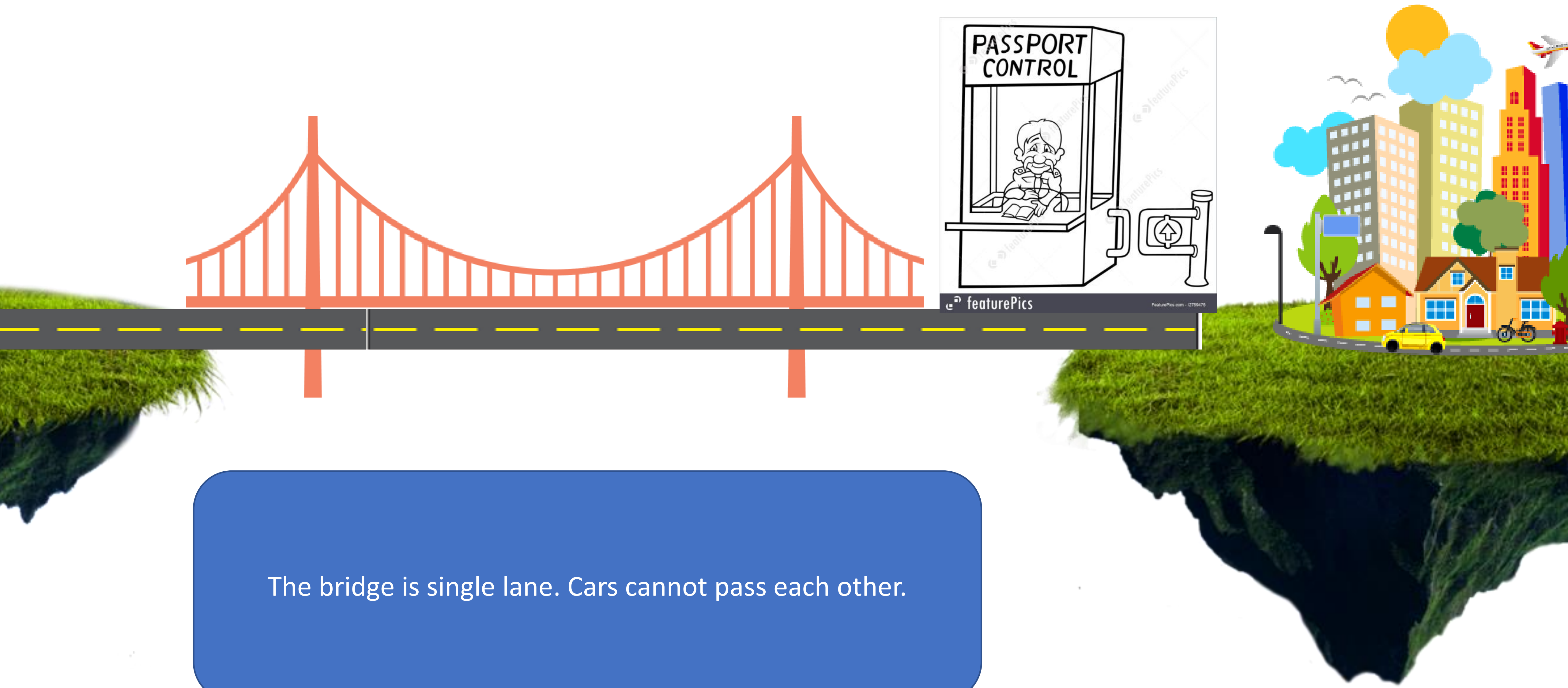
Course Assignment 3

The idea, Bridgeville

The idea

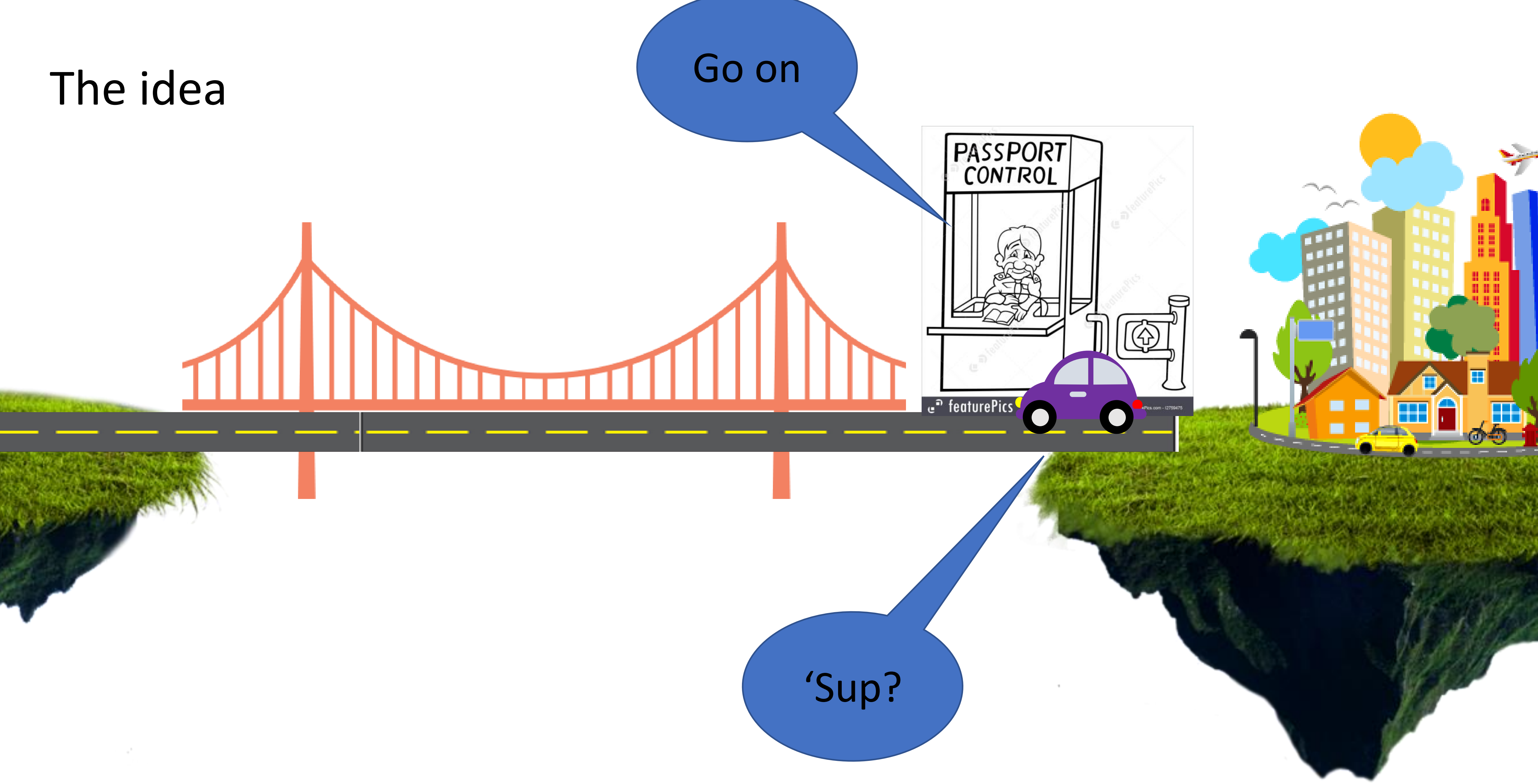
When you exit the city, you just say your ID.
When you enter again, the guy must validate more of your info, to make sure only the right people come in.

The idea

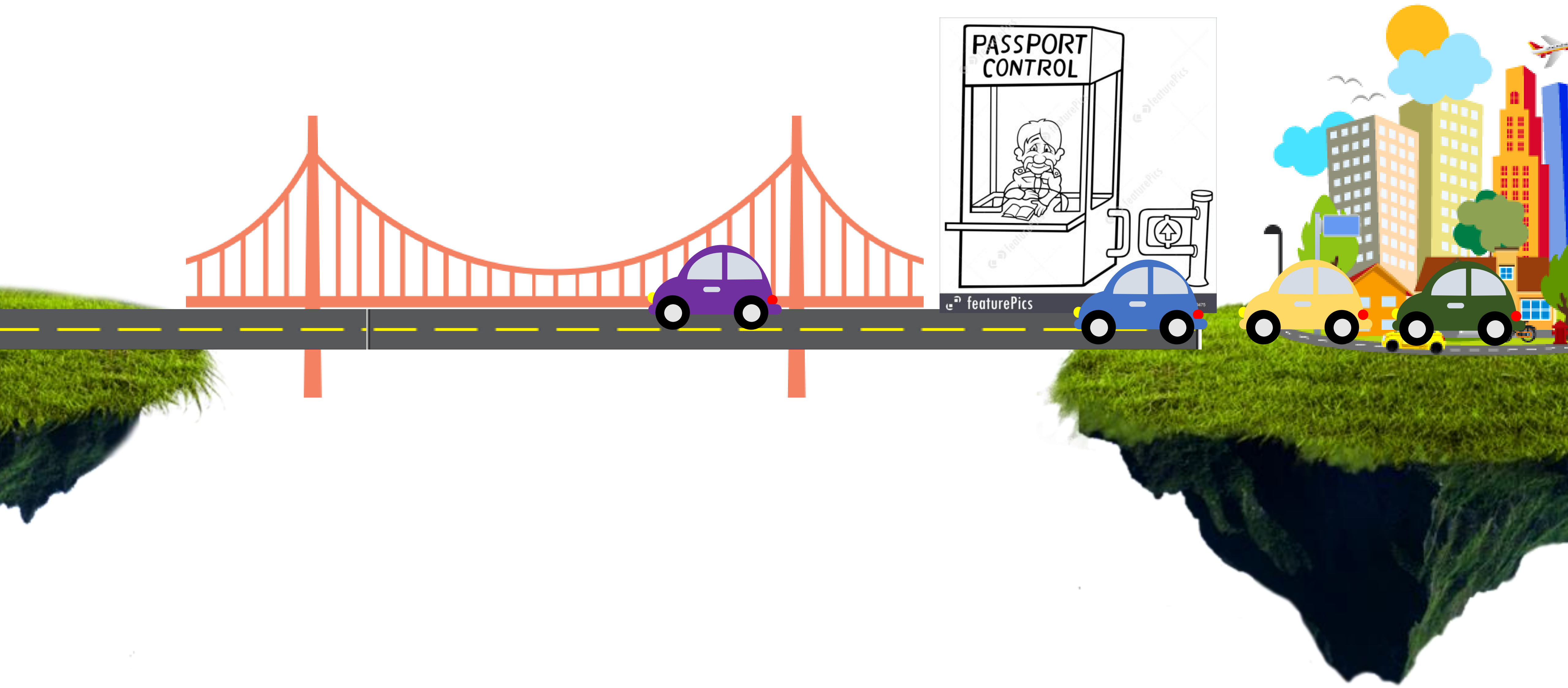


The bridge is single lane. Cars cannot pass each other.

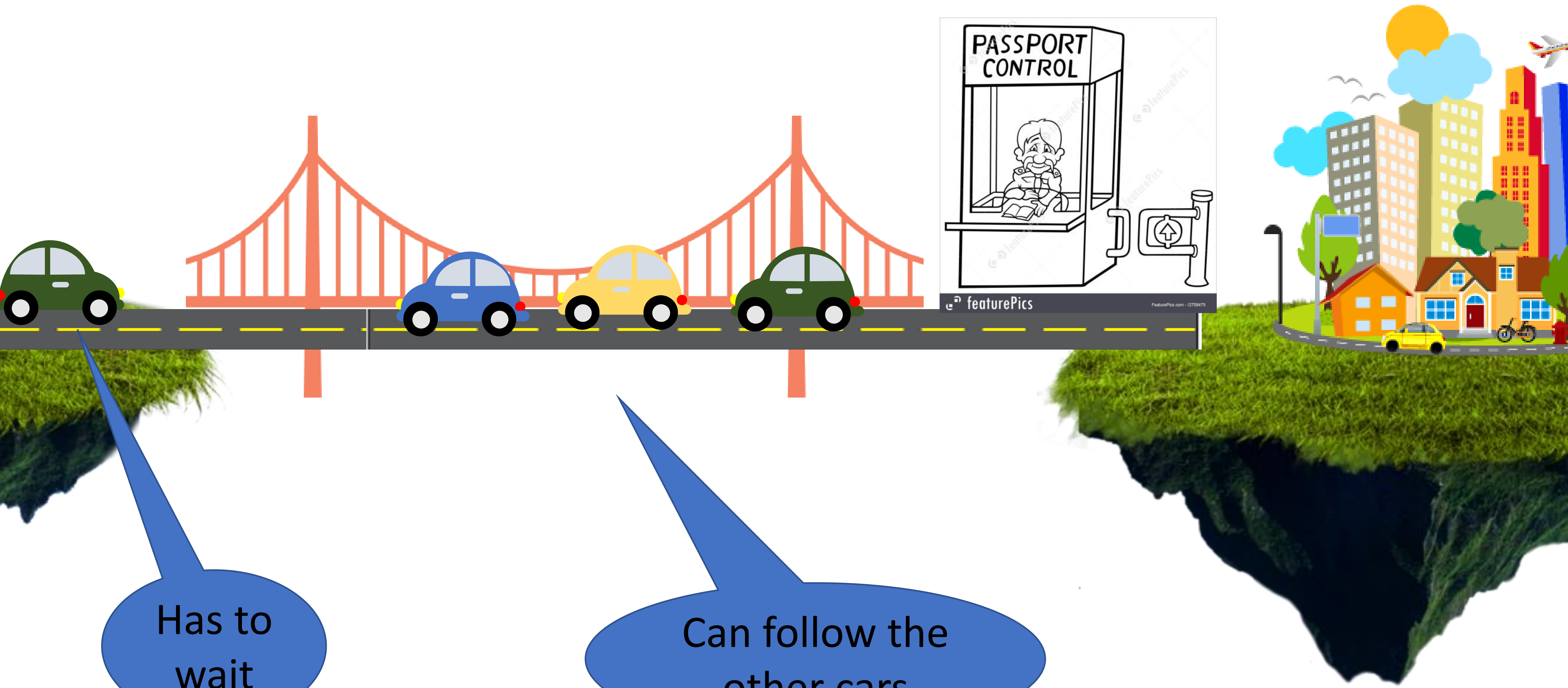
The idea



The idea



The idea



Has to wait

Can follow the other cars

The idea

The idea

The idea

The idea

The idea

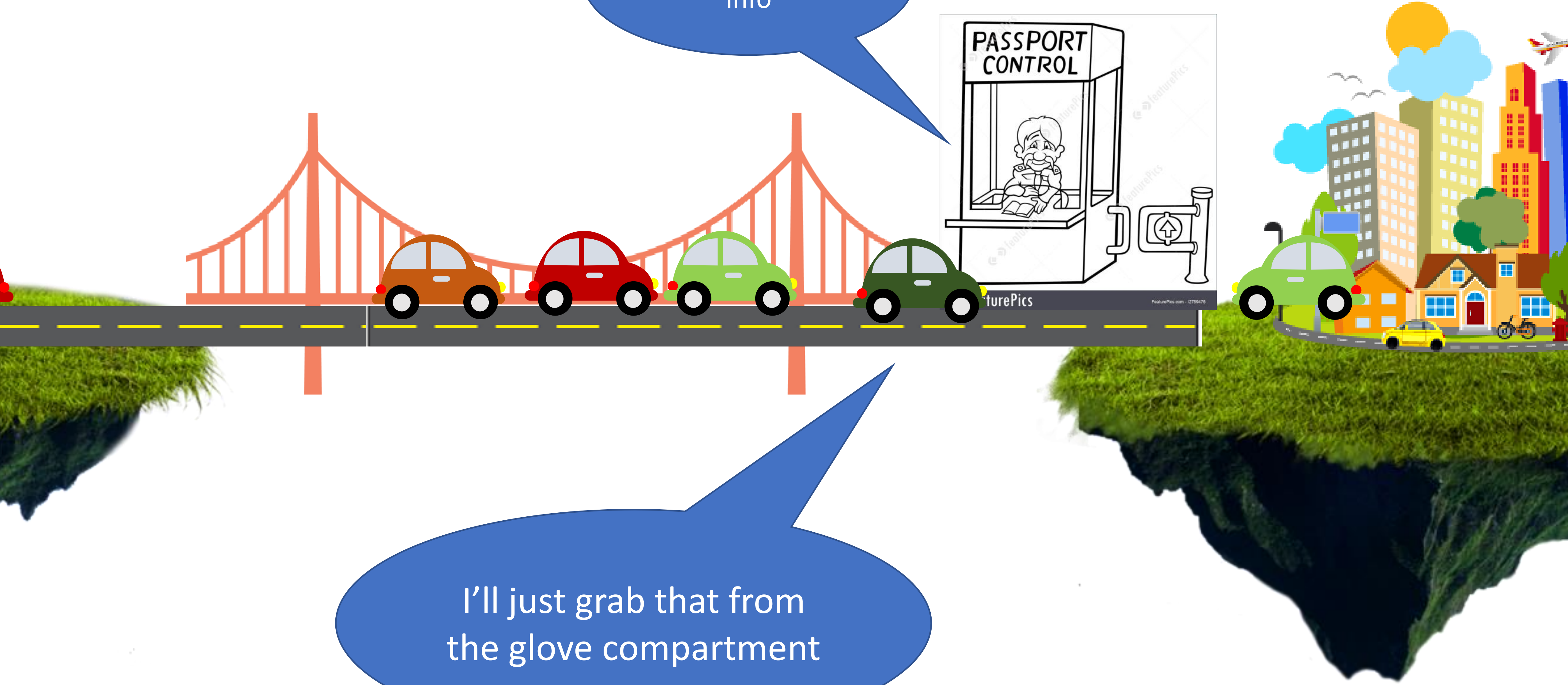
Road clear, left side can go

The idea

PASSPORT CONTROL
featurePics
Has to wait

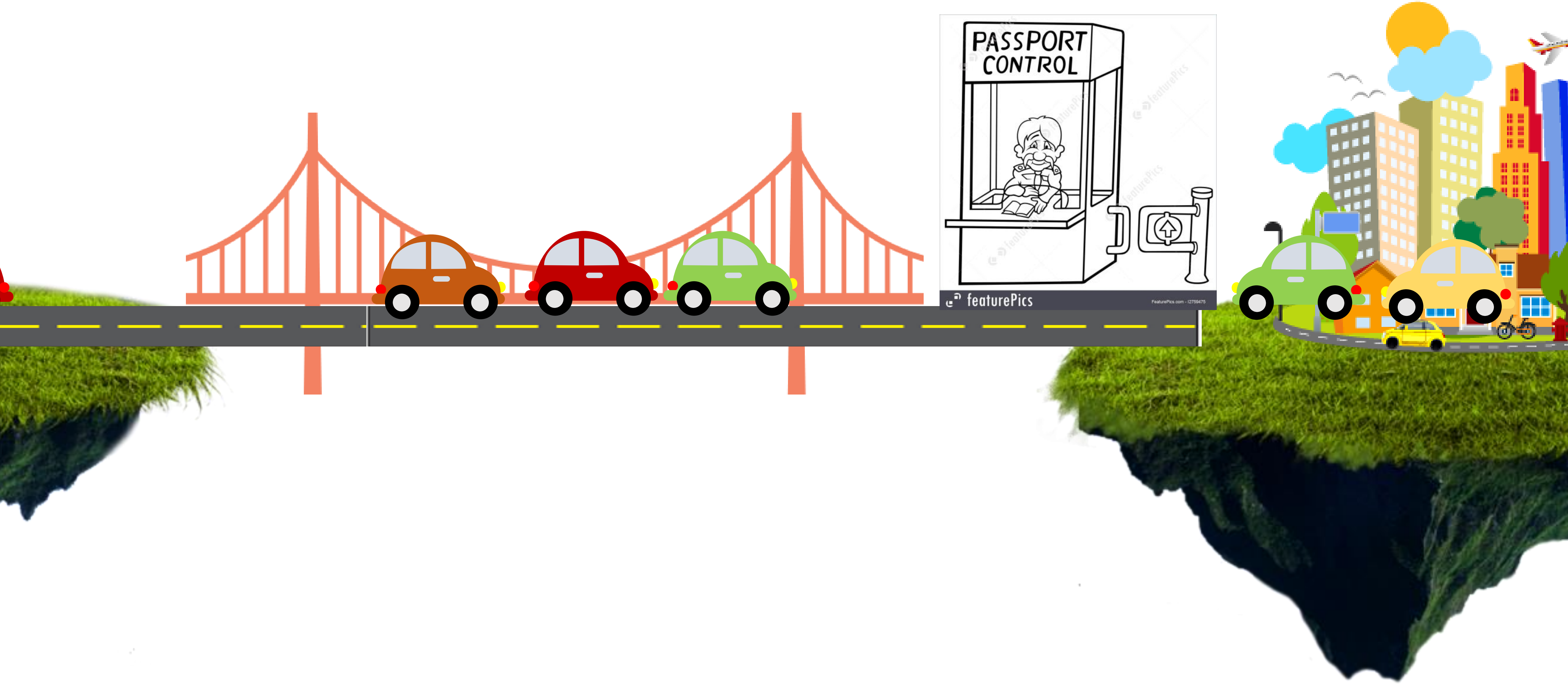
The idea

Show me your info



I'll just grab that from the glove compartment

The idea



The idea

PASSPORT CONTROL
featurePics

Road clear, right side can go again

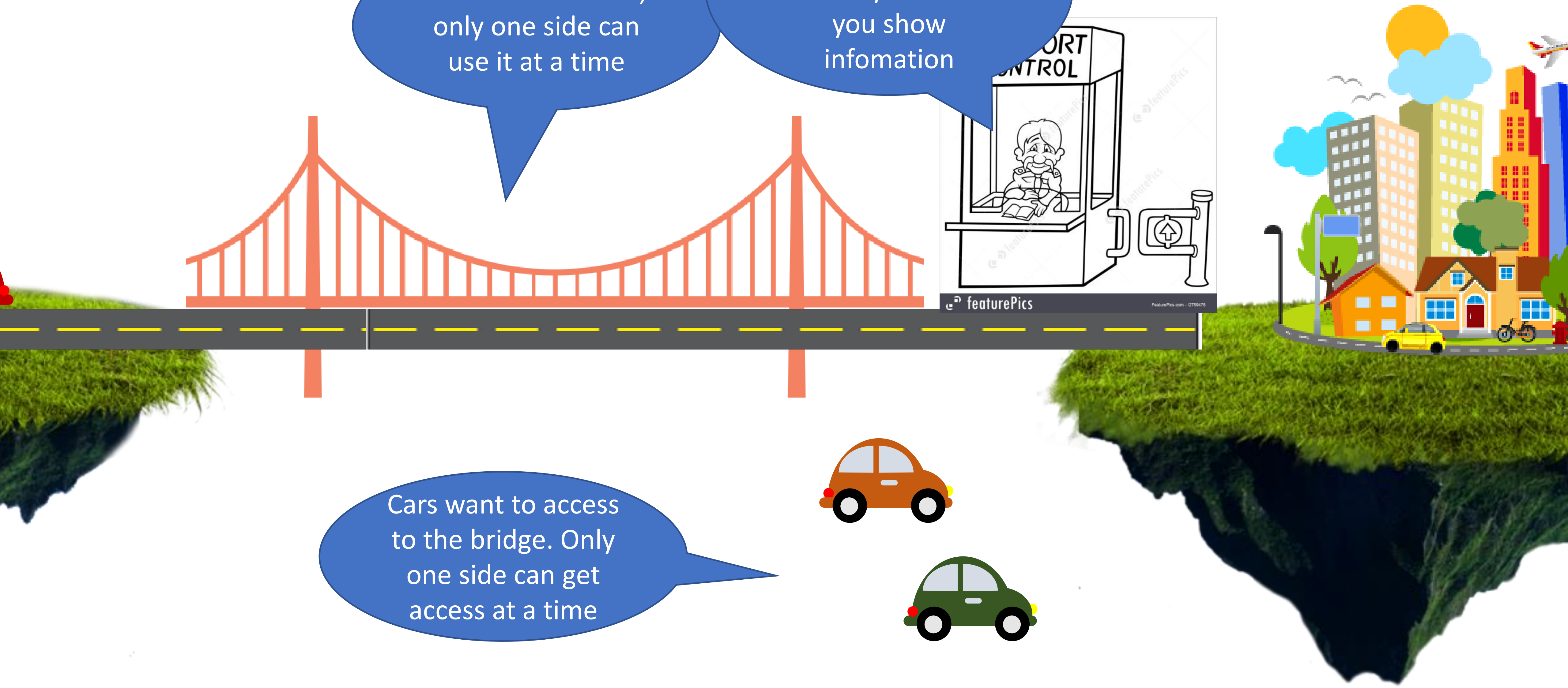
The idea

Road clear, right side can go again

The idea

“shared resource”, only one side can use it at a time

When you exit you just show ID. When you enter you show infomation



Cars want to access to the bridge. Only one side can get access at a time

Topics covered

- Readers & writers (red and blue car crossing bridge exercise)
- Proxy
- Flyweight
- Unit testing

Topics covered

- Readers & writers
  - Single lane bridge. Only one type of thread can get access at a time: Left side cars or right side cars. Your usual readers/writers, but there can be multiple writers. Either Readers have access, or Writers have access, but not at the same time
- Proxy
- Flyweight
- Unit testing

Topics covered

- Readers & writers
- Proxy
  - The Car is an interface. Initially a ProxyCar is used. When the Car reaches the border control, extra information is loaded from a file. We use lazy instantiation, so this information is loaded into the RealCar only when needed.
- Flyweight
- Unit testing

Topics covered

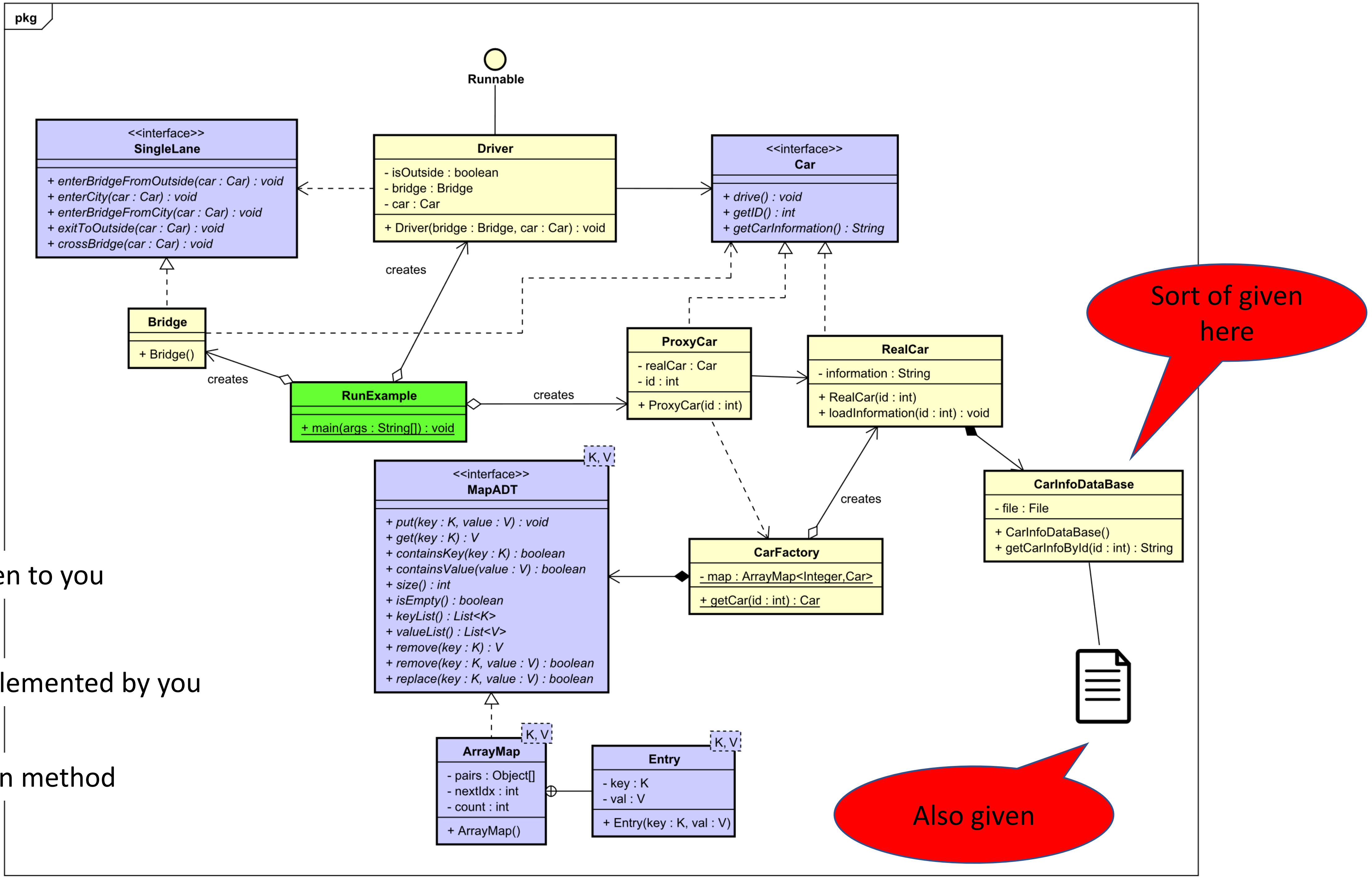
- Readers & writers
- Proxy
- Flyweight
    - Used to return RealCars. There are 10 different legal types of cars in the BridgeVille.
- Unit testing

Topics covered

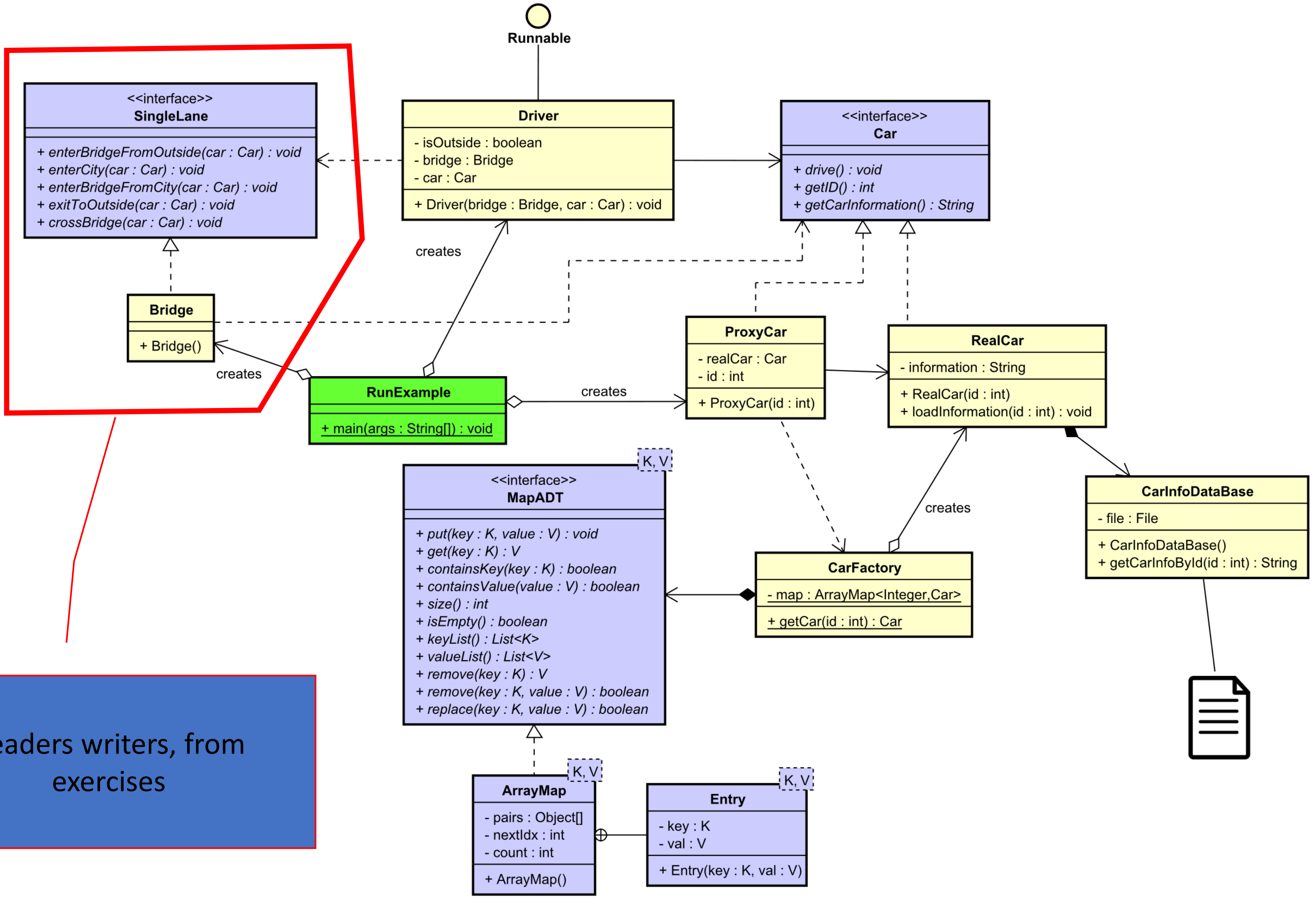
- Readers & writers
- Proxy
- Flyweight
- Unit testing
  - Do testing of the map collection used in the flyweight.

