Interaction Diagrams

COMMUNICATION DIAGRAM

Interaction Diagrams

- Interaction diagram model the interactions between objects in some behavior
- Help visualize the interaction behavior of the system in context of objects
- Also describe the message flow in the system
- Typically, used to look at behavior of the system within a single use case
- This interaction can be modelled with two diagrams
  - Collaboration/communication
  - Sequence

Communication Diagram

- Communication diagram shows how objects associate with each other
- The focus in on link, rather than the order
- Communication diagram gives a big picture view of the interactions between different objects of the system
- Also known as interaction/collaboration diagram
- Easy to draw during discussion

Basic Elements

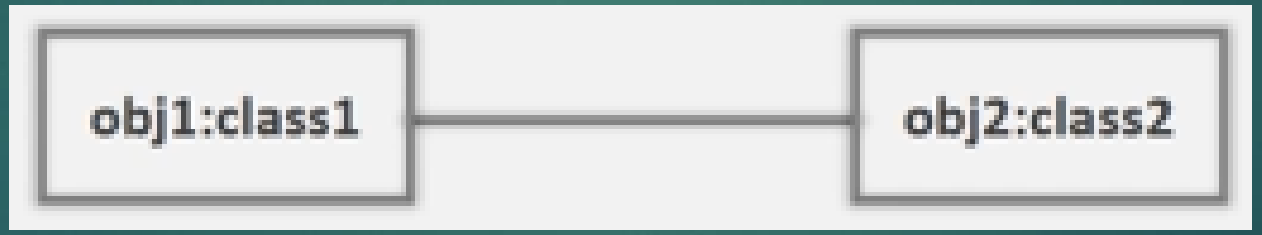
- Contains three elements
  - Participants
  - Communication link
  - Messages that can be passed using the link

Participants

- Represented by a rectangle
- Name of the participant and its class are placed in the rectangle as <name>:<class>
- Both name & class may be specified
- However, either of them can be left out if the participant is anonymous

Communication Link

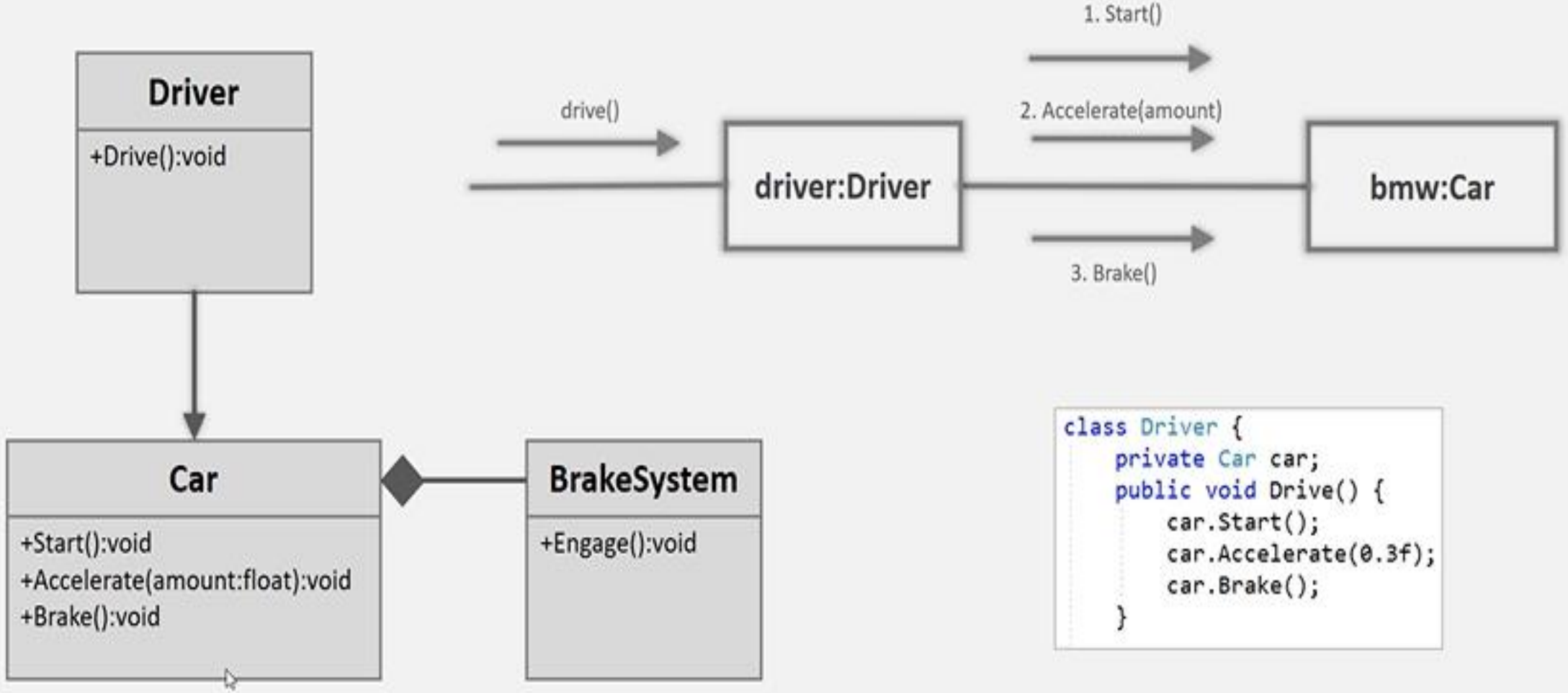
- Shown as a single line
- Connects two participants
- Allows messages to be passed between different participants
- Without the link, participants cannot interact with each other



