Correction Generics and Collections - 0

C - Some text containing @ symbol

Type parameter should not be a Java keyword & a valid Java identifier. Naming convention for Type parameter is to use uppercase single character.

In class Printer, ‘String’ is a valid Java identifier and hence a valid type parameter even though it doesn’t follow the naming convention of uppercase single character.

Printer obj = new Printer<>(100); => Type argument is Integer and it correctly creates an instance of Printer class passing Integer object 100.

Printer class doesn’t override toString() method and hence ‘System.out.println(obj);’ prints some text containing @ symbol.

Correction Generics and Collections - 1

A - Runtime exception

forEach, count, toArray, reduce, collect, findFirst, findAny, anyMatch, allMatch, sum, min, max, average etc. are considered as terminal operations.

Once the terminal operation is complete, all the elements of the stream are considered as used.

Any attempt to use the stream again causes IllegalStateException.

In this example, count() is used after using forEach() method and hence IllegalStateException is thrown.

Correction Generics and Collections - 2

D - T

T is a valid identifier in Java, hence can be used as class name. toString() method has been correctly overridden by class T.

No issues with class T.

Type parameter should not be a Java keyword and naming convention for Type parameter is to use uppercase single character.

class Printer correctly uses type parameter, T.

When using Generic types in the code, you need to specify the type argument. In Test class, ‘Printer obj’ => T refers to class T and not type parameter, T.

System.out.println(obj); => Prints ‘T’ on to the console.

Correction Generics and Collections - 3

A - {11=Sri Nagar, 25=Pune}

TreeMap is sorted map based on the natural ordering of keys. So, map has entries: {11=Sri Nagar, 25=Pune, 32=Mumbai, 39=Chennai}.

headMap(K toKey, boolean inclusive) => returns the map till toKey, if inclusive is true. Hence the output is: {11=Sri Nagar, 25=Pune}.

For the exam, you should know some of the methods from NavigableMap map. Below are the method calls and outputs for the map object used in this example:

//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.

System.out.println(map.tailMap(25, true)); //{25=Pune, 32=Mumbai, 39=Chennai}

//Map.Entry<K,V> firstEntry(); => Returns a key-value mapping associated with the least key in this map.

System.out.println(map.firstEntry()); //11=Sri Nagar

//Map.Entry<K,V> lastEntry(); => Returns a key-value mapping associated with the greatest key in this map.

System.out.println(map.lastEntry()); //39=Chennai

//NavigableMap<K,V> descendingMap(); => Returns a reverse order view of the mappings contained in this map.

System.out.println(map.descendingMap()); //{39=Chennai, 32=Mumbai, 25=Pune, 11=Sri Nagar}

//K floorKey(K key); => Returns the greatest key less than or equal to the given key.

System.out.println(map.floorKey(30)); //25

//K ceilingKey(K key); => Returns the least key greater than or equal to the given key.

System.out.println(map.ceilingKey(30)); //32

Correction Generics and Collections - 4

A -

```
Tokyo
Singapore
Seoul
Paris
London
Hong Kong
```

Arrays class has overloaded stream(...) method to convert arrays to Stream.

sorted(...) method of Stream interface is also overloaded:

sorted() => sorts on natural order

and

sorted(Comparator) => sorts on passed Comparator.

sorted((s1,s2) -> s2.compareTo(s1)) => sorts on descending order.

forEach(System.out::println); => Prints all the Stream data.

Correction Generics and Collections - 5

A -

```
interface Operation {
  long operate(long x, long y);
}
```

B -

```
interface Operation {
  int operate(int x, int y);
}
```

C -

```
interface Operation<T extends Integer> {
  T operate(T x, T y);
}
```

From the given syntax inside main method, it is clear that interface name is Operation and it has an abstract method operate which accepts 2 parameters of numeric type and returns the numeric result (as result of adding 5 and 10 is 15).

So, int and long versions can be easily applied here.

Operation will not work here as inside main method, raw type is used, which means x and y will be of Object type and x + y will cause compilation error as + operator is not defined when both the operands are of Object type.

For Operation, even though main method uses raw type, but x and y will be of Integer type and hence x + y will not cause any compilation error.

Correction Generics and Collections - 6

A - NullPointerException is thrown at runtime

TreeMap cannot contain null keys.

Hence, ‘map.put(null, “null”);’ throws NullPointerException.

Correction Generics and Collections - 7

B - Compilation error

For ‘List<? super String>’ type of read objects is ‘Object’ and type of write objects is ‘String’ and its subclasses (no subclass of String as String is final).

‘for(String str : list)’ causes compilation failure.

