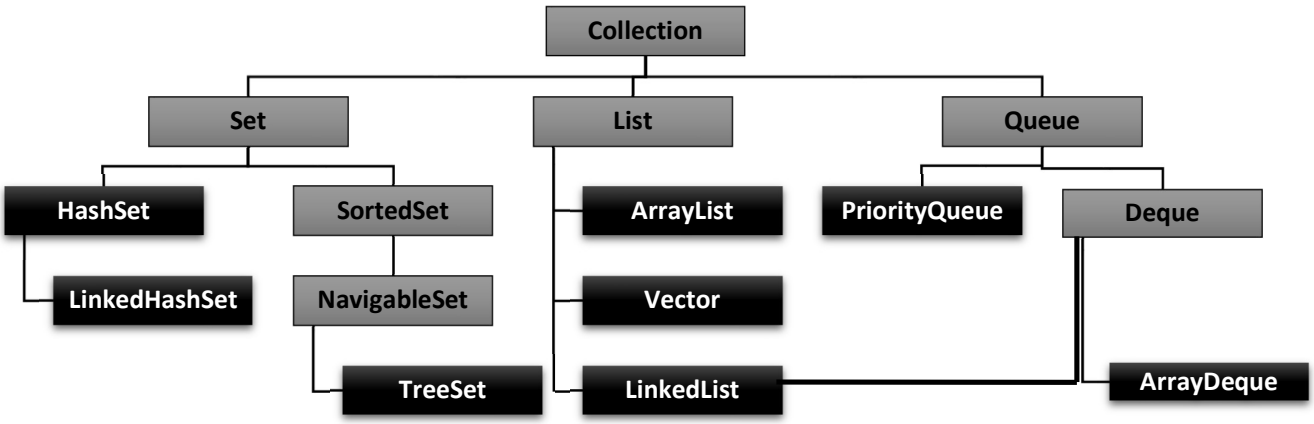
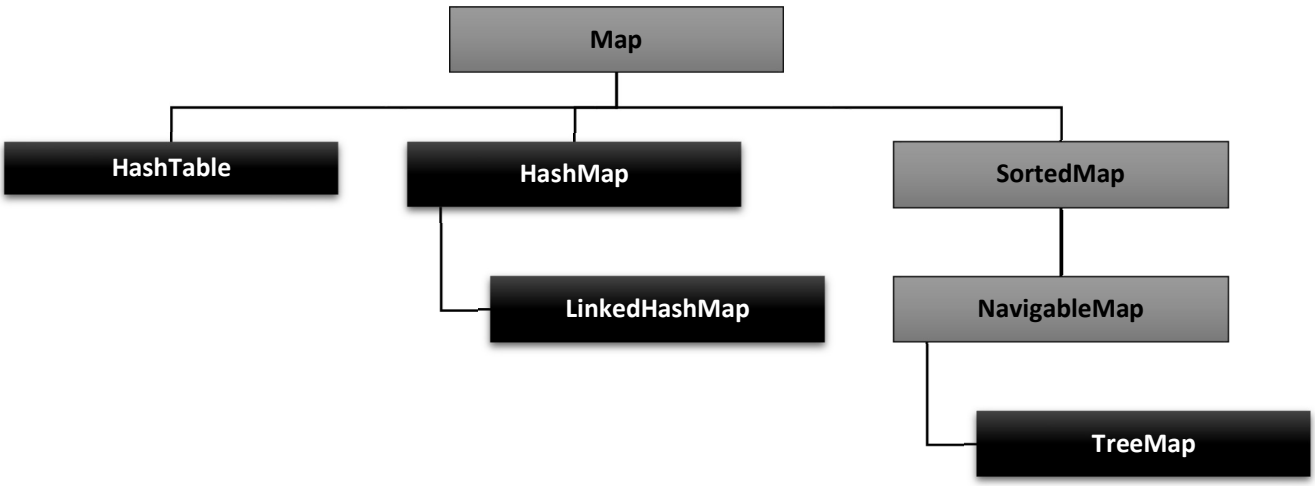
Java Collections Framework (Day2)

