

CIS 27P

Class 9

java.lang.*

- Wrapper types
- Runtime

```
Runtime rt = Runtime.getRuntime();  
Process p = rt.exec("ls /usr/");  
long l = rt.freeMemory();  
l = rt.totalMemory();  
rt.gc();
```

- System

```
long l = System.currentTimeMillis();
```

- Object

```
super class of all classes  
defines protected clone();
```

java.lang.* (cont)

- Math
 - type abs(type arg)
 - double pow(double x, double y)
 - min, max, round, geometry, etc.

Collections & Maps

- part of java.util.*
- Goals
 - high performance
 - interoperable
 - extensible / adaptable
 - integrated with standard arrays

Collection Interface

- Foundation upon which the collections framework is build
- Useful Methods:
 - boolean add(Object obj)
 - boolean remove(Object obj)
 - boolean contains(Object obj)
 - int size()
 - Iterator iterator()

List Interface

- Extends Collection
- A collection that stores a sequence of elements
- Useful Methods:
 - `void add(int index, Object obj);`
 - `Object get(int index);`
 - `ListIterator listIterator();`

Set Interface

- Extends Collection
- Does not allow duplicate elements
- add() returns false if elements already exists in list
- Has no new of its own

SortedSet Interface

- Extends Set
- A collection of non-duplicate elements sorted in ascending order
- Useful Methods:
 - Object first()
 - Object last()
 - Comparator comparator()

Concrete Collections

- **ArrayList:**
 - extends `AbstractList`
 - supports dynamic arrays that can grow as needed
- **LinkedList**
 - Extends `AbstractSequentialList`
 - adds useful methods:
 - `void addFirst(Object obj)`
 - `void addLast(Object obj)`
 - `Object getFirst()`
 - `Object getLast()`
 - `Object removeFirst()`
 - `Object removeLast()`

```

import java.util.*;

public class ArrayListSample
{
    public static void main( String [] args )
    {
        ArrayList al = new ArrayList();
        al.add("String");
        al.add(1, "String");
        al.add( new Long(5) );
        String s = (String) al.get(1);
        Long l = (Long) al.get(2);

        //al.add(17, "Not allowed, index too big");
        //al.add( 5 ); illegal, primitive type

        System.out.println( "s = "+s );
        System.out.println( "l = "+l.intValue() );
    }
}

```

OUTPUT:

s = String

l = 5

```
import java.util.*;

public class LinkedListSample
{
    public static void main( String [] args )
    {
        LinkedList ll = new LinkedList();
        ll.add("middle");
        ll.add( 1, new Long(7));
        ll.addFirst("First Element");
        ll.addLast("Last element");

        String first = (String)ll.getFirst();
        String last = (String)ll.getLast();

        System.out.println( "First:"+ first );
        System.out.println( "Last :"+ last );
    }
}
```

OUTPUT:

First: First Element

Last: Last Element

Concrete Collections

- HashSet Class
 - Extends AbstractSet
 - Stores elements in hash
 - No additional methods
 - Useful for sets containing a large number of elements
 - Does not guarantee the order of its elements
- TreeSet Class
 - Extends AbstractSet
 - Stores elements sorted in Tree

```
import java.util.*;

public class HashSetSample
{
    public static void main( String [] args )
    {
        HashSet hs = new HashSet();
        hs.add("B");
        hs.add("A");
        hs.add("A");
        hs.add( new Long(7) );

        System.out.println( "hs:" + hs );
    }
}
```

OUTPUT:

hs: [A, 7, B]

```
import java.util.*;

public class TreeSetSample
{
    public static void main( String [] args )
    {
        TreeSet ts = new TreeSet();
        ts.add("C");
        ts.add("B");
        ts.add("A");
        //ts.add( new Long(7) );

        System.out.println( "ts:" + ts );
    }
}
```

OUTPUT:

ts: [A, B, C]

Iterators

- Iterator
 - obtained from `instance.iterator()`
 - boolean `hasNext()`
 - Object `next()`
 - void `remove()`
- ListIterator
 - obtained from `intance.listIterator()`
 - Object `previous()`
 - int `nextIndex()`
 - int `previousIndex()`
 - void `add(Object obj)`
 - void `set(Object obj)`

```
import java.util.*;

public class IteratorSample
{
    public static void main( String [] args )
    {
        TreeSet ts = new TreeSet();
        ts.add(new Long(7));
        ts.add(new Long(5));
        ts.add(new Long(3));
        Iterator i = ts.iterator();
        while ( i.hasNext() )
        {
            System.out.println( "Item: "+i.next() );
        }
    }
}
```

