Chapter 13
Generics and Collections

THE OCP EXAM TOPICS COVERED IN THIS PRACTICE TEST INCLUDE THE FOLLOWING:

- ✓**Generics and Collections**
- Create and use a generic class
- Create and use ArrayList, TreeSet, TreeMap, and ArrayDeque objects
- Use java.util.Comparator and java.lang.Comparable interfaces
- Collections Streams and Filters
- Iterate using forEach methods of Streams and List
- Describe Stream interface and Stream pipeline
- Filter a collection by using lambda expressions
- Use method references with Streams

1. Which of the following can fill in the blank to make the code compile?

```
public class News<_______> {}
```

1. `?`
2. `News`
3. `Object`

1. None of them
2. I
3. II and III
4. I, II, and III

2. Which method is available on both `List` and `Stream` implementations?

1. `filter()`
2. `forEach()`
3. `replace()`
4. `sort()`

3. We are running a library. Patrons select books by name. They get at the back of the checkout line. When they get to the front, they scan the book's ISBN. The checkout system finds the book based on this number and marks the book as checked out. Of these choices, which data structures best represent the line to check out the book and the book lookup to mark it as checked out, respectively?

1. ArrayDeque, TreeMap

3. `ArrayList`, `TreeMap`
4. `ArrayList`, `TreeSet`

4. Which cannot fill in the blank for this code to compile?

```
Collection<String> c = new ____________<>();
c.add("pen");
c.remove("pen");
System.out.println(c.isEmpty());
```

1. `ArrayDeque`
2. `TreeMap`
3. `TreeSet`
4. All of these can fill in the blank.

5. Suppose we want to implement a `Comparator<String>` so that it sorts the longest strings first. You may assume there are no nulls. Which method could implement such a comparator?

1.
```
public int compare(String s1, String s2) {
return s1.length() - s2.length();
}
```
2.
```
public int compare(String s1, String s2) {
return s2.length() – s1.length();
}
```
3.
```
public int compare(Object obj1, object obj2) {
String s1 = (String) obj1;
String s2 = (String) obj2;
return s1.length() - s2.length();
}
```
4.
```
public int compare(Object obj1, object obj2) {
String s1 = (String) obj1;
String s2 = (String) obj2;
return s2.length() – s1.length();
}
```

6. Suppose we want to store `JellyBean` objects. Which of the following pairs require `JellyBean` to implement the `Comparable` interface or create a `Comparator` in order to add them to the `Collection`?

1. `ArrayList` and `ArrayDeque`
2. `HashMap` and `HashSet`
3. `HashMap` and `TreeMap`
4. `TreeMap` and `TreeSet`

7. What is a common reason for a stream pipeline not to run?

1. The source doesn't generate any items.
2. There are no intermediate operations.
3. The terminal operation is missing.
4. None of the above

8. We want this code to print the titles of each book twice. Why doesn't it?

```
LinkedList<String> list = new LinkedList<>();
list.add("Grapes of Wrath");
list.add("1984");

list.forEach(System.out::println);

Iterator it = list.iterator();
while (it.hasMore())
  System.out.println(it.next());
```

   1. The generic type of `Iterator` is missing.
   2. The `hasMore()` method should be changed to `hasNext()`.
   3. The iteration code needs to be moved before the `forEach()` since the stream is used up.
   4. None of the above. The code does print each book title twice.

