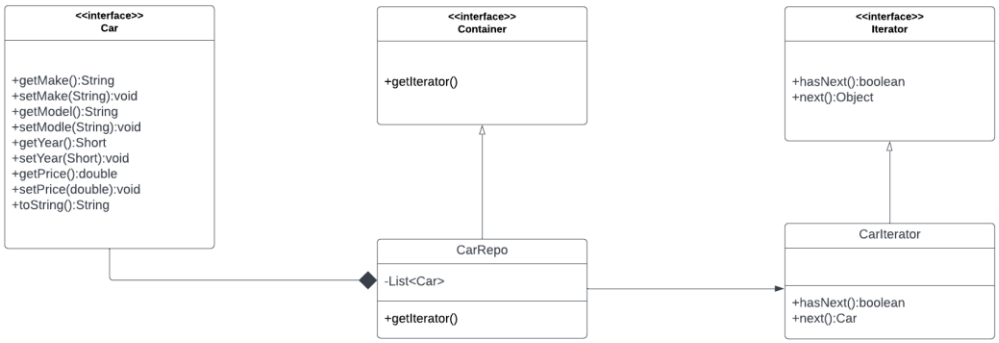
Jarosław Rybak

Iterator Pattern

Team 4: Used Car Dealership

UML Diagram



Code

Iterato.java

```
package IteratorPattern;

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

Container.java

```
package IteratorPattern;

public interface Container
{
	public Iterator getIerator();
}
```

Car.java

```
package IteratorPattern;

public interface Car
{
	public String getMake();
	public void setMake(String make);
```

```
        public String getModel();
        public void setModel(String model);
        public short getYear();
        public void setYear(short year);
        public double getPrice();
        public void setPrice(double price);
        public String toString();
}
```

CarRepo.java

```
package IteratorPattern;

import java.util.List;

public class CarRepo implements Container
{
        private  List<Car> Cars;

        public CarRepo()
        {
                DemoList DL =  new DemoList();
                Cars = DL.getCars(); //This would be replace by the real list of cars
        }

        //For unit testing
        public CarRepo(List<Car> cars)
        {
                Cars = cars;
        }

        @Override
        public Iterator getIerator()
        {
                return new CarIterator();
        }

        private class CarIterator implements Iterator
        {
                int index;

                @Override
                public boolean hasNext()
                {
                        if(index < Cars.size())
                        {
                       return true;
                  }
                        return false;
                }

                @Override
                public Car next()
                {
                        if(this.hasNext()){
```

```
                return Cars.get(index++);
            }
                return null;
        }
    }

    //For unit testing
    public List<Car> getCars()
    {
        return Cars;
    }

    //for unit testing
    public void setCars(List<Car> cars)
    {
        Cars = cars;
    }
}
```

Unit Test

CarRepoTest.java

```
package IteratorPattern;


import static org.junit.Assert.*;


import java.util.ArrayList;

import java.util.List;


import org.junit.After;

import org.junit.AfterClass;

import org.junit.Before;

import org.junit.BeforeClass;

import org.junit.Test;


public class CarRepoTest

{
```

```
private static DemoList DL;

private CarRepo CR;

private Iterator ier;


@BeforeClass

public static void setUpBeforeClass() throws Exception

{

	//Generating RNG list for testing.

	DL = new DemoList();

}


@AfterClass

public static void tearDownAfterClass() throws Exception

{


}


@Before

public void setUp() throws Exception

{

	//Making a clone for each test to not edit original list

	List<Car> ListClone =  new ArrayList<Car>();

	for(int i = 0;i<DL.getCars().size();i++)

	{

		ListClone.add(DL.getCars().get(i));
```

```
            }

            CR = new CarRepo(ListClone);


            ier = CR.getIerator();
        }


        @After
        public void tearDown() throws Exception
        {


        }


        //Testing Clone: Testing if editing List generator edits usable list
        @Test(expected=IndexOutOfBoundsException.class)
        public void testSetCars()
        {
            DL.getCars().add(DL.RNGCar());


            assertNull("Testing if editing original list will edit CarRepo list.",
CR.getCars().get(DL.getCars().size()));
        }


        //Testing Clone: testing if both lists are the same
        @Test
        public void test_Lists()
        {
```

```
		for(int i =0; i<DL.getCars().size()||i<CR.getCars().size();i++)
		{
			assertEquals("Testing Car: "+i+" If failed scrap all tests",DL.getCars().get(i),CR.getCars().get(i));
		}
	}

	//Testing Iterator: Testing if 2 iterators are the same
	@Test
	public void testing_Diff_Iterators()
	{
		assertFalse("Testing if 2 Iterators are the same.",ier.equals(CR.getIerator()));
	}

	//Testing Iterator: Testing if 2 iterators give the same output
	@Test
	public void testing_Diff_output()
	{
		Iterator newIer = CR.getIerator();
		while(ier.hasNext()||newIer.hasNext())
		{
			assertEquals("Testing same list but diff iterator output.",ier.next(),newIer.next());
		}
	}
```

```
//Testing hasNext: Testing if the first hasNext returns true
@Test
public void hasNext_BaseCase()
{
	assertTrue("Teting hasNext base case",ier.hasNext());
}

//Testing hasNext: Testing if ending hasNext returns false
@Test
public void hasNext_OverFlow()
{
	while(ier.hasNext())
	{
		ier.next();
	}

	assertFalse("Testing hasNext overflow",ier.hasNext());
}

//Testing Next: Testing all Cars at once
@Test
public void next_compared_orginal_List()
{
	for(int i =0; i<DL.getCars().size()||i<CR.getCars().size();i++)
	{
```

```
                    assertEquals("Testing Car: "+i,DL.getCars().get(i),ier.next());
                }
            }
}
```

Component Test

Make 2 different instances of the Iterator. Use them separately for different purposes without one interfering with the other. Iterator 1 will filter Cars from “Make”: “Beta” and return them. Iterator 2 will return Cars in ascending order of year. Neither will know of the other nor have an effect on the others output.