1.1　パッケージとは

パッケージは、複数のクラスをグループ化したものです。パッケージを利用する主なメリットは次の通りです。

- クラスを役割ごとにまとめ、クラスの意味をわかりやすくする
- パッケージごとにクラス名を管理できるので、クラス名がつけやすくなる
- パッケージ単位にアクセス制御できる

以下のサンプルプログラムで、パッケージの定義方法と使い方を確認しましょう。

● サンプルプログラムの概要

test パッケージの UseTestPackage クラスと test.sample パッケージの TestPackage クラスを使い、メッセージを表示します。

● コンパイル・実行結果

```
> javac  -d  .  TestPackage.java

> javac  -d  .  UseTestPackage.java

> java  test.UseTestPackage
This is test message.
```

- ソースプログラム

TestPackage.java

```
01 : // test.sampleにパッケージ化
02 : package test.sample;
03 :
04 : // TestPackageクラスの定義
05 : public class TestPackage{
06 :     public static void printMessage(){
07 :         System.out.println("This is test message.");
08 :     }
09 : }
```

UseTestPackage.java

```
01 : // testにパッケージ化
02 : package test;
03 :
04 : // TestPackageクラスをインポート
05 : import test.sample.TestPackage;
06 :
07 : // UseTestPackageクラスの定義
08 : public class UseTestPackage{
09 :     public static void main(String[] args){
10 :         // TestPackageクラスのメソッド呼び出し
11 :         TestPackage.printMessage();
12 :     }
13 : }
```

1.1.1　パッケージの定義

クラスをパッケージ化するには、package 文を使用します。

【パッケージの指定】

```
package パッケージ名;
```

package 文の記述には、次のルールがあります。

- package文はソースファイルの先頭に記述する
- 1つのソースファイルにpackage文は1つしか指定できない

TestPackage.java（抜粋）

```
01: // test.sampleにパッケージ化
02: package test.sample;
03:
04: // TestPackageクラスの定義
05: public class TestPackage{
06:     :
```

コーディング規約

パッケージ名

・パッケージ名はすべて小文字にする

・インターネットドメインを逆にしたものをパッケージ名の先頭につける

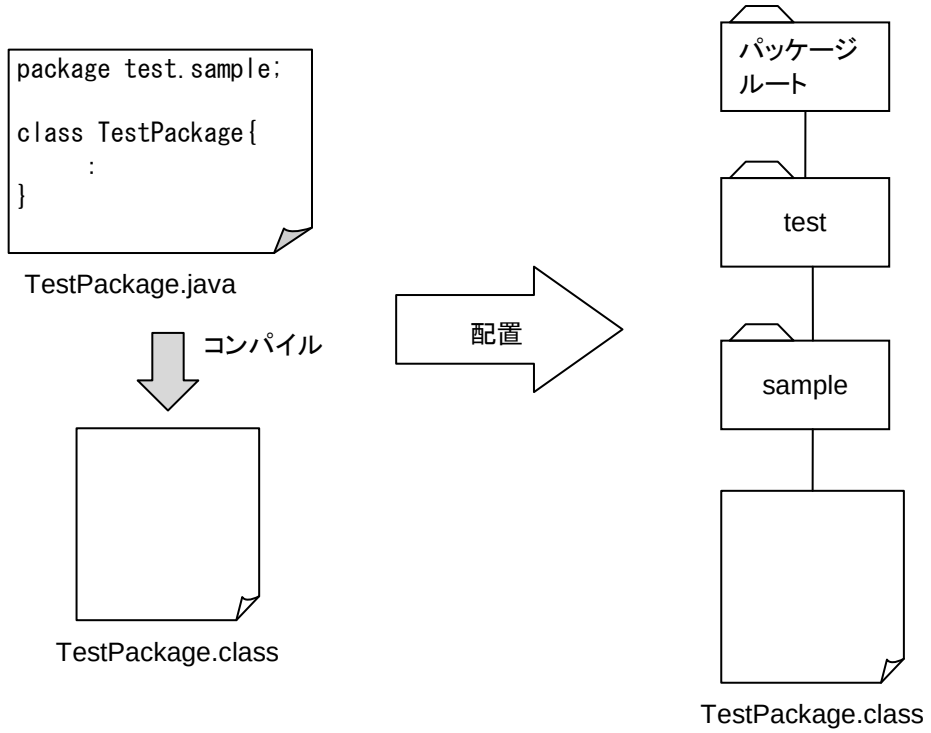
例）インターネットドメインが「nec.com」の場合

```
package  com.nec.xxx;
```

1.1.2　パッケージ化したクラスのコンパイル

パッケージ化したクラスのクラスファイルは、パッケージ名に対応するフォルダに配置します。

■ パッケージ化したクラスの配置フォルダ



javac コマンドに-d オプションをつけてコンパイルすると、次の作業を自動化できます。

1. パッケージのルートフォルダにパッケージ名に対応したフォルダを作成する
   パッケージ名に対応したフォルダが存在する場合は何も行わない
2. クラスファイルを生成し、パッケージ名に対応したフォルダに配置する