Correct syntax should be: ‘for(Object str : list)’

Correction Generics and Collections - 8

C - RIF

listIterator(index); method allows to have the starting point at any index. Allowed values are between 0 and size of the list.

If next() method is called, then element at ‘specified index’ is returned and if previous() method is called, then element at ‘specified index - 1’ is returned.

Correction Generics and Collections - 9

A - true:false:3

push, pop and peek are Stack's terminology.

push(E) calls addFirst(E), pop() calls removeFirst() and peek() invokes peekFirst(), it just retrieves the first element (HEAD) but doesn't remove it.

deque.push(new Boolean("abc")); => [*false]. * represents HEAD of the deque.

deque.push(new Boolean("tRuE")); => [*true, false].

deque.push(new Boolean("FALSE")); => [*false, true, false].

deque.push(true); => [*true, false, true, false].

deque.pop() => removes and returns the HEAD element, true in this case. deque => [*false, true, false].

deque.peek() => retrieves but doesn't remove the HEAD element, false in this case. deque => [*false, true, false].

deque.size() => 3.

Hence output is 'true:false:3'.

Correction Generics and Collections - 10

B -

```
{Yusuf, Pathan}
{Tom, Hanks}
{Tom, Riddle}
```

In this case, sorted method accepts an instance of Comparator type.

Comparator.comparing(Person::getFirstName) => Returns a Comparator for sorting the records in ascending order of first name.

Comparator.comparing(Person::getFirstName).reversed() => Returns a Comparator for sorting the records in descending order of first name.

Comparator.comparing(Person::getFirstName).reversed().thenComparing(Person::getLastName) => Returns a Comparator for sorting the records in descending order of first name and in case first name matches, then ascending order of last name.

So correct answer is:

```
{Yusuf, Pathan}
{Tom, Hanks}
{Tom, Riddle}
```

Correction Generics and Collections - 11

B - 1

TreeSet requires you to provide either Comparable or Comparator. NOTE: To be used with TreeSet, it is not needed to override equals(Object) and hashCode() methods.

But in real world projects, it is a good practice to override hashCode() and equals(Object) methods for the classes to be used in Collection framework.

In this case ‘new TreeSet<>(Student::compareByName);’ provides the instance of Comparator type, which compares the names only.

All 3 Student objects have same name and hence only first Student object was added to this set.

Correction Generics and Collections - 12

E - Compilation error

print1(A<? extends Animal> obj) => print1 method can accept arguments of A or A or A types at runtime.

Suppose you have passed ‘new A()’ as the argument of print1 method. Line 22 will not work in this case.

As compiler is not sure about the data that would come at runtime, hence it doesn’t allow Line 22. Line 22 causes compilation failure.

print2(A<? super Dog> obj) => print2 method can accept arguments of A or A or A types at runtime.

All 3 arguments works with Line 27, hence no issues with Line 27.

Correction Generics and Collections - 13

D - [2, 4, 6, 8, 10]

Arrays.asList(...) method returns a list backed with array, so items cannot be added to or removed from the list.

But if this list is passed to the constructor of ArrayList, then new ArrayList instance is created which copies the elements of passed list and elements can be added to or removed from this list.

List list = new ArrayList<>(Arrays.asList(1,2,3,4,5,6,7,8,9,10)); => [1,2,3,4,5,6,7,8,9,10].

list.removeIf(i -> i % 2 == 1); => [2,4,6,8,10]. Remove the element for which passed Predicate is true.

Correction Generics and Collections - 14

B - E1 D2 C3 B4 A5

Sorting is working on 2nd letter of the array elements, which means 5, 4, 3, 2, 1.

Sorting is in ascending order (1, 2, 3, 4, 5) hence the output is: E1 D2 C3 B4 A5

Correction Generics and Collections - 15

A - [1, 3, 5, 7, 9]

replaceAll(UnaryOperator operator) is a default method available in List interface, it replaces each element of this list with the result of applying the operator to that element.

list.replaceAll(i -> i + 1); => Adds 1 to each element of the list. Result is [1, 3, 5, 7, 9].

Correction Generics and Collections - 16

B - Runtime Exception

Deque’s add() method invokes addLast(E) method and remove() method invokes removeFirst() method.

chars.add(‘A’); => [A],

chars.remove(); => [],

chars.remove(); => No elements left to remove() and hence java.util.NoSuchElementException is thrown at runtime.

Correction Generics and Collections - 17

B -

```
STOP
READY TO STOP
GO
```

TreeMap is the sorted map on the basis on natural ordering of keys (if comparator is not provided).

enum TrafficLight is used as a key for TreeMap.

The natural order for enum elements is the sequence in which they are defined. Value corresponding to ‘RED’ is printed first, followed by value corresponding to ‘YELLOW’ and finally value for ‘GREEN’ is printed.