- Set
- Map





Methods of Collection interface

public abstract int size()
public abstract boolean isEmpty()
public abstract boolean contains(java.lang.Object)
public abstract java.util.Iterator<E> iterator()
public abstract java.lang.Object[] toArray()
public abstract <T> T[] toArray(T[])
public abstract boolean add(E)
public abstract boolean remove(java.lang.Object)

public abstract boolean containsAll(java.util.Collection<?>)
public abstract boolean addAll(java.util.Collection<? extends E>)
public abstract boolean removeAll(java.util.Collection<?>)
public boolean removeIf(java.util.function.Predicate<? super E>)
public abstract boolean retainAll(java.util.Collection<?>)
public abstract void clear()
public abstract boolean equals(java.lang.Object)
public abstract int hashCode()
public java.util.Spliterator<E> spliterator()
public java.util.stream.Stream<E> stream()
public java.util.stream.Stream<E> parallelStream()

Example: TestCollection.java

Methods of Set interface
NOTE: Set inherits only methods from Collection
public static <E> java.util.Set<E> of()
public static <E> java.util.Set<E> of(E)
public static <E> java.util.Set<E> of(E, E)
public static <E> java.util.Set<E> of(E, E, E)
public static <E> java.util.Set<E> of(E, E, E, E)
public static <E> java.util.Set<E> of(E, E, E, E, E)
public static <E> java.util.Set<E> of(E, E, E, E, E, E)
public static <E> java.util.Set<E> of(E, E, E, E, E, E, E)
public static <E> java.util.Set<E> of(E, E, E, E, E, E, E, E)
public static <E> java.util.Set<E> of(E, E, E, E, E, E, E, E, E)
public static <E> java.util.Set<E> of(E, E, E, E, E, E, E, E, E, E)
public static <E> java.util.Set<E> of(E...)
public static <E> java.util.Set<E> copyOf(java.util.Collection<? extends E>)

Example: TestSet.java

Methods of HashSet
NOTE: HashSet implements Set interface and provide implementation of Set Methods

Methods of LinkedHashSet
Apart from HashSet methods LinkedHashSet provides following methods:
public void addFirst(E)
public void addLast(E)
public E getFirst()
public E getLast()
public E removeFirst()
public E removeLast()
public java.util.SequencedSet<E> reversed()

Methods of NavigableSet
E ceiling(E element) // Returns the least element >= E element,else null
E floor(E element) // Returns the least element <= E element,else null
Iterator<E> descendingIterator() // Returns iterator in descending order

```
NavigableSet<E> descendingSet ()                          // Returns NavigableSet in descending order
SortedSet<E> headSet(E thisElement)                       // Returns a view of this set elements < thisElement
NavigableSet<E> headSet(E thisElement, boolean incl)    // Returns a view of this set elements < or <= thisElement
SortedSet<E> tailSet(E thisElement)                     // Returns a view of this set elements >= thisElement
NavigableSet<E> tailSet(E thisElement, boolean incl)    // Returns a view of this set elements > or >= thisElement
SortedSet<E> subSet(E startElement, E endElement)       // Returns a view of this set elements > and <= element
NavigableSet<E> subSet(E startElement, boolean startIncl, E endElement, boolean endIncl)
                                                          // Returns a view of this set elements > and < element
E higher(E element)                                       //> or null
E lower(E element)                                        //< or null
Iterator<E> iterator()                                    //Returns iterator in ascending order
E pollFirst()                                             //Retrieves and removes the first(lowest)
E pollLast()                                              //Retrieves and removes the last(highest)
NavigableSet<E> reversed()                                //Returns reversed set
```

Example: TreeSetRep.java
Example: NavigableSetRep.java

Methods of Map interface

```
public abstract int size()
public abstract boolean isEmpty()
public abstract boolean containsKey(java.lang.Object)
public abstract boolean containsValue(java.lang.Object)
public abstract V get(java.lang.Object)
public abstract V put(K, V)
public abstract V remove(java.lang.Object)
public abstract void putAll(java.util.Map<? extends K, ? extends V>)
public abstract void clear()
public abstract java.util.Set<K> keySet()
public abstract java.util.Collection<V> values()
public abstract java.util.Set<java.util.Map$Entry<K, V>> entrySet()
public abstract boolean equals(java.lang.Object)
public abstract int hashCode()
public V getOrDefault(java.lang.Object, V)
public void forEach(java.util.function.BiConsumer<? super K, ? super V>)
public void replaceAll(java.util.function.BiFunction<? super K, ? super V, ? extends V>)
public V putIfAbsent(K, V)
public boolean remove(java.lang.Object, java.lang.Object)
public boolean replace(K, V, V)
public V replace(K, V)
```