9. What is the result of the following?

```
ArrayList<Integer> list = new ArrayList<>();
list.add(56);
list.add(56);
list.add(3);

TreeSet<Integer> set = new TreeSet<>(list);
System.out.print(set.size());
System.out.print(" ");
System.out.print(set.iterator().next());
```

   1. `2 3`
   2. `2 56`
   3. `3 3`
   4. `3 56`

10. What best describes a reduction?

   1. An intermediate operation where it filters the stream it receives
   2. An intermediate operation where it mathematically divides each element in the stream
   3. A terminal operation where a single value is generated by reading each element in the prior step in a stream pipeline
   4. A terminal operation where one element is returned from the prior step in a stream pipeline without reading all the elements

11. What is the output of the following?

```
5:   ArrayDeque<Integer> d = new ArrayDeque<>();
6:   d.offer(18);
7:   d.offer(5);
8:   d.push(13);
9:   System.out.println(d.poll() + " " + d.poll());
```

   1. `13 18`
   2. `18 5`
   3. `18 13`
   4. None of the above

12. What is the output of the following?

```
class Magazine {
   private String name;
   public Magazine(String name) {
      this.name = name;
   }
   public int compareTo(Magazine m) {
      return name.compareTo(m.name);
   }
   public String toString() {
      return name;
   }
}
public class Newstand {
   public static void main(String[] args) {
      Set<Magazine> set = new TreeSet<>();
      set.add(new Magazine("highlights"));
      set.add(new Magazine("Newsweek"));
      set.add(new Magazine("highlights"));
```

```
        System.out.println(set.iterator().next());
    }
}
```

1. `highlights`
2. `Newsweek`
3. The code does not compile.
4. The code compiles but throws an exception at runtime.

13. What is the result of the following?

```
6:   List<String> list = new ArrayList<>();
7:   list.add("Monday");
8:   list.add(String::new);
9:   list.add("Tuesday");
10:  list.remove(0);
11:  System.out.println(list.get(0));
```

1. An empty String
2. `Monday`
3. The code does not compile.
4. The code compiles but throws an exception at runtime.

14. How many lines does this code output?

```
List<String> list = new LinkedList<>();
list.add("Archie");
list.add("X-Men");

list.stream().forEach(s -> System.out.println(s));
list.stream().forEach(s -> System.out.println(s));
```

1. Two
2. Four
3. The code does not compile.
4. The code compiles but throws an exception at runtime.

15. Which line in the `main()` method doesn't compile or points to a class that doesn't compile?

```
1:   interface Comic<C> {
2:      void draw(C c);
3:   }
4:   class ComicClass<C> implements Comic<C> {
5:      public void draw(C c) {
6:         System.out.println(c);
7:      }
8:   }
9:   class SnoopyClass implements Comic<Snoopy> {
10:     public void draw(Snoopy c) {
11:        System.out.println(c);
12:     }
13:  }
14:  class SnoopyComic implements Comic<Snoopy> {
15:     public void draw(C c) {
16:        System.out.println(c);
17:     }
18:  }
19:  public class Snoopy {
20:     public static void main(String[] args) {
21:       Comic<Snoopy> c1 = c -> System.out.println(c);
22:       Comic<Snoopy> c2 = new ComicClass<>();
23:       Comic<Snoopy> c3 = new SnoopyClass();
24:       Comic<Snoopy> c4 = new SnoopyComic();
25:     }
26:  }
```

1. Line 21
2. Line 22
3. Line 23
4. Line 24

16. What is the output of the following?

```
Stream<String> s = Stream.of("Atlanta", "Chicago", "New York");
long count = s.filter(c -> c.startsWith("C")).count();
System.out.print(count);
```

1. 1

2. 2
3. The code does not compile.
4. The code compiles but throws an exception at runtime.

17. Fill in the blank to make this code compile:

```
public class Truck implements Comparable<Truck> {
   private int id;
   public Truck(int id) {
      this.id = id;
   }
   @Override
      _____ {
      return id - t.id;
   }
}
```

1. `public int compare(Truck t)`
2. `public int compare(Truck t1, Truck t2)`
3. `public int compareTo(Truck t)`
4. `public int compareTo(Truck t1, Truck t2)`

18. In a stream pipeline, which can return a value other than a `Stream`?

1. Source
2. Intermediate operation
3. Terminal operation
4. None of the above

19. Rewrite this lambda using a constructor reference:

```
n -> new ArrayList<>(n)
```

1. `ArrayList::new;`
2. `ArrayList::new();`
3. `ArrayList::new(n);`
4. `ArrayList::new[n];`

20. What is the result of the following?

```
Comparator<Integer> c = (x, y) -> y-x;
List<Integer> ints = Arrays.asList(3, 1, 4);
Collections.sort(ints, c);
System.out.println(Collections.binarySearch(ints, 1));
```

1. `0`
2. `1`
3. The code does not compile.
4. The result is not defined.

