2023/2024(1)
EF234302 Object Oriented Programming

Lecture #7

Collection & Generics

Misbakhul Munir IRFAN SUBAKTI
司馬伊凡
Мисбакхул Мунир Ирфан Субакти

Interface: Recap

- Constants in interfaces. Interfaces can also contain `constants` (variables) as well as `method`. E.g., within the interface `Measurable`, these are declared as:

```
int LOSE = 0;
int DRAW = 1;
int WIN = 2; /*...*/
```

```
public interface Measurable {
    int LOSE = 0;
    int DRAW = 1;
    int WIN = 2;
```

- You must not declare these as `public` or `private`, as they are automatically declared as `public static final`. We can use these constants as follows:

```
Measurable.WIN
```

```
if (status == Measurable.WIN) {
    System.out.println("You win!");
} else if (status == Measurable.DRAW) {
    System.out.println("It's a draw!");
} else { // Measurable.LOSE
    System.out.println("You lose!");
}
```

- Constants are useful for variables that *never change*, e.g., the conversion rate for pounds to kilograms.

Recursion: Recap

- A recursive method is simply a *method that calls itself.* Recursion can provide a much more elegant solution to many problems when compared to an iterative method, and in some instances much simpler methods.
- Recursion can be a little difficult to comprehend when we first come across it—well, it's *unintuitive*, though. It is quite easy to look at a recursive method and be very confused about what is happening, while the iterative version is relatively *straightforward*. The best way to think about recursion is that it splits a single, large problem into multiple, simpler, smaller operations—*divide and conquer*, ring a bel? ☺
- There are two parts of a recursive method
  - The *base case*, i.e., the simplest part of the problem, it represents the deepest layer of the recursive calls. It provides a method of terminating the repetition
  - The *recursive call*, i.e., calling the method itself with the different way/parameter compared to the original calling → to make (recursive) process keeps going.

```
24    /** Sample method: Mirror myself */
25    public void mirror() {
26        if (left != null) { // Left branch, please mirror yourself
27            left.mirror();  // This works by delegation
28        }
29        if (right != null) { // Right branch, please mirror yourself
30            right.mirror();  // This works by delegation
31        }
32        Tree originalLeft = left; // Swap the branches, mirror myself
33        left = right;
34        right = originalLeft;
35    }
```

Main.java

```
1  public class Main {
2      static void printNums(int x) {
3          if (x == 0 ) {
4              System.out.print(0);
5          } else {
6              System.out.print(x + ", ");
7              printNums(x - 1);
8          }
9      }
10     public static void main(String[] args) {
11         printNums(10);
12     }
13 }
```

Problems | Javadoc | Declaration | Search | Console

```
<terminated> Main (4) [Java Application] D:\Program\Java\JDK\bin\javaw.exe (22 Oct 2021,
10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0
```

Recursion: Helper methods

- The “standard” recursive version of the method to calculate the *n*-th Fibonacci number, usually use two recursive calls as below:

```
static int fibonacci(int n) {
      if (n == 0) {
            return 0;
      } else if (n == 1) {
            return 1;
      } else {
            return fibonacci(n – 1) + fibonacci(n – 2);
      }
}
```

- In some cases, a recursive method will need to take extra parameters as input to be able to carry out recursive operations. If we take, for instance, a tail recursive version of the method to calculate the *n*-th Fibonacci number:

```
static int fibonacci(int n, int sum, int prev, int count) {
      if (count == n - 1) {
            return sum + prev;
      }
      // Call with n, new sum, old sum and incremented counter
      return fibonacci(n, sum + prev, sum, count + 1);
}
```

- This method calculates Fibonacci numbers by passing the previous two values along with the recursive call (`sum` and `prev`). This method is much more efficient than the “standard” version above.

Main.java

```
1  package fibonacci;
2  public class Main {
3      static int fibonacci(int n) {
4          if (n == 0) {
5              return 0;
6          } else if (n == 1) {
7              return 1;
8          } else {
9              return fibonacci(n - 1) + fibonacci(n - 2);
10         }
11     }
12     public static void main(String[] args) {
13         System.out.println(fibonacci(10));
14     }
15 }
```

Problems Javadoc Declaration Search Console

<terminated> Main (5) [Java Application] D:\Program\Java\JDK\bin\javaw.exe (22 Oct 2021, 19:22:03 – 19:22:03)

55

Main.java Main2.java

```
1  package fibonacci;
2  public class Main2 {
3      static int fibonacci(int n, int sum, int prev, int count) {
4          if (count == n - 1) {
5              return sum + prev;
6          }
7          // Call with n, new sum, old sum and incremented counter
8          return fibonacci(n, sum + prev, sum, count + 1);
9      }
10     public static void main(String[] args) {
11         System.out.println(fibonacci(10, 1, 0, 1));
12     }
13 }
```

Problems Javadoc Declaration Search Console

<terminated> Main2 [Java Application] D:\Program\Java\JDK\bin\javaw.exe (22 Oct 2021, 21:32:01 – 21:32:03)

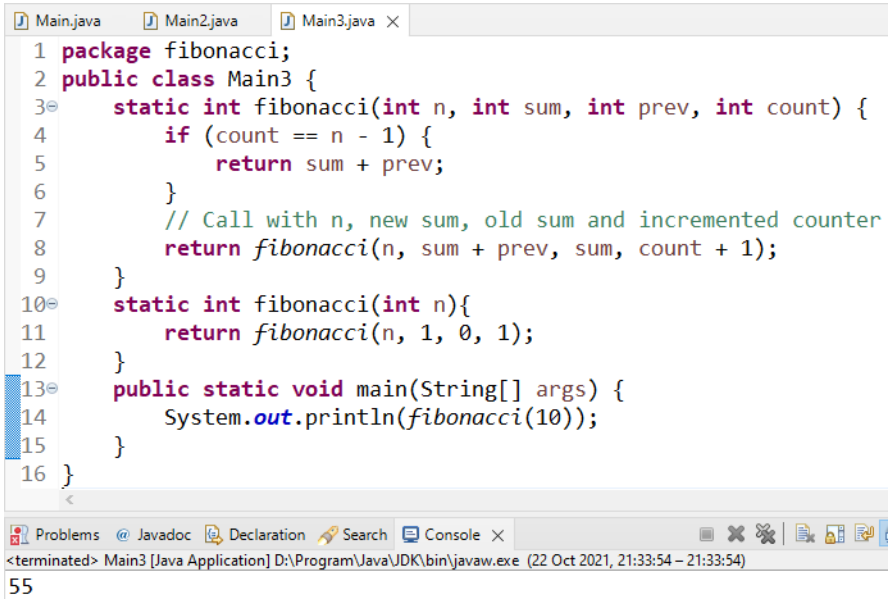
55

Recursion: Helper methods (continued)

- The main problem here, however, is that the method call is much more complicated. To call this method for `n = 10`, we will have to write `fibonacci(10, 1, 0, 1)`.
- We can improve this for the user by making use of a helper method. These can be used to simplify the method call. For example, here we could write a second method:

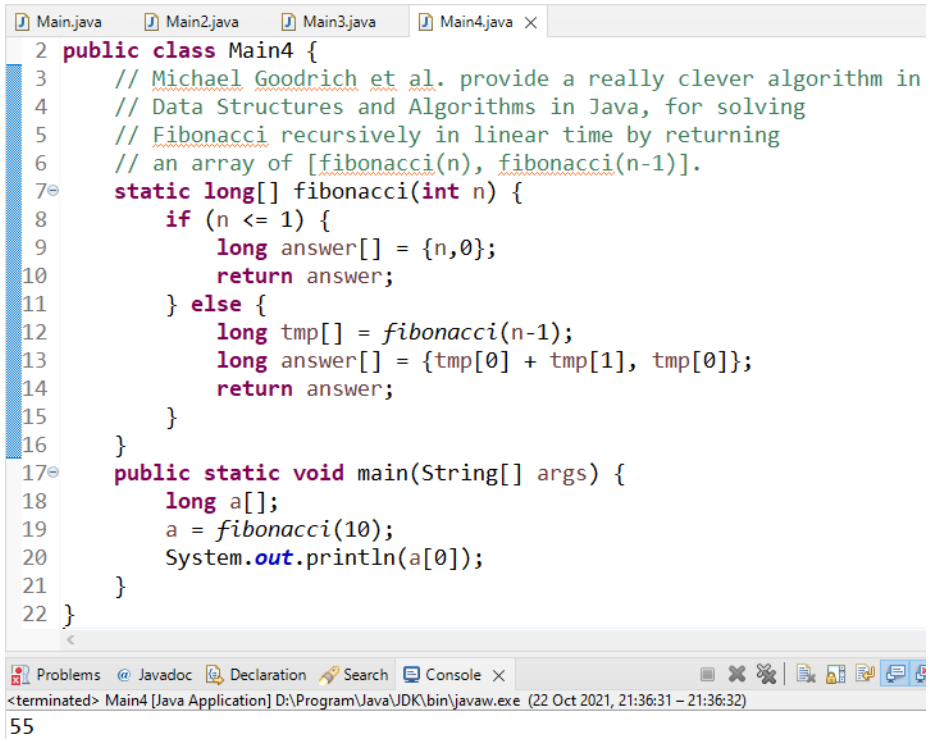
```
static int fibonacci(int n){
    return fibonacci(n, 1, 0, 1);
}
```

- The use now simply has to call `fibonacci(10)` and this new method adds in the extra parameters.



Recursion: Helper methods (continued)

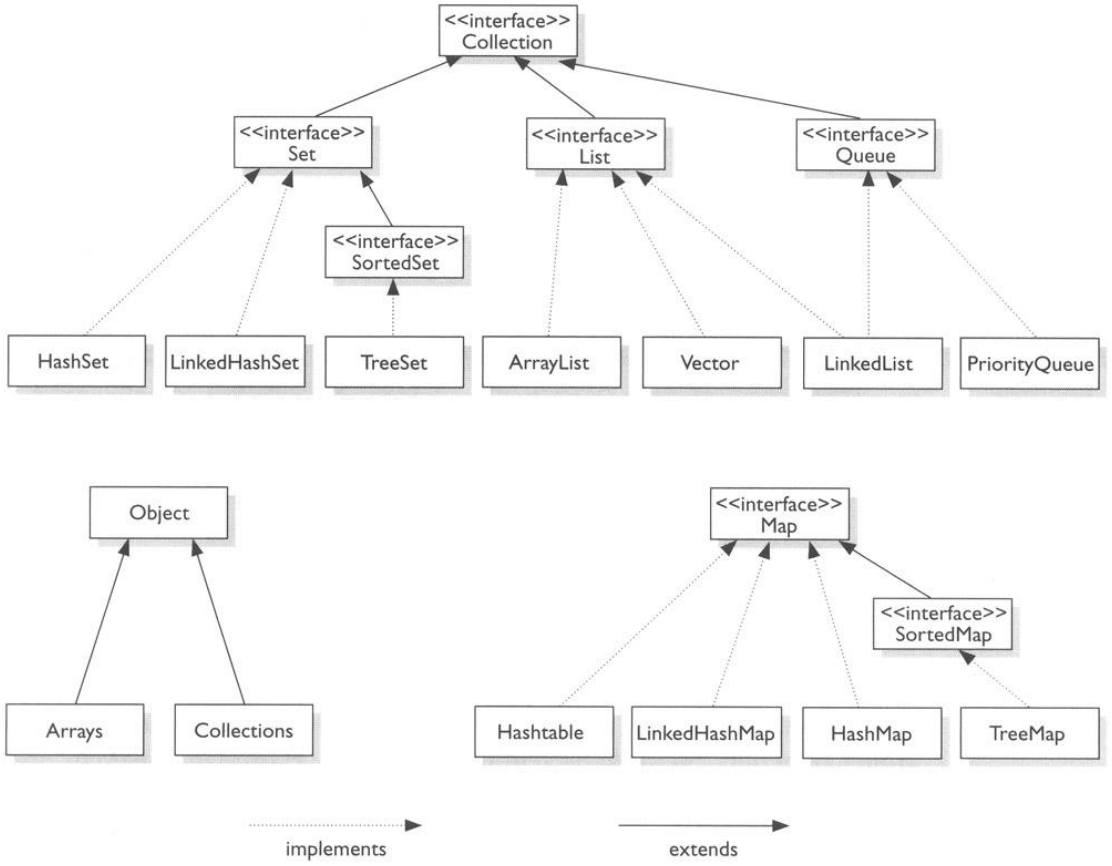
- Of course, we still can optimise this method.
- For instance, Michael Goodrich et al. provide a really clever algorithm in Data Structures and Algorithms in Java, for solving Fibonacci recursively in linear time by returning an array of `[fibonacci(n), fibonacci(n-1)]`.



Collection

- Set
- List
- Map

Collection: Diagram



Collection: Comparison

Set	List	Map
No duplicates	Duplicates	Key → value association
Order is not a concern	Order over its elements	Order is not a concern. Some subclass concerns the order: `TreeMap`
Anagram: one → neo	Get & Set	Get & Put
		Like “dictionary” or associative array: a word → a key of its value

Interface: Usage

- Interface used as general as possible → represent *data type*
- Use

```
List<String> myList =
    new ArrayList<String>();
```

- than

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

MyList.java

```
1  package myCollection;
2
3  import java.util.ArrayList;
5
6  public class MyList {
7      public static void main(String[] args) {
8          //ArrayList<String> myList = new ArrayList<String>();
9          List<String> myList = new ArrayList<String>();
10         myList.add("Durian");
11         myList.add("Mango");
12         for (String s : myList) {
13             System.out.println(s);
14         }
15     }
16 }
```

Problems Javadoc Declaration Search Console

<terminated> Main (6) [Java Application] D:\Program\Java\JDK\bin\javaw.exe (25 Oct 2021, 22:30:44 – 22:30:44)

```
Durian
Mango
```