Messages

- Depicted as a filled arrow from message sender to the message receiver
- The signature is made up of name & list of parameters
- We need to show the order in which the message are invoked
  - Shown as a number before each message

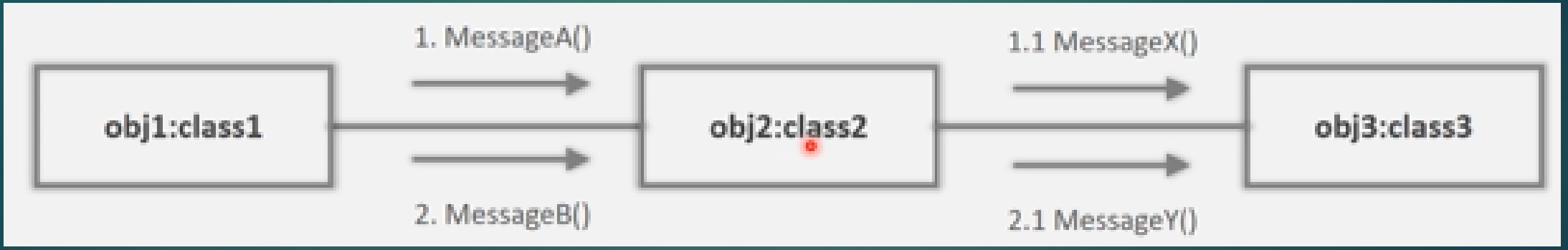
Example



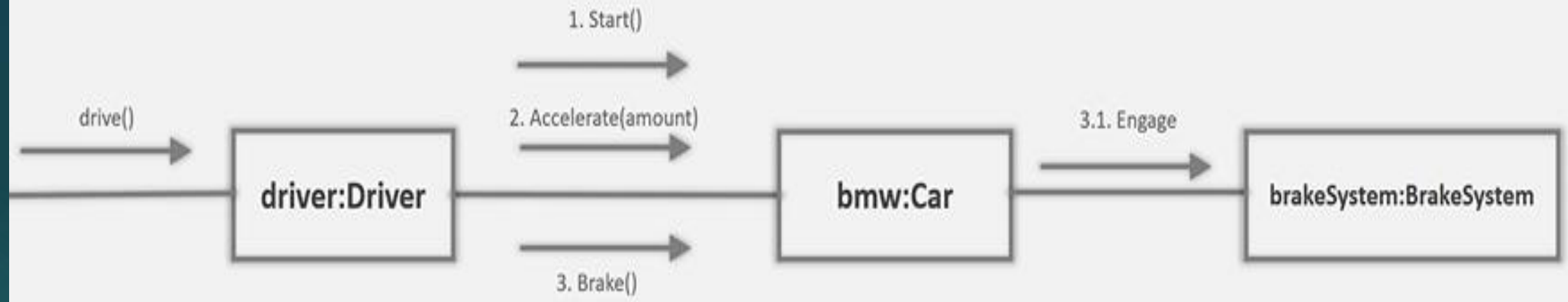

```
class Driver {
    private Car car;
    public void Drive() {
        car.Start();
        car.Accelerate(0.3f);
        car.Brake();
    }
```

