7-3.Stringクラス型

Stringはデータ型ではない

Stringは、いつも何気なく使用しているかと思いますが、仕様については少し注意が必要です。
細かい仕様についての問題は、JavaSilverにおいて頻出です。

```
String a = "aaa";
int b = 123;
char c = 'a';
```

Javaには、`boolean,byte,char,short,int,long,float,double`などの様々なデータ型がありますが、 **Stringはデータ型の仲間ではありません。**
Stringは、クラス型になります。
JavaのAPIの、`java.lang`パッケージ内に、Stringクラスが存在しており、それを使用しています。

比較すると、データ型と違い、Stringだけ大文字から始まる点が異なりますね。
また、普段クラスのインスタンスを作成し、クラス型のオブジェクトに代入する際、

```
Employee employee = new Employee();
//Employeeクラス型のオブジェクトemployeeを作成し、Employeeクラスのインスタンスを格納
```

のように記述しますが、クラス型の記述は先頭が大文字であり、Stringはこちらの仲間であることが分かります。

クラス型であれば、通常はクラスのインポートが必要になります。
しかし、Stringクラスが入っている`java.lang`パッケージはjavaにおいて必須機能が集まっているため、import文を明記しなくても、自動的にimportされている仕組みになっています。
よって、初めから記述が可能となっています。

Stringクラスのインスタンスは、

```
String a = "aaa";
```

のように、ダブルクォーテーションで括られた文字列(文字列リテラルと呼びます)を記述するだけで作られます。
また、他のクラスと同じように、

```
String a = new String("a");
```

とすることでインスタンスを作成することができます。

しかし、この2つの記述方法でも仕様が異なるので注意が必要です。

コンスタントプール

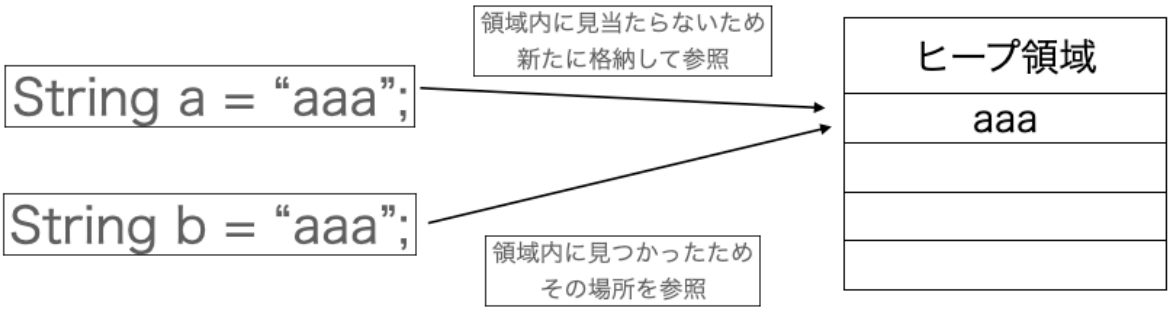
例として、以下のような記述を考えます。

?

```
public class Main {
    public static void main(String[] args) {
        String a = "aaa";
        String b = "aaa";
        String c = new String("aaa");

        System.out.println("---equals---");
        System.out.println(a.equals(b));
        System.out.println(a.equals(c));
        System.out.println(a.equals("aaa"));

        System.out.println("--- == ---");
        System.out.println(a == b);
        System.out.println(a == c);
        System.out.println(a == "aaa");

        System.out.println("--- intern ---");
        System.out.println(a == c.intern());
        System.out.println(a.intern() == c);

        String d = new String("aaa").intern();
        System.out.println(a == d);
    }
}

//出力結果
---equals---
true
true
true
--- == ---
true
false
true
--- intern ---
true
false
true
```

文字列リテラルでの記述は、プログラム中に頻繁に現れますが、その度にStringクラスのインスタンスを生成していては、処理の不可が増え、大量のメモリを消費してしまうことになります。

それを回避するため、Stringではコンスタントプールという仕組みを使っています。

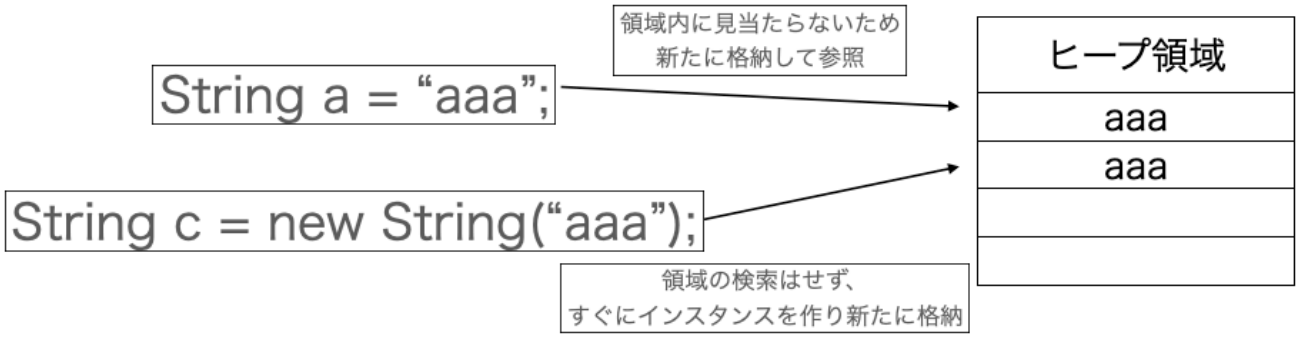


Javaでシステムを実行する際、メモリ内に様々なデータを格納しますが、
その一部にヒープ領域と呼ばれる部分が存在します。
ここには`new クラス名()`などで作成したインスタンスなどが格納されますが、Stringクラスのインスタンスもここに入ります。