Set: Example

```
Set<Integer> myNumbers =
  new TreeSet<Integer>();
myNumbers.add(1);
myNumbers.add(2);
myNumbers.add(3);
System.out.println(myNumbers);
// "[1, 2, 3]"

System.out.println(myNumbers.contains(7));
// "false"

System.out.println(myNumbers.add(3));
// "false"

System.out.println(myNumbers.size());
// "3"

int sum = 0;
for (int n : myNumbers) {
    sum += n;
}
```

```
System.out.println("Sum = " + sum);
// "Sum = 6"

myNumbers.addAll(Arrays.asList(1,  2,  3,  4,
5, 6, 7));

System.out.println(myNumbers);
// "[1, 2, 3, 4, 5, 6, 7]"

myNumbers.removeAll(Arrays.asList(4,  5,  6,
7, 8, 9, 10));
System.out.println(myNumbers);
// "[1, 2, 3]"

myNumbers.retainAll(Arrays.asList(2,  3,  4,
5));
System.out.println(myNumbers); // "[2, 3]"

System.out.println();
```

Set: Example (continued)

MySet.java

```
1  package myCollection;
2  import java.util.Arrays;
3  import java.util.Set;
4  import java.util.TreeSet;
5  public class MySet {
6      public static void main(String[] args) {
7          Set<Integer> myNumbers = new TreeSet<Integer>();
8          myNumbers.add(1);
9          myNumbers.add(2);
10         myNumbers.add(3);
11         System.out.println(myNumbers);
12         // "[1, 2, 3]"
13         System.out.println(myNumbers.contains(7));
14         // "false"
15         System.out.println(myNumbers.add(3));
16         // "false"
17         System.out.println(myNumbers.size());
18         // "3"
19         int sum = 0;
20         for (int n : myNumbers) {
21             sum += n;
22         }
23         System.out.println("Sum = " + sum);
24         // "Sum = 6"
25         myNumbers.addAll(Arrays.asList(1, 2, 3, 4, 5, 6, 7));
26         System.out.println(myNumbers);
27         // "[1, 2, 3, 4, 5, 6, 7]"
28         myNumbers.removeAll(Arrays.asList(4, 5, 6, 7, 8, 9, 10));
29         System.out.println(myNumbers);
30         // "[1, 2, 3]"
31         myNumbers.retainAll(Arrays.asList(2, 3, 4, 5));
32         System.out.println(myNumbers); // "[2, 3]"
33         System.out.println();
34     }
35 }
```

Problems | Javadoc | Declaration | Search | Console

<terminated> MySet [Java Application] D:\Program\Java\JDK\bin\javaw.exe

```
[1, 2, 3]
false
false
3
Sum = 6
[1, 2, 3, 4, 5, 6, 7]
[1, 2, 3]
[2, 3]
```

List: Example

```
List<String> myList =
    new ArrayList<String>();
myList.add("you");
myList.add("and");
myList.add("us");
System.out.println(myList);
// "[you, and, us]"
System.out.println(myList.contains("they"))
; // "false"
System.out.println(myList.add("us"));
// "true", duplication is allowed
System.out.println(myList.size());
// "4"

String s = "";
for (String w : myList) {
    s += w + " ";
}
```

```
System.out.println("Sentence = " + s);
// "Sentence = you and us us "
myList.addAll(Arrays.asList("love",
"them"));
System.out.println(myList);
// "[you, and, us, us, love, them]"

myList.removeAll(Arrays.asList("love",
"them", "dude"));
System.out.println(myList);
// "[you, and, us, us]"

myList.retainAll(Arrays.asList("and",  "us",
"love"));
System.out.println(myList);
// "[and, us, us]"

System.out.println();
```

List: Example (continued)

MyListEx.java

```
1  package myCollection;
2  import java.util.ArrayList;
3  import java.util.Arrays;
4  import java.util.List;
5  public class MyListEx {
6      public static void main(String[] args) {
7          List<String> myList = new ArrayList<String>();
8          myList.add("you");
9          myList.add("and");
10         myList.add("us");
11         System.out.println(myList);
12         // "[you, and, us]"
13         System.out.println(myList.contains("they")); // "false"
14         System.out.println(myList.add("us"));
15         // "true", duplication is allowed
16         System.out.println(myList.size());
17         // "4
18         String s = "";
19         for (String w : myList) {
20             s += w + " ";
21         }
22         System.out.println("Sentence = " + s);
23         // "Sentence = you and us us "
24         myList.addAll(Arrays.asList("love", "them"));
25         System.out.println(myList);
26         // "[you, and, us, us, love, them]"
27         myList.removeAll(Arrays.asList("love", "them", "dude"));
28         System.out.println(myList);
29         // "[you, and, us, us]"
30         myList.retainAll(Arrays.asList("and", "us", "love"));
31         System.out.println(myList); // "[and, us, us]"
32         System.out.println();
33     }
34 }
```

Problems | Javadoc | Declaration | Search | Console

<terminated> MyListEx [Java Application] D:\Program\Java\JDK\bin\javaw.

```
[you, and, us]
false
true
4
Sentence = you and us us
[you, and, us, us, love, them]
[you, and, us, us]
[and, us, us]
```

Map: Example

```
Map<Object, String> map = new HashMap<>();
// Actually it should be:
// Map<Object, String> map =
//   new TreeMap<Object, String>();
// but above is enough
map.put(1, "you");
map.put(2, "and");
map.put(3, "us");

Set info = map.entrySet();
Iterator iterator = info.iterator();
while (iterator.hasNext()) {
    Map.Entry m =
       (Map.Entry) iterator.next();
    int key = (Integer) m.getKey();
    String value = (String) m.getValue();
    System.out.println("key: " + key +
       " -> value: " + value);
}
```

```
// key: 1 -> value: you
// key: 2 -> value: and
// key: 3 -> value: us
System.out.println();
map.remove(2);
info = map.entrySet();
iterator = info.iterator();
while (iterator.hasNext()) {
    Map.Entry m =
       (Map.Entry) iterator.next();
    int key = (Integer) m.getKey();
    String value = (String) m.getValue();
    System.out.println("key: " + key +
       " -> value: " + value);
}
// key: 1 -> value: you
// key: 3 -> value: us
System.out.println();
```

Map: Example (continued)

MyMap.java

```
1  package myCollection;
2  import java.util.HashMap;
3  import java.util.Iterator;
4  import java.util.Map;
5  import java.util.Set;
6  public class MyMap {
7      public static void main(String[] args) {
8          Map<Object, String> map = new HashMap<>();
9          // Actually it should be:
10         // Map<Object, String> map =
11         // new TreeMap<Object, String>();
12         // but above is enough
13         map.put(1, "you");
14         map.put(2, "and");
15         map.put(3, "us");
16         Set info = map.entrySet();
17         Iterator iterator = info.iterator();
18         while (iterator.hasNext()) {
19             Map.Entry m = (Map.Entry) iterator.next();
20             int key = (Integer) m.getKey();
21             String value = (String) m.getValue();
22             System.out.println("key: " + key +
23                     " -> value: " + value);
24         }
25         // key: 1 -> value: you
26         // key: 2 -> value: and
27         // key: 3 -> value: us
28         System.out.println();
29         map.remove(2);
30         info = map.entrySet();
31         iterator = info.iterator();
32         while (iterator.hasNext()) {
33             Map.Entry m = (Map.Entry) iterator.next();
34             int key = (Integer) m.getKey();
35             String value = (String) m.getValue();
36             System.out.println("key: " + key +
37                     " -> value: " + value);
38         }
39         // key: 1 -> value: you
40         // key: 3 -> value: us
41         System.out.println();
42     }
43 }
```

