Container Classes

Nested Collections

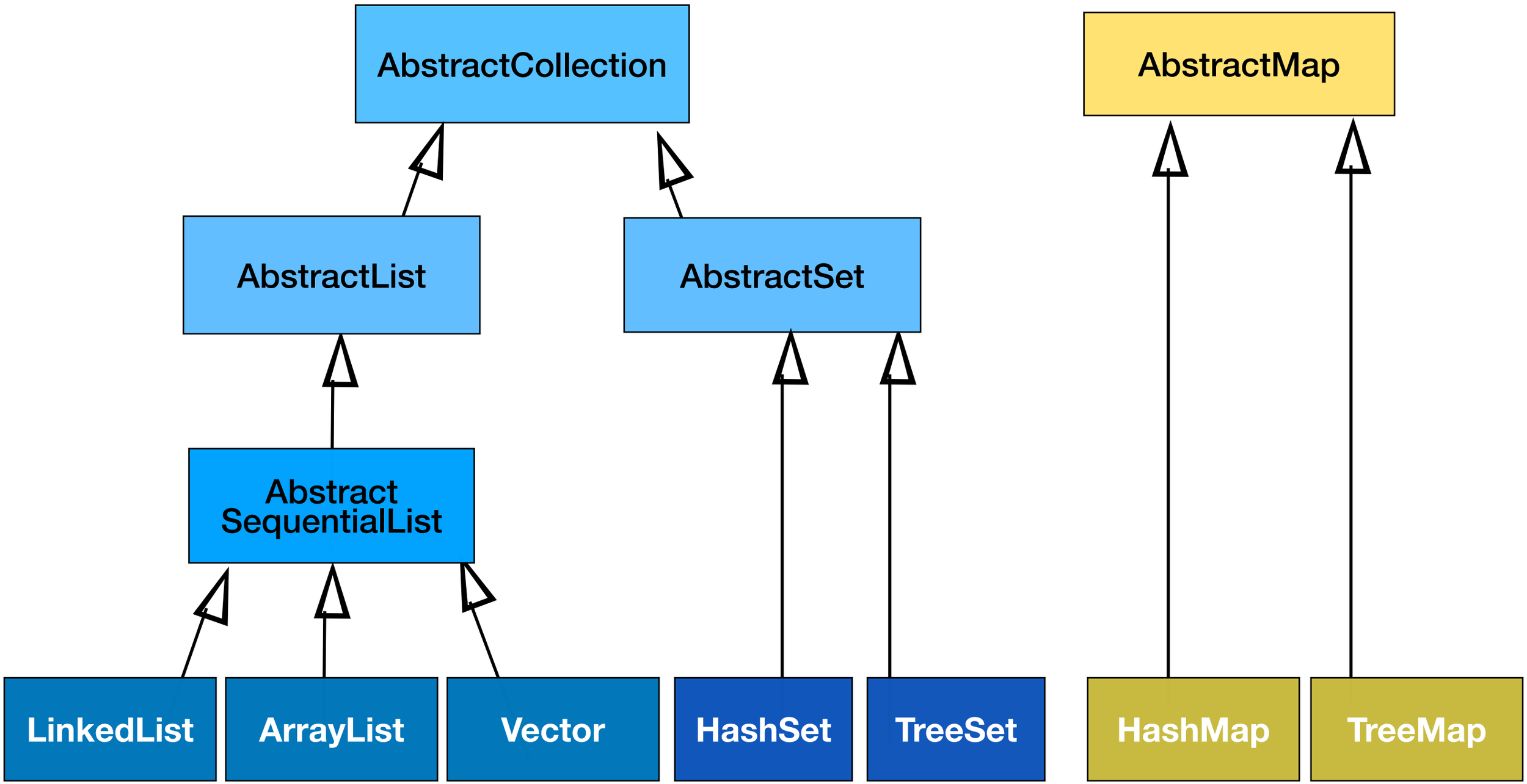
COMP2603
Object Oriented Programming 1

Week 10

Outline

- Nested Collections
  - Sets
  - Maps
- Choosing collections
- Scenarios

Classes in the Java Collections Framework



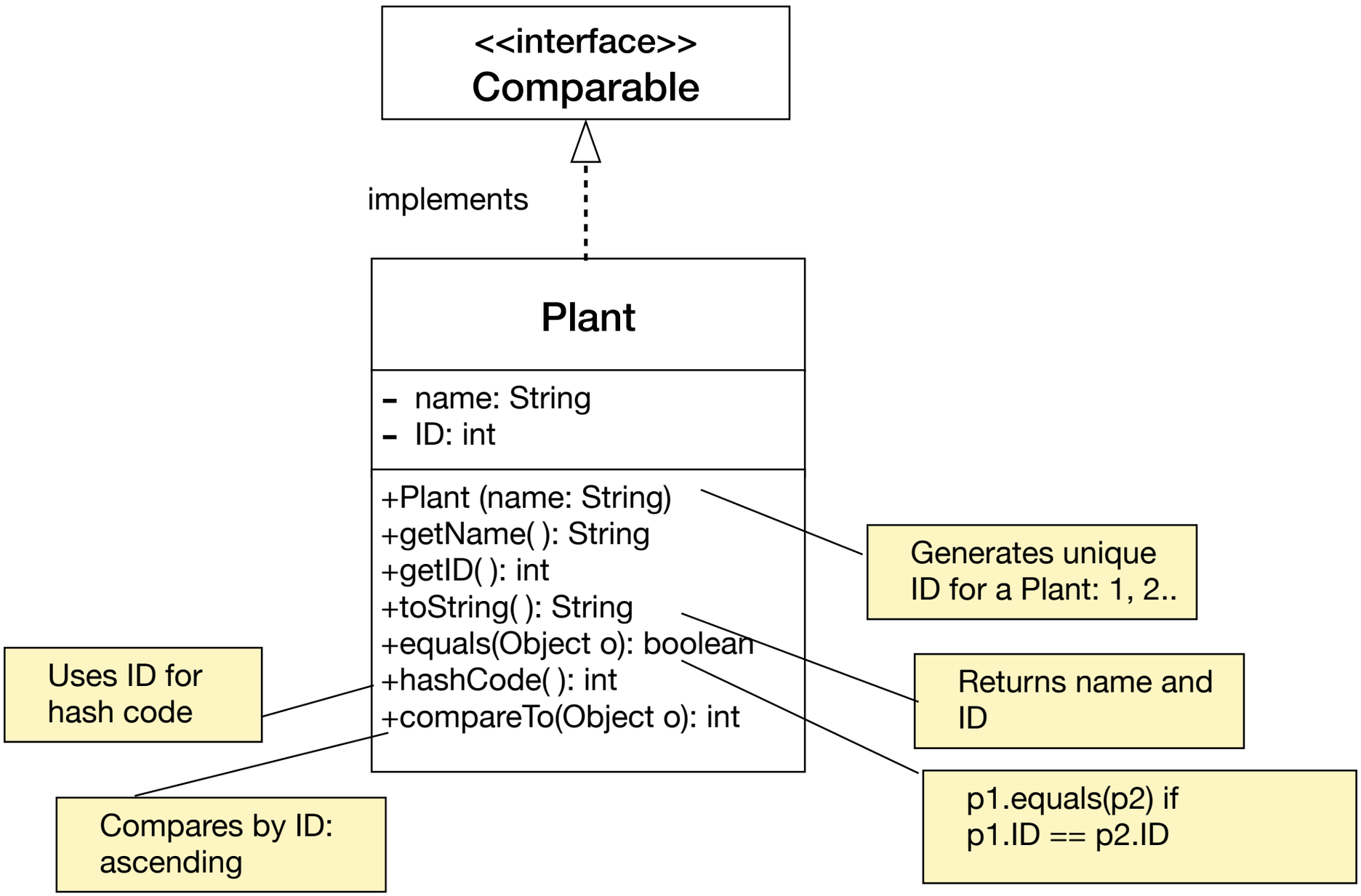
Comparing Collections

Collection	Ordering Possible	Random Access	Key-Value	Duplicate Elements	Null Element	Thread Safe
ArrayList	Yes	Yes	No	Yes	Yes	No
LinkedList	Yes	No	No	Yes	Yes	No
HashSet	No	No	No	No	Yes	No
TreeSet	Yes	No	No	No	No	No
HashMap	No	Yes	Yes	No	Yes	No
TreeMap	Yes	Yes	Yes	No	No	No
Vector	Yes	Yes	No	Yes	Yes	Yes

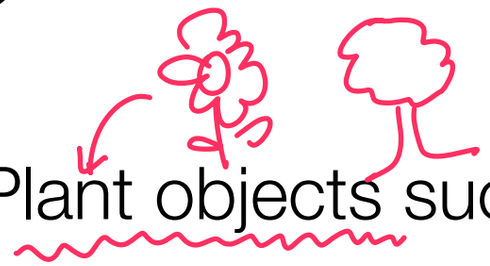
Source: http://www.journaldev.com/1260/java-collections-framework-tutorial

Plant Class

Suppose we have a Plant class as follows:



Choosing Collections



Suppose we want to store Plant objects such that:

- The appropriate Plant object is retrieved in a thread-safe manner across a distributed application.

