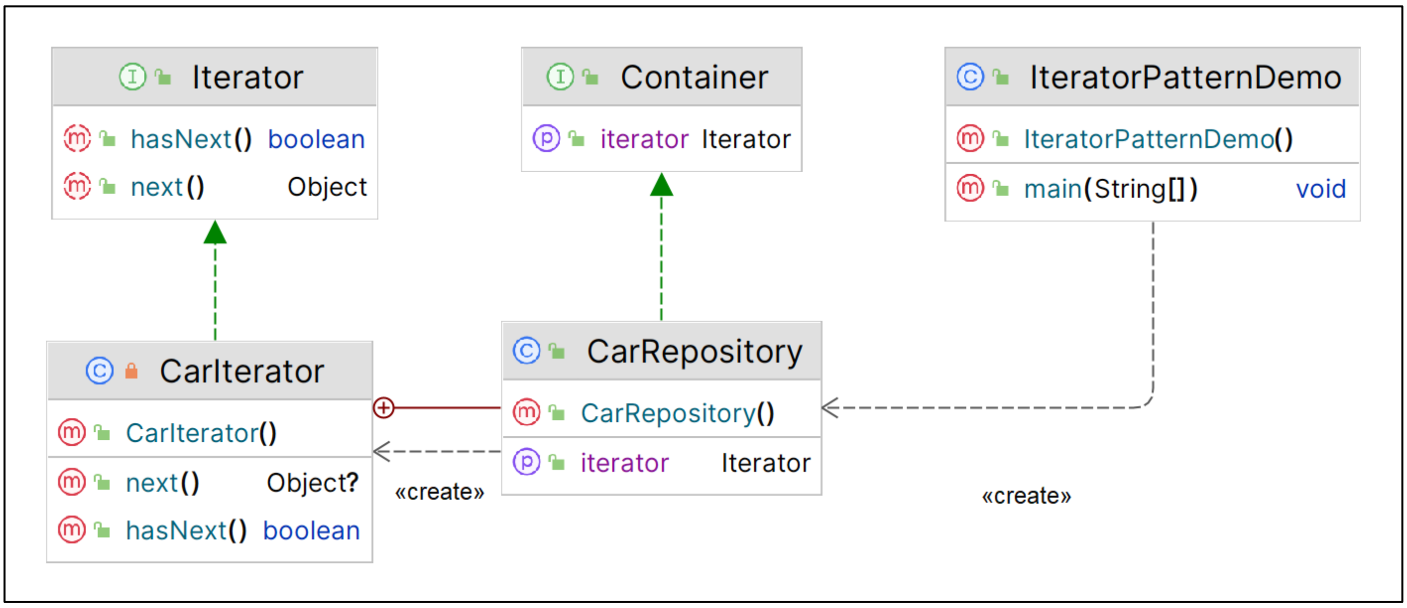
Assignment 13: Iterator Design Pattern

What is Iterator Design Pattern?

The **Iterator** design pattern allows us to **traverse** a collection of objects **without exposing** the underlying **implementation** of the collection.

Structure (Class Diagram)



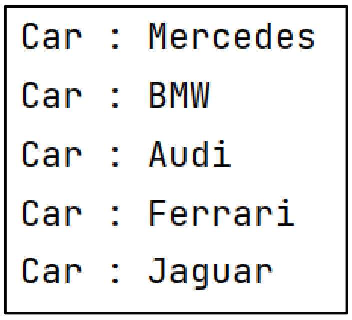
Implementation (Code)

```
// Interface for Iterator
public interface Iterator {
    public boolean hasNext();
    public Object next();
}

// Returns new instances of Iterator
public interface Container {
    public Iterator getIterator();
}

// Concrete Iterator to implement traversal of repository
public class CarRepository implements Container {
    public String cars[] = {"Mercedes", "BMW", "Audi", "Ferrari", "Jaguar"};

    public Iterator getIterator() {
        return new CarIterator();
    }

    private class CarIterator implements Iterator {
        int index;
        public boolean hasNext() {
            if (index < cars.length) {
                return true;
            }
            return false;
        }

        public Object next() {
            if (this.hasNext()) {
                return cars[index++];
            }
            return null;
        }
    }
}
```

// Demo - Main

```
public class IteratorPatternDemo {
    public static void main(String[] args) {
        CarRepository carRepo = new CarRepository();

        for (Iterator iter = carRepo.getIterator(); iter.hasNext();) {
            String name = (String)iter.next();
            System.out.println("Car : " + name);
        }
    }
}
```

Output



```
Car : Mercedes
Car : BMW
Car : Audi
Car : Ferrari
Car : Jaguar
```

Applicability

1. Use the **Iterator** pattern when your collection has a **complex data structure** under the hood, but you want to **hide its complexity from clients** (either for convenience or security reasons).
2. Use the pattern to **reduce duplication** of the traversal code across your app.
3. Use the Iterator when you want your code to be able to traverse different data structures or **when types of these structures are unknown** beforehand.