

20 String クラス, 入出力 と 例外処理

20-1 String クラス

- Java では、**文字列を表すために** `String` というクラスが `java.lang` という標準パッケージの中に用意されている。

[`java.lang` は Java 言語のコアとなる標準パッケージで、自動的に `import` される。]

- `String` クラスの文字列 (オブジェクト) を明示的に表したい場合は、**文字の並びを2重引用符で囲めば**よい。

[表面上は C 言語の場合と同じだ！]

- Java プログラムの中では、文字列オブジェクトを参照する変数を用意し、これらの変数や文字列定数を通して文字列の操作をする。

- 2項演算子 **+** は、**オペランド** (i.e. 演算対象) がStringインスタンスである場合は**文字列結合演算子**として振る舞い、
新たにStringインスタンスを生成。

もし、オペランドのうち1つだけがString 型であるなら、暗黙に残りの1つもString型に変換された上で文字列結合演算が適用される。

- 文字の並びは、char 型配列で表すことも出来るが、
Stringクラスのオブジェクトとして表すとStringクラスの中で定義された次のメソッドを利用することが出来る。

コンストラクタ

メソッド	機能の説明
<code>String()</code>	... 空文字列を表すString オブジェクトを生成する。
<code>String(String str)</code>	... 引数のオブジェクトをコピーする。
<code>String(char[] array)</code>	... 引数の文字配列からString オブジェクトを生成する。

メソッド	機能の説明
<code>int length()</code>	…文字列の長さを返す。
<code>char charAt(int index)</code>	…文字列の中から、インデックスが <code>index</code> の文字を取り出して返す。ここで、インデックスは配列の添字に相当するもので、文字列の先頭の文字のインデックスを 0、2 番目の文字のインデックスを 1、... と考える。
<code>boolean equals(String str)</code>	… <code>str</code> と同じ文字列かどうかを判定する。

注意：メソッドの実行は

文字列オブジェクトの指定 . メソッドの指定

という風に書き表す。実行の主体は文字列オブジェクト。

⇒ `equals` メソッドの説明は、

自分自身 (i.e. メッセージを送られた文字列オブジェクト)

と `str` が同じ文字列かどうかを判定する、

という意味である。

メソッド 機能の説明

int compareTo(string str)

... **str**と**比較**して、Unicodeの辞書順の下で **str** より小の時は負、同じ時は0、**str** より大の時は正の値を返す。

int indexOf(char ch)

... **文字chが最初に現われる場所** (インデックス) を返す。

String concat(String str)

... **文字列strを末尾につなげて**得られる文字列を新たに生成し返す。

String replace(char ch1 , char ch2)

... **文字ch1をch2で置き換え**て得られる文字列を新たに生成し返す。

String substring(int index1 , int index2)

... インデックスが **index1** から **index2-1** までの**部分文字列を切り出して**新たに文字列を生成し返す。

String toLowerCase()

... **英大文字を小文字に置き換え**て得られる文字列を新たに生成し返す。

String toUpperCase()

... **英小文字を大文字に置き換え**て得られる文字列を新たに生成し返す。

例 20. 1 (String クラス内で定義されたメソッド)

```
[motoki@x205a]$ cat -n UseMethodsInStringMain.java
```

```
1  /* Stringクラス内で定義されたメソッドの使用例 */
2
3  public class UseMethodsInStringMain {
4      public static void main(String args[]) {
5          String str1 = "Welcome-to-JAVA-World!";
6          String str2;
7
8          int index = str1.indexOf('J');
9          str2 = str1.substring(0, index);
10         System.out.println("str2(length "
11                             + str2.length() + "):  " + str2);
12         str2 = str2.concat("Java");
13         System.out.println("str2(length "
14                             + str2.length() + "):  " + str2);
15         str2 = str2.concat(str1.substring(index+4,
16                                             str1.length()));
17         System.out.println("str2(length "
18                             + str2.length() + "):  " + str2);
19     }
20 }
```

```
15      }
```

```
16 }
```

```
[motoki@x205a]$ javac UseMethodsInStringMain.java
```

```
[motoki@x205a]$ java UseMethodsInStringMain
```

```
str2(length 11): Welcome-to-
```

```
str2(length 15): Welcome-to-Java
```

```
str2(length 22): Welcome-to-Java-World!
```

```
[motoki@x205a]$
```