Example: TestMap.java

Methods of NavigableMap

```
Map.Entry<K,V> ceilingEntry(K key)          >=, null
Map.Entry<K,V> higherEntry(K key)           >, null
K floorKey(K key)                           <=, null
K lowerKey(K key)                           <, null
Map.Entry<K,V> pollFirstEntry()             // Removes and Returns First
```

Map.Entry<K,V> pollLastEntry() // Removes and Returns Last
SortedMap<K,V> subMap(K startKey, K endKey) // startKey>= and < endKey
NavigableMap<K,V> descendingMap()

Methods of TreeMap

SortedMap<K,V> headMap(K endKey) //less than the key
SortedMap<K,V> tailMap(K endKey, boolean include) //> the key
SortedMap<K,V> subMap(K startKey, boolean include, K endKey, boolean include) //> the key and < the key

Example: NavigableMapRep.java
Example: SubMapRep.java

Set 1: Factory Machine Management System

You are developing a **Factory Machine Management System** to manage unique machine IDs in a factory. The system should use a **HashSet** to store machine IDs, ensuring uniqueness. Your task is to implement a **menu-driven program** that allows users to perform various operations on the machine ID set.

Menu Options:

1. **Add Machine IDs**

- The user can input multiple unique machine IDs and store them in the HashSet.
- If a duplicate ID is entered, it should not be added.
- The system should display:

Machine IDs added successfully.

2. **Check for a Machine ID**

- The user enters a machine ID to check whether it exists in the HashSet.
- Expected output:
    - SetIf the machine ID is found:

The machine ID [ID] is present in the HashSet.

    - If the machine ID is not found:

The machine ID [ID] is not present in the HashSet.

3. **Delete a Machine ID**

- The user enters a machine ID for deletion.
- Expected output:
    - If the machine ID exists and is removed:

The machine ID [ID] was removed from the HashSet.

    - If the machine ID does not exist:

The machine ID [ID] was not found in the HashSet.

4. **Display the Updated List of Machine IDs**

- The program should display the remaining machine IDs stored in the HashSet.
- Expected output:

Updated list of machine IDs:

Machine ID: [ID1]

Machine ID: [ID2]

...

- If the HashSet is empty, display:

No machine IDs available.

5. **Count Odd Machine IDs**

- The program counts how many stored machine IDs are odd numbers.
- Expected output:

Number of odd machine IDs: [count]

6. **Exit the Program**

- The program should terminate when the user selects this option.
- Expected output:

Exiting the Factory Machine Management System. Goodbye!

Set 2: Vehicle Parking Management System

You are developing a **Vehicle Parking Management System** for a parking lot that tracks the unique license plate numbers of parked vehicles. The system should use a **HashSet** to store license plate numbers, ensuring no duplicate entries. Implement a **menu-driven program** to perform the following operations.

Menu Options:

1. **Park a Vehicle (Add License Plate)**

- The user enters a **license plate number** to park a vehicle.
- If the vehicle is already parked (duplicate entry), it should not be added.
- The system should display:

Vehicle with license plate [LICENSE_PLATE] parked successfully.

- If it's a duplicate:

Vehicle with license plate [LICENSE_PLATE] is already parked.

2. **Check if a Vehicle is Parked**

- The user enters a license plate number to check if the vehicle is in the parking lot.
- Expected output:
    - If the vehicle is parked:

The vehicle with license plate [LICENSE_PLATE] is parked in the lot.

    - If the vehicle is **not found**:

The vehicle with license plate [LICENSE_PLATE] is not in the lot.

3. **Remove a Vehicle (Unpark)**

- The user enters a license plate number to remove the vehicle from the parking lot.
- Expected output:

- If the vehicle is found and removed:

The vehicle with license plate [LICENSE_PLATE] has left the parking lot.

- If the vehicle is not found:

The vehicle with license plate [LICENSE_PLATE] was not found in the parking lot.

4. **Display All Parked Vehicles**

- The program should display **all parked vehicles** stored in the HashSet.
- Expected output:

Currently parked vehicles:

License Plate: [LICENSE_PLATE_1]

License Plate: [LICENSE_PLATE_2]

...

- If the HashSet is empty, display:

No vehicles are parked in the lot.

5. **Count Vehicles with Odd-Ending License Plates**

- The program counts how many parked vehicle license plates end with an **odd digit**.
- Expected output:

Number of vehicles with odd-ending license plates: [COUNT]

6. **Exit the Program**

- The program should terminate when the user selects this option.
- Expected output:

Exiting the Vehicle Parking Management System. Goodbye!