Problems | Javadoc | Declaration | Search | Console

<terminated> MyMap [Java Application] D:\Program\Java\JDK\bin\javaw.e

```
key: 1 -> value: you
key: 2 -> value: and
key: 3 -> value: us

key: 1 -> value: you
key: 3 -> value: us
```

Generics

- A way to create classes that doesn't care about the object they are manipulating, as long as they are all of the same class
- An example: a `General List` (`GList.java`) class from previous class defined in `List.java`

```
public class GList<T> {
    private boolean empty;
    private T head;
    private GList<T> tail;
    public GList() {
        empty = true;
    }
    public GList(T head, GList<T> tail) {
        this.head = head;
        this.tail = tail;
    }
    public static GList nil() {
        return new GList();
    }
```

Generics: GList.java

```
public boolean empty() {
     return empty;
}
public T head() {
     return head;
}
public GList tail() {
     if (empty()) {
          throw new IllegalStateException("Trying to access tail of an empty list");
     }
     return tail;
}
@Override
public String toString() {
     return toStringHelper(this, new java.util.ArrayList<T>());
}
private String toStringHelper(GList list, java.util.List<T> accumulator) {
     if (list == null) {
        throw new IllegalStateException("next element of list was null. Should either be another list or nil. "
         + "List till now: " + accumulator);
     } else if (list.empty()) {
        return accumulator.toString();
     } else {
        accumulator.add((T)list.head());
        return toStringHelper(list.tail(), accumulator);
     }
}
}
```

Generics: `GList.java` (continued)

GList.java

```java
package myCollection;
public class GList<T> {
    private boolean empty;
    private T head;
    private GList<T> tail;
    public GList() {
        empty = true;
    }
    public GList(T head, GList<T> tail) {
        this.head = head;
        this.tail = tail;
    }
    public static GList nil() {
        return new GList();
    }
    public boolean empty() {
        return empty;
    }
    public T head() {
        return head;
    }
    public GList tail() {
        if (empty()) {
            throw new IllegalStateException("Trying to access tail of an empty list");
        }
        return tail;
    }
    @Override
    public String toString() {
        return toStringHelper(this, new java.util.ArrayList<T>());
    }
    private String toStringHelper(GList list, java.util.List<T> accumulator) {
        if (list == null) {
                throw new IllegalStateException("next element of list was null. " +
                        "Should either be another list or nil. " +
                        "List till now: " + accumulator);
        } else if (list.empty()) {
            return accumulator.toString();
        } else {
            accumulator.add((T)list.head());
            return toStringHelper(list.tail(), accumulator);
        }
    }
}
```

GList: Testing

```
GList<Integer> myGList = new GList<>();
System.out.printf("myGList: %s\n", myGList);
// myGList: []
int head = 10;
GList<Integer> myGList2 = new GList<>(head, myGList);
System.out.printf("myGList2: %s\n", myGList2);
// myGList2: [10]
head = 50;
GList<Integer> myGList3 = new GList<>(head, myGList2);
System.out.printf("myGList3: %s\n\n", myGList3);
// myGList3: [50, 10]
```

GListTest.java

```
1  package myCollection;
2
3  public class GListTest {
4      public static void main(String[] args) {
5          GList<Integer> myGList = new GList<>();
6          System.out.printf("myGList: %s\n", myGList);
7          // myGList: []
8          int head = 10;
9          GList<Integer> myGList2 = new GList<>(head, myGList);
10         System.out.printf("myGList2: %s\n", myGList2);
11         // myGList2: [10]
12         head = 50;
13         GList<Integer> myGList3 = new GList<>(head, myGList2);
14         System.out.printf("myGList3: %s\n\n", myGList3);
15         // myGList3: [50, 10]
16     }
17 }
```

Problems | Javadoc | Declaration | Search | Console

```
<terminated> GListTest [Java Application] D:\Program\Java\JDK\bin\javaw.
myGList: []
myGList2: [10]
myGList3: [50, 10]
```

- There is a problem with the above code, since `T` could be any class, and not all classes implement the functionality needed for the methods we've written.
  - ✓ Java classes don't have a natural ordering, i.e., (1, 2, 3, …)
  - ✓ A bounded type comes to fix the problem. In our new class we need to put the term "`<T extends Comparable<T>>`" like in the following.
  - ✓ An example: a General List 2 (`GList2.java`) class from previous class defined in `List.java`

GList2.java

```
public class GList2<T extends Comparable<T>> {
    private boolean empty;
    private T head;
    private GList2<T> tail;
    public GList2() {
        empty = true;
    }
    public GList2(T head, GList2<T> tail) {
        this.head = head;
        this.tail = tail;
    }
    public static GList2 nil() {
        return new GList2();
    }
    public boolean empty() {
        return empty;
    }
    public T head() {
        return head;
    }
    public GList2 tail() {
        if (empty()) {
            throw new IllegalStateException("Trying to access tail of an empty list");
        }
        return tail;
    }
```

GList2.java (continued)

```
public <U extends Comparable<U>> boolean sorted(GList2<T> list) {
   if (list.empty() || list.tail().empty()) {
      return true;
   } else {
      T a = list.head();
      T b = (T) list.tail().head();
      int compareResult = a.compareTo(b);
      // 0 means they are equal
      // negative (<0) means a comes before b
      // positive (>0) means a comes after b
      if (compareResult > 0) {
         return false;
      } else {
         return sorted(list.tail());
      }
   }
}
```

GList2.java (continued)

```
   @Override
   public String toString() {
      return toStringHelper(this, new java.util.ArrayList<T>());
   }
   private String toStringHelper(GList2 list, java.util.List<T> accumulator) {
      if (list == null) {
         throw new IllegalStateException("next element of list was null. " +
            "Should either be another list or nil. " + "List till now: " +
            accumulator);
      } else if (list.empty()) {
         return accumulator.toString();
      } else {
         accumulator.add((T)list.head());
         return toStringHelper(list.tail(), accumulator);
      }
   }
}
```

GList2.java (continued)

GList2.java

```java
package myCollection;
public class GList2<T extends Comparable<T>> {
    private boolean empty;
    private T head;
    private GList2<T> tail;
    public GList2() {
        empty = true;
    }
    public GList2(T head, GList2<T> tail) {
        this.head = head;
        this.tail = tail;
    }
    public static GList2 nil() {
        return new GList2();
    }
    public boolean empty() {
        return empty;
    }
    public T head() {
        return head;
    }
    public GList2 tail() {
        if (empty()) {
            throw new IllegalStateException("Trying to access tail" +
                " of an empty list");
        }
        return tail;
    }
    public <U extends Comparable<U>> boolean sorted(GList2<T> list) {
        if (list.empty() || list.tail().empty()) {
            return true;
        } else {
            T a = list.head();
            T b = (T) list.tail().head();
            int compareResult = a.compareTo(b);
            // 0 means they are equal
            // negative (<0) means a comes before b
            // positive (>0) means a comes after b
            if (compareResult > 0) {
                return false;
            } else {
                return sorted(list.tail());
            }
        }
    }
    @Override
    public String toString() {
        return toStringHelper(this, new java.util.ArrayList<T>());
    }
    private String toStringHelper(GList2 list, java.util.List<T> accumulator) {
        if (list == null) {
            throw new IllegalStateException("next element of list was null. " +
                "Should either be another list or nil. " + "List till now: " +
                accumulator);
        } else if (list.empty()) {
            return accumulator.toString();
        } else {
            accumulator.add((T) list.head());
            return toStringHelper(list.tail(), accumulator);
        }
    }
}
```