20-2 コマンドラインからの引数指定

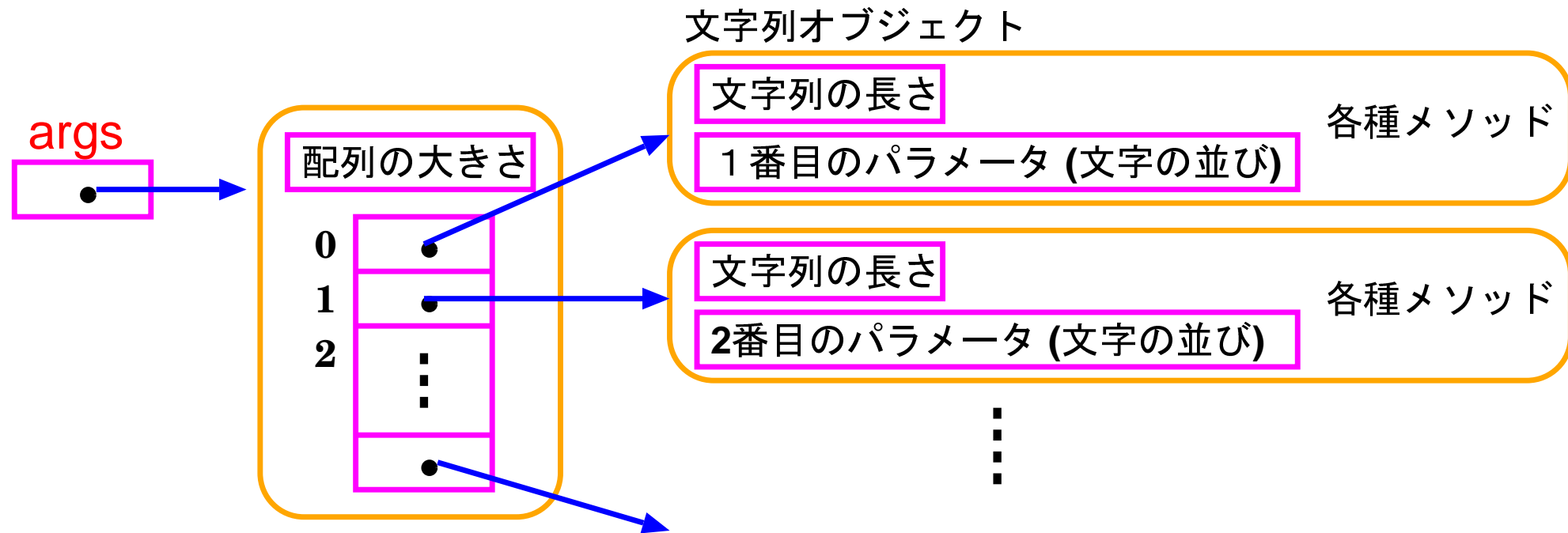
- class ファイル (バイトコード) のインタプリタを起動する際、

`java` クラス名 パラメータ 1 (文字の並び) パラメータ 2 (文字の並び) ...

という風に、パラメータ列を与えることが出来る。

- 指定するパラメータ (文字の並び) 同士は、空白で区切っておく。
- 1つのパラメータ (文字の並び) の中に空白を含めたい場合は、そのパラメータを2重引用符で囲んでおく。
- コマンドラインにおけるパラメータ受渡しの方法はC言語の場合に似ている。

main メソッドの引数部が `main(String args[])` と宣言されている場合には、main メソッドが起動された直後に args は次の様に設定される。



例 20. 2 (コマンドラインからのパラメータ指定) コマンドラインで指定したパラメータをそのまま順に表示するJavaプログラムを次に示す。

```
[motoki@x205a]$ cat -n UseCommandlineArgumentsMain.java
```

```
1  /* コマンドラインで実行パラメータを指定する例 */
2
3  public class UseCommandlineArgumentsMain {
4      public static void main(String args[]) {
5          System.out.println(
6              "Given arguments are as follows:");
7          for (int i=0; i<args.length; i++)
8              System.out.println("(" + (i+1)
9                  + "-th argument) " + args[i]);
10     }
11 }
```

```
[motoki@x205a]$ javac UseCommandlineArgumentsMain.java
```

```
[motoki@x205a]$ java UseCommandlineArgumentsMain
```

Welcome to JAVA World!

