Working with Collections

Jim Wilson

Mobile Solutions Developer & Architect

@hedgehogjim | jwhh.com

Overview

The role of collections

Collections and type safety

Common collection methods

Collections and entry equality

New collection methods in Java 8

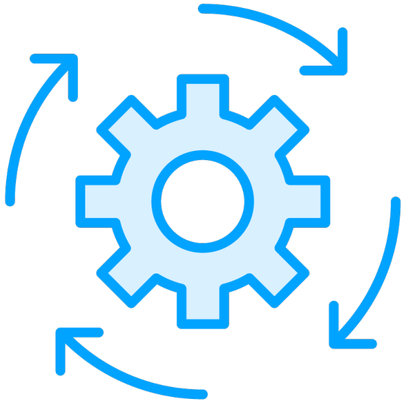
Converting between collections and arrays

Common collection interfaces and classes

Sorting behavior

Map collections

Managing Groups of Data



Apps manage groups of data

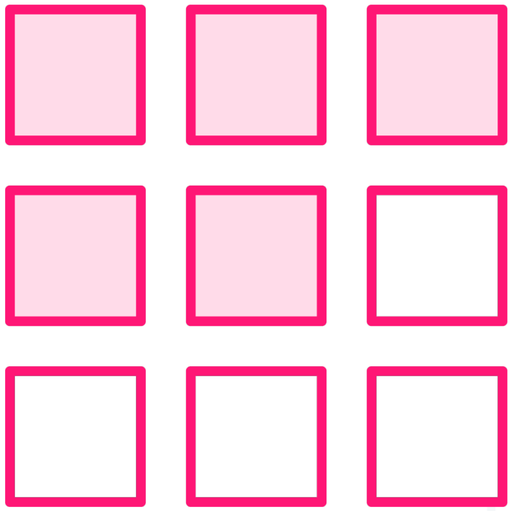
Most basic solution is arrays

Array limitations

Statically sized

Require explicit position management

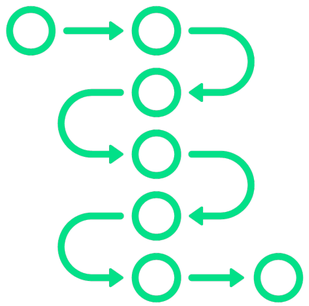
Just a bunch of values



Collections

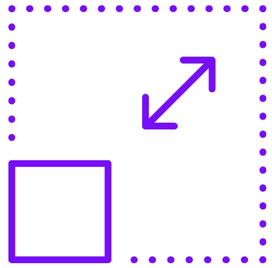
Provide more powerful options

Collections hold and organize values

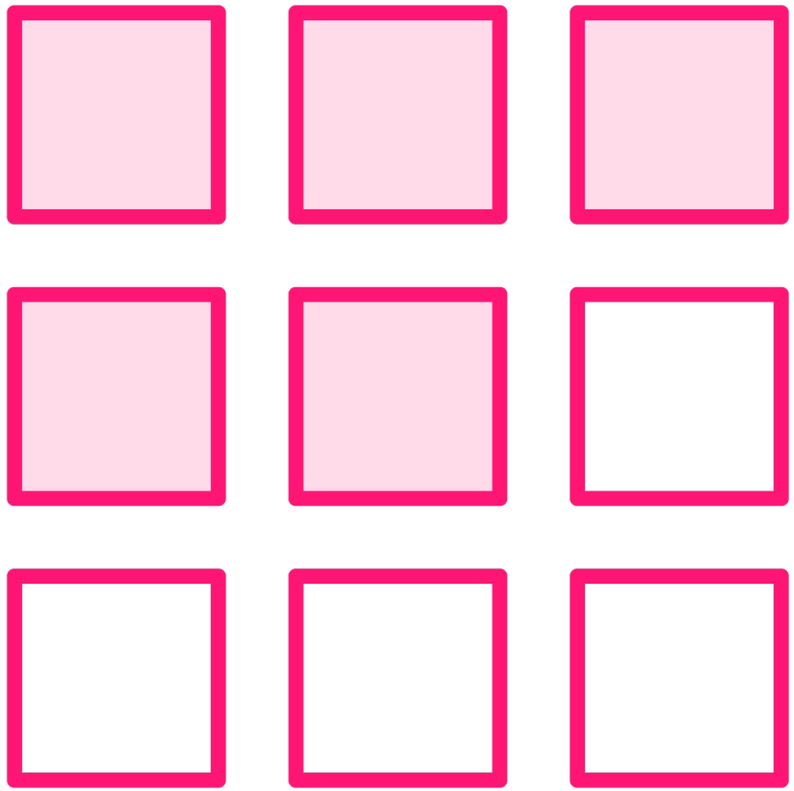


Iterable

Can provide type safety



Tend to dynamically size



A wide variety of collections are available

- May be a simple list of values
- Can provide optimization or sophistication
  - Ordering
  - Prevent duplicates
  - Manage data as name/value pairs