Correction Generics and Collections - 18

C - [{Hou Jian, 42}, {Udayan, 31}, {Smita, 29}, {Neha, 23}]

Comparable interface has compareTo(…) method and Comparator interface has compare(…) method.

In this case, class Employee correctly implements Comparable interface.

return o.age - this.age; => This will help to sort the Employee objects in descending order of age and not in ascending order.

As no Comparator is passed in TreeSet, hence it sorts on the basis of implementation of Comparable interface, which means Employee objects will be sorted in descending order of their age.

Correction Generics and Collections - 19

A - Runtime exception

You cannot add or remove elements form the list returned by Arrays.asList(T...) method but elements can be re-positioned.

list.add("U"); throws UnsupportedOperationException at runtime.

Correction Generics and Collections - 20

D - Compilation error

list1 is of List type and contains 2 elements “A” and “B”.

list2 is of List<? extends Object> type, which means any List whose type extends from Object. As String extends Object, hence ‘List<? extends Object> list2 = list1;’ works.

list2.remove(“A”); => remove is non-generic method. remove(Object) will be invoked and it will successfully remove “A” from list2.

list2.add(“C”); => add is a generic method. add(? extends Object) would be invoked. This means it can take an instance of any UnknownType (extending from Object class).

Compiler can never be sure whether passed argument is a subtype of UnknownType (extending from Object class). Line 14 causes compilation failure.

NOTE: Compiler works with reference types and not instances.

Simple way to remember is that as upper-bounded wildcard is used, hence add operation is not supported. Line 14 causes compilation failure.

Correction Generics and Collections - 21

C - %

Test is generic type and Test is raw type. When raw type is used then T is Object, which means set method will have signature: set(Object t).

Test obj = new Test(); => Test object is created and obj refers to it.

obj.set(“OCP”); => Instance variable t refers to “OCP”.

obj.set(85); => Instance variable t refers to Integer object, 85. Auto-boxing converts int literal to Integer object.

obj.set(‘%’); => Instance variable t refers to Character object, %. Auto-boxing converts char literal to Character object.

obj.get() => this returns Character object as Character class overrides toString() method, % is printed on to the console.

Correction Generics and Collections - 22

A - Runtime Exception

TreeSet cannot contain null values.

Hence, ‘new TreeSet<>(Arrays.asList(null,null,null));’ throws NullPointerException.

Correction Generics and Collections - 23

D - Compilation error in Printer class

Type parameter should not be a Java keyword & a valid Java identifier. Naming convention for Type parameter is to use uppercase single character.

In class Printer, ‘String’ is a valid Java identifier and hence a valid type parameter even though it doesn’t follow the naming convention of uppercase single character.

But within Printer class, ‘String’ is considered as type parameter and not java.lang.String class. Return value of toString() method is java.lang.String class and not type parameter ‘String’.

So toString() method caused compilation error in Printer class.

To resolve the compilation error, you can use below code:

```
public java.lang.String toString() {
    return null;
}
```

Correction Generics and Collections - 24

B -

```
7 Seven
Lucky 7
77
07ne
```

contains(“7”) method checks if ‘7’ is available anywhere in the String object.

All 4 String objects contain 7 and hence all 4 String objects are printed on to the console.

Correction Generics and Collections - 25

C - Only class A compiles successfully

super is used with wildcard (?) only.

Correction Generics and Collections - 26

A -

```
Collections.sort(names, (01, 02) ->
        o1.getFirst().compareToIgnoreCase(o2.getFirst()));
```

B -

```
Collections.sort(names, (01, 02) ->
        o1.getFirst().toLowerCase().compareTo(o2.getFirst().toLowerCase()));
```

D -

```
Collections.sort(names, (01, 02) ->
        o1.getFirst().toUpperCase().compareTo(o2.getFirst().toupperCase()));
```

Collections.sort(names, (o1, o2) -> o1.getFirst().compareTo(o2.getFirst())); => It sorts in the ascending order of first name in case-sensitive manner and displays [John Smith, Peter Lee, bonita smith] in the output.

Collections.sort(names, (o1, o2) -> o1.getFirst().toLowerCase().compareTo(o2.getFirst().toLowerCase())); => At the time of comparison, first names in lower case are considered, this doesn't change the case of displayed output. Output is: [bonita smith, John Smith, Peter Lee].

Collections.sort(names, (o1, o2) -> o1.getFirst().toUpperCase().compareTo(o2.getFirst().toUpperCase())); => At the time of comparison, first names in upper case are considered, this doesn't change the case of displayed output. Output is: [bonita smith, John Smith, Peter Lee].

