Basic Java : Collections & Maps

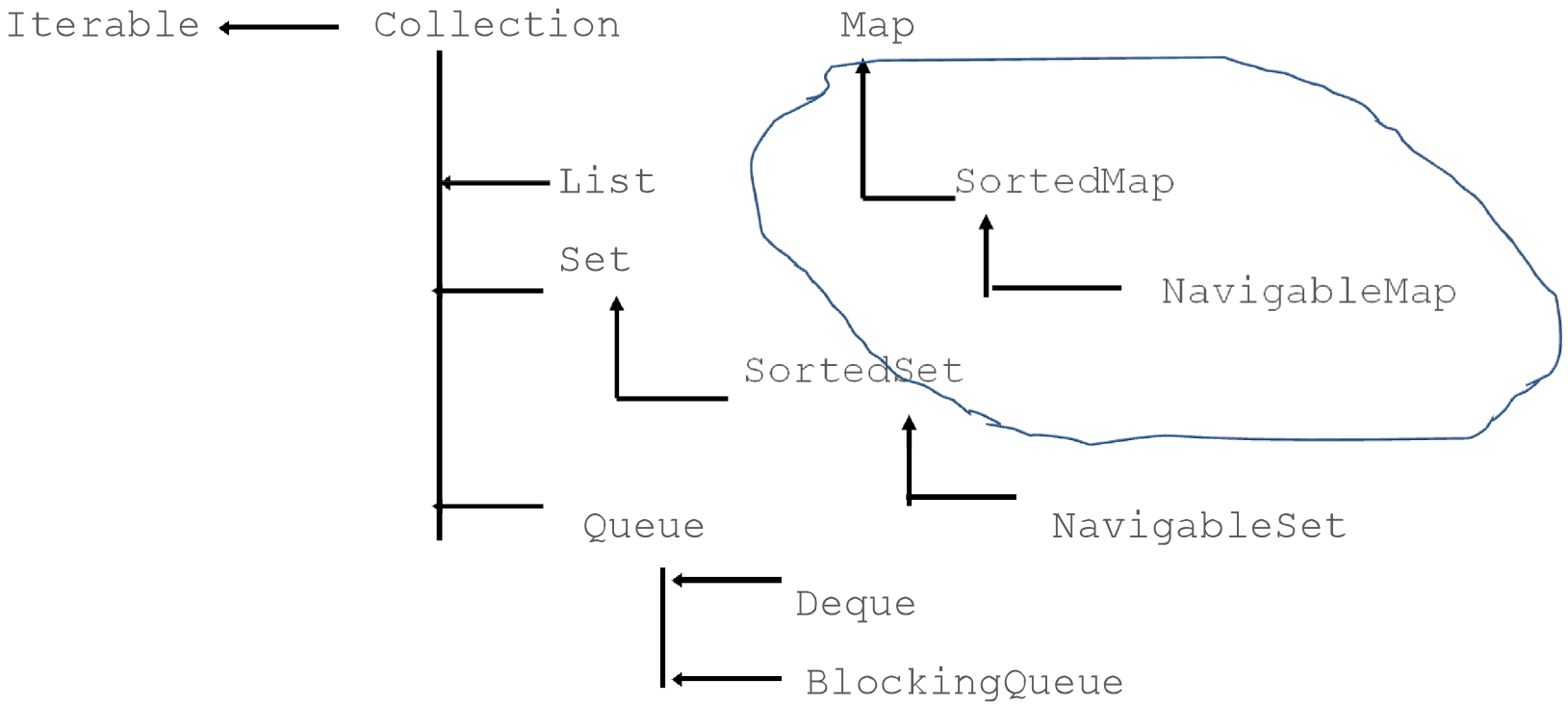
Persistent University

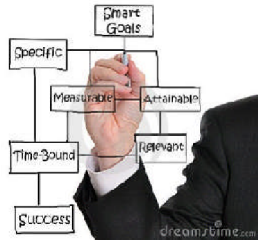
PERSISTENT

Objectives :

- Introduction to Core Interfaces and Classes
- Introduction to Generics, use of generics in Collection framework
- List, Set and Map implementations

The collections framework.....core interfaces





The collections framework……..concrete classes

Interface	Class	Description
List	ArrayList	A unsynchronized dynamic array
	LinkedList	A linked list. Supports methods for use as a stack, queue and deque
	Vector	A synchronized dynamic array
Set	HashSet	Unordered container backed by a hash Map. Allows null element.
	LinkedHashSet	Hash table and linked list implementation of the Set interface. Predictable iteration order. Allows null element
	TreeSet	The elements are ordered using their natural ordering.

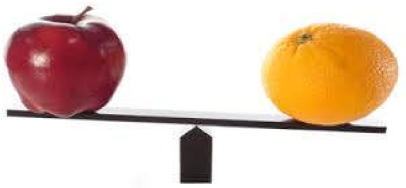
The collections framework……..concrete classes

Interface	Class	Description
Map	HashMap	A unsynchronized key-value container which allows null key and values.
	Hashtable	A synchronized key-value container wherein any non-null object can be used as a key or value.
	LinkedHash Map	Maintains a doubly-linked list running through all of its entries.
	TreeMap	The map is sorted according to the natural ordering of its keys

Difference between List and Set Implementations

- List and Set both implementations hold multiple elements.

- However List can hold duplicate elements and Set holds only Unique elements.