A Simple Collection of Objects

```
ArrayList list = new ArrayList();

list.add(“Foo”);
list.add(“Bar”);

System.out.println(“Elements: ” + list.size());

for(Object o:list)
  System.out.println(o.toString());

String s = (String)list.get(0);

SomeClassIMadeUp c = new SomeClassIMadeUp();
list.add(c);
```

Collections and Type Safety

By default collections hold Object types

- Must convert return values to desired type
- Doesn't restrict types of values added

Collections can be type restricted

- Uses the Java concept of generics
- Type specified during collection creation

Collection type restriction is pervasive

- Return values appropriately typed
- Adding values limited to appropriate type

A Strongly Typed Collection

```
ArrayList<String> list
```

A Strongly Typed Collection

```
ArrayList<String> list = new ArrayList<>();

list.add(“Foo”);
list.add(“Bar”);

System.out.println(“Elements: ” + list.size());

for(Object o:list)
  System.out.println(o.toString());
```

A Strongly Typed Collection

```
ArrayList<String> list = new ArrayList<>();

list.add(“Foo”);
list.add(“Bar”);

System.out.println(“Elements: ” + list.size());

for(String o:list)
  System.out.println(o.toString());
```

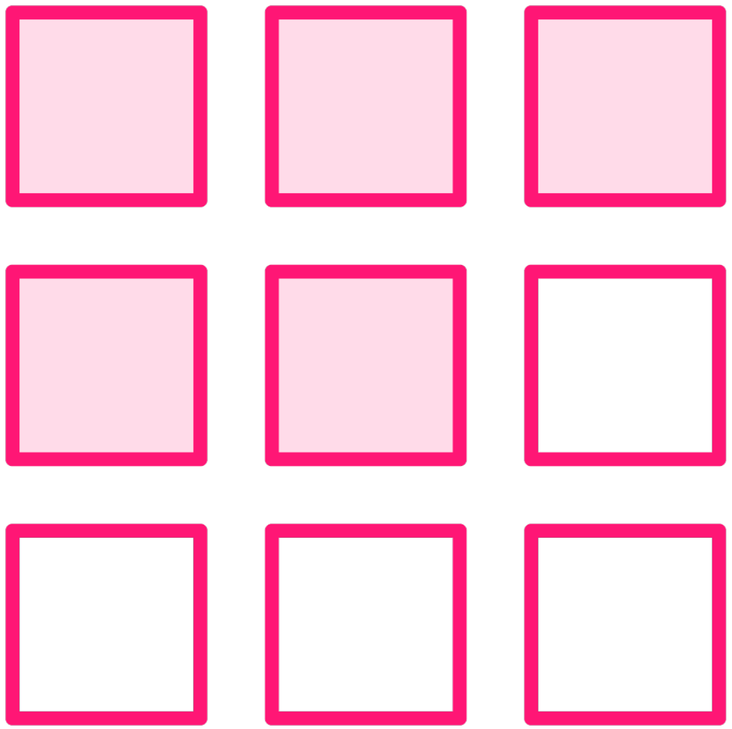
A Strongly Typed Collection

```
ArrayList<String> list = new ArrayList<>();

list.add(“Foo”);
list.add(“Bar”);

System.out.println(“Elements: ” + list.size());

for(String o:list)
  System.out.println(o);

String s = (String) list.get(0);
```

A Strongly Typed Collection

```
ArrayList<String> list = new ArrayList<>();

list.add(“Foo”);
list.add(“Bar”);

System.out.println(“Elements: ” + list.size());

for(String o:list)
  System.out.println(o);

String s = list.get(0);

SomeClassIMadeUp c = new SomeClassIMadeUp();
list.add(c);
```

Collection Interface



Each collection type has it's own features

- But there are many that are common

Collection interface

- Provides common collection methods
- Implemented by most collection types
- Map collections are notable exception
- Extends Iterable interface

Common Collection Methods

Method	Description

Adding Members from Another Collection

Main.java

```
ArrayList<String> list1 = new ArrayList<>();
list1.add(“Foo”);
list1.add(“Bar”);

LinkedList<String> list2 = new LinkedList<>();
list2.add(“Baz”);
list2.add(“Boo”);

list1.addAll(list2);

for(String s:list1)
  System.out.println(s);
```