-d オプションを指定した場合の javac コマンドの書式は次の通りです。

【-d オプション】

```
> javac  -d  パッケージのルートフォルダ  ソースファイル名
```

1.1.3　パッケージ化したクラスのインポート

パッケージ化したクラスを別のパッケージのクラスが利用する場合は、クラス名の前にパッケージ名を付けます。パッケージ名とクラス名の間には区切りとして「.」を入れます。例えば、test.sample パッケージの TestPackage クラスは「test.sample.TestPackage」と指定します。パッケージ名を付けたクラス名を完全修飾クラス名といいます。

完全修飾クラス名を用いると、プログラムが長くなり、読みにくくなります。このような場合は、import 文を使用します。import 文は、パッケージ名なしで他のパッケージのクラスを利用できるようにします。

【パッケージ化したクラスのインポート】

```
import パッケージ名.クラス名;
```

```
// test.sampleパッケージのAクラスをインポート
import test.sample.A;

class UseImport{
    public void func(){
        A a = new A();                          // インポートしたクラス
        test.sample.B b = new test.sample.B();      // インポートしていないクラス
        test.sample.C c = new test.sample.C();      // インポートしていないクラス
              :
```

パッケージに含まれるすべてのクラスをインポートする場合は、import 文のクラス名の部分に「*」を指定します。

【パッケージに含まれるすべてのクラスをインポート】

```
import パッケージ名.*;
```

```
// test.sampleパッケージの全クラスをインポート
import test.sample.*;

class UseImport2{
    public void func(){
        A a = new A();
        B b = new B();
        C c = new C();
              :
```

1.2 標準ライブラリ

標準ライブラリとは、Java であらかじめ定義しているクラスライブラリです。標準ライブラリを使用すると、ディスプレイへの出力や文字列操作、DB 連携などの処理を簡単に記述できます。

1.2.1 代表的なパッケージ

Java の標準ライブラリは、同じ役割や目的を持ったクラスやインタフェースを 1 つにまとめてパッケージ化しています。ここでは、代表的なパッケージを紹介します。

パッケージ名	説明
java.lang	文字列の操作やシステムオペーレーションなど、基本的な機能を扱うクラスを提供する。
java.util	コレクションフレームワーク、イベントモデル、日時機能、国際化、およびさまざまなユーティリティクラスを提供する。
java.io	ストリーム、直列化、入出力に使用するクラスを提供する。
java.sql	データソース（通常はリレーショナルデータベース）のデータを扱うクラスを提供する。
java.awt	ユーザインタフェースの作成及びグラフィックスとイメージのペイント用クラスを提供する。
java.net	ネットワーク機能を扱うクラスを提供する。

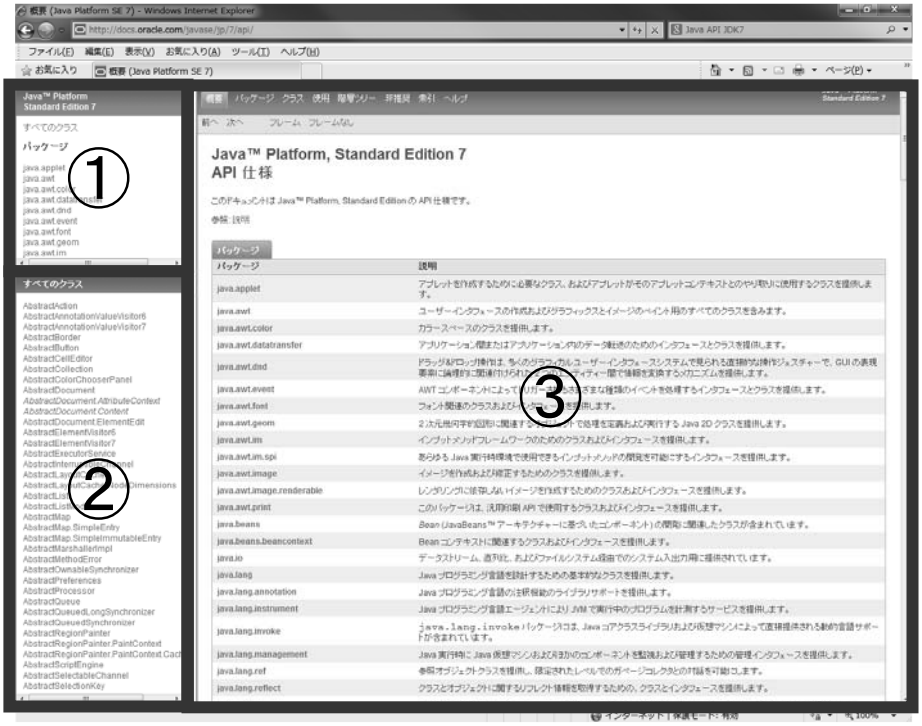
1.2.2 APIドキュメント

標準ライブラリのクラスを使いこなすには、クラスが持つメソッドやフィールドの仕様（API仕様）を知る必要があります。標準ライブラリの API 仕様は API ドキュメントとして用意されています。