Collections.sort(names, (o1, o2) -> o1.getFirst().compareToIgnoreCase(o2.getFirst())); => compareToIgnoreCase method compares the first names in case-insensitive manner and displays

[bonita smith, John Smith, Peter Lee] in the output.

Correction Generics and Collections - 27

D - 0

IntStream.range(int start, int end) => start is inclusive and end is exclusive and incremental step is 1.

count() => Returns the count of elements in this stream.

IntStream.range(10,1) => Returns an empty stream as start > end, this means stream doesn’t have any elements. That is why count() returns 0.

Correction Generics and Collections - 28

C -

```
STOP
READY TO STOP
GO
```

TreeMap is the sorted map on the basis on natural ordering of keys (if comparator is not provided).

enum TrafficLight is used as a key for TreeMap. The natural order for enum elements is the sequence in which they are defined.

A map doesn't allow duplicate keys. 'map.put(TrafficLight.YELLOW, "READY TO STOP");' replaces the previous value corresponding to 'TrafficLight.YELLOW' with the new value 'READY TO STOP'.

Value corresponding to 'RED' is printed first, followed by value corresponding to 'YELLOW' and finally value for 'GREEN' is printed.

Correction Generics and Collections - 29

E - Line 9 causes compilation error

Line 8 is a valid syntax but as upper-bounded wildcard is used, hence add operation is not supported.

Line 9 causes compilation failure.

Correction Generics and Collections - 30

A -

```
{Rob, OCP}
{John, OCA}
{Jack, OCP}
```

In real world programming you will hardly find a bean class implementing Comparator but it is a legal code. A bean class generally implements Comparable interface to define natural ordering

Student class in this case correctly implements Comparator interface by overriding compare(Student, Student) method.

Note, this compare method will sort in descending order of Student's name.

list.sort(...) accepts an argument of Comparator type.

new Student() provides the instance of Comparator type. It sorts list in descending order of Students' name.

Output is:

```
{Rob, OCP}
{John, OCA}
{Jack, OCP}
```

Correction Generics and Collections - 31

D - None of the other optionsAMPR

Streams are lazily evaluated, which means if terminal operations such as: forEach, count, toArray, reduce, collect, findFirst, findAny, anyMatch, allMatch, sum, min, max, average etc. are not present, the given stream pipeline is not evaluated and hence peek() method doesn’t print anything on to the console.

Correction Generics and Collections - 32

D - NEWS

chars() method in String class returns IntStream, all the elements in this stream are stored as int value of corresponding char.

filter(Test::isDirection) => Returns the stream consisting of int (char) for which isDirection method returns true.

isDirection method returns true for ‘N’, ‘E’, ‘W’ and ‘S’ only for other characters (including whitespace character) it returns false.

forEach(c -> System.out.print((char)c)); => forEach method typecast int value to char and hence NEWS is printed on to the console.

Correction Generics and Collections - 33

D - None of the other options

Reference variable to which lambda expression is assigned is known as target type. Target type can be a static variable, instance variable, local variable, method parameter or return type. Lambda expression doesn't work without target type and target type must be a functional interface.

In this case, println(Object) method is invoked but Object is a class and not a functional interface, hence no lambda expression can be passed directly to println method.

But you can first assign lambda expression to target type and then pass the target type reference variable to println(Object) method:

```
Operator operator = (s1, s2) -> s1 + s2;

System.out.println(operator);
```

Or you can typecast lambda expression to target type. e.g. following works:

```
System.out.println((Operator)(String s1, String s2) -> s1 + s2);

System.out.println((Operator)(s1, s2) -> s1 + s2);

System.out.println((Operator)(s1, s2) -> { return s1 + s2; });
```

Correction Generics and Collections - 34

A - No

static declaration of type parameter is not allowed, only instance declaration is possible.

‘static T obj;’ causes compilation failure.

Correction Generics and Collections - 35

D - Existing code without any changes displays above output.

Even though lambda expression is for the compare method of Comparator interface, but in the code name “Comparator” is not used hence import statement is not needed here.

Expressions (s1, s2) -> s2.length()-s1.length() and (s2, s1) -> s1.length()-s2.length() displays the output in reversed order.

Correction Generics and Collections - 36

C - NullPointerException is thrown at runtime

ArrayDeque cannot store null, hence Line n2 throws NullPointerException exception.

ArrayDeque doesn’t store null because its poll() method returns null in case ArrayDeque is empty. If null element was allowed, then it would be not possible to find out whether poll() method is returning null element or ArrayDeque is empty.

Correction Generics and Collections - 37

B - FIRST

listIterator(index); method allows to have the starting point at any index. Allowed values are between 0 and size of the list.