  Options:

  - Vector, Plant equality based on ID

```
Vector<Plant> plants = new Vector<>();
```

Choosing Collections

Suppose we want to store Plant objects efficiently such that no duplicates are stored and the objects are iterable

Options:

- HashSet, Plant equality based on ID, override hashCode( ) to use IDs. (More efficient retrieval, no traversal is done).

Choosing Collections

Suppose we want to store Plant objects such that no duplicates are stored and sorted is ascending order by ID.

Options:

- TreeSet, Plant equality based on ID, String class implements Comparable interface, compareTo( ) method based on ID.

```
TreeSet<Plant> plants = new TreeSet<>();
```

Choosing Collections

Suppose we want to store Plant objects such that:

- Given any (plant) name, the appropriate Plant object is retrieved efficiently.

  Options:

  - HashMap, Store names as keys, Plant equality based on name, override hashCode( ) method to use name. (More efficient retrieval, no traversal is done).

HashMap<~~Plant~~> plants = new HashMap<>();

Choosing Collections

Suppose we want to store Plant objects in sorted order such that:

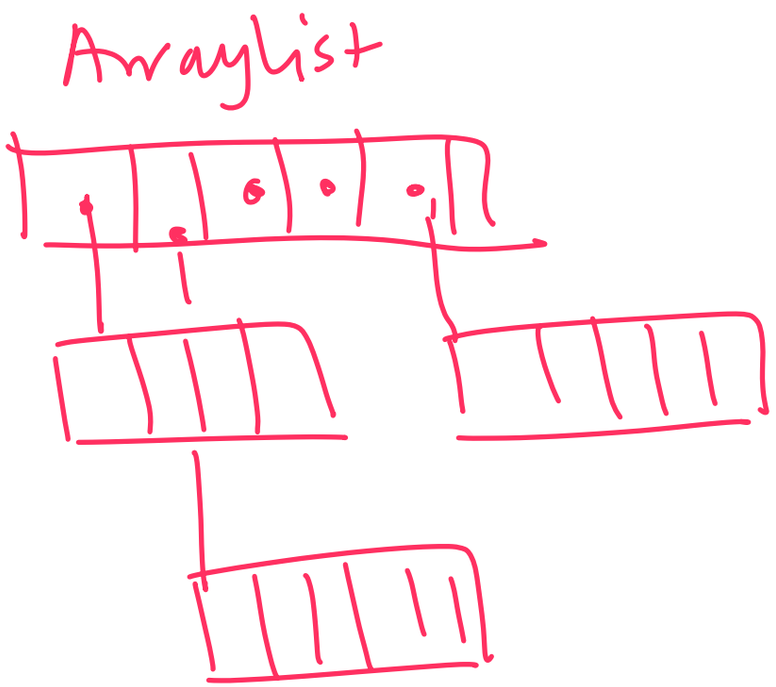
- Given any name, the appropriate Plant object is retrieved efficiently.

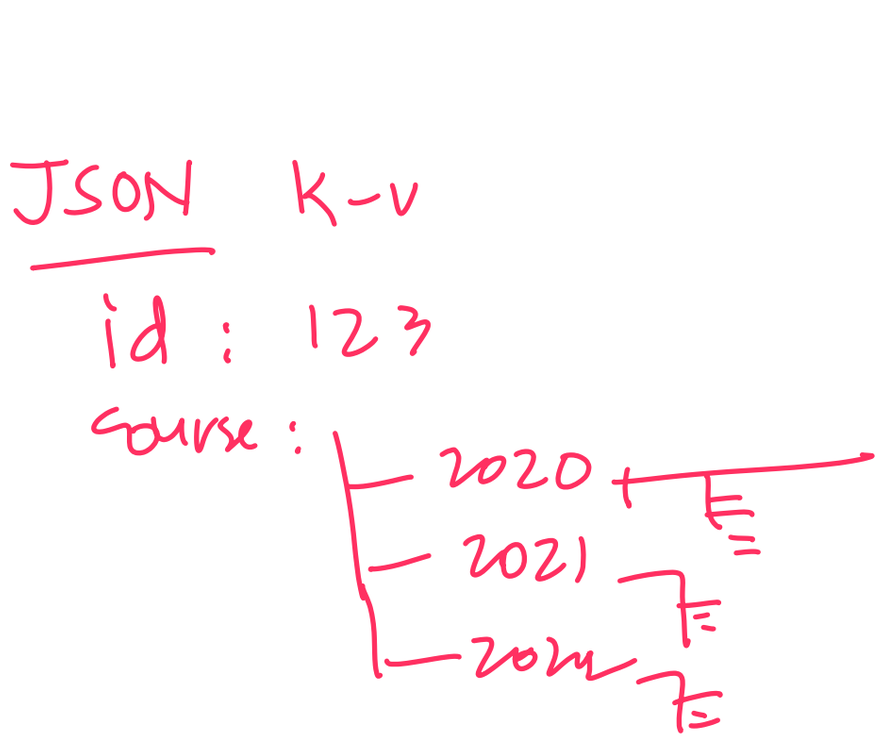
  Options:

  - TreeMap, Store names as keys (String implements Comparable), Plant equality based on name. (More efficient retrieval, no traversal is done).

```
TreeMap<String, Plant> plants = new TreeMap<>();
```

Nested Collection Exercises



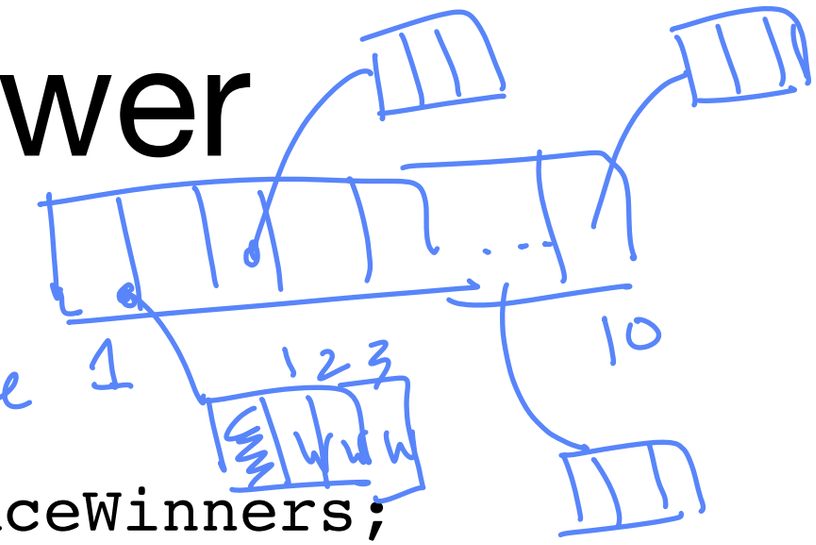


Exercise 1

- A competition stores a list of winners for first, second and third place in order.
- The competition has 10 races identified by a number from 1… 10
- A race is not modelled as an object but only by a unique race numbers
  - A winner is modelled by a String
  - Given a race number, any of winners for first, second or third place can be *randomly accessed.*

Write Java code for the Collection used in the Competition class to model this data.

Exercise 1 Answer

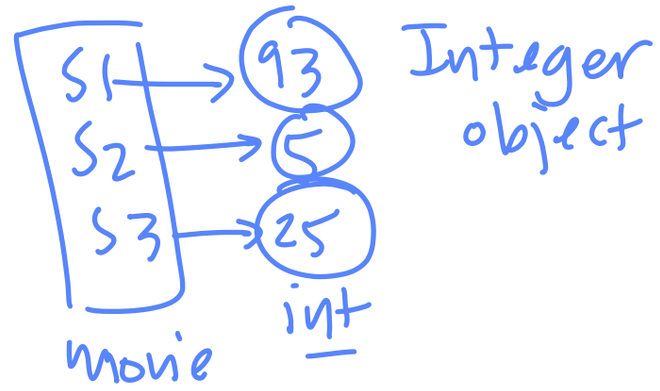



```
public class Competition {

    private ArrayList<List<String>> raceWinners;

    public Competition() {
        raceWinners = new ArrayList<>();
        for(int i = 1; i<=10; i++)
            raceWinners.add(I, new ArrayList(4));
    }
}
```

Exercise 2



A cinema stores a list of movies, **movies**, that are now showing. The list is in alphabetical order. For any movie, the cinema is able to tell a customer how many seats are available.

1. Write Java code for the Cinema class that has either a Collection or a Map for ordering the movies.
2. Write Java code for a method *getNumSeats(String movie): int* the returns the number of seats left for the given movie if found, otherwise it returns -1.

Exercise 2 Answer

```
public class Cinema {

    private TreeMap<String, Integer> movies;

    public Cinema() {
        movies = new TreeMap<>();
    }

    public int getNumSeats(String movie){
        Integer seats = movies.get(movie);
        if(seats!= null)
            return Integer.parseInt(seats);
        return -1;
    }
}
```

```
if (movies.contains (movie))
    return Integer.parseInt (movies.get (movie));
```