● 授業のポイント

① ファイル入出力について学習します。
② ストリームの連結について学習します。
③ ファイル操作やフォルダ操作について学習します。

● 学習項目

・「File クラス（実践編 P. 171)」
・「ファイルへの出力（実践編 P. 153)」
・「ファイルからの入力（実践編 P. 160)」
・「バッファ（実践編 P. 156)」
・「ストリームの連結（実践編 P. 158)」
・「フォルダの操作（実践編 P. 173)」

● J2Kad17D「ファイルへの書き出し（FileWriter)」

１．File クラス（実践編 P.171）

File クラスはファイルを表すオブジェクトでファイルの新規作成（createNewFile）や削除（delete)、名前の変更（rename）といったファイルの操作ができます。

```
File file = new File(ファイル名);
```

なお、File クラスのインスタンス名（file）を記述すると、そのインスタンスが扱うファイル名になります（toString をオーバーライドしていると思われます)。

２．FileWriter（実践編 P.153「ファイルへの出力」）

ここでは FileWriter を使って直接ファイルへ文字列を書き込みます。手順は以下の通りです。

```
FileWriter fw = new FileWriter(file);      // file の代わりにファイル名でも OK
fw.write(書き込む文字列);
fw.close();                                // クローズ（忘れないこと）
```

なお、ファイル操作をする場合、ファイルやフォルダが存在しない、書き込みができないといった例外（IOException）が発生する可能性があるので、try～catch でくくります。

```
try {
    // ファイル操作
} catch(IOException e) {
    // 例外発生時の処理
}
```

● J2Kad17C「ファイルからの読み込み（FileReader）」

１．FileReader（実践編 P.160「ファイルからの入力」）

FileReader を使って直接ファイルからの読み込みを行います。

```
FileReader fr = new FileReader(file);
int data;
while((data = fr.read()) != -1) {                    // データの最後まで読み込む
    System.out.println((char)data + "を読み出しました！");
}
fr.close();                                          // クローズ（忘れないこと）
```

read メソッドでファイルから 1 文字ずつ読み込みます。データがなくなると-1 を返すので、それまで繰り返し読み込みを行います。

● J2Kad17B「ストリームの連結（BufferedWriter、BufferedReader）」

１．バッファ（実践編 P.156）

プリンターへ印刷を行う際、昔（すでに大昔かもしれない）はアプリから直接印刷を行っていたため、印刷が終わるまでアプリを使うことができませんでした。今ではアプリとプリンターの間にスプーラーが入っており、アプリはスプーラーに印刷データを渡してしまえば、あとはスプーラーがプリンターとやり取りするので、アプリを快適に使うことができます。このように間に１クッションおくことで処理がスムーズに流れるようになり、かつアプリ側の処理も簡単になります。

ファイル入出力も同じで、FileWriter や FileReader で頻繁にやりとりするよりもある程度データがまとまった時点で書き出したり読み込んだりする方が効率よく処理できます。このある程度データをためておく場所をバッファと呼び、BufferedWriter クラスや BufferedReader クラスがその機能を提供します。

２．ストリームの連結（実践編 P.158）

J2Kad17D や J2Kad17C では main メソッドから直接 FileWriter や FileReader を使っていましたが、ここでは間に BufferedWriter、BufferedReader を挟みます。

・Before　main メソッド → FileWriter → test.txt
・After　main メソッド → BufferedWriter → FileWriter → test.txt

main メソッドは BufferedWriter に対して書き込みたいデータを渡すだけです（あとは BufferedWriter が FileWriter へデータを渡して FileWriter がファイルへの書き込みを行います）。また BufferedWriter が提供する機能を使うこともできます（課題では改行をするための newLine メソッドを使用）。さらに main メソッドと BufferedWriter の間に PrintWriter を挟むと println メソッドで文字列に改行を付けて出力することもできます。

入力では BufferedReader を挟むことによって１行単位の入力を使うことができるようになります。

● J2Kad17A「ドットパターンの表示」

１．フォルダの操作（実践編 P.173）

File クラスではファイルだけでなくフォルダの操作も行えます。ここでは data フォルダ内のファイル一覧を表示します。ドットパターンの表示は指定したファイルの内容を表示するだけなので、J2Kad17B のモンスター表示と基本的に同じ流れです。なお、ストリームの連結は以下のように記述することも可能です。

```
BufferedReader br = new BufferedReader(new FileReader(file));  // file はファイル名でも可
```

● J2Kad17S1「文字列描画システム①（BigChar クラス）」

１．文字列描画システム

J2Kad17S1・S2・X1・X2 で文字列描画を作成します。8×8 ドットのフォントで構成される文字列を Canvas クラスへ描画します。最終的には Flyweight パターン（Singleton パターンも含む）を作成します。