ListIterater extends Iterator and can be used to iterate in both the directions. If you want to iterate backward then pass the no. of elements in the list to listIterator(index) method. In this case, ‘list.listIterator(5);’.

To iterate backward, use hasPrevious() and previous() methods.

Correction Generics and Collections - 38

A - [A, M, P, R]

If null Comparator is passed to sort method, then elements are sorted in natural order (based on Comparable interface implementation).

As list is of String type and String implements Comparable, hence list elements are sorted in ascending order.

Correction Generics and Collections - 39

B -

```
[ONE]
[TWO]
```

LinkedList implements both List and Queue. In this example reference type controls the LinkedList behavior.

list.add(“ONE”); => [ONE].

list.add(“TWO”); => [ONE,TWO]. Adds to the last.

list.remove(1); => [ONE]. Removes from the specified index.

System.out.println(list); => [ONE].

queue.add(“ONE”); => [*ONE]. * represents HEAD of the queue.

queue.add(“TWO”); => [*ONE,TWO]. Adds to the end of the queue.

queue.remove(); => [*TWO]. Removes from the HEAD of the queue.

System.out.println(queue); => [TWO].

Correction Generics and Collections - 40

D -

```
ocpjp@gmail.com
training@gmail.com
training@outlook.com
ocpjp@outlook.com
```

Comparator is comparing on the basis of email domain: gmail.com and outlook.com.

Insertion order is:

```
training@outlook.com
ocpjp@outlook.com
ocpjp@gmail.com
training@gmail.com
```

gmail records should appear before outlook records. So sorting order is:

```
ocpjp@gmail.com
training@gmail.com
training@outlook.com
ocpjp@outlook.com
```

NOTE: It is not specified, what to do in case email domain is matching. So, for matching email domain, records are left at insertion order.

Correction Generics and Collections - 41

B - Yes

‘GenericPrinter’ is generic type and is defined correctly.

‘AbstractGenericPrinter<X,Y,T>’ is also a generic type and extends another generic type ‘GenericPrinter’.

NOTE: If a class extends from generic type, then it must pass type arguments to its super class. Third type parameter, ‘T’ in ‘AbstractGenericPrinter<X,Y,T>’ correctly passed type argument to super class, ‘GenericPrinter’.

Below codes will not compile:

abstract class AbstractGenericPrinter<X,Y> extends GenericPrinter{} => No way to pass type argument to the type parameter, T of super class.

abstract class AbstractGenericPrinter extends GenericPrinter{} => No way to pass type argument to the type parameter, T of super class.

But below codes will compile successfully:

abstract class AbstractGenericPrinter<X,Y> extends GenericPrinter{} => Type argument, ‘String’ is passed to the type parameter, T of super class.

abstract class AbstractGenericPrinter extends GenericPrinter{} => Type argument, ‘String’ is passed to the type parameter, T of super class.

Correction Generics and Collections - 42

E - Generic obj = new Generic<>();

T is with multiple bounds, so the type argument must be a subtype of all bounds.

Correction Generics and Collections - 43

A - 12 is displayed on to the console

B - Code compiles with some warnings

Compiler warning for unchecked call to add and forEach.

list can store all objects and when each element is passed to System.out.print() method, toString() method for passed element is invoked.

Both Integer and String class overrides toString() method and hence 12 is printed on to the console.

Correction Generics and Collections - 44

C - Compilation error

A generic method is defined in non-generic class.

Type parameter for the method should be defined just before the return type of the method.

In this case, ‘’ is not appearing just before void and hence compilation error.

Correction Generics and Collections - 45

B - Aabc

TreeSet requires you to provide either Comparable or Comparator. If you don’t provide Comparator explicitly, then for natural ordering your class should implement Comparable interface.

Character and all wrapper classes implement Comparable interface, hence Characters are sorted in ascending order. Uppercase characters appears before lowercase characters.

Set doesn’t allow duplicate, hence output will always be: ‘Aabc’.

Correction Generics and Collections - 46

B - [Point(6, 7), Point(4, 5), Point(2, 2)]

return o2.getX() - o1.getX(); means the Comparator is sorting the Point objects on descending value of x of Point objects.

To sort the Point objects in ascending order of x, use: return o1.getX() - o2.getX();

To sort the Point objects in ascending order of y, use: return o1.getY() - o2.getY();

To sort the Point objects in descending order of y, use: return o2.getY() - o1.getY();

Correction Generics and Collections - 47

A - Compilation error for Printer class

For bounds, extends keyword is used for both class and interface.

