文字列を扱う時のポイント

１．==演算子

等値であるかを調べる。

プリミティブ型であれば、値が等しいか否か、参照型であれば、アドレスが等しいか否かを調べる。

プリミティブ型は、変数が直接値を保持する型　int,charなど

**参照型**は、オブジェクトへの参照を保持する型　Stringなど

２．equalsメソッド

等価であるかを調べる。

保持する値（Stringクラスの場合は文字列）が等しいか否かを調べる。

３.Stringクラスはイミュータブル（不変）

一度作成したインスタンスの状態を変更することができない

変更するならば、新たにインスタンスが作成される

同じ文字列は再利用される

４．+演算子

文字列を連結する際、+演算子を使った方が可読性が高い。

文字列定数同士の連結なら、速度低下もないので、+演算子を使う方が適切。

5. StringBuilderクラス

Stringクラスでは文字列を結合する際に新しいオブジェクトを作成するが、StringBuilderクラスは既存のオブジェクトを変更する

そのため、ループ内で大量の文字列を連結する場合などはStringBuilderクラスを使うべき

数値を扱う時のポイント

丸め誤差を許さず、正確に計算したいときはBigDecimalクラスを使用する

丸め誤差とは

コンピュータで数値を扱う際に、ある桁よりも小さな数 字を削除することにより、本来の値との間にわずかに生じる誤差

１．“+”演算子などは使えない

２．四則演算はメソッドで行う

インスタンスの値と引数の間で四則演算を行う

• 加算：add(BigDecimal)

• 減算：subtract(BigDecimal)

• 乗算：multiply(BigDecimal)

• 除算：divide(BigDecimal)

３．Stringと同じくイミュータブル。

四則演算のメソッドも、値を変更しない。

コレクションフレームワークを扱う時のポイント

コレクションフレームワークとは、コレクション（データの集合）を 操作するためのJava標準のAPI

インターフェースベースで設計

List

順序有り。重複可。

Set

順序無し。重複不可。

Map

キーと値のペアを格納。 キーの重複不可。

使い方

・宣言方法

左辺はインターフェース名 右辺は実装クラス名

左辺をインターフェース名にする理由　保守性向上

・実装クラスの使い分け

Listの場合

基本的にArrayList

リストの途中に対する追加や削除が頻発するなら、LinkedList

理由

ArrayListの実現方法は配列、LinkedListの実現方法はリスト構造

Setの場合

基本的にHashSet

順序が必要ならTreeSet

Mapの場合

基本的にHashMap

順序が必要なTreeMapかLinkedHashMap

TreeMapはキーでソートされる

LinkedHashMapは追加された順でソートされる

・繰り返し処理

Listの場合

拡張for文を利用

Mapの場合

Map.Entryインターフェースを利用