Nested Messages

- A message that is invoked because of another message is called nested message
- The number scheme for such message uses the base message number followed by the new number e.g 1.1. MessageX()



Example

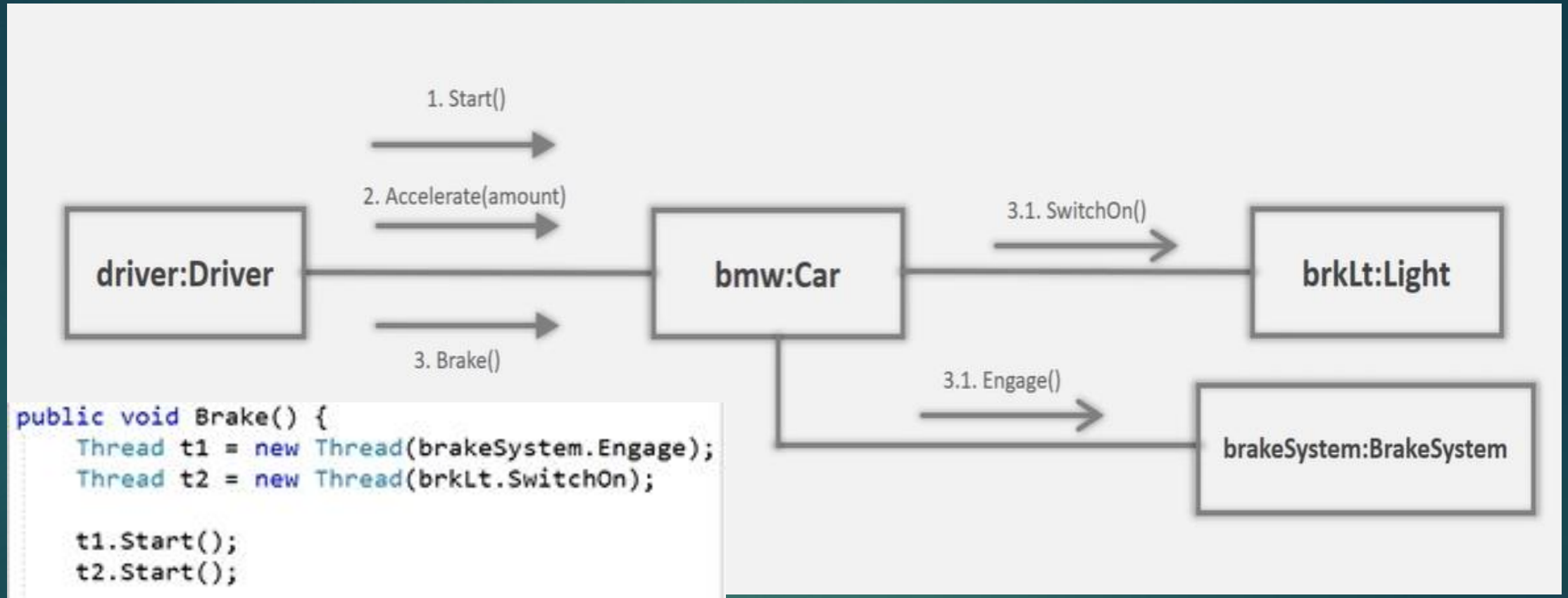



```
class Driver {
    private Car car;
    public void Drive() {
        car.Start();
        car.Accelerate(0.3f);
        car.Brake();
    }
```

```
class Car {
    BrakeSystem brakeSystem;
    public void Accelerate(float amount) ...
    public void Start() ...
    public void Brake() {
        brakeSystem.Engage();

    }
```