API ドキュメントはクラスやインタフェースの仕様をまとめた HTML 形式のドキュメントです。このドキュメントを利用すると、標準ライブラリのクラスやインタフェースの詳細を調べられます。ここでは、API ドキュメントの参照方法を紹介します。

以下は、API ドキュメントの画面です。

URL: http://docs.oracle.com/javase/jp/7/api/



① パッケージフレーム
パッケージの一覧が表示されている。調べたいクラス、インタフェースを含むパッケージを選択する。

② クラスフレーム
①で選択したパッケージに含まれるクラス、インタフェースの一覧が表示される。起動時はすべてのクラス、インタフェースが表示されている。調べたいクラスを選択する。

③ クラスデータフレーム
②で選択したクラス、インタフェースの詳細な説明が表示される。起動時はパッケージの説明が表示されている。

1.3 java.langパッケージ

java.lang パッケージは基本的なクラスを提供するパッケージです。ここでは、java.lang パッケージに含まれるクラスの中から、次のクラスを紹介します。

- Objectクラス
- Stringクラス
- StringBuilderクラス
- Integerクラス

java.lang パッケージに含まれるすべてのクラス、インタフェースは自動的にインポートされるため、明示的な import 文の記述やパッケージ名によるクラスの修飾は不要です。

1.3.1 Objectクラス

Object クラスはすべてのクラスのスーパークラスです。クラスを定義する際にスーパークラスを指定しなかった場合は、コンパイラが「extends java.lang.Object」の記述を追加します。

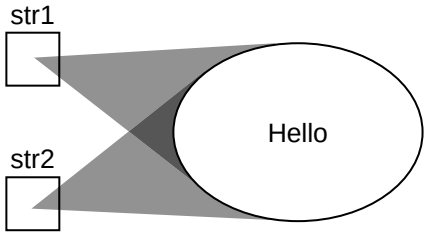
Object クラスに定義されているメソッドは、すべてのオブジェクトで利用可能です。

java.lang.Object クラスのメソッド	
public boolean equals (Object obj)	
説明	このオブジェクトと引数で指定したオブジェクトが、「等しい」オブジェクトであるかどうかを判定する。必要に応じて、サブクラスでオーバーライドする。
引数	obj 比較対象のオブジェクト
public int hashCode ( )	
説明	このオブジェクトのハッシュコード値を返す。一般にはハッシュテーブルで使用するために用意されている。
public String toString ( )	
説明	このオブジェクトの文字列表現を返す。すべてのサブクラスでオーバーライドすることが推奨されている。

1.3.2　参照型の値の比較（==演算子とequalsメソッド）

「==」を用いた比較では、2 つの変数が同じオブジェクトを参照している場合は true、異なるオブジェクトを参照している場合は false になります。

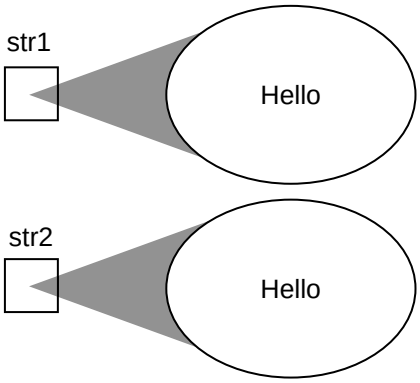
■ ==演算の結果がtrueになる場合




```
String str1 = "Hello";
String str2 = str1;

str1 == str2 ⇒ true
```

■ ==演算の結果がfalseになる場合



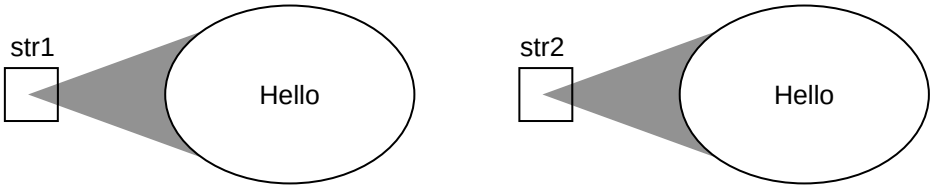

```
String str1 = new String("Hello");
String str2 = new String("Hello");

str1 == str2 ⇒ false
```

オブジェクトが等しいかどうかの判定基準は､オブジェクトの持つ意味によって変わります。そのため、「==」による比較では、オブジェクトが等しいかどうかを正しく判定できない場合があります。例えば、String オブジェクトは同じ文字列を持っていれば等しいオブジェクトと判定すべきです。しかし、「==」を用いた比較では同じ文字列を持つ 2 つのオブジェクトを異なるオブジェクトと判定します。

オブジェクトが等しいかどうかの判定には、「==」ではなく equals メソッドを使います。自分でクラスを定義するときは、等しいオブジェクトであれば「true」を返すように equals メソッドをオーバーライドします。

■ equalsメソッドによるオブジェクトの比較



str1.equals(str2) ⇒ true