GList2: Testing

```
GList2<Integer> myGList4 = new GList2<>();
System.out.printf("myGList4: %s\n", myGList4); // myGList4: []
System.out.printf("myGList4 is sorted: %b\n", myGList4.sorted(myGList4));
// myGList4 is sorted: true
int head = 5;
GList2<Integer> myGList5 = new GList2<>(head, myGList4);
System.out.printf("myGList5: %s\n", myGList5); // myGList5: [5]
System.out.printf("myGList5 is sorted: %b\n", myGList5.sorted(myGList5));
// myGList5 is sorted: true
head = 10;
GList2<Integer> myGList6 = new GList2<>(head, myGList5);
System.out.printf("myGList6: %s\n", myGList6); // myGList6: [10, 5]
System.out.printf("myGList6 is sorted: %b\n", myGList6.sorted(myGList6));
// myGList6 is sorted: false
head = 3;
GList2<Integer> myGList7 = new GList2<>(head, myGList5);
System.out.printf("myGList7: %s\n", myGList7); // myGList7: [3, 5]
System.out.printf("myGList7 is sorted: %b\n", myGList7.sorted(myGList7));
// myGList7 is sorted: true
```

GList2: Testing (continued)

GList2Test.java

```java
package myCollection;
public class GList2Test {
    public static void main(String[] args) {
        GList2<Integer> myGList4 = new GList2<>();
        System.out.printf("myGList4: %s\n", myGList4); // myGList4: []
        System.out.printf("myGList4 is sorted: %b\n", myGList4.sorted(myGList4));
        // myGList4 is sorted: true
        int head = 5;
        GList2<Integer> myGList5 = new GList2<>(head, myGList4);
        System.out.printf("myGList5: %s\n", myGList5); // myGList5: [5]
        System.out.printf("myGList5 is sorted: %b\n", myGList5.sorted(myGList5));
        // myGList5 is sorted: true
        head = 10;
        GList2<Integer> myGList6 = new GList2<>(head, myGList5);
        System.out.printf("myGList6: %s\n", myGList6); // myGList6: [10, 5]
        System.out.printf("myGList6 is sorted: %b\n", myGList6.sorted(myGList6));
        // myGList6 is sorted: false
        head = 3;
        GList2<Integer> myGList7 = new GList2<>(head, myGList5);
        System.out.printf("myGList7: %s\n", myGList7); // myGList7: [3, 5]
        System.out.printf("myGList7 is sorted: %b\n", myGList7.sorted(myGList7));
        // myGList7 is sorted: true
    }
}
```

Problems Javadoc Declaration Search Console

<terminated> GList2Test [Java Application] D:\Program\Java\JDK\bin\javaw

```
myGList4: []
myGList4 is sorted: true
myGList5: [5]
myGList5 is sorted: true
myGList6: [10, 5]
myGList6 is sorted: false
myGList7: [3, 5]
myGList7 is sorted: true
```

Wildcards

AnimalTest0.java

```
1 package myCollection;
2 import java.util.ArrayList;
3 import java.util.List;
4 public class AnimalTest0 {
5     public static void main(String[] args) {
6         List<Cat> cats = new ArrayList<Cat>();
7         List<Animal> animals = cats; // Error
8     }
9 }
```

- ✓ A problem with Generics is that the following code is wrong (even though Cat extends Animal)

```
List<Cat> cats = new ArrayList<Cat>();
List<Animal> animals = cats; // Error
```

- ✓ Since Dog is also an instance of Animal, and if we allow these codes then the following code will break the contract that the list cats only contained Cats.

```
animals.add(new Dog("wondergirl"));
```

- ✓ For the detail, see the following codes.

AnimalTest0.java

```
1 package myCollection;
2 import java.util.ArrayList;
3 import java.util.List;
4 public class AnimalTest0 {
5     public static void main(String[] args) {
6         List<Cat> cats = new ArrayList<Cat>();
7         List<Animal> animals = cats; // Error
8         animals.add(new Dog("wondergirl")); // Error
9     }
10 }
```

Animal.java

```
public class Animal {
    private String name;
    public Animal() {
    }
    public Animal(String name) {
        this.name = name;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    @Override
    public String toString() {
        return "This is Animal class";
    }
}
```

Animal.java

```
1  package myCollection;
2  public class Animal {
3      private String name;
4      public Animal() {
5      }
6      public Animal(String name) {
7          this.name = name;
8      }
9      public String getName() {
10         return name;
11     }
12     public void setName(String name) {
13         this.name = name;
14     }
15     @Override
16     public String toString() {
17         return "This is Animal class";
18     }
19 }
```

Cat.java

```
public class Cat extends Animal {
    public Cat() {
    }
    @Override
    public String toString() {
        return "This is Cat class";
    }
}
```

Cat.java

```
1 package myCollection;
2 public class Cat extends Animal {
3     public Cat() {
4     }
5     @Override
6     public String toString() {
7         return "This is Cat class";
8     }
9 }
```

Dog.java

```
public class Dog extends Animal {
    public Dog() {
    }
    @Override
    public String toString() {
        return "This is Dog class";
    }
}
```

Dog.java

```
1 package myCollection;
2 public class Dog extends Animal {
3     public Dog() {
4     }
5     @Override
6     public String toString() {
7         return "This is Dog class";
8     }
9 }
```

The Code: Testing 1

```
Cat cat1 = new Cat();
cat1.setName("superman");
Cat cat2 = new Cat();
cat2.setName("spiderman");
List<Cat> cats = new ArrayList<Cat>();
cats.add(cat1);
cats.add(cat2);
Iterator iterator = cats.iterator();
while (iterator.hasNext()) {
    Cat c = (Cat) iterator.next();
    System.out.println(c.getName());
}
// superman
// spiderman
System.out.println();

Dog dog1 = new Dog();
dog1.setName("supergirl");
Dog dog2 = new Dog();
dog2.setName("spidergirl");
List<Dog> dogs = new ArrayList<>();
dogs.add(dog1);
dogs.add(dog2);
```

The Code: Testing 1 (continued)

```
iterator = dogs.iterator();
while (iterator.hasNext()) {
    Dog d = (Dog) iterator.next();
    System.out.println(d.getName());
}
// supergirl
// spidergirl
System.out.println();

//      List<Animal> animals = cats;
// Above is ERROR, since Dog is also an instance of Animal and
// we allow the above, then the following code will break the contract
// that the list cats only contained Cats = male superhero name
//      animals.add(new Dog("wondergirl"));
```

