**文字を格納できるchar型！？**

あれ!? char型って整数を格納するデータ型だったんじゃないの？

その通り！だけどchar型は、Unicode(ユニコード)という文字コード規格での一文字を、0～65535の範囲の整数で表すこともできるんだ！

テキスト

自動的に生成された説明

Javaのデータは「文字としてではなくUnicodeの文字コードに基づいて」を格納しているんだ。例えば、「A」であればUnicodeの文字コードでは、「65」(16進数表記だと0x41)といった風に、それぞれの文字に0～65535までの一意（onlyone）の番号が割り当てられているんだ。

文字と数字の紐付け方には、たくさんのルールがあるんだ。Javaではその一つであるUnicodeが最初から使用されていて、その中のUTF-16というルールが使われているんだ！このUTF-16の“16”は、16ビットが処理単位だよということで、だからcharは16ビットなんだ！

同じようなルールには、日本語環境ではいわゆるシフトJISやEUC-JP、JISなどがあるんだ。Unicodeの中でも、UTF-8やUTF-32というものもある。Javaでもこれらのルールはもちろん扱えることができることを知っておこう！

サンプルソース

**public** **class** UnicodeExample {

**public** **static** **void** main(String[] args) {

// 文字列を定義

String myString = "A";

// 文字列からUnicodeのコードポイントを取得

**int** unicodeCodePoint = myString.codePointAt(0);

// Unicodeのコードポイントを16進数で表示

String hexRepresentation = Integer.*toHexString*(unicodeCodePoint);

// 結果を出力

System.***out***.println("文字列: " + myString);

// 文字列: A

System.***out***.println("Unicodeのコードポイント: " + unicodeCodePoint);

// Unicodeのコードポイント: 65

System.***out***.println("16進数表記: 0x" + hexRepresentation.toUpperCase());

// 16進数表記: 0x41

}

}

## Stringとcharの違い

あれ？先ほどのcharでも文字を格納することができたけど、Stringとcharって何が違うんですか？

char型とString型の大きな違いは、「扱える文字の範囲（文字数）」なんだ！

**String**では、**複数の文字列**を表すことができる

でもchar型は、**1文字だけ**しか表すことができないんだ。

下の例を見てみよう

ｃhar型で複数の文字列を入れた状態で実行するとエラーになっているのがわかるよね！

これがchar型とString型の違いになるから覚えてこう！

グラフィカル ユーザー インターフェイス, テキスト, アプリケーション

自動的に生成された説明

実行結果

グラフィカル ユーザー インターフェイス, テキスト

自動的に生成された説明

エラーの意味を説明するよ。

「java.lang.Error: Unresolved compilation problem」

で「未解決のコンパイル問題があります」。

何が原因かというと「文字定数が無効」つまり文字数に問題がありそうだね！

問題のある部分というのが、

「at practice/string.CharType.main(CharType.java:7)」

で　CharType.javaファイルの７行目に問題があることを教えてくれてるよ。

**メモリと変数**

まず、イメージ話なんだけど、私たちが配列を作成した時って下の図を使って理解を進めてきたよね。

だけど、これは「私たち人間が理解しやすいイメージ図」であって「コンピュータの中では図のような構造になっていない」のが現実なんだ。

ダイアグラム

中程度の精度で自動的に生成された説明

ゲームの画面

中程度の精度で自動的に生成された説明そもそも、コンピュータは使用するデータを「メモリ」に記録するんだ。みんなも学校で使うものは、ロッカーや机の中にしまっていつでも使えるようにしておく感覚と一緒かな

そして、そのメモリの中は、囲碁や将棋盤見たく区画がきちんと整理されていて、その区画には住所（アドレス）が振られているんだ。

テーブル

自動的に生成された説明

土地に例えると、わかりやすいかもしれないね。土地がいくつかあって、そこには住所がちゃんとある。

そして、その土地に、家が建てられていたら、もうその土地に新たに家を建てることはできないから、空いている土地を探す必要がある。この土地がメモリというイメージで家がデータというイメージがつけると良いかもしれない

ダイアグラム が含まれている画像

自動的に生成された説明

まず、変数を宣言すると、空いている区画（空いている土地）を変数（データ（新たな家を作る））のために、確保するんだ。で、その時に気を付けるのがデータ型（家の大きさ）！