21. How many lines does this code output?

```
List<String> list = new LinkedList<>();
list.add("Archie");
list.add("X-Men");

Stream<String> s = list.stream();
s.forEach(System.out::println);
s.forEach(System.out::println);
```

1. Two
2. Four
3. The code does not compile.
4. The code compiles but throws an exception at runtime.

22. Which option cannot fill in the blank to print `Clean socks`?

```
class Wash<T> {
   T item;
   public void clean(T item) {
      System.out.println("Clean " + item);
```

```
    }
}
public class LaundryTime {
    public static void main(String[] args) {
        _____________________
        wash.clean("socks");
    }
}
```

1. `Wash wash = new Wash();`
2. `Wash wash = new Wash<String>();`
3. `Wash<String> wash = new Wash<>();`
4. All three can fill in the blank.

23. We want this code to print the titles of each book twice. Why doesn't it?

```
LinkedList<String> list = new LinkedList<>();
list.add("Grapes of Wrath");
list.add("1984");

list.stream().forEach(System.out::println);

Iterator it = list.iterator();
while (it.hasNext())
   System.out.println(it.next());
```

1. The generic type of `Iterator` is missing.
2. The `hasNext()` method should be changed to `isNext()`.
3. The iteration code needs to be moved before the `forEach()` since the stream is used up.
4. None of the above. The code does print each book title twice.

24. Rewrite this lambda using a method reference:

```
() -> Math.random()
```

1. `Math.random`
2. `Math::random`
3. `Math::random()`
4. None of the above

25. Which operation can occur more than once in a stream pipeline?

1. Source
2. Intermediate operation
3. Terminal operation
4. None of the above

26. Which type allows inserting a `null` value?

1. `ArrayDeque`
2. `ArrayList`
3. `TreeSet`
4. All of these allow nulls.

27. Fill in the blank so this code outputs three lines:

```
List<String> list = new ArrayList<>();
list.add("Atlanta");
list.add("Chicago");
list.add("New York");

list.stream().filter(_____________).forEach(System.out::println);
```

1. `String::isEmpty`
2. `!String::isEmpty`
3. `String::! isEmpty`

4. None of the above

28. What is the output of the following?

```
TreeMap<String, Integer> map = new TreeMap<>();
map.put("3", 3);
map.put("three", 3);
map.put("THREE", 3);
System.out.println(map.firstKey() + " " + map.lastKey());
```

1. `3 three`
2. `3 THREE`
3. `three 3`
4. `THREE 3`

29. Which fills in the blank in the method signature to allow this code to compile?

```
import java.util.*;
public class ExtendingGenerics {
   private static <_____________ , U> U add(T list, U element) {
      list.add(element);
      return element;
   }
   public static void main(String[] args) {
      List<String> values = new ArrayList<>();
      add(values, "duck");
      add(values, "duck");
      add(values, "goose");
      System.out.println(values);
   }
}
```

1. `? extends Collection<U>`
2. `? implements Collection<U>`
3. `T extends Collection<U>`
4. `T implements Collection<U>`

30. What is the result of the following?

```
List<String> list = new ArrayList<>();
list.add("Austin");
list.add("Boston");
list.add("San Francisco");

list.removeIf(a -> a.length() > 10);
System.out.println(list.size());
```

1. 1
2. 2
3. 3
4. None of the above

31. What does the following output?

```
ArrayDeque<Integer> dice = new ArrayDeque<>();
dice.offer(3);
dice.offer(2);
dice.offer(4);
System.out.print(dice.stream().filter(n -> n != 4));
```

