객체지향프로그래밍및실습 – 5주차 실습활동지(10월2일)

성명: _____ 학과: _____ 학번: _____ 실습반: _____

제출방식: pdf 형태로 작성된 보고서(파일명: report_5)와 HCTS.java 파일을 압축해서(.zip 등) **압축파일**의 형태로 제출한다.

제출기한: 10월 3일(금) 23시59분까지 (지연 제출 0점 처리함, 미완성이라도 반드시 제출하길 바람)

I. Objectives

1. 객체와 클래스, 객체의 배열, 객체의 매개변수 전달 등의 기본 개념을 종합적으로 활용하여 프로그램을 작성할 수 있다.
2. 구조적 프로그래밍 기법으로 작성된 코드를 객체지향 프로그램으로 변환할 수 있다.
3. Object composition 개념을 이해하고 여러 개의 클래스로 구성된 객체지향 프로그램의 구조를 이해할 수 있다.

II. Exercises (15점)

이번 실습부터 변수명과 메소드 명은 라이브러리 및 교재 코드와의 일관성을 위해 모두 CamelCase 형식으로 작성하도록 한다.
1주차 및 4주차 실습활동지를 참조하여 다음 물음에 답하시오.
Car 및 Time 클래스는 5주차 실습 자료의 첨부파일로 제공된 Car.java, Time.java를 사용하고, 클래스의 내용은 4주차 실습활동지를 참고한다.

1. 1주차 실습활동지 HCTS.java의 initialize() 메소드(아래 코드 참조)를 다음과 같이 수정하시오. [initialize() 메소드 코드 첨부, 2점]

- Car 및 Time 클래스의 constructor를 이용하여 Car 배열 car_list(새 이름: carList) 및 current_time (새 이름: currentTime)을 초기화 한다.

```
// 1주차 실습활동지 initialize 메소드
    public static void initialize()
    {
        car_list = new Car[3];

        for(int i=0; i<car_list.length; i++)
            car_list[i] = new Car();

        car_list[0].no = 1111;
        car_list[0].speed = 80;
        car_list[0].position = 0;

        car_list[1].no = 2222;
```

```
        car_list[1].speed = 100;
        car_list[1].position = 0;

        car_list[2].no = 3333;
        car_list[2].speed = 120;
        car_list[2].position = 0;

        currentTime = new Time();
        currentTime.hour = 0;
        currentTime.minute = 0;

        in = new Scanner(System.in);
    }

    public static void initialize()
    {
        // 해당 내용 작성
        in = new Scanner(System.in);

    }
```

2. 1주차 실습활동지의 HCTS.java의 moveAllCars() 메소드(아래 코드 참조)를 다음과 같이 수정하시오. [moveAllCars() 메소드 코드 첨부, 2점]

- 4주차 실습활동지의 move() 메소드를 사용한다.
- Car 클래스에 getXXX(), setXXX() 메소드를 추가하여 사용할 수 있음.

```
// 1주차 실습활동지 move_all_cars 메소드
    public static void move_all_cars(int min)
    {
        for(int i=0; i<carList.length; i++)
        {
            carList[i].position += (double) carList[i].speed/60*min;
            if(carList[i].position > 500.0)
                carList[i].position = 500.0;
        }
    }

    public static void moveAllCars(int min)
    {
        //해당 내용 작성
    }
```

3. Time 클래스에 다음 메소드를 추가하시오. [calculateDifference() 메소드 코드 첨부, 1점]

- this의 시간과 매개변수 t의 시간 차이(t-this, minute단위)를 계산하여 반환
- 1주차의 calculate_time_difference 메소드(아래 첨부코드) 참조

```
// 1주차 실습활동지 calculate_time_difference 메소드
public static int calculate_time_difference(Time t)
{
        return (t.hour-current_time.hour)*60 + (t.minute-current_time.minute);
}


public int calculateDifference(Time t)
{
      //해당 내용 작성
}
```

4. 1주차 실습활동지의 HCTS.java의 handle_setting_time() 메소드(아래 코드 참조)를 다음과 같이 수정하시오. [handleSettingTime() 메소드 코드 첨부, 2점]

- 4주차 실습활동지에서 정의한 Time 클래스의 constructor를 이용하여 Time 객체를 초기화한다.
- Time 클래스의 calculateDifference()를 이용하여 min을 계산한다.

```
// 1주차 실습활동지 handle_setting_time 메소드
public static void handle_setting_time()
{
      Time t = new Time();
      t.hour = in.nextInt();
      t.minute = in.nextInt();

      int min = calculate_time_difference(t);
      set_current_time(t);
      move_all_cars(min);
}

public static void handleSettingTime()
{
      //해당 내용 작성
      setCurrentTime(t);
      moveAllCars(min);
}
```

5. 1주차 실습활동지의 HCTS.java의 handle_locating_cars() 메소드(아래 코드 참조)를 4주차 실습활동지에서 정의한 toString()메소드를 사용하도록 수정하시오. [handleLocatingCars() 메소드 코드 첨부, 1점]

- 4주차 실습활동지에서 정의한 Car 클래스의 toString()를 이용하여 Car 객체를 출력한다.

```
// 1주차 실습활동지 handle_locating_cars 메소드
public static void handle_locating_cars()
```

```
{
        for(int i=0; i<car_list.length; i++)
            print_car_info(car_list[i]);
}


public static void handleLocatingCars()
{
    //해당 내용 작성
}
```

6. 위에서 구현한 모든 코드를 통합하여 1주차 실습활동지에서 구현한 HCTS와 동일하게 동작하는 객체지향적인 HCTS를 완성하시오. [HCTS.java 파일 첨부 및 내부 구현 코드 확인, 2점]

- 활동지 내부에 첨부하지 않음. HCTS.java를 다운로드하여 활동지 pdf 파일과 함께 압축하여 제출

7. Car 클래스를 이용하여 MovingCar 클래스를 다음과 같이 정의하고자 한다. calcTollFee() 메소드를 채우시오. [calcTollFee() 메소드 코드 첨부, 3점]

- MovingCar는 하나의 Car를 instance variable로 가진다.
- MovingCar는 departure와 destination을 가진다. departure는 출발지점이고 destination은 목적지점이다. 단, 두 변수 모두 int형으로 0보다 크거나 같고 500보다 작거나 같으며 departuare는 항상 destination 값보다 작다고 가정한다.
- Toll Fee를 계산하는 공식은 다음과 같다. 계산 후 소수점 이하는 절삭한다.

  Toll Fee = 기본요금(1,000원) + 거리 x 거리요율 x 속도요율

  거리요율 = 50

  속도요율 = 속도/100
- 필요하면 Car 클래스에 getXXX() 메소드를 추가할 수 있다.

```
// MovingCar.java
public class MovingCar
{
    Car car;
    int departure;
    int destination;
    public MovingCar(Car c, int dept, int dest)
    {
        car = c; departure = dept; destination = dest;
    }
   public int calcTollFee()
    {
        // 해당 부분 작성
```

```
    }

}
```

8. 7번의 MovingCar 클래스를 테스트하는 코드를 아래와 같이 작성하고자 한다. main 메소드를 적절하게 채운 후 실행하시오. [main 메소드 코드 & 실행결과 첨부, 2점]

1) 하나의 Car 객체를 생성한다. no="c1111", speed=120, position=100
2) 스텝 1에서 생성한 Car 객체를 이용하여 하나의 MovingCar 객체를 생성한다.
   departure = 0, destination = 400
3) 스텝 2에서 만든 MovingCar 객체의 Car 객체를 출력한 후 toll fee를 계산하여 출력한다. 단, Car객체를 출력하는 방식은 4주차실습활동지의 toString()메소드를 이용한다.

※ 필요하면 MovingCar에 getXXX() 메소드를 추가할 수 있다.

```
public class MovingCarTest
{
    public static void main(String[] args)
    {
        // 해당 부분 작성
    }

}
```