Does not affect list2

```
Foo
Bar
Baz
Boo
```

Common Equality-based Methods

Method	Description

Tests all use the equals method

Removing a Member

```
public class MyClass {
  String label, value; // getters elided for clarity

  public MyClass(String label, String value) {
    // assign label & value to member fields
  }

  public boolean equals(Object o) {
    MyClass other = (MyClass) o;
    return value.equalsIgnoreCase(other.value);
  }
}
```

Removing a Member

Main.java

```
ArrayList<MyClass> list = new ArrayList<>();

MyClass v1 = new MyClass("v1", "abc");
MyClass v2 = new MyClass("v2", "abc");
MyClass v3 = new MyClass("v3", "abc");

list.add(v1);
list.add(v2);
list.add(v3);

list.remove(v3);

for(MyClass m:list)
  System.out.println(m.getLabel());
```

Uses equals method to find match

```
v2
v3
```

Java 8 Collection Methods

Java 8 introduced lambda expressions

- Simplify passing code as arguments

Collection methods that leverage lambdas

- forEach – Perform code for each member
- removeIf – Remove element if test is true

Using forEach Method

Main.java

```
ArrayList<MyClass> list = new ArrayList<>();

MyClass v1 = new MyClass("v1", "abc");
MyClass v2 = new MyClass("v2", “xyz");
MyClass v3 = new MyClass("v3", "abc");

list.add(v1);
list.add(v2);
list.add(v3);

list.forEach(                                    );
```

```
v1
v2
v3
```

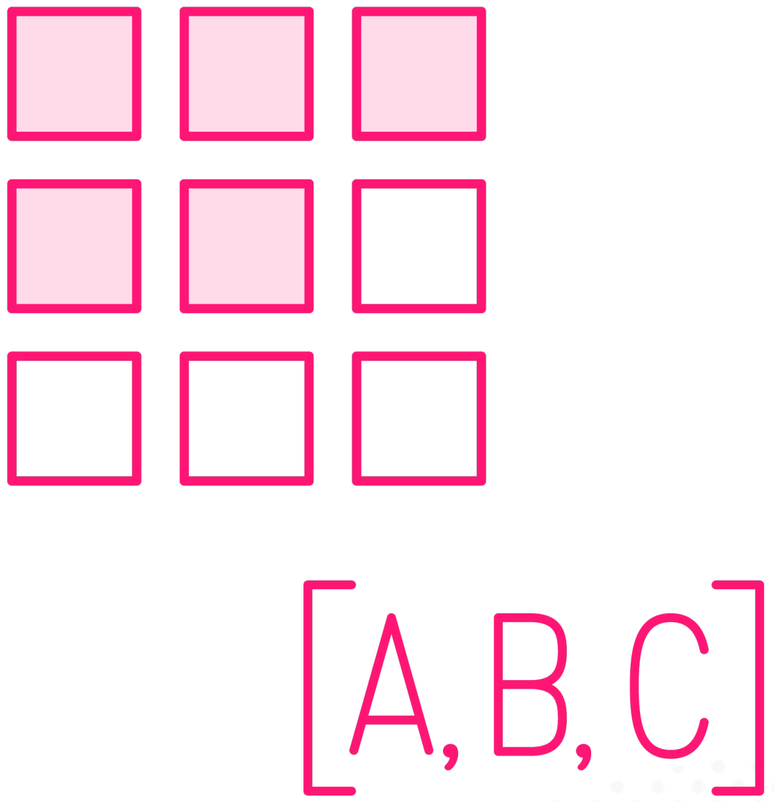
Using removeIf Method

Main.java

```
ArrayList<MyClass> list = new ArrayList<>();
MyClass v1 = new MyClass("v1", "abc");
MyClass v2 = new MyClass("v2", “xyz");
MyClass v3 = new MyClass("v3", "abc");
list.add(v1);
list.add(v2);
list.add(v3);
list.removeIf(                                    );
list.forEach(m -> System.out.println(m.getLabel()));
```

v2

Converting Between Collections and Arrays



Sometimes APIs require an array

- Often due to legacy or library code

Collection interface can return an array

- toArray() method
  - Returns Object array
- toArray(T[] array) method
  - Returns array of type T