Concurrent Messages

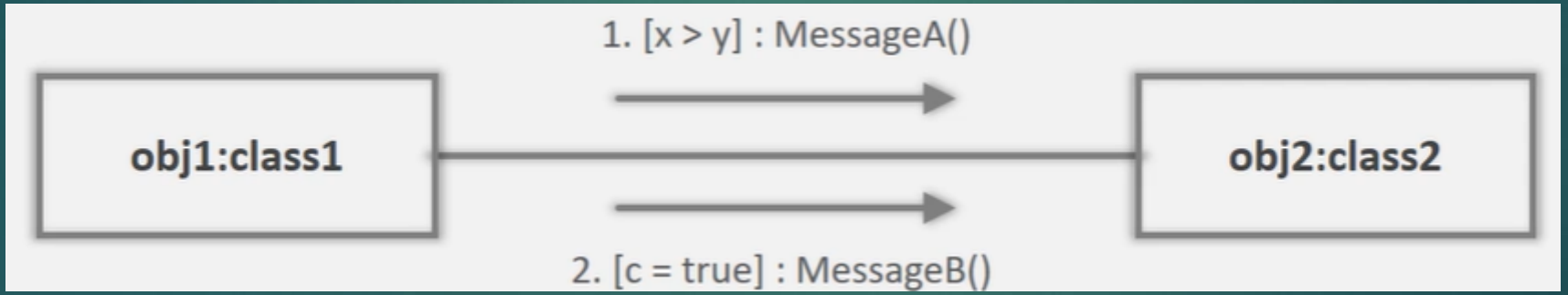
- These are represented using number and letter notation e.g 2.a. MessageX(), 2.b. MessageY(), etc
- The other notation is to use an open arrowhead

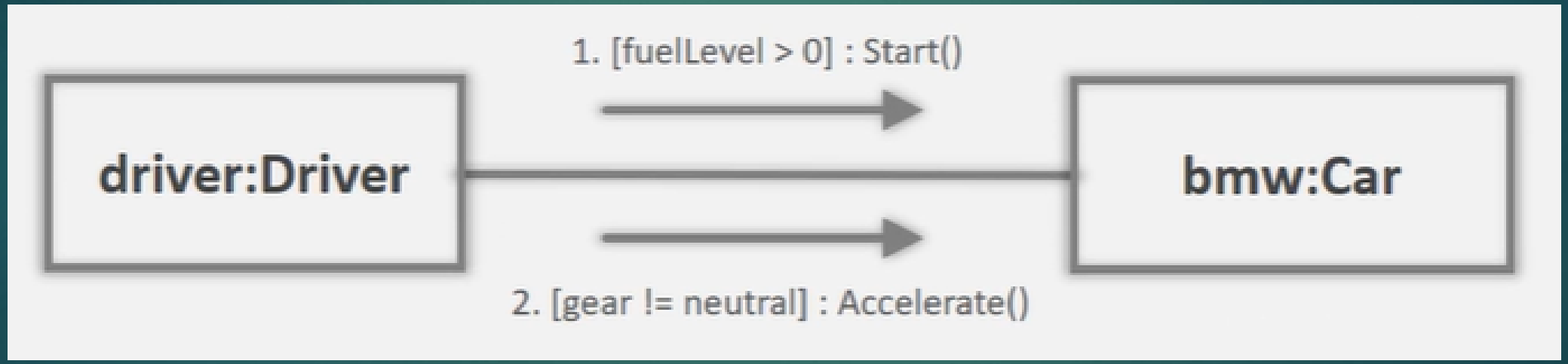



```
public void Brake() {
    Thread t1 = new Thread(brakeSystem.Engage);
    Thread t2 = new Thread(brkLt.SwitchOn);

    t1.Start();
    t2.Start();

    t1.Join();
    t2.Join();
    //Rest of the code
```