The UML

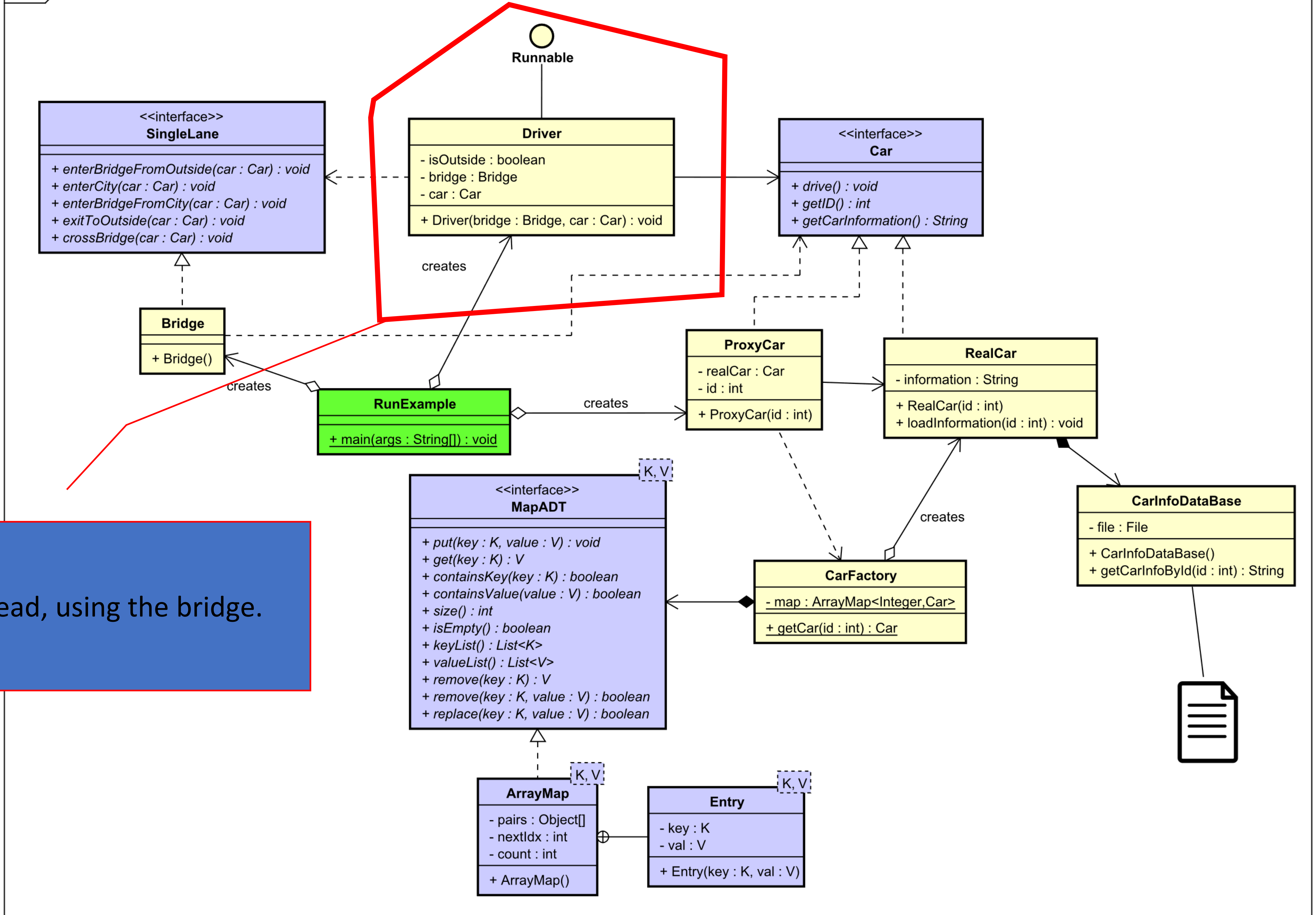
- First the different parts, the overview
- Then details



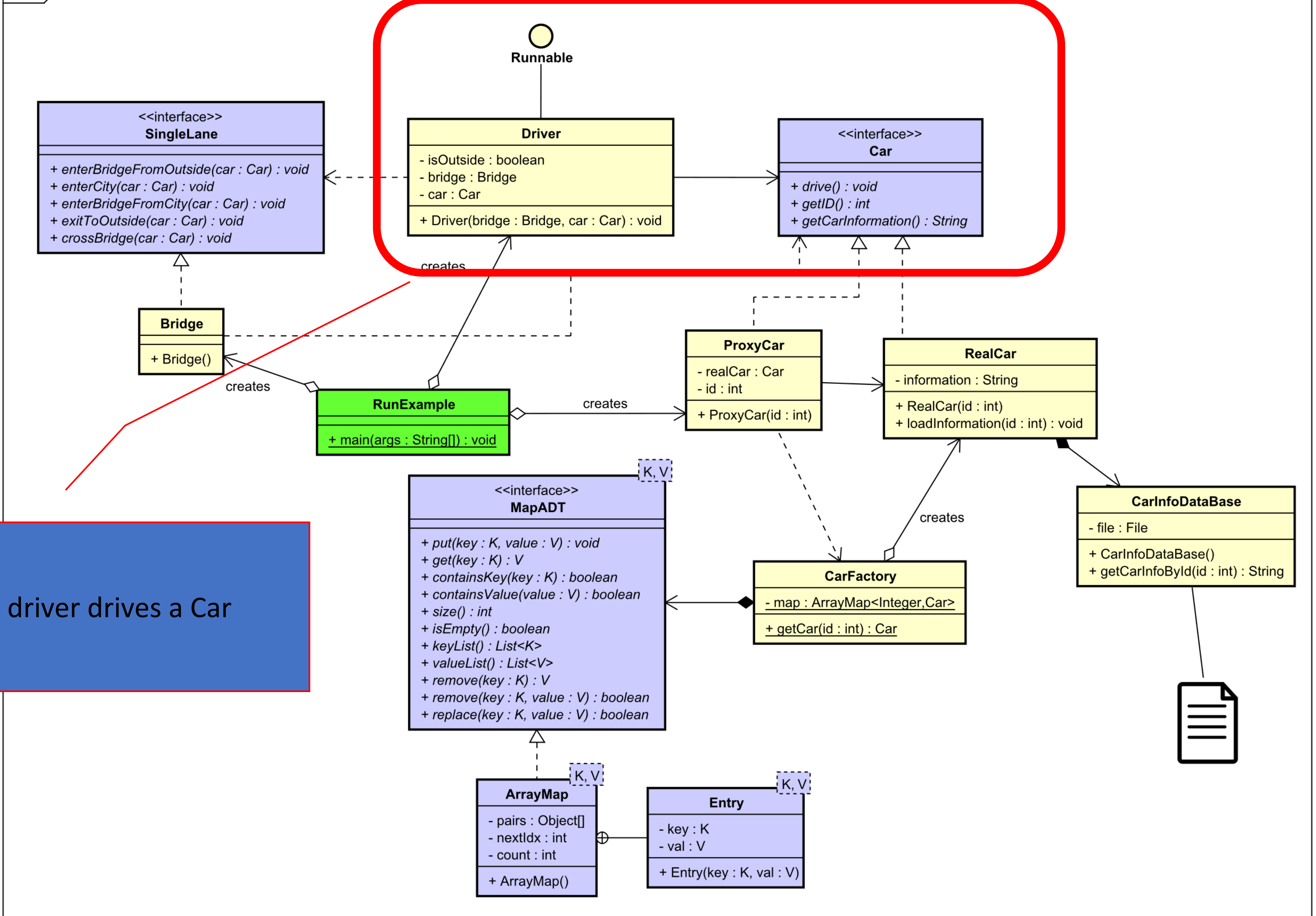
🟦	Given to you
🟨	Implemented by you
🟩	Main method



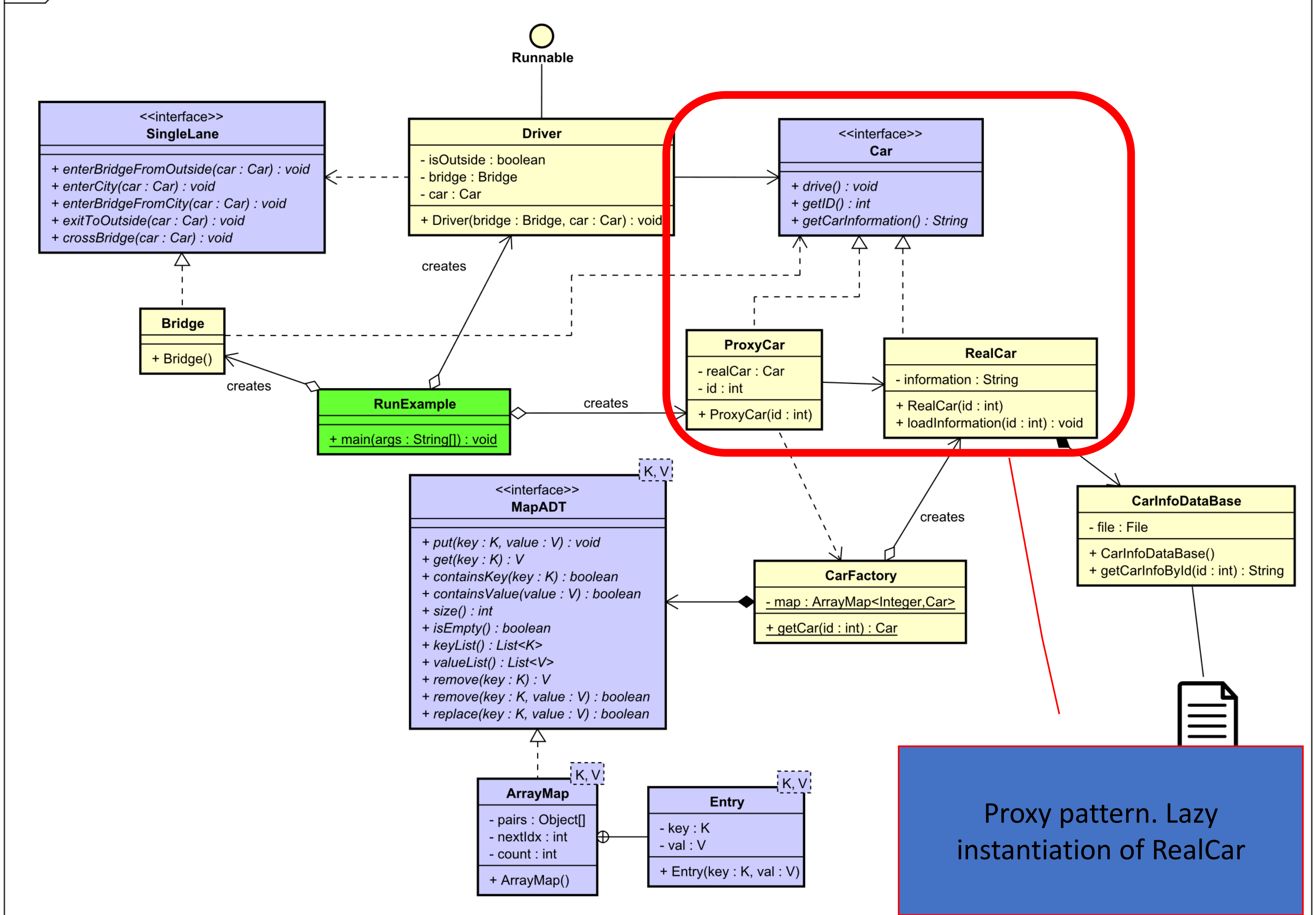
Readers writers, from exercises



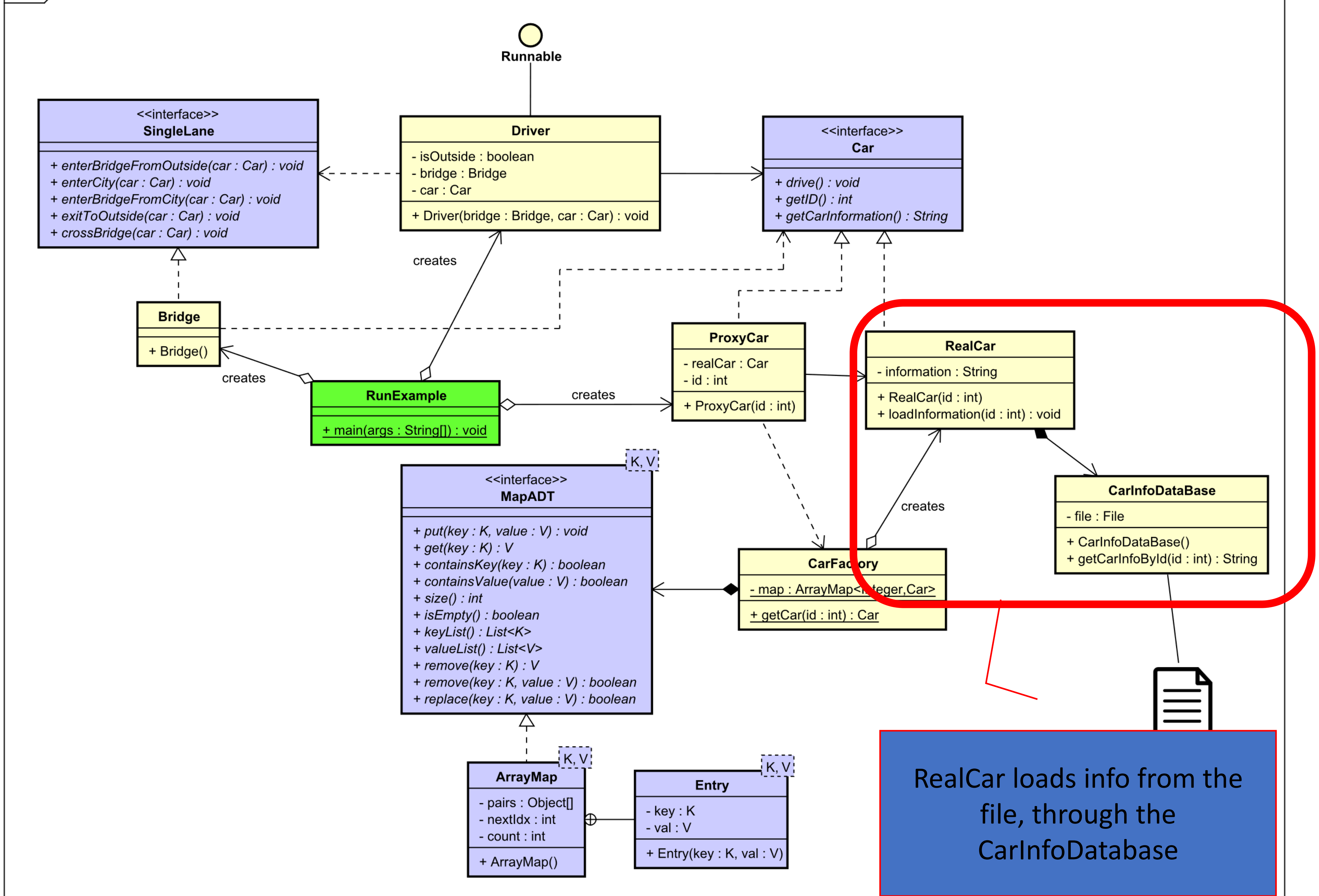
Thread, using the bridge.



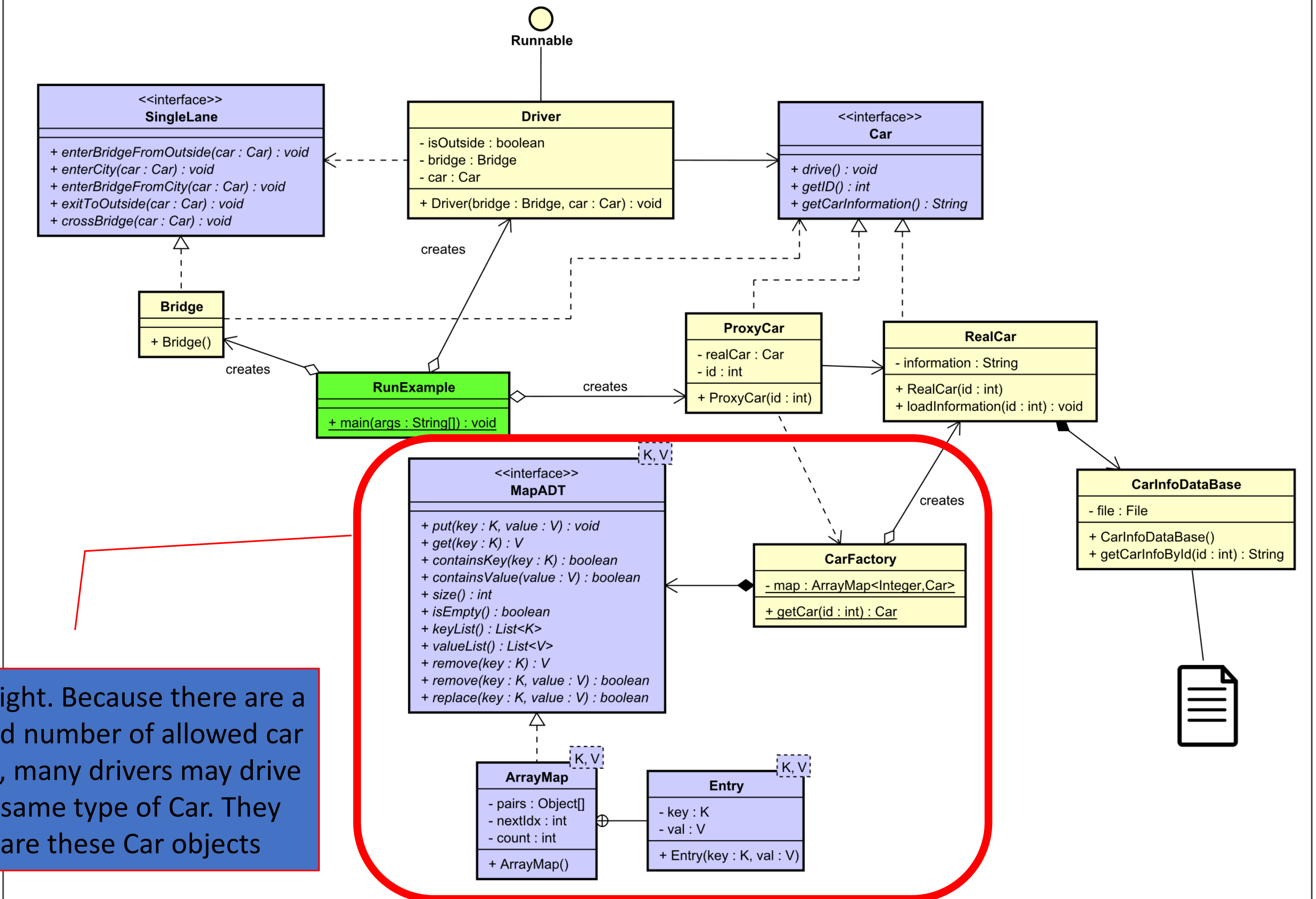
A driver drives a Car



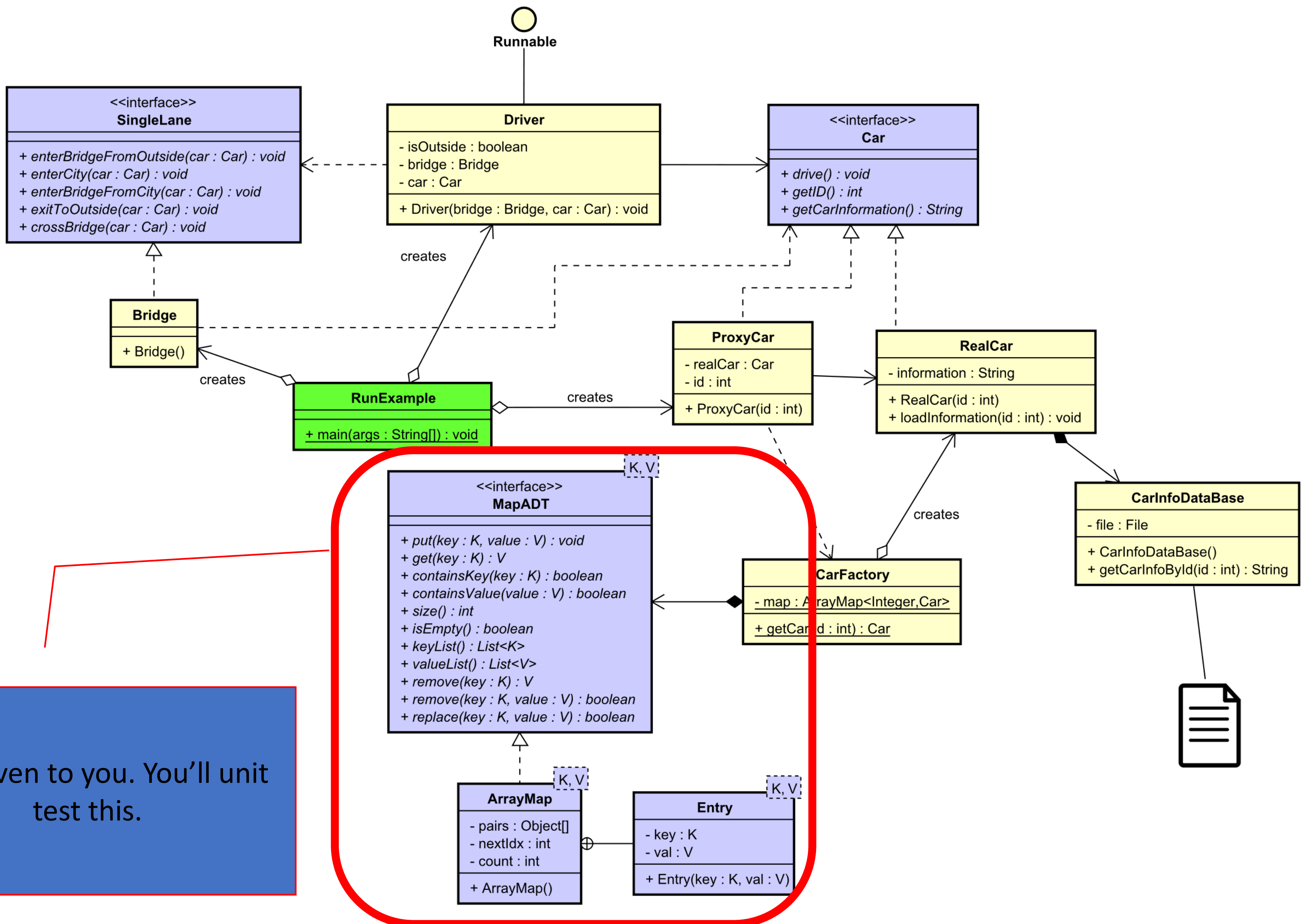
Proxy pattern. Lazy instantiation of RealCar



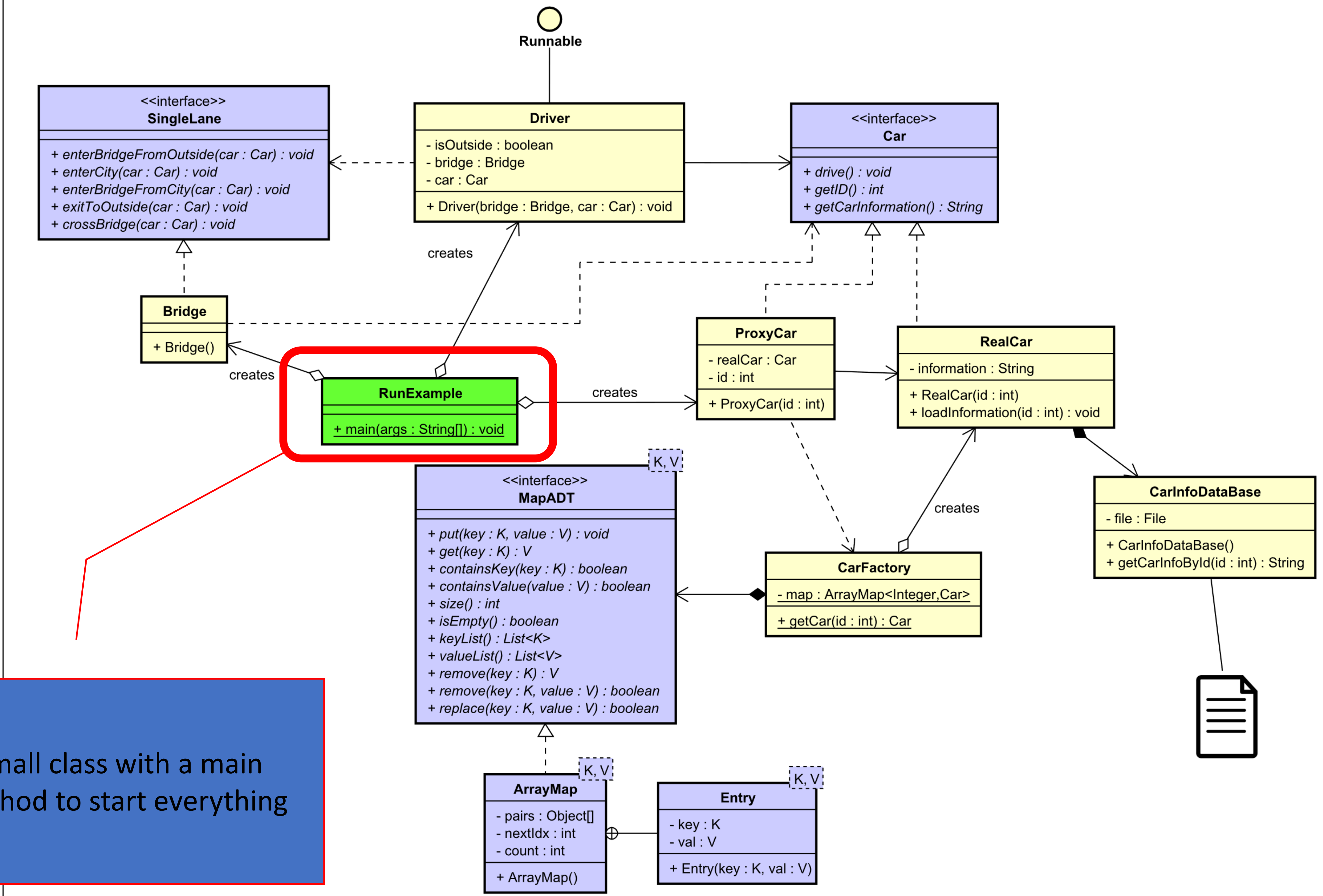
RealCar loads info from the file, through the CarInfoDatabase



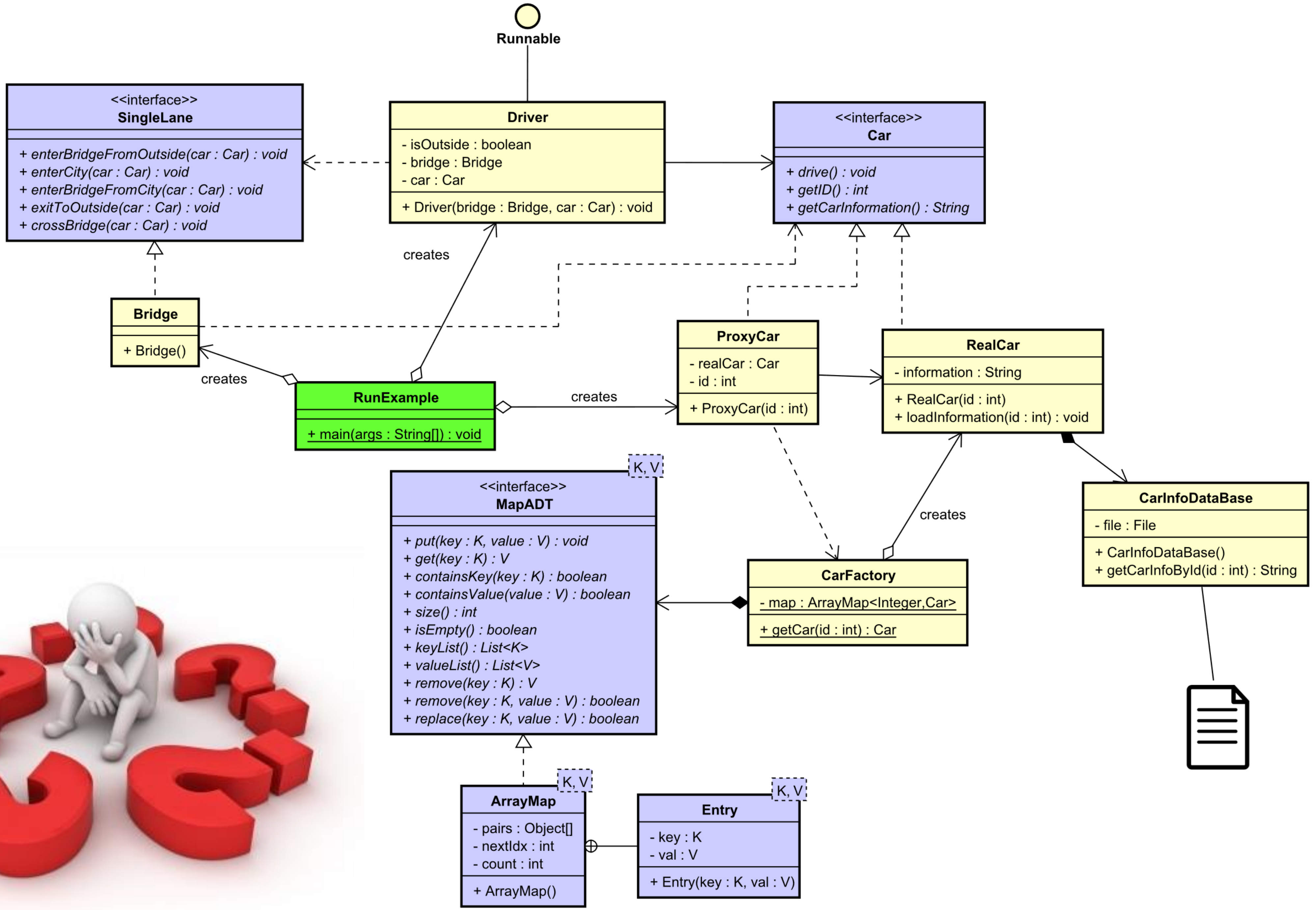
Flyweight. Because there are a limited number of allowed car types, many drivers may drive the same type of Car. They share these Car objects


