1 Collections

The following are some of the most commonly Collection classes used in Java:

- List (a sequence of elements)
  - ArrayList
  - LinkedList
- Set (a sequence of unique elements)
  - HashSet
  - TreeSet
  - LinkedHashSet
- Map (a sequence of pairs, each of which one maps a key to a value)
  - HashMap
  - TreeMap
  - LinkedHashMap

The **ArrayList** class has already been described in a set of previous lecture notes.

2 Set

Object of **Set** subclasses can only contain unique objects, i.e. an instance of an object cannot be contained more than once in a set.

- To determine whether an object is the same with another object (so that the set will not insert a second instance of the same object), the **equals()** method has to be overridden.
2.1 HashSet

A set with very fast lookups based on a hash value. Objects of classes stored in a HashSet must implement (override) the hashCode() method.

Example:

```java
import java.util.*;

class Student {
    private String name;
    private int id;

    Student(String name, int id) {
        this.name = name;
        this.id = id;
    }

    public boolean equals(Object o) {
        return (o instanceof Student) &&
               name.equals(((Student) o).name) &&
               (id == ((Student) o).id);
    }

    public int hashCode() {
        int result = 17;
        result = 37*result + name.hashCode();
        result = 37*result + id;

        return result;
    }

    public String toString() {
        return "Name: " + name + ", id: "+ id;
    }
}

public class HashSetExample {
    public static void main(String[] args) {
        Student st1 = new Student("John Williams", 199110);
        Student st2 = new Student("George Smith", 199111);
        Student st3 = new Student("Helen Jones", 199112);
        Student st4 = new Student("John Williams", 199110);

        HashSet<Student> h = new HashSet<Student>();
        h.add(st1);
        h.add(st2);
        h.add(st3);
```
h.add(st4);

for (Student s : h)
    System.out.println(s);
}
}

When the program is run, it displays:

Name: George Smith, id: 199111
Name: Helen Jones, id: 199112
Name: John Williams, id: 199110

Note that the order of objects extracted out of a HashSet, is not the same with the order that they were inserted into it.

2.2 A Simple Algorithm to calculate the hash code of an object

The following is an algorithm to implement hashCode(), based on Joshua Bloch:

1. Store some non-zero value to an int variable, for example a variable called result.
2. For each significant field in the object (each field taken into account in the implementation of equals), calculate an integer hashcode c.
3. Calculate the hashcode based on the above hashcodes:
   \[ \text{result} = 37 \times \text{result} + c \]
4. Return result.
5. Make sure that in the hashCode() implementation, equal objects have equal hash codes.

2.3 TreeSet

The order of elements is fixed and it is the ascending order defined by the natural order of the elements.

The natural order of elements is determined by:

- the compareTo method if elements implement the Comparable interface.
- the comparator passed to the constructor of TreeSet.
Example:

```java
import java.util.*;

class Book implements Comparable<Book> {
    String name;
    private int isbn;

    Book(String name, int isbn) {
        this.name = name;
        this.isbn = isbn;
    }

    public boolean equals(Object o) {
        return (o instanceof Book) &&
                (isbn == ((Book) o).isbn);
    }

    public int compareTo(Book other) {
        if (isbn > other.isbn)
            return 1;
        else if (isbn < other.isbn)
            return -1;
        else
            return 0;
    }

    public String toString() {
        return "Name: " + name + ", isbn: " + isbn;
    }
}

public class TreeSetExample {
    public static void main(String[] args) {
        Book st1 = new Book("Life of a tiger", 5556);
        Book st2 = new Book("How to program", 5557);
        Book st3 = new Book("The Book", 5555);
        Book st4 = new Book("Swimming dolphins", 5555);

        TreeSet<Book> h = new TreeSet<Book>();
        h.add(st1);
        h.add(st2);
        h.add(st3);
        h.add(st4);

        for (Book s : h)
            System.out.println(s);
    }
}
```
3 Maps

A map is an association between keys and values (both of which should be objects).

- Every key in a map has a unique value.
- A value can be associated with many keys.

Note that the type of the key can be different than the type of the value.

3.1 HashMap

A fast lookup map based on hashcodes. The ordering of key-value pairs is not guaranteed.

The `equals()` and the `hashCode()` methods must be overridden for the type used as the key of the map.

Example:

```java
import java.util.*;

public class HashMapExample {
    public static void main(String[] args) {
        HashMap<String, Integer> phoneExtensions = new HashMap<String, Integer>();
        phoneExtensions.put("David", new Integer(4413));
        phoneExtensions.put("Andrew", new Integer(5678));
        phoneExtensions.put("John Williams", new Integer(4561));
        phoneExtensions.put("Peter", new Integer(7782));

        // get the extension for Andrew
        Integer ext1 = phoneExtensions.get("Andrew");
        System.out.println("Andrew extension: " + ext1);

        // get all the keys in the map
        Set<String> keys = phoneExtensions.keySet();

        // print all extensions
        System.out.println("\nAll people/extensions:");
        for (String k : keys) {
            Integer ext = phoneExtensions.get(k);
        }
    }
}
```

When the program is run, it displays:

Andrew extension: 5678

All people/extensions:
Name: Peter, extension: 7782
Name: John Williams, extension: 4561
Name: Andrew, extension: 5678
Name: David, extension: 4413

3.2 TreeMap

TreeMap iterates over the keys in ascending order. Therefore, the ordering of keys is guaranteed.

Example:

```java
import java.util.*;

public class TreeMapExample {
    public static void main(String[] args) {
        TreeMap<String, Integer> phoneExtensions = new TreeMap<String, Integer>();
        phoneExtensions.put("David", new Integer(4413));
        phoneExtensions.put("Andrew", new Integer(5678));
        phoneExtensions.put("John Williams", new Integer(4561));
        phoneExtensions.put("Peter", new Integer(7782));

        // get all the pairs from the map
        Set<Map.Entry<String, Integer>> pairs = phoneExtensions.entrySet();

        // print all extensions
        System.out.println("\nAll people/extensions: ");
        for (Map.Entry<String, Integer> e : pairs) {
            String person = e.getKey();
            Integer ext = e.getValue();
            System.out.println("Name: "+ person + ", extension: "+ ext);
        }
    }
}
```

As expected, the above example produces the list of known names and extensions, in alphabetical order of names:
4 Other Collections

- **LinkedList**: As the name implies, this is an implementation of a list as a linked list. Methods of the class allow it to be used as a stack, queue, or double ended queue.

  A LinkedList provides a fast way to insert an element into a list, but the access of elements is slower than an ArrayList.

- **LinkedHashSet**: An implementation of a hash set using a linked list. The ordering of elements is defined, and performance is better than TreeSet. The order of elements is defined by the order that the elements were inserted.

- **LinkedHashMap**: An implementation of a HashMap as a linked list with predictable iteration order. Performance is better than TreeMap. The order of elements is defined by the order that the elements were inserted.