Conditional Message

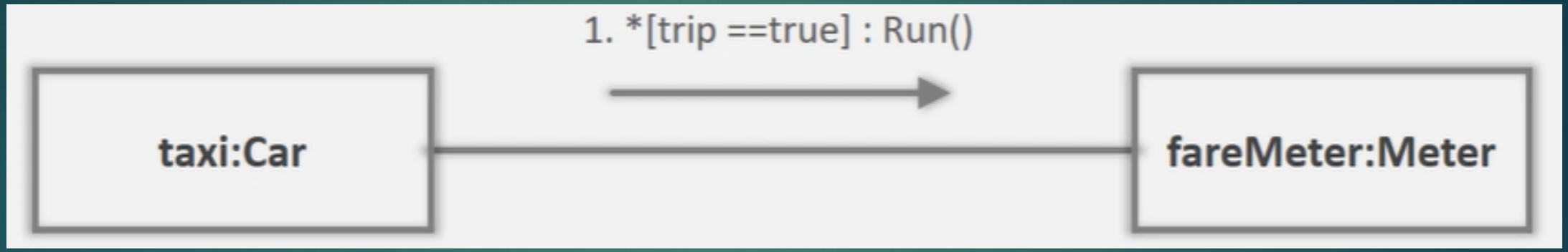
- Some messages have to be sent based on a condition
- The conditions for such messages are represented as guards
- A guard is made up of a Boolean statement, usually represented through pseudocode




1. [fuelLevel > 0] : Start()
driver:Driver
bmw:Car
2. [gear != neutral] : Accelerate()

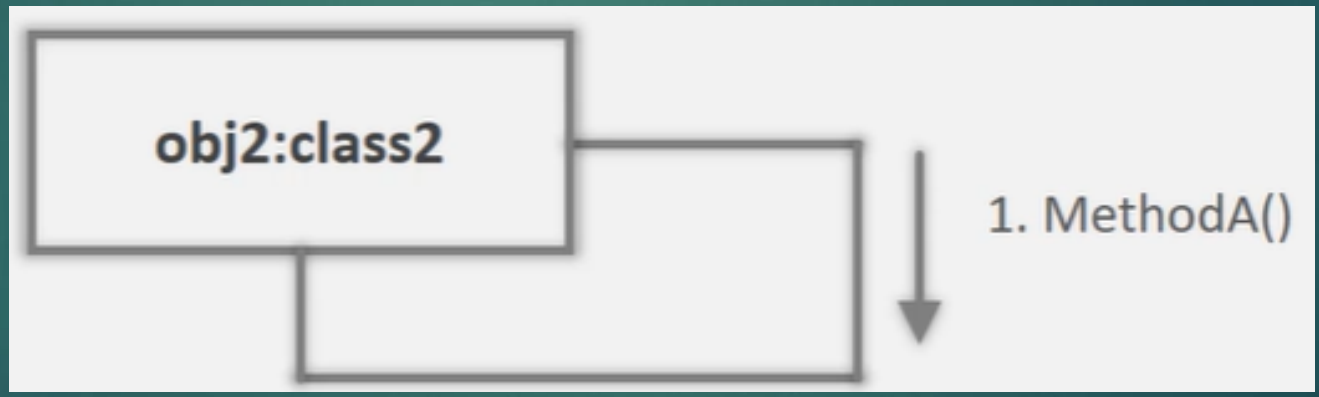
Looping Messages

- A looping message is shown with an asterisk (*) before a looping constraints
- This constraint is represented just like a guard statement
- May contain a condition with Boolean statement or counter variable