The Code: Testing 1 (continued)

AnimalTest1.java

```java
package myCollection;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
public class AnimalTest1 {
    public static void main(String[] args) {
        Cat cat1 = new Cat();
        cat1.setName("superman");
        Cat cat2 = new Cat();
        cat2.setName("spiderman");
        List<Cat> cats = new ArrayList<>();
        cats.add(cat1);
        cats.add(cat2);
        Iterator iterator = cats.iterator();
        while (iterator.hasNext()) {
            Cat c = (Cat) iterator.next();
            System.out.println(c.getName());
        }
        // superman
        // spiderman
        System.out.println();
        Dog dog1 = new Dog();
        dog1.setName("supergirl");
        Dog dog2 = new Dog();
        dog2.setName("spidergirl");
        List<Dog> dogs = new ArrayList<>();
        dogs.add(dog1);
        dogs.add(dog2);
        iterator = dogs.iterator();
        while (iterator.hasNext()) {
            Dog d = (Dog) iterator.next();
            System.out.println(d.getName());
        }
        // supergirl
        // spidergirl
        System.out.println();
        //            List<Animal> animals = cats;
        // Above is ERROR, since Dog is also an instance of Animal and
        // we allow the above, then the following code will break the contract
        // that the list cats only contained Cats = male superhero name
        //            animals.add(new Dog("wondergirl"));
    }
}
```

Problems | Javadoc | Declaration | Search | Console

<terminated> AnimalTest1 [Java Application] D:\Program\Java\JDK\bin\jav

```
superman
spiderman

supergirl
spidergirl
```

The Code: Testing 1 (continued)

- In Generics there's a concept of a **wildcard**. It allows us to specify that we don't care about the actual generic type, just that it conforms some behaviour. See the code below as an example.

```
List<? extends Animal> animals = cats;
Animal animal1 = animals.get(0);
System.out.println(animal1.getName()); // superman
Animal animal2 = animals.get(1);
System.out.println(animal2.getName()); // spiderman
// Here we cannot use add or set or any method that has a generic type
// as an argument, since it might break the contract that cats only
// contained Cats and not other Animals
//      so: animals.add(dog1) will raise ERROR
// We only able to access the elements in a list, but not add any elements
// to a generic list
System.out.println();
```

The Code: Testing 1 (continued)

AnimalTest1a.java

```
1  package myCollection;
2  import java.util.ArrayList;
4  public class AnimalTest1a {
5      public static void main(String[] args) {
6          Cat cat1 = new Cat();
7          cat1.setName("superman");
8          Cat cat2 = new Cat();
9          cat2.setName("spiderman");
10         List<Cat> cats = new ArrayList<>();
11         cats.add(cat1);
12         cats.add(cat2);
13
14         List<? extends Animal> animals = cats;
15         Animal animal1 = animals.get(0);
16         System.out.println(animal1.getName()); // superman
17         Animal animal2 = animals.get(1);
18         System.out.println(animal2.getName()); // spiderman
19         // Here we cannot use add or set or any method that has a generic type
20         // as an argument, since it might break the contract that cats only
21         // contained Cats and not other Animals
22         //            so: animals1.add(dog1) will raise ERROR
23         // We only able to access the elements in a list, but not add any elements
24         // to a generic list
25         System.out.println();
26     }
27 }
```

Problems Javadoc Declaration Search Console

<terminated> AnimalTest1a [Java Application] D:\Program\Java\JDK\bin\jav

```
superman
spiderman
```

The Code: Testing 1 (continued)

- So, we only able to access the elements in a list, but cannot add any elements to a generic list. Instead of writing "`? extends X`" we'll write "`? super X`". It means it's guaranteed that the wildcard is a supertype of X.
- For the detail, see the following codes.
  - ✓ Add a function to the main program.

```
public static <T> void addAll(List<T> src, List<? super T> dest) {
    for (T element : src) { // add every element in src to dest
       dest.add(element);
    }
}
```

The Code: Testing 2

```
List<Animal> newAnimals = new ArrayList<>();
addAll(cats, newAnimals);
Iterator iterator = newAnimals.iterator();
while (iterator.hasNext()) {
    Animal a = (Animal) iterator.next();
    System.out.println(a.getName());
}
// superman
// spiderman
System.out.println();

// Then we add dogs to newAnimals, so newAnimals now contains
// both cats and dogs
addAll(dogs, newAnimals);
iterator = newAnimals.iterator();
while (iterator.hasNext()) {
    Animal a = (Animal) iterator.next();
    System.out.println(a.getName());
}
// superman
// spiderman
// supergirl
// spidergirl
```

AnimalTest2.java

```
1  package myCollection;
2  import java.util.ArrayList;
3  import java.util.Iterator;
4  import java.util.List;
5  public class AnimalTest2 {
6      public static <T> void addAll(List<T> src, List<? super T> dest) {
7          for (T element : src) { // add every element in src to dest
8              dest.add(element);
9          }
10     }
11     public static void main(String[] args) {
12         Cat cat1 = new Cat();
13         cat1.setName("superman");
14         Cat cat2 = new Cat();
15         cat2.setName("spiderman");
16         List<Cat> cats = new ArrayList<>();
17         cats.add(cat1);
18         cats.add(cat2);
19         Dog dog1 = new Dog();
20         dog1.setName("supergirl");
21         Dog dog2 = new Dog();
22         dog2.setName("spidergirl");
23         List<Dog> dogs = new ArrayList<>();
24         dogs.add(dog1);
25         dogs.add(dog2);
26
27         List<Animal> newAnimals = new ArrayList<>();
28         addAll(cats, newAnimals);
29         Iterator iterator = newAnimals.iterator();
30         while (iterator.hasNext()) {
31             Animal a = (Animal) iterator.next();
32             System.out.println(a.getName());
33         }
34         // superman
35         // spiderman
36         System.out.println();
37         // Then we add dogs to newAnimals, so newAnimals now contains
38         // both cats and dogs
39         addAll(dogs, newAnimals);
40         iterator = newAnimals.iterator();
41         while (iterator.hasNext()) {
42             Animal a = (Animal) iterator.next();
43             System.out.println(a.getName());
44         }
45         // superman
46         // spiderman
47         // supergirl
48         // spidergirl
49     }
50 }
```

Problems | Javadoc | Declaration | Search | Console

```
<terminated> AnimalTest2 [Java Application] D:\Program\Java\JDK\bin\java
superman
spiderman

superman
spiderman
supergirl
spidergirl
```

Reading a file

```
import java.io.*;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class MyFileReader {

	public static void main(String[] args) {
		// 1st method
		Scanner fileScan = null;
		try {
			fileScan = new Scanner(new File("D:/word.txt"));

			// Read and process each word of the file
			String word;
			while (fileScan.hasNext()) {
				word = fileScan.nextLine().toLowerCase();
			// ...
			}
		} catch (IOException e) {
			System.err.println(e);
		} finally {
			fileScan.close();
		}
```

