

# Collections & Generics



## CONTENTS

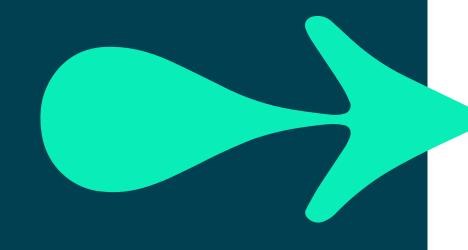


- Compare functionality offered by arrays & collections
- Understand generic concepts, use generic types & syntax

#### Contents

- Recap arrays, introduce collection classes
- Generic concepts
  - Collections Framework Generic classes

#### Two hands on Labs



# Arrays vs Collection classes

#### Limitations of Arrays

- Fixed size, (until resized), can't append, insert or delete
- No built-in method to reject duplicates
- Must continually watch out for ArrayIndexOutOfBoundsException
- But are type-safe!, a Car array can only contain Car references

#### Java offers a collection classes

- Queue, Stack, List, Set, Map, Dictionary, SortedList ...
  - Know their Capacity & Count
  - Support append, insertions / deletions / searching
- Generic version of these are type-safe

### Collection classes in Java 5

Developers had to use the ArrayList which could only hold a collection of Object type

```
ArrayList myList = new ArrayList();
myList.add(123);
myList.add("Bob");
myList.add(new Car());
```

#### Java's get() method returns an Object at an index

#### Generic collection classes

Can hold a collection of a specific type and always returns the expected type

import java.util.ArrayList

## Iterating through an generic ArrayList

```
ArrayList<String> friends = new ArrayList<>();
friends.add("Tom");
friends.add("Sue");
                                         add( ) expects a String
friends.add("Sanjeev");
for (String name : friends)
                                                      enumerable
     System.out.printf(name);
for(int i = 0; i < friends.size(); i++)</pre>
                                               Note the size()
     System.out.printf( friends.get(i) );
friends.set(1, "Susan");
                                       // Susan replaces Sue
```

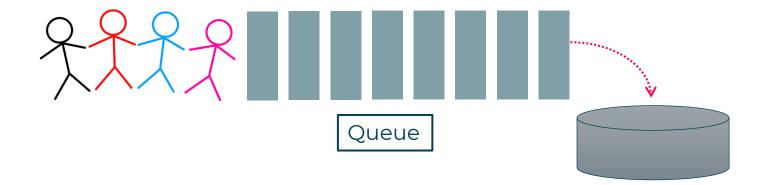
# Collections framework Generic Types

In package java.util

```
public class ArrayList<T> { ... }
public class LinkedList<T> { ... }
public class ArrayDeque<T> { ... } // Stacks and Queues

public class HashSet<T> { ... }
public class TreeSet<T> { ... } // Sorted

public class HashMap <K,V> { ... } Multiple type
public class TreeMap <K,V> { ... }
```



# Java: Queue – ArrayDeque<T> FIFO

Provide both FIFO (Queue) & LIFO (Stack) behaviour

```
ArrayDeque<String> queue = new ArrayDeque<String>();
queue.add("Dave");
queue.add("Mike");
queue.add("Linda");
queue.add("Joe");

while(!queue.isEmpty()) {
    String item = queue.pop();
    System.out.println(item);
}
```

```
ArrayDeque<Car> queue= new ArrayDeque<>();

queue.add(new Car("Ford"));
queue.add(new Car("Honda"));

for(Car car : queue) {
    System.out.println(car.getModel());
    queue.remove(car);
}

Ford
Honda
```



## Hands On Labs

#### **Stack and Queue behaviours**

- Part 1 Using Lists
- Part 2 Using Queues
- Please only do part 1 & 2

# Usage of HashMap of key/value pairs

keys are unique

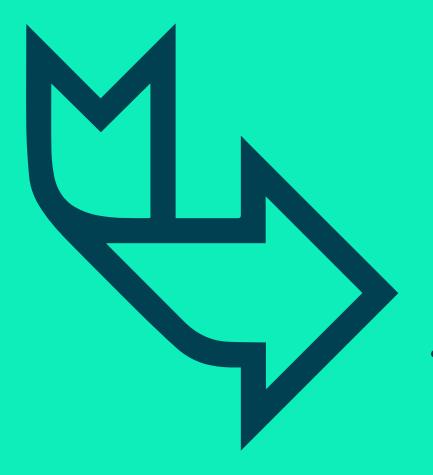
```
HashMap<String, Car> hm = new HashMap<>();
hm.put("Sam", new Car("Ford"));
hm.put("Joe", new Car("BMW"));
Car car = hm.get("Sam");
System.out.println(car.getModel());
Ford
```

```
if(hm.containsKey("Bob"))
hm.put("Bob", new Car("Ferrari"));
```

Searching and replacing an item

# Usage of HashMap of key/value pairs

```
HashMap<String, Car> hm = new HashMap<>();
hm.put("Sam", new Car("Ford"));
                                                     Joe drives a BMW
hm.put("Joe", new Car("BMW"));
                                                     Sam drives a Ford
for (String key : hm.keySet()) {
     System.out.printf("%s drives a %s\n", key, hm.get(key).getModel());
for (Car car : hm.values()) {
                                                     BMW
     System.out.println(car.getModel());
                                                     Ford
```



# Hands On Labs (Part 2)

- Zoo Animals
  - Using HashMap<K, V>

## Review



#### Java 5.0 (2004) introduced generic types

- Improve performance and type safety, less casting
- ArrayList<E> & Hashmap<TKey, TValue> widely used
- Stack and Queue behaviour exhibited by ArrayDeque<E>
- java.util package
- Map's are key value pairs (like a Dictionary)
  - Implemented by HashMap and TreeMap(sorted)
- Collections utility class

### Java Collections class

```
ArrayList<String> flavours = new ArrayList<>();
..add add add
flavours.sort();
Collections.sort(flavours);
Collections.reverse(flavours);
```