Runnable
<<interface>>
SingleLane
+ enterBridgeFromOutside(car : Car) : void
+ enterCity(car : Car) : void
+ enterBridgeFromCity(car : Car) : void
+ exitToOutside(car : Car) : void
+ crossBridge(car : Car) : void
Driver
- isOutside : boolean
- bridge : Bridge
- car : Car
+ Driver(bridge : Bridge, car : Car) : void
<<interface>>
Car
+ drive() : void
+ getID() : int
+ getCarInformation() : String
creates
Bridge
+ Bridge()
creates
RunExample
+ main(args : String[]) : void
creates
ProxyCar
- realCar : Car
- id : int
+ ProxyCar(id : int)
RealCar
- information : String
+ RealCar(id : int)
+ loadInformation(id : int) : void
K, V
<<interface>>
MapADT
+ put(key : K, value : V) : void
+ get(key : K) : V
+ containsKey(key : K) : boolean
+ containsValue(value : V) : boolean
+ size() : int
+ isEmpty() : boolean
+ keyList() : List<K>
+ valueList() : List<V>
+ remove(key : K) : V
+ remove(key : K, value : V) : boolean
+ replace(key : K, value : V) : boolean
creates
CarFactory
- map : ArrayMap<Integer,Car>
+ getCarId(id : int) : Car
CarInfoDataBase
- file : File
+ CarInfoDataBase()
+ getCarInfoById(id : int) : String
K, V
ArrayMap
- pairs : Object[]
- nextIdx : int
- count : int
+ ArrayMap()
K, V
Entry
- key : K
- val : V
+ Entry(key : K, val : V)

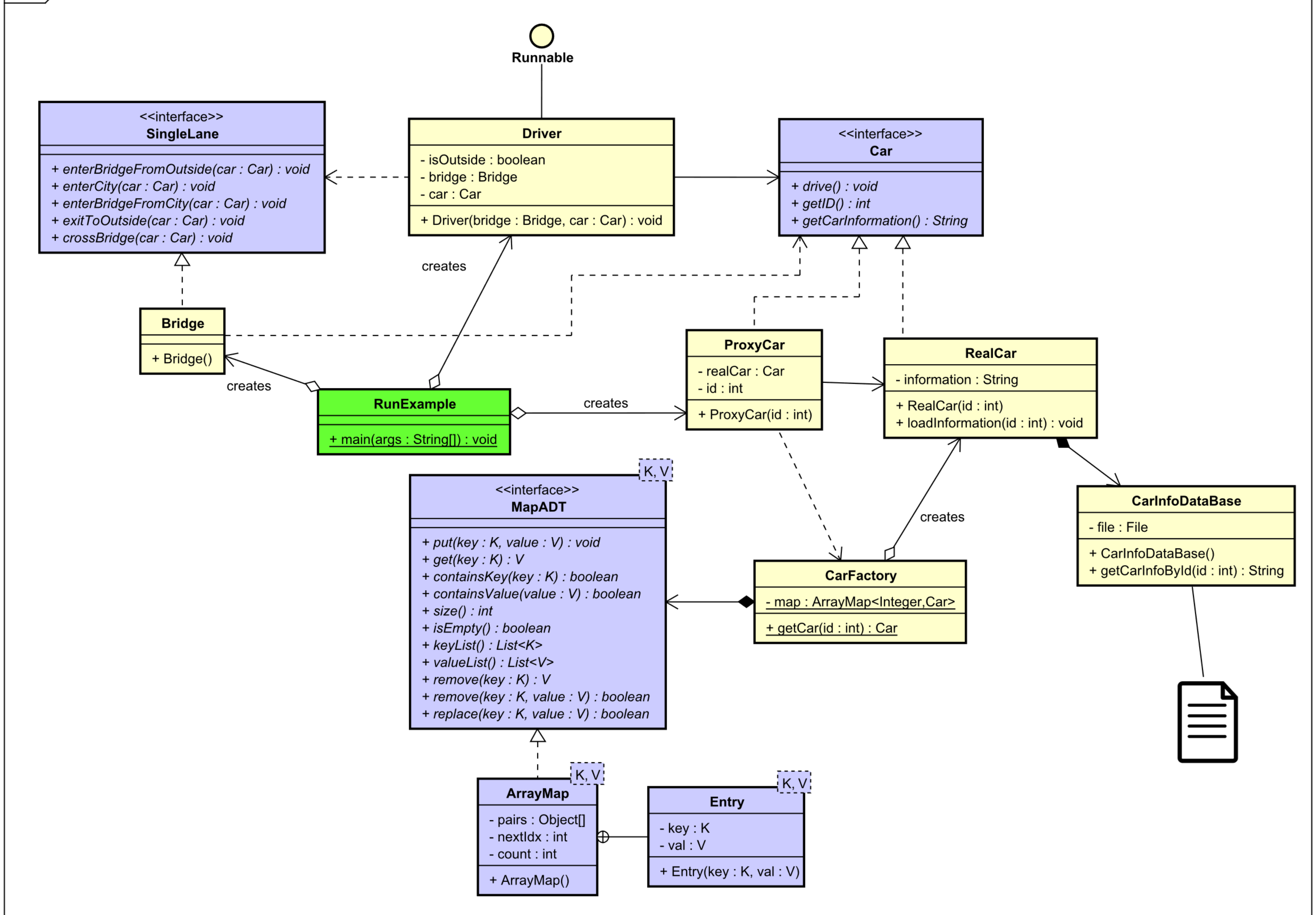
Map given to you. You’ll unit test this.

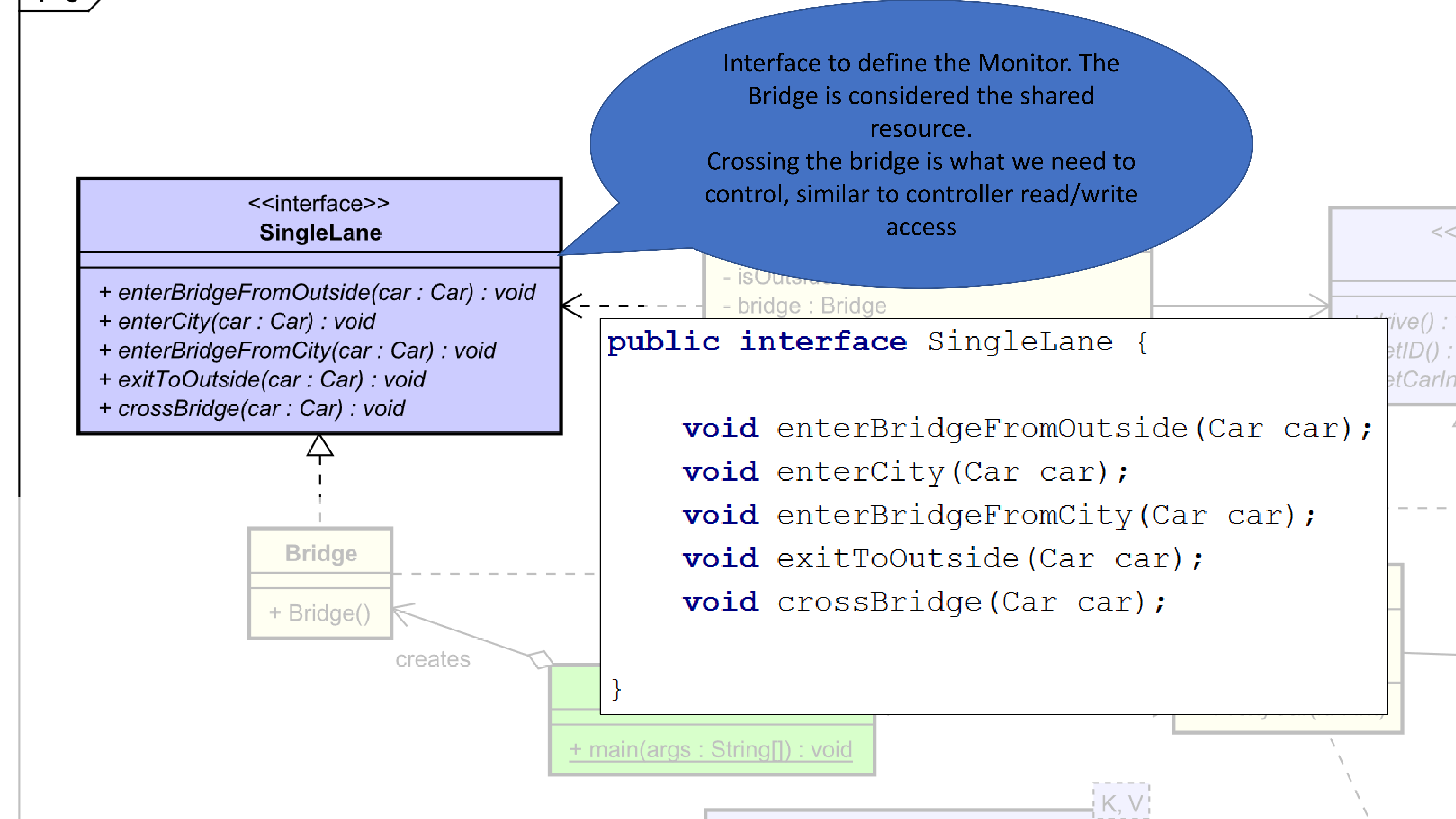


Small class with a main method to start everything


Runnable
<<interface>>
SingleLane
+ enterBridgeFromOutside(car : Car) : void
+ enterCity(car : Car) : void
+ enterBridgeFromCity(car : Car) : void
+ exitToOutside(car : Car) : void
+ crossBridge(car : Car) : void
Driver
- isOutside : boolean
- bridge : Bridge
- car : Car
+ Driver(bridge : Bridge, car : Car) : void
<<interface>>
Car
+ drive() : void
+ getID() : int
+ getCarInformation() : String
creates
Bridge
+ Bridge()
creates
RunExample
+ main(args : String[]) : void
creates
ProxyCar
- realCar : Car
- id : int
+ ProxyCar(id : int)
RealCar
- information : String
+ RealCar(id : int)
+ loadInformation(id : int) : void
K, V
<<interface>>
MapADT
+ put(key : K, value : V) : void
+ get(key : K) : V
+ containsKey(key : K) : boolean
+ containsValue(value : V) : boolean
+ size() : int
+ isEmpty() : boolean
+ keyList() : List<K>
+ valueList() : List<V>
+ remove(key : K) : V
+ remove(key : K, value : V) : boolean
+ replace(key : K, value : V) : boolean
creates
CarFactory
- map : ArrayMap<Integer,Car>
+ getCar(id : int) : Car
CarInfoDataBase
- file : File
+ CarInfoDataBase()
+ getCarInfoById(id : int) : String
K, V
ArrayMap
- pairs : Object[]
- nextIdx : int
- count : int
+ ArrayMap()
K, V
Entry
- key : K
- val : V
+ Entry(key : K, val : V)

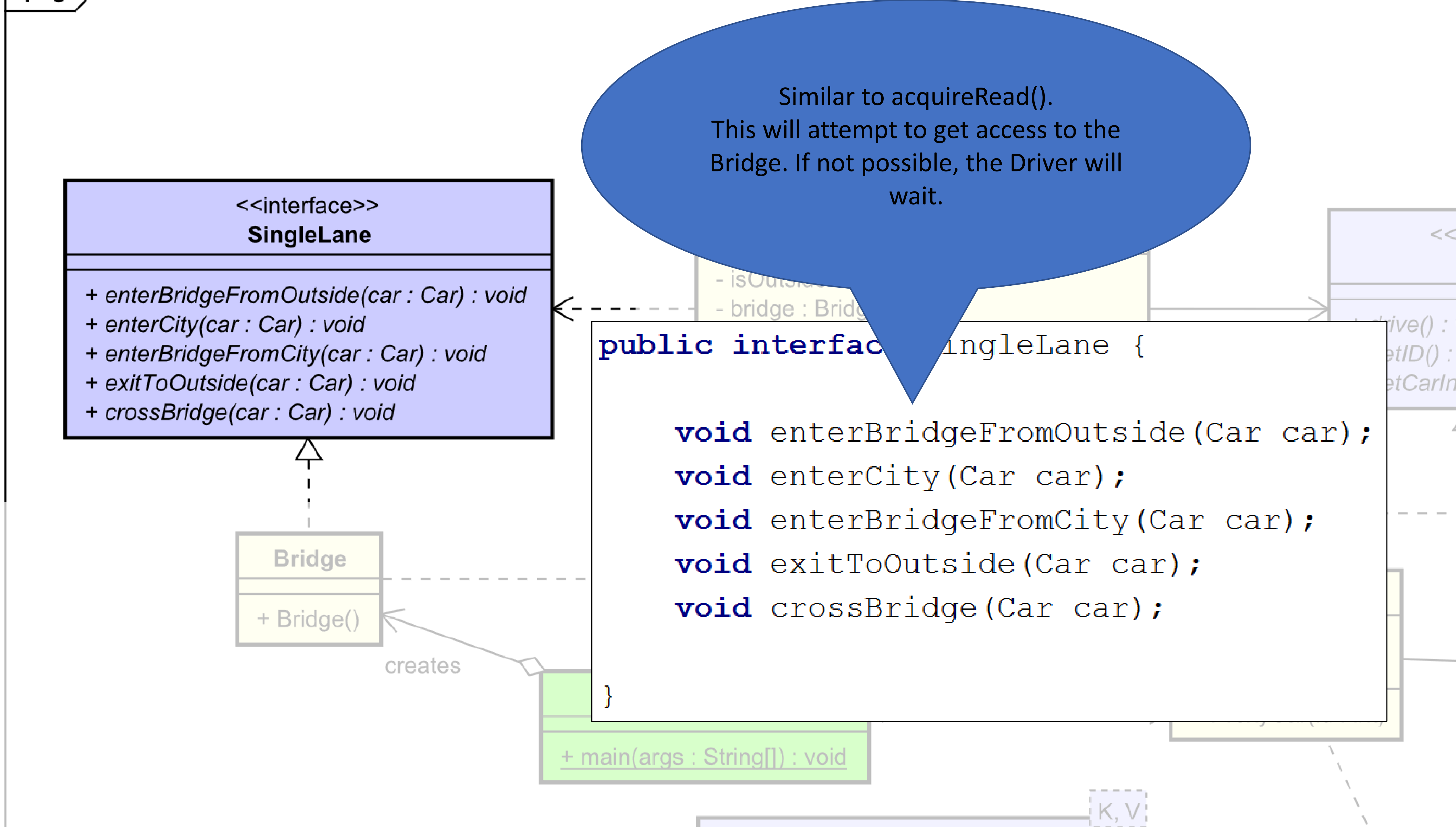
The details


Runnable
<<interface>>
SingleLane
+ enterBridgeFromOutside(car : Car) : void
+ enterCity(car : Car) : void
+ enterBridgeFromCity(car : Car) : void
+ exitToOutside(car : Car) : void
+ crossBridge(car : Car) : void
Driver
- isOutside : boolean
- bridge : Bridge
- car : Car
+ Driver(bridge : Bridge, car : Car) : void
<<interface>>
Car
+ drive() : void
+ getID() : int
+ getCarInformation() : String
creates
Bridge
+ Bridge()
creates
RunExample
+ main(args : String[]) : void
creates
ProxyCar
- realCar : Car
- id : int
+ ProxyCar(id : int)
RealCar
- information : String
+ RealCar(id : int)
+ loadInformation(id : int) : void
K, V
<<interface>>
MapADT
+ put(key : K, value : V) : void
+ get(key : K) : V
+ containsKey(key : K) : boolean
+ containsValue(value : V) : boolean
+ size() : int
+ isEmpty() : boolean
+ keyList() : List<K>
+ valueList() : List<V>
+ remove(key : K) : V
+ remove(key : K, value : V) : boolean
+ replace(key : K, value : V) : boolean
creates
CarFactory
- map : ArrayMap<Integer,Car>
+ getCar(id : int) : Car
CarInfoDataBase
- file : File
+ CarInfoDataBase()
+ getCarInfoById(id : int) : String
K, V
ArrayMap
- pairs : Object[]
- nextIdx : int
- count : int
+ ArrayMap()
K, V
Entry
- key : K
- val : V
+ Entry(key : K, val : V)


<<interface>>
SingleLane
+ enterBridgeFromOutside(car : Car) : void
+ enterCity(car : Car) : void
+ enterBridgeFromCity(car : Car) : void
+ exitToOutside(car : Car) : void
+ crossBridge(car : Car) : void
Bridge
+ Bridge()
creates
- bridge : Bridge
+ main(args : String[]) : void
K, V

Interface to define the Monitor. The Bridge is considered the shared resource.
Crossing the bridge is what we need to control, similar to controller read/write access

```
public interface SingleLane {

    void enterBridgeFromOutside(Car car);
    void enterCity(Car car);
    void enterBridgeFromCity(Car car);
    void exitToOutside(Car car);
    void crossBridge(Car car);

}
```

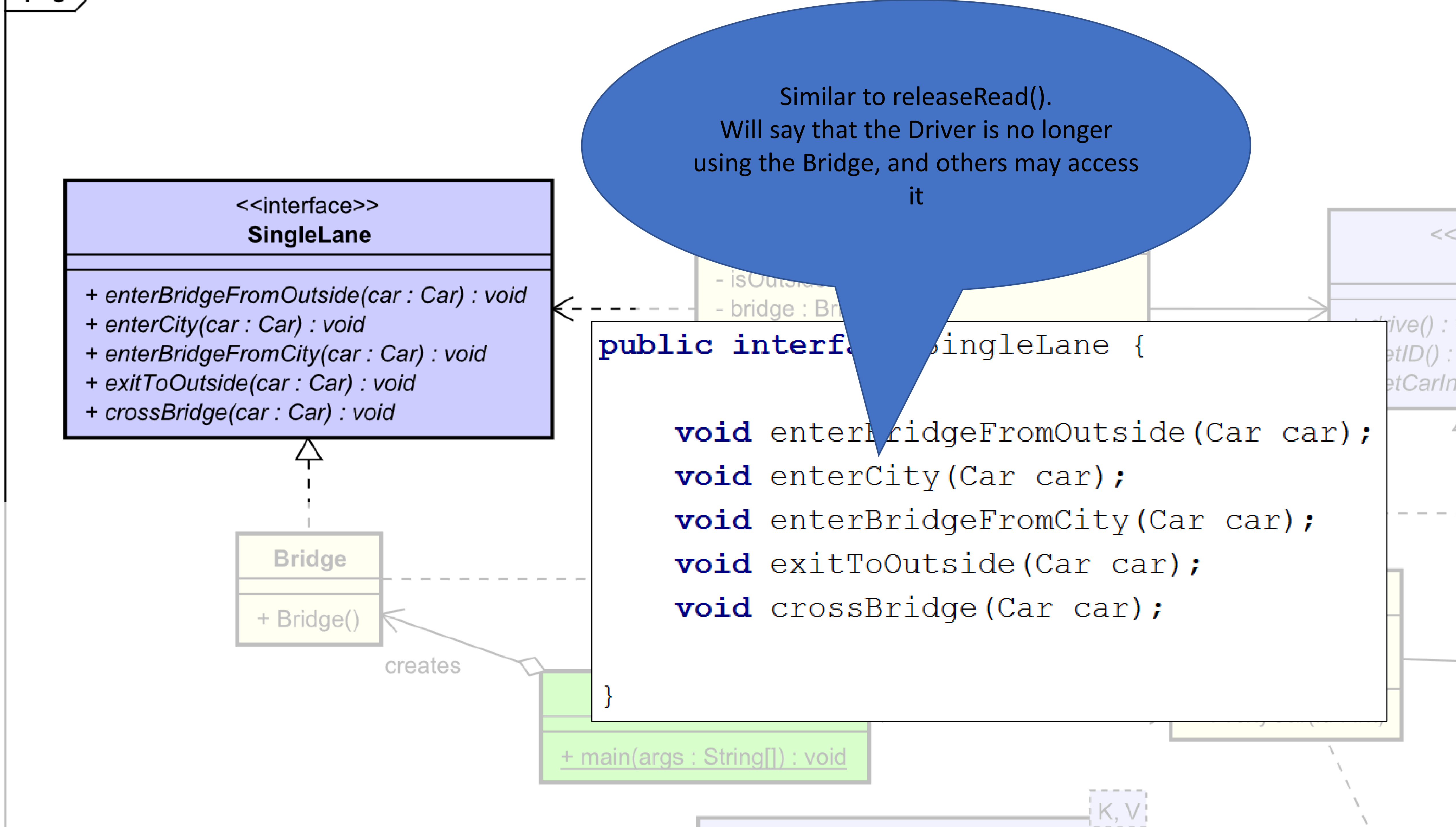


<<interface>>
SingleLane
+ enterBridgeFromOutside(car : Car) : void
+ enterCity(car : Car) : void
+ enterBridgeFromCity(car : Car) : void
+ exitToOutside(car : Car) : void
+ crossBridge(car : Car) : void
Bridge
+ Bridge()
creates
+ main(args : String[]) : void
- bridge : Bridg
K, V

Similar to acquireRead().
This will attempt to get access to the Bridge. If not possible, the Driver will wait.

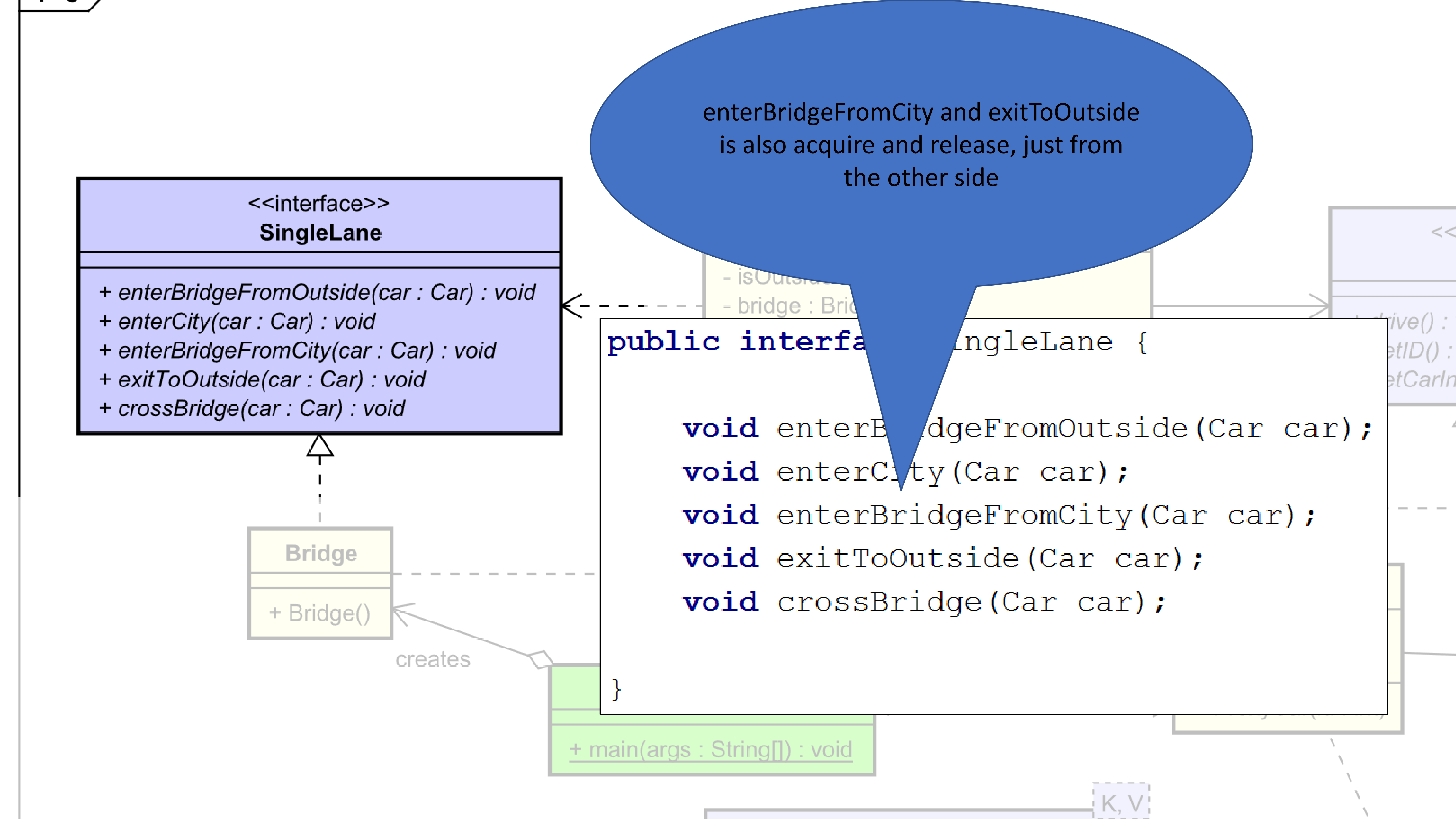
```
public interfac   ingleLane {

    void enterBridgeFromOutside(Car car);
    void enterCity(Car car);
    void enterBridgeFromCity(Car car);
    void exitToOutside(Car car);
    void crossBridge(Car car);

}
```

Similar to releaseRead().
Will say that the Driver is no longer using the Bridge, and others may access it


<<interface>>
SingleLane
+ enterBridgeFromOutside(car : Car) : void
+ enterCity(car : Car) : void
+ enterBridgeFromCity(car : Car) : void
+ exitToOutside(car : Car) : void
+ crossBridge(car : Car) : void
Bridge
+ Bridge()
creates
+ main(args : String[]) : void
- bridge : Br
K, V

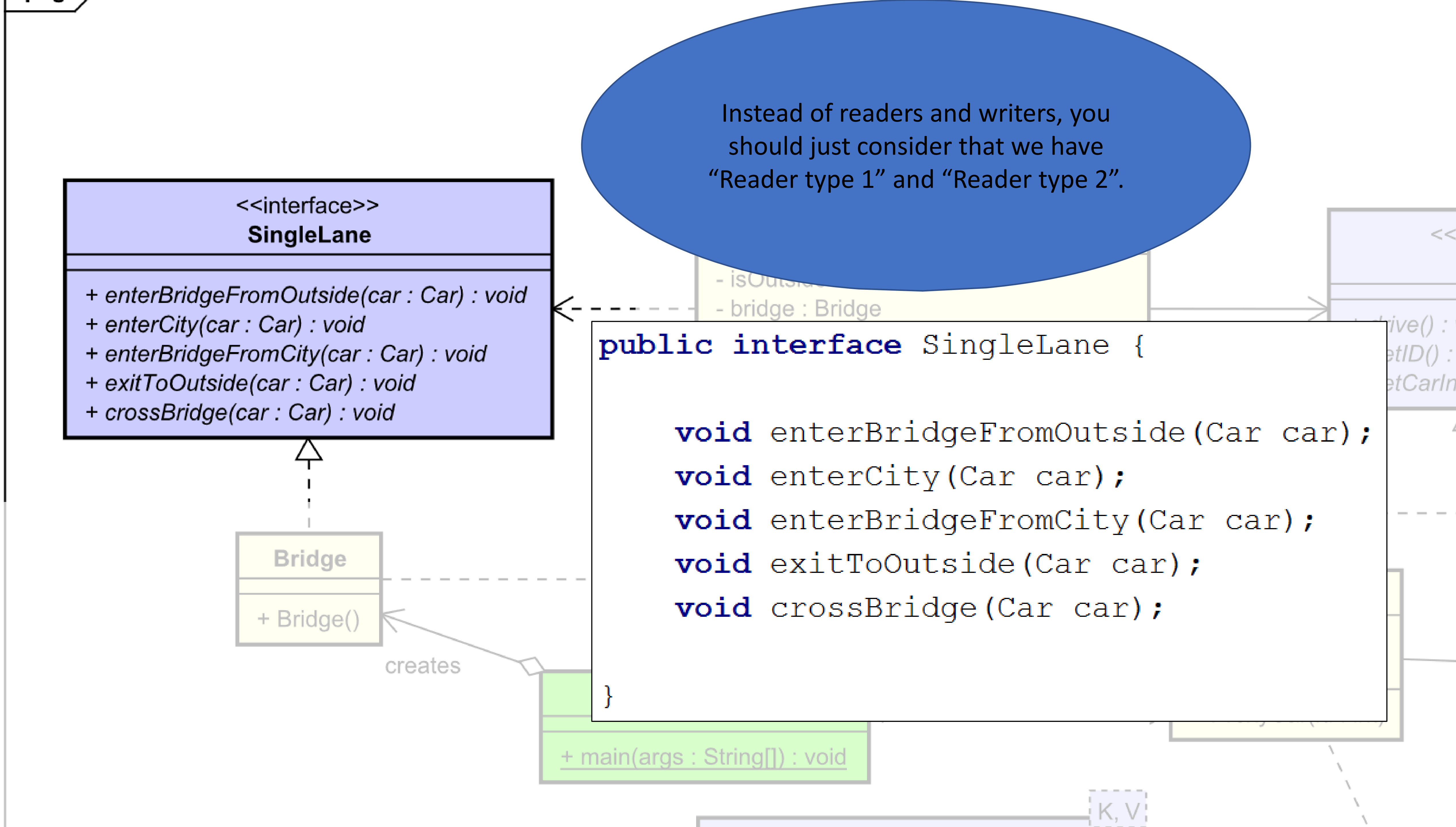
```
public interface SingleLane {

    void enterBridgeFromOutside(Car car);
    void enterCity(Car car);
    void enterBridgeFromCity(Car car);
    void exitToOutside(Car car);
    void crossBridge(Car car);

}
```

enterBridgeFromCity and exitToOutside is also acquire and release, just from the other side



<<interface>>
SingleLane
+ enterBridgeFromOutside(car : Car) : void
+ enterCity(car : Car) : void
+ enterBridgeFromCity(car : Car) : void
+ exitToOutside(car : Car) : void
+ crossBridge(car : Car) : void
Bridge
+ Bridge()
creates
+ main(args : String[]) : void
- bridge : Bri
K, V

