12 Java Generics and Collections

Problema 1

Which of these statements concerning maps are true?

You had to select 1 option

- It is permissible for a map to contain itself as a key.
    - But it can contain itself as a value.
- values() method returns an instance of Set.
    - It returns a Collection.
- The Map interface extends the Collection interface.
- All keys in a map are unique.
- All Map implementations keep the keys sorted.
    - Only TreeMap does. Implementations of SortedMap keep the entries sorted.

Explanation:
While all the keys in a map must be unique, multiple identical values may exist. Since values are not unique, the values() method returns a Collection instance and not a Set instance.

Problema 2

Which of the following statements are correct?
You had to select 1 option

- A List stores elements in a Sorted Order.
    - List just keeps elements in the order in which they are added. For sorting, you'll need some other interface such as a SortedSet.
- A Set keeps the elements sorted and a List keeps the elements in the order they were added.
    - A Set keeps unique objects without any order or sorting. A List keeps the elements in the order they were added. A List may have non-unique elements.
- A SortedSet keeps the elements in the order they were added.
    - A SortedSet keeps unique elements in their natural order.
- An OrderedSet keeps the elements sorted.
    - There is no interface like OrderedSet.
- An OrderedList keeps the elements ordered.
    - There is no such interface. The List interface itself is meant for keeping the elements in the order they were added.
- A NavigableSet keeps the elements sorted.
    - A NavigableSet is a SortedSet extended with navigation methods reporting closest matches for given search targets. Methods lower, floor, ceiling, and higher return elements respectively less than, less than or equal, greater than or equal, and greater than a given element, returning null if there is no such element. Since NavigableSet is a SortedSet, it keeps the elements sorted.

Problema 3

Given the following class:

```
public class Food{  // LINE 1
  String name;
  int caloriesPerServing;
  public Food(String name, int calories){
    this.name = name; this.caloriesPerServing = calories;
  }
  //accessors not shown

  //LINE 2
}
```

```
This class is used in an application as follows -

ArrayList<Food> al = new ArrayList<>();
//code that adds Food objects to al not shown
Collections.sort(al);
```

What changes must be done to Food class so that the call to Collections.sort(al) will work as expected?

You had to select 2 options

- Replace line 1 with :
  ```
  public class Food implements Comparable<Food>{
  ```
- Replace line 1 with :
  ```
  public class Food implements Comparator<Food>{
  ```
- Replace line 1 with :
  ```
  public class Food extends Comparator<Food>{
  ```
- Replace line 1 with :
  ```
  public class Food extends Comparable<Food>{
  ```
- Add the following a line 2 :
  ```
  public int compareTo(Food f){
    return this.name.compareTo(f.name); }
  ```
- Add the following a line 2 :
  ```
  public boolean compare(Food f){
    return this.name.compare(f.name); }
  ```
- Add the following a line 2 :
  ```
  public int compare(Food f){
          return this.name.compare(f.name); }
  ```

Explanation:

For `Collections.sort(List<T> )` method to sort a `Collection` of any class T, that class must implement `java.lang.Comparable` interface. This interface has only one method: `public int compareTo(T o)`, where T is same as the class to which Comparable is typed. In this case, since we want to sort Food instances, you can type Comparable to Food i.e. `Comparable<Food>` and therefore, the method parameter will be `Food` i.e. `public int compareTo(Food f)`.

Problema 4

Which of these interfaces are in the Java Collections framework?

You had to select 4 options

- Set
- Union
  - There is nothing like Union in Java.
- BitSet
  - It has nothing to do with Collections.
- Collection
- Map
- NavigableMap

Explanation:

This is not required for the exam but in case you want to know what a `BitSet` is:

```
public class BitSet extends Object implements Cloneable, Serializable
```

This class implements a vector of bits that grows as needed. Each component of the bit set has a boolean value. The bits of a BitSet are indexed by nonnegative integers. Individual indexed bits can be examined, set, or cleared. One BitSet may be used to modify the contents of another BitSet through logical AND, logical inclusive OR, and logical exclusive OR operations.

By default, all bits in the set initially have the value false.

Every bit set has a current size, which is the number of bits of space currently in use by the bit set. Note that the size is related to the implementation of a bit set, so it may change with implementation. The length of a bit set relates to logical length of a bit set and is defined independently of implementation.

Problema 5

Given:

```
class Game{ }
class Cricket extends Game{ }
class Instrument{ }
class Guitar extends Instrument{ }

interface Player<E>{ void play(E e); }
interface GamePlayer<E extends Game> extends Player<E>{ }
interface MusicPlayer<E extends Instrument> extends Player{ }
```

Identify valid declarations.
You had to select 1 option

```
class Batsman implements GamePlayer<Cricket>{
     public void play(Game o){ }
}
```

Observe the declaration of GamePlayer. It stipulates that it be typed with a class that extends Game (named temporarily as E) and that Player be typed to the exact same type E. Here, GamePlayer is typed to Cricket. Thus, Player is also typed to Cricket. Therefore, public void play(Game o){ } does not satisfy Player interface (because Player now requires a method public void play(Cricket o){ }).

```
class Bowler implements GamePlayer<Guitar>{
    public void play(Guitar o){ }
}
```

GamePlayer declaration stipulates that it be typed to a class that extends Game. Guitar does not extend Game.

```
class Bowler implements Player<Guitar>{
    public void play(Guitar o){ }
}
```

This is valid because Player has no restriction on what it can be typed to. It is only restriction is that the play() method must also use the same type. So play(Instrument) will not be valid here.

```
class MidiPlayer implements MusicPlayer {
    public void play(Guitar g){ }
}
```

Observe that MusicPlayer extends plain untyped Player. This means, MusicPlayer interface gets the abstract method play(Object obj) from Player. Thus, any non-abstract class that implements MusicPlayer must have play(Object) method. Thus, MidiPlayer must have a method play(Object). If MusicPlayer were defined like this: interface MusicPlayer extends Player<Instrument>{ }, then it would have been a typed usage of Player (typed to Instrument) and then it would have been ok for any MusicPlayer to implement play(Instrument).

```
class MidiPlayer implements MusicPlayer<Instrument> {
    public void play(Guitar g){ }
}
```

MidiPlayer must have a method play(Object).

Problema 6

Following is a program to capture words from command line and create two collections. One that keeps only unique words and one that keeps all the words in the order they were entered. What should replace AAA and BBB?

```
static Collection unique = new AAA();
static Collection ordered = new BBB();
public static void main(String args[]) throws Exception
{
   BufferedReader bfr = new BufferedReader( new InputStreamReader( System.in ) );
   String s = bfr.readLine();
   while(s != null && s.length() >0)
   {
      unique.add(s);
      ordered.add(s);
      s = bfr.readLine();
   }
   System.out.println(unique);
   System.out.println(ordered);
}
```

You had to select 2 options

- Set, List
  - Set and List are interfaces. They cannot be instantiated. So you cannot do new Set(); or new List();
- LinkedList, HashSet
- HashSet, LinkedList
- HashSet, ArrayList
- Vector, TreeSet

Explanation:

unique means you need a class that implements `Set`. So options are: `HashSet`, `LinkedHashSet`, and `TreeSet`

ordered means you need a class that implements `List`. So options are: `ArrayList`, `LinkedList`.