Correct declaration of Printer class should be:

class Printer {}

Correction Generics and Collections - 48

A -

```
gray
gray
green
blue
```

new TreeSet<>(Arrays.asList(“red”, “green”, “blue”, “gray”)); => [blue, gray, green, red].

set.ceiling(“gray”) => Returns the least value greater than or equal to the given value, ‘gray’.

set.floor(“gray”) => Returns the greatest value less than or equal to the given value, ‘gray’.

set.higher(“gray”) => Returns the least value strictly greater than the given value, ‘green’.

set.lower(“gray”) => Returns the greatest value strictly less than the given value, ‘blue’.

Correction Generics and Collections - 49

A - Runtime exception

Iterator and ListIterator allow to remove elements while iterating. But next() should be called before remove().

In this case, remove() is called before next() and hence IllegalStateException is thrown at runtime.

Correction Generics and Collections - 50

A - {null=zero, 1=one}

HashMap and LinkedHashMap can accept 1 null key but TreeMap cannot accept null keys.

LinkedHashMap by default keeps an insertion order so every time you iterate the map, you get same result.

Output will always be: {null=zero, 1=one}

Correction Generics and Collections - 51

B - deque.forEach(s -> System.out.println(s));

C - deque.forEach(System.out::println);

Iterator interface has forEach(Consumer) method. As Consumer is a Functional Interface, hence a lambda expression or method reference syntax can be passed as argument to forEach() method.

deque.forEach(System.out::print); => This will print 100020003000 without any newline character in between.

deque.forEach(System.out::println); => This prints desired output

deque.forEach(i -> System.out.println(i)); => Causes compilation failure as lambda expression variable ‘i’ conflicts with local variable.

deque.forEach(s -> System.out.println(s)); => Prints desired output.

NOTE: ‘System.out::print’ is a method reference syntax corresponding to lambda expression ‘s -> System.out.println(s)’.

Correction Generics and Collections - 52

D -

```
Collections.sort(names, new Comparator<String>() {
  public int compare(String 01, String 02) {
    return 0o1.compareToIgnoreCase(o2);
  }
});
```

If you sort String in ascending order, then upper case letters appear before the lower case letters.

So in this case if I sort the list in ascending order then the output will be [Anna, James, diana] and this is what Collections.sort(names); and o1.compareTo(o2); method calls do.

o2.compareTo(o1); sorts the same list in descending order: [diana, James, Anna] but you have to sort the list such that [Anna, diana, James] is printed in the output,

which means sort the names in ascending order but in case-insensitive manner. String class has compareToIgnoreCase() method for such purpose.

Correction Generics and Collections - 53

D -

```
{Jack, 11000.0}
{Lucy, 13000.0}
```

employees.stream() => [{“Jack”,10000.0},{“Lucy”,12000.0}].

peek(e -> e.setSalary(e.getSalary() + 1000)) => [{“Jack”,11000.0}, {“Lucy”,13000.0}]. peek(Consumer) method applies the passed lambda expression to all the elements of the stream and returns the same elements in the stream.

forEach(System.out::println); => Prints both the elements of the stream.

Arrays.asList(…) method returns sequential List object, so order of elements remain same.

Output is:

{Jack, 11000.0} {Lucy, 13000.0}

NOTE: peek() method is for debugging the streams and should not be used in production ready code.

Correction Generics and Collections - 54

C - Compilation error

x and y are private variables and are accessible within the boundary of Point class.

TestPoint class is outside the boundary of Point class and hence o1.x and o2.x cause compilation error.

Make sure to check the accessibility before working with the logic.

Correction Generics and Collections - 55

A - Lucy

employees.stream() => Gets the stream for employees list.

filter(x -> x.getSalary() > 10000) => Returns a stream consisting of elements of the stream that match the given Predicate. In this case {“Lucy”, 12000} is returned.

map(e -> e.getName()) => Returns a stream consisting of the results of applying the given function to the elements of this stream. In this case {“Lucy”} is returned.

forEach(System.out::println) => Prints ‘Lucy’ on to the console.

Correction Generics and Collections - 56

D - AbBc

Uppercase characters are different from Lowercase characters.

distinct() method of Stream returns a stream consisting of the distinct elements (according to Object.equals(Object)) of this stream.

“A” and “A” are same, “b” and “B” are different & “c” and “c” are same.

Arrays.asList(…) method returns sequential List object, so order of elements remain same.

Output is: AbBc

Correction Generics and Collections - 57

A - [C]

Deque’s add() method invokes addLast(E) method and remove() method invokes removeFirst() method.

chars.add(‘A’); => [*A], {* represents HEAD element}

chars.add(‘B’); => [*A,B],

chars.remove(); => [*B],

chars.add(‘C’); => [*B,C],

chars.remove(); => [*C],

System.out.println(chars); => Prints [C] on to the console.