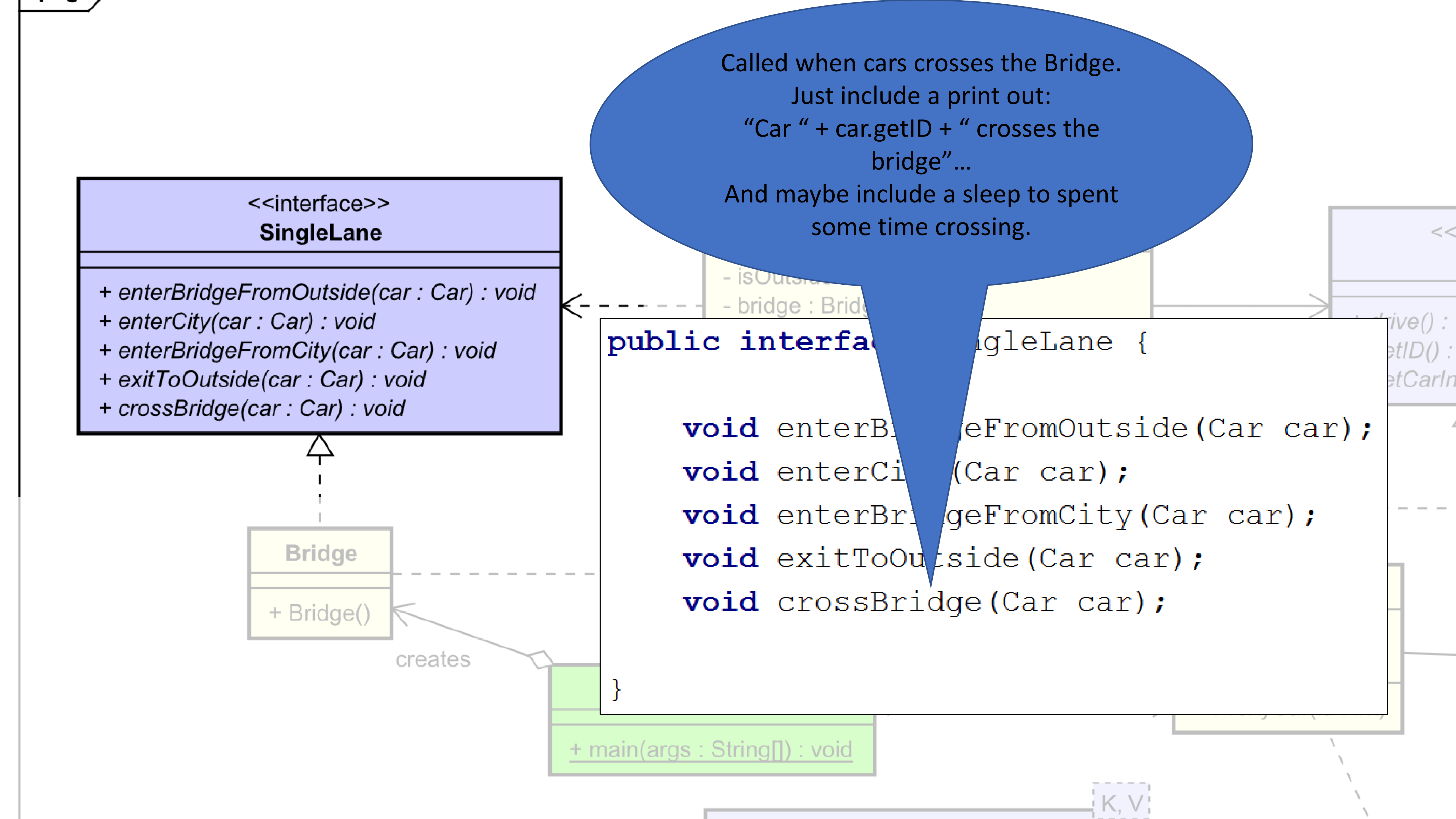
```
public interfa      ingleLane {

    void enterB  dgeFromOutside(Car car);
    void enterC  ty(Car car);
    void enterBridgeFromCity(Car car);
    void exitToOutside(Car car);
    void crossBridge(Car car);

}
```


<<interface>>
SingleLane
+ enterBridgeFromOutside(car : Car) : void
+ enterCity(car : Car) : void
+ enterBridgeFromCity(car : Car) : void
+ exitToOutside(car : Car) : void
+ crossBridge(car : Car) : void
- bridge : Bridge
Bridge
+ Bridge()
creates
+ main(args : String[]) : void
K, V

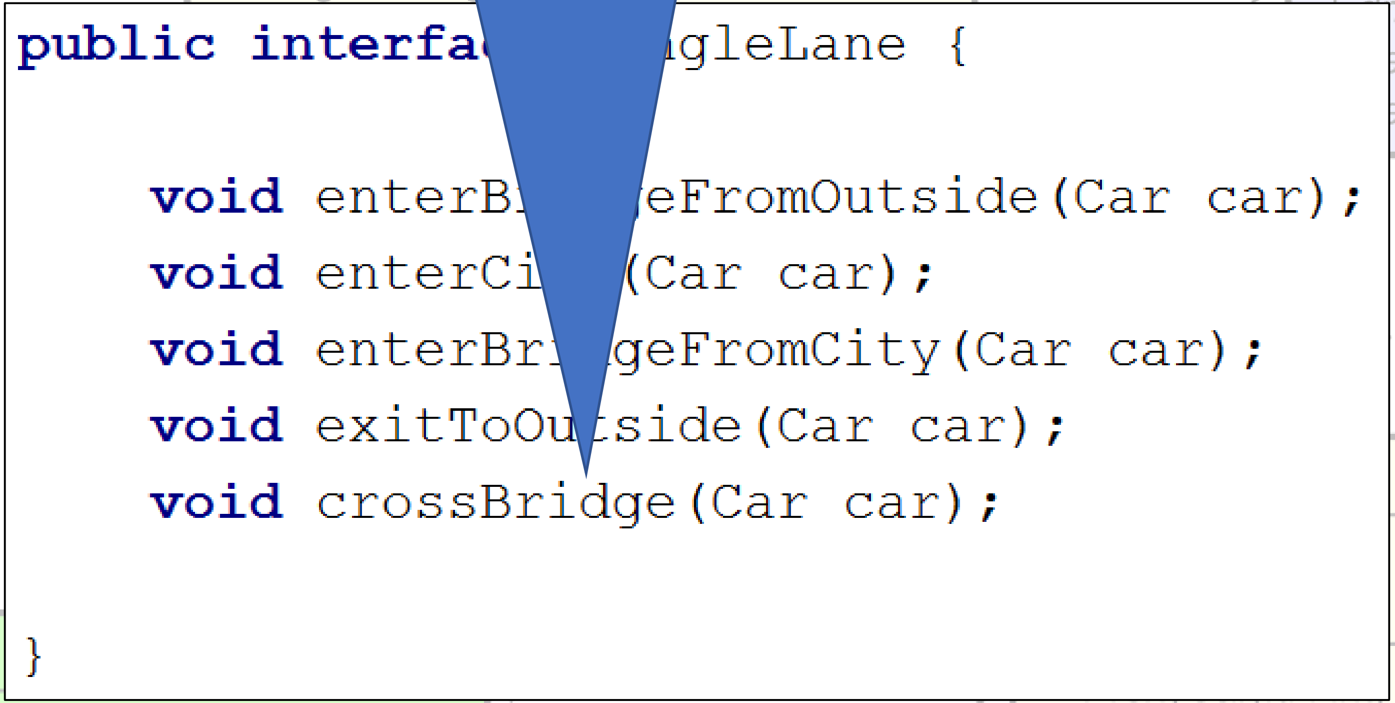
Instead of readers and writers, you should just consider that we have “Reader type 1” and “Reader type 2”.

```java
public interface SingleLane {

    void enterBridgeFromOutside(Car car);
    void enterCity(Car car);
    void enterBridgeFromCity(Car car);
    void exitToOutside(Car car);
    void crossBridge(Car car);

}
```

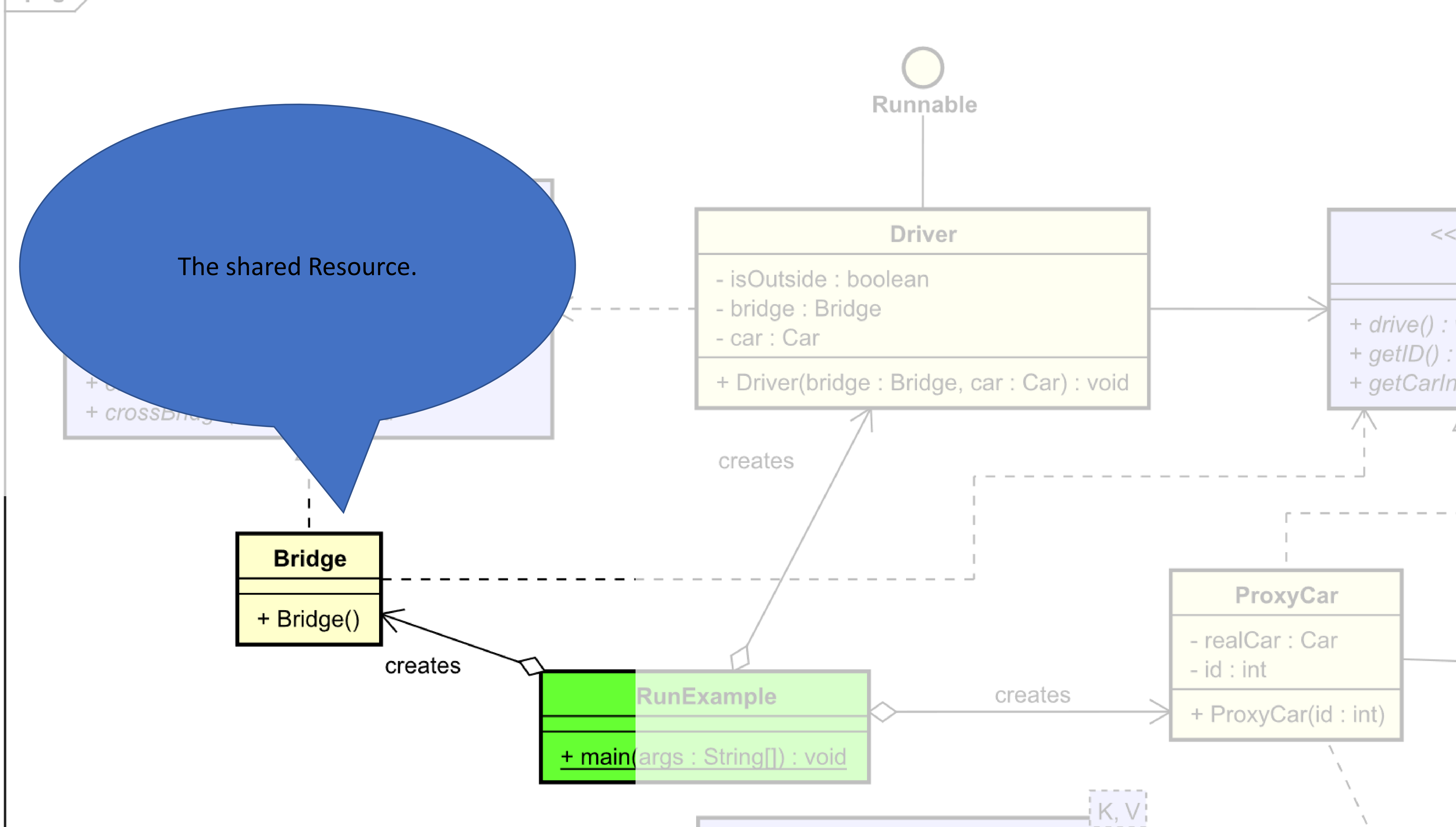
Called when cars crosses the Bridge.
Just include a print out:
“Car “ + car.getID + “ crosses the bridge”…
And maybe include a sleep to spent some time crossing.

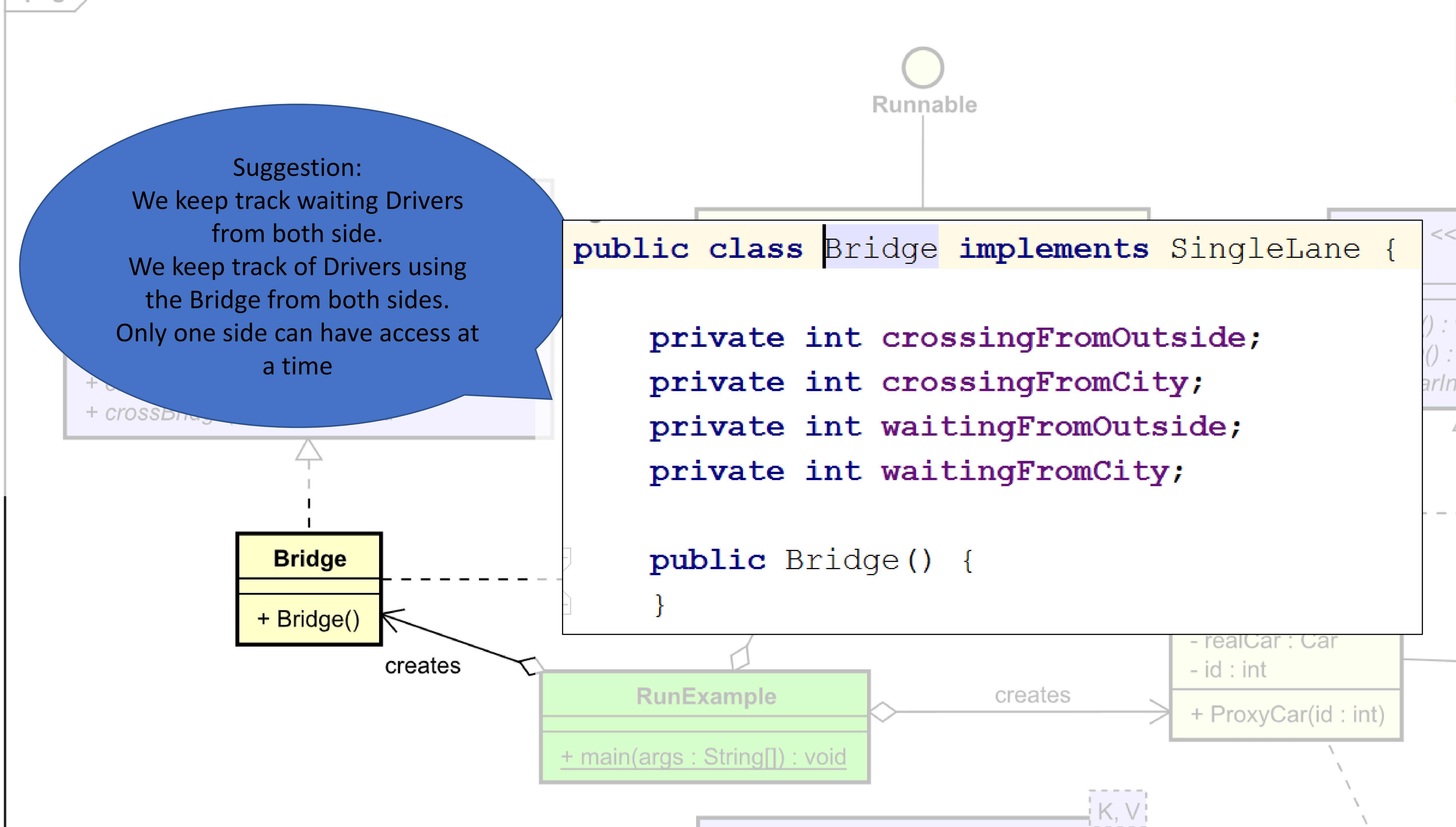


<<interface>>
SingleLane
+ enterBridgeFromOutside(car : Car) : void
+ enterCity(car : Car) : void
+ enterBridgeFromCity(car : Car) : void
+ exitToOutside(car : Car) : void
+ crossBridge(car : Car) : void
Bridge
+ Bridge()
creates
- bridge : Brid
+ main(args : String[]) : void
K, V



```
public interfa      ngleLane {

    void enterB    eFromOutside(Car car);
    void enterCi   (Car car);
    void enterBr   geFromCity(Car car);
    void exitToOu  side(Car car);
    void crossBridge(Car car);

}
```

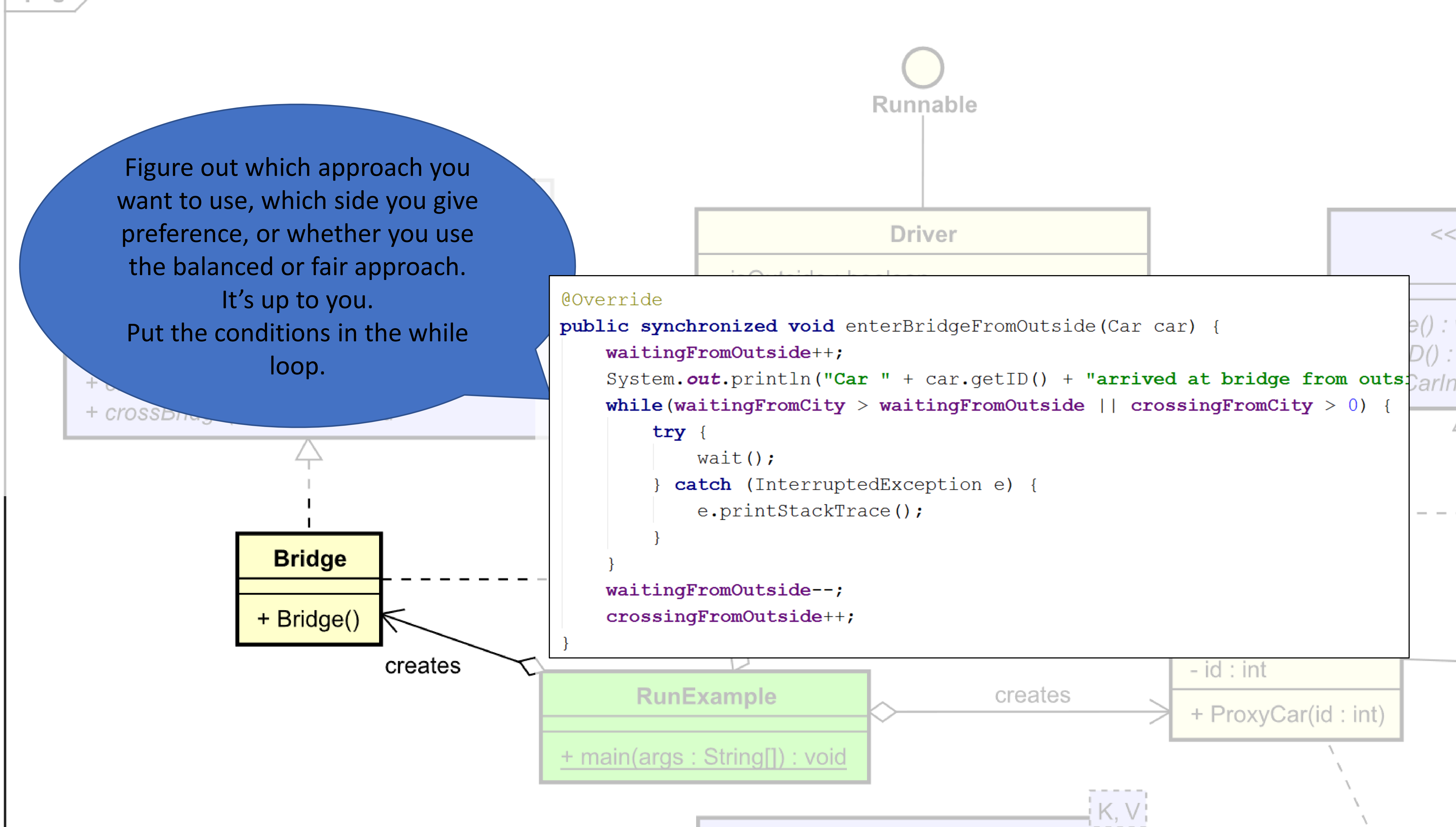

The shared Resource.
Runnable
Driver
- isOutside : boolean
- bridge : Bridge
- car : Car
+ Driver(bridge : Bridge, car : Car) : void
+ drive() :
+ getID() :
+ getCarIn
+ crossBrid
creates
Bridge
+ Bridge()
creates
RunExample
+ main(args : String[]) : void
creates
ProxyCar
- realCar : Car
- id : int
+ ProxyCar(id : int)
K, V

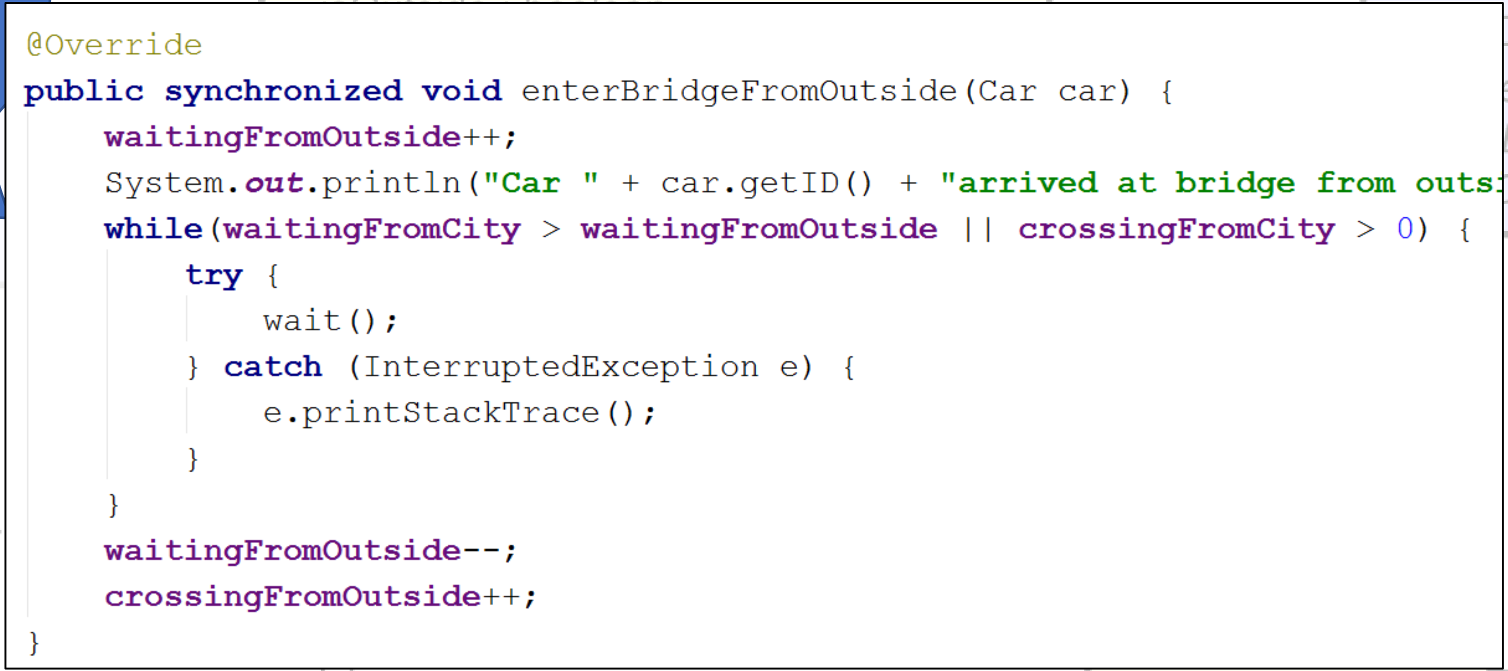


Suggestion:
We keep track waiting Drivers from both side.
We keep track of Drivers using the Bridge from both sides.
Only one side can have access at a time

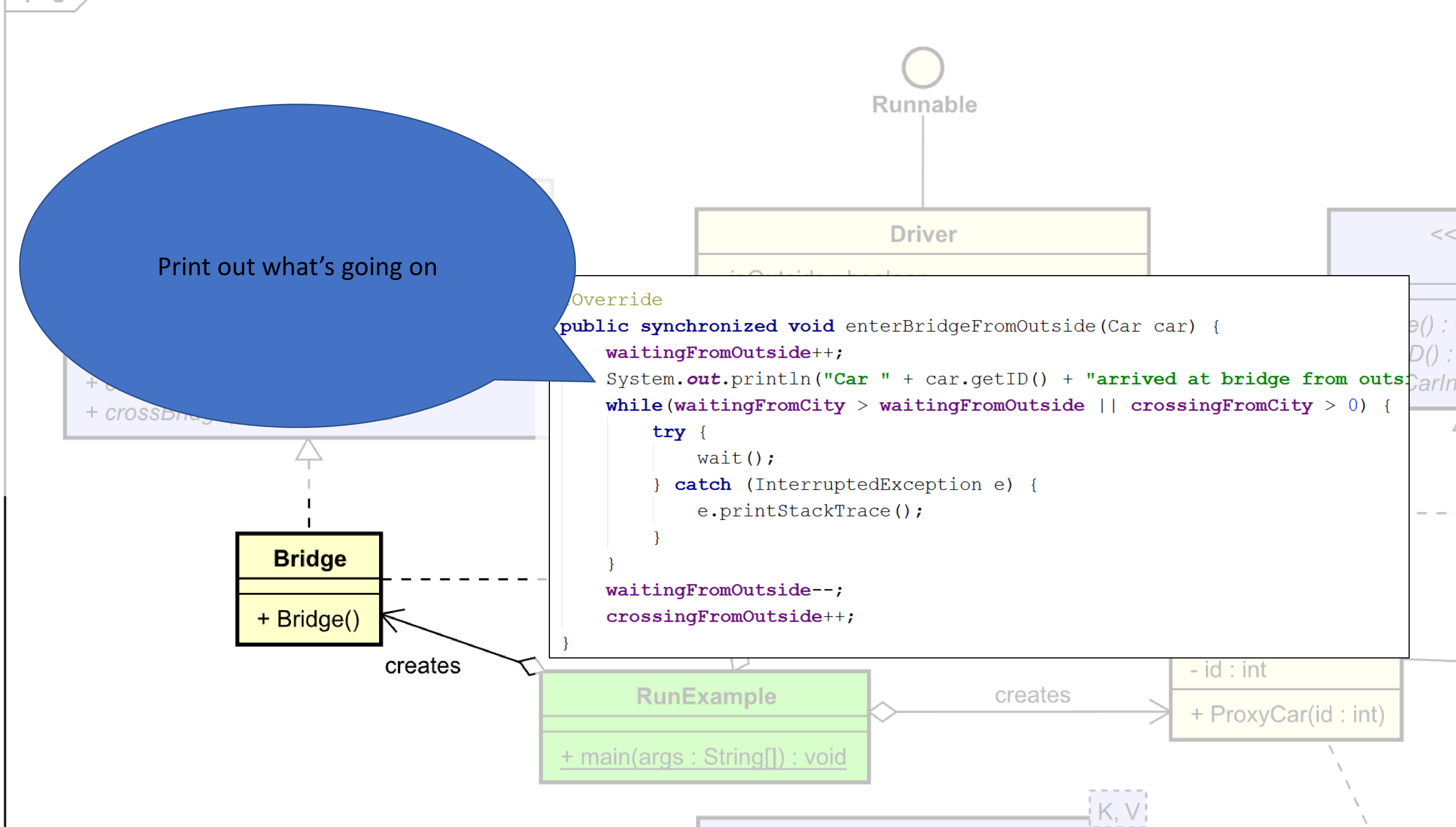
```
public class Bridge implements SingleLane {

    private int crossingFromOutside;
    private int crossingFromCity;
    private int waitingFromOutside;
    private int waitingFromCity;

    public Bridge() {
    }
```

Figure out which approach you want to use, which side you give preference, or whether you use the balanced or fair approach.
It’s up to you.
Put the conditions in the while loop.



