Aggregation

Dr. Jason Barron

South East Technological University

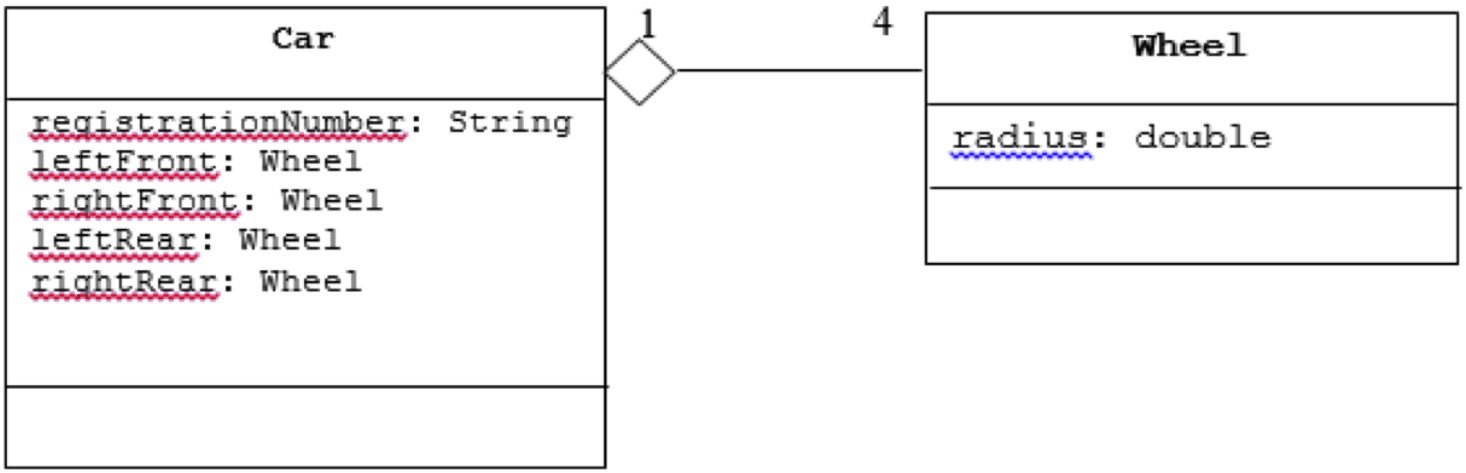
October 2, 2024

Aggregation

- Aggregation is whereby a class can have objects of other classes as members
- A car, for instance, is made up of its engine, its chassis, its wheels and several other parts
- Each of those parts could be considered separate objects

Aggregation

- Therefore, we can say that a car is an **aggregation** - it is composed, at least in part, of other objects
- Aggregation is sometimes described as a "has-a" relationship
- For instance, a (typical) car has four wheels
- In UML, we can model this relationship as follows:



Aggregation

```
public class Wheel
{
    private double radius;
    // other attributes here
    public Wheel(double r)
    {
        radius = r;
    }
    // other methods here...
}
```

Aggregation

```
public class Car
{
    private String registrationNumber;
    private Wheel leftFront;
    private Wheel rightFront;
    private Wheel leftRear;
    private Wheel rightRear;
    public Car(double r)
    {
        leftFront = new Wheel(r); // create each Wheel with radius r
        rightFront = new Wheel(r);
        leftRear = new Wheel(r);
        // initialise the reg. Number also...
    }
    // other methods here...
}
```

Aggregation

- In software, aggregate objects are those that contain references to other objects as instance data
- For example a Car object contains a String object that represents the registration number
- Strings are objects, technically, this feature alone makes the Car object an aggregate object

Aggregation - Example

- Class Exercise:
- Consider the following problem statement:
  - "An **AlarmClock** stores an alarm time and the current time. The clock will sound if the current time matches the alarm time AND the alarm is set (primed)".
  - Draw a UML class diagram to show the relationship between class **AlarmClock** and class **Time**

Aggregation - Example

- We will now consider the implementation details for class **AlarmClock**
- Clearly, each **AlarmClock** instance needs to store two instances of **Time**
  - One instance to store the current time
  - One instance to store the alarm time

Aggregation - Example

- Class **AlarmClock** will provide the following interface:
  - **setAlarmTime()** - to set the alarm time
  - **getAlarmTime()** - to return, as a String, the current **Alarm** time
  - **enableAlarm()** - to activate the alarm (to sound at the current alarm time)

Aggregation - Example

- **disableAlarm()** - to deactivate the alarm
- **alarmStatus()** - to return, as a **boolean** value, the current status of the alarm
- **checkAlarmTime()** - to compare the current time (hours and minutes) against the stored alarm time. If they are equal, ring the alarm!
- Method **checkAlarmTime** returns a boolean value of false if the current time does not match the alarm time

Aggregation

Class Exercise

- Write the definition for a class **Dice**
- A dice has a face value (between 1 and 6)
- The constructor method should call method **roll**
- Provide a method **roll** to simulate the roll of the dice
- Provide a **get** method to read the face value (i.e. **getFaceValue())**

Aggregation

- Write the definition for a class **PairOfDice**
- A **PairOfDice** is composed of two **Dice** objects
- Provide a method **roll** which should invoke (call) method roll on each **Dice**
- Provide another method **getFaceValues()** to return the face value of each **Dice**