定してください。

**設定値（オブジェクト）のクラスが誤っています。配列の要素クラスのサブクラスを指定してください。**

**対処**

[配列](#)要素に、誤ったクラスのオブジェクトを設定しようとしてしました。正しいクラスのオブジェクトを設定してください。設定できるのは、配列要素クラスのオブジェクトまたはそのサブクラスのオブジェクトです。

**内部論理エラーが発生しました。（復帰値とオブジェクト参照が矛盾） Jアダプタクラスジェネレータの提供元に連絡してください。**

**対処**

Jアダプタクラスジェネレータの障害です。資料を採取して技術員（SE）に連絡してください。

**Java VMがエラーを検出しました。エラーの原因を取り除いてください。（例外名：補足情報**

**対処**

Java VMが実行時エラーを検出しました。例外名および補足情報からエラーの原因を特定し、原因を取り除いてください。

**内部論理エラーが発生しました。（エラー検出機構の障害） Jアダプタクラスジェネレータの提供元に連絡してください。**

**対処**

Jアダプタクラスジェネレータの障害です。資料を採取して技術員（SE）に連絡してください。



---

## 付録B 例外種別一覧

この章では、[FJ-JAVA-ERRORクラス](#)の[GET-CODEメソッド](#)により獲得できる例外種別について説明します。

1

### 意味

Java VMがエラーを検出しました。エラーの原因を取り除いてください。

### 対処

Java VMが実行時エラーを検出しました。[FJ-JAVA-ERRORクラス](#)の[GET-EXCEPTIONメソッド](#)で取得できるJavaの例外情報からエラーの原因を特定し、原因を取り除いてください。

2

### 意味

Javaメソッドが見つかりません。Jアダプタクラスジェネレータ実行後に、Javaクラス／インタフェースを変更していないか確認してください。

### 対処

アダプタクラス生成後にJavaクラス／インタフェースを変更した可能性があります。Javaのクラス／インタフェースを見直してください。

3

### 意味

Javaクラスの初期化に失敗しました。Jアダプタクラスジェネレータの提供元に連絡してください。

### 対処

Jアダプタクラスジェネレータの障害です。資料を採取して技術員（SE）に連絡してください。

4

### 意味

メモリ不足が発生しました。環境変数 ([COBJNI\\_MAX\\_NSTACK](#)、[COBJNI\\_JAVA\\_STACK](#)、[COBJNI\\_MIN\\_HEAP](#)、[COBJNI\\_MAX\\_HEAP](#)) の値を大きくして、実行してください。

### 対処

Java VMでメモリ不足が発生しました。環境変数 (COBJNI\_MAX\_NSTACK、COBJNI\_JAVA\_STACK、COBJNI\_MIN\_HEAP、COBJNI\_MAX\_HEAP) の値を変更して、Java VMに割り当てるメモリを増やしてください。

5

### 意味

Javaフィールドが見つかりません。Jアダプタクラスジェネレータ実行後に、Javaクラス／インタフェースを変更していないか確認してください。

### 対処

アダプタクラス生成後にJavaクラス／インタフェースを変更した可能性があります。Javaのクラス／インタフェースを見直してください。

## 6

### 意味

[配列](#)オブジェクトの添字が誤っています。配列範囲内の添字を指定してください。

### 対処

添字の値が 0 ～ (要素数-1) の範囲にありません。正しい添字を指定してください。

## 7

### 意味

ジェネレータが生成したアダプタクラス中のJavaクラス名形式に誤りがありました。Jアダプタクラスジェネレータの提供元に連絡してください。

### 対処

Jアダプタクラスジェネレータの障害です。資料を採取して技術員 (SE) に連絡してください。

## 8

### 意味

Javaクラス／インタフェース定義の親と子の名前が重複しています。重複しない名前に変更してください。

### 対処

Javaのクラス／インタフェース定義に誤りがあります。Javaのクラス／インタフェースを見直してください。

## 9

### 意味

設定値 (オブジェクト) のクラスが誤っています。[配列](#)の要素クラスのサブクラスを指定してください。

### 対処

配列要素に、誤ったクラスのオブジェクトを設定しようとしてしました。正しいクラスのオブジェクトを設定してください。設定できるのは、配列要素クラスのオブジェクトまたはそのサブクラスのオブジェクトです。

## 10

### 意味

Javaクラス／インタフェース定義が見つかりません。環境変数 ([COBJNI\\_CLASSPATH](#)) の値を確認してください。

### 対処

検索パス上で、Javaクラス／インタフェースが見つかりません。環境変数COBJNI\_CLASSPATHの値が正しいか確認してください。

## 11

### 意味

Javaインタフェース／抽象クラスのインスタンスは作れません。Jアダプタクラスジェネレータ実行後に、Javaクラス／インタフェースを変更していないか確認してください。