２．BigChar クラス

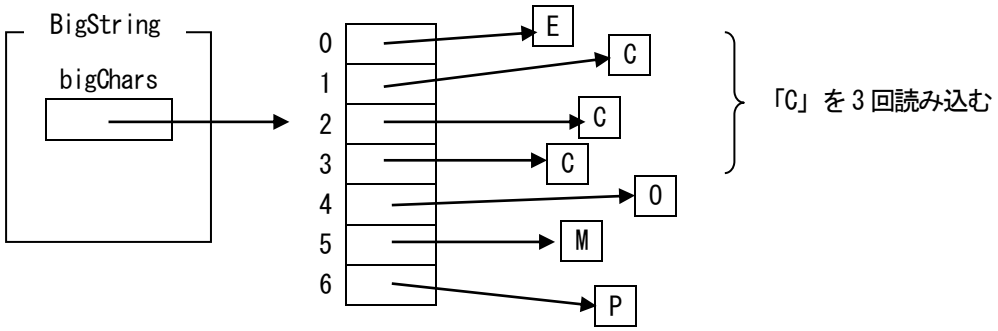
まずは8×8 ドットの文字データを表すBigChar クラスを定義して１文字だけ Canvas クラスへ描画します。BigChar クラス自体はファイルから読み込んだデータを２次元配列 fontdata に設定するだけです(特に目新しい箇所はありません)。描画も Canvas クラスの drawFont メソッドを使うだけです。

● J2Kad17S2「文字列描画システム②（BigString クラス）」

１．BigString クラス

BigChar で構成される文字列を表す BigString クラスを定義して Canvas クラスへの文字列描画を作成します。BigString クラスのコンストラクタで BigChar の配列 bigChars と文字列を構成する BigChar のインスタンスを生成し、描画では for 文で BigChar の描画を呼び出すだけです。Canvas クラスへ連続して描画を行うと自動的に文字送りをしてくれます。なお、この段階では同じ文字データを何度も読み込むようになっており、リソース管理に無駄があります。

配列 bigChars のイメージ



● J2Kad17X1「文字列描画システム③（Flyweight パターン）」

※　main メソッドは J2Kad17S2 と同じです。BigCharManager クラスの作成と BigString クラスの修正をします。

１．Flyweight パターン

Flyweight パターンとは、同じリソース（インスタンス）を複数使用するときに、1 つのインスタンスを再利用することで、省リソース化できるようにしたパターンです。ここではリソース管理をするための BigCharManager クラスを導入します。

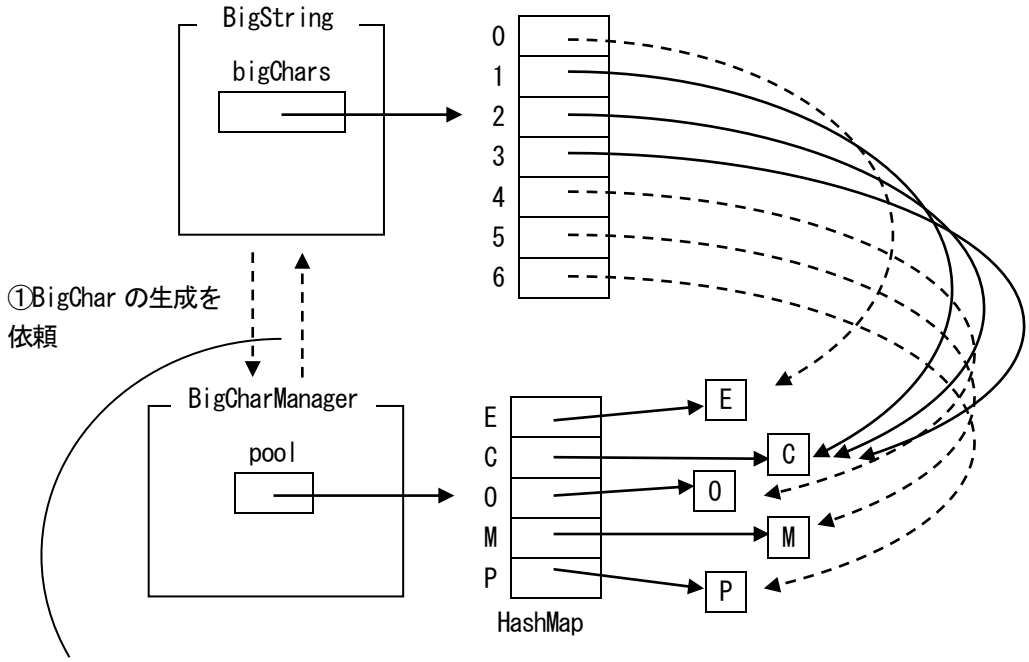
・Before　BigString が勝手に BigChar を生成
・After　BigCharManager が BigChar を生成・管理、BigString は BigCharaManager に BigChar の生成を依頼