MyFileReader.java

```
1  import java.io.*;
2  import java.util.ArrayList;
3  import java.util.List;
4  import java.util.Scanner;
5
6  public class MyFileReader {
7      public static void main(String[] args) {
8          // 1st method
9          Scanner fileScan = null;
10         try {
11             fileScan = new Scanner(new File("D:/word.txt"));
12             // Read and process each word of the file
13             String word;
14             while (fileScan.hasNext()) {
15                 word = fileScan.nextLine().toLowerCase();
16                 // ...
17             }
18         } catch (IOException e) {
19             System.err.println(e);
20         } finally {
21             fileScan.close();
22         }
23         // 2nd method
24         BufferedReader br = null;
25         List<String> stringList = new ArrayList<String>();
26         try {
27             String currentLine;
28             br = new BufferedReader(new FileReader("D:/word.txt"));
29             while ((currentLine = br.readLine()) != null) {
30                 stringList.add(currentLine);
31             }
32         } catch (IOException e) {
33             e.printStackTrace();
34         } finally {
35             try {
36                 if (br != null) {
37                     br.close();
38                 }
39             } catch (IOException ex) {
40                 ex.printStackTrace();
41             }
42         }
43         // print the list out
44         System.out.println("\nNow we want to print the list out:");
45         for (String s : stringList) {
46             System.out.println(s);
47         }
48     }
49 }
```

Reading a file (continued)

```
		// 2nd method
		BufferedReader br = null;
		List<String> stringList = new ArrayList<String>();

		try {
			String currentLine;
			br = new BufferedReader(new FileReader("D:/word.txt"));
			while ((currentLine = br.readLine()) != null) {
				stringList.add(currentLine);
			}
		} catch (IOException e) {
			e.printStackTrace();
		} finally {
			try {
				if (br != null) {
					br.close();
				}
			} catch (IOException ex) {
				ex.printStackTrace();
			}
		}
		// print the list out
		System.out.println("\nNow we want to print the list out:");
		for (String s : stringList) {
			System.out.println(s);
		}

	}
}
```

word.txt - Notepad

File Edit Format View Help

```
import java.io.*;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class MyFileReader {
	public static void main(String[] args) {
		// 1st method
		Scanner fileScan = null;
		try {
			fileScan = new Scanner(new File("D:/word.txt"));
			// Read and process each word of the file
			String word;
			while (fileScan.hasNext()) {
				word = fileScan.nextLine().toLowerCase();
				// ...
			}
		} catch (IOException e) {
			System.err.println(e);
		} finally {
			fileScan.close();
		}
		// 2nd method
```

Problems Javadoc Declaration Search Console

<terminated> MyFileReader [Java Application] D:\Program\Java\JDK\bin\javaw.exe (26 Oct 2021, 22:24:45 – 22:24:49)

```
Now we want to print the list out:
import java.io.*;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class MyFileReader {
	public static void main(String[] args) {
		// 1st method
		Scanner fileScan = null;
		try {
			fileScan = new Scanner(new File("D:/word.txt"));
			// Read and process each word of the file
			String word;
			while (fileScan.hasNext()) {
				word = fileScan.nextLine().toLowerCase();
				// ...
			}
```

Collection: Other examples

- Student.java

```
public class Student {
    final private int age;
    Student(int age) {
        this.age = age;
    }
    public int getAge() {
        return age;
    }
}
```

Student.java

```
1  package myCollection;
2  public class Student {
3      final private int age;
4      Student(int age) {
5          this.age = age;
6      }
7      public int getAge() {
8          return age;
9      }
10 }
```

Collection: Other examples (continued)

- AgeComparator.java

```
// Using list -> sort all the element in order, by Collections.sort()
// Not all lists can be sorted.
// To sort a generic type of list:
// (1) implement Comparable
// (2) write a Comparator class for the objects we want to sort
public class AgeComparator implements Comparator<Student> {
    @Override
    public int compare(Student s1, Student s2) {
        if (s1 == s2) {
            return 0; // the same age
        } else if (s1 == null) {
            return -1; // s1 is older than s2
        } else if (s2 == null) {
            return 1; // s2 is older than s1
        } else {
            return s1.getAge() - s2.getAge();
        }
    }
}
```

AgeComparator.java

```
1  package myCollection;
2  import java.util.Comparator;
3  //Using list -> sort all the element in order, by Collections.sort()
4  //Not all lists can be sorted.
5  //To sort a generic type of list:
6  //(1) implement Comparable
7  //(2) write a Comparator class for the objects we want to sort
8  public class AgeComparator implements Comparator<Student> {
9      @Override
10     public int compare(Student s1, Student s2) {
11         if (s1 == s2) {
12             return 0; // the same age
13         } else if (s1 == null) {
14             return -1; // s1 is older than s2
15         } else if (s2 == null) {
16             return 1; // s2 is older than s1
17         } else {
18             return s1.getAge() - s2.getAge();
19         }
20     }
21 }
```

Collection: Other examples (continued)

- MyCollection.java

```
public class MyCollection {
    public static<T> List<T> removeDuplicatesFromSortedList(List<T> input) {
        T previous = null;
        List<T> result = new ArrayList<>();
        result.add(input.get(0));
        Iterator<T> iter = input.iterator();
        while (iter.hasNext()) {
            T current = iter.next();
            if (previous != null && !previous.equals(current)) {
                result.add(current);
            }
            previous = current;
        }
        return result;
    }
```

Collection: Other examples (continued)

- MyCollection.java (continued)

```
public static void main(String[] args) {
    System.out.println("Part 1: Iterator");
    Collection<String> myCollection = new ArrayList<>();
    myCollection.add("cat");
    Iterator<String> iterator = myCollection.iterator();
    while (iterator.hasNext()) {
        System.out.println(iterator.next());
    }

    System.out.printf("\nPart 2: Iterator is only read not change\n");
    Iterator<String> iterator2  = myCollection.iterator();
    while (iterator2.hasNext()) {
        String element = iterator2.next(); // there is no iterator2.prev()
        // Iterator only read the contents and not change it
        element += " and dog";
        System.out.println(element);
    }
```

Collection: Other examples (continued)

- MyCollection.java (continued)

```
System.out.printf("\nPart 3: Removing duplicate values\n");
myCollection.add("dog");
myCollection.add("cat");
myCollection.add("monkey");
Iterator<String> iterator3  = myCollection.iterator();
while (iterator3.hasNext()) {
    System.out.println(iterator3.next());
}

System.out.printf("\nAfter removing the duplicate values\n");
Collection<String> myCollection2 =
    removeDuplicatesFromSortedList((List<String>) myCollection);
Iterator<String> iterator4  = myCollection2.iterator();
while (iterator4.hasNext()) {
    System.out.println(iterator4.next());
}
```

Collection: Other examples (continued)

- MyCollection.java (continued)

```
System.out.printf("\nPart 4: Iterable\n");
Collection<Integer> it = new ArrayList<>();
it.add(3);
it.add(5);
it.add(7);
int sum = 0;
for (Integer i : it) {
     sum += i;
}
System.out.println("sum = " + sum);

System.out.printf("\nPart 5: Comparator\n");
Collection<Student> students = new ArrayList<>();
Student s = new Student(15);
students.add(s);
Student s1 = new Student(20);
students.add(s1);
Student s2 = new Student(12);
students.add(s2);
```