Given arguments are as follows:

(1-th argument) Welcome

(2-th argument) to

(3-th argument) JAVA

(4-th argument) World!

[motoki@x205a]\$

20-3 標準入出力の機構

- Java 言語では、標準入出力を行うオブジェクトとして、次の3つが予め用意されている。

`System.in` ... 標準入力担当のオブジェクトを表す。

正確には、クラス `System` 内の `in` という名前の `InputStream` 型クラス変数を表す。このクラス変数が標準入力担当のオブジェクトを参照しているということ。

`System.out` ... 標準出力担当のオブジェクトを表す。

正確には、クラス `System` 内の `out` という名前の `PrintStream` 型クラス変数を表す。

`System.err` ... 標準エラー出力担当のオブジェクトを表す。

正確には、クラス `System` 内の `err` という名前の `PrintStream` 型クラス変数を表す。

- オブジェクト `System.out` と `System.err` には人間が読みやすい形で出力を行うメソッド `print`, `println`, `printf` が備わっているが、オブジェクト `System.in` には入力した文字のコード番号を値として返す程度のメソッド (e.g. `read`; C 言語の `getchar` 関数の様なもの) しか備わっていない。

[これらは個々のオブジェクトの性質ではなく、元になっているクラス `PrintStream` と `InputStream` の違いによるものである。]

- Java では、プログラム実行時に起こり得るエラーを処理するために **例外処理** の機構が用意されている。

[詳しくは、次節で説明。]

例題 20. 3 (文字を入力してそのコード番号を出力; try-catch 文を用いた文字を入力してそのコード番号を出力する Java プログラムを作成せよ。

(考え方) 標準入力担当のオブジェクト `System.in` に備わっている `read` メソッドを用いて文字を入力すれば良い。

ただ、データ入力の際にはエラーの可能性があり、Java プログラムではそういったエラーにどう対処するかを指定しておかなければならない。そういった指定を明示的行いたい場合は `try-catch` 構文を用いる。

(プログラミング)

```
[motoki@x205a]$ cat -n InputCharacterMain.java
```

```
1  /* 文字を入力してそのコード番号を出力するJavaプログラム */  
2  /* (try-catch構文の使用例)  
3  
4  import java.io.*;  
5  
6  public class InputCharacterMain {  
7      public static void main(String args[]) {  
8          try {  
9              System.out.print("Input a character: ");  
10             int c = System.in.read();  
11             System.out.println("character '"  
12                                     + (char)c +  
13                                     "' has a code number "  
14                                     + c + ".");  
15             }catch(IOException e) {  
16                 System.err.println("IO Error");  
17             }  
18         }  
19     }  
20 }
```

```
16      }
```

```
17 }
```

```
[motoki@x205a]$ javac InputCharacterMain.java
```

```
[motoki@x205a]$ java InputCharacterMain
```

```
Input a character: a
```

```
character 'a' has a code number 97.
```

```
[motoki@x205a]$
```

ここで、

- プログラムの 8,13,15 行目の枠組は例外処理のための機構で、この場合は8~ 15行目は次の様に実行される。

① まず、本来の処理の流れを記述した9~ 12行目の処理を試みる。(try)

② 9行目のreadメソッド呼び出しの際に入力エラーが検出されると、IOExceptionというクラスのオブジェクト(例外という)が発生する。

[9行目のprintメソッド, 11行目のprintlnメソッドでは例外オブジェクトは発生しない。]

③ 9~ 12行目の処理の間にIOExceptionというクラスの例外が発生していれば、その例外を捕捉(catch)し14行目の処理を行う。

[13行目のeは、捕捉した例外オブジェクトを参照する変数である。この変数を使えば、例外の詳細をエラー処理に利用することも可能である。]