Array content can be retrieved as collection

- Use Arrays class' asList method

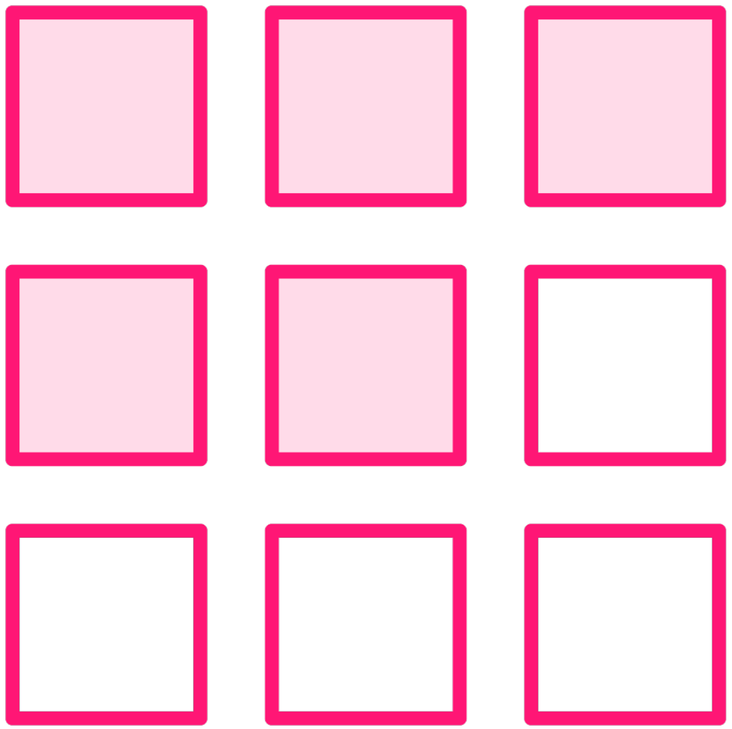
Retrieving an Array

```
ArrayList<MyClass> list = new ArrayList<>();
list.add(new MyClass("v1", "abc"));
list.add(new MyClass("v2", “xyz"));
list.add(new MyClass("v3", "abc"));

                    list.toArray();

               list.toArray(              );

MyClass[] a2 = new MyClass[3];
               list.toArray(  );

if(a2 == a3)
  System.out.println(“a2 & a3 reference the same array”);
```

Retrieving a Collection from an Array

```
MyClass[] myArray= {
  new MyClass("val1", "abc"),
  new MyClass("val2", “xyz"),
  new MyClass("val3", “abc")
};

                          Arrays.asList(       );
list.forEach(c -> System.out.println(c.getLabel()));
```

Collection Types



Java provides a wide variety of collections

- Each with specific behaviors

Collection interfaces

- Provide contract for collection behavior

Collection classes

- Provide collection implementation
- Implement 1 or more collection interfaces

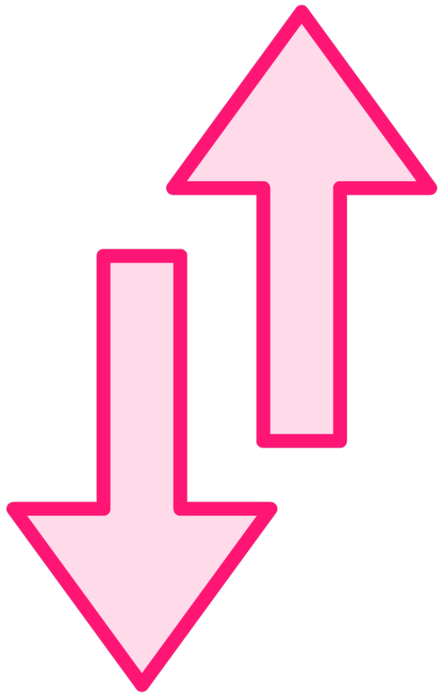
Common Collection Interfaces

Interface	Description

Common Collection Classes

Class	Description

Sorting



Some collections rely on sorting

- Two ways to specify sort behavior

Comparable interface

- Implemented by the type to be sorted
- Type specifies own sort behavior
- Should be consistent with equals

Comparator interface

- Implemented by type to perform sort
- Specifies sort behavior for another type

Implementing Comparable

```
public class MyClass                                    {
  String label, value; // Other members elided for clarity
  public String toString() { return label + " | " + value;}

  public boolean equals(Object o) {
    MyClass other = (MyClass) o;
    return value.equalsIgnoreCase(other.value);
  }

  public int compareTo(MyClass other) {
    return value.compareToIgnoreCase(other.value);
  }
}
```

```
- : this < other
0 : this = other
+ : this > other
```