Set 3: Employee Attendance Tracking System

You are developing an **Employee Attendance Tracking System** for an office to manage unique employee ID numbers of employees who check in daily. The system should use a **HashSet** to store employee IDs, ensuring no duplicates. Implement a **menu-driven program** to perform the following operations.

Menu Options:

1. **Mark Employee Attendance (Check-In)**

- The user enters an **employee ID number** to mark attendance for the day.
- If the employee has already checked in, the system should not allow duplicate entries.
- Expected output:

Employee ID [ID] has checked in successfully.

- If duplicate entry:

Employee ID [ID] has already checked in today.

2. **Check If an Employee Checked In**

- The user enters an employee ID to verify if they have checked in for the day.
- Expected output:
    - If the employee is found:

Employee ID [ID] has checked in today.

    - If the employee is not found:

Employee ID [ID] has not checked in today.

3. **Remove Employee from Attendance (Check-Out)**

- The user enters an employee ID to **remove their attendance record** (indicating they checked out).
- Expected output:
    - If the employee is found and removed:

Employee ID [ID] has checked out.

    - If the employee is not found:

Employee ID [ID] was not found in the attendance list.

4. **Display All Employees Who Checked In**

- The program should display **all checked-in employees** stored in the HashSet.
- Expected output:

Employees who checked in today:

Employee ID: [ID_1]

Employee ID: [ID_2]

...

- If the HashSet is empty, display:

No employees have checked in today.

5. **Count Employees with Even Employee IDs**

- The program counts how many checked-in employees have an **even employee ID number**.
- Expected output:

Number of employees with even ID numbers: [COUNT]

6. **Exit the Program**

- The program should terminate when the user selects this option.
- Expected output:

Exiting the Employee Attendance Tracking System. Goodbye!

Set 4: Smart Home Device Management

Develop a **Smart Home Device Management System** to manage unique smart devices in a house. Each device is represented by a custom Device class with attributes like deviceID, deviceName, and deviceType (e.g., thermostat, light, security camera).

Menu Options:

1. **Add Devices**
   • Add multiple unique smart devices by entering their deviceID, deviceName, and deviceType.
   • Duplicate devices (same deviceID) are not allowed.
   • Output: *Devices added successfully.*
2. **Check for a Device**
   • Enter the deviceID to check if the device exists in the HashSet.
   • Output:
   o If found: *The device with ID [deviceID] is present in the HashSet.*
   o If not found: *The device with ID [deviceID] is not present in the HashSet.*
3. **Delete a Device**
   • Remove a device by entering its deviceID.
   • Output:
   o If found and removed: *The device with ID [deviceID] was removed from the HashSet.*
   o If not found: *The device with ID [deviceID] was not found in the HashSet.*
4. **Display All Devices**
   • Show the details (deviceID, deviceName, deviceType) of all devices in the HashSet.
   • If empty, display: *No devices available.*
5. **Count Devices by Type**
   • Count the number of devices for a given deviceType (e.g., lights).
   • Output: *Number of [deviceType] devices: [count].*
6. **Exit**
   • Gracefully terminate the program.

Set 5: Wildlife Conservation Tracking

Create a **Wildlife Conservation Tracking System** to manage the details of unique animals being monitored in a wildlife sanctuary. Use a WildlifeAnimal custom class with attributes such as animalID, speciesName, and habitatType (e.g., forest, grassland, wetland).

Menu Options:

1. **Add Animals**
   • Add multiple unique animals by specifying their animalID, speciesName, and habitatType.
   • Ensure animals with duplicate animalID are not added.
   • Output: *Animals added successfully.*
2. **Check for an Animal**
   • Enter the animalID to verify whether the animal exists in the HashSet.
   • Output:

o If found: *The animal with ID [animalID] is present in the HashSet.*
o If not found: *The animal with ID [animalID] is not present in the HashSet.*

3. **Delete an Animal**
   • Remove an animal from the HashSet using its animalID.
   • Output:
   o If found and removed: *The animal with ID [animalID] was removed from the HashSet.*
   o If not found: *The animal with ID [animalID] was not found in the HashSet.*

4. **Display All Animals**
   • Show the details (animalID, speciesName, habitatType) of all animals.
   • If the HashSet is empty, display: *No animals available.*

5. **Count Animals by Habitat**
   • Count how many animals are located in a specified habitatType (e.g., wetland).
   • Output: *Number of animals in [habitatType]: [count].*

6. **Exit**
   • Gracefully terminate the program.