- 入出力関連の例外のクラスは `java.io` というパッケージの中で定義されているため、プログラム4行目の `import` 文でこのパッケージを読み込んでいる。

- プログラム 8,13,15 行目の例外処理機構を使わないと、次の様にコンパイルエラーになる。

```
[motoki@x205a]$ cat -n InputCharacterMain_test1.java  
                (コメント省略)
```

```
 4 public class InputCharacterMain_test1 {  
 5     public static void main(String args[]) {  
 6         System.out.print("Input a character: ");  
 7         int c = System.in.read();  
 8         System.out.println("character '" + (char)c + "  
 9                                 "' has a code number " + c  
10     }  
11 }
```

```
[motoki@x205a]$ javac InputCharacterMain_test1.java
```

InputCharacterMain_test1.java:7: 例外

java.io.IOException は報告されません。

スローするにはキャッチまたは、スロー宣言をしなければなりません...

```
    int c = System.in.read();  
                        ^
```

エラー 1 個

```
[motoki@x205a]$
```

```
[motoki@x205a]$ javac InputCharacterMain_test1.java
```

```
InputCharacterMain_test1.java:7: 例外
```

```
java.io.IOException は報告されません。
```

```
スローするにはキャッチまたは、スロー宣言をしなければなりません...
```

```
int c = System.in.read();
```

^

エラー 1 個

```
[motoki@x205a]$
```

補足：

エラーメッセージの日本語訳がおかしいようです。元の英文は、

```
InputCharacterMain_test1.java:7:
```

```
unreported exception java.io.IOException;
```

```
must be caught or declared to be thrown
```

というものなので、「報告されません」じゃなく「報告されていない」ということ。また、例外 `java.io.IOException` が `catch` 節に現われるか、またはメソッドの宣言において `throws` 節に現われるかしなければならない、と言っている。

例題 20. 4 (10進→2進変換; try-catch 文を用いた例外処理)

非負整数 (を表す 10 進数字列) を入力して、それを 2 進表記したものを出力する Java プログラムを作成せよ。

例題 20. 4 (10進→2進変換; try-catch 文を用いた例外処理)

非負整数 (を表す 10 進数字列) を入力して、それを 2 進表記したものを出力する Java プログラムを作成せよ。

(考え方) 10 進数字列を数として認識するには、上位の桁の数字から 1 文字ずつ読んで行き、常にそこまで表す値を保持する様にすれば良い。例えば ”234” という文字列なら $((0 \times 10 + 2) \times 10 + 3) \times 10 + 4$ と計算する。

また、認識した値 `num` の最下位の桁は `num` を 2 で割った余りとして表すことが出来るので、再帰的に考えて、`num` の 2 進表示は

`num/2` の 2 進表示 `num%2`
と表すことが出来る。

(プログラミング)

```
[motoki@x205a]$ cat -n TranslateDecimalToBinaryMain.java
 1  /* 非負整数を表す10進数字列を標準入力から読み取り、  */
 2  /* それを2進表記に直したものを出力するJavaプログラム */
 3
 4  import java.io.*;
 5
 6  public class TranslateDecimalToBinaryMain {
 7      public static void main(String args[]) {
 8          try {
 9              //10進数字列を標準入力から読み取り、
 10                                     非負整数 num として認識
 11
 12              int ch;
 13              System.out.print("Input an integer: ");
 14              while ((ch = System.in.read()) == ' ')
 15                  ;
 16              if (ch < '0' || '9' < ch)
 17                  throw new IOException();
 18          }
 19      }
 20  }
```

```
17         int num = ch - '0';
18         while ('0' <= (ch = System.in.read())
19                && ch <= '9')
20             num = 10 * num + ch - '0';
21         //非負整数 num の値を2進表記に直したものを出力
22         System.out.print("The decimal number "
23                           + num +
24                           " can be expressed as"
25                           + " a binary number ");
26         printByBinaryNotation(num);
27         System.out.println(".");
28     } catch (IOException e) {
29         System.err.println("IO Error(exit)");
30         System.exit(-1);
31     }
```



```
32      //num の値を2進表記で出力
33      private static void printByBinaryNotation(int num)
34          if (num > 1)
35              printByBinaryNotation(num/2);
36          System.out.print(num%2);
37      }
38 }
```

```
[motoki@x205a]$ javac TranslateDecimalToBinaryMain.java
```

```
[motoki@x205a]$ java TranslateDecimalToBinaryMain
```

```
Input an integer: 43
```

```
The decimal number 43 can be expressed as
```

```
a binary number 101011.
```

```
[motoki@x205a]$ java TranslateDecimalToBinaryMain
```

```
Input an integer: 555
```

```
The decimal number 555 can be expressed as
```

```
a binary number 1000101011.
```

```
[motoki@x205a]$ java TranslateDecimalToBinaryMain
```

```
Input an integer: abc
```

```
IO Error(exit)
```