使用するデータ型によって容量が異なったよね！

家も住む人数によって家の大きさを調整するのと一緒で、使用するデータ型によってどのくらい区画（土地）を使うのか異なるんだ。

そして、使うデータ型（家の大きさ）がきまったら、確保していた区画（土地）に、値（家）を、記録する（建てる）ことになるんだ。

では、早速なんだけど、例えば、以下の配列を作成したとしよう

テキスト

自動的に生成された説明

まず、メモリ上でどうなっているかを見ていくよ

まず例として、int型の３つの要素を格納する配列を作成したとき、メモリの中ではどのようになっているかを考えていくね。

ダイアグラム

自動的に生成された説明

配列変数の宣言によってint[]型変数が、new演算子によって配列の実態（要素の集まり）がメモリ上の区画に作成されるんだ。

そして、配列変数には、3つの要素まるごとじゃなくて「最初の要素のアドレス」が代入されるんだ。

図形, 四角形

自動的に生成された説明

**配列変数名[n]と指定されたら**

図を見ながら考えていこう！scores[n]と指定されたら、要素の先頭の番地を見つけて、見つけたら、配列の先頭の要素からn個後ろの要素の区画を読み書きするんだ。

Index番号（添え字の番号）が0から始まるのは、定規の考え方と似ていて、定規のスタートは１ｃｍからではなく,

0からスタートだよね。それと一緒で添え字も0からスタートして、自分自身つまり、0から何個後ろにあるかというのを考える必要があるんだ。下の例では、1081から1個後ろを見たいから１を指定しているよね。

グラフィカル ユーザー インターフェイス が含まれている画像

自動的に生成された説明

図形, 四角形

自動的に生成された説明

**配列の後始末**

**ガベージコレクション**

諒君どう？配列の裏側は理解できそうかな？

はい！大丈夫そうです。

そういえば疑問なんですけど、例えば以下のソースを見てほしいのですが、

前回のif文の時に、if文で宣言した変数はif文の中でしか使えないといってたと思うのですが、

この場合、メモリの中ってどうなっているのでしょうか。

グラフィカル ユーザー インターフェイス, テキスト, アプリケーション

自動的に生成された説明

なるほど！いい質問だね！確かにif文の中に入れない限りつかわれないのだからずーっとメモリの中にあったらじゃまだよね。

じゃあソースを見ていこう

グラフィカル ユーザー インターフェイス

自動的に生成された説明

このif文が通った場合だけ配列の初期化をしているのがこのソースだと思うんだけど、

諒君のいった通り、変数には寿命があって、自分が宣言されたブロックが終了するまでがその寿命だったよね。この考え方をスコープというのだけど詳しくは第4章でお話をするね。

つまり、このソースは、8 行目の時点で配列変数getCoinsは消滅する。なぜかといえブロックの範囲外（寿命が切れる）から。だけど、ここで紛らわしいのが、初期化された要素3つ分。

グラフィカル ユーザー インターフェイス, テキスト, アプリケーション

自動的に生成された説明

つまりソースのこの赤で囲んだ部分だね。この3つの要素は普通の変数じゃないからブロックが終了してもメモリに残ってしまうんだ。その結果、何が起こるのかというと、配列はどの配列変数からも参照されない状態になるからメモリに残ってしまうんだ。

グラフィカル ユーザー インターフェイス, アプリケーション

自動的に生成された説明

残ってしまった配列は、JAVAのプログラムからどのような方法を使ったとしても、読み書きすることはできないから、メモリの中ではごみになるよね。

ゴミとなった配列を放置し続けると当たり前だけどメモリを圧迫しちゃうよね。

皆も部屋のごみをすてるも忘れて部屋が汚くなるということになったことはない？

それと一緒なんだ。

本来であれば、「使用できなくなった配列（ゴミ）は、使わないのだから、破棄してメモリの領域を確保する」というメモリの後片付けを指示しないといけない、私たちの日常に例えると、ごみ収集業者にゴミを持って行ってもらうことをしないといけないんだ。

だけど、JAVAにはガベージコレクションという仕組みが常に動いていて、実行中のプログラムが生み出したメモリ上のゴミを自動的に探して片づけてくれるんだ。

へー！すごいですね。自動掃除機のルンバみたいですね。

**Stringの配列**

そういえば、String型の配列ってメモリはどうなっているのでしょうか？

というのも、配列は参照型だったからメモリの番地を格納するということだったけど、Stringも同じだったじゃないですか？この場合、メモリ中ってどうなっているのですか？

