Kotlin Collections

Types of Collections in Kotlin

Kotlin provides various collection types to manage and manipulate data. Each type is classified based on whether it is mutable (modifiable) or immutable (non-modifiable).

1. **List**:

- Ordered collection of elements.
- Elements can be accessed by their **index**.
- Immutable by default (`listOf`).

2. **MutableList**:

- A modifiable version of `List` .
- Can add, remove, or update elements dynamically.

3. **Set**:

- A collection of **unique elements**.
- Immutable by default (`setOf`).
- Does **not allow duplicates**, ensuring all elements are distinct.

4. **MutableSet**:

- A modifiable version of `Set` .
- Can add or remove elements dynamically.

5. **Map**:

- A collection of **key-value pairs**.
- Keys must be **unique**.
- Immutable by default (`mapOf`).

6. **MutableMap**:

- A modifiable version of `Map` .
- Keys and values can be added, updated, or removed.

Set Example

A `Set` in Kotlin is a generic collection of unique elements. The example below demonstrates a `Set` of Strings:

```
val computer: Set<String> = setOf(
    "Mark 4",
    "Personal computer",
    "Tablet",
    "Cell phone"
)
```

- This `Set` contains four unique elements.
- It is immutable and ensures no duplicate entries.

Map Example

Maps are collections that store data in **key-value pairs**. The keys must be unique, but the values can be duplicated. A common example might map strings to integers, like:

```
val nameToAge: Map<String, Int> = mapOf(
    "Alice" to 30,
    "Bob" to 25,
    "Charlie" to 35
)
```

ForEach, Map, and Filter Operations

Kotlin collections provide powerful operations for iterating, transforming, and filtering data. Let’s dive into each one:

ForEach:

- Iterates through each element in a collection.
- Example:

```
val numbers = listOf(1, 2, 3, 4)
numbers.forEach { println(it) }
```

Output:

```
1
2
3
4
```

Map:

- Applies a transformation function to each element in the collection.
- Creates a new collection with the transformed values.
- Example:

```
val numbers = listOf(1, 2, 3, 4)
val squared = numbers.map { it * it }
println(squared)  // Output: [1, 4, 9, 16]
```

Filter:

- Filters elements based on a condition.
- Returns a new collection containing only the elements that satisfy the condition.
- Example:

```
val cookies = listOf(Cookie("Choco", true), Cookie("Oatmeal", false))
val softBakedMenu = cookies.filter { it.softBaked }
println(softBakedMenu)  // Output: [Cookie(name=Choco, softBaked=true)]
```

GroupBy: Transforming a List into a Map

The `groupBy` function is used to group elements in a list into a map based on a condition or property.

- Example:

```
val cookies = listOf(
    Cookie("Choco", true),
    Cookie("Oatmeal", false),
    Cookie("Sugar", true)
)

val groupedMenu = cookies.groupBy { it.softBaked }
val softBakedMenu = groupedMenu[true] ?: listOf()
val crunchyMenu = groupedMenu[false] ?: listOf()
```

```
println("Soft cookies:")
softBakedMenu.forEach { println("${it.name} - $${it.price}") }

println("Crunchy cookies:")
crunchyMenu.forEach { println("${it.name} - $${it.price}") }
```

Output:

```
Soft cookies:
Choco - $2.5
Sugar - $1.5
Crunchy cookies:
Oatmeal - $3.0
```

Here, the list of cookies is grouped by whether they are soft-baked or crunchy.

Fold: Reducing a Collection to a Single Value

The `fold` function reduces a collection to a single value by applying an operation on each element along with an accumulator.

- **Accumulator**: The intermediate result that is carried forward during the reduction.
- **Base Case**: The initial value of the accumulator.
- Example:

```
val cookies = listOf(
    Cookie("Choco", true, 2.5),
    Cookie("Oatmeal", false, 3.0),
    Cookie("Sugar", true, 1.5)
)

val totalPrice = cookies.fold(0.0) { total, cookie ->
    total + cookie.price
}
println("Total Price: $totalPrice")  // Output: 7.0
```

Here, `fold` calculates the sum of the prices of all cookies.

SortedBy: Sorting a Collection

The `sortedBy` function sorts elements in a collection based on a property or condition.

- Example:

```
val cookies = listOf(
    Cookie("Choco", true, 2.5),
    Cookie("Oatmeal", false, 3.0),
    Cookie("Sugar", true, 1.5)
)

val alphabeticalMenu = cookies.sortedBy { it.name }
alphabeticalMenu.forEach { println(it.name) }
```

Output:

```
Choco
Oatmeal
Sugar
```

The cookies are sorted alphabetically based on their names.

Key Concepts Recap

1. Kotlin collections are categorized into mutable and immutable types.
2. Powerful operations like `forEach`, `map`, `filter`, `groupBy`, `fold`, and `sortedBy` simplify data manipulation.
3. Collections can be transformed and reduced to derive meaningful insights and results.

Would you like to include explanations of more Kotlin-specific features or examples?