[motoki@x205a]\$

20-4 例外処理

- C言語で正しくて頑健な (i.e. どんな状況に対してもうまく動く) プログラムを作ろうとすると、エラー検出とその回復のコードがプログラムのあちこちに埋め込まれ、**本来の処理の流れが見えなくなってしまう**。

例えば、

getchar 関数を使って正整数を2個入力してそれらを元に何らかの計算をするという場合は、正整数が正しく入力されたかどうかのチェックが必要である。場合によっては、**入力ミスに対して再入力を促す**などの処理を行うことになる。この様な、エラー検出／回復のコードをプログラム中に埋め込むと、往々にして本来の処理の流れが見え難くなってしまう。

- Javaでは、本来の処理の流れの明瞭さを損なわずにエラー検出等を行うために、例外処理の機構が導入されている。すなわち、
 - ① 実行中にエラーが検出されると、そのことを明らかにするために例外(exception)と呼ばれるオブジェクトを生成し送出(throw)する。
 - ② 生成された例外オブジェクトは、(必要ならメソッド呼出しの履歴を逆にたどる等もして)例外を処理してくれるコードを探す。
 - ③ 例外処理のコードが見つければそれを実行する。見つからなければ、Javaのエラー処理ルーチンがエラー報告を出して実行を中止する。

- Javaでは、大きく分けて次の3種類の例外を考える。

- ◇ エラー (error) ... アプリケーションプログラムの外側で起こり、プログラム側としては発生の予期や回復も見込めない。それゆえ「もはや実行の続行不能」と判断するしかない状態を表す。
- ◇ 実行時例外 (runtime exception) ... アプリケーションプログラムの論理的な誤り等を反映して実行時の適切な回復の可能性もあまり望めない状態を表す。
- ◇ チェックされる例外 (checked exception) ... プログラム側である程度発生の予期ができ、発生時の回復の可能性も高い状態を表す。
 - ⇒ この種の例外に限っては、コンパイル時に例外発生時の対処方法が明記されているかどうかのチェック
 - 明記されてなければ コンパイルエラー

- Javaでは次の様な種類(クラス)の例外が想定されています。

例外クラスの名前	例外発生の原因
<u>エラー</u>	
OutOfMemoryError	...新しいオブジェクトのためにメモリが確保できなかった。
StackOverflowError	...再帰の回数が多過ぎてスタックの容量が足りなくなった。
.....	

例外クラスの名前	例外発生の原因
<u>実行時例外</u>	
ArithmeticException	...整数を0で割るなどの算術演算エラー。
ArrayIndexOutOfBoundsException	...配列の添字の値が許容範囲を超えている。
NullPointerException	...オブジェクトの指定が必要な場面で null を使おうとした。
NumberFormatException	...数を表すはずの文字列を数として読み取ることが出来なかった。
.....	

例外クラスの名前	例外発生の原因
<u>チェックされる例外</u>	
<code>IOException</code>	...入出力時に異常が起きた。
<code>FileNotFoundException</code>	...指定したファイルが見つからない。
<code>EOFException</code>	...ファイルの終端記号を読んだ。
<code>ClassNotFoundException</code>	...指定したクラスが見つからない。
.....	