Set 6: E-Commerce Product Inventory

Build a **Product Inventory System** for an e-commerce platform, managing unique product objects. Use a custom class Product with attributes like productID, productName, and category (e.g., electronics, clothing).

Menu Options:

1. **Add Products**
   • Add unique products to the inventory using their productID, productName, and category.
   • Prevent duplicates based on productID.
   • Output: *Products added successfully.*

2. **Check for a Product**
   • Enter the productID to check if the product exists.
   • Output:
   o If found: *The product with ID [productID] is present in the inventory.*
   o If not found: *The product with ID [productID] is not present in the inventory.*

3. **Delete a Product**
   • Remove a product from the inventory using its productID.
   • Output:
   o If found and removed: *The product with ID [productID] was removed from the inventory.*
   o If not found: *The product with ID [productID] was not found in the inventory.*

4. **Display All Products**
   • Show details (productID, productName, category) of all available products.
   • If the inventory is empty, display: *No products available.*

5. **Count Products by Category**
   • Count the products in a given category (e.g., electronics).
   • Output: *Number of products in [category]: [count].*

6. **Exit**
   • Exit the program gracefully.

Map 1: Advertising Campaign Budget Management System

You are developing an **Advertising Campaign Budget Management System** to track **unique advertising campaigns** using their **IDs** and **budgets**. The system should use a **HashMap** to store campaign IDs as keys and their corresponding budgets as values, ensuring uniqueness. Implement a **menu-driven program** to allow users to manage advertising campaigns efficiently.

Menu Options:

1. **Add Advertising Campaigns**

- The user can enter multiple **unique campaign IDs** and their corresponding **budgets** to store in the system.
- If a **duplicate campaign ID** is entered, it should not be added.
- Expected output:

Campaigns added successfully.

2. **Check if a Campaign Exists**

- The user enters a **campaign ID** to check whether it exists in the system.
- Expected output:
    - If the campaign is found:

The campaign ID [ID] is present in the system.

    - If the campaign is not found:

The campaign ID [ID] is not present in the system.

3. **Remove an Advertising Campaign**

- The user enters a **campaign ID** to remove it from the system.
- Expected output:
    - If the campaign exists and is removed:

The campaign ID [ID] was removed from the system.

    - If the campaign is not found:

The campaign ID [ID] was not found in the system.

4. **Display the Updated List of Advertising Campaigns**

- The program should display **all stored advertising campaigns** along with their **budgets**.
- Expected output:

Updated list of campaigns:

Campaign ID: [ID1], Budget: [BUDGET1]

Campaign ID: [ID2], Budget: [BUDGET2]

...

- If no campaigns are present, display:

No campaigns available in the system.

5. **Calculate Average Campaign Budget**

- The program calculates and displays the **average budget** of all stored campaigns, rounded to **two decimal places**.
- Expected output:

Average Budget of All Campaigns: [average]

- If no campaigns are stored, display:

No campaigns available to calculate the average budget.

6. **Exit the Program**

- The program should terminate when the user selects this option.
- Expected output:

Exiting the Advertising Campaign Budget Management System. Goodbye!

Map 2: Digital Marketing Ad Performance Tracker

You are developing a **Digital Marketing Ad Performance Tracker** to monitor the **performance of different ads** based on their unique Ad IDs and the number of clicks they receive. The system should use a **TreeMap** to store Ad IDs as **keys** and their corresponding click counts as **values**, ensuring sorted order based on Ad IDs. Implement a **menu-driven program** that allows users to efficiently manage and analyze ad performance.

Menu Options:

1. **Register New Ads**

- The user can enter multiple **unique Ad IDs** and their corresponding **initial click counts**.
- If a **duplicate Ad ID** is entered, it should not be added.
- Expected output:

Ads registered successfully.

2. **Check Ad Performance**

- The user enters an **Ad ID** to check whether it exists and retrieve its click count.
- Expected output:
    - If the Ad is found:

The Ad ID [ID] has [click_count] clicks.

    - If the Ad is not found:

The Ad ID [ID] is not present in the system.

3. **Remove an Ad Record**

- The user enters an **Ad ID** to remove its record from the system.

