Chapter 13 - Java

// **Overloading vs Overriding**

1. overloading → two methods have the same name but differ in the type or number of arguments
2. overriding → method shares the same name and function signature as another method in its super class

// **Collection Framework**

1. ArrayList → dynamically resizing array, which grows as you insert elements
2. Vector → similar to ArrayList except that is synchronized
3. LinkedList
4. HashMap

// **Private Constructor** → In terms of inheritance, what is the effect of keeping a constructor private?

// **Return from Finally** → the finally block gets executed even if we insert a return statement inside the try block of a try-catch-finally

// **Difference between final, finally and finalize**

1. final → controls if a variable, method or class is changeable
    a. primitive → value can't change
    b. reference → can't point to any other object on the heap
    c. method → can't be overridden
    d. class → cannot be subclassed
2. finally → used in a try-catch block to esnure that a segment of code is always executed
    a. will always be executed even if exception is thrown
    b. except if the JVM exits
3. finalize() → method called by the garbage collector once it determines that no more references exist
    a. a class can therefore override the finalize() method from the Object class in order to define custom behaviour during garbage collection

// **Generics**

- syntactic sugar → Vector<String> is rewritten during compilation and replaced with type casts

- in Java, static variables are shared across instances of MyClass, regardless of the different type parameters
- in Java you can't use primitive types as generics
- in Java, you can restrict the template's type parameters to be of a certain type
- in Java, you can't instantiate the type parameter
- in Java, the type parameter can't be used for static methods and variables
- in Java, type parameters are erased at runtime

// **TreeMap, HashMap and LinkedHashMap** → all offer key-value map and a way to iterate through the keys

- HashMap → offers O(1) lookup and insertion, the order of the keys is arbitrary and it's implemented by an array of linked lists
- TreeMap → offers O(log N) lookup and insertion, keys are ordered and must implement the Comparable interface and its implemented using a Red-Black Tree
- LinkedHashMap → offers O(1) lookup and insertion. Keys are ordered by their insertion order and its implemented by doubly-linked buckets

// **Object Reflection** → feature that provides a way to get reflective information about Java classes and objects

- get information about the methods and fields present inside the class at runtime
- creating a new instance of a class
- getting/setting the object fields directly by getting field reference regadless what the access modifier is

// **Lambda Expressions**

```
int getPopulation(List<Country> countries, String continent) {
        Stream<Country> sublist = countries.stream()
                .filter(country -> country.getContinent().equals(continent));
        Stream<Integer> populations = sublist.map(c -> c.getPopulation());
        int population = populations.reduce(0, (a, b) -> a + b);
        return population;
}
```

// **Lambda Random**

```
Random random = new Random();
Predicate<Object> flipCoin = o -> random.nextBoolean();

List<Integer> getRandomSubset(List<Integer> list) {
        List<Integer> subset = list.stream().filter(flipCoin).collect(Collectors.toList());
        return subset;
}
```