1. *[trip ==true] : Run()
taxi:Car
fareMeter:Meter

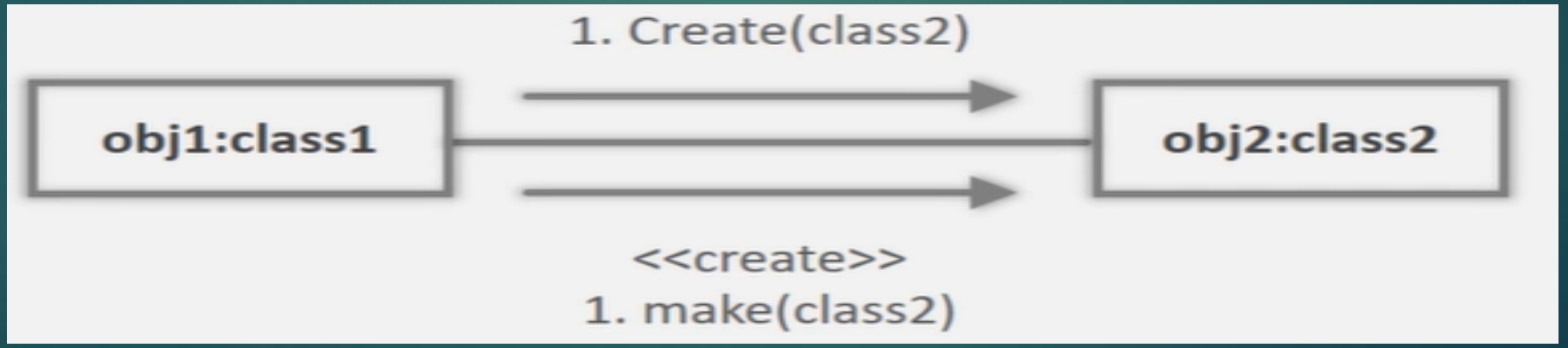
Messages to Self

- An object may call a method on itself
  - E.g. a method may call a private internal method
  - Represented through the link from participant to itself with a message



