

# Java Review

---



**Nelson Padua-Perez**  
**Chau-Wen Tseng**

**Department of Computer Science**  
**University of Maryland, College Park**

## Java Topics

- **Running Java programs** ←
- **Stream I/O**
- **Collections framework**
- **Java 1.5**

## Running Java Programs

- **Java Virtual Machine (JVM)**
  - Program running on computer
  - Simulates a virtual computer running Java
- **JVM loads classes dynamically**
  - Upon first reference to class in program
  - Looks at directory / jar files in CLASSPATH
- **CLASSPATH**
  - Colon separated list of names
  - Example
    - CLASSPATH = . : \$HOME/Java

## Jar Files

- Zip file containing 1 or more .class files
- Useful for bundling many Java files
- Treated by JVM as an entire directory
- Create using
  - `jar cf [filename] [files / directories to put in jar]`

## Static Variable Initialization

### ■ Initialization Order

1. Static initializations
2. Initialization block
3. Constructor

### ■ Example

```
class Foo {  
    static Integer A = new Integer (1);           // static init  
    { A = new Integer (2); }                     // init block  
    public Foo( ) { A = new Integer (3); }       // constructor  
}
```

## Static Variable Initialization Example

```
class Foo {  
    static Integer A = new Integer (1);           // 1) static init  
    { A = new Integer (2); }                     // 2) init block  
    public Foo( ) { A = new Integer (3); }       // 3) constructor  
    public static void main (String args[]) {  
        System.out.print("A="+A+"\n");         // prints A=1  
        Foo f = new Foo();  
        System.out.print("A="+A+"\n");         // prints A=3  
    }  
}
```

## Java Topics

- Running Java programs
- Stream I/O
- Collections framework
- Java 1.5



## Stream Input/Output

- Stream
  - A connection carrying a sequence of data
    - Bytes → InputStream, OutputStream
    - Characters → FileReader, PrintWriter
  - From a source to a destination
    - Keyboard
    - File
    - Network
    - Memory
  - Basis for modern I/O

## Using Streams

- **Opening a stream**
  - Connects program to external data
  - Location of stream specified at opening
  - Only need to refer to stream
- **Usage**
  1. `import java.io.* ;`
  2. Open stream connection
  3. Use stream → read and / or write
    - Catch exceptions if needed
  4. Close stream

## Using Streams – Example

- **Reading a file**
  - **FileReader**
    - Stream used to connect to a file
  - **FileReader myFile** → `new FileReader(fileName);`
  - **fileName** → where (external) file is to be found
  - Never use **fileName** again, use **myFile** instead

## Standard Input/Output

- **Standard I/O**
  - **Provided in System class in java.lang**
  - **System.in**
    - **An instance of InputStream**
  - **System.out**
    - **An instance of PrintStream**
  - **System.err**
    - **An instance of PrintStream**

## Simple Keyboard Input

```
import java.io.* ;  
  
class BufferedReaderTest {  
    public static void main(String [] args) throws IOException {  
        // Create a BufferedReader wrapping standard input:  
        BufferedReader in =  
            new BufferedReader(new InputStreamReader(System.in)) ;  
  
        String s ;  
        s = in.readLine() ;    // Reads any string terminated by \n  
        System.out.println("s = " + s) ; // Print what was read  
    }  
}
```

## Java Topics

- Running Java programs
- Stream I/O
- Collections framework
- Java 1.5



## Java Collections Framework

- **Collection**
  - Object that groups multiple **elements** into one unit
  - Also called container
- **Collection framework** consists of
  - **Interfaces**
    - Abstract data type
  - **Implementations**
    - Reusable data structures
  - **Algorithms**
    - Reusable functionality

# Java Collections Framework

## ■ Goals

- Reduce programming effort
- Make APIs easier to learn
- Make APIs easier to design and implement
- Reuse software
- Increase performance