Correction Generics and Collections - 58

D -

```
0
1
```

IntStream.range(int start, int end) => start is inclusive and end is exclusive. If start >= end, then empty stream is returned.

IntStream.range(-10, -10) => returns empty stream.

IntStream.rangeClosed(int start, int end) => Both start and end are inclusive. If start > end, then empty stream is returned.

IntStream.rangeClosed(-10, -10) => returns a stream containing just 1 element, which is -10.

Correction Generics and Collections - 59

A - No

If multiple bounds are available and one of the bounds is a class, then it must be specified first.

class Generic<T extends M & N & A> {} => A is specified at last and hence compilation error.

Correction Generics and Collections - 60

C -

```
BARBINBB011
ICICINBBRT4
BOTKINDD075
ICICINDD016
SBBJINDD062
ABNATHBK8&65
BKCHTHBK012
```

Default thenComparing method helps to chain the Comparators.

First the list is sorted on the basis of country code, if matching country code is found then sorted on the basis of location code and if location code matches then list is sorted on bank code.

Correction Generics and Collections - 61

B - 1357911

IntStream.iterate(1, i -> i + 1) => [1,2,3,4,5,6,7,...]. This is an infinite stream.

limit(11) => [1,2,3,4,5,...,11]. Eleven elements from the above stream.

filter(i -> i % 2 != 0) => [1,3,5,7,9,11]. Returns a stream consisting of all odd numbers.

forEach(System.out::print); => Prints '1357911' on to the console.

Correction Generics and Collections - 62

A - print(new Double(5.5));

B - print(new Integer(1));

Number is an abstract class, so ‘new Number(0)’ cannot be used.

Character class doesn’t extend Number.

Object class doesn’t extend Number.

Correction Generics and Collections - 63

C - 3

HashSet makes use of hashCode to find out the correct bucket, it then makes use of equals(Object) method to find out duplicate objects.

Student class correctly overrides equals(Object) method but it doesn't override hashCode() method. This means you get different hashCode for different objects.

HashSet in this case cannot find out duplicate Student objects and 3 Student objects are added to the Set.

System.out.println(students.size()); => Prints 3 on to the console.

To avoid duplicate in the given Set, override hashCode() method in Student class:

```
public int hashCode() {
    return name.hashCode() + age;
}
```

Correction Generics and Collections - 64

D - Program compiles and executes successfully but nothing is printed on to the console

Operator is a generic interface, hence it can work with any java class. There are absolutely no issues with lambda expressions but we are not capturing or printing the return value of operation method, hence nothing is printed on to the console.

System.out.println(opr1.operation(“Hello”, “World”)); => Prints HelloWorld.

System.out.println(opr2.operation(10, 40)); => Prints 50. Over here int literals 10 and 40 are converted to Integer instances by auto-boxing.

Correction Generics and Collections - 65

B - HELLO

Return type of generic method ‘get’ is T, which is correctly defined before the return type of the method.

get(“HELLO”); passed String so return value should be String only.

String result is stored in str variable and same is printed using System.out.println statement.

Correction Generics and Collections - 66

B -

```
{11=Sri Nagar}
{25=Pune, 32=Mumbai, 39=Chennai}
```

If you don’t use 2nd parameter for headMap() and tailMap() methods to indicate whether keys are inclusive or exclusive,

then by default ‘toKey’ used in headMap() method is exclusive and ‘fromKey’ used in tailMap() method is inclusive.

You can confirm this by checking the definition of headMap() and tailMap() methods in TreeMap class.

Methods defined in TreeMap class:

```
public SortedMap<K,V> headMap(K toKey) {
    return headMap(toKey, false);
}

public SortedMap<K,V> tailMap(K fromKey) {
    return tailMap(fromKey, true);
}
```

Correction Generics and Collections - 67

A - 300

deque.add(100); => {*100}.* represents HEAD of the deque.

deque.add(200); => {*100, 200}. add(E e) invokes addLast(e) method.

deque.addFirst(300); => {*300, 100, 200}.

deque.addLast(400); => {*300, 100, 200, 400}.

deque.remove(200); => {*300, 100, 400}. Deque interface doesn't have remove(int index) method.

System.out.println(deque.getFirst()); => Prints 300 on to the console.

You should be aware of other methods from Deque interface as well, such as:

removeFirst(); => Removes the first element from the Deque.

removeLast(); => Removes the last element from the Deque.

Correction Generics and Collections - 68

B - Compilation error

Comparator interface has compare(…) method and not compareTo(…) method.

Anonymous inner class's syntax doesn't implement compare(…) method and thus compilation error.