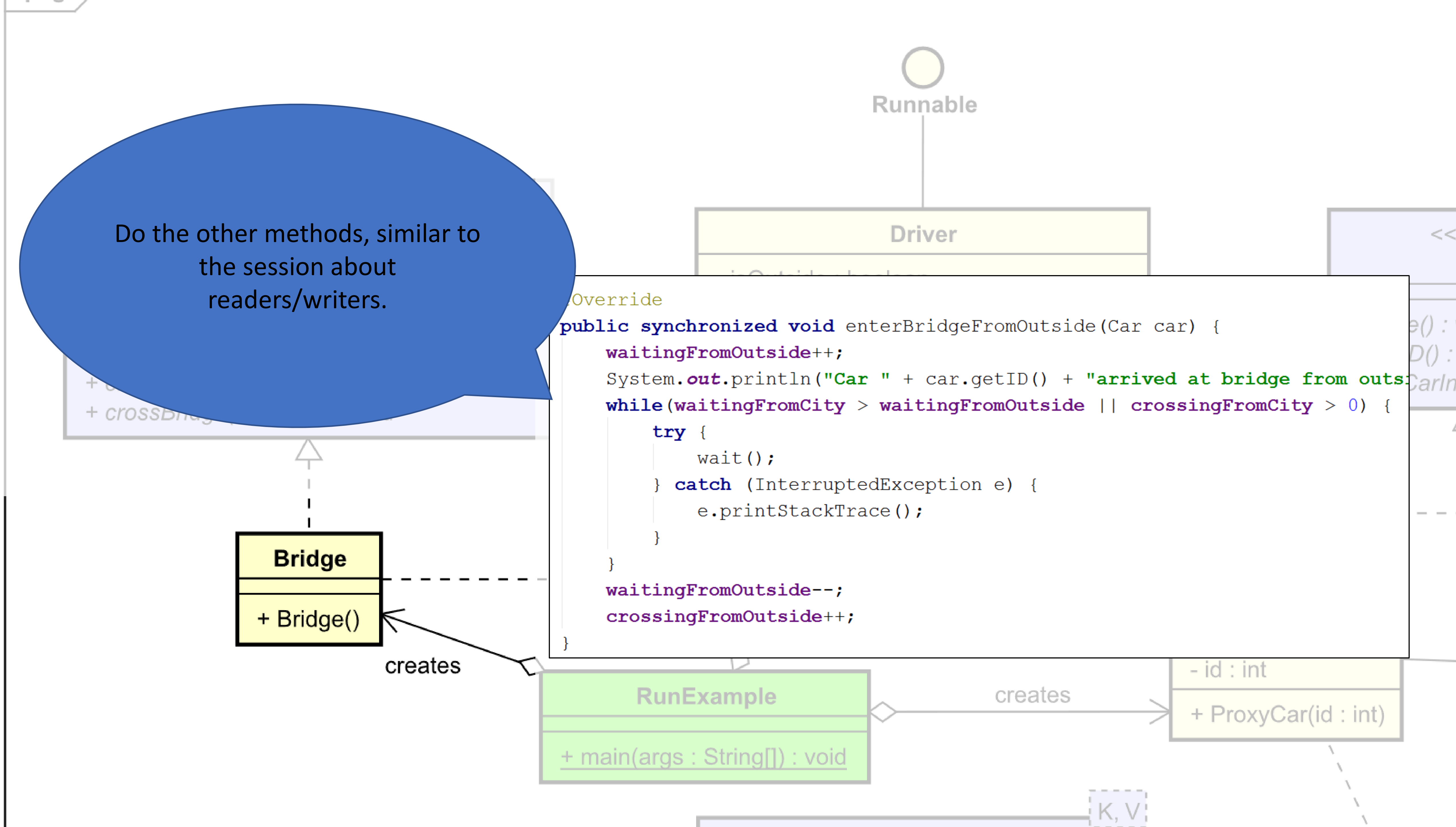


```
@Override
public synchronized void enterBridgeFromOutside(Car car) {
    waitingFromOutside++;
    System.out.println("Car " + car.getID() + "arrived at bridge from outs
    while(waitingFromCity > waitingFromOutside || crossingFromCity > 0) {
        try {
            wait();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
    waitingFromOutside--;
    crossingFromOutside++;
}
```



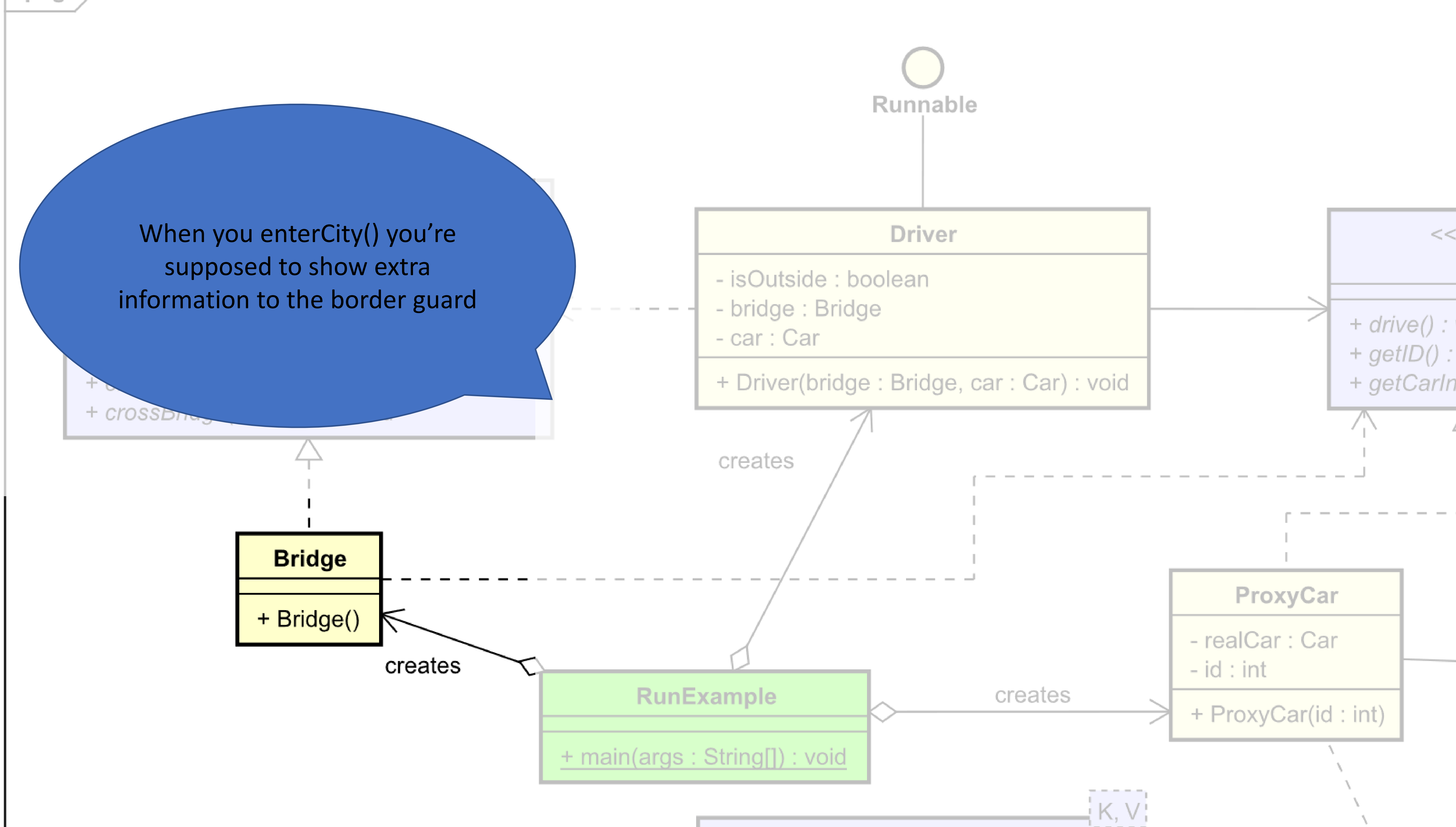
Print out what's going on

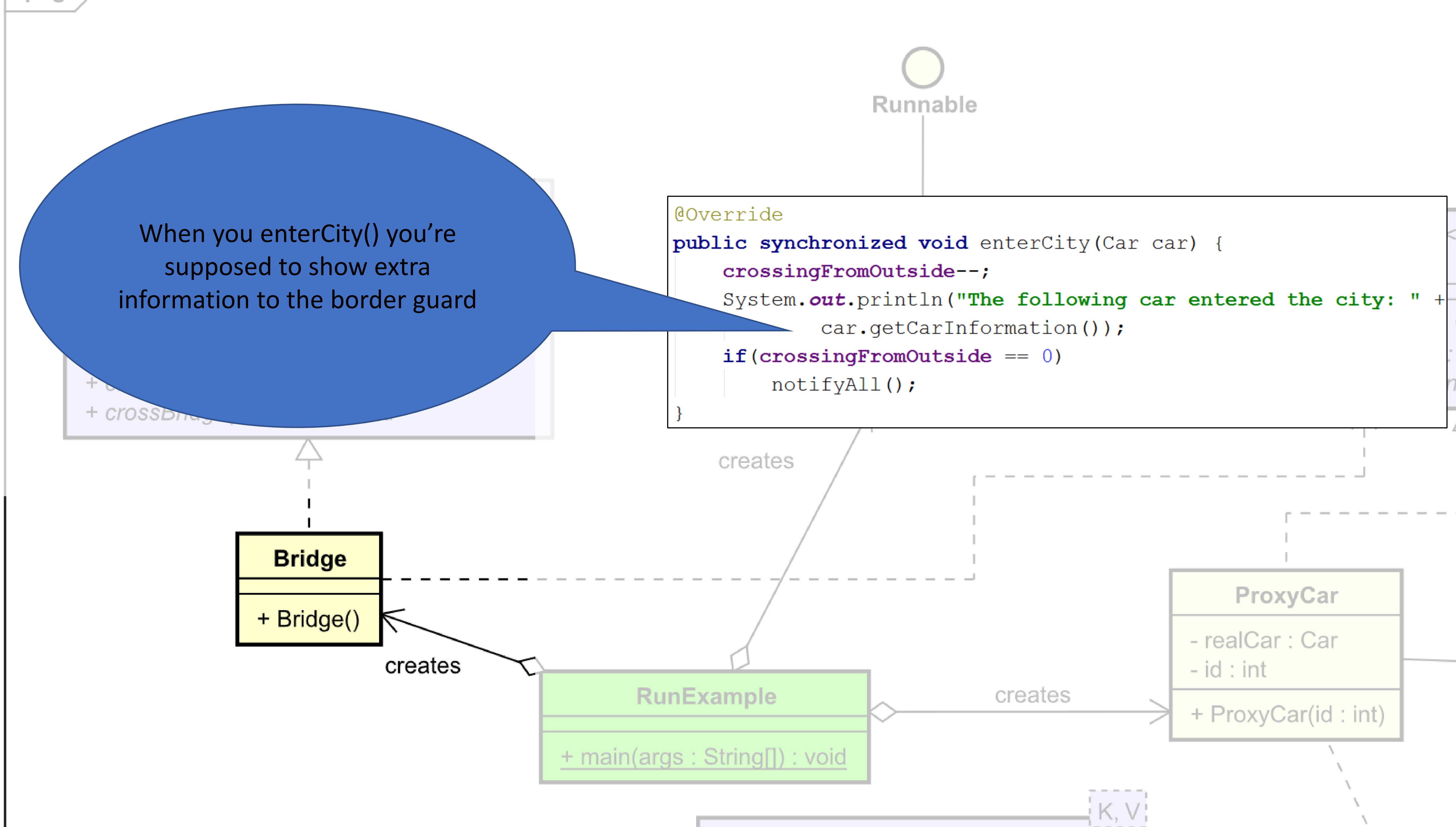
```
@Override
public synchronized void enterBridgeFromOutside(Car car) {
    waitingFromOutside++;
    System.out.println("Car " + car.getID() + "arrived at bridge from outs
    while(waitingFromCity > waitingFromOutside || crossingFromCity > 0) {
        try {
            wait();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
    waitingFromOutside--;
    crossingFromOutside++;
}
```



Do the other methods, similar to the session about readers/writers.

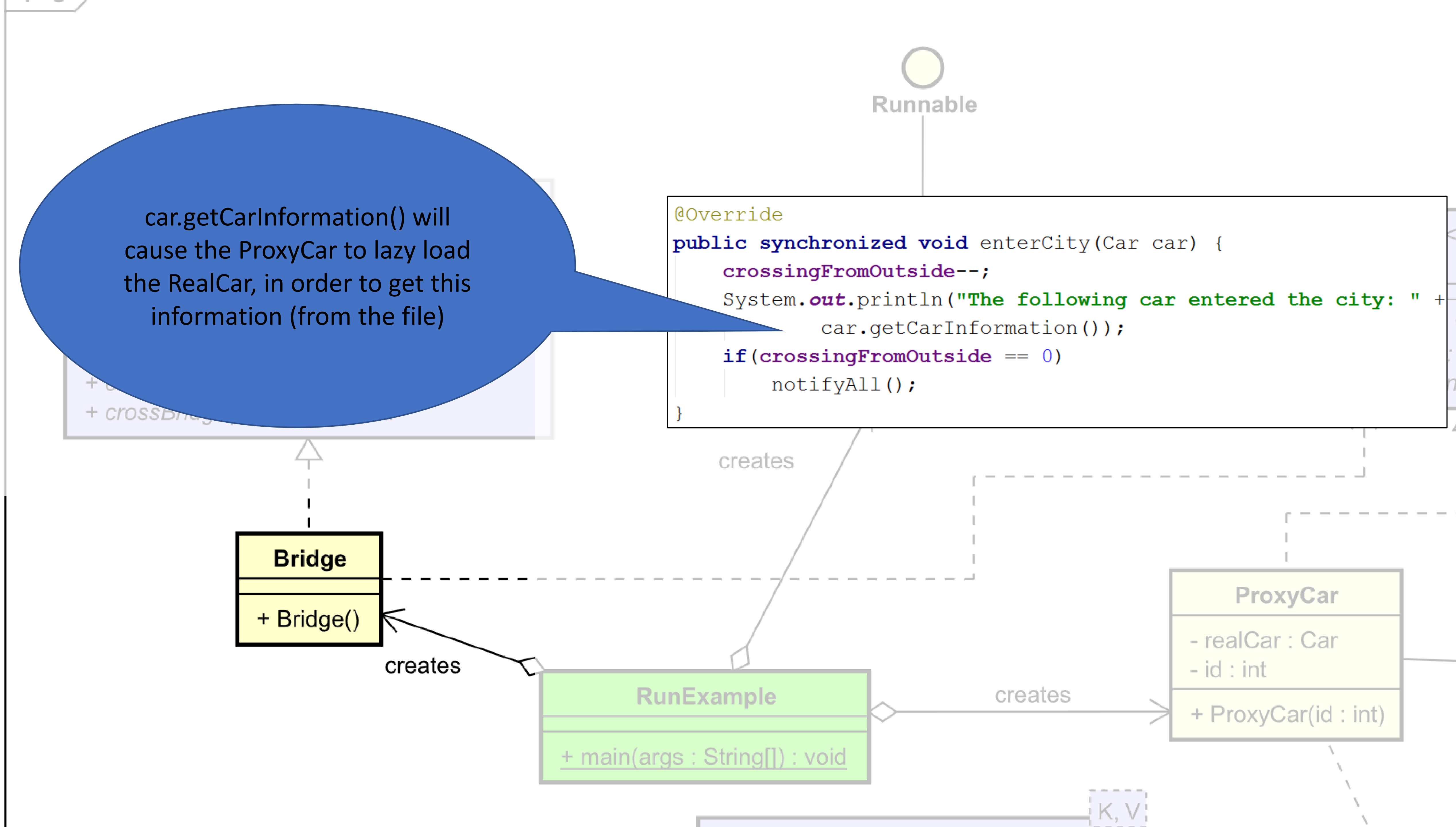
```
@Override
public synchronized void enterBridgeFromOutside(Car car) {
    waitingFromOutside++;
    System.out.println("Car " + car.getID() + "arrived at bridge from outsi
    while(waitingFromCity > waitingFromOutside || crossingFromCity > 0) {
        try {
            wait();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
    waitingFromOutside--;
    crossingFromOutside++;
}
```


When you enterCity() you're supposed to show extra information to the border guard
Runnable
Driver
- isOutside : boolean
- bridge : Bridge
- car : Car
+ Driver(bridge : Bridge, car : Car) : void
+ crossBridg
Bridge
+ Bridge()
creates
creates
RunExample
+ main(args : String[]) : void
creates
ProxyCar
- realCar : Car
- id : int
+ ProxyCar(id : int)
+ drive() :
+ getID() :
+ getCarIn
K, V



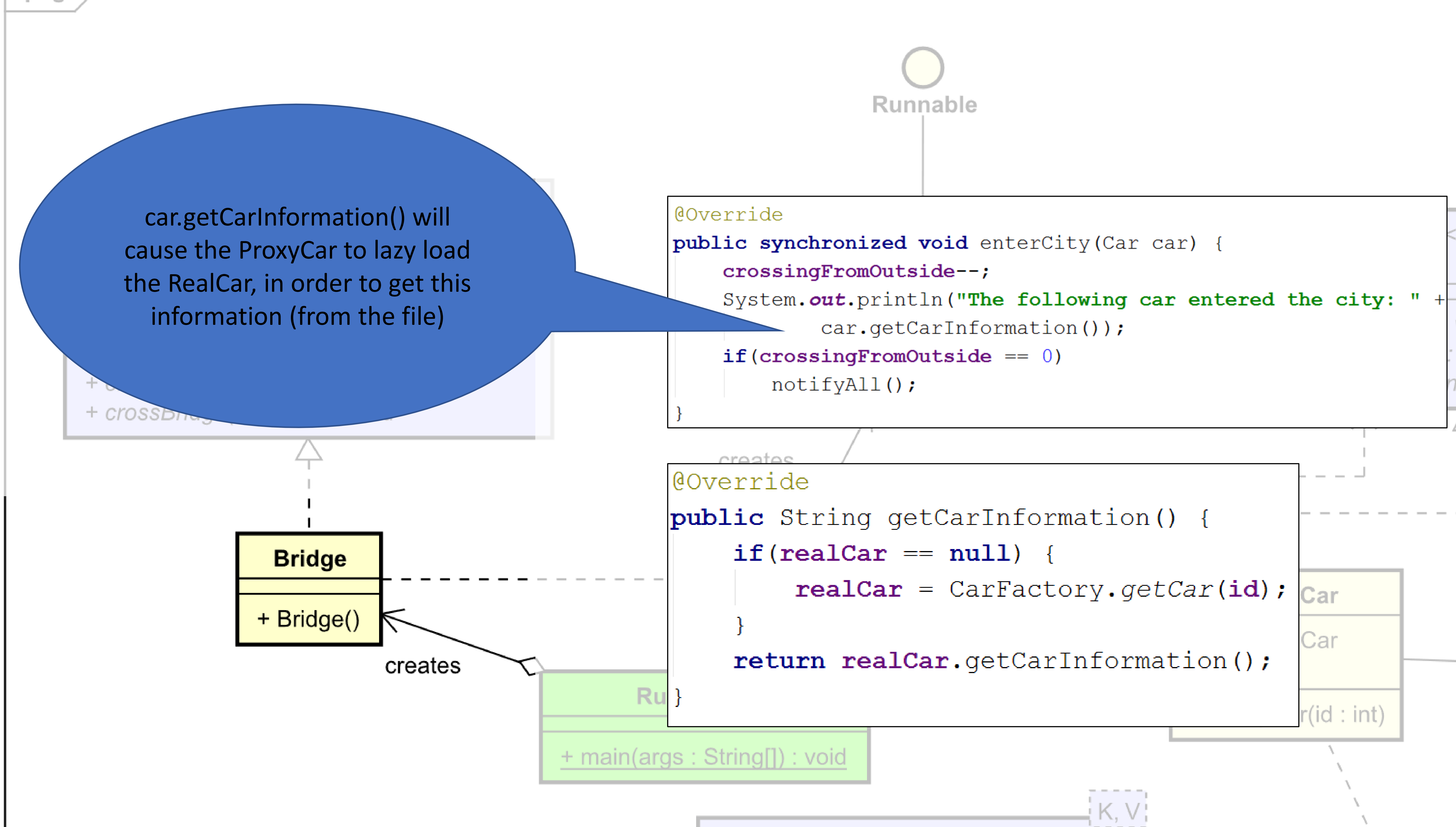
When you enterCity() you're supposed to show extra information to the border guard

```
@Override
public synchronized void enterCity(Car car) {
    crossingFromOutside--;
    System.out.println("The following car entered the city: " +
            car.getCarInformation());
    if(crossingFromOutside == 0)
        notifyAll();
}
```



car.getCarInformation() will cause the ProxyCar to lazy load the RealCar, in order to get this information (from the file)

```
@Override
public synchronized void enterCity(Car car) {
    crossingFromOutside--;
    System.out.println("The following car entered the city: " +
            car.getCarInformation());
    if(crossingFromOutside == 0)
        notifyAll();
}
```

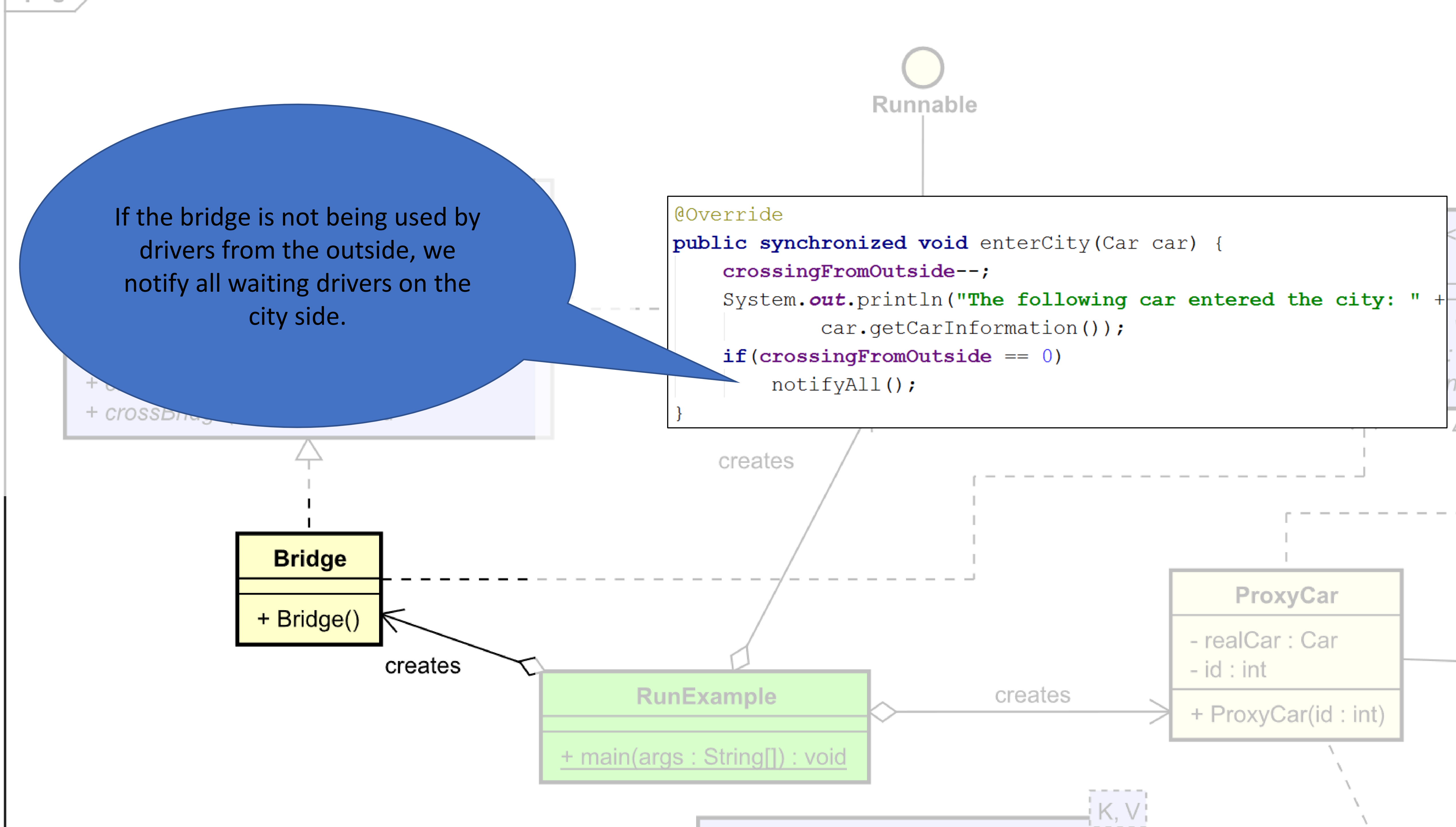


car.getCarInformation() will cause the ProxyCar to lazy load the RealCar, in order to get this information (from the file)

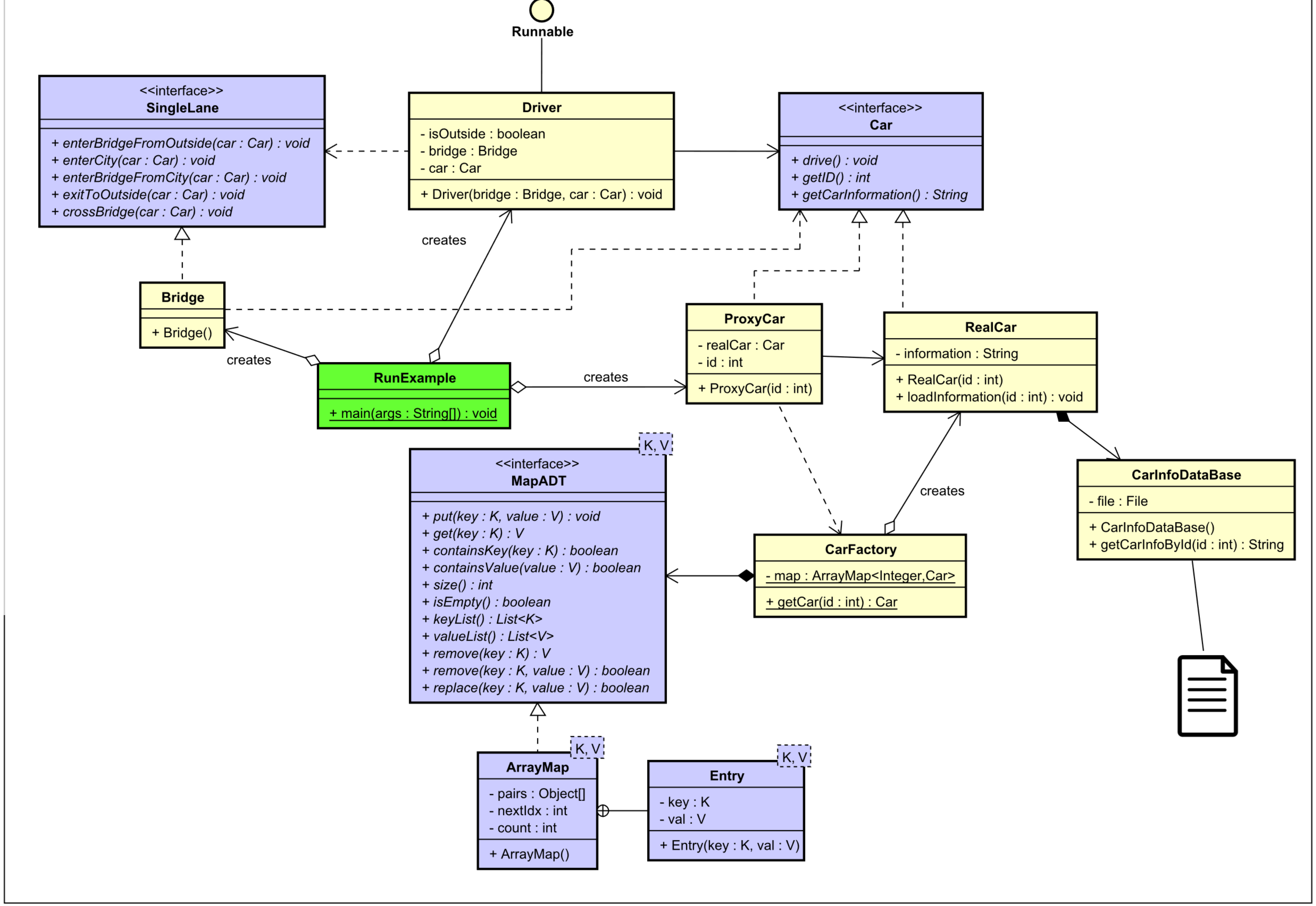
```
@Override
public synchronized void enterCity(Car car) {
    crossingFromOutside--;
    System.out.println("The following car entered the city: " +
            car.getCarInformation());
    if(crossingFromOutside == 0)
        notifyAll();
}
```

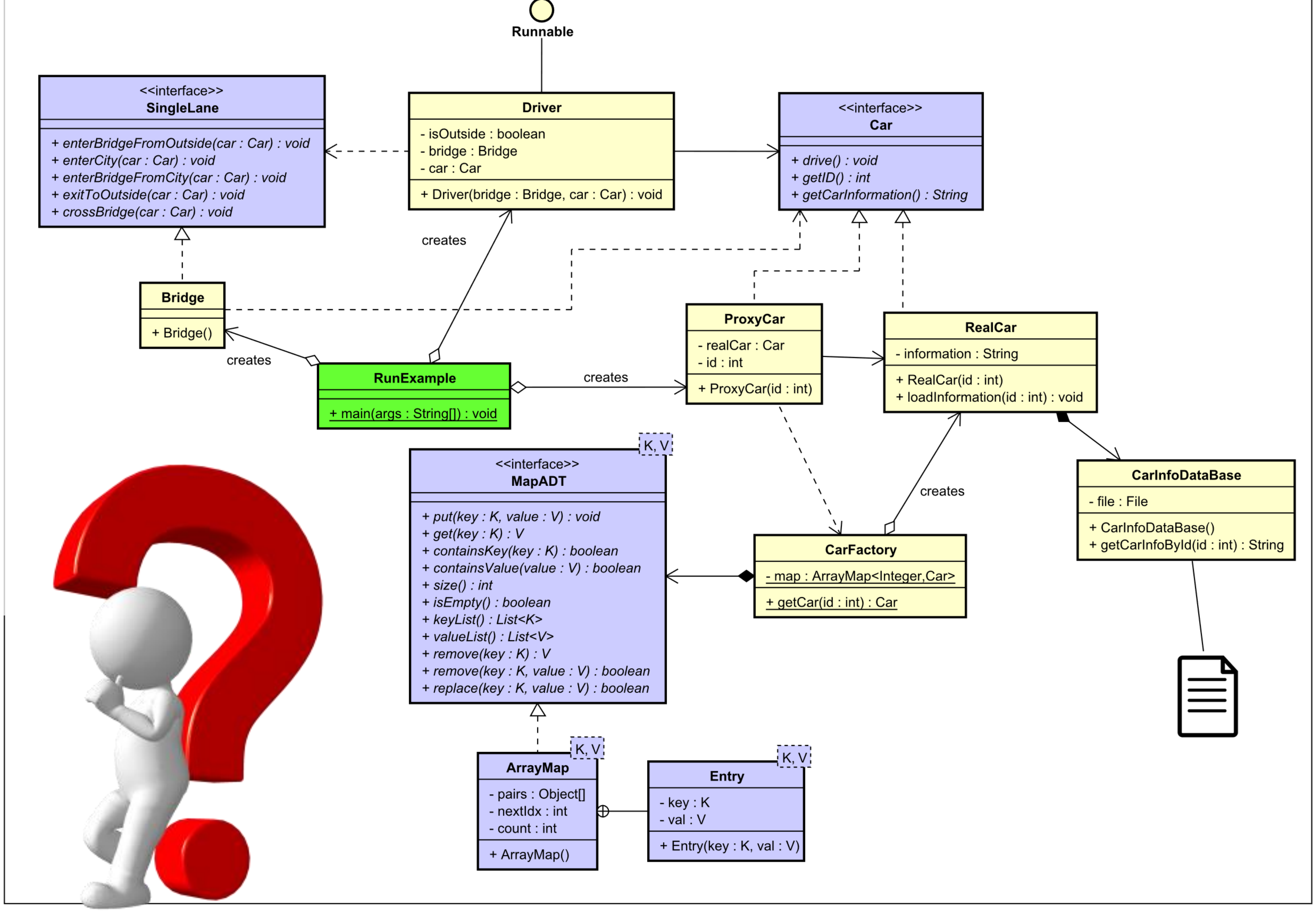
```
@Override
public String getCarInformation() {
    if(realCar == null) {
        realCar = CarFactory.getCar(id);
    }
    return realCar.getCarInformation();
}
```

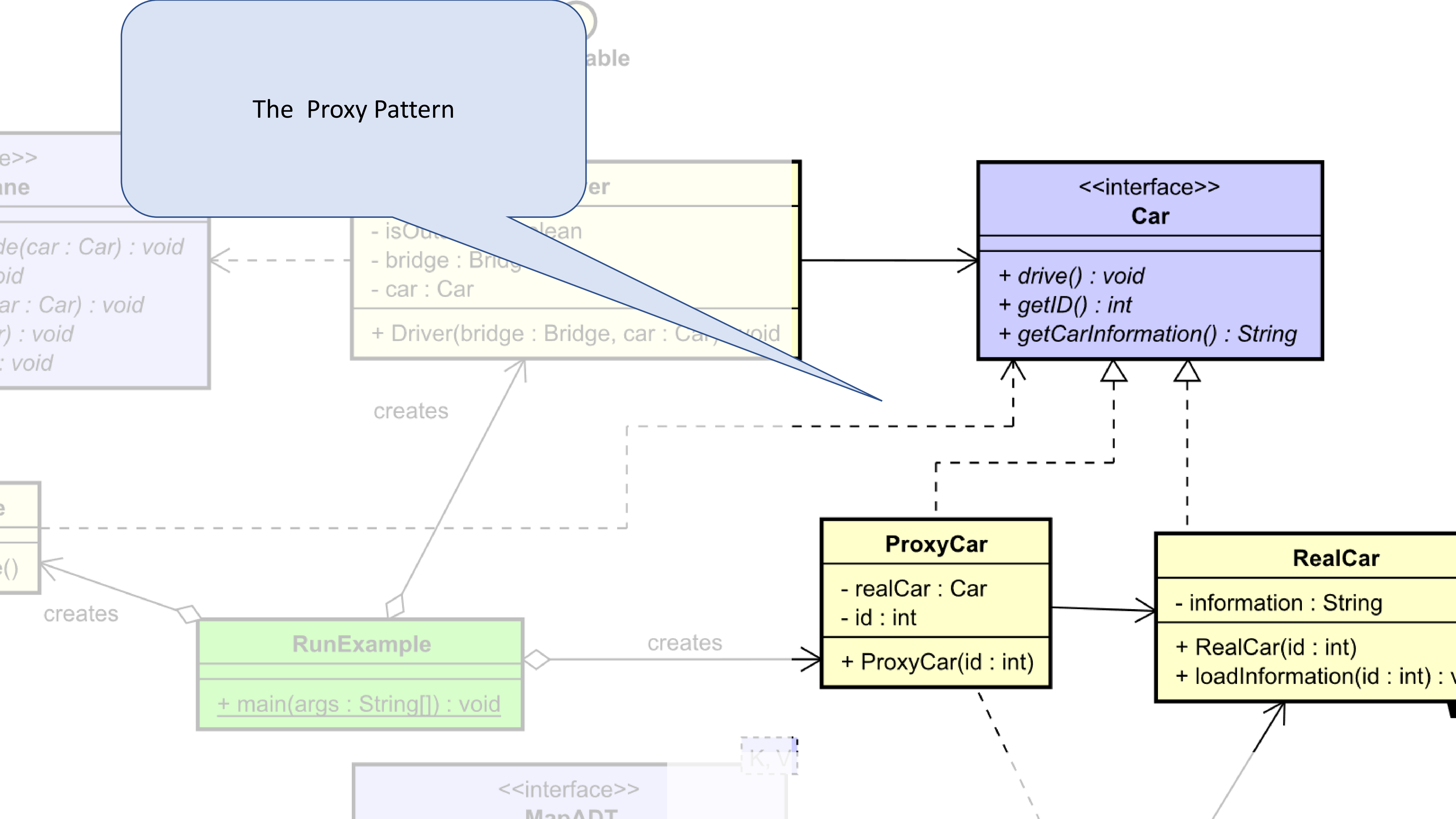
If the bridge is not being used by drivers from the outside, we notify all waiting drivers on the city side.