文字列リテラルでStringを記述した際、まずはこの領域を検索します。
`String a = "aaa";` と実行した時、領域を検索し、同じ文字列がみあたらないので、新しく1つの部屋に文字が入ったインスタンスを格納、その場所を参照します。

その後`String b = "aaa";` と実行した時、領域を参照し、同じ文字列が見つかるとします。
その時、オブジェクトbは新たにインスタンスを作ることはなく、同じ文字列が入っている場所を参照する仕組みとなっています。

注意なのがnewでインスタンスを作成した場合です。



`String c = new String("aaa");` などとした場合、先程のように領域内を検索することは無く、同じ文字列があるか無いかに関わらず、新しくインスタンスを作成し、別の場所に格納されます。

よって、オブジェクト`a,b,c`には全て`aaa`という文字列が指定されていますが、
a,bは1つの同じ場所を参照、cだけ別の場所を参照しているということになります。

Stringの比較

同じ文字列かどうかを比較する方法として、`equals`メソッドがありました。

```
System.out.println(a.equals(b)); //true
System.out.println(a.equals(c)); //true
System.out.println(a.equals("aaa")); //true
```

`equals`は、領域の参照場所は関係無く、単純に文字列が一致するかどうかを調べます。

一方、`==`を使うと、文字列そのものではなく、参照するヒープ領域の場所が一致するかどうかを調べます。

```
System.out.println(a == b); //true
System.out.println(a == c); //false
System.out.println(a == "aaa"); //true
```

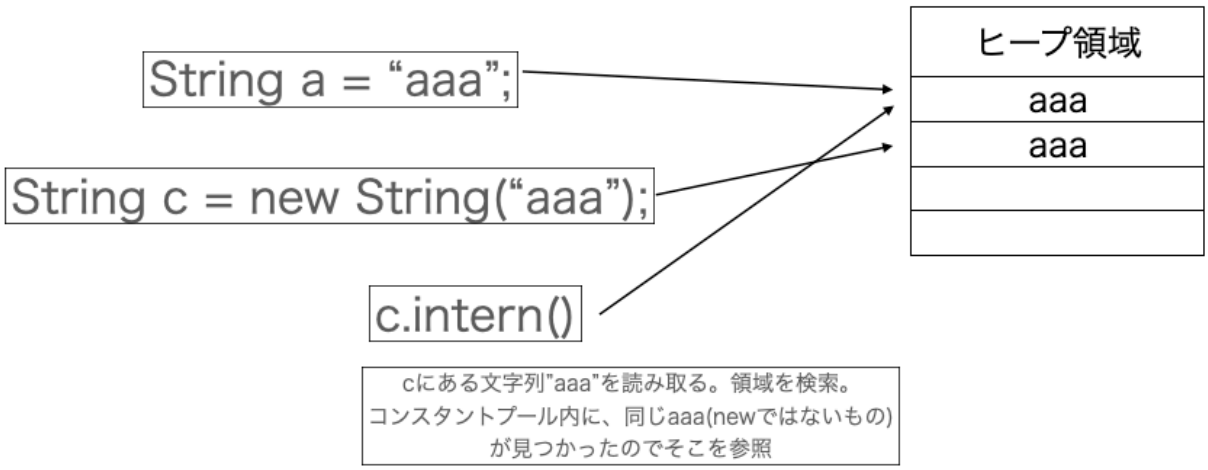
オブジェクトaとbは、同じ場所を参照していました。なのでtrueが返ります。
cだけは、別の場所を参照していましたね。なので`a == c`はfalseが返ります。

"aaa"と文字リテラルを書くと、また領域内を検索します。
`String a = "aaa";` で作成した時の領域が見つかり、そこを参照します。
よって、`a == "aaa"`は両辺とも同じ場所を参照しているので、trueが返ります。

internメソッド

また、文字列の参照に関わるinternメソッドについて説明します。

internメソッドは、文字列を読み込んで、それと同じ文字列がコンスタントプール内にあるかを確認します。
見つかれば、その場所を参照します。
見つからなければ、その文字列をコピーしてプール内に作成し、そこを参照します。
つまり、`String a = "aaa";` と、文字リテラルで記述した時と全く同じような処理になります。



`c.intern()`と書くと、cにある文字列`aaa`をまず読み取ります。
`aaa`という文字列がプール内にあるか検索します。
見つかったので、そこを参照します。

よって、`a == c.intern()`はどちらも同じ場所を参照していることになるのでtrueとなります。

このような仕様を考えると、

```
String a = "aaa";
String a = new String("aaa").intern();
```

上記の2行は、全く同一の動きとなります。

これらの仕様はとてもややこしいかと思いますので、自分でも色々と試しながら確認してみましょう！
普段システムを作る際はそこまで気にせずとも大丈夫なポイントですが、
JavaSilverでは問われるので把握は必要となります！

課題

満点を取れるまで小テストを受験して下さい。

制限時間: 30 分

評定方法: 最高評点

合格点: 100 / 100

受験件数: 94