- Expected output:
    - If the Ad exists and is removed:

The Ad ID [ID] was removed from the system.

    - If the Ad is not found:

The Ad ID [ID] was not found in the system.

4. **Update Click Count for an Ad**

- The user enters an **Ad ID** and provides a **new click count** to update its performance.
- Expected output:
    - If the Ad exists:

The Ad ID [ID] click count updated to [new_click_count].

    - If the Ad is not found:

The Ad ID [ID] was not found in the system.

5. **Display Ads Sorted by ID**

- The program should display **all stored Ad IDs** in **sorted order**, along with their **click counts**.
- Expected output:

Ad Performance Summary:

Ad ID: [ID1], Clicks: [CLICK1]

Ad ID: [ID2], Clicks: [CLICK2]

...

- If no ads are present, display:

No ad records available in the system.

6. **Find the Most Popular Ad**

- The program finds and displays the **Ad ID with the highest click count**.
- Expected output:

The most popular Ad is [ID] with [click_count] clicks.

- If no ads are present, display:

No ads available to determine the most popular one.

7. **Exit the Program**

- The program should terminate when the user selects this option.
- Expected output:

Exiting the Digital Marketing Ad Performance Tracker. Goodbye!

Map 3: Employee Overtime Pay Management System

You are developing an **Employee Overtime Pay Management System** to track **overtime hours and payments** for employees. The system should use a **LinkedHashMap** to store Employee IDs as **keys** and the corresponding **overtime hours** as **values**, preserving insertion order. Implement a **menu-driven program** that helps HR efficiently manage overtime records.

Menu Options:

1. **Record Overtime for Employees**

- The user can enter multiple **unique Employee IDs** and their **overtime hours**.
- If a **duplicate Employee ID** is entered, it should not be added.
- Expected output:

Overtime records added successfully.

2. **Check Overtime for an Employee**

- The user enters an **Employee ID** to check their **overtime hours**.
- Expected output:
    - If the Employee is found:

The Employee ID [ID] has worked [hours] overtime hours.

    - If the Employee is not found:

The Employee ID [ID] is not recorded in the system.

3. **Remove an Employee's Overtime Record**

- The user enters an **Employee ID** to remove their overtime record from the system.
- Expected output:
    - If the Employee exists and is removed:

The Employee ID [ID] overtime record was removed from the system.

    - If the Employee is not found:

The Employee ID [ID] was not found in the system.

4. **Update Overtime Hours for an Employee**

- The user enters an **Employee ID** and provides a **new overtime hour count** to update their record.
- Expected output:
    - If the Employee exists:

The Employee ID [ID] overtime hours updated to [new_hours].

    - If the Employee is not found:

The Employee ID [ID] was not found in the system.

5. **Display Overtime Records in Order of Entry**

- The program should display **all stored Employee IDs** in **insertion order**, along with their **overtime hours**.
- Expected output:

Overtime Records:

Employee ID: [ID1], Overtime Hours: [HOURS1]

Employee ID: [ID2], Overtime Hours: [HOURS2]

...

- If no records are present, display:

No overtime records available in the system.

6. **Calculate Total Overtime Payout**

- The program calculates and displays the **total overtime pay**, assuming **each hour is paid at a fixed rate** (e.g., $20 per hour).
- Expected output:

Total Overtime Payout: $[total_payout]

- If no overtime is recorded, display:

No overtime records available to calculate payout.

7. **Exit the Program**

- The program should terminate when the user selects this option.
- Expected output:

Exiting the Employee Overtime Pay Management System. Goodbye!

Map 4: Library Book Management System

You are developing a **Library Book Management System** to track books in a library. Each book is uniquely identified by its **ISBN (International Standard Book Number)** and contains **detailed information** such as **title, author, and price**. The system should use a **HashMap**, where the **key is the ISBN (String)** and the **value is a custom class Book** that stores book details.

Custom Class:

```
class Book {
   String title;
   String author;
   double price;
}
```

Menu Options:

1. **Add New Books**

- The user enters multiple **ISBNs** along with their **title, author, and price** to add new books.
- If an ISBN already exists, the book should not be added.
- Expected output:

Books added successfully.

2. **Check if a Book Exists**

- The user enters an **ISBN** to check if it exists in the library.
- Expected output:
    - If found:

The book with ISBN [ISBN] exists in the library.

    - If not found:

The book with ISBN [ISBN] is not available in the library.