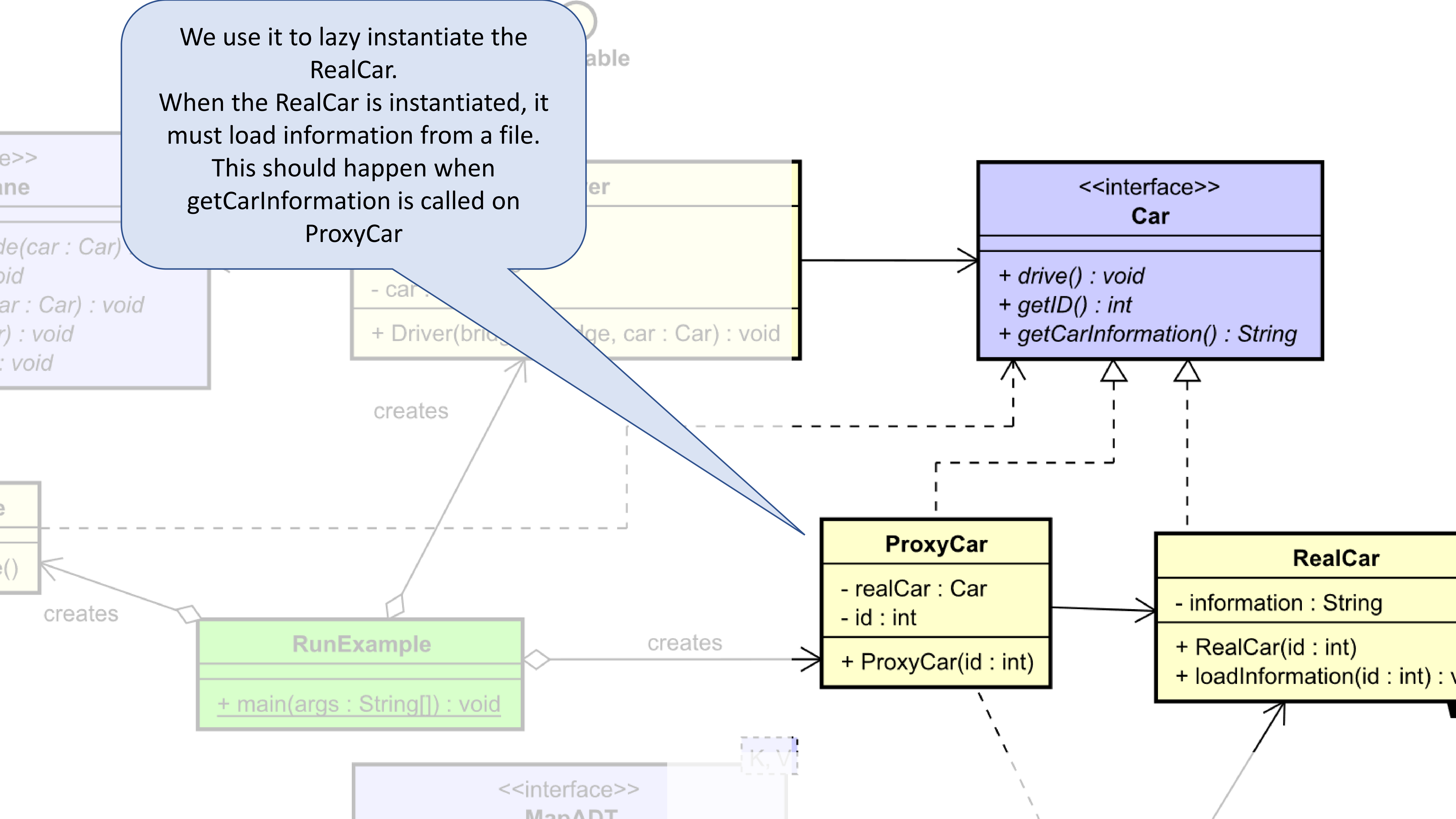
```
@Override
public synchronized void enterCity(Car car) {
    crossingFromOutside--;
    System.out.println("The following car entered the city: " +
            car.getCarInformation());
    if(crossingFromOutside == 0)
        notifyAll();
}
```


Runnable
<<interface>>
SingleLane
+ enterBridgeFromOutside(car : Car) : void
+ enterCity(car : Car) : void
+ enterBridgeFromCity(car : Car) : void
+ exitToOutside(car : Car) : void
+ crossBridge(car : Car) : void
Driver
- isOutside : boolean
- bridge : Bridge
- car : Car
+ Driver(bridge : Bridge, car : Car) : void
<<interface>>
Car
+ drive() : void
+ getID() : int
+ getCarInformation() : String
creates
Bridge
+ Bridge()
creates
RunExample
+ main(args : String[]) : void
creates
ProxyCar
- realCar : Car
- id : int
+ ProxyCar(id : int)
RealCar
- information : String
+ RealCar(id : int)
+ loadInformation(id : int) : void
K, V
<<interface>>
MapADT
+ put(key : K, value : V) : void
+ get(key : K) : V
+ containsKey(key : K) : boolean
+ containsValue(value : V) : boolean
+ size() : int
+ isEmpty() : boolean
+ keyList() : List<K>
+ valueList() : List<V>
+ remove(key : K) : V
+ remove(key : K, value : V) : boolean
+ replace(key : K, value : V) : boolean
creates
CarFactory
- map : ArrayMap<Integer,Car>
+ getCar(id : int) : Car
CarInfoDataBase
- file : File
+ CarInfoDataBase()
+ getCarInfoById(id : int) : String
K, V
ArrayMap
- pairs : Object[]
- nextIdx : int
- count : int
+ ArrayMap()
K, V
Entry
- key : K
- val : V
+ Entry(key : K, val : V)


Runnable
<<interface>>
SingleLane
+ enterBridgeFromOutside(car : Car) : void
+ enterCity(car : Car) : void
+ enterBridgeFromCity(car : Car) : void
+ exitToOutside(car : Car) : void
+ crossBridge(car : Car) : void
Driver
- isOutside : boolean
- bridge : Bridge
- car : Car
+ Driver(bridge : Bridge, car : Car) : void
<<interface>>
Car
+ drive() : void
+ getID() : int
+ getCarInformation() : String
creates
Bridge
+ Bridge()
creates
RunExample
+ main(args : String[]) : void
creates
ProxyCar
- realCar : Car
- id : int
+ ProxyCar(id : int)
RealCar
- information : String
+ RealCar(id : int)
+ loadInformation(id : int) : void
K, V
<<interface>>
MapADT
+ put(key : K, value : V) : void
+ get(key : K) : V
+ containsKey(key : K) : boolean
+ containsValue(value : V) : boolean
+ size() : int
+ isEmpty() : boolean
+ keyList() : List<K>
+ valueList() : List<V>
+ remove(key : K) : V
+ remove(key : K, value : V) : boolean
+ replace(key : K, value : V) : boolean
creates
CarFactory
- map : ArrayMap<Integer,Car>
+ getCar(id : int) : Car
CarInfoDataBase
- file : File
+ CarInfoDataBase()
+ getCarInfoById(id : int) : String
K, V
ArrayMap
- pairs : Object[]
- nextIdx : int
- count : int
+ ArrayMap()
K, V
Entry
- key : K
- val : V
+ Entry(key : K, val : V)

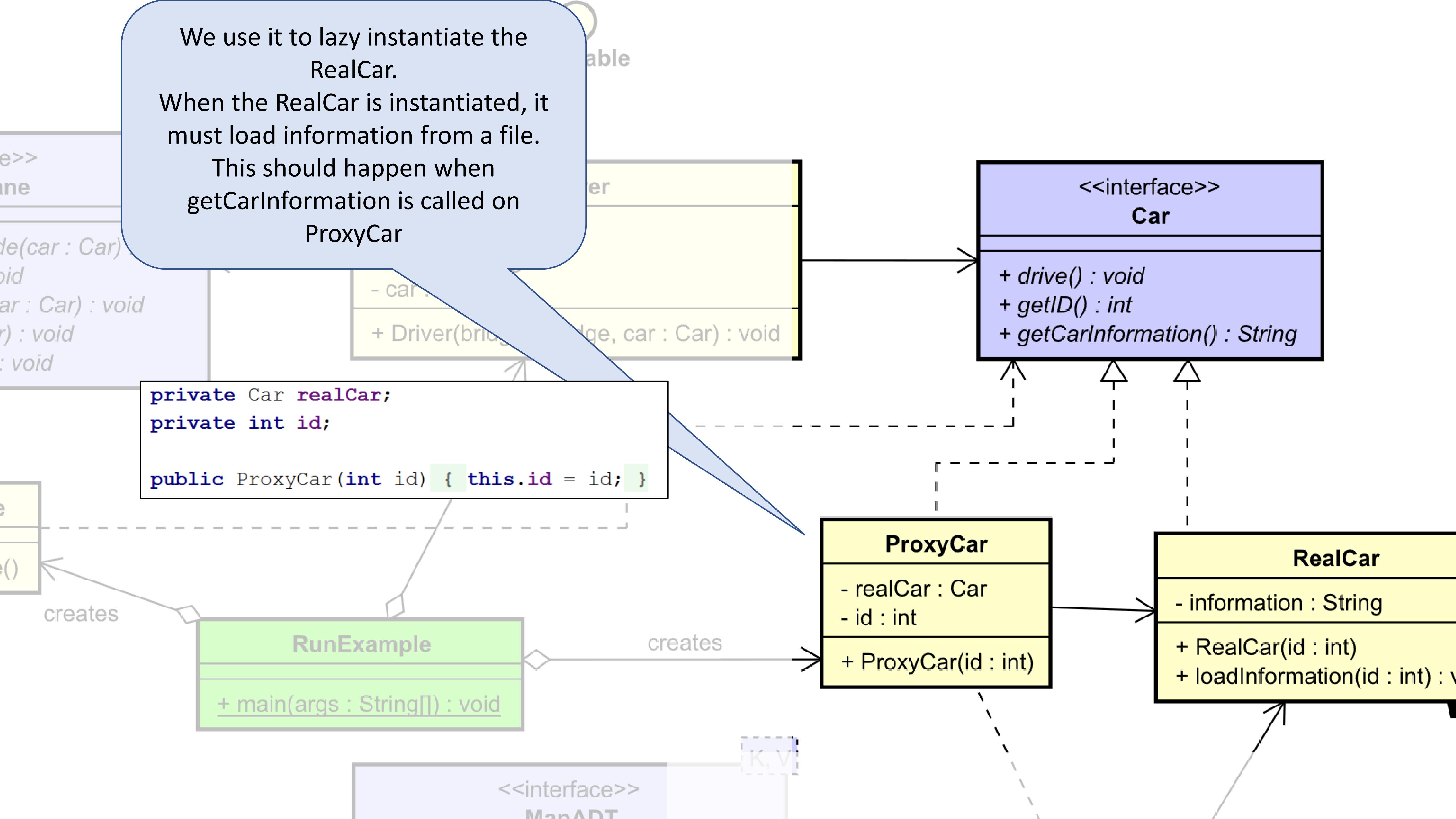

The Proxy Pattern
<<interface>>
Car
+ drive() : void
+ getID() : int
+ getCarInformation() : String
ProxyCar
- realCar : Car
- id : int
+ ProxyCar(id : int)
RealCar
- information : String
+ RealCar(id : int)
+ loadInformation(id : int) : v
- isOu lean
- bridge : Brid
- car : Car
+ Driver(bridge : Bridge, car : Car) void
creates
creates
creates
RunExample
+ main(args : String[]) : void
<<interface>>
MapADT
K, V
de(car : Car) : void
ar : Car) : void
r) : void
: void
able

We use it to lazy instantiate the RealCar.
When the RealCar is instantiated, it must load information from a file.
This should happen when getCarInformation is called on ProxyCar

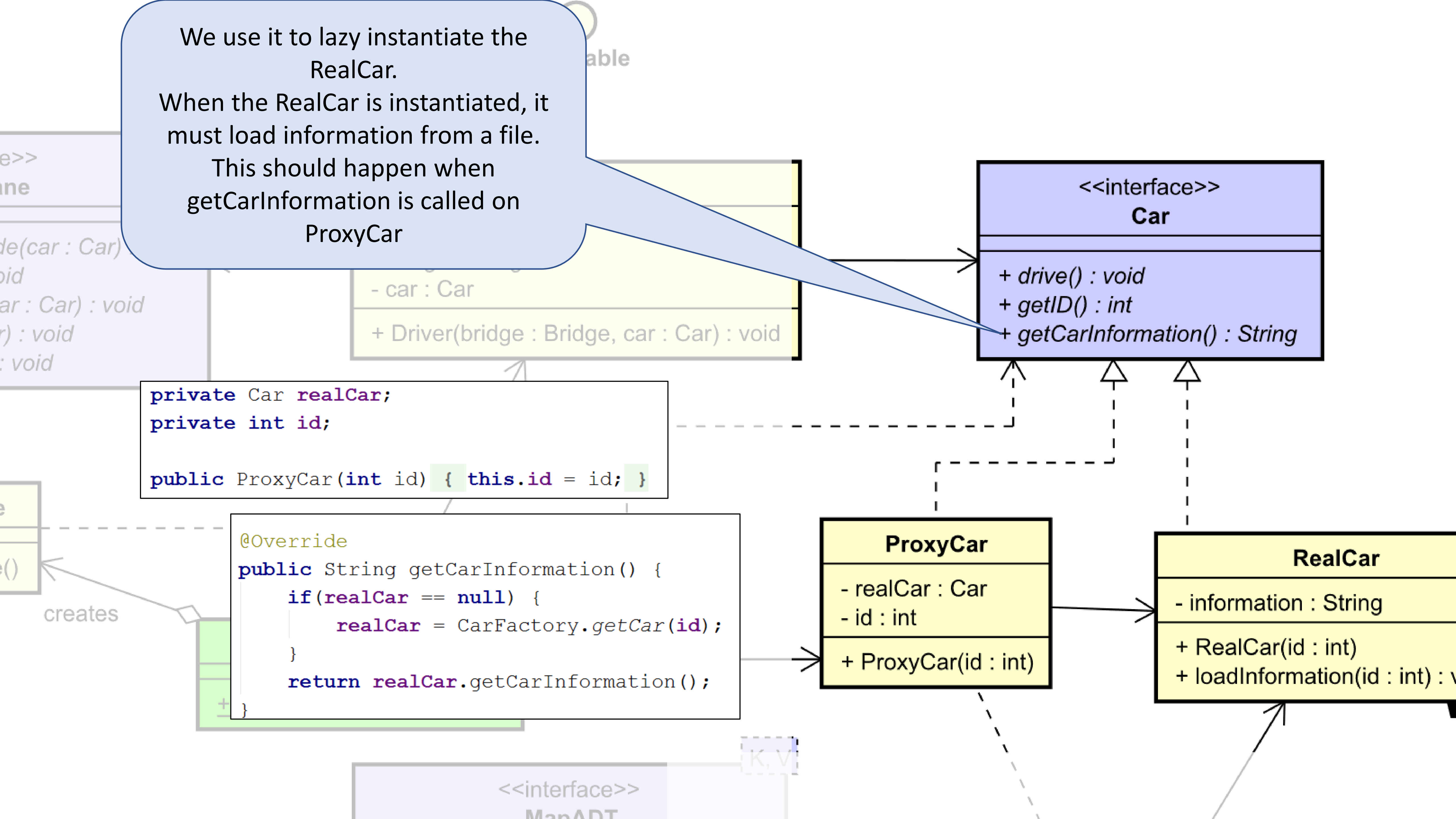


We use it to lazy instantiate the RealCar.
When the RealCar is instantiated, it must load information from a file.
This should happen when getCarInformation is called on ProxyCar

```
private Car realCar;
private int id;

public ProxyCar(int id) { this.id = id; }
```



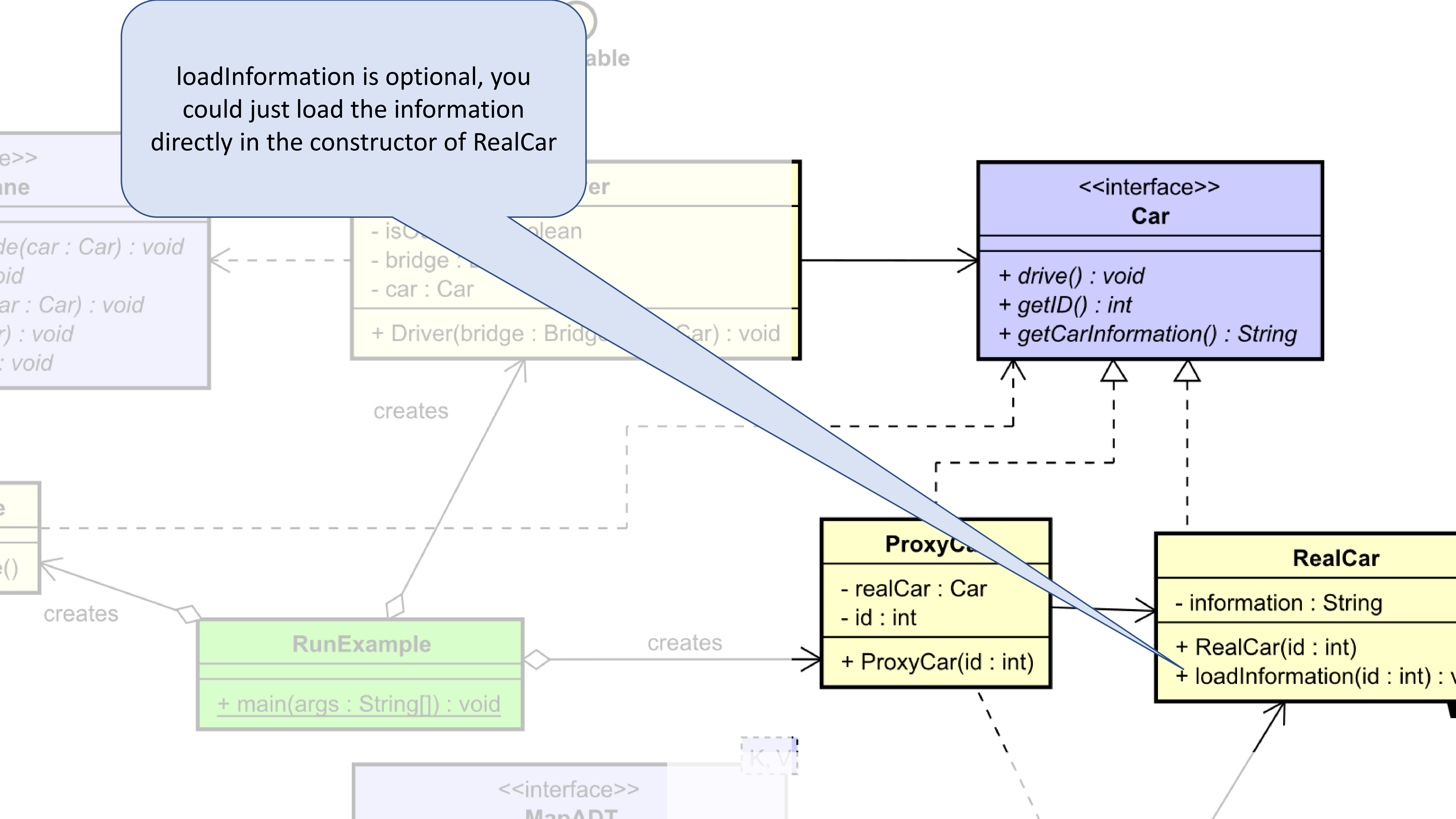
We use it to lazy instantiate the RealCar.
When the RealCar is instantiated, it must load information from a file.
This should happen when getCarInformation is called on ProxyCar



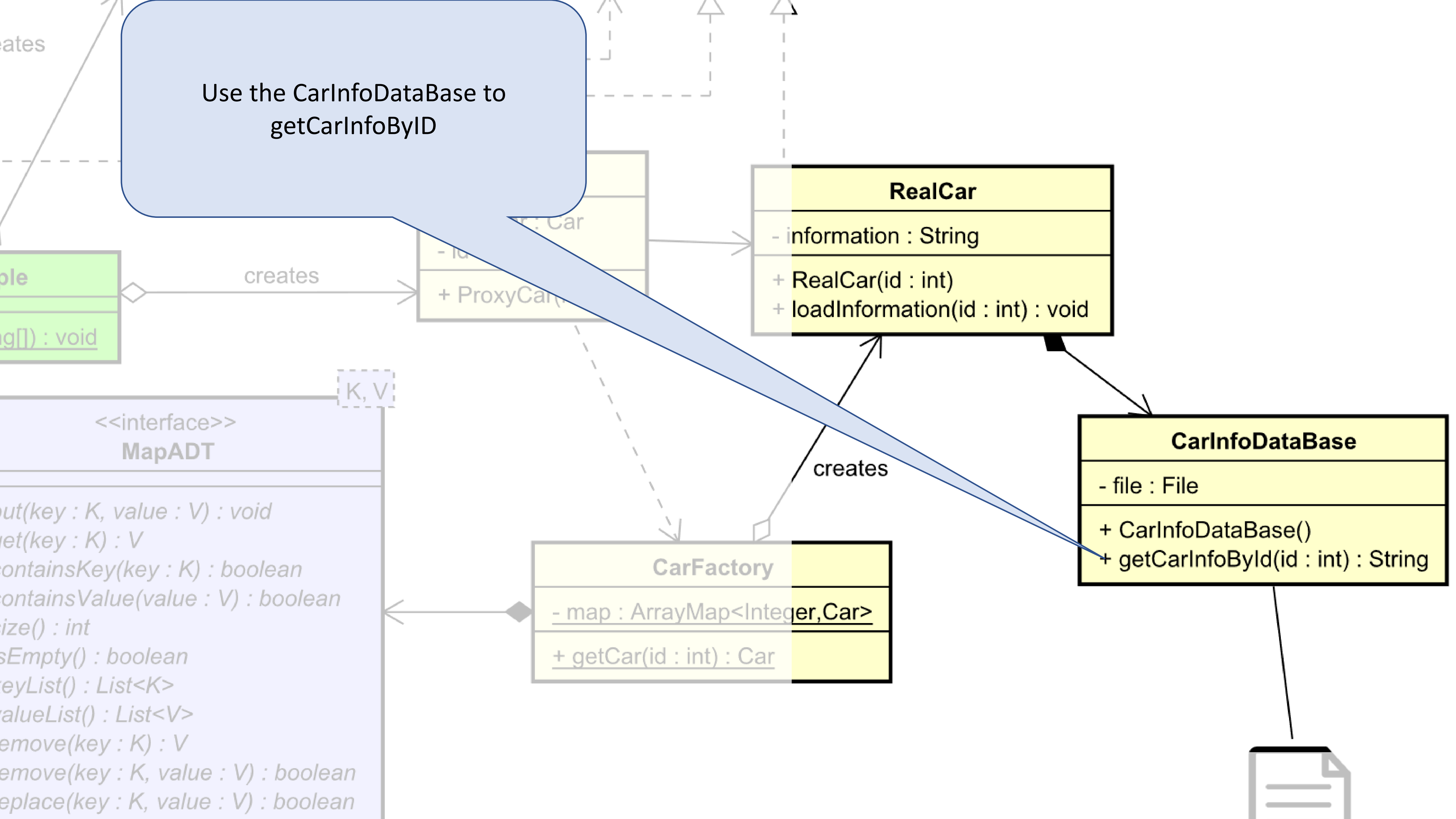
```java
private Car realCar;
private int id;

public ProxyCar(int id) { this.id = id; }
```

```java
@Override
public String getCarInformation() {
    if(realCar == null) {
        realCar = CarFactory.getCar(id);
    }
    return realCar.getCarInformation();
}
```

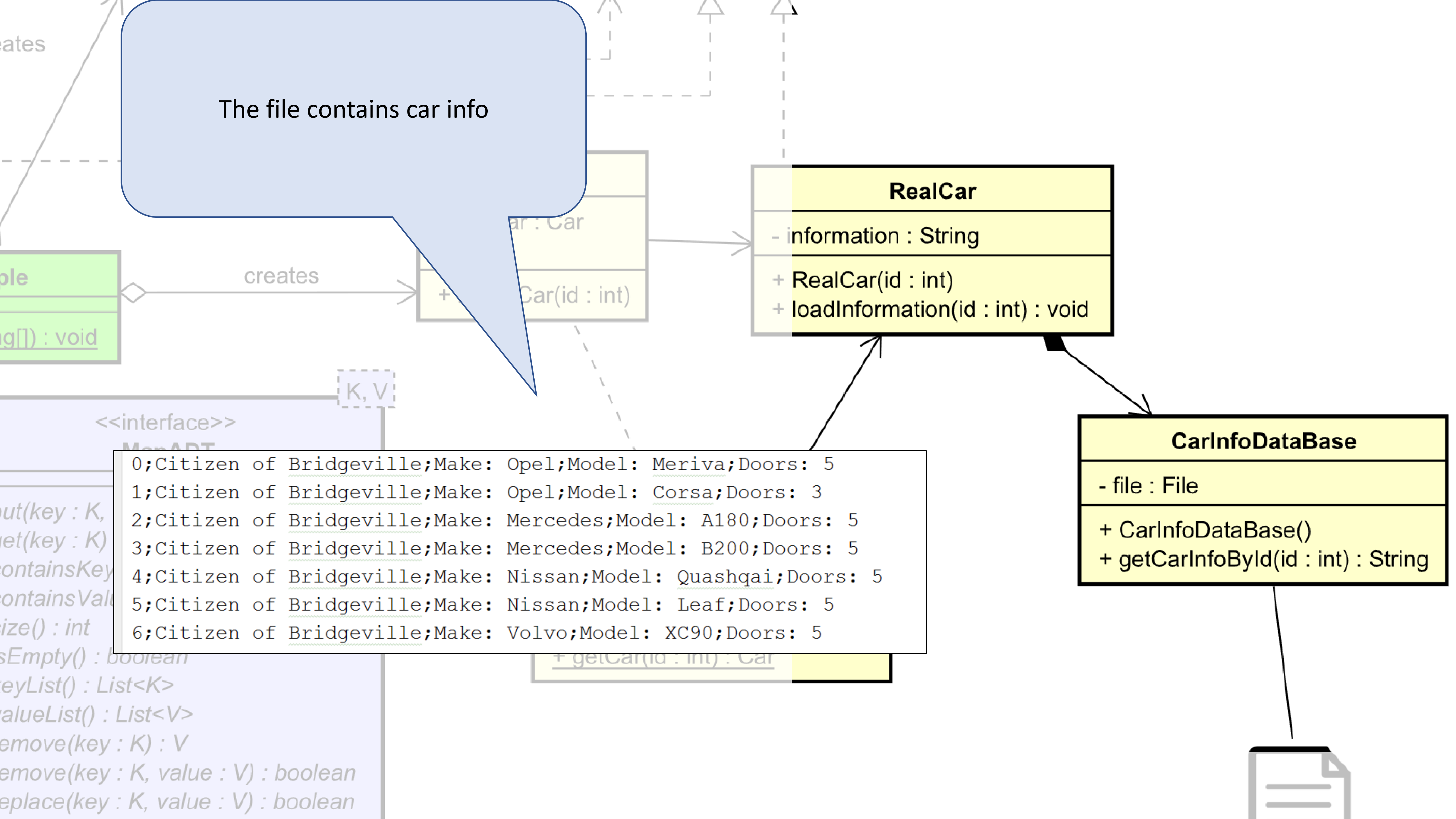
loadInformation is optional, you could just load the information directly in the constructor of RealCar



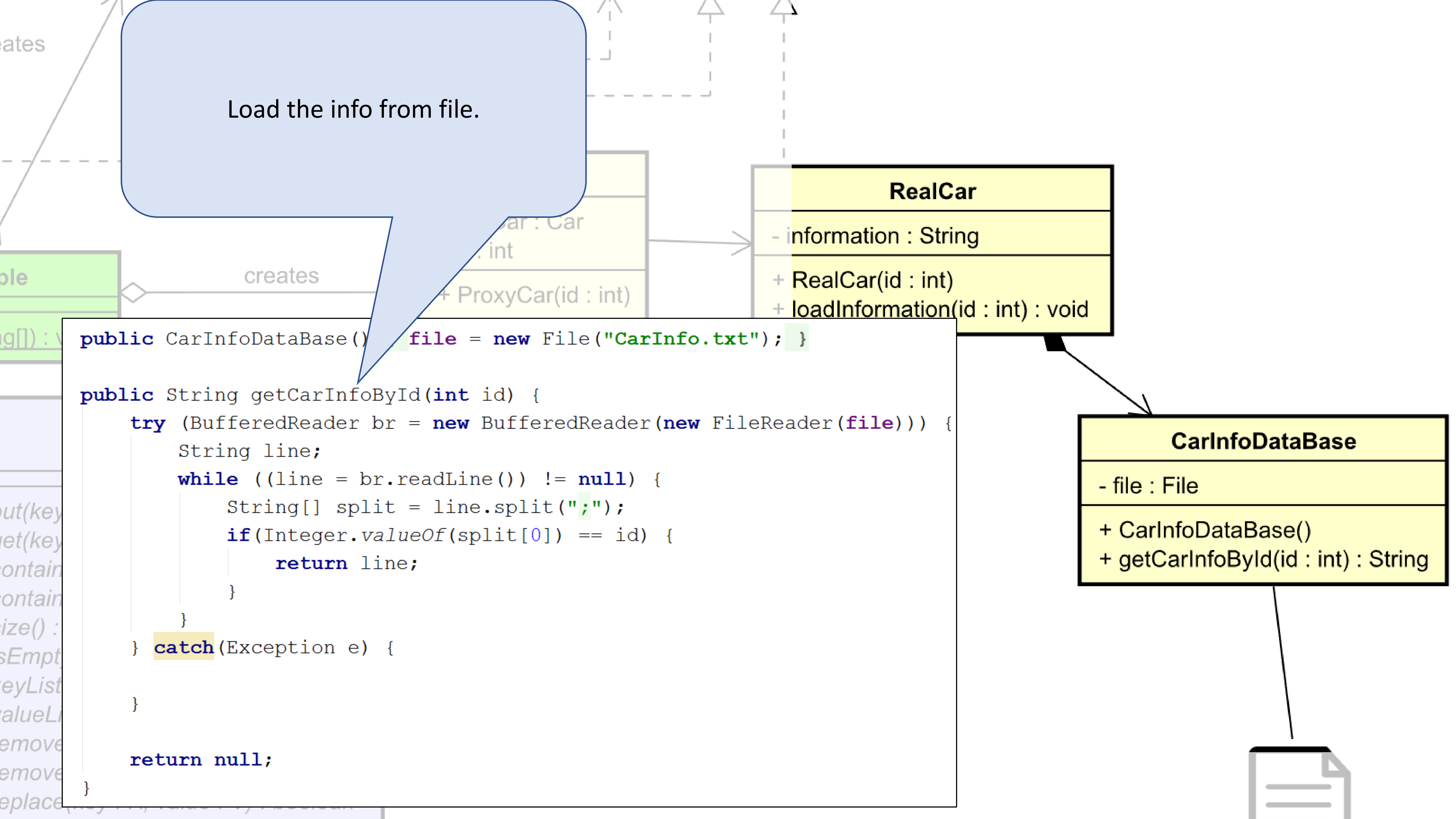
Use the CarInfoDataBase to getCarInfoByID



The file contains car info



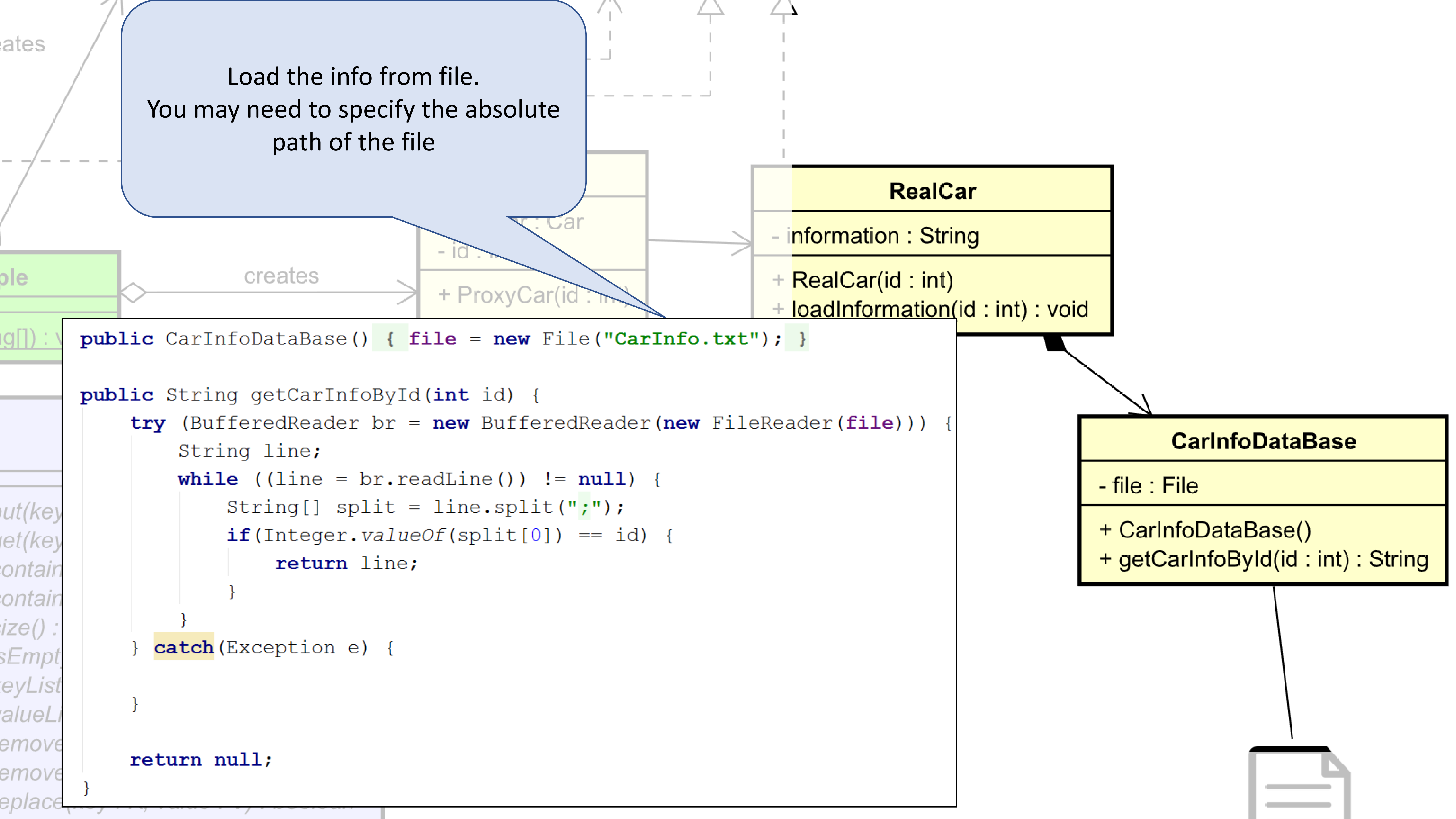
```
0;Citizen of Bridgeville;Make: Opel;Model: Meriva;Doors: 5
1;Citizen of Bridgeville;Make: Opel;Model: Corsa;Doors: 3
2;Citizen of Bridgeville;Make: Mercedes;Model: A180;Doors: 5
3;Citizen of Bridgeville;Make: Mercedes;Model: B200;Doors: 5
4;Citizen of Bridgeville;Make: Nissan;Model: Quashqai;Doors: 5
5;Citizen of Bridgeville;Make: Nissan;Model: Leaf;Doors: 5
6;Citizen of Bridgeville;Make: Volvo;Model: XC90;Doors: 5
```



Load the info from file.

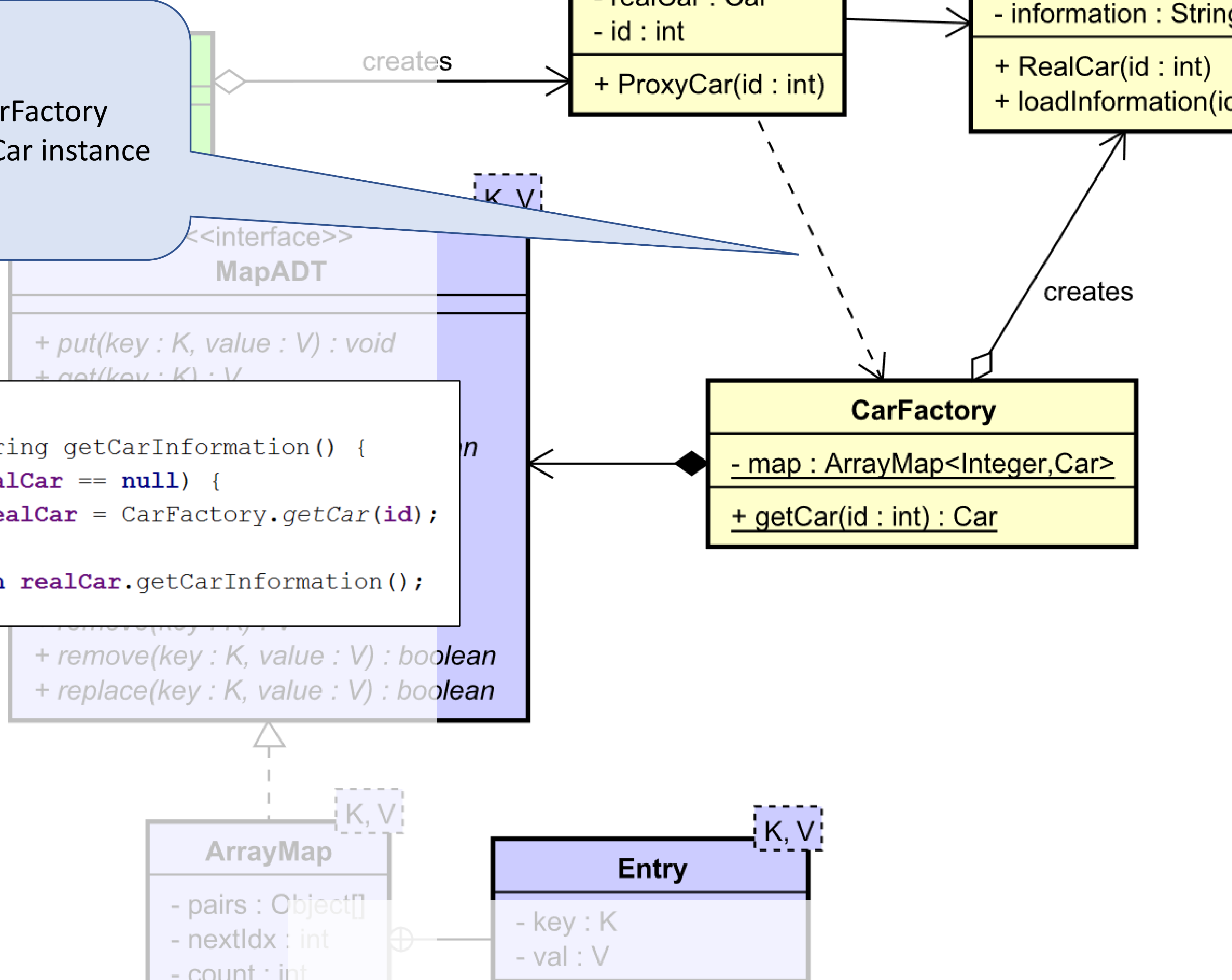
```
public CarInfoDataBase()  file = new File("CarInfo.txt"); }

public String getCarInfoById(int id) {
    try (BufferedReader br = new BufferedReader(new FileReader(file))) {
        String line;
        while ((line = br.readLine()) != null) {
            String[] split = line.split(";");
            if(Integer.valueOf(split[0]) == id) {
                return line;
            }
        }
    } catch(Exception e) {

    }

    return null;
}
```

Load the info from file.
You may need to specify the absolute path of the file

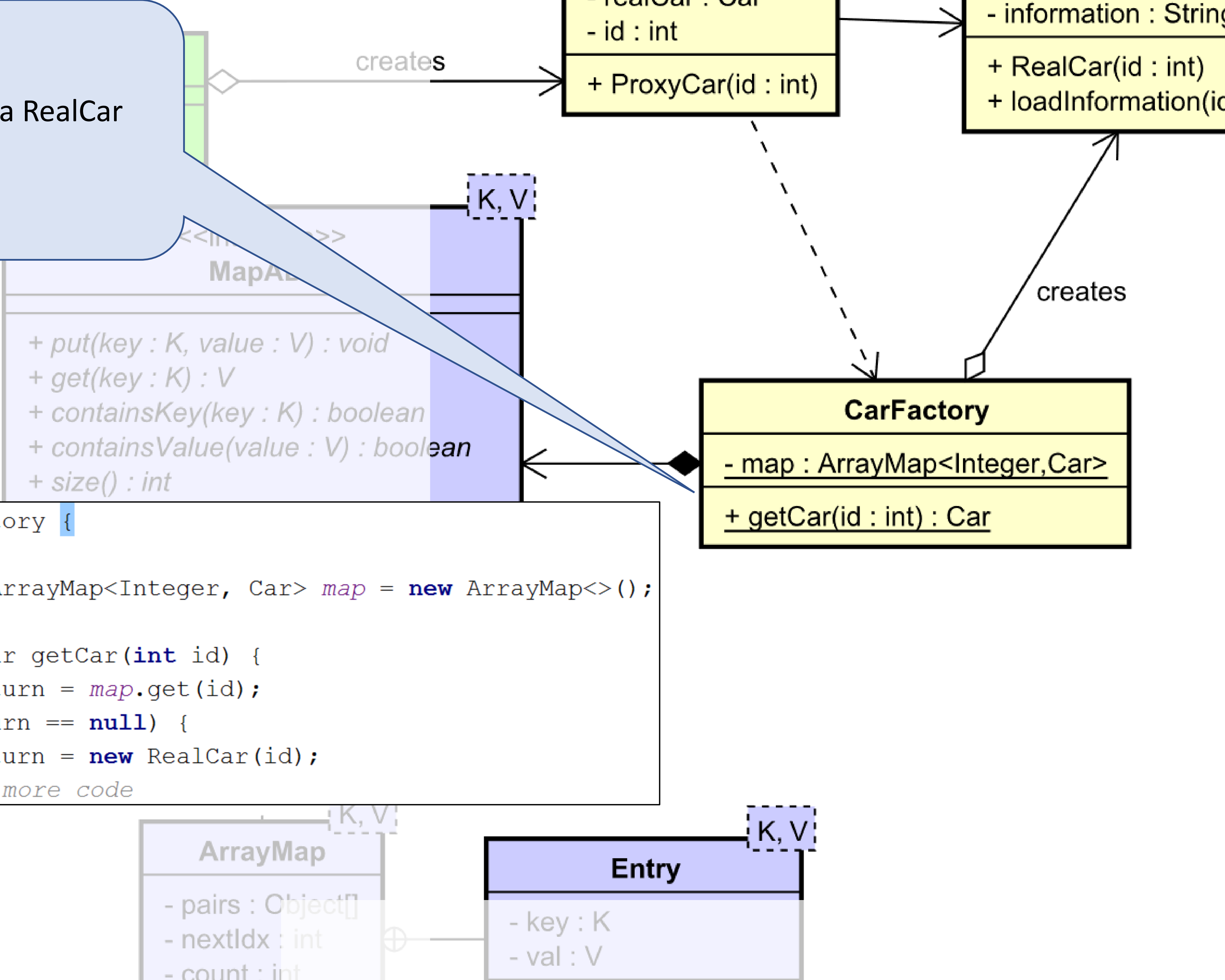


```java
public CarInfoDataBase() { file = new File("CarInfo.txt"); }

public String getCarInfoById(int id) {
    try (BufferedReader br = new BufferedReader(new FileReader(file))) {
        String line;
        while ((line = br.readLine()) != null) {
            String[] split = line.split(";");
            if(Integer.valueOf(split[0]) == id) {
                return line;
            }
        }
    } catch(Exception e) {

    }

    return null;
}
```

The ProxyCar uses CarFactory (flyweight) to get a RealCar instance



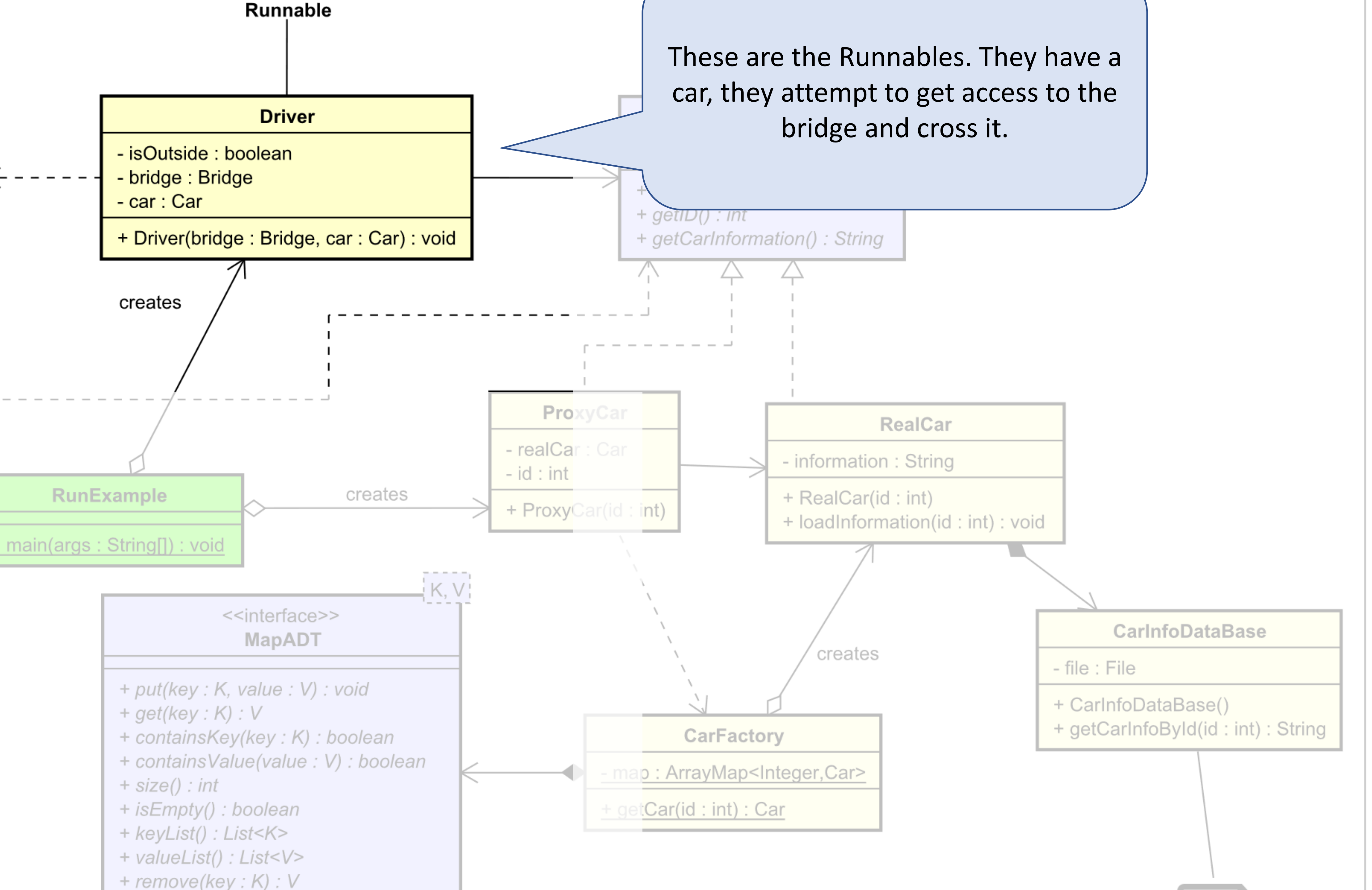
```java
@Override
public String getCarInformation() {
    if(realCar == null) {
        realCar = CarFactory.getCar(id);
    }
    return realCar.getCarInformation();
}
```

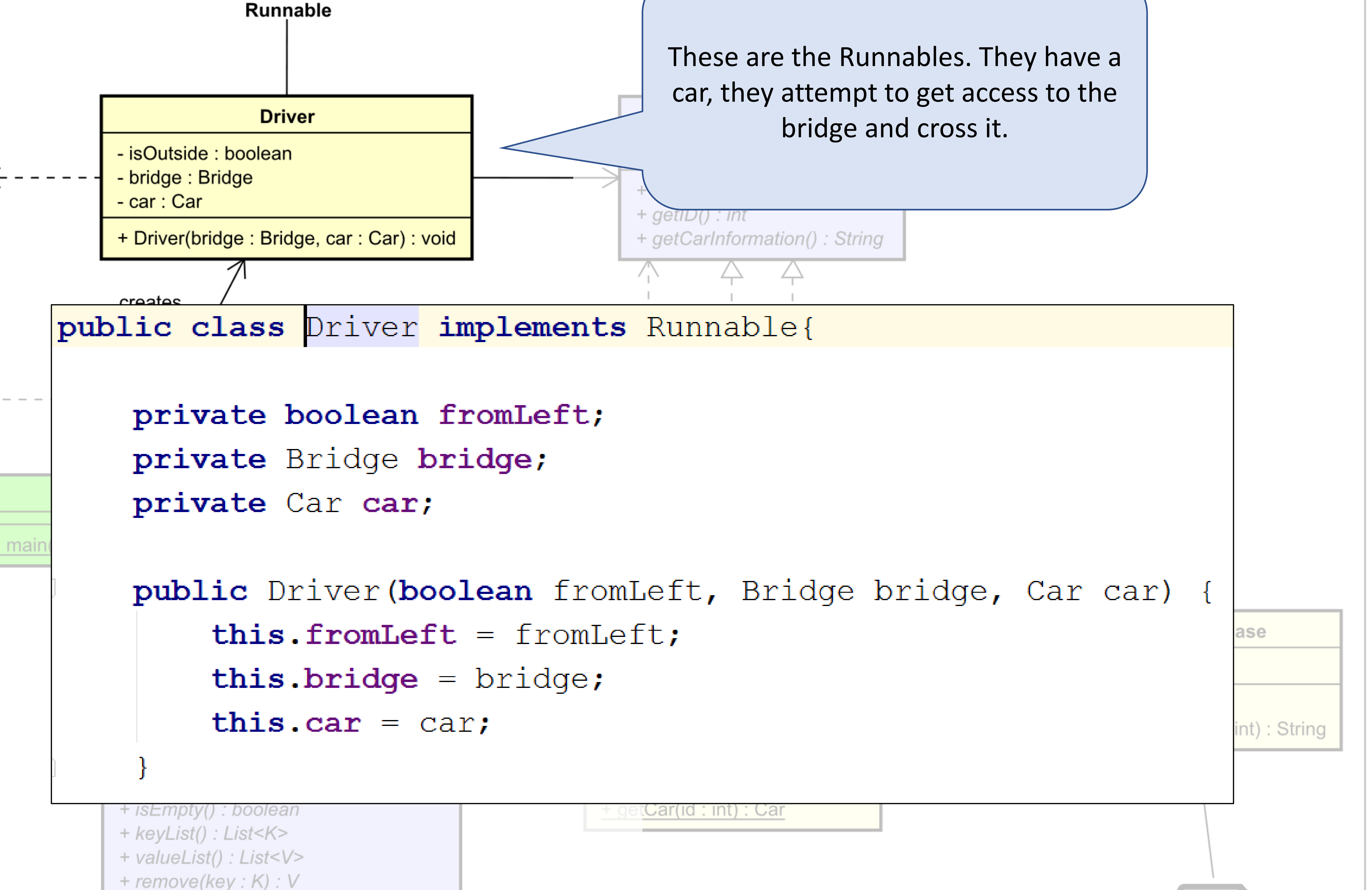


The CarFactory creates a RealCar from this id.

```
public class CarFactory {

    private static ArrayMap<Integer, Car> map = new ArrayMap<>();

    public static Car getCar(int id) {
        Car carToReturn = map.get(id);
        if(carToReturn == null) {
            carToReturn = new RealCar(id);
            // .... more code
```

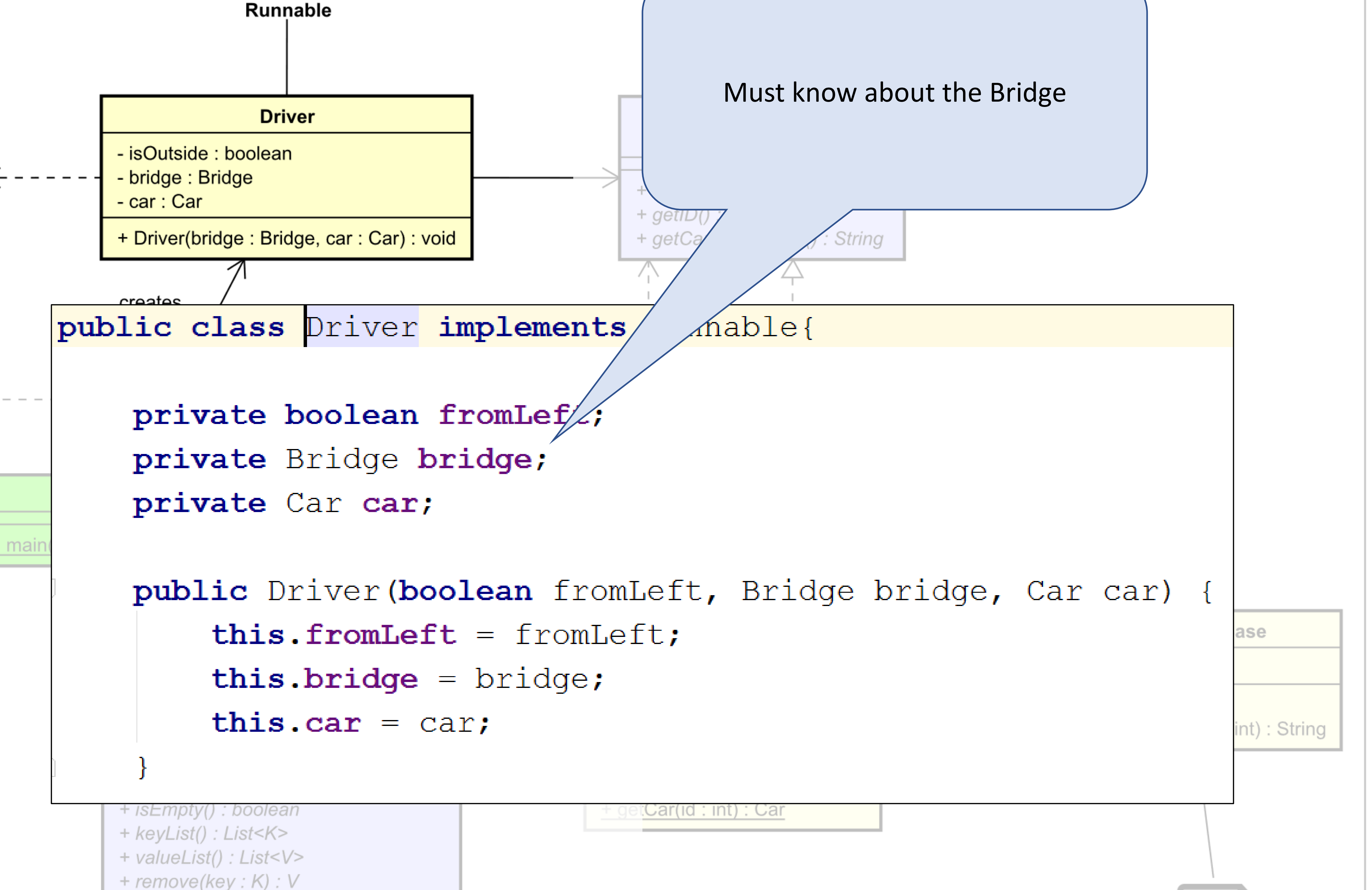
These are the Runnables. They have a car, they attempt to get access to the bridge and cross it.



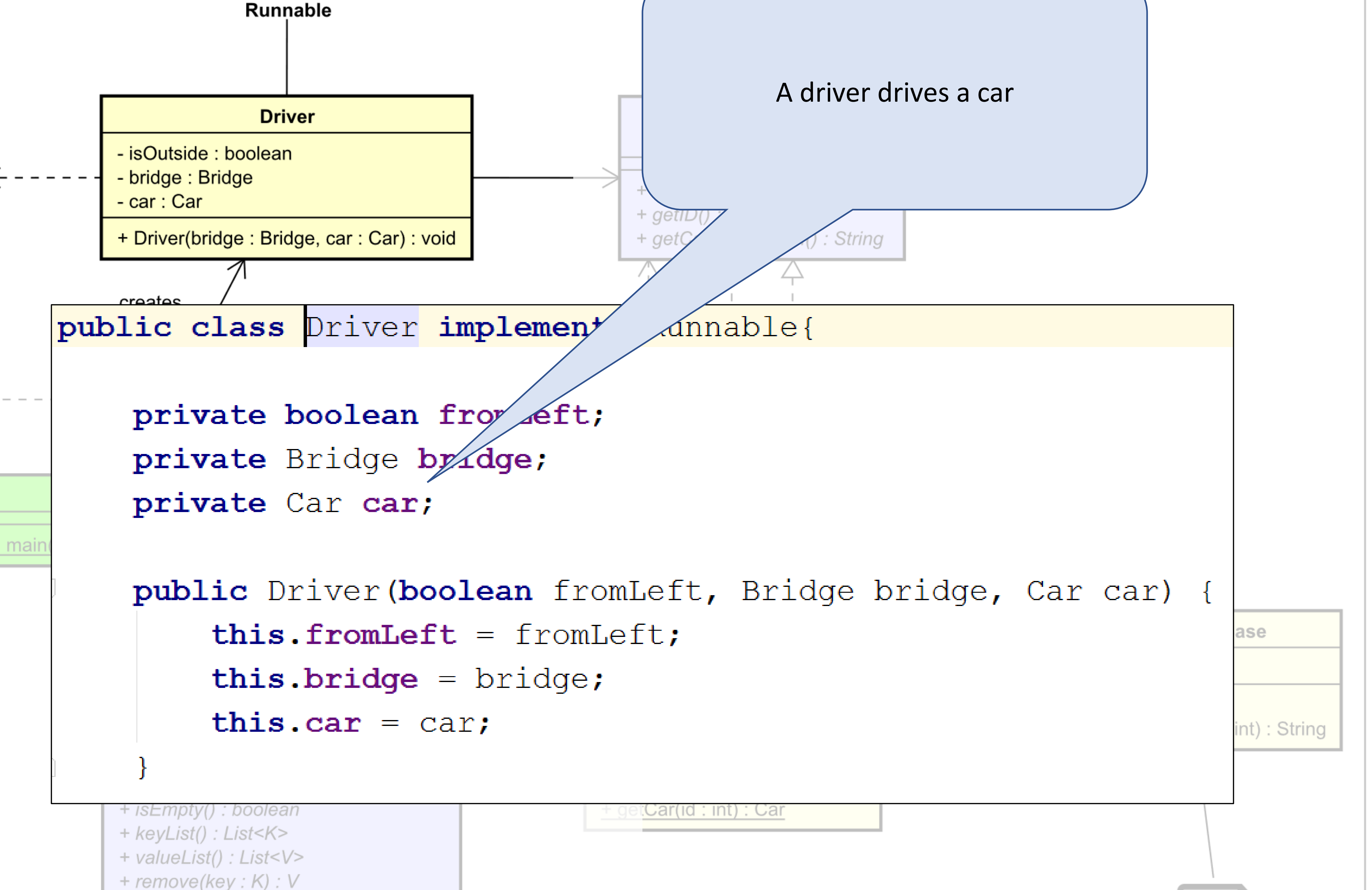


These are the Runnables. They have a car, they attempt to get access to the bridge and cross it.

```java
public class Driver implements Runnable{

    private boolean fromLeft;
    private Bridge bridge;
    private Car car;

    public Driver(boolean fromLeft, Bridge bridge, Car car) {
        this.fromLeft = fromLeft;
        this.bridge = bridge;
        this.car = car;
    }
```

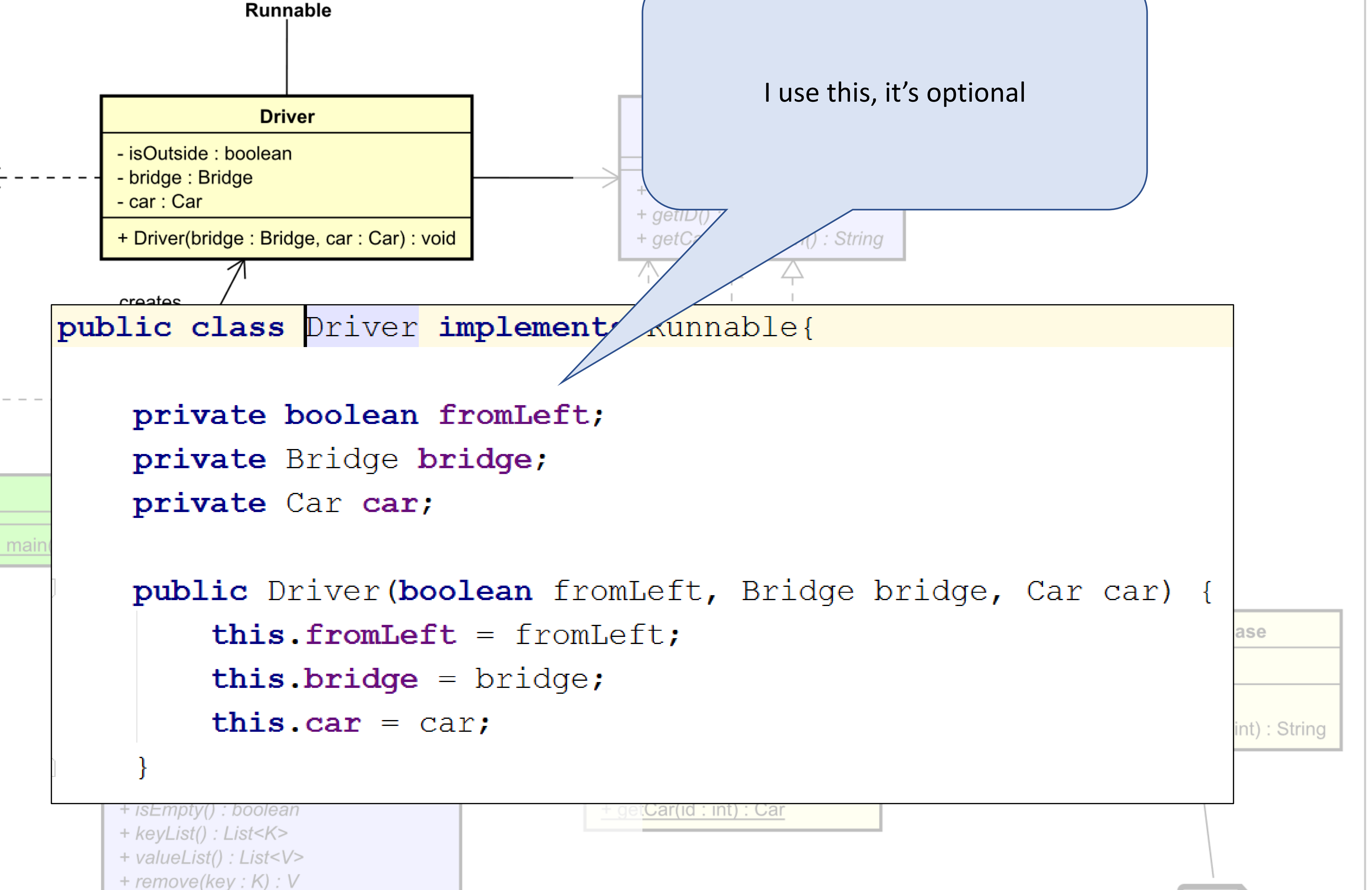


```
public class Driver implements Runnable{

    private boolean fromLeft;
    private Bridge bridge;
    private Car car;

    public Driver(boolean fromLeft, Bridge bridge, Car car) {
        this.fromLeft = fromLeft;
        this.bridge = bridge;
        this.car = car;
    }
```



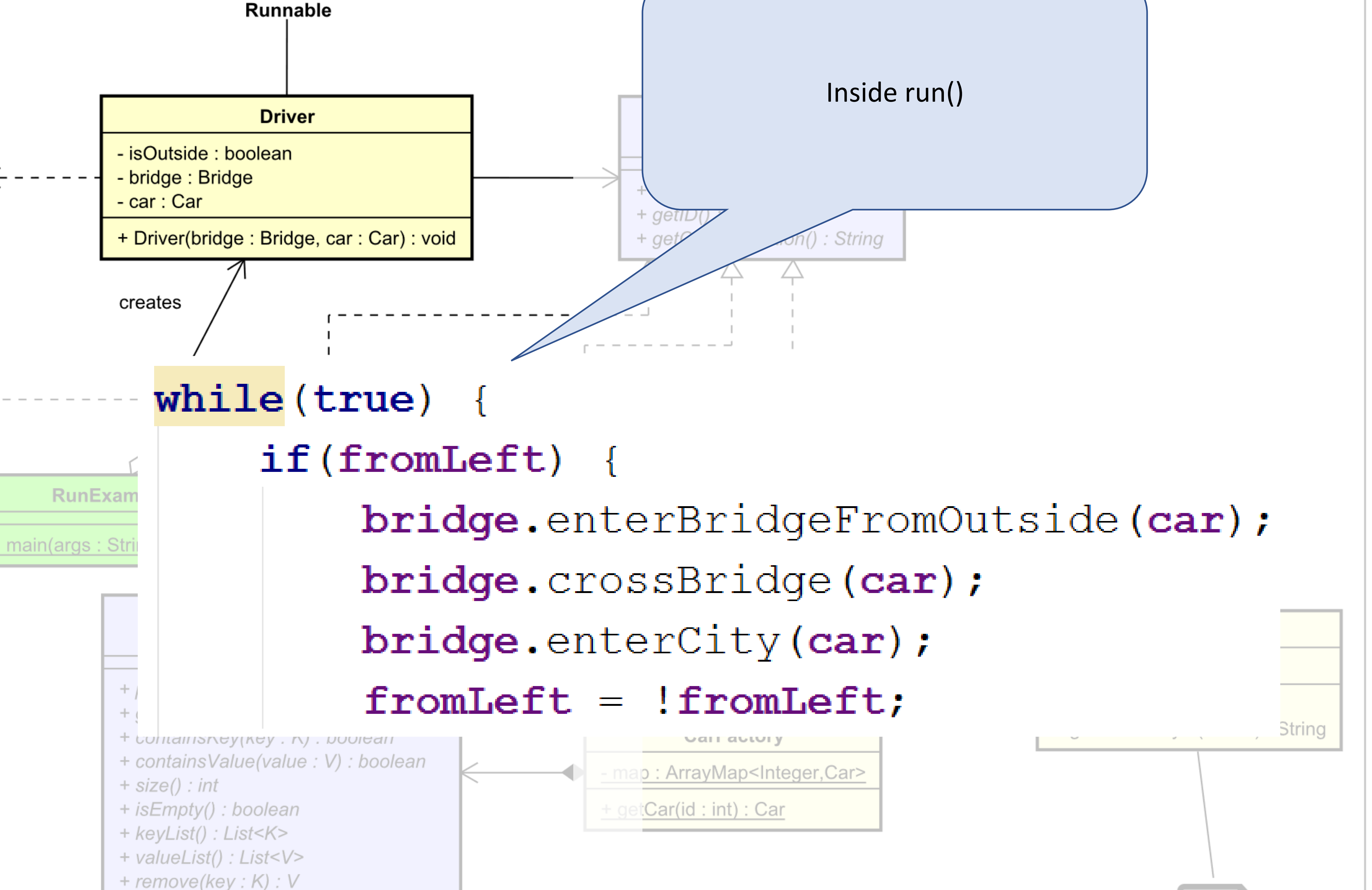
A driver drives a car

```
public class Driver implements Runnable{

    private boolean fromLeft;
    private Bridge bridge;
    private Car car;

    public Driver(boolean fromLeft, Bridge bridge, Car car) {
        this.fromLeft = fromLeft;
        this.bridge = bridge;
        this.car = car;
    }
```



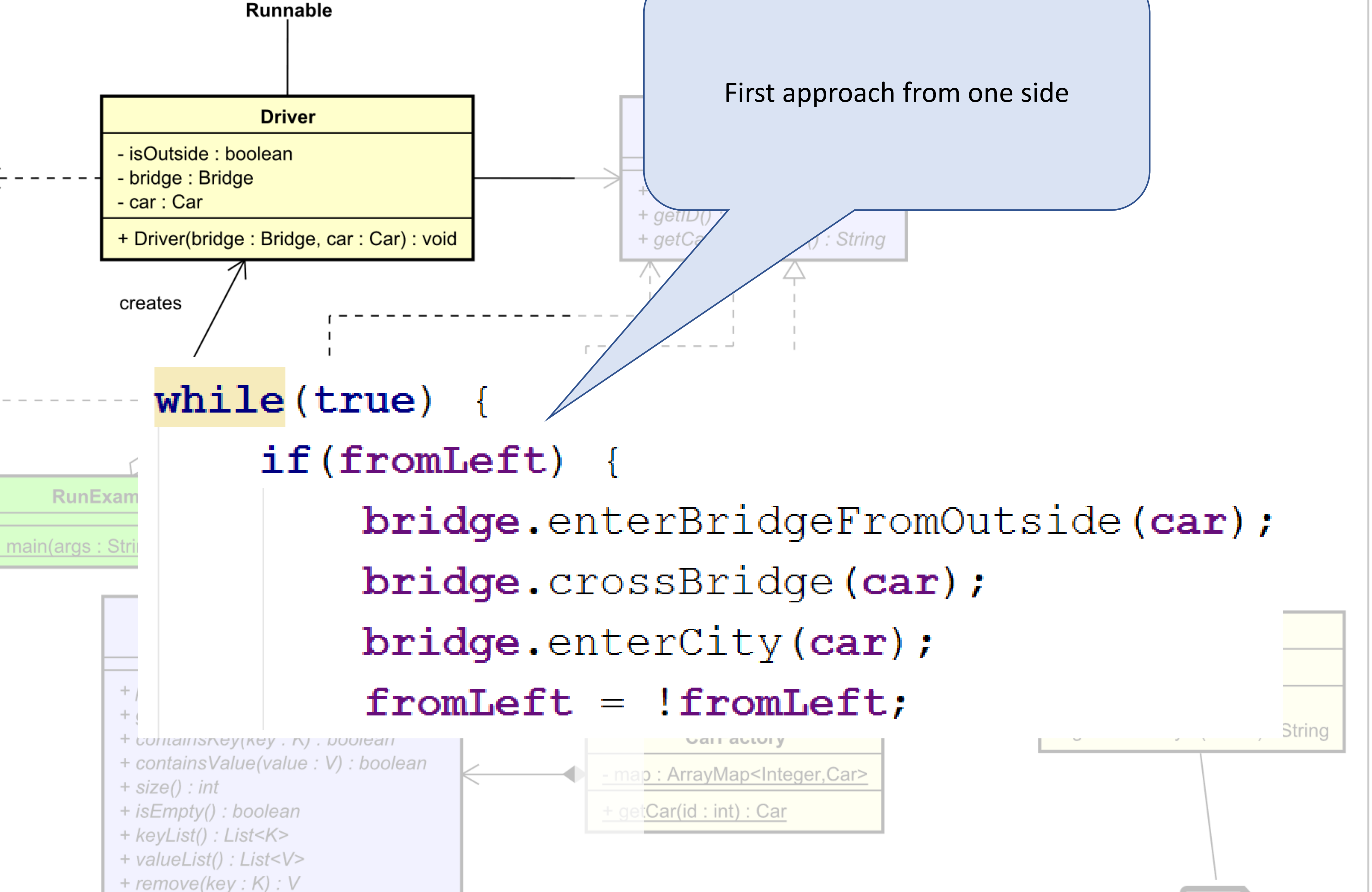
I use this, it’s optional

```java
public class Driver implements Runnable{

    private boolean fromLeft;
    private Bridge bridge;
    private Car car;

    public Driver(boolean fromLeft, Bridge bridge, Car car) {
        this.fromLeft = fromLeft;
        this.bridge = bridge;
        this.car = car;
    }
```



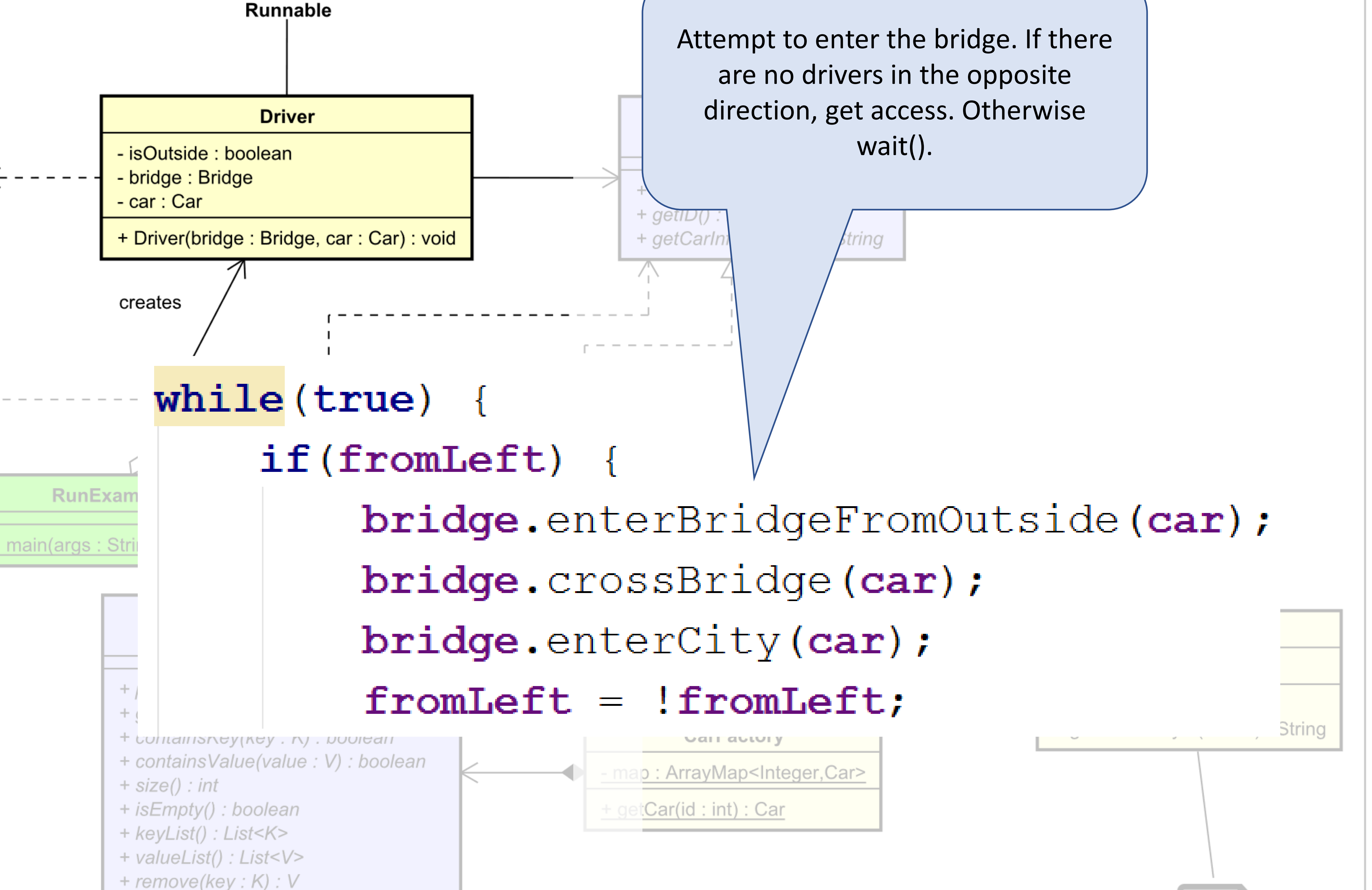
Inside run()

```
while(true) {
    if(fromLeft) {
        bridge.enterBridgeFromOutside(car);
        bridge.crossBridge(car);
        bridge.enterCity(car);
        fromLeft = !fromLeft;
```



First approach from one side

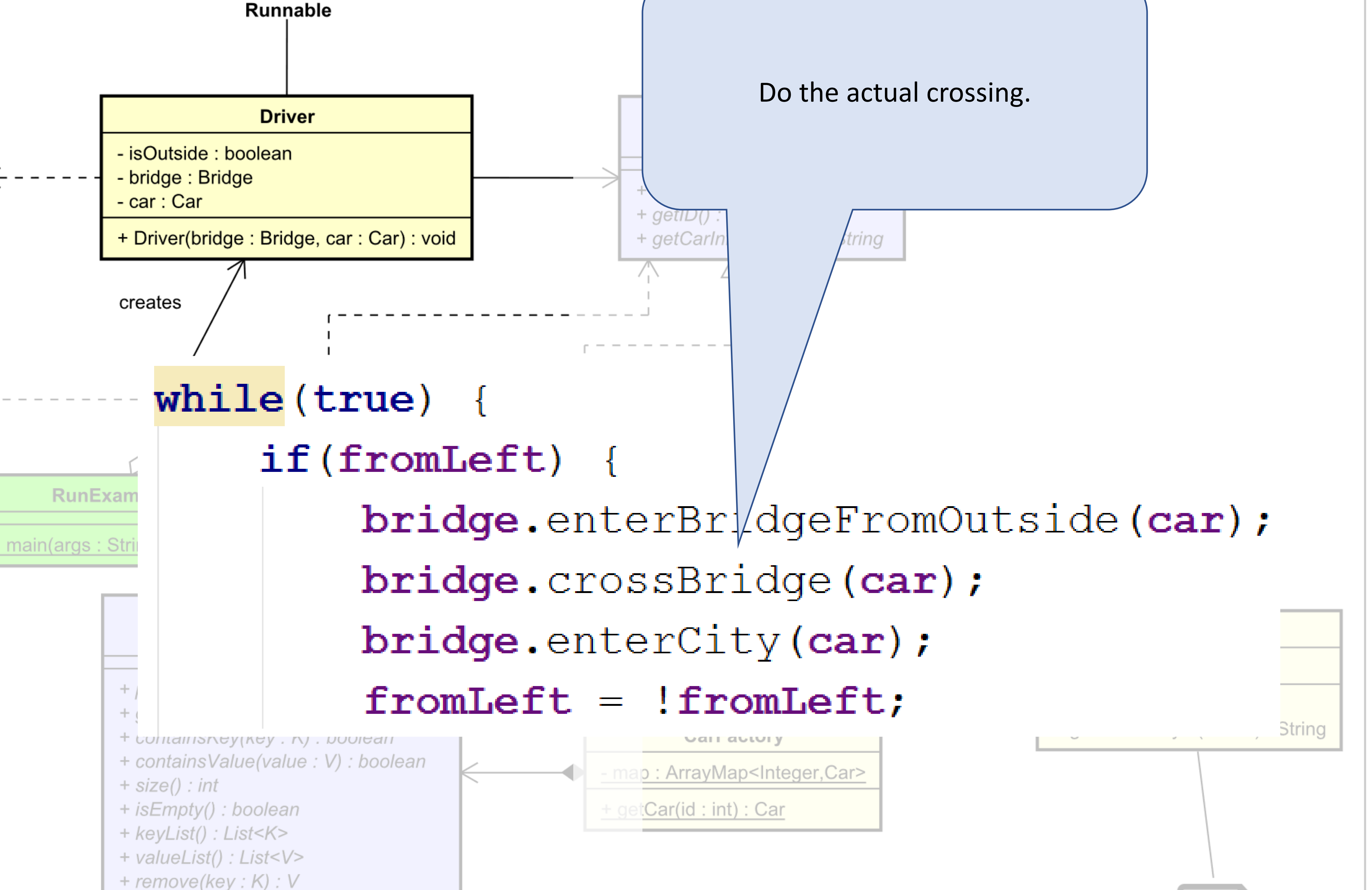
```
while(true) {
    if(fromLeft) {
        bridge.enterBridgeFromOutside(car);
        bridge.crossBridge(car);
        bridge.enterCity(car);
        fromLeft = !fromLeft;
```



Attempt to enter the bridge. If there are no drivers in the opposite direction, get access. Otherwise wait().

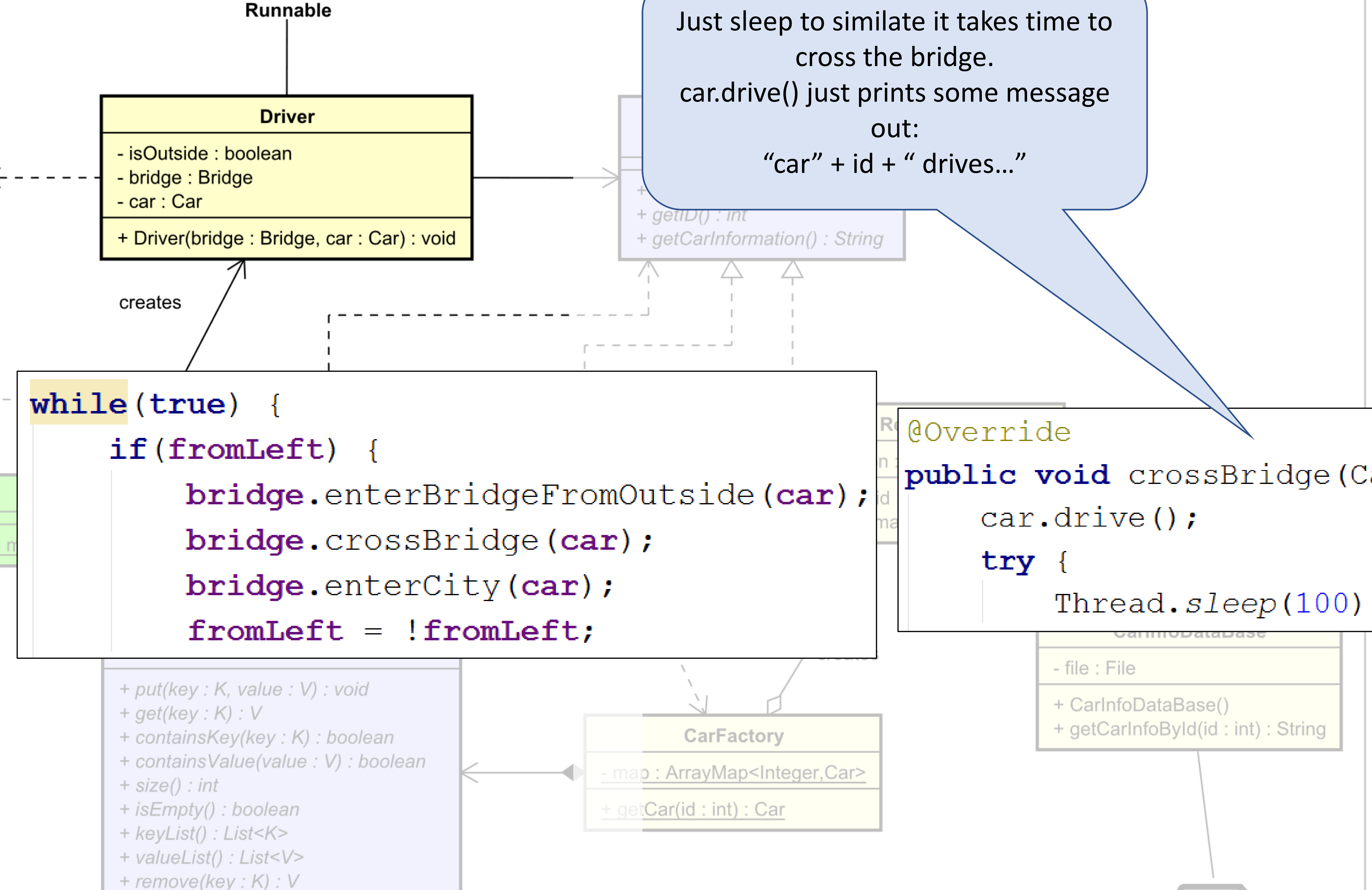
```
while(true) {
    if(fromLeft) {
        bridge.enterBridgeFromOutside(car);
        bridge.crossBridge(car);
        bridge.enterCity(car);
        fromLeft = !fromLeft;
```

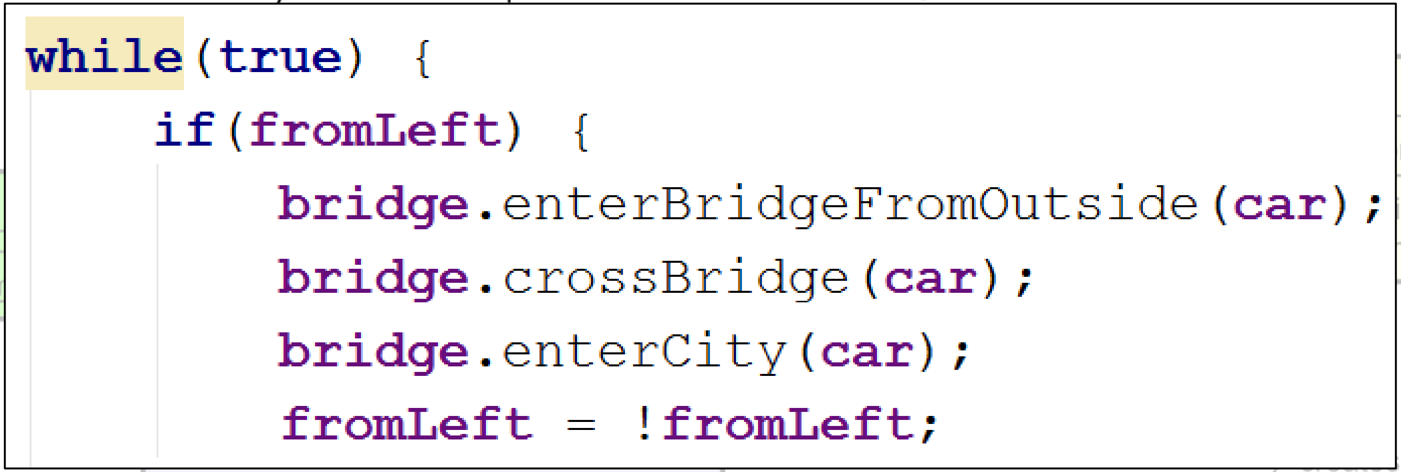
Do the actual crossing.



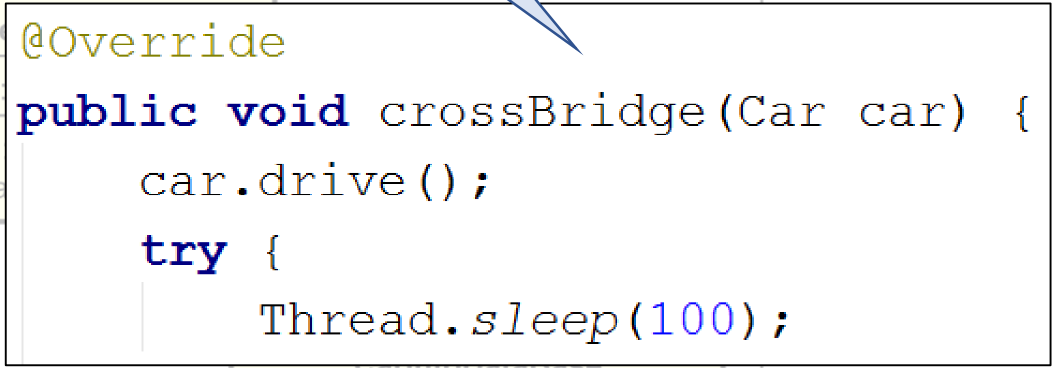
```
while(true) {
    if(fromLeft) {
        bridge.enterBridgeFromOutside(car);
        bridge.crossBridge(car);
        bridge.enterCity(car);
        fromLeft = !fromLeft;
```

Just sleep to similate it takes time to cross the bridge.
car.drive() just prints some message out:
“car” + id + “ drives…”



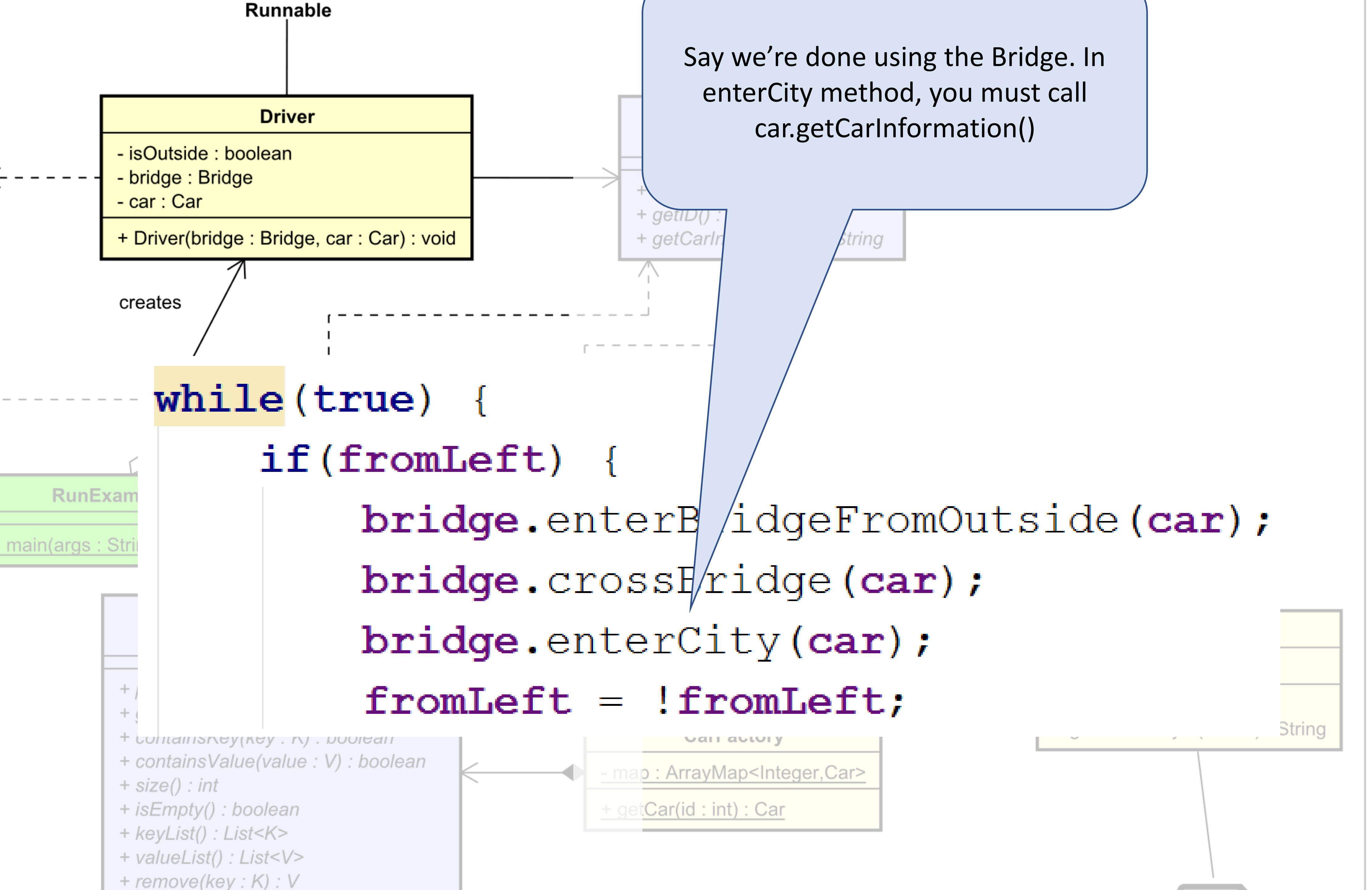


```
while(true) {
    if(fromLeft) {
        bridