プログラミングⅡ　2022　第９回 演習課題　「継承」

1. 下記のプログラムにおいて，(a)〜(e)の指示に従って main メソッドに処理を追加し，プログラムを完成させよ。さらに， Driver クラスと Car クラスの関係を UML のクラス図で表せ。

```
// ファイル Driver.java
public class Driver {  // ドライバー

    private String name;  // 名前
    private int license;  // 免許証番号

    public Driver(String name, int license) {
        this.name = name;
        this.license = license;
    }

    // println などでの文字列への変換方法を指定
    public String toString() {
        return this.name + "（第" + this.license + "号）";
    }

}


// ファイル Car.java
public class Car {  // 自動車

    private String number;    // ナンバー
    protected double gas;     // ガソリン量 [L]
    protected Driver driver;  // ドライバー

    public Car(String number) {
        this.number = number;
        this.gas = 0.0;
    }

    public String getNumber() {
        return this.number;
    }

    public void setDriver(Driver d) {
        this.driver = d;
    }

    public void fillGas(double g) {
        this.gas += g;
        System.out.printf("燃料残り：%4.1fL%n", this.gas);
    }

    public void run(double dist) {
        // 燃費 10km/L（ガソリン 1L で 10km 走行）でガソリンの消費量を計算
        this.gas -= dist / 10.0;
        System.out.printf("自 動 車：%s%n", this.number);
        System.out.printf("運 転 者：%s%n", this.driver.toString()); //「.toString()」は省略可
        System.out.printf("走行距離：%4.1fkm%n", dist);
        System.out.printf("燃料残り：%4.1fL%n", this.gas);
    }
}
```

```
// ファイル Main.java
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("ナンバー？ ");
        String number = sc.next();

        // (a) ナンバーを使って car1 を生成

        Car car1 =

        System.out.print("給油量？ ");
        double gas = sc.nextDouble();

        // (b) 指定量のガソリンを car1 に給油（注ぎ足す）

        car1.

        System.out.print("ドライバー？ ");
        String dname = sc.next();
        System.out.print("免許証番号？ ");
        int lisence = sc.nextInt();

        // (c)ドライバーdriver1 を生成

        Driver driver1 =

        // (d) driver1 を car1 に設定（driver1 が car1 に乗る）

        car1.

        System.out.print("走行距離？ ");
        double dist = sc.nextDouble();

        // (e) 指定距離を走行（run を実行） もし走行距離が 10km ならガソリンが 1L 減ることを確認せよ

        car1.
    }

}
```

2. 1.のプログラムに Car クラスを継承した KeiCar クラスと Truck クラスを追加し，main メソッドも適当に修正して動作を確認せよ。その際，KeiCar と Truck の run メソッドでは，燃費をそれぞれ 15 km/L と 5 km/L で計算するようにせよ。さらに，Car と Truck と Driver の関係を UML のクラス図で表せ。

```
public class KeiCar             {

    public KeiCar(String number) {

        super

    }

    public void run(double dist) {


        System.out.printf("軽自動車：%s%n",              );
```

```
    }
}


public class Truck              {

    private double loadage;          // 積載量 [kg]
    public final double maxLoadage;  // 最大積載量 [kg]

    public Truck(String number, double maxLoadage) {

        super

        this.maxLoadage = maxLoadage;  // final フィールドへの代入はコンストラクタでのみ可能

    }

    public void run(double dist) {


        System.out.printf("トラック：%s%n",              );







    }

    // x kg の荷物を積む（最大積載量までしか積めない）
    public void load(double x) {







    }

    // x kg の荷物を下ろす（積載量がマイナスにはならない）
    public void unload(double x) {







    }

    // 現在の積載量を返す
    public double getLoadage() {


    }

}
```

【補足説明】

- toString メソッド（教科書 14 章 14.2.4）

  どんなクラスにでも public String toString()というメソッドを作ると，文字列への自動変換方法を定義することができる。これによって，クラス型の変数を println で表示したときや，「+」演算子で文字列と結合したときに，toString メソッドが自動的に実行されて文字列に変換される。

  実は，すべての Java のクラスは暗黙のうちに Object という名前のクラスを継承している。つまり，Object クラスはすべてのクラスの親（祖先）であり，クラスに toString メソッドを作ることは Object クラスの toString をオーバーライド（上書き定義）していることになる。

- System.out.printf メソッド（教科書 15 章 15.5）

  printf の f はフォーマット（書式）の略で，データの書式を整えて表示するという意味を表す。このメソッドの基本的な文法は System.out.printf("書式文字列", 変数や式)であり，書式文字列の中に「%」で始まる書式指定があると，そこに変数や式の値が指定の書式で埋め込まれて表示される。書式文字列の中には，書式指定を複数書くことができ，その場合は引数である変数や式もその個数だけ並べて書く（可変長引数）。

  代表的な書式指定には次のようなものがある。

`%d`	この部分に引数の整数（10 進数）の値を埋め込む。
`%f`	この部分に引数の実数（float または double）の値を埋め込む。
`%s`	この部分に引数の文字列の値を埋め込む。
`%n`	この位置で改行する。環境（Windows，Mac 等）に応じて適切な改行処理を行う。

  次のような複雑な書式を指定することもできる。

`%10d`	10 文字分の表示欄を確保し，右寄せで整数を表示する。
`%-10s`	10 文字分の表示欄に確保し，左寄せで文字列を表示する（「-」を付けると左寄せ）。
`%10.3f`	10 文字分の表示欄を確保し，右寄せで小数第 3 位まで実数を表示する。 例：123.45 を表示すると「`___123.450`」となる（空白 3 個＋「123」＋小数点＋「450」）。

  さらに，\（または¥）で始まる記法はエスケープシーケンスと呼ばれ，下記にいくつか例を挙げたような特殊な文字や制御コードを表す。なお，Windows の日本語環境では通常「\」と「¥」は同じ文字コードであり，環境やフォントによってどちらが表示されるかが決まる。

`\t`	水平タブ	`\"`	「"」文字	`\\`	「\」文字
`\n`	改行	`\r`	復帰	←	「\n」と「\r」は Java では使わないのが安全

- 継承における private と protected（教科書 13 章 p.519 コラム）

  クラスの private メンバ（フィールドとメソッド）は，他のクラスからアクセスすることができない。つまり，そのメンバの存在自体が見えなくなる。このルールは厳格であり，継承したサブクラス（子クラス，派生クラス）からでも，スーパークラス（親クラス，基本／基底クラス）の private メンバを参照することはできない。

  しかし，だからといってサブクラスでだけ使えればいいフィールドやメソッドを public にしてしまい，その他のクラスにも完全に公開してしまうのは，カプセル化として好ましくない。

  そこで，private と public の中間として，サブクラス（および，同一パッケージ内のクラス）にだけ公開することを意味する protected（限定公開／保護）というアクセス指定が用意されている。

  protected メンバはサブクラス（さらに“子孫”クラス）の中では，通常のフィールドやメソッドのようにアクセスすることができる。protected は，UML のクラス図では「#」記号で表す。

- 「オーバーライド」と「オーバーロード」

  言葉が似ているが，これらの違いをよく理解しておこう。次の例文の意味がわかりますか？「継承では，メソッドの“オーバーライド”のしかたを間違えると，意図しない“オーバーロード”になってしまい，バグの原因になります」