- 例外を意識的に送出するには次の様を書く。

```
throw new 例外クラス名 ( 引数; 省略可 )
```

- Javaでは、発生した例外を処理するために次の2つの機構が用意されています。

try-catch 文

… 前節の例 20.3 ~ 20.4 で紹介した機構。送出される可能性のある例外の各々に対して別個の処理を記述できる。

throws 節

… 指定した例外の処理を呼出し元のメソッドに任せることを (メソッド本体の直前に) 宣言する。(→ 例 20.5)

try-catch 文を用いた例外処理 :

● 構文 :

```

try {
    本来の処理
} catch ( 例外クラス名 変数名 ) {
    発生した例外オブジェクト
    に対する処置
} catch ( 例外クラス名 変数名 ) {
    ⋮
} finally {
    try ブロックや catch ブロック
    の後に実行する処置
    (この中で return 文 不可)
}

```

try ブロック
 0 個以上の
catch ブロック
 0 ~ 1 個の
finally ブロック

- 効果： 次の流れに従って実行が進む。
 - ① まず、本来の処理の流れを記述した `try` ブロックの処理が試行される。この中でエラー等が検出されると、それに相当する例外オブジェクトが送出され `try` ブロックの処理は中止される。
 - ② `try` ブロック終了時点に例外オブジェクトが発生していれば、`catch` 節が上から順に調べられる。
 - ◇ もし、送出された例外と同じ (またはその上位の) クラスが `catch` 節の中で宣言されていれば、`catch` 節の中の仮引数が例外オブジェクトを参照するようにセットされ、その `catch` ブロックの処置が施される。(調べられずに残った `catch` 節は無視される。)
 - ◇ 該当する `catch` 節が見つからない場合は、実行中のメソッドを呼び出した外部のメソッドに対して例外が送出される。
 - ③ `try` ブロックが正常に終了した場合も、`try` ブロックで例外が発生して例外処理された場合も、最後に (もしあれば) `finally` ブロックが実行される。

- 注意 (finally ブロック) :

finally ブロックの中で return 文や break 文を書くと

→ どういう経緯で finally ブロックに入ったかの情報

(e.g. try ブロックが正常に完了,
try ブロックを return 文で出た,
try ブロック内で例外が送出された)

が上書きされ失われてしまい、

→ 例えば

try ブロック内で実行されたはずの return 文が無視されたり、
catch されなかった例外が消え失せてしまったり、
といったことが起こる。

⇒ finally ブロック内では return 文や break 文は不可

- 使用例 : 例題 20.3, 例題 20.4,, 例題 23.2, 例題 23.3,

throws 節を用いた例外処理 :

- 構文 : メソッド定義の形を次の様にする。
[キーワード throws から処理の本体ブロックの前までが
throws 節。]

```
修飾子 型 メソッド名 ( 仮引数の並び )  
    throws 例外クラス名 , ..., 例外クラス名 {  
        処理の本体  
    }
```

- 効果 : このメソッドを呼び出したメソッドに、
指定した例外処理を全面的に任せてしまう。
- 使用例 : 例題 20.5

例題 20. 5 (文字を入力してそのコード番号を出力; throws節を用いた例)

例題 20.3 では、try-catch 構文を用いて文字を入力してそのコード番号を出力する Java プログラムを作成した。ここでは、同等の処理を行うプログラムを try-catch 構文ではなく throws 節を用いることによって書き表せ。

(考え方) 例題 20.3 で作ったプログラムを次の様に書き換えるだけ

- try-catch の構文全体 → try ブロック内の処理の列
- main メソッドのヘッダ部の最後に throws 節を追加する。

(プログラミング)

```
[motoki@x205a]$ cat -n InputCharacterMain2.java
```

```
1  /* 文字を入力してそのコード番号を出力する Java プログラム */
2  /* (throws 節の使用例)
3
4  import java.io.*;
```

```
5
6 public class InputCharacterMain2 {
7     public static void main(String args[])
8                                     throws IOException {
9         System.out.print("Input a character: ");
10        int c = System.in.read();
11        System.out.println("character '" + (char)c +
12                               "' has a code number " +
13                               c + ".");
14    }
15 }
```

```
[motoki@x205a]$ javac InputCharacterMain2.java
```

```
[motoki@x205a]$ java InputCharacterMain2
```

```
Input a character: a
```

```
character 'a' has a code number 97.
```

```
[motoki@x205a]$
```
