Introduction to Programming

Lab 11

Alaa Aldin Hajjar, Mahmoud Naderi, Marko Pezer,
Mosab Mohamed, Rawan Ali

Agenda

- Enumerations in Java
- UML Class Diagram
- Java Collections Framework

Enums in Java

The enum is a special "class" that represents a group of constants (unchangeable variables, like **final** variables).

```java
enum Level {
  LOW,
  MEDIUM,
  HIGH
}
```

```java
public class Main {
  public static void main(String[] args) {
    Level myVar = Level.MEDIUM;

    switch(myVar) {
      case LOW:
        System.out.println("Low level");
        break;
      case MEDIUM:
         System.out.println("Medium level");
        break;
      case HIGH:
        System.out.println("High level");
        break;
    }
  }
}
```

Enums in Java: Fields & Methods

```java
public enum Level {
   HIGH  (3),    //calls constructor with value 3
   MEDIUM(2), //calls constructor with value 2
   LOW   (1)    //calls constructor with value 1
   ; // semicolon needed when fields / methods follow

   private final int levelCode;

   Level(int levelCode) {
       this.levelCode = levelCode;
   }

   public int getLevelCode() {
       return this.levelCode;
   }
}
```

```java
public enum Level {
   HIGH {
       @Override
       public String asLowerCase() {
           return HIGH.toString().toLowerCase();
       }
   },
   MEDIUM {
       @Override
       public String asLowerCase() {
           return MEDIUM.toString().toLowerCase();
       }
   },
   LOW {
       @Override
       public String asLowerCase() {
           return LOW.toString().toLowerCase();
       }
   };
   public abstract String asLowerCase();
}

```

Enums in Java

```
enum Color {
    RED,
    GREEN,
    BLUE;
    private Color()
    {
            System.out.println("Constructor called for : " + this.toString());
    }
    public void colorInfo()
    {
            System.out.println("Universal Color");
    }
}
public class Test {
    public static void main(String[] args) {
                    Color c1 = Color.RED;
                    System.out.println(c1);
                    c1.colorInfo();
            }
}
```

Output:

```
Constructor called for : RED
Constructor called for : GREEN
Constructor called for : BLUE
RED
Universal Color
```

Exercise 1: Enums

- Write a simple *Vending Machine* program, which allows money insertion and buying a single drink and returning the money (unlimited in machine). Before money insertion, the *Vending Machine* should show the menu with prices
- Create enum *Drinks* with beverage drinks (Coke Cola, Sprite, Fanta) with parameters *name* and *price*. Create enum *Money* with applicable banknotes with parameter *denomination*. Assume that coins cannot be used
- Handle exceptions if needed (not enough money, negative values, etc.), if familiar with exceptions. Otherwise, use error messages
- Assume that if the *Vending Machine* cannot return the money, because of missing such a banknote in *Money* enum, it will return banknote with the closest lesser nomination, *e.g.* instead of 5.5$ the customer will be returned 5$. Provide adequate interaction with the customer

Exercise 2: Hospital Management System

We want to implement a **hospital management system** where we can manage appointments, bills, patients and doctors.

The **users** in this system can choose from the main menu what type of user they are, depending on that, they can make some actions.

We want to keep track of the bills. A **bill** is defined by a unique **ID**, it has a **name** and an **amount**.

Don’t forget to **Draw UML diagram first then implement it**

Exercise 2: Hospital Management System

We have three types of users:

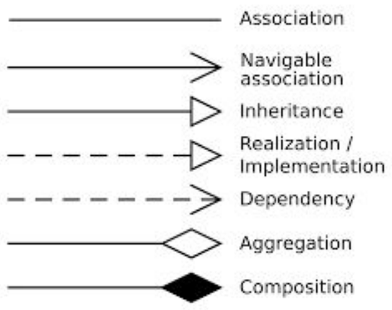
- **Patient**: this user is identified by a unique **ID** and have a **name**. They can pay the bill. A bill **belongs** to a patient and each patient has **one** bill.

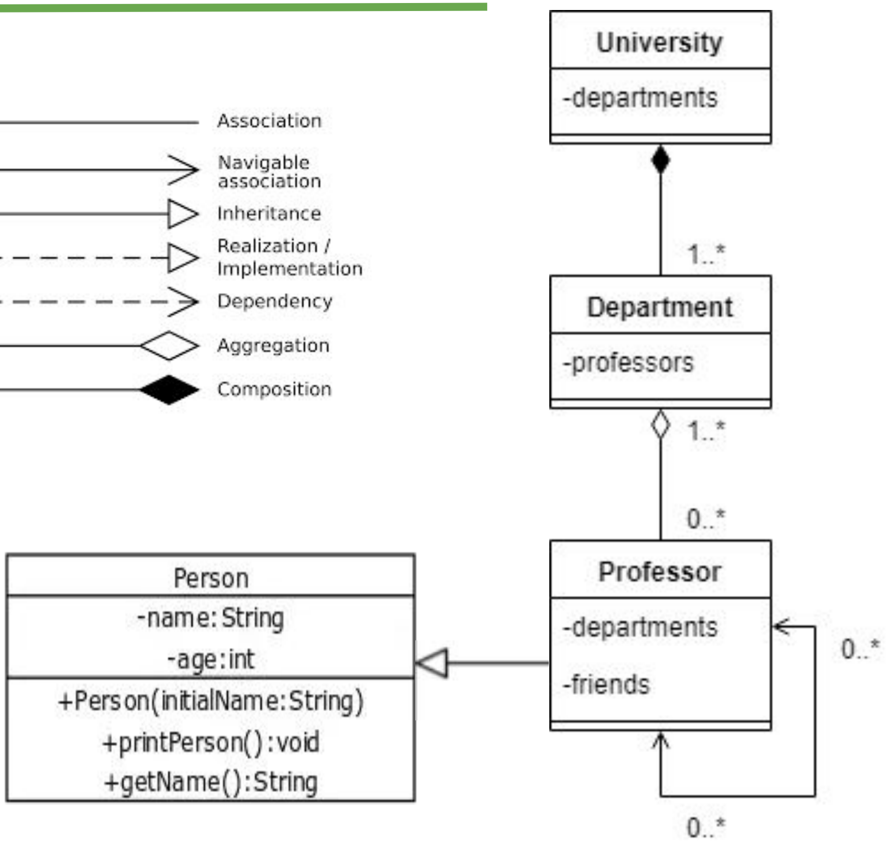
- **Receptionist**: this user can give appointments to as **many** patients as they want. A patient gets an appointment from **one** receptionist.
  The receptionist can also generate **bills**. A bill is generated by **one** receptionist.

- **Doctor**: this user can **check** as **many** patients as he wants, a patient is checked by **one** doctor.

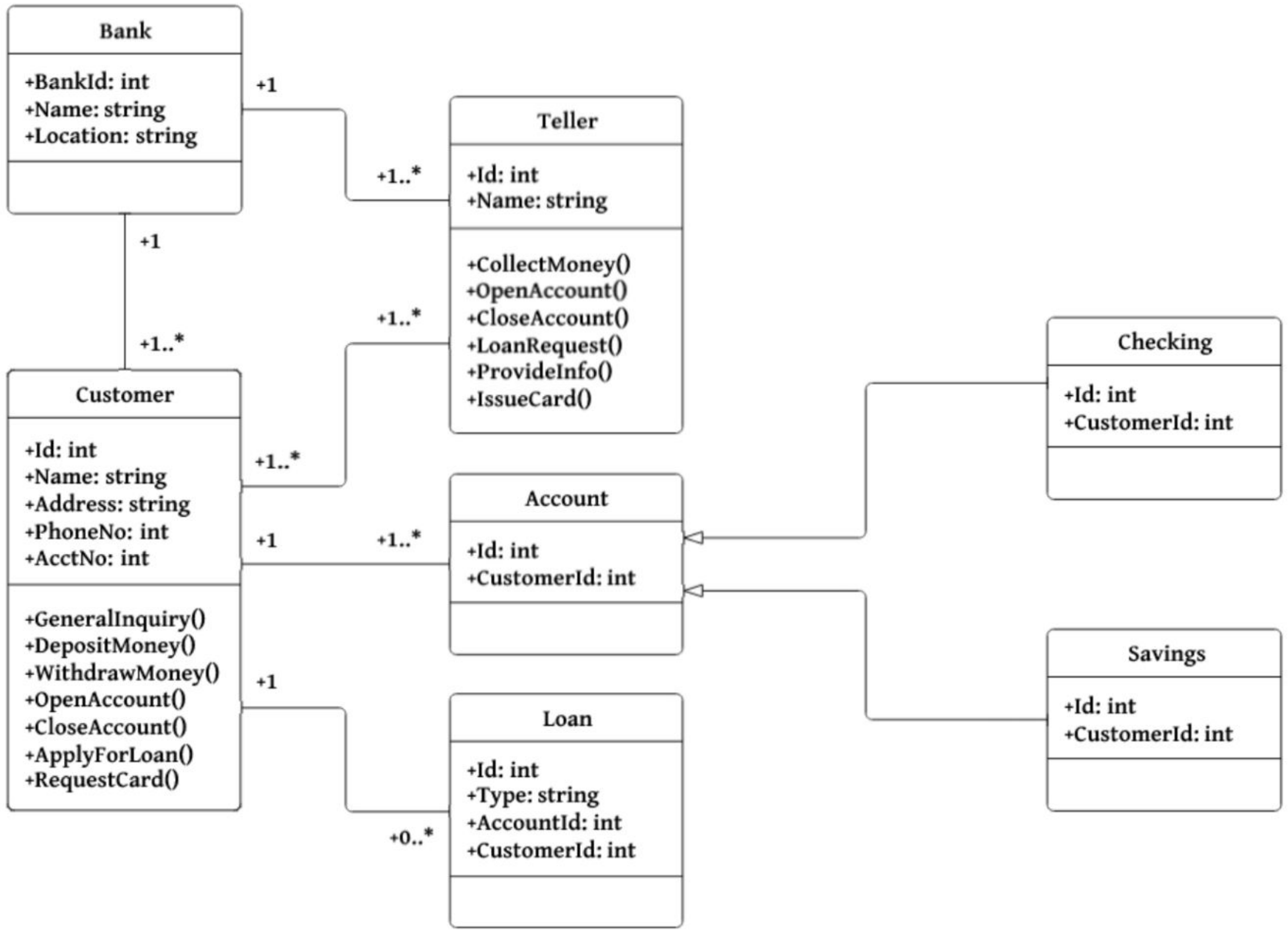
UML - Class diagram simple example

```
public class Person {
    private String name;
    private int age;

    // constructor
    // other methods
}

class University {
    List<Department> department;
}

class Department {
    List<Professor> professors;
}

class Professor extends Person {
    List<Department> department;
    List<Professor> friends;
}
```





UML


Bank
+BankId: int
+Name: string
+Location: string
+1
+1..*
+1
+1..*
Teller
+Id: int
+Name: string
+CollectMoney()
+OpenAccount()
+CloseAccount()
+LoanRequest()
+ProvideInfo()
+IssueCard()
+1..*
+1..*
Customer
+Id: int
+Name: string
+Address: string
+PhoneNo: int
+AcctNo: int
+GeneralInquiry()
+DepositMoney()
+WithdrawMoney()
+OpenAccount()
+CloseAccount()
+ApplyForLoan()
+RequestCard()
+1
+1..*
Account
+Id: int
+CustomerId: int
Checking
+Id: int
+CustomerId: int
Savings
+Id: int
+CustomerId: int
+1
+0..*
Loan
+Id: int
+Type: string
+AccountId: int
+CustomerId: int

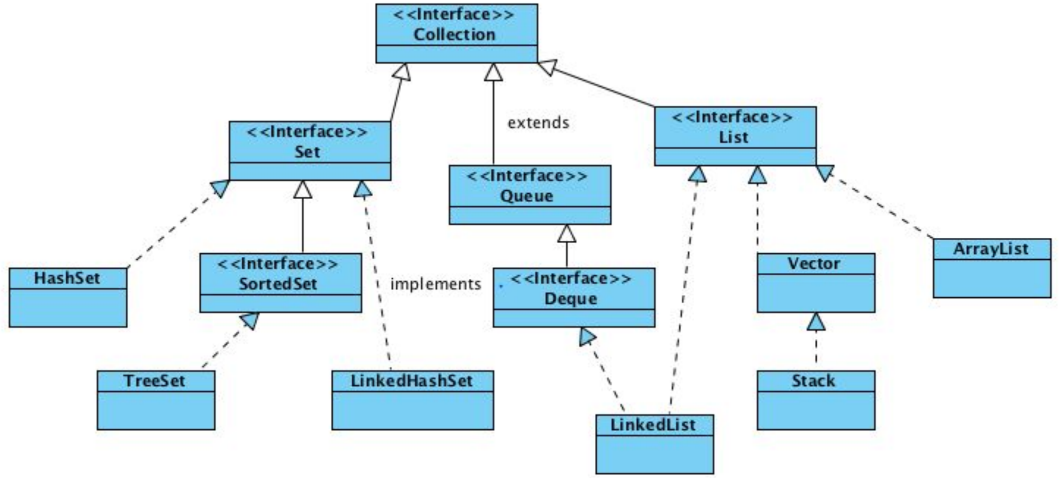
Collections

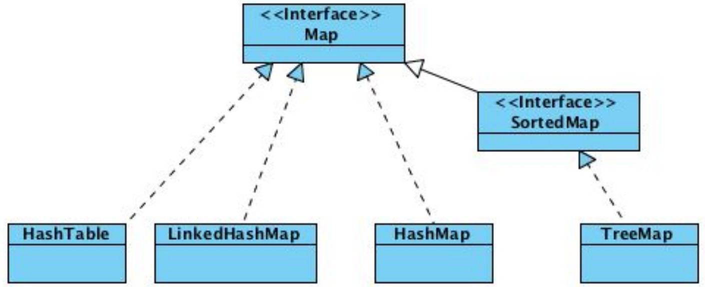
1. **List:**
   An ordered list of objects, which are stored in the order in which they are added to the list. The elements of the list are accessed by index

2. **Set:**
   A set of non-repeating objects. Only one null reference is allowed in a collection of this type

3. **Map:**
   Map is used to map each element from one set of objects (keys) to another (values). In this case, each element from the set of keys is assigned a set of values. At the same time, one element from a set of values can correspond to 1, 2 or more elements from a set of keys

Collections Hierarchy





List Template

```
interface ListADT<E>{
    int size();
    void clear();
    boolean isEmpty();
    boolean add(E e);
    boolean remove(E o);

    E get(int index);
    E set(int index, E element);
    void add(int index, E element);
    E remove(int index);
}
```

List Example

```
// Importing all utility classes
import java.util.*;

// Main class
class GFG {

        // Main driver method
        public static void main(String args[])
        {
                // Creating an object of List interface,
                // implemented by ArrayList class
                List<String> al = new ArrayList<>();

                // Adding elements to object of List interface
                // Custom elements
                al.add("Geeks");
                al.add("Geeks");
                al.add(1, "For");

                // Print all the elements inside the
                // List interface object
                System.out.println(al);
        }
}
```

Exercise 3 (List)

Write a program to create a List of animals. Create 4 methods: adding, removing, updating and displaying the animals

Set Template

```
interface SetADT<E> {
    int size();
    void clear();
    boolean isEmpty();

    boolean add(E e);
    boolean remove(E o);
}
```

Set Example

```java
// Importing utility classes
import java.util.*;

// Main class
public class GFG {

    // Main driver method
    public static void main(String[] args)
    {
        // Demonstrating Set using HashSet
        // Declaring object of type String
        Set<String> hash_Set = new HashSet<String>();

        // Adding elements to the Set
        // using add() method
        hash_Set.add("Geeks");
        hash_Set.add("For");
        hash_Set.add("Geeks");
        hash_Set.add("Example");
        hash_Set.add("Set");

        // Printing elements of HashSet object
        System.out.println(hash_Set);
    }
}
```

Exercise 4 (Set)

Write a program program that creates a Set with Strings and then removes all elements of the Set with the odd length and leaves elements with even length

Map Template

```
interface MapADT<K,V> {
    int size();
    void clear();
    boolean isEmpty();

    V get(K key);
    V put(K key, V value);
    V remove(K key);
}
```

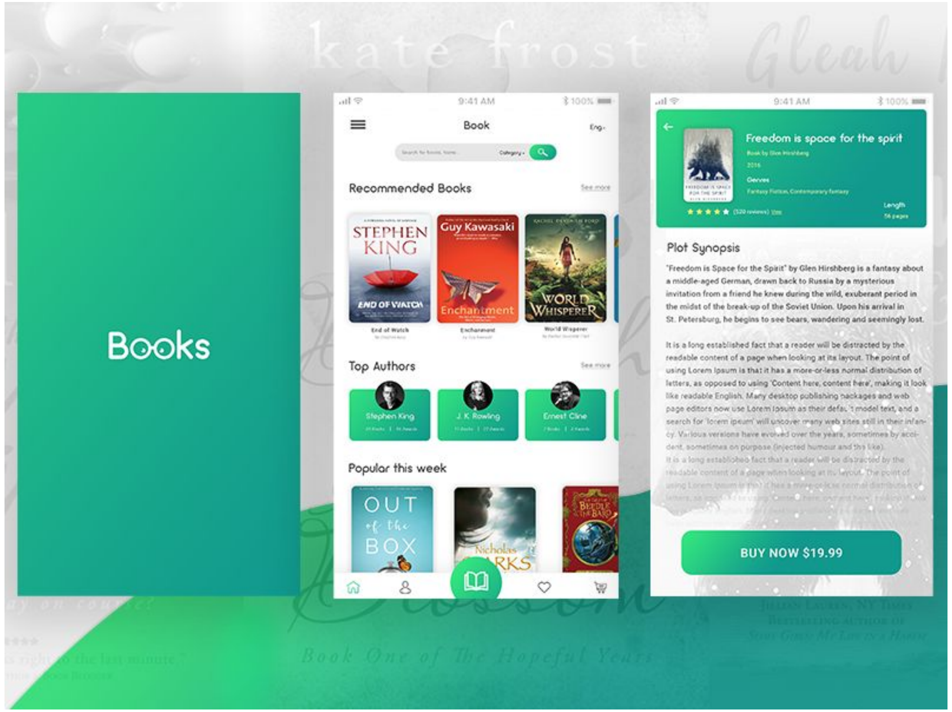
Map Example

```java
// Java Program to Demonstrate Working of Map interface
import java.util.*;
class GFG {
    // Main driver method
    public static void main(String args[])
    {
        // Creating an empty HashMap
        Map<String, Integer> hm = new HashMap<String, Integer>();

        // Inserting pairs in above Map
        // using put() method
        hm.put("a", new Integer(100));
        hm.put("b", new Integer(200));
        hm.put("c", new Integer(300));
        hm.put("d", new Integer(400));

        // Traversing through Map using for-each loop
        for (Map.Entry<String, Integer> me : hm.entrySet()) {
            // Printing keys
            System.out.print(me.getKey() + ":");
            System.out.println(me.getValue());
        }
    }
}
```

Exercise 5 (Map)

Write a program which creates the Map<String, Integer> and then reports if user input contains repetitive values (and their count) or not

Exercise 3 (Homework): Online Book Reader



"Freedom is Space for the Spirit" by Glen Hirshberg is a fantasy about a middle-aged German, drawn back to Russia by a mysterious invitation from a friend he knew during the wild, exuberant period in the midst of the break-up of the Soviet Union. Upon his arrival in St. Petersburg, he begins to see bears, wandering and seemingly lost.

It is a long established fact that a reader will be distracted by the readable content of a page when looking at its layout. The point of using Lorem Ipsum is that it has a more-or-less normal distribution of letters, as opposed to using 'Content here, content here', making it look like readable English. Many desktop publishing packages and web page editors now use Lorem Ipsum as their default model text, and a search for 'lorem ipsum' will uncover many web sites still in their infancy. Various versions have evolved over the years, sometimes by accident, sometimes on purpose (injected humour and the like).

Asked in Amazon, Microsoft, and many more interviews

Exercise 6 (Homework): Online Book Reader

Hint: Let’s assume we want to design a basic online reading system which provides the following functionality:

- Searching the database of books and reading a book.
- User membership creation and extension.
- Only one active user at a time and only one active book by this user

The class OnlineReaderSystem represents the body of our program. We could implement the class such that it stores information about all the books, deals with user management, and refreshes the display, but that would make this class rather hefty. Instead, we’ve chosen to tear off these components into Library, UserManager, and Display classes.

Exercise 6 (Homework): Online Book Reader

- First try to design the logic with UML.
- Then start coding…

References

- Overview of Inheritance, Interfaces and Abstract Classes in Java | by Isaac Jumba | Medium
- Polymorphism in Java
- List Interface in Java with Examples
- Set Interface in Java
- Map Interface in Java
- Set (Java Platform SE 7)
- Map (Java Platform SE 8)
- Java Iterator
- Java - How to Use Iterator?
- Interface vs abstract classes
- Interface vs abstract classes