## Core Collection Interfaces

### ■ Collection

- Group of elements

### ■ Set

- No duplicate elements

### ■ List

- Ordered collection

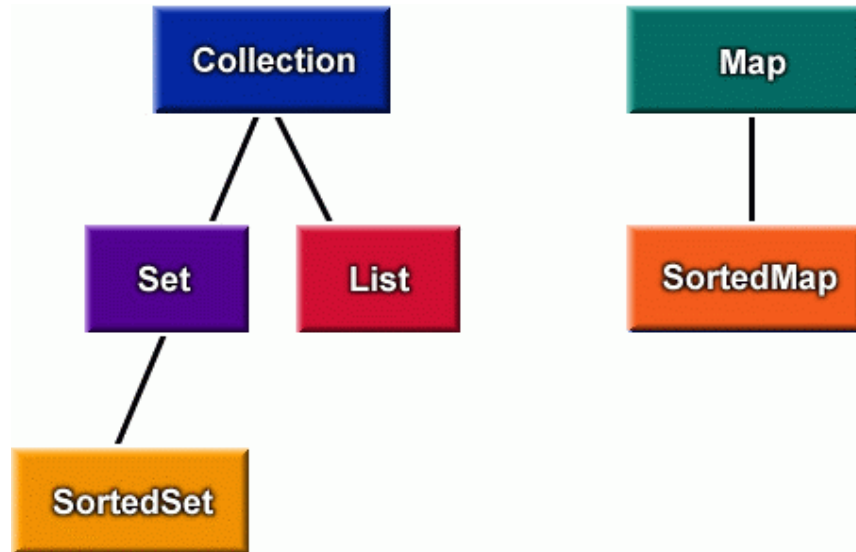
### ■ Map

- Maps keys to elements

### ■ SortedSet, SortedMap

- Sorted ordering of elements

## Core Collection Hierarchy



## Collections Interface Implementations

- **General implementations**
  - Primary public implementation
  - Example
    - List – ArrayList, LinkedList
    - Set – TreeSet, HashSet
    - Map – TreeMap, HashMap
- **Wrapper implementations**
  - Combined with other interfaces
  - Example
    - `synchronizedArrayList`, `unmodifiableHashMap`

## Collections Interface Methods

- **boolean add(Object o)**
  - Add specified element
- **boolean contains(Object o)**
  - True if collection contains specified element
- **boolean remove(Object o)**
  - Removes specified element from collection
- **boolean equals(Object o)**
  - Compares object with collection for equality
- **Iterator iterator()**
  - Returns an iterator over the elements in collection

## Collections Interface Methods

- **boolean addAll(Collection c)**
  - Adds all elements in specified collection
- **boolean containsAll(Collection c)**
  - True if collection contains all elements in collection
- **boolean removeAll(Collection c)**
  - Removes all elements in specified collection
- **boolean retainAll(Collection c)**
  - Retains only elements contained in specified collection

## Collections Interface Methods

- **void clear()**
  - Removes all elements from collection
- **boolean isEmpty()**
  - True if collection contains no elements
- **int size()**
  - Returns number of elements in collection
- **Object[] toArray()**
  - Returns array containing all elements in collection

## Collections Interface Methods

- **void shuffle(List list, Random rnd)**
  - Randomly permute list using rnd
- **void sort(List list, Comparator c)**
  - Sorts list into ascending order
  - According Comparator ordering of elements

## Iterator Interface

### ■ Iterator

- Common interface for all Collection classes
- Used to examine all elements in collection

### ■ Properties

- Order of elements is unspecified (may change)
- Can remove current element during iteration
- Works for any collection

## Iterator Interface

### ■ Interface

```
public interface Iterator {  
    boolean hasNext();  
    Object next();  
    void remove(); // optional, called once per next()  
}
```

### ■ Example usage

```
Iterator i = myCollection.iterator();  
while (i.hasNext()) {  
    myCollectionElem x = (myCollectionElem) i.next();  
}
```

## Java Topics

- Running Java programs
- Stream I/O
- Collections framework
- Java 1.5



## Java 1.5 (Tiger)

- **Description**
  - Released September 2004
  - Largest revision to Java so far
- **Goals**
  - Less code complexity
  - Better readability
  - More compile-time type safety
  - Some new functionality (generics, scanner)

## New Features in Java 1.5

- Enhanced for loop
- Enumerated types
- Autoboxing & unboxing
- Generic types
- Scanner
- Variable number of arguments (varargs)
- Static imports
- Annotations

### Enhanced For Loop

- For loop handles Iterator automatically
  - Test hasNext(), then get & cast next()
- Example

```
Iterator it = myL.iterator(); // old usage of Iterator
while (it.hasNext()) {
    Integer num = (Integer) it.next();
    // do something with num...
}
for (Integer num : myL) { // new enhanced for loop
    // do something with num...
}
```

## Enumerated Types

- **New type of variable with set of fixed values**
  - Establishes all possible values by listing them
  - Supports values(), valueOf(), name(), compareTo()...
- **Example**

```
public Class Color { // old approach to enumeration
    private int c;
    public static final Color Black = new Color(1);
    public static final Color White = new Color(2);
}
public enum Color { Black, White } // new enumeration
Color myC = Color.Black;
for (Color c : Color.values()) System.out.println(c);
```

## Autoboxing & Unboxing

- **Automatically convert primitive data types**
  - Data value  $\Leftrightarrow$  Object (of matching class)
  - Data types & classes converted
    - Boolean, Byte, Double, Short, Integer, Long, Float
- **Example**

```
ArrayList myL = new ArrayList();
myL.add(1); // previously myL.add(new Integer(1));

Integer X = new Integer(2);
int y = X; // previously int y = X.intValue();
```