Problema 7

What will the following program print?
Specify the words in the order they will be printed

```
import java.util.Arrays;
import java.util.Collections;
import java.util.List;
public class TestClass
{
    public static void main(String[] args)
    {
        String[] names = new String[]{ "bb23", "BB23", "23bB", "b23B", "bB23", "B23b", "23Bb"};

        List<String> list = Arrays.asList(names);
        Collections.sort(list);
        System.out.println(list);

    }
 }
```

23Bb 23bB B23b BB23 b23B bB23 bb23

In the exam, you will be asked to order words that are different only in case. You need to remember a simple rule to answer such questions:

```
spaces < numbers < uppercase < lowercase
```

Problema 8

Which of the following collection implementations are thread-safe?

You had to select 1 option

- ArrayList
- HashSet
- HashMap
- TreeSet
- None of the Above.

Explanation:

Of all the collection classes of the `java.util` package, only `Vector` and `Hashtable` are Thread-safe. `java.util.Collections` class contains a `synchronizedCollection` method that creates thread-safe instances based on collections which are not.

For example:

```
Set s = Collections.synchronizedSet(new HashSet());
```

Problema 9

Given:

```
public class Book{
    String isbn;
    String title;
    public Book(String isbn, String title){
        this.isbn = isbn;
        this.title = title;
    }
    //accessors not shown

   //assume appropriate implementations of equals and hashCode based on isbn property
}

and the following code snippet:

List<Book> books = getBooksByAuthor("Ludlum");
books.stream().sorted().forEach(b->System.out.println(b.getIsbn())); //1
```

Assuming that getBooksByAuthor is a valid method that returns a List of Books, which of the following statements is/are true?

You had to select 1 option

- Compilation failure at //1.
- It will throw an exception at run time due to code at //1.
- It will throw a ClassCastException.
- It will print isbns of the books in sorted fashion.
- It will print isbns of the books but NOT in sorted fashion.

Explanation:

Remember that any time you want to sort a collection of items, you need code that "sorts" the elements. `equals` method can only tell whether two objects are equal or not but it cannot tell which object comes before and which after. To figure this out, either the objects themselves should be able to tell that (by implementing `java.lang.Comparable` interface) or you need to use a separate `java.util.Comparator` instance that can compare the objects.

Here, no `Comparator` is supplied to the `sorted()` method. Therefore, at run time, when the sorted method tries to cast Book object to `Comparable`, it will fail.

Problema 10

You are implementing a special sorting algorithm that can sort objects of different classes. Which of the following class declarations will you use?

You had to select 1 option

- `public class SpecialSorter<>{ ... }`

- `public class SpecialSorter<K>{ ... }`
    - This is the correct way to define a generic class. Within the class, you can use K as a type, for example:

```
public class SpecialSorter<K>{
   public void sort(ArrayList<K> items){
      K item = items.get(0);
      //...
   }
}
```

- `public class <SpecialSorter>{ ... }`

- `public class SpecialSorter(K){ ... }`

Problema 11

Consider the following code :

```
import java.util.*;
class Request { }
class RequestCollector{
    //1 : Insert declaration here

    public synchronized void addRequest(Request r){
    container.add(r);
    }
    public synchronized Request getRequestToProcess(){
    return container.poll();
    }
}
```

What can be inserted at //1?

You had to select 1 option

- `Queue<Request> container = new LinkedList<Request>();`
- `LinkedList container = new LinkedList();`
- `Queue container = new PriorityQueue();`
- `Queue<Request> container = new Queue<Request>();`
    - Queue is an interface so it cannot be instantiated. PriorityQueue is a class that implements Queue.
- `List al = new ArrayList();`
- None of these.

Explanation

Since the return type of `getRequestToProcess()` method is `Request`, and the return statement does not have a cast, it means that container must be a typed generic collection that returns `Request` objects only. Further, the use of `poll()` method indicates that it must be a `Queue`.

Problema 12

Consider the following code:

```
import java.util.*;
public class TestClass {
    public static void main(String[] args)
    {
      // put declaration here

      m.put("1", new ArrayList());    //1
      m.put(1, new Object());    //2
      m.put(1.0, "Hello");     //3
      System.out.println(m);
    }
}
```

How can 'm' be declared such that the above code will compile and run without errors?

You had to select 2 options

- `Map m = new TreeMap();`
    - This will throw an exception at runtime because the keys of a TreeMap must be mutually comparable. Here, String, Integer, and Double are not mutually comparable.

- `Map<Object, Object> m = new TreeMap<Object, Object>();`
    - Same problem as the first option.

- `Map<Object, ?> m = new LinkedHashMap<Object, Object>();`

- `Map<Object, ? super ArrayList> m = new LinkedHashMap<Object, ArrayList>();` will work if lines //2 and //3 are commented out.

- `Map<Object, ? super ArrayList> m = new LinkedHashMap<Object, ArrayList>();` will work if lines //1 and //3 are commented out.

- `Map m = new HashMap();`

Explanation:

There is an important concept to be understood here so please read this carefully:

```
Map<Object, ?> m = new LinkedHashMap<Object, Object>();
```

While this is a valid declaration, it will not allow you to put anything into 'm'. The reason is that m is declared to be of type Map that takes an instance of Object class as a key and instance of 'Unknown class' as value. Therefore, if you try to put an Object, or Integer, or anything, the compiler will not allow it because that 'Unknown' class is not necessarily Object or Integer or any other class. Even though the actual object pointed to by 'm' is of type LinkedHashMap<Object, Object>, the compiler looks only at the

reference type of the variable. Thus, 'm' is read-only. It would have worked if m were declared as `Map<Object, Object> m = ....` Because in this case the compiler *knows* that m can take an instance of Object as value. Instance of Object covers all kind of objects.

```
Map<Object, ? super ArrayList> m = new LinkedHashMap<Object, ArrayList>();
```

You should read it aloud as follows: 'm' is declared to be of type Map that takes an instance of Object class as a key and an instance of 'a class that is either ArrayList or a superclass of Arraylist' as value. This means that the value can be an instance of ArrayList or its subclass (since an ArrayList object or its subclass object can be assigned to a reference of type ArrayList or its super class.). However, you cannot put Object (which is a superclass of ArrayList) in it because the compiler doesn't know the exact superclass that 'm' can take. It could be AbstractList, or Object, or any other super class of ArrayList. The compiler only knows that it is a superclass but not the exact type. So option 4 is correct but 5 is wrong.

Thus, you can do: `m.put(anyObj, new ArrayList());`

ust the opposite of super is extends. Consider the following method:

```
public void m1(List<? extends Number> list)
{
   list.add(new Integer(10));  //Error at compile time because the compiler
                        //only knows that list contains Number or its subclass objects.
But it doesn't know the exact type.
                        //Therefore, it will not allow you to add anything to it.

   Number n = list.get(1);  //This will work because the compiler knows that every
object in list IS-A Number.

}
```

Option 6 is untyped. So it will allow anything to be put into 'm'.

Problema 13

Consider the following code:

```
//Assume appropriate imports
class Person{
    String name;
    String dob;
    public Person(String name, String dob){
        this.name = name; this.dob = dob;
    }
}
public class SortTest {
    public static void main(String[] args) {
        ArrayList<Person> al = new ArrayList<>();
        al.add(new Person("Paul", "01012000"));
        al.add(new Person("Peter", "01011990"));
        al.add(new Person("Patrick", "01012002"));

        //INSERT CODE HERE

        for(Person a : al) System.out.println(a.name+" "+ a.dob);
    }
}
```

What can be inserted in the code so that it will sort the collection of Persons by Person's dob attribute?