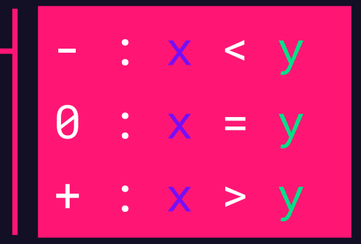
Using TreeSet with Comparable

Main.java

```
TreeSet<MyClass> tree = new TreeSet<>();

tree.add(new MyClass("2222", "ghi"));
tree.add(new MyClass("3333", "abc"));
tree.add(new MyClass("1111", "def"));

tree.forEach(m -> System.out.println(m));
```

```
3333 | abc
1111 | def
2222 | ghi
```

Implementing Comparator




```
public class MyComparator                                    {
  public int compare(MyClass x, MyClass y) {
    return x.getLabel().compareToIgnoreCase(y.getLabel());
  }
}
```

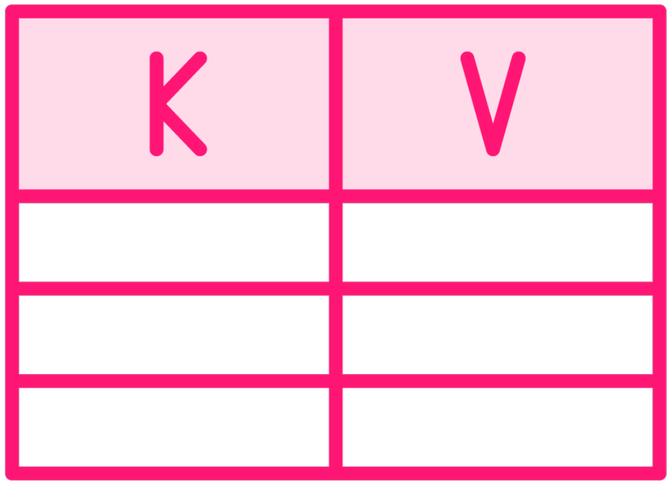
Using TreeSet with Comparator

Main.java

```
TreeSet<MyClass> tree =
    new TreeSet<>(                    );

tree.add(new MyClass("2222", "ghi"));
tree.add(new MyClass("3333", "abc"));
tree.add(new MyClass("1111", "def"));

tree.forEach(m -> System.out.println(m));
```

```
1111 | def
2222 | ghi
3333 | abc
```

Map Collections



Maps store key/value pairs

- Key used to identify/locate values
- Keys are unique
- Values can be duplicated
- Values can be null

Common Map Types

Interface	Description

Class	Description

Common Map Methods

Method	Description

Using Map

```
Map<                  > map
map.put(“2222", “ghi");
map.put(“3333", “abc");
map.put(“1111", “def");

String s1 = map.get(“3333”);

String s2 = map.get(“9999”);

String s3 = map.getOrDefault(“9999”, “xyz”);
```

Using Map

Main.java

```
Map<String, String> map = new HashMap<>();
map.put(“2222", “ghi");
map.put(“3333", “abc");
map.put(“1111", “def");
map.forEach(
                                        );

map.replaceAll(                       );

map.forEach(
    (k, v) -> System.out.println(k + “ | ” + v));
```

```
2222 | ghi
3333 | abc
1111 | def

2222 | GHI
3333 | ABC
1111 | DEF
```

Common SortedMap Methods

Method	Description

Using SortedMap

Main.java

```
SortedMap<String, String> map

map.put(“2222", “ghi");
map.put(“3333", “abc");
map.put(“1111", “def");
map.put(“6666", “xyz");
map.put(“4444", “mno");
map.put(“5555", “pqr");

map.forEach(
    (k, v) -> System.out.println(k + “ | ” + v));
```

```
1111 | def
2222 | ghi
3333 | abc
4444 | mno
5555 | pqr
6666 | xyz
```

Using SortedMap

Main.java

```
SortedMap<String, String> map = new TreeMap<>();
// Add same 6 key/value pairs as last slide

SortedMap<String, String> hMap = map.headMap("3333");
hMap.forEach( (k, v) ->
  System.out.println(k + " | " + v));

SortedMap<String, String> tMap = map.tailMap("3333");
tMap.forEach( (k, v) ->
  System.out.println(k + " | " + v));
```

```
1111 | def
2222 | ghi

3333 | abc
4444 | mno
5555 | pqr
6666 | xyz
```

Summary

Collections hold and organize values

- Iterable
- Tend to dynamically size
- Can provide optimization or sophistication

Collections can be type restricted

- Uses Java generics to specify type
- Return values appropriately typed
- Typing enforced on added values

Summary

Can convert between collections & arrays

- Collections provide toArray method
- Arrays class' provides toList method

Some collections provide sorting

- Support Comparable interface
  - Type defines own sort
- Support Comparator interface
  - Specifies sort for another type

Summary

Map collections

- Stores key/value pairs
- Keys are unique
- Some maps sort keys