REVIEW EXERCISE

Generic Classes

R18.1What is a type parameter?

Ans: In Java, a type parameter is a way to design classes, interfaces, and methods to operate on a specified data type without knowing what that type will be. It allows you to create classes, methods, or interfaces that can work with any data type, providing flexibility and reusability in your code. Type parameters are specified using angle brackets (`<>`) and are commonly used in generics to create generic classes and methods.

R18.2 What is the difference between a generic class and an ordinary class?

Ordinary Class:

1. **Specific Data Types:** In an ordinary class, the data types of its attributes and methods are specific and fixed. For example, if you have a class representing a stack, the data type of the elements in the stack might be integers (`int`), and the methods are designed to work specifically with integers.
2. **Reusability:** Ordinary classes are not inherently designed to work with different data types. If you want to create a stack for a different data type, you would need to create a new class specifically tailored for that data type.

Generic Class:

3. **Parameterized Data Types:** Generic classes, on the other hand, allow you to create classes, methods, interfaces, and delegates with a placeholder for the data type. These classes can work with any data type, making them highly versatile and reusable.
4. **Reusability and Type Safety:** Generic classes promote code reusability since the same class can be used with different data types without modification. They also provide type safety, ensuring that the data types used are compatible and reducing the risk of runtime errors.

R18.3What is the difference between a generic class and a generic method?

Generic Class:

A generic class in Java is a class that can operate on any data type. It's defined with one or more type parameters, specified within angle brackets (<>). These type parameters can then be used throughout the class definition to represent the data type of class members, methods, or method parameters.

Generic Method:

A generic method, on the other hand, is a method that can operate on different data types independently of the class to which it belongs. Generic methods are declared with their own type parameters, which are specified before the return type of the method.

R18.4 **Why is it necessary to provide type arguments when instantiating a generic class, but not when invoking an instantiation of a generic method?**

Solution: We must provide a type argument for instantiating a generic class so that the compiler knows the memory size for each element within the generic class. There is no need to supply type arguments when invoking an instantiation of a generic class since the type class is already known.

R18.5 Find an example of a non-static generic method in the standard Java library.

Built-in methods in Java's standard library, such as those provided by classes like `String`, `List`, `Map`, etc., are typically non-static methods. One example of a non-static generic method in the standard Java library is the toArray(T[] a) method in the java.util.Collections class. This method returns an array containing all of the elements in this collection.

R18.6 Find four examples of a generic class with two type parameters in the standard Java library.

```
1. HashMap<String, Integer> myMap = new HashMap<>();
myMap.put("One", 1);
myMap.put("Two", 2);
```

2. ArrayList<String> stringList = new ArrayList<>();

stringList.add("Hello");

stringList.add("World");

3. Pair<String, Integer> pair = new Pair<>("Age", 25);

4. LinkedHashSet<Double> doubleSet = new LinkedHashSet<>();

doubleSet.add(3.14);

doubleSet.add(2.71);

R18.7 Find an example of a generic class in the standard library that is not a collection class.

java.lang.Class<T>

java.lang.Enum<T>

java.util.concurrent.Exchanger<V>

java.util.concurrent.ExecutorCompletionService<V>

java.util.concurrent.FutureTask<T>

R18.8 Why is a bound required for the type parameter T in the following method?

<T extends Comparable> int binarySearch(T[] a, T key)

A binary search locates a value in a sorted array by determining whether the value occurs in the first or second half, then repeating the search in one of the halves. To know if the key occurs in the first half or second half, the binary search algorithm

needs to be able to determine whether the key is smaller or greater than the middle element. The type bound allows it to use the compareTo method of the Comparable interface to compare values because any value in the array must implement Comparable.

R18.9 Why is a bound not required for the type parameter E in the HashSet<E> class?

All classes are descendants of the class Object. The class Object has a

method to calculate the hash code of an object. Thus, objects of any class

can be stored in a hash set.