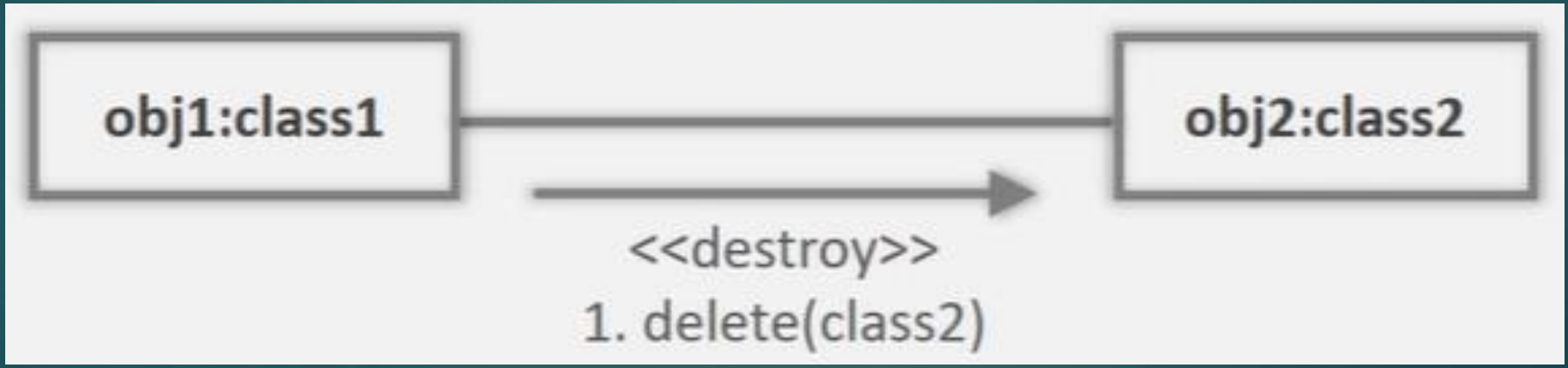
Create Message

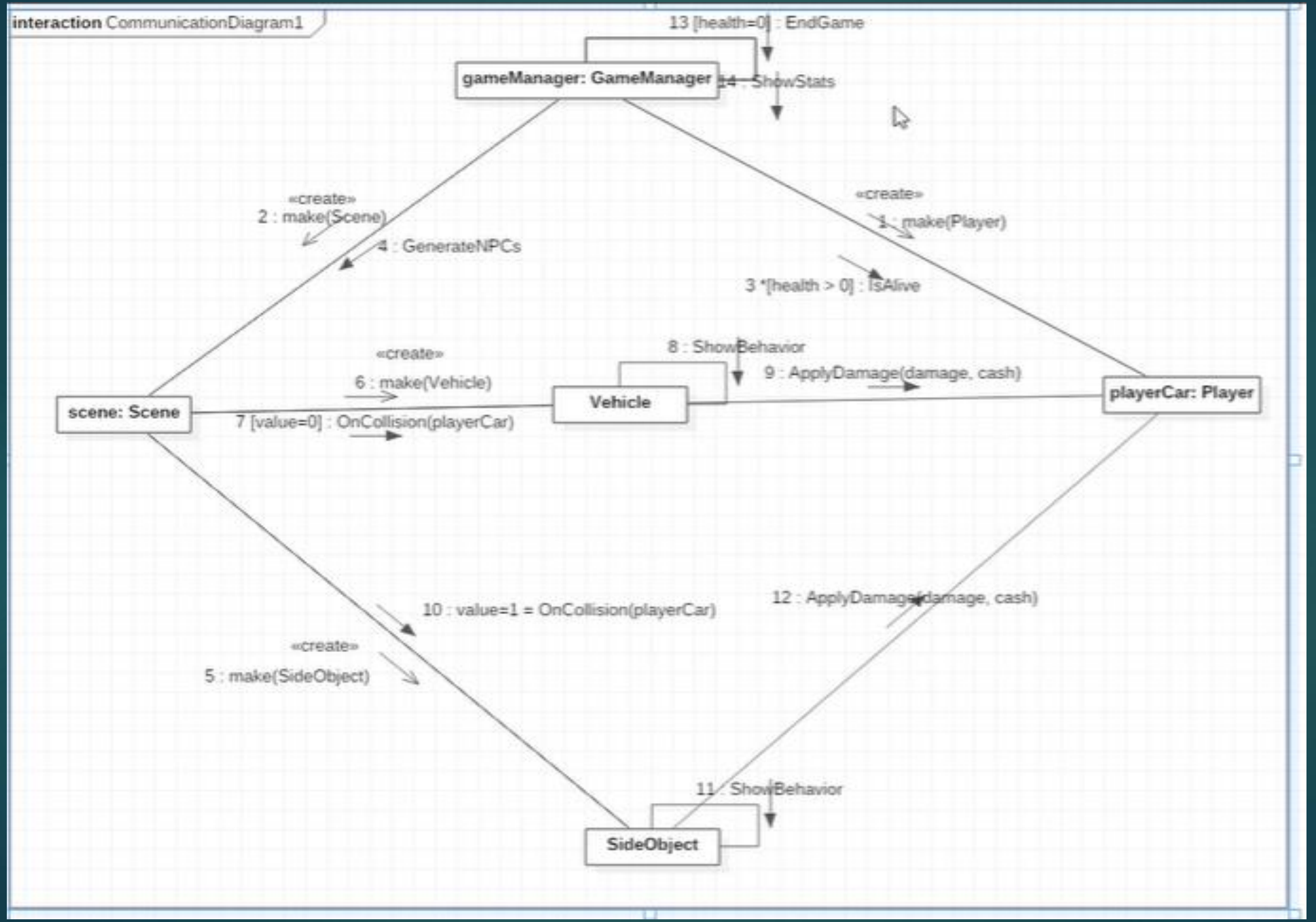
- This message creates an instance of a class
- It can be represented in multiple ways
  - Message name is Create with argument as the name of the class (signifies call to constructor)
  - Use the stereotype <<create>> on the message with any name



Destroy Message

- This message destroys an instance of a class
- Use the stereotype <<destroy>> on the message
- Can be used to show explicit destruction of an object e.g. in C++, we use the delete operator




interaction CommunicationDiagram1
13 [health=0] : EndGame
gameManager: GameManager
14 : ShowStats
«create»
2 : make(Scene)
«create»
1 : make(Player)
4 : GenerateNPCs
3 *[health > 0] : IsAlive
8 : ShowBehavior
«create»
6 : make(Vehicle)
9 : ApplyDamage(damage, cash)
scene: Scene
Vehicle
playerCar: Player
7 [value=0] : OnCollision(playerCar)
10 : value=1 = OnCollision(playerCar)
12 : ApplyDamage(damage, cash)
«create»
5 : make(SideObject)
11 : ShowBehavior
SideObject