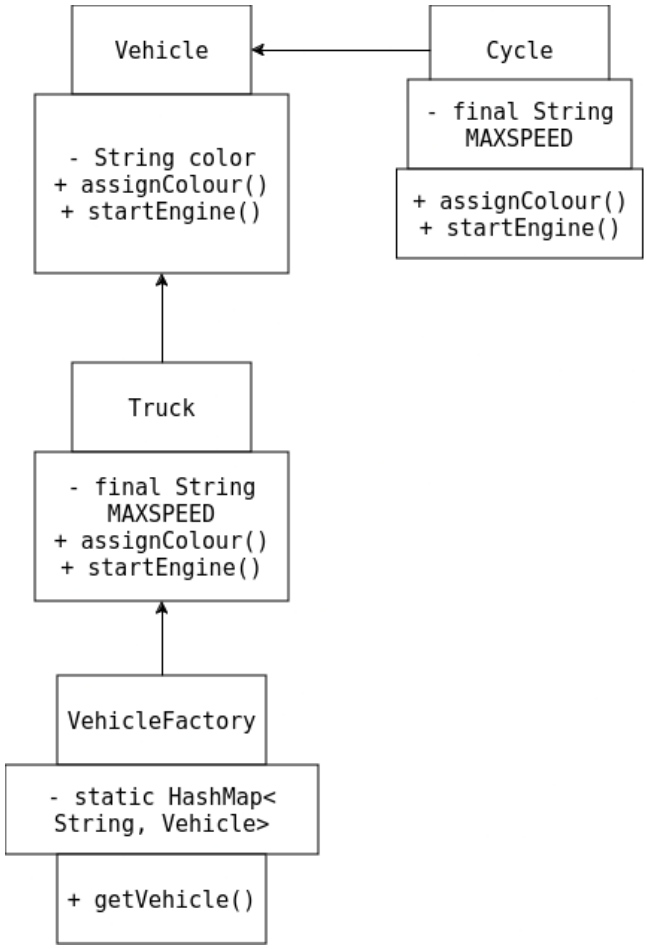
Flyweight Design Pattern

Description:

The flyweight design pattern is a structural pattern that aims to minimize memory usage and improve performance by sharing common parts of objects among multiple similar objects. It achieves this by separating intrinsic (shared) and extrinsic (unique) state within objects. Common elements are stored externally and shared among objects, reducing memory overhead. By isolating intrinsic state, the flyweight pattern enables the same object to be reused across different contexts, reducing the overall number of objects instantiated. This pattern is particularly useful in scenarios where a large number of similar objects need to be managed efficiently, such as in graphical applications for managing graphical elements like characters in a text editor or tiles in a game environment.

Class diagram:



Implementation:

1. VehicleFactory.java

```
package FW;
import java.util.HashMap;
public class VehicleFactory {

    private static HashMap<String,Vehicle> hashMap=new HashMap<String,Vehicle>();
```

```
	public static Vehicle getVehicle(String type)
	{
		Vehicle v=null;
		if(hashMap.containsKey(type))
		{
			v=hashMap.get(type);
		}
		else
		{
		switch(type)
		{
		case "Cycle":
			System.out.println("Cycle is created");
			v=new Cycle();
			break;
		case "Truck":
			System.out.println("Truck is created");
			v=new Truck();
			break;
		default:
			throw new IllegalArgumentException("Vehicle type "+type+" does not exist");
		}
		hashMap.put(type,v);
		}
		return v;
	}
}
```

2. Vehicle.java

```
package FW;

public interface Vehicle {

	public void assignColour(String color);
	public void startEngine();
}
```

3. Cycle.java

```
package FW;

public class Cycle implements Vehicle{

	private final String MAXSPEED;  //Intrensic property
	private String color;		// Extrinsic property

	Cycle()
	{
		MAXSPEED="15 km/hr";
```

```
	}

	@Override
	public void assignColour(String color) {
	this.color=color;
	}

	@Override
	public void startEngine() {
		System.out.println(color+" colored Cycle with Max speed is:"+MAXSPEED);
	}
}
```

4. Truck.java

```
package FW;

public class Truck implements Vehicle{

	private final String MAXSPEED;  //Intrensic property
	private String color;

	Truck()
	{
		MAXSPEED="120 km/hr";
	}

	@Override
	public void assignColour(String color) {
	this.color=color;
	}

	@Override
	public void startEngine() {
		System.out.println(color+" colored Truck with Max speed is:"+MAXSPEED);
	}
}
```