Make sure to check the accessibility and interface method details before working with the logic.

Correction Generics and Collections - 69

B - Some text containing @ symbol

Even though String is a final class but T extends String is a valid syntax.

As no class extends from java.lang.String class so parameterized type will always be Printer.

Generic class Printer doesn't override toString() method, hence Object version is invoked.

Correction Generics and Collections - 70

B -

```
#####
####
###
##
#
```

Comparator.comparing(s -> s); compares the passed Strings only. As all the characters in the String are '#', this means strings are sorted on the basis of their lengths.

Comparator referred by comp sorts on the basis of strings' lengths.

Default reversed() method just reverses the ordering of the Comparator referred by comp, which means sorts the strings in descending order of their lengths.

Correction Generics and Collections - 71

B - Compilation error

Instantiation of a type parameter ‘new T()’ or an array of type parameter ‘new T[5]’ are not allowed.

‘obj = new T[100];’ causes compilation failure.

Correction Generics and Collections - 72

B - Compilation error at Line 12

list is of List (raw) type. So, it can accept any object. Line 11 doesn't cause any compilation error.

As list is raw list, which means it is of Object type, hence in Predicate's lambda expression, i is of Object type.

Modulus operator (%) cannot be applied to Object type. So, Line 12 causes compilation error.

NOTE: This questions checks whether you can find out the issues when raw and generic types are mixed.

Correction Generics and Collections - 73

A - Runtime Exception

TreeSet requires you to provide either Comparable or Comparator.

If you don’t provide Comparator explicitly, then for natural ordering your class should implement Comparable interface.

Student class doesn’t use Comparable, hence ClassCastException is thrown at runtime.

Correction Generics and Collections - 74

D - 30

If a generic method is defined in a non-generic class then type parameters must appear before the return type of the method.

Integer is also a final class so parameters X and Y can only be of Integer type.

add(10, 20); => Auto-boxing converts int literals to Integer objects. 30 is printed on to the console.

Correction Generics and Collections - 75

D - [abc, xyz]

Streams are lazily evaluated, which means if terminal operations such as: forEach, count, toArray, reduce, collect, findFirst, findAny, anyMatch, allMatch, sum, min, max, average etc. are not present, the given stream pipeline is not evaluated and hence map() method doesn't reverse the stream elements.

'[abc, xyz]' is printed on to the console.

If you replace 'list.stream().map(x -> x.reverse());' with 'list.stream().map(x -> x.reverse()).count();' then output will be: '[cba, zyx]'.

Correction Generics and Collections - 76

A - 0246

LongStream.iterate(long seed, LongUnaryOperator f) => ‘seed’ is the initial element and ‘f’ is a function to be applied to the previous element to produce a new element.

LongUnaryOperator is a functional interface and has method ‘long applyAsLong(long operand);’. This means lambda expression should accept long parameter and return long value.

‘i -> i + 2’ is the correct lambda expression.

LongStream.iterate(0, i -> i + 2) => This results in an infinite stream consisting of elements [0,2,4,6,8,10,12,…]

limit(4) => Returns a stream consisting of 4 elements [0,2,4,6] from the given stream.

Hence, the output is: 0246

Correction Generics and Collections - 77

C - 123456789

IntStream.range(int start, int end) => start is inclusive and end is exclusive and incremental step is 1.

So, stream consists of value from 1 to 9 and these values are printed by forEach method.

NOTE: For IntStream.rangeClosed(int start, int end), both start and end are inclusive.

Correction Generics and Collections - 78

B -

```
{Jack, 12000.0}
{Lucy, 14400.0}
```

Iterator interface has forEach(Consumer) method. As Consumer is a Functional Interface and it has ‘void accept(T t)’ method, hence a lambda expression for 1 parameter can be passed as argument to forEach(...) method.

‘e -> e.setSalary(e.getSalary() + (e.getSalary() * .2))’ => increments the salary of all the employees by 20%.

‘System.out::println’ => prints employee object on to the console.

As salary is of double type, so decimal point (.) is shown in the output.

Correction Generics and Collections - 79

A - Compilation error

List is super type and ArrayList is sub type, hence List l = new ArrayList(); is valid syntax.

Animal is super type and Dog is sub type, hence Animal a = new Dog(); is valid syntax. Both depicts Polymorphism.

But in generics syntax, Parameterized types are not polymorhic, this means ArrayList is not super type of ArrayList. Remember this point. So below syntaxes are not allowed:

ArrayList list = new ArrayList(); OR List list = new ArrayList();

Correction Generics and Collections - 80

C - Runtime exception

HashSet cares about uniqueness and allows 1 null value.