Generics

- What is generics ?

```
public class Stack {
          private int size;
          private Object values[];
          --------------------

          public Stack(int size) {
                    --------------------
                    values = new Object[size];
          }

          public void push(Object value) {
                    --------------------
                    --------------------
          }

          public Object pop() {
                    --------------------
                    --------------------
          }
}
```

Design issues/mistakes

```
Stack stack = new Stack(10);
String str;

stack.push("I need a stack of strings");
stack.push("This is string");
stack.push("Same here");
stack.push("Me too");

/* downcast works */
str = (String) stack.pop();

/* whoops */
stack.push(new Integer(1000));

/* cast fails. throws ClassCastException */
str = (String) stack.pop();
```

Need for generics

- Type safety
- Improved robustness
- Improved readability

List implementations

- List holds the objects.
- Can hold duplicate data.
- Also holds multiple null values.
- Implementations are
  - ArrayList
  - LinkedList
  - Vector

ArrayList.....quick preview

```
List<Integer> intList = new ArrayList<Integer>();

// ArrayList<Integer> intList = new ArrayList<Integer>(20);

intList.add(1);
intList.add(10);
intList.add(100);
intList.add(10000);

intList.add(3, 1000);

if(intList.contains(100))
        System.out.println("Found");

System.out.println(intList.indexOf(1000));
System.out.println(intList.size());
System.out.println(intList.remove(4));
System.out.println(intList.get(0));
```

How to store custom objects?

- Collections can hold user defined objects as well.

```
public class Employee {
        private int empId;
        private String name;
        private String designation;
        public Employee(int empId, String name, String
designation) {
                this.empId = empId;
                this.name = name;
                this.designation = designation;
        }
        @Override
        public String toString() {
                return "Employee [empId=" + empId + ",
        name=" + name + ", designation="
                                + designation + "]";
        }
}
```

How to store custom objects?

```
import java.util.ArrayList;
import java.util.List;
public class EmployeeList {
    public static void main(String[] args) {
        List<Employee> employees = new ArrayList<Employee>();
        employees.add(new Employee(1001, "John", "Developer"));
        employees.add(new Employee(1886, "Riya", "Developer"));
        employees.add(new Employee(6723, "Smith", "Project
Manager"));
        employees.add(new Employee(8954, "Pooja", "Tester"));
        for (Employee employee : employees) {
                System.out.println(employee);
        }
    }
}
```

Set Implementations

- Set holds multiple values
- However it holds only unique values.
- Implementations are
  - HashSet
  - LinkedHashSet
  - TreeSet

HashSet.....quick preview

```
Set<String> hashSet = new HashSet<String>();
// HashSet<String> hashSet = new HashSet<String>(30);
// HashSet<String> hashSet = new HashSet<String>(20, 0.8f);

hashSet.add("This");
hashSet.add("is");
hashSet.add("a");
hashSet.add("hash");
hashSet.add("set");

System.out.println(hashSet.size());

hashSet.clear();
```

Map Implementations

- Map stores data in key-value pairs.
- Key is an unique identifier for every value.
- Keys are unique however values can be repeated.
- Both key and value are objects.
- Implementations are
  - HashMap
  - LinkedHashMap
  - TreeMap

Hashtable.....quick preview

```
Hashtable<Integer, String> hashtable =
        new Hashtable<Integer, String>();
// Hashtable<Integer, String> hashtable =
        new Hashtable<Integer, String>(20, 0.8f);

hashtable.put(11, "Persistent");
hashtable.put(22, "Infocepts");
hashtable.put(33, "Ebix");
hashtable.put(44, "Infospectrum");

System.out.println(hashtable.get(11));

System.out.println(hashtable);

System.out.println(hashtable.remove(44));
System.out.println(hashtable.containsKey(44));

hashtable.clear();
```

The collections framework.....iterators

Iterator ListIterator

An forward only iterator over a collection

A bi-directional iterator over a collection

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

sObj.add("Great");
sObj.add("Cool");
sObj.add("Fine");
sObj.add("Nice");

for(String s : sObj) System.out.println(s);

Iterator<String> itr = sObj.iterator();

while(itr.hasNext()) System.out.println(itr.next());
```

An *ArrayList* is *Iterable* with the “foreach“ loop

An *ArrayList* is accessible with methods from the *Iterator* interface

Summary : Session

With this we have come to an end of our session, where we discussed about ....

- Collection framework
- List Set & Map Interfaces
- Use of generics
- Iterator interface
- How to store custom objects

At the end of this session, we see that you are now able to answer following questions:

- What is collection framework?
- What are core interfaces & implementations?
- How to use collection framework in applications?

Reference Material : Websites & Blogs

- https://docs.oracle.com/javase/tutorial/collections/intro/
- http://www.tutorialspoint.com/java/java_collections.htm
- http://www.javatpoint.com/collections-in-java

Reference Material : Books

- ***Head First Java***
  - *By: Kathy Sierra, Bert Bates*
  - *Publisher: O'Reilly Media, Inc.*

- ***Java Complete Reference***
  - *By Herbert Schildt*

Key Contacts :

Java Interactive :

- Asif Immanad

asif_immanad@persistent.co.in

- Nisha Waikar

nisha_waikar@persistent.co.in

- Varsha Mulik

varsha_mulik@persistent.co.in

Persistent University :

- Poorva Kulkarni

poorva_kulkarni@persistent.co.in

Vice President :

- Shubhangi Kelkar

shubhangi_kelkar@persistent.co.in

Thank You !!!

Persistent University