3. **Remove a Book from the Library**

- The user enters an **ISBN** to remove the corresponding book from the system.
- Expected output:
    - If found and removed:

The book with ISBN [ISBN] was removed from the library.

    - If not found:

The book with ISBN [ISBN] was not found in the library.

4. **Update Book Price**

- The user enters an **ISBN** and a **new price** to update the book's price.
- Expected output:
    - If found:

The price for book [Title] has been updated to $[New Price].

    - If not found:

The book with ISBN [ISBN] was not found in the library.

5. **Display All Books**

- The program should display **all books** in the library along with their details.
- Expected output:

Library Book List:

ISBN: [ISBN1], Title: [Title1], Author: [Author1], Price: $[Price1]

ISBN: [ISBN2], Title: [Title2], Author: [Author2], Price: $[Price2]

...

- If no books are available, display:

No books are available in the library.

6. **Find the Most Expensive Book**

- The program finds and displays the book with the **highest price**.
- Expected output:

The most expensive book is [Title] by [Author] with a price of $[Price].

- If no books exist, display:

No books available to determine the most expensive book.

7. **Exit the Program**

- The program should terminate when the user selects this option.
- Expected output:

Exiting the Library Book Management System. Goodbye!

Map 5: Vehicle Registration Management System

You are developing a **Vehicle Registration Management System** to store vehicle details. Each vehicle is uniquely identified by its **registration number** and contains **information like make, model, and owner name**. The system should use a **TreeMap**, where the **key is the Registration Number (String)** and the **value is a custom class Vehicle** that holds vehicle details.

Custom Class:

```
class Vehicle {
   String make;
   String model;
   String owner;
}
```

Menu Options:

1. **Register New Vehicles**

- The user enters multiple **registration numbers** along with their **make, model, and owner name**.
- If a duplicate registration number is entered, it should not be added.
- Expected output:

Vehicles registered successfully.

2. **Check Vehicle Ownership**

- The user enters a **registration number** to check if the vehicle exists.

- Expected output:
    - If found:

The vehicle with registration number [RegNo] belongs to [Owner].

    - If not found:

No vehicle found with registration number [RegNo].

3. **Remove a Vehicle from Records**

- The user enters a **registration number** to remove the vehicle from the system.
- Expected output:

The vehicle with registration number [RegNo] was removed from the system.

4. **Update Owner of a Vehicle**

- The user enters a **registration number** and a **new owner name**.
- Expected output:

Ownership of vehicle [RegNo] updated to [New Owner].

5. **Display All Vehicles (Sorted Order)**

- The program should display **all registered vehicles** in **sorted order** based on **registration number**.
- Expected output:

Registered Vehicles:

Reg No: [RegNo1], Make: [Make1], Model: [Model1], Owner: [Owner1]

Reg No: [RegNo2], Make: [Make2], Model: [Model2], Owner: [Owner2]

6. **Find Vehicle by Owner**

- The user enters an **owner name** to find all vehicles owned by that person.
- Expected output:

Vehicles owned by [Owner]:

Reg No: [RegNo1], Make: [Make1], Model: [Model1]

- If no vehicles are found, display:

No vehicles found for owner [Owner].

7. **Exit**

Exiting the Vehicle Registration Management System. Goodbye!

Map 6: Employee Payroll Management System

You are developing an **Employee Payroll Management System** to store employee payroll details. Each employee is uniquely identified by their **Employee ID**, and the system stores **salary, department, and job title**. The system should use a **LinkedHashMap**, where the **key is Employee ID (Integer)** and the **value is a custom class Employee** that holds salary details.

Custom Class:

```
class Employee {
   double salary;
   String department;
   String jobTitle;
}
```

Menu Options:

1. **Add Employee Payroll Data**

Employee payroll records added successfully.

2. **Check Employee Salary**

Employee [ID] earns $[Salary].

3. **Remove Employee Payroll Record**

Employee ID [ID] payroll record removed.

4. **Update Salary**

Salary for Employee [ID] updated to $[New Salary].

5. **Display All Employees (Order of Entry)**

Employee Payroll Records:

ID: [ID1], Salary: $[Salary1], Dept: [Dept1], Job Title: [Title1]

6. **Find Employees by Department**

Employees in [Department]:

ID: [ID1], Job Title: [Title1], Salary: $[Salary1]

7. **Exit**

Exiting Employee Payroll Management System. Goodbye!