Collection: Other examples (continued)

- MyCollection.java (continued)

```
            Iterator<Student> itStudents = students.iterator();
            while (itStudents.hasNext()) {
                System.out.println(itStudents.next().getAge());
            }
            AgeComparator age = new AgeComparator();
            System.out.printf("Age difference between s1 (age %d) and s2 (age %d) = %d\n",
                s1.getAge(), s2.getAge(), age.compare(s1, s2));
            System.out.println("After sort is called, the list of student becomes:");
            Collections.sort((List<Student>) students, new AgeComparator());
            Iterator<Student> itStudents2 = students.iterator();
            while (itStudents2.hasNext()) {
                System.out.println(itStudents2.next().getAge());
            }
        }
    }
```

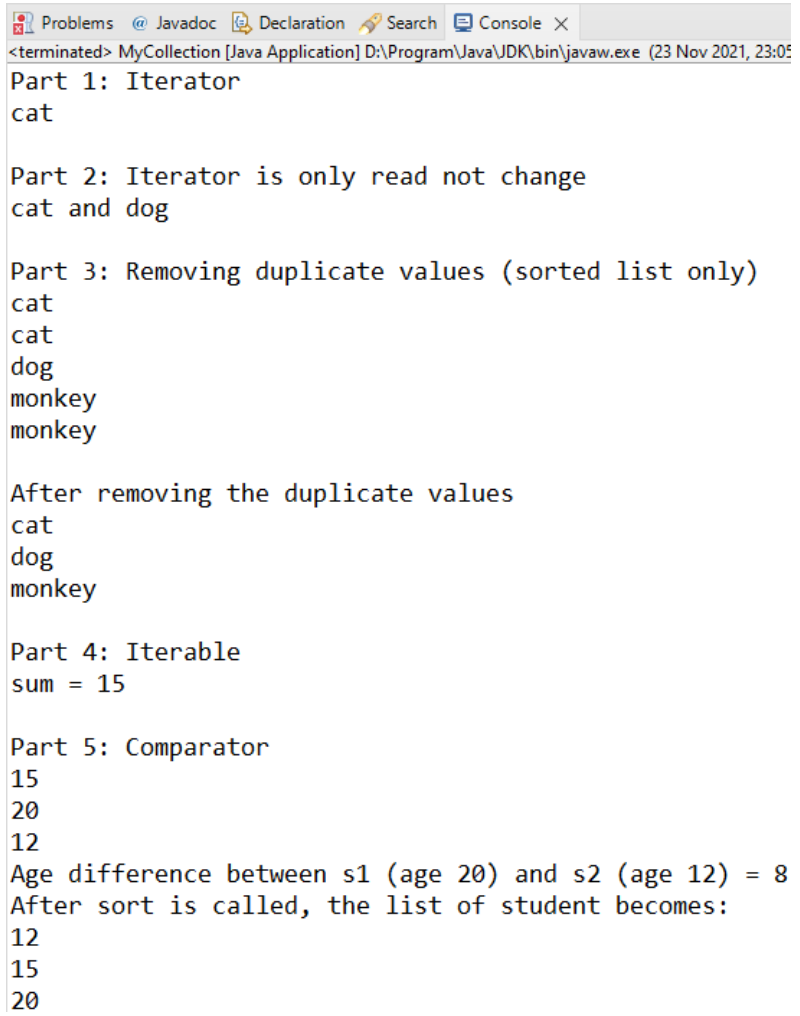
Collection: Other examples (continued)

MyCollection.java

```java
package myCollection;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Collections;
import java.util.Iterator;
import java.util.List;
public class MyCollection {
    public static <T> List<T> removeDuplicatesFromSortedList(List<T> input) {
        T previous = null;
        List<T> result = new ArrayList<>();
        result.add(input.get(0));
        Iterator<T> iter = input.iterator();
        while (iter.hasNext()) {
            T current = iter.next();
            if (previous != null && !previous.equals(current)) {
                result.add(current);
            }
            previous = current;
        }
        return result;
    }
    public static void main(String[] args) {
        System.out.println("Part 1: Iterator");
        Collection<String> myCollection = new ArrayList<>();
        myCollection.add("cat");
        Iterator<String> iterator = myCollection.iterator();
        while (iterator.hasNext()) {
            System.out.println(iterator.next());
        }
        System.out.printf("\nPart 2: Iterator is only read not change\n");
        Iterator<String> iterator2 = myCollection.iterator();
        while (iterator2.hasNext()) {
            String element = iterator2.next(); // there is no iterator2.prev()
            // Iterator only read the contents and not change it
            element += " and dog";
            System.out.println(element);
        }
        System.out.printf("\nPart 3: Removing duplicate values (sorted list only)\n");
        myCollection.add("cat");
        myCollection.add("dog");
        myCollection.add("monkey");
        myCollection.add("monkey");
        Iterator<String> iterator3 = myCollection.iterator();
        while (iterator3.hasNext()) {
            System.out.println(iterator3.next());
        }
```

Collection: Other examples (continued)

```
47          System.out.printf("\nAfter removing the duplicate values\n");
48          Collection<String> myCollection2 =
49                  removeDuplicatesFromSortedList((List<String>) myCollection);
50          Iterator<String> iterator4 = myCollection2.iterator();
51          while (iterator4.hasNext()) {
52              System.out.println(iterator4.next());
53          }
54          System.out.printf("\nPart 4: Iterable\n");
55          Collection<Integer> it = new ArrayList<>();
56          it.add(3);
57          it.add(5);
58          it.add(7);
59          int sum = 0;
60          for (Integer i : it) {
61              sum += i;
62          }
63          System.out.println("sum = " + sum);
64          System.out.printf("\nPart 5: Comparator\n");
65          Collection<Student> students = new ArrayList<>();
66          Student s = new Student(15);
67          students.add(s);
68          Student s1 = new Student(20);
69          students.add(s1);
70          Student s2 = new Student(12);
71          students.add(s2);
72          Iterator<Student> itStudents = students.iterator();
73          while (itStudents.hasNext()) {
74              System.out.println(itStudents.next().getAge());
75          }
76          AgeComparator age = new AgeComparator();
77          System.out.printf("Age difference between s1 (age %d) and s2 (age %d) = %d\n",
78                  s1.getAge(), s2.getAge(), age.compare(s1, s2));
79          System.out.println("After sort is called, the list of student becomes:");
80          Collections.sort((List<Student>) students, new AgeComparator());
81          Iterator<Student> itStudents2 = students.iterator();
82          while (itStudents2.hasNext()) {
83              System.out.println(itStudents2.next().getAge());
84          }
85      }
86  }
```



Collection: Other examples (continued)

- Output

```
Part 1: Iterator
cat

Part 2: Iterator is only read not change
cat and dog

Part 3: Removing duplicate values (sorted list only)
cat
cat
dog
monkey
monkey
After removing the duplicate values
cat
dog
monkey

Part 4: Iterable
sum = 15

Part 5: Comparator
15
20
12
Age difference between s1 (age 20) and s2 (age 12) = 8
After sort is called, the list of student becomes:
12
15
20
```