Fall-2024

1.a) Aggregation: It is a relationship where one class uses another class, but both can exist independently.

Example: A Car has an Engine, but the engine can exists without the car.

```
class Engine {
        void start () {
            System.out.println ("Engine starts");
        }
    }
class Car {
    Engine engine = new Engine (); // Aggre-
                                      gation.

    void startCar () {
        engine.start ();
    }
}
```

Composition: It is a stronger form of association where one class contains another class as part of itself, and the contained class can not exist independently. (একটা ক্লাস আরেকটা ক্লাসকে নিজের অংশ হিসেবে রাখে এবং সেই অংশ ছাড়া তার অস্তিত্ব নেই)

Example: A Human has a Heart, and the heart can not function independently.

```
class Heart {
    void beat() {
        System.out.println("Heart is beating");
    }
}

class Human {
    private Heart heart = new Heart(); // Composition

    void live() {
        heart.beat();
    }
}
```

1.b)

```
import java.util.ArrayList;
import java.util.Collections;

public class DanceCompetition {
    public static void main(String[] args) {

(1)     ArrayList<String> cityA = new ArrayList<>();
        ArrayList<String> cityB = new ArrayList<>();

(2)     cityA.add("Sophia");
        cityA.add("Emma");
        cityA.add("Olivia");

        cityB.add("Liam");
        cityB.add("Noah");
        cityB.add("Ava");
```

// DanceCompetition নামে একটি ক্লাস যার মধ্যে main method যেখান থেকে program শুরু হবে

// cityA এবং cityB -এর জন্য দুটি আলাদা ArrayList তৈরি করা হয়েছে

// নাম Add করা হয়েছে।

③ ArrayList <String> finalList = new ArrayList <>();
finalList.addAll(cityA);
finalList.addAll(cityB);
// cityA এবং cityB কে finalList-এ রাখতে হবে।

④ finalList.remove(finalList.size()-1);
// FinalList এর শেষ নামটি (Ava) বাদ দিয়ে হলো।

⑤ // তিনটি print করা হলো।
System.out.println("CityA :" + cityA);
System.out.println("CityB :" + cityB);
System.out.println("FinalList :" + finalList);

⑥ if(finalList.contains("Sophia")){
int index = finalList.indexOf("Sophia");
System.out.println("Sophia is ready to dance!
Index:" + index);
} else {
System.out.println("Sophia is not in the competition
}

// finalList - কে A-Z ক্রম সাজানো হয়েছে।

⑦ Collections.sort(finalList);
System.out.println("Sorted FinalList: "+finalList);

⑧ System.out.println("Total participants: "+
finalList.size());

// finalList - এ কতজন আছে.
সেটি প্রিন্ট করবে।

}
}

1.c)

```
public class Movie {          // Movie নামে একটি ক্লাস আছে

    private String title;
    private String director;   // দুটি variable declare করা হয়েছে, যে variable গুলো শুধুমাত্র এই class এর ভিতরে Access করা যাবে.

    public Movie () {          // default constructor, যা কোনো মান না দিলে বসত করে

    }

    public Movie (String title, String director) {
        this.title = title;
        this.director = director;
    }                          // Parameterized constructor. যখন new Movie("Inception", "Christopher Nolan") এর মতো কল করবো তখন title ও director সেট হয়ে যাবে.

    public void setTitle(String title) {
        this.title = title;    // Setter Method বাইরে থেকে title ও director এর মান সেট করতে পারি.
    }
```

```
public void setDirector(String director){
    this.director = director;
}

public String getTitle(){
    return title;
}

public String getDirector(){
    return director;
}
```

// getter Method বাইরে থেকে title ও director এর মান রিড করতে পারি।

Output:

Interstellar

James Cameron.

Inception.

Christopher Nolan.

1.d) Difference between import java.util.*; and import java.util.Scanner.

java.uitil.*;	java.util.Scanner.
Imports all utility classes from java.util package.	Imports only Scanner class.
May consume more memory or space.	Loads only the required class, saving memory.
Example: ArrayList, Scanner etc can be used.	Example: Only Scanner can be used.

2.a)

① interface WBehavior { // WBehavior একটি interface যা attack() নামের একটি Method declare করে, যা অস্ত্রের আক্রমণের ভিন্ন ভিন্ন আচরণ বোঝাতে কাজ করবে।
    void attack();
}

② interface WType {
    void WeaponType(); // অস্ত্রটি Melee নাকি Ranged তা বোঝাবে।
}

③ abstract class Weapon implements WBehavior, WType {
    abstract void hasWeapon(); → নির্দিষ্ট অস্ত্রটি আছে কিনা তা দেখাবে।
}

④ class Sword extends Weapon {
    public void attack() {
        System.out.println("Swing the Sword!");
    }
}

1. attack() - তীর ছোঁড়া বোঝায়
2. WeaponType() - Ranged type
3. hasWeapon() - জানায় যে বন্দুক আছে।

⑤ interface
abstract class MeleeFactory {
    abstract void createMeleeWeapon();
}
→ কাছাকাছি যুদ্ধের অস্ত্র তৈরির জন্য ব্যবহৃত হয়।

abstract class RangeFactory {
    abstract void createRangedWeapon();
}
→ দূর থেকে আক্রমণের অস্ত্র তৈরির factory

⑥ class WeaponFactory implements MeleeFactory, RangedFactory {
// java তে multiple inheritance support করে না, তাই interface ব্যবহার করা ভালো হবে।

```
public void WeaponType(){
System.out.println("Weapon Type: Melee");
}
public void hasWeapon(){
System.out.println("You have a sword!");
}
}
```

// Method implementation কেমন-
1. attack() - তলোয়ার চালানোর action
2. WeaponType() - Melee type দেখায়
3. hasWeapon() - জানায় যে তলোয়ার আছে

```
class Bow extends Weapon{
public void attack(){
System.out.println("Shoot an arrow!");
}
public void WeaponType(){
System.out.println("Weapon Type: Ranged");
}
public void hasWeapon(){
System.out.println("You have a bow!");
```

```
public void createMeleeWeapon () {
    Weapon sword = new Soword();
    sword.attack();
    sword.WeaponType();
    sword.hasWeapon();
System.out.println("Malee weapon is created");
}
```

// Sword ক্লাসের একটি object, যা. Weapon টাইপের রেফারেন্স ধরে রাখে।

```
public void createRangedWeapon (){
    Weapon bow = new Bow();
    bow.attack();
    bow.weaponType();
    bow.hasWeapon();
System.out.println("Ranged weapon is created");
}
}
```

⑦

```
public class Main {
  public static void main(String [] args){
    WeaponFactory factory= new WeaponFactory();
    factory.createMeleeWeapon();
    factory.createRangedWeapon();
```

2. খ)

```
import java.io.*;
```

(java তে IOException একটা checked exception, তাই এটা use করার আগে import করতে হয়)

(নিজের custom exception তৈরি করে, যা exception ক্লাসকে extend করে)

```
class UnderAgeException extends Exception {
    public UnderAgeException(String message) {
        super(message);
    }
```

(message accept করে Exception ক্লাসে পাঠায়)

```
class LibraryySystem {
    public void checkAge(int age) throws
            UnderAgeException, IOException {
```

(যদি ছোট হয়) (যদি বড় হয়) দুটি Exception throw করতে পারে

```
    if (age < 12) {
    throw new UnderAgeException("UnderAgeException");
    }
    else if (age > 30) {
    throw new IOException("User is old enough");
    }
```

```
        else {
            System.out.println("Congratulations! Your age
                                is perfect.");
        }
    }

    public static void main(String[] args){
        LibraryManagementSystem l = new LibraryManagementSystem();

        int[] testAges = {10, 25, 35};
                          ↓   ↓   ↘
                        under perfect over

        for(int age: testAges){
            try {
                l.checkAge(age);   (checkAge কল করা হচ্ছে)
            }
            catch (UnderAgeException e) {
                System.out.println(e.getMessage());
            }
            catch (IOException e) {
                System.out.println(e.getmessage());
            }
        }
    }
```