確かに、そうだね。答えを言うなら、「参照の参照」をしているんだけど、言葉だけだど、意味不明だから、図を見ながら一緒に確認していくね！

時計, テレビ が含まれている画像

自動的に生成された説明

なるほど、String型の配列の場合は、要素の中に値ではなく、「アドレス値」は入るのですね！

だから参照の参照なんですね！

確かに、言葉だけだとわかりづらいけど、図で見ると、イメージが付きやすいです！

良かった！物事の本質を知らないと、エラーやプログラムを書く時に、困ってしまうから、知識の一環として理解しておこう！

**Null**

先生、たまにソースを書いていてnullと出てきたりするじゃないですか

そのnullってなんなのですか？

確かにそうだね、Nullというのは、「何もない」という意味なんだけど、それだけだとイメージが付きにくいからトイレットペーパーでイメージをしてもらうよ。

座る, 記号, メーター, 時計 が含まれている画像

自動的に生成された説明

値が0の状態の時は、トイレットペーパーの芯は残っているけど紙が残ってないから、お尻がふけない状態。

値がnullの状態は、もはやトイレットペーパーの芯すら残ってない状態。

じゃあ、いまの考えを踏まえたうえで、ソースコードをみながらnullについて詳しく見ていこう！

テキスト

中程度の精度で自動的に生成された説明

テキスト

自動的に生成された説明

グラフィカル ユーザー インターフェイス, アプリケーション

自動的に生成された説明このソースを理解するためにまずは状況を図で確認していこう！

なるほど、nullを入れた場合は、index番号0の先頭番地を見ることができないから、何もない状態と一緒になるということですね！

その通り、さすがだね！

じゃあ、繰り返しfor文で要素の中身を確認した場合、どうなるのですか？

とても良い質問だね！では実行して内容を確認してみよう

テキスト

自動的に生成された説明

テキスト

自動的に生成された説明

なんかとんでもないエラーがでてますね！

これはどういう意味のエラーなのですか？

「NullPointerException」というエラーで、「参照型変数にnullが格納されているときに、参照型変数を参照しようとした場合」に、発生する例外エラーというものになるだ。

ダイアグラム

自動的に生成された説明

なるほど、何もないのに、探そうとしたって、そりゃあ探せないので、エラーの内容については、納得です。

もう一つの疑問なのですが、配列を初期化させた場合に、その値の中にnullが入っている場合は、どうなるのでしょうか？

確かに、当然の疑問だね。

まずは、ソースを実行した後、図で一緒に内容を確認していこう！

お願いいたします！

テキスト

自動的に生成された説明

テキスト

自動的に生成された説明

今度は、nullが入っているにも関わらず、「NullPointerException」がでていないのですね。

色々と、頭の中がこんがらがっています。どうしてエラーがでてこないのでしょうか？

テレビゲームの画面

自動的に生成された説明確かにそう思うのは当然だと思うから、次は現状を図で確認して理解を深めていくね！

なるほど、先ほどの例では、そもそも参照先がない。つまり１０８１番地そのものがないからエラーになってしまったものに対して、今回は、箱そのものは存在しているから、エラーがおきていないのですね！

その通り！

なるほど、どこにnullを入れるかによってとても意味が違うのですね！非常に勉強になります！

それは良かった！

ここまでが、配列の基本なのだけど、理解の方は大丈夫そう？

何となくです。ただ、まだ頭の中の整理が完璧ではないです。

そうだよね。この部分は概念として本当に大事なんだ。どうしてかというと、他の言語でも同じ考え方をするから。

例えばC言語とか、この部分の理解を深めておくと、非常に言語の取得はスムーズなんだ。だからもう一度自分で図を書いて頭の中を整理整頓しておこう！

**多次元配列**

**多次元配列とは？**

先生、「多次元配列」と聞いただけでちょっと嫌なんですけど。

だって言葉通りにとらえてたら、配列がたくさんあるみたいな意味ですよね。

まあ、諒君の言っていることは間違えではないね。

多次元配列というのは、簡単に言えば、「配列の配列を作れる」という意味なんだ。

今回は多次元配列の中の2次元配列を例に話をしていくよ

まずは、復習なのだけど、今までの配列って下の図のように考えてきたよね。

時計と文字の加工写真

中程度の精度で自動的に生成された説明

これを「1次元配列」という言い方をするんだ。

じゃあ2二元配列はどうかというと下の図のように、横のならびに縦の並びが追加されるんだ。

ダイアグラム

自動的に生成された説明

多次元配列というのは、「2次元以上の配列」のことを多次元配列と呼ぶことも覚えておいてほしいかな！

で、現場でこれが使われるかというと、あまり使う機会は正直少ないと思う。だけど科学計算技術等では多く利用されているんだ。だから皆がお目にかかる機会は少ないと思うけど、考え方は非常に大事だから覚えておいてね！

では、早速構文を見ていこう

テキスト

自動的に生成された説明

構文を見たところで、早速ソースコードを書いてみよう

テーブル

自動的に生成された説明

インデクス番号指定の仕方が複雑すぎて意味不明ですね。

グラフィカル ユーザー インターフェイス, アプリケーション

自動的に生成された説明確かにそうだね。ではどうやって指定するのかの部分を図で確認していこう

なるほど、このように表で考えるとわかりやすいのですね！

これであれば、僕も指定できそうです。例えば3を表示させたい場合は、縦のindex番号が0横のindex番号が２

だから、[0][2]と指定すればよいということですよね！

その通り！さすが諒君、理解が早いね！ではメモリ上はどうなっているのかも一緒に確認していくよ！

ダイアグラム

自動的に生成された説明

なるほど、メモリの中はこんな感じになっているのですね、

図があると、どういう状態なのかが非常にわかりやすいです。

それは、良かった、それでは下のソースは、何行何列の配列かわかる？

グラフィカル ユーザー インターフェイス

自動的に生成された説明

いやわからないです。

だよね。どうやって何行何列の配列なのかを判断するのか一緒に見ていくよ！

ゲームの画面

低い精度で自動的に生成された説明

なるほど！｛｝の中にある{}の個数が、縦列の数になって

｛｝の中にある値が横列の個数になるのですね！  
  
その通り！これで、初期化された２次元配列を見てもすぐに何行何列か判断することができそうだね！