２．BigCharManager クラス

BigCharManager は BigChar の生成と管理を行います。BigChar の生成依頼があったとき（createBigChar メソッド）次のようにします。

・指定された BigChar がすでにあるとき　その BigChar の参照を返す。
・指定された BigChar がまだないとき　BigChar を新規作成して返す（作成した BigChar は登録する）。

BigChar の生成



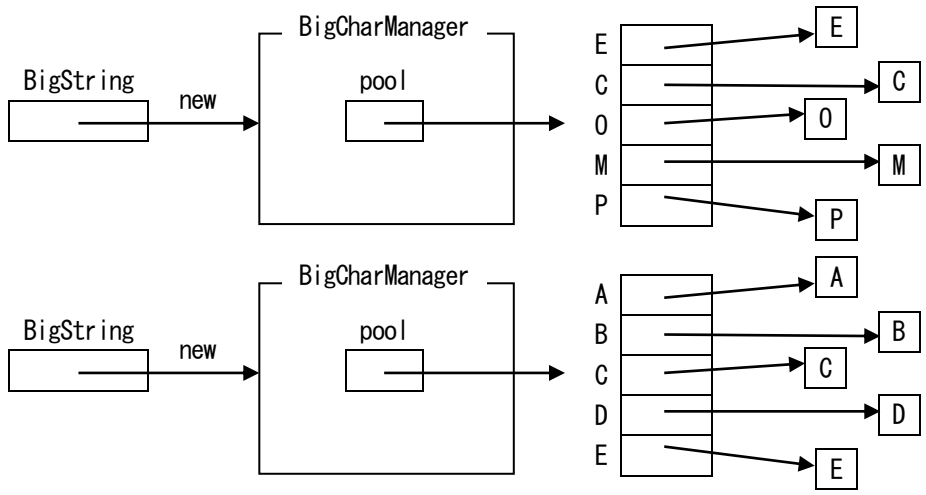
● J2Kad17X2「文字列描画システム④（Singleton パターン）」

※　main メソッドは J2Kad17S2 と同じです。BigCharManager クラスと BigString クラスを修正します。

１．リソース管理の甘さの問題

BigCharManager を導入することでリソース管理の問題は一見解決したように見えますが、BigString の描画と表示を複数回行うと同じ BigChar の読み込みが発生します。これは BigString を生成するたびに BigCharManager も生成しているためです。BigCharManager が複数存在するため、個々の BigCharManger でのリソースの重複はありませんが、BigCharManager 間での重複が発生します。

1 回目「ECCCOMP」、2 回目「ABCDE」と入力した場合（Singleton 適用前）



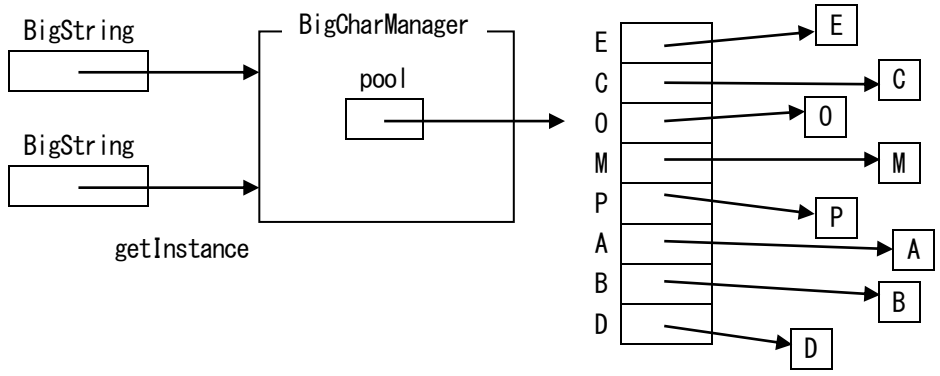
個々の BigCharManager での重複はないが、BigCharManager 間では重複が発生

２．Singleton パターン

システム内に存在するインスタンスが１つであることを保証します。外部からのインスタンス生成を禁止し、代わりにインスタンスへのアクセス手段を提供することで、システム内にそのクラスのインスタンスが１つしか作成されないようにします。Singleton パターンを適用するには次のようにします。

① コンストラクタを private にする。これによりクラス外で自由にインスタンスの生成を行うことができなくなる。
② 外部に唯一の BigCharManager のインスタンスを返す getInstance メソッドをクラスメソッドとして用意する。これにより BigCharManager を使うためには getInstance でインスタンスを取得するしかなくいつも同じ BigCharManager を使うことになる。

1 回目「ECCCOMP」、2 回目「ABCDE」と入力した場合（Singleton 適用後）



BigString が異なっても同じ BigCharManager を使うので重複が発生しない。