OUTPUT:

Item: 3

Item: 5

Item: 7

```

import java.util.*;
public class ListIteratorSample
{
    public static void main( String [] args ){
        LinkedList ll = new LinkedList();
        ll.add(new Long(7));
        ll.add(new Long(5));
        ll.add(new Long(3));
        ListIterator li = ll.listIterator();
        while ( li.hasNext() )
        {
            System.out.print( li.next() );
            if ( li.hasNext() ) System.out.print(", ");
        }
        System.out.println("");
        while ( li.hasPrevious() )
        {
            System.out.print( li.previous() );
            if ( li.hasPrevious() ) System.out.print(", ");
        }
    }
}

```

OUTPUT

7 5 3

3 5 7

Map Interface

- Map interface: maps unique keys to values
- Useful Methods:
 - boolean containsKey(Object k)
 - boolean containsValue(Object v)
 - Set entrySet()
 - Object get(Object k)
 - Object put(Object k, Object v)
 - Collection values()

Map.Entry Interface

- Represents a single Map entry
- Useful Methods:
 - Object getKey()
 - Object getValue()
 - Object setValue(Object val)

Sorted Map Interface

- Extends Map
- Maintains a sorted list of keys
- Useful Methods:
 - Comparator comparator()
 - Object firstKey()
 - Object lastKey()

HashMap Class

- Extends AbstractMap
- Uses a Hash Table to implement Map, making get() and put() efficient
- No methods of its own
- Does not guarantee the order of its elements

```
import java.util.*;
public class HashMapSample
{
    public static void main( String [] args )
    {
        HashMap hm = new HashMap();
        hm.put("Key", new Long(2));
        hm.put(new Long(5), "Value" );
        Set s = hm.entrySet();
        Iterator i = s.iterator();
        while ( i.hasNext() )
        {
            Map.Entry me = (Map.Entry)i.next();
            System.out.print("KEY: "+me.getKey());
            System.out.println(", VAL: "+me.getValue());
        }
    }
}
```

OUTPUT:

KEY: Key, VAL: 2

KEY: 5, VAL: Value

TreeMap Class

- Extends AbstractMap and implements SortedMap
- Stores key/value pairs in sorted key order

```

import java.util.*;

public class TreeMapSample
{
    public static void main( String [] args )
    {
        TreeMap tm = new TreeMap();
        tm.put("CKEY", "Just A String");
        tm.put("BKEY", new Long(2));
        tm.put("AKEY", new Long(1));
        Set s = tm.entrySet();
        Iterator i = s.iterator();
        while ( i.hasNext() )
        {
            Map.Entry m = (Map.Entry)i.next();
            System.out.print("KEY: "+m.getKey());
            System.out.println(",VAL:"+m.getValue());
        }
    }
}

```

OUTPUT

KEY: AKEY, VALUE: 1

KEY: BKEY, VALUE: 2

KEY: CKEY, VALUE: Just a String

Comparators

- Passed into constructors some Collections classes
- Override compare method
 - `int compare(Object a, Object b)`
 - return negative # if first is less
 - return 0 if equal
 - return positive # if first is more

```
import java.util.*;
```

```
class MyComparator implements Comparator
```

```
{
```

```
    public int compare( Object a, Object b )
```

```
    {
```

```
        long la = ((Long)a).longValue();
```

```
        long lb = ((Long)b).longValue();
```

```
        long ra = la % 2;
```

```
        long rb = lb % 2;
```

```
        if ( ( ra == 0 && rb == 1 ) || la < lb )
```

```
            return -1;
```

```
        else if ( ( rb == 0 && ra == 1 ) || la > lb )
```

```
            return 1;
```

```
        return 1;
```

```
    }
```

```
}
```

```
import java.util.*;

public class UseComparator
{
    public static void main( String [] args )
    {
        TreeSet ts = new TreeSet( new MyComparator() );
        ts.add( new Long(1) );
        ts.add( new Long(6) );
        ts.add( new Long(5) );
        ts.add( new Long(10) );

        Iterator i = ts.iterator();

        while ( i.hasNext() )
        {
            System.out.print(i.next());
            if (i.hasNext()) System.out.print(", ");
        }
    }
}
```

OUTPUT:

6, 10, 1, 5

Miscellaneous

- Static algorithm methods
- Arrays
- Legacy Classes
 - Enumeration
 - Vector
 - Hashtable
 - Stack
 - Dictionary
 - Hashtable
 - Properties