You had to select 1 option

```
Collections.sort(al, new Comparable<Person>(){
    public int compare(Person o1, Person o2) {
            return o1.dob.compareTo(o2.dob);
    }
});
```

```
Collections.sort(al, new Comparator<Person>(){
    public int compare(Person o1, Person o2) {
            return o1.dob.compareTo(o2.dob);
    }
});
```

```
Collections.sort(al, new Comparable<Person>(){
    public int compare(Person o1, Person o2) {
            return o1.dob.compare(o2.dob);
    }
});
```

```
Collections.sort(al, new Comparator<Person>(){
    public int compare(Person o1, Person o2) {
            return o1.dob.compare(o2.dob);
    }
});
```

Explanation:

When the class of the objects that you want to compare does not implement `java.lang.Comparable` interface (which has one method named `compareTo(T t)`), or when you want to compare the objects using some other comparison criteria than the one implemented by its compareTo method, you can use `Collections.sort(List, java.util.Comparator)` method. This method allows you to pass your own custom comparator to compare the objects. It has two methods:
`int compare(T o1, T o2)` : Compares its two arguments for order.
`boolean equals(Object obj)` : Indicates whether some other object is "equal to" this comparator.

Problema 14

Given the following class:

```
public class SpecialPicker<K>{
   public K pickOne(K k1, K k2){
     return k1.hashCode()>k2.hashCode()?k1:k2;
   }
}
```

what will be the result of running the following lines of code:

```
  SpecialPicker<Integer> sp = new SpecialPicker<>(); //1
  System.out.println(sp.pickOne(1, 2).intValue()+1); //2
```

You had to select 1 option

- The first line of code will not compile.
- The second line of code will not compile.
- It will throw an exception at run time.
- It will print 3.

Explanation:

This class illustrates how you can create a generic class. This is very simple example and if you have trouble understanding this code, you should read about writing generic classes from a good book.
FYI, Integer's hashCode simply returns the int value contained in the Integer object.

Problema 15

Given : StringBuffer and String implement CharSequence and CharSequence has charAt() method. Complete the following code such that it will compile and run without any errors or warnings. (Do not leave any blanks unfilled. Assume required imports.)

```
public class TestClass {
    public < E extends CharSequence > Collection < E>
                    getWordsStartingWith(Collection < E > input , char ch )  {
        Collection< E > returnValue = new ArrayList <E >();
        int len = input.size();
        for(E e : input){
            if(e.charAt(0) == ch) returnValue.add(e);
        }
        return returnValue;
    }

    public void checkIt()    {
        List<String> a = new ArrayList<String>(); a.add("apple"); a.add("cherry");
        Set<StringBuffer> b = new HashSet<StringBuffer>(); b.add( new StringBuffer("pineapple"));
     Collection < String > ac = getWordsStartingWith(a, 'a');
     Collection < StringBuffer > bc = getWordsStartingWith(b, 'b');
    }
```

extends E ArrayList

super String StringBuffer

CharSequence Collection

Problema 16

Complete the following code by inserting declaration for stateCitiesMap:

```
//Insert line of code here

List<String> cities = new ArrayList<>();
cities.add("New York");
cities.add("Albany");
stateCitiesMap.put("NY", cities);
```

You had to select 2 options

- Map<String, ArrayList<String>> stateCitiesMap = new HashMap<>();
  - Here, you are declaring a map that contains ArrayList as values. But later in the code, you are adding "cities", which of type List.

- Map<String, List<String>> stateCitiesMap = new HashMap<String, List<>>();
  - You cannot embed a diamond operator within another generic class instantiation. Thus, new HashMap<String, List<>> is invalid because of List<>.

- Map<String , ArrayList<String>> stateCitiesMap = new HashMap<<>, List<>>();
  - You cannot embed a diamond operator within another generic class instantiation.

- Map<String, List<String>> stateCitiesMap = new HashMap<String, ArrayList<String>>();
  - ArrayList<String> is not compatible with List<String> in this case. Both the sides must have the same type.

- Map<String, List<String>> stateCitiesMap = new HashMap<>();
  - This is the right way to use the diamond operator.

- Map<String, List<String>> stateCitiesMap = new HashMap<String, List<String>>();
  - This is how you would do it without using the diamond operator.

Problema 17

Given :

```
import java.util.*;
class MyStringComparator implements Comparator
{
    public int compare(Object o1, Object o2)
    {
      int s1 = ((String) o1).length();
      int s2 = ((String) o2).length();
      return s1 - s2;
    }
}
and

static String[] sa = { "d", "bbb", "aaaa" };
```

Select correct statements.
You had to select 2 options

- This is not a valid Comparator implementation.

- Arrays.binarySearch(sa, "cc", new MyStringComparator()); will return -2.
    - Since there is no string of length 2 in sa, nothing in sa matches the string "cc". So the return value has to be negative. Further, if the value "cc" were to be inserted in sa, it would have to be inserted after "d" i.e. at index 1. Thus, the return value will be -(index+1) = -2.

- Arrays.binarySearch(sa, "c", new MyStringComparator()); will return 0.
    - There is only one string of length 1 in sa, and it is at index 0.

- Arrays.binarySearch(sa, "c", new MyStringComparator()); will return -1.

- Arrays.binarySearch(sa, "c", new MyStringComparator()); will throw an exception.

Explanation:

This Comparator returns the difference in length of two strings. Thus, two strings of same length are considered equal by this comparator.

`Arrays.binarySearch()` method returns the index of the search key, if it is contained in the list; otherwise, `(-(insertion point) - 1)`. The insertion point is defined as the point at which the key would be inserted into the list: the index of the first element greater than the key, or `list.size()`, if all elements in the list are less than the specified key. Note that this guarantees that the return value will be >= 0 if and only if the key is found.

Problema 18

Consider the following code :

```
import java.util.*;
class Request { }
class RequestCollector{
    //1 : Insert declaration here

    public synchronized void addRequest(Request r){
    container.add(r);
    }
    public synchronized Request getRequestToProcess(){
    return container.poll();
    }
}
```

What can be inserted at //1?

You had to select 1 option

- Queue<Request> container = new LinkedList<Request>();
- LinkedList container = new LinkedList();
- Queue container = new PriorityQueue();
- Queue<Request> container = new Queue<Request>();
    - Queue is an interface so it cannot be instantiated. PriorityQueue is a class that implements Queue.
- List al = new ArrayList();
- None of these.

Since the return type of `getRequestToProcess()` method is `Request`, and the return statement does not have a cast, it means that container must be a typed generic collection that returns `Request` objects only. Further, the use of `poll()` method indicates that it must be a `Queue`.

Both `LinkedList` and `PriorityQueue` classes implement `Queue` interface.

Problema 19

What will the following class print when compiled and run?

```
import java.util.*;

public class TestClass {

    public static void main(String[] args) {

        TreeSet<Integer> s = new TreeSet<Integer>();
        TreeSet<Integer> subs = new TreeSet<Integer>();

        for(int i = 324; i<=328; i++)
        {
            s.add(i);
        }
        subs = (TreeSet) s.subSet(326, true, 328, true );
        subs.add(329);
        System.out.println(s+" "+subs);

    }

}
```

You had to select 1 option

- [324, 325, 326, 327, 328] [326, 327, 329]
- [324, 325, 326, 327, 329] [326, 327, 329]
- [324, 325, 326, 327, 328] [326, 327]
- It will throw an exception at runtime.
- It will not compile.

Explanation:
`TreeSet` is a `NavigableSet` and so it supports `subSet()` method :

```
NavigableSet<E> subSet(E fromElement, boolean fromInclusive, E toElement,
boolean toInclusive)
```

Returns a view of the portion of this set whose elements range from fromElement to toElement.

The returned subset is backed by the original set. So if you insert or remove an element from the subset, the same will be reflected on the original set.

Further, since the subset is created using a range (fromElement to toElement), the element that you are inserting must fall within that range. Otherwise an IllegalArgumentException is thrown with a message "key out of range.". This is what is happening in this question. The range of subs is 326 to 328 and 329 is out of that range. Therefore, an IllegalArgumentException is thrown at runtime.

Problema 20

What will be the contents of d at the end of the following code?

```
Deque<Integer> d = new ArrayDeque<>();
d.push(1);
d.offerLast(2);
d.push(3);
d.peekFirst();
d.removeLast();
d.pop();
System.out.println(d);
```

You had to select 1 option

- 1
  - push() is a stack method that adds the element to the front. Therefore, the contents of d change as follows: 1
  - offer(e) is a queue method that adds the element to the end and offerLast(e) is equivalent to offer(e). So the deque now contains: 1, 2
  - push(3) changes it to: 3, 1, 2
  - Now, peek methods don't modify the structure, therefore even though peekFirst returns 3, the deque doesn't change. removeLast() removes the element from the end, so d now contains: 3, 1
  - pop() is a stack method that removes the element from the front. Therefore, the contents of d change to: 1
- 2
- 3
- Exception at run time.
- It will not compile.

Problema 21

Given:

```
String[] p = {"1", "2", "3" };
```

Which of the following lines of code is/are valid?

You had to select 1 option

- `List<?> list2 = new ArrayList<Integer>(Arrays.asList(p));`
- `List<Integer> list2 = new ArrayList<Integer>(Arrays.asList(p));`
- `List<Integer> list2 = new ArrayList<>(Arrays.asList(p));`
- `List<?> list2 = new ArrayList<>(Arrays.asList(p));`
    - Here, list2 is a list of any thing. You cannot add any thing to it and you can only retrieve Objects from it:
      list2.add(new Object()); list2.add("aaa"); //both will not compile.
      Object obj = list2.get(0); //Valid
      String str = list2.get(0); //will not compile.
    - Note that you can add null to it though i.e. list2.add(null); is valid.

Problema 22

Given the following complete contents of TestClass.java:

```
import java.util.ArrayList;
import java.util.Collections;

class Address implements Comparable<Address>{
    String street;
    String zip;
    public Address(String street, String zip){
        this.street = street; this.zip = zip;
    }
    public int compareTo(Address o) {
        int x = this.zip.compareTo(o.zip);
        return x == 0? this.street.compareTo(o.street) : x;
    }
}
public class TestClass {
    public static void main(String[] args) {
        ArrayList<Address> al = new ArrayList<>();
        al.add(new Address("dupont dr", "28217"));
        al.add(new Address("sharview cir", "28217"));
        al.add(new Address("yorkmont ridge ln", "11223"));
        Collections.sort(al);
        for(Address a : al) System.out.println(a.street+" "+ a.zip);
    }
}
```

What will be printed?

You have to select 1 option:

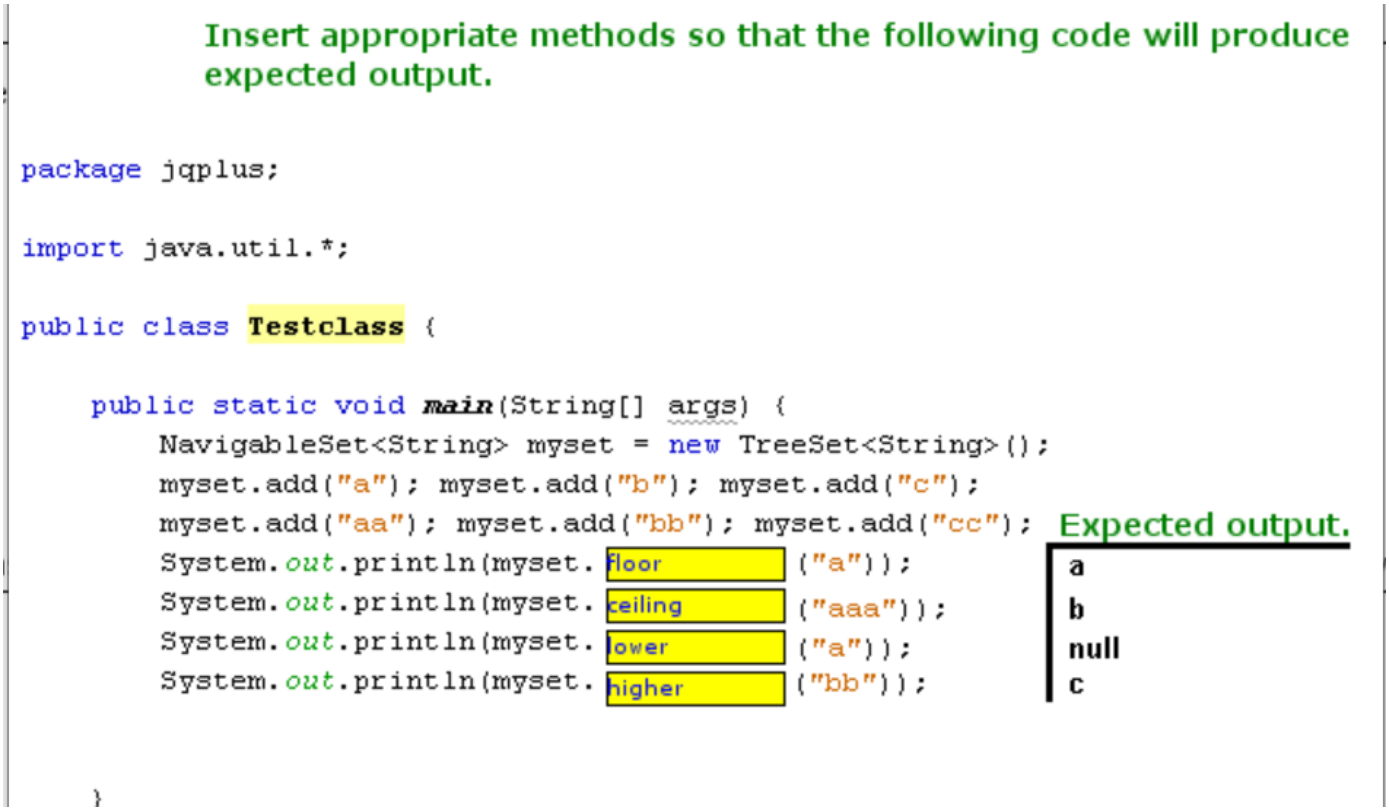
- yorkmont ridge ln 11223
  dupont dr 28217
  sharview cir 28217

- sharview cir 28217
  yorkmont ridge ln 11223
  dupont dr 28217

- sharview cir 28217
  dupont dr 28217
  yorkmont ridge ln 11223

- The order of output messages cannot be determined.
- It will throw an exception at run time.
- It will not compile.

Problema 23



The following are the descriptions of the methods.

E ceiling(E e)
Returns the least element in this set greater than or equal to the given element, or null if there is no such element.

E floor(E e)
Returns the greatest element in this set less than or equal to the given element, or null if there is no such element.

E higher(E e)
Returns the least element in this set strictly greater than the given element, or null if there is no such element.

E lower(E e)
Returns the greatest element in this set strictly less than the given element, or null if there is no such element.

Problema 24

Which of the following interface is suitable for a class that stores associations of keys to values?

You had to select 1 option

- Collection
- SortedSet
- Map
    - The classes HashMap and TreeMap implement Map.
- Set
- None of the above.

Collections API has two separate trees of classes/interfaces - `java.util.Collection` and `java.util.Map`. A Collection (such as a Set or a List) stores objects, while a Map stores key-value pairs.

Problema 25

Which interfaces does java.util.NavigableMap extend directly or indirectly?

You had to select 2 options

- java.util.SortedSet
- java.util.Map
- java.util.SortedMap
- java.util.TreeMap
    - TreeMap is a class that implements NavigableMap interface. ConcurrentSkipListMap is the other such class.
- java.util.List

Problema 26

Given:

```
public class PlaceHolder<K, V> { //1
    private K k;
    private V v;
    public PlaceHolder(K k, V v){ //2
        this.k = k;
        this.v = v;
    }
    public K getK() { return k; }
    public static <X> PlaceHolder<X, X> getDuplicateHolder(X x){ //3
        return new PlaceHolder<X, X>(x, x); //4
    }
}
```

Which line will cause compilation failure?

You had to select 1 option

- 1
- 2
- 3
- 4
- Some other unnumbered line.
- There is no problem with the code.

Explanation:

The given code illustrates how to define a generic class. It has two generic types K and V. The only interesting piece of code here is this method:

```
public static <X> PlaceHolder<X, X> getDuplicateHolder(X x){
   return new PlaceHolder<X, X>(x, x);
}
```

You could write the same method in the following way also:

```
public static <X> PlaceHolder<X, X> getDuplicateHolder(X x){
    return new PlaceHolder<>(x, x); //use diamond operator because we
already know that X is a generic type.
}
```

Problema 27

Identify the correct statements about the following code:

```
import java.util.*;
class Person {
    private String name;
    public Person(String name) {  this.name = name;    }
    public String getName() { return name;  }
    public void setName(String name) { this.name = name; }

    public String toString() { return name; }
}

class Helper {

    public void helpPeople(Queue people, Queue helped) {
        do {
            Person p = (Person) people.poll();
            System.out.println("Helped : " + p + " ");
            helped.offer(p.getName());
        } while (!people.isEmpty());
    }

    public static void main(String[] args) {
        Queue<Person> q = new LinkedList<Person>();
        q.offer(new Person("Pope"));
        q.offer(new Person("John"));
        Queue<Person> helpedQ = new LinkedList<Person>();
        Helper h = new Helper();
        h.helpPeople(q, helpedQ);
    }
}
```

You had to select 2 options

- It will print :
  ```
  Helped : Pope
  Helped : John
  ```
- It will compile with a warning.
- It will throw an exception at runtime.
- It will compile without warning but will throw an exception at runtime.
- It will print :
  ```
  Helped : John
  Helped : Pope
  ```
- It will print : Helped : John
- It will print : Helped : Pope

The helpPeople() method is an old style (legacy) method that takes type-unsafe Queues. At Runtime, JVM does not have any type information for the generic classes. Therefore, this method is free to add any kind of object in the queues without any exception even though the 'helped' Queue was supposed to have only Person objects.

Problema 28

What will the following code print?

```
List<String> vowels = new ArrayList<String>();
vowels.add("a");
vowels.add("e");
vowels.add("i");
vowels.add("o");
vowels.add("u");
Function<List<String>, List<String>> f = list->list.subList(2, 4);
f.apply(vowels);
vowels.forEach(System.out::print);
```

You had to select 1 option

- iou
- io
- ou
- ei
- aeiou
- Compilation error

Explanation:

There is no problem with the code.
The List.subList method returns a view backed by the original list. It doesn't change the existing list. Therefore, when you print the elements from the original list after calling subList, you will see all the elements of the original list.

Remember that, however, if you modify the sub list, the changes will be visible in the original list. For example, the following will print `aeioxu`:

```
List<String> view = f.apply(vowels);//get a view backed by the original list
view.add("x");//modify the view
vowels.forEach(System.out::print); //updates visible in original list
```

Problema 29

What MAY the following program print?
(If you want to leave one or more Yellow targets empty, please make sure that they are at the end.)

```
import java.util.HashSet;


enum SIZE

   TALL, GRANDE, JUMBO;


class CoffeeMug


   public static void main(String[] args)
   {
       HashSet<SIZE> hs = new HashSet<SIZE>();
       hs.add(SIZE.TALL); hs.add(SIZE.JUMBO); hs.add(SIZE.GRANDE);
       hs.add(SIZE.TALL); hs.add(SIZE.TALL); hs.add(SIZE.JUMBO);

       for(SIZE s : hs) System.out.println(s);

   }
```

Drop the blue labels on the right on yellow targets below.
Any label can be used multiple times.

TALL GRANDE JUMBO

GRANDE TALL JUMBO

There are two concepts involved in this question:
1. A Set (such as a HashSet) does not allow duplicate elements. If you add a duplicate element, it is ignored. Thus, only three unique SIZE elements are stored.

It is important to understand how the add() method of a Set works :
boolean add(E o)
Adds the specified element to this set if it is not already present (optional operation). More formally, adds the specified element, o, to this set if this set contains no element e such that (o==null ? e==null : o.equals(e)). If this set already contains the specified element, the call leaves this set unchanged and returns false. In combination with the restriction on constructors, this ensures that sets never contain duplicate elements.

2. The order of elements is not defined in HashSet. So while retrieving elements, it can return them in any order.

Remember that, TreeSet does store elements in their natural sorted order.

Problema 30

Identify the correct statements about the following code:

```
import java.util.*;
class Person {
    private static int count = 0;
    private String id = "0"; private String interest;
    public Person(String interest){ this.interest = interest; this.id = "" + ++count; }
    public String getInterest(){ return interest;     }
    public void setInterest(String interest){ this.interest = interest; }
    public String toString(){ return id; }
}

public class StudyGroup
{
    String name = "MATH";
    TreeSet<Person> set = new TreeSet<Person>();
    public void add(Person p) {
      if(name.equals(p.getInterest())) set.add(p);
    }

    public static void main(String[] args) {
      StudyGroup mathGroup = new StudyGroup();
      mathGroup.add(new Person("MATH"));
      System.out.println("A");
      mathGroup.add(new Person("MATH"));
      System.out.println("B");
      System.out.println(mathGroup.set);
    }
}
```

You had to select 1 option

- It will print : A, B, and then the contents of mathGroup.set.
- It will compile with a warning.
- It will NOT throw an exception at runtime.
- It will compile without warning but will throw an exception at runtime.
- It will only print : A
- It will print : A and B.

Explanation:

Note that `TreeSet` is an ordered set that keeps its elements in a sorted fashion. When you call the `add()` method, it immediately compares the element to be added to the existing elements and puts the new element in its appropriate place. Thus, the foremost requirement of a `TreeSet` is that the elements must either implement `Comparable` interface (which has the `compareTo(Object)` method) and they must also be mutually comparable or the TreeSet must be created with by passing a Comparator (which has a compare(Object, Object)) method). For example, you might have two classes \\\A\\\ and \\\B\\\ both

implementing `Comparable` interface. But if their `compareTo()` method does not work with both the types, you cannot add both type of elements in the same `TreeSet`.

In this question, `Person` class does not implement `Comparable` interface. Ideally, when you add the first element, since there is nothing to compare this element to, there should be no exception. But when you add the second element, `TreeSet` tries to compare it with the existing element, thereby throwing `ClassCastException` because they don't implement `Comparable` interface. However, this behavior was changed in the `TreeSet` implementation recently and it throws a `ClassCastException` when you add the first element itself.

The compiler knows nothing about this requirement of `TreeSet` since it is an application level requirement and not a language level requirement. So the program compiles fine without any warning.

Problema 31

The signature of a method in a class is as follows:

```
public static <E extends CharSequence> List<? super E> doIt(List<E> nums)

This method is being called in the following code:

result = doIt(in);
```

Given that String implements CharSequence interface, what should be the reference type of 'in' and 'result' variables?

You had to select 1 option

- `ArrayList<String> in;`
  `List<CharSequence> result;`

- `List<String> in;`
  `List<Object> result;`

- `ArrayList<String> in;`
  `List result;`

- `List<CharSequence> in;`
  `List<CharSequence> result;`

- `ArrayList<Object> in;`
  `List<CharSequence> result;`

- None of these.

Explanation:

The input parameter has been specified as `List<E>`, where E has to be some class that extends `CharSequence`. So `ArrayList<String>, List<String>`, or `List<CharSequence>` are all valid as reference types for '`in`'.

The output type of the method has been specified as `List<? super E>` , which means that it is a List that contains objects of some class that is a super class of E. Here, E will be typed to whatever is being used for '`in`'. For example, if you declare `ArrayList<String> in`, E will be String.

The important concept here once the method returns, there is no way to know what is the exact class of objects stored in the returned List. So you cannot declare out in a way that ties it to any particular class, not even Object.

Thus, the only way to accomplish this is to either use non-typed reference type, such as: `List result;` or use the same type as the return type mentioned in the method signature i.e. `List<? super String>` (because E will be bound to String in this case.)

Problema 32

Consider the following code:

```
import java.util.*;
public class TestClass {
    public static void main(String[] args)
    {
      // put declaration here

      m.put("1", new ArrayList());    //1
      m.put(1, new Object());    //2
      m.put(1.0, "Hello");     //3
      System.out.println(m);
    }
}
```

How can 'm' be declared such that the above code will compile and run without errors?

You had to select 2 options

- Map m = new TreeMap();
    - This will throw an exception at runtime because the keys of a TreeMap must be mutually comparable. Here, String, Integer, and Double are not mutually comparable.

- Map<Object, Object> m = new TreeMap<Object, Object>();
    - Same problem as the first option.

- Map<Object, ?> m = new LinkedHashMap<Object, Object>();

- Map<Object, ? super ArrayList> m = new LinkedHashMap<Object, ArrayList>(); will work if lines //2 and //3 are commented out.

- Map<Object, ? super ArrayList> m = new LinkedHashMap<Object, ArrayList>(); will work if lines //1 and //3 are commented out.

- Map m = new HashMap();

Explanation:

There is an important concept to be understood here so please read this carefully:

```
Map<Object, ?> m = new LinkedHashMap<Object, Object>();
```

While this is a valid declaration, it will not allow you to put anything into 'm'. The reason is that m is declared to be of type Map that takes an instance of Object class as a key and instance of 'Unknown class' as value. Therefore, if you try to put an Object, or Integer, or anything, the compiler will not allow it because that 'Unknown' class is not necessarily Object or Integer or any other class. Even though the actual object pointed to by 'm' is of type LinkedHashMap<Object, Object>, the compiler looks only at the reference type of the variable. Thus, 'm' is read-only. It would have worked if m were declared as `Map<Object, Object> m = ....` Because in this case the compiler *knows* that m can take an instance of Object as value. Instance of Object covers all kind of objects.

```
Map<Object, ? super ArrayList> m = new LinkedHashMap<Object, ArrayList>();
```

You should read it aloud as follows: 'm' is declared to be of type Map that takes an instance of Object class as a key and an instance of 'a class that is either ArrayList or a superclass of Arraylist' as value. This means that the value can be an instance of ArrayList or its subclass (since an ArrayList object or its subclass object can be assigned to a reference of type ArrayList or its super class.). However, you cannot put Object (which is a superclass of ArrayList) in it because the compiler doesn't know the exact superclass that 'm' can take. It could be AbstractList, or Object, or any other super class of ArrayList. The compiler only knows that it is a superclass but not the exact type. So option 4 is correct but 5 is wrong.

Thus, you can do: `m.put(anyObj, new ArrayList());`

Just the opposite of super is extends. Consider the following method:

```
public void m1(List<? extends Number> list)
{
   list.add(new Integer(10));  //Error at compile time because the compiler
                        //only knows that list contains Number or its
subclass objects. But it doesn't know the exact type.
                        //Therefore, it will not allow you to add anything to
it.

   Number n = list.get(1);  //This will work because the compiler knows that
every object in list IS-A Number.

}
```

Problema 33

Which of the following are valid?

You had to select 2 options

- List<String> list = new List<>();
    - List is an interface. It cannot be instantiated, therefore new List<>() is invalid.
- ArrayList<String> list = new List<>();
- ArrayList<> list = new ArrayList<>();
- List<String> list = new ArrayList<>();
- List<String> list = new ArrayList<>(10);
    - While creating an ArrayList you can pass in an integer argument that specifies the initial size of the ArrayList. This doesn't actually insert any element into the list. So list.size() would still return 0.

Problema 34

Given:

```
Map<String , List<? extends CharSequence>> stateCitiesMap = new HashMap<String, List<? extends CharSequence>>();
```

Which of the following options correctly achieves the same declaration using type inference?

You had to select 1 option

- `Map<String , List<? extends CharSequence>> stateCitiesMap = new HashMap<String, List<>>();`
    - This will not compile because you can't use a diamond operator within a generic type specification i.e. <String, List<>> is wrong because of List<> being inside of another < and >.

- `Map<String , List<? extends CharSequence>> stateCitiesMap = new HashMap();`
    - While this is valid code but this does not take advantage of type inferencing and will generate a warning at compile time. To use type inferencing, the diamond operator must be used.

- `Map<String , List<? extends CharSequence>> stateCitiesMap = new HashMap<>();`
    - This option use the diamond operators correctly to indicate the type of objects stored in the HashMap to the compiler. The compiler inferences the type of the objects stored in the map using this information and uses it to prevent the code from adding objects of another type.

- `Map<String , List<? extends CharSequence>> stateCitiesMap = new HashMap<<>>();`
    - This will not compile because new HashMap<<>>() is uses invalid syntax. You cannot use a blank <> inside another <>.

Problema 35

You are designing a class that will cache objects. It should be able to store and retrieve an object when supplied with an object identifier.
Further, this class should work by tracking the "last accessed times" of the objects. Thus, if its capacity is full, it should remove only the object that hasn't been accessed the longest.

Which collection class would you use to store the objects?

You had to select 1 option

- HashSet
- ArrayList
- LinkedHashMap
- LinkedList
- TreeMap

Explanation
The LinkedHashMap class maintains the elements in the order of their insertion time. This property can be used to build the required cache as follows:
1. Insert the key-value pairs as you do normally where key will be the object identifier and value will be the object to be cached.
2. When a key is requested, remove it from the LinkedHashMap and then insert it again. This will make sure that this pair marked as inserted latest.
3. If the capacity is full, remove the first element.

Note that you cannot simply insert the key-value again (without first removing it) because a reinsertion operation does not affect the position of the pair.

Problema 36

What will the following code print when run?

```
Deque<Integer> d = new ArrayDeque<>();
d.push(1);
d.push(2);
d.push(3);
System.out.println(d.pollFirst());
System.out.println(d.poll());
System.out.println(d.pollLast());
```

You had to select 1 option

- 1
  2
  3

- 3
  2
  1

    - If you use push, objects will be added in the front and if you then use remove, items will be removed from the front. In this question, you push 1, 2, and 3, that means, the Deque now contains 3, 2, 1. Now, pollFirst() will return 3, poll() will return 2, and pollLast() will return 1. Hence, it will print 3, 2, 1.

- Compilation error

- Exception at run time

Explanation:

`Deque` is an important interface for the exam. To answer the questions, you must remember that a `Deque` can act as a `Queue` as well as a `Stack`. Based on this fact, you can deduce the following:

1. Since Queue is a FIFO structure (First In First Out i.e. add to the end and remove from the front), it has methods `offer(e)/add(e)` (for adding an element to the end or tail) and `poll()/remove()` (for removing an element from the front or head) for this purpose. Note that offer and add are similar while poll and remove are similar.

2. Since Stack is a LIFO structure (Last In First Out i.e. add to the front and remove from the front), it provides methods `push(e)` and `pop()` for this purpose, where push adds to the front and pop removes from the front.

Besides the above methods, `Deque` also has variations of the above methods. But it is easy to figure out what they do:

`pollFirst()/pollLast()` - poll is a Queue method. Therefore `pollFirst` and `pollLast` will remove elements from the front and from the end respectively.
`removeFirst()/removeLast()` - These are Deque specific methods. They will remove elements from the front and from the end respectively. These methods differ from `pollFirst/pollLast` only in that they throw an exception if this deque is empty.

`offerFirst(e)/offerLast(e)` - offer is a Queue method. Therefore `offerFirst` and `offerLast` will add elements to the front and to the end respectively.
`addFirst(e)/addLast(e)` - add is a Queue method. Therefore `addFirst` and `addLast` will add elements to the front and to the end respectively.

`peek(), peekFirst()`: return the first element from the front of the queue but does not remove it from the queue.
`peekLast()` : returns the last element from the end of the queue but does not remove it from the queue.
`element()`: retrieves, but does not remove, the head of the queue represented by this deque (in other words, the first element of this deque). This method differs from peek only in that it throws an exception if this deque is empty.

Notice that there is no `pushFirst(e)` and `pushLast(e)`.

You may wonder why there are multiple methods for the same thing such as `offer(e)` and `add(e)`. Well, they are not exactly same. `add(e)` throws an exception if the element cannot be added to the queue because of lack of capacity, while `offer(e)` does not.

Problema 37

Given :

```
import java.util.*;
class MyStringComparator implements Comparator
{
    public int compare(Object o1, Object o2)
    {
      int s1 = ((String) o1).length();
      int s2 = ((String) o2).length();
      return s1 - s2;
    }
}
```

and

```
static String[] sa = { "d", "bbb", "aaaa" };
```

Select correct statements.

You had to select 2 options

- This is not a valid Comparator implementation.
- Arrays.binarySearch(sa, "cc", new MyStringComparator()); will return -2.
    - Since there is no string of length 2 in sa, nothing in sa matches the string "cc". So the return value has to be negative. Further, if the value "cc" were to be inserted in sa, it would have to be inserted after "d" i.e. at index 1. Thus, the return value will be -(index+1) = -2.
- Arrays.binarySearch(sa, "c", new MyStringComparator()); will return 0.
    - There is only one string of length 1 in sa, and it is at index 0.
- Arrays.binarySearch(sa, "c", new MyStringComparator()); will return -1.
- Arrays.binarySearch(sa, "c", new MyStringComparator()); will throw an exception.

Explanation:

This Comparator returns the difference in length of two strings. Thus, two strings of same length are considered equal by this comparator.

`Arrays.binarySearch()` method returns the index of the search key, if it is contained in the list; otherwise, `(-(insertion point) - 1)`. The insertion point is defined as the point at which the key would be inserted into the list: the index of the first element greater than the key, or `list.size()`, if all elements in the list are less than the specified key. Note that this guarantees that the return value will be >= 0 if and only if the key is found.

Problema 38

Your application requires to store name value pairs such that the order of entries returned while iterating through the structure is deterministic. In other words, if you iterate over the same structure twice, the order of elements returned in both the iterations will be the same. Which of the following classes would you use? You had to select 2 options

- HashMap
  - HashMap does not guarantee to return the elements in a pre-determined order while iterating.

- LinkedHashSet
  - It is a Set and not a Map so it cannot store Name-Value pairs.

- Hashtable
  - Hashtable does not guarantee to return the elements in a pre-determined order while iterating.

- LinkedHashMap
  - It is a linked list implementation of the Map interface, with predictable iteration order. This implementation differs from HashMap in that it maintains a doubly-linked list running through all of its entries. This linked list defines the iteration ordering, which is normally the order in which keys were inserted into the map (insertion-order). Note that insertion order is not affected if a key is re-inserted into the map.

- TreeMap
  - It always returns the entries in sorted order.

Problema 39

Given:

```
class Person{
    String name;
    String dob;
    public Person(String name, String dob){
        this.name = name; this.dob = dob;
    }
}
class MySorter {
    public int compare(Person p1, Person p2){
        return p1.dob.compareTo(p2.dob);
    }
}
public class SortTest {
    public static int diff(Person p1, Person p2){
        return p1.dob.compareTo(p2.dob);
    }

    public static int diff(Date d1, Date d2){
        return d1.compareTo(d2);
    }
    public static void main(String[] args) {
        ArrayList<Person> al = new ArrayList<>();
        al.add(new Person("Paul", "01012000"));
        al.add(new Person("Peter", "01011990"));
        al.add(new Person("Patrick", "01012002"));

         INSERT CODE HERE
    }
}

and the following lines of code:
I     java.util.Collections.sort(al, (p1, p2)->p1.dob.compareTo(p2.dob));

II    java.util.Collections.sort(al, SortTest::diff);

III   java.util.Collections.sort(al, new MySorter()::compare);

IV    java.util.Arrays.sort(al, SortTest::diff);
```

How many of the above lines can be inserted into the given code, independent of each other, to sort the list referred to by al?

You had to select 1 option

- 1
- 2
- 3
- 4
- None of these

Problema 40

Consider the following program:

```
import java.util.*;
public class TestClass
{
    static String[] sa = { "a", "aa", "aaa", "aaaa" };
    static
    {
        Arrays.sort(sa);
    }
    public static void main(String[] args)
    {
        String search = "";
        if(args.length != 0) search = args[0];
        System.out.println(Arrays.binarySearch(sa, search));
    }
}
```

What are all possible values this program can print when run?

You had to select 1 option

- Any number from -5 to 5.
- Any number from -5 to 5 except -1.
- Any number from -5 to 3.
- Any number from -5 to 3 except -1.
- One of a, aa, aaa, or aaaa.
- One of a, aa, aaa, aaaa, or null.

Explanation:
It is important for you to understand what binarySearch(...) method returns:

public static int binarySearch(Object[] a, Object key)

Searches the specified array for the specified object using the binary search algorithm. The array must be sorted into ascending order according to the natural ordering of its elements (as by Sort(Object[]), above) prior to making this call. If it is not sorted, the results are undefined. (If the array contains elements that are not mutually comparable (for example,strings and integers), it cannot be sorted according to the natural order of its elements, hence results are undefined.) If the array contains multiple elements equal to the specified object, there is no guarantee which one will be found.

Parameters:

a - the array to be searched.

key - the value to be searched for.

Returns:

iindex of the search key, if it is contained in the array; otherwise, (-(insertion point) - 1). The insertion point is defined as the point at which the key would be inserted into the array: the index of the

first element greater than the key, or a.length if all elements in the array are less than the specified key. Note that this guarantees that the return value will be >= 0 if and only if the key is found.

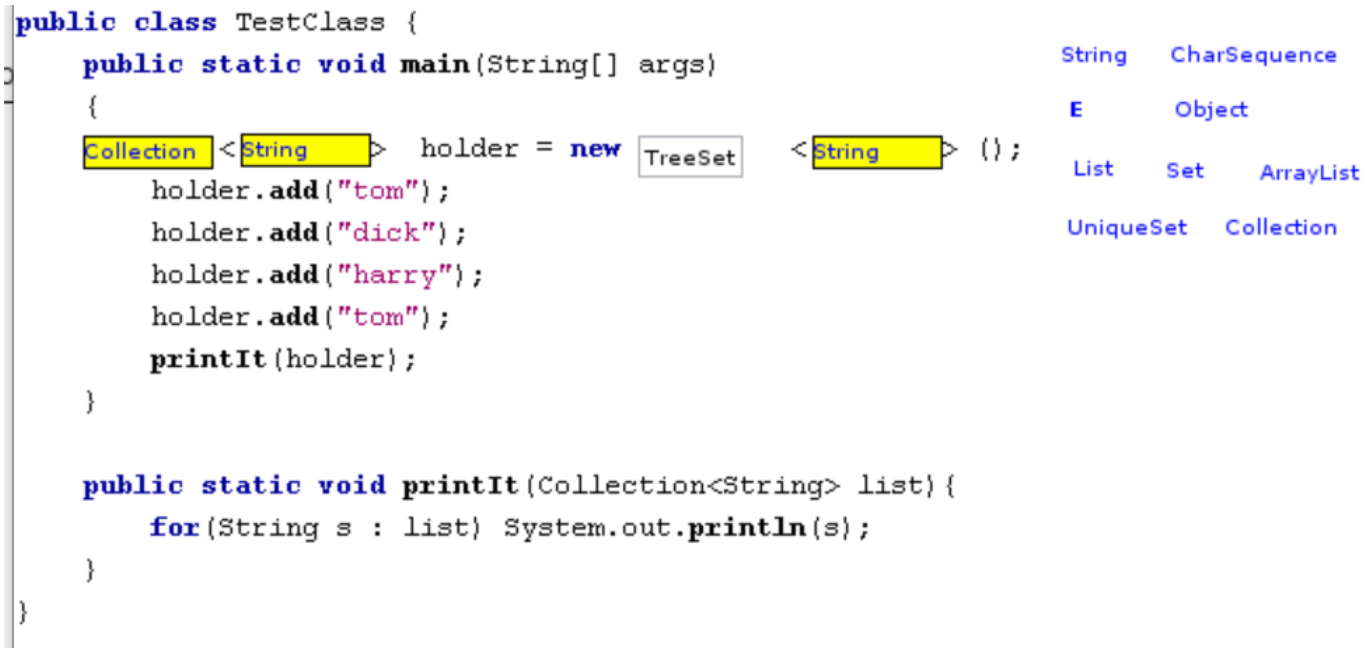
Throws:

ClassCastException - if the search key is not comparable to the elements of the array.

Problema 41

Complete the following code so that it will print dick, harry, and tom in that order

Complete the following code so that it will print dick, harry, and tom in that order. To complete this code you need to type a value in the textbox as well as drag the blue items shown on the right onto the yellow targets. (Assume required imports. Fill all blanks and targets.)



Explanation:

The output is expected to contain unique items. This implies that you need to use a Set. The output is also expected to be sorted. Thus, `TreeSet` is the only option.

The `printIt()` method expects a Collection of Strings. Therefore, the reference type of holder can be `Collection<String>` or any subclass of `Collection<String>` such as `Set<String>`. It cannot be `List` or `ArrayList` because the object on the right hand side is `TreeSet`.

Problema 42

Consider the following code:

```
package jqplus;
import java.util.*;

public class Testclass {

    public static void main(String[] args) {

        NavigableMap<String, String> mymap = new TreeMap<String, String>();
        mymap.put("a", "apple"); mymap.put("b", "boy"); mymap.put("c", "cat");
        mymap.put("aa", "apple1"); mymap.put("bb", "boy1"); mymap.put("cc", "cat1");

        mymap.pollLastEntry(); //LINE 1
        mymap.pollFirstEntry(); //LINE 2

        NavigableMap<String, String> tailmap = mymap.tailMap("bb", false); //LINE 3

        System.out.println(tailmap.pollFirstEntry()); //LINE 4
        System.out.println(mymap.size()); //LINE 5

    }
}
```

What will be returned by the call to tailmap.pollFirstEntry() at //LINE 4 and mymap.size() at //LINE 5?

You had to select 1 option

- A ConcurrentModificationException will be thrown at //LINE 1
- It will return c - cat and 3
- It will return c - cat and 4
- It will return bb - boy1 and 4
- It will return null and 4

Explanation:
Map.Entry<K,V> pollFirstEntry()
Removes and returns a key-value mapping associated with the least key in this map, or null if the map is empty.
Map.Entry<K,V> pollLastEntry()
Removes and returns a key-value mapping associated with the greatest key in this map, or null if the map is empty.

Note that the poll methods remove the entry from the map. Thus, //LINE 2 and //LINE 1 remove a - apple and cc - cat1 entries respectively.

NavigableMap<K,V> tailMap(K fromKey, boolean inclusive)

Returns a view of the portion of this map whose keys are greater than (or equal to, if inclusive is true) fromKey.

Thus, //LINE 3 returns a NavigableMap that contains all the elements above bb (not including bb itself because of the second parameter), which means this new map contains only c - cat.

//LINE 4 removes and returns the first entry (which is also the only entry in this case), which is c - cat.