### 対処

抽象クラス上でコンストラクタを実行しました。アダプタクラス生成後にJavaクラス／インタフェースを変更した可能性があります。Javaのクラス／インタフェースを見直してください。

---

## 12

### 意味

Java VMの初期化に失敗しました。環境変数 (PATH、[COBJNI\\_CLASSPATH](#)) の値、JDKまたはJREのインストール環境を確認してください。

### 対処

JDK、J2SDK、JREまたはJ2REの環境に誤りがあります。環境変数PATHおよびCOBJNI\_CLASSPATHが正しいか、JDK、J2SDK、JREまたはJ2REのインストールが正しく行われているか、確認してください。

## 13

### 意味

Stringオブジェクトが持つ文字列の取り出しに失敗しました。Jアダプタクラスジェネレータの提供元に連絡してください。

### 対処

Jアダプタクラスジェネレータの障害です。資料を採取して技術員 (SE) に連絡してください。

## 14

### 意味

型変換できません。[J-NARROWメソッド](#)に渡したパラメータを確認してください。

### 対処

パラメータに指定したオブジェクトが、そのクラスまたはサブクラスのオブジェクトではありません。J-NARROWメソッドのパラメータを確認してください。

## 15

### 意味

内部論理エラーが発生しました。(復帰値とオブジェクト参照が矛盾) Jアダプタクラスジェネレータの提供元に連絡してください。

### 対処

Jアダプタクラスジェネレータの障害です。資料を採取して技術員 (SE) に連絡してください。

## 16

### 意味

Java VMがエラーを検出しました。エラーの原因を取り除いてください。(例外名: 補足情報)

### 対処

Java VMが実行時エラーを検出しました。例外名および補足情報からエラーの原因を特定し、原因を取り除いてください。

## 17

### 意味

内部論理エラーが発生しました。(エラー検出機構の障害) Jアダプタクラスジェネレータの提供元に連絡してください。

### 対処

Jアダプタクラスジェネレータの障害です。資料を採取して技術員 (SE) に連絡してください。

## 18

### 意味

カレントスレッドをJava VMへ接続できませんでした。

### 対処

プログラムで[JVM-INITメソッド](#)または[JVM-ATTACHメソッド](#)を正しく呼び出しているか確認してください。

## 19

### 意味

カレントスレッドをJava VMから分離できませんでした。

### 対処

プログラムで[JVM-TERMINATEメソッド](#)または[JVM-DETACHメソッド](#)を正しく呼び出しているか確認してください。

---

## 付録C 例題プログラム一覧

この製品では、以下のプログラムをサンプルとして提供しています。

プログラムは本製品のインストールフォルダ配下に格納されているので、コピーして使用してください。

- COBOLからJavaのjava.lang.System、java.io.PrintStreamおよびjava.util.Dateクラスを使用する例題プログラム

インストールフォルダ¥samples¥jadb01
---------------------------

- コンストラクタ／メソッド／フィールドを指定してアダプタクラスを生成する例題プログラム

インストールフォルダ¥samples¥jadb02
---------------------------

例題プログラムの詳細については、上記フォルダに格納されている“プログラム説明書”（ファイル名は“jadb01.txt”または“jadb02.txt”）を参照してください。





---

# 索引

## C

Class クラス名／インタフェース名 .....	33
CLASSPATH .....	32
COBJNI_CLASSPATH .....	45
COBJNI_JAVA_STACK .....	45
COBJNI_MAX_HEAP .....	45
COBJNI_MAX_NSTACK .....	45
COBJNI_MIN_HEAP .....	45
COBJNI_NOMESSAGE .....	72
COBOL97 .....	6
COBOL97ランタイムシステム .....	6
COBOLの英数字項目を直接受け渡す .....	22

## F

FJ-JAVA-BASEクラス .....	43
FJ-JAVA-CONTROLクラス .....	45
FJ-JAVA-ERRORクラス .....	47

## G

GET-ARRAY-ELEMENTメソッド .....	63
GET-ARRAY-LENGTHメソッド .....	63
GET-CODEメソッド .....	47
GET-EXCEPTIONメソッド .....	47
GET-MESSAGEメソッド .....	47
GET-STRING-LENGTH-Nメソッド .....	60
GET-STRING-LENGTH-Xメソッド .....	60
GET-STRING-Nメソッド .....	60
GET-STRING-Xメソッド .....	59