1. 2
2. 3
3. The code does not compile.
4. None of the above

32. Which of the following cannot fill in the blank to make the code compile?

```
private void output(____________<?> x) {
   x.forEach(System.out::println);
}
```

1. `ArrayDeque`
2. `Collection`
3. `TreeMap`
4. None of the above

33. How many lines does this code output?

```
List<String> list = new LinkedList<>();
list.add("Archie");
list.add("X-Men");

list.stream().forEach(System.out.println);
list.stream().forEach(System.out.println);
```

1. Two
2. Four
3. The code does not compile.
4. The code compiles but throws an exception at runtime.

34. What is the output of the following?

```
class Magazine implements Comparable<Magazine> {
   private String name;
   public Magazine(String name) {
      this.name = name;
   }
   @Override
   public int compareTo(Magazine m) {
      return name.compareTo(m.name);
   }
   @Override
   public String toString() {
      return name;
   }
}
public class Newstand {
   public static void main(String[] args) {
      Set<Magazine> set = new TreeSet<>();
      set.add(new Magazine("highlights"));
      set.add(new Magazine("Newsweek"));
      set.add(new Magazine("highlights"));
      System.out.println(set.iterator().next());
   }
}
```

1. `highlights`
2. `Newsweek`
3. The code does not compile.
4. The code compiles but throws an exception at runtime.

35. How many lines does the following code output?

```
import java.util.*;
class Blankie {
   String color;
   String getColor() {
      return color;
   }
}
public class PreSchool {
   public static void main(String[] args) {
      Blankie b1 = new Blankie();
      Blankie b2 = new Blankie();
      b1.color = "pink";
      List<Blankie> list = Arrays.asList(b1, b2);
      list.stream().filter(Blankie::getColor).forEach(System.out::p
   }
}
```

1. One
2. Two
3. The code does not compile.
4. The code compiles but throws an exception at runtime.

36. Which statement about a source in a `Stream` is true?

1. The source is mandatory in a stream pipeline.
2. The source is only allowed to return primitives.

3. The source must be retrieved by calling the `stream()` method.
4. The source must return a finite number of elements.

37. What does the following output?

```
List<String> list = new ArrayList<>();
list.add("Austin");
list.add("Boston");
list.add("San Francisco");

long c = list.stream().filter(a -> a.length() > 10).count();
System.out.println(c + " " + list.size());
```

1. `1 1`
2. `1 3`
3. `2 3`
4. None of the above

38. Which options can fill in the blanks to print `Cleaned 2 items`?

```
import java.util.*;
class Wash<T  ____________ Collection> {
   T item;
   public void clean(T items) {
      System.out.println("Cleaned " + items.size() + " items");
   }
}
public class LaundryTime {
   public static void main(String[] args) {
      Wash<List> wash = new ____________
      wash.clean(Arrays.asList("sock", "tie"));   }
}
```

1. `extends, Wash<ArrayList>();`
2. `extends, Wash<List>();`
3. `super, Wash<ArrayList>();`
4. `super, Wash<List>();`

39. Which of the following declares a `Comparator` where all objects are treated as equal?

1. `Comparator<Character> comp = (c1)-> 0;`
2. `Comparator<Character> comp = (c1)-> {0};`
3. `Comparator<Character> comp = (c1, c2)-> 0;`
4. `Comparator<Character> comp = (c1, c2)-> {0};`

40. Why can't `String::charAt` be used as a method reference?

1. Method references can only be used on `static` methods.
2. Method references can pass either the instance or the parameter from the lambda, but not both.
3. The `charAt()` method takes an `int` rather than `Integer` parameter.
4. There is no `charAt()` method in the `String` class.

PREV
Chapter 12 Advanced Java Class Design

NEXT
Chapter 14 Lambda Built-in Functional Interfaces