5. Buyer.java

```
package FW;

public class Buyer {
	public static void main(String args[])
	{
		Vehicle cycle=VehicleFactory.getVehicle("Cycle");
		cycle.assignColour("Blue");
		cycle.startEngine();
		cycle.assignColour("Black");
		cycle.startEngine();

		//Vehicle truck1=VehicleFactory.getVehicle("Car");
```

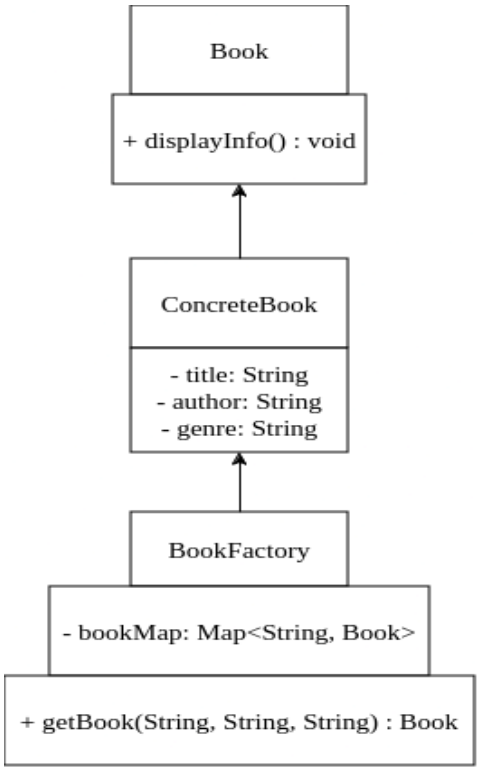
```
                Vehicle truck=VehicleFactory.getVehicle("Truck");
                truck.assignColour("Brown");
                truck.startEngine();

                Vehicle spaceGrayTruck=VehicleFactory.getVehicle("Truck");
                spaceGrayTruck.assignColour("SpaceGrayTruck");
                spaceGrayTruck.startEngine();
        }
}
```

Output:

```
Cycle is created
Blue colored Cycle with Max speed is:15 km/hr
Black colored Cycle with Max speed is:15 km/hr
Truck is created
Brown colored Truck with Max speed is:120 km/hr
SpaceGrayTruck colored Truck with Max speed is:120 km/hr
```

Flyweight Design Pattern for Library Management System

Class diagram:



Implementation:

1. LibraryManagement.java

```
import java.util.HashMap;
import java.util.Map;

// Flyweight interface
interface Book {
    void displayInfo();
}

// Concrete Flyweight
class ConcreteBook implements Book {
    private final String title;
    private final String author;
    private final String genre;

    public ConcreteBook(String title, String author, String genre) {
        this.title = title;
        this.author = author;
        this.genre = genre;
    }
```

```
    @Override
    public void displayInfo() {
        System.out.println("Title: " + title);
        System.out.println("Author: " + author);
        System.out.println("Genre: " + genre);
        System.out.println("--------------------------");
    }
}

// Flyweight Factory
class BookFactory {
    private static final Map<String, Book> bookMap = new HashMap<>();

    public static Book getBook(String title, String author, String genre) {
        String key = title + "_" + author + "_" + genre;
        if (!bookMap.containsKey(key)) {
            bookMap.put(key, new ConcreteBook(title, author, genre));
        }
        return bookMap.get(key);
    }
}

// Client code
public class LibraryManagement {
    public static void main(String[] args) {
        // Create and display books
        Book book1 = BookFactory.getBook("Java Programming", "John Doe", "Programming");
        book1.displayInfo();

        Book book2 = BookFactory.getBook("Python Basics", "Jane Smith", "Programming");
        book2.displayInfo();

        Book book3 = BookFactory.getBook("Harry Potter", "J.K. Rowling", "Fantasy");
        book3.displayInfo();

        // Demonstrate that the same book instance is reused
        Book book4 = BookFactory.getBook("Java Programming", "John Doe", "Programming");
        book4.displayInfo();
    }
}
```

Output:

```
zoro-d-code@wado-ichimonji:~/Roger/College$ cd "/home/zoro-d-code/Roger/College/Design_lab_Codes/Flyweight_design/2/" && javac L
ibraryManagement.java && java LibraryManagement
Title: Java Programming
Author: John Doe
Genre: Programming
--------------------------
Title: Python Basics
Author: Jane Smith
Genre: Programming
--------------------------
Title: Harry Potter
Author: J.K. Rowling
Genre: Fantasy
--------------------------
Title: Java Programming
Author: John Doe
Genre: Programming
--------------------------
```