## J

J2RE .....	6
J2SDK .....	6
Java2 .....	6
移行 .....	27
java2cob .....	30
java2cobコマンドのメッセージ .....	69
java-lang-Stringクラス .....	58
Java VMの終了 .....	18
Java VMの初期化 .....	18
Java開発キット .....	6
Javaクラスの調査 .....	14
Javaランタイム環境 .....	6
JDK .....	6
J-DUPLICATEメソッド .....	43
J-EQUALSメソッド .....	43
J-NARROWメソッド .....	43
JRE .....	6
JVM-ATTACHメソッド .....	46
JVM-DETACHメソッド .....	46

JVM-INITメソッド .....	45
JVM-TERMINATEメソッド .....	45
Jアダプタクラスジェネレータ .....	2

## N

NEW-ARRAYメソッド .....	62
NEW-STRING-Nメソッド .....	59
NEW-STRING-Xメソッド .....	58

## O

-omオプションまたはOption ReduceClassパラメータを指定する .....	16
Option ClassPath .....	34
Option Code .....	34
Option CommandOptions .....	34
Option GenOnlyUsed .....	35
Option MethodTable .....	35
Option OutPutPath .....	35
Option OverWrite .....	35
Option ReduceClass .....	36
Option String .....	36
Option Terminal .....	36

## S

SET-ARRAY-ELEMENTメソッド .....	63
Stringオブジェクトで受け渡す .....	21

## U

Unicode .....	15, 25, 30, 34, 49
---------------	--------------------

## あ

アダプタオブジェクト .....	11
アダプタクラス .....	10
アダプタクラスの構築 .....	15
アダプタクラスのサイズ縮小 .....	16
アダプタクラスの作成 .....	14
アダプタクラスのソース生成 .....	14
アダプタクラスのソースファイル .....	38
アダプタクラスリファレンス .....	41
アプリケーションの開発 .....	18

## い

インスタンス変数 .....	55
インスタンスメソッド .....	56
メソッド .....	56
インナークラス .....	30

## え

エラー処理 .....	24
-------------	----

---

エンハンス機能 .....	7
---------------	---

## お

オブジェクト参照の比較 .....	20
オブジェクトの生成 .....	19
オプションファイル .....	33

## か

開発方法 .....	13
環境変数 .....	45, 72

## き

起動方法 .....	30
------------	----

## く

クラス	
インタフェース .....	50
クラス構成 .....	42
クラスのアダプタクラス	
インタフェースのアダプタクラス .....	49
クラスファイルがない場合の生成方法 .....	15
クラス変数	
変数 .....	52
クラスメソッド	
メソッド .....	54
クラスリテラル .....	5

## こ

コンストラクタ .....	51
コンストラクタを指定する .....	16

## さ

サブクラスへの代入 .....	21
-----------------	----

## し

ジェネレータコマンド .....	29
ジェネレータの使い方 .....	29
実行時のコード系 .....	30, 34
実行時のメッセージ .....	72
実行単位全体で一意にする名前 .....	66
シフトJIS .....	30, 34
出力 .....	38
準備するもの .....	6

## す

スーパークラス・サブクラス間で一意にする名前	65
------------------------	----

## せ

生成規則により必ず一意になる名前 .....	65
生成時のメッセージ .....	69

生成名管理ファイル .....	38
-----------------	----

## て

データ型 .....	49
できないこと .....	5
できること .....	4

## な

内部クラス .....	30
名前の番号付け	
番号付け .....	65
名前の有効範囲 .....	65

## は

配列クラス .....	61
配列のアダプタクラス	
アダプタクラス .....	61

## ふ

フィールド .....	52, 55
フィールドを指定する .....	16
プログラムの書き方 .....	18
プログラムの構築 .....	25
プログラムの実行	
実行 .....	26

## へ

変数の操作 .....	20
-------------	----

## ま

マルチスレッドアプリケーション .....	15, 18, 25
マルチスレッドアプリケーションの開発 .....	19

## め

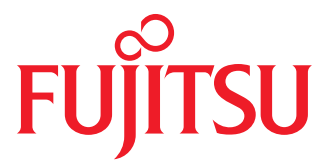
メソッド名対応表ファイル .....	39
メソッド呼出し .....	20
メソッドを指定する .....	16
メッセージ .....	69
メッセージの抑止	
実行時のメッセージの抑止 .....	72

## も

文字列の受け渡し .....	21
文字列の終端制御 .....	23

## れ

例外種別一覧 .....	77
例外処理 .....	24, 72
例題プログラム一覧 .....	81



このマニュアルはエコマーク認定の再生紙を使用しています。