Lab 8 - Object-Oriented Programming (CS F213)

Interfaces in Java

An interface in Java is a blueprint of a class, defining a set of abstract methods that a class implementing the interface must provide. In essence, it specifies a contract that the implementing classes must adhere to. Interfaces are declared using the interface keyword and can be thought of as 100% abstract classes because they only contain method signatures and constants (public static final fields) without any implementation.

```
interface MyInterface {
    void method1();
    void method2();
}
```

In this example, `MyInterface` defines two abstract methods `method1` and `method2`. Any class that implements `MyInterface` is required to provide concrete implementations for these methods.

To create a class that adheres to an interface, you use the `implements` keyword. A class can implement multiple interfaces, enabling it to inherit and define behavior from multiple sources.

Here, `MyClass` implements `MyInterface` and provides concrete implementations for `method1` and `method2`.

```
class MyClass implements MyInterface {
    public void method1() {
        // Implementation for method1
    }

    public void method2() {
        // Implementation for method2
    }
}
```

Why Use Interfaces

1. **Multiple inheritance** - Java supports single inheritance for classes, meaning a class can only extend one superclass. However, by implementing multiple interfaces, a class can inherit behavior from multiple sources, effectively achieving a form of multiple inheritance. This allows for greater flexibility and code reuse in your software design.
2. **Polymorphism** - Interfaces enable polymorphism, a fundamental concept in OOP. Polymorphism allows different classes to be treated as instances of a common interface, simplifying code and making it more extensible. For example, you can create a collection of objects implementing an interface and interact with them uniformly.

Example

```java
interface Printable {
    void print();
}

interface Showable{
    void show();
}

class GameObject implements Printable, Showable {
    public void print() {
        System.out.println("Hello");
    }
    public void show(){
        System.out.println("Welcome");
    }
}

public class Main {
    public static void main(String args[]) {
        GameObject obj = new GameObject();
        obj.print();
        obj.show();
    }
}
```

Arrays & Comparisions in Java

`sort()`

In Java, the `sort` method is used to sort elements in an array in ascending order. It is part of the `Arrays` class in the `java.util` package and is available for arrays of primitive data types and objects. This method uses `compareTo()` to determine the natural order of elements in an array of objects. The `compareTo` method is part of the `Comparable` interface, which allows objects to define their natural ordering.

```
int[] intArray = {5, 3, 1, 4, 2};
Arrays.sort(intArray); // The 'intArray' is now sorted in ascending order:
{1, 2, 3, 4, 5}
```

> ⚠ For this lab, it is safe to ignore the warning: "`Comparable` is a raw type. References to generic type `Comparable<T>` should be parameterized"

`copyOf()`

In Java, the `copyOf` method is typically used to create a copy of an array with a specified length. This method is available in the `java.util.Arrays` class and is overloaded for different types of arrays (e.g., int[], long[], etc.).

```
int[] originalArray = {1, 2, 3, 4, 5};

// Create a copy of the original array with a specified length
int[] copyArray = Arrays.copyOf(originalArray, 3); // [1, 2, 3]
```

Additional Reading - Some Caveats regarding Java Interfaces

Default Methods

Starting from Java 8, interfaces can have default methods. These methods have an implementation in the interface itself and can be overridden by implementing classes. This feature allows you to add methods to interfaces without breaking compatibility with existing implementing classes.

```
interface MyInterface {
    void method1();

    default void method3() {
        // Default implementation for method3
    }
}
```

Static Methods

Java 8 also introduced static methods in interfaces. These methods are defined at the interface level and are accessible through the interface itself, not its implementing classes. They provide utility methods related to the interface.

```
interface MyInterface {
    static void staticMethod() {
        // Static method implementation
    }
}
```

`extend`-ing Interfaces

Interfaces can be extended in Java, similar to how inheritance occurs in classes. However, interfaces support multiple inheritance. A child interface can extend two different parent interfaces.

```
interface A {
    // Parent Interface 1
}

interface B {
    // Parent Interface 2
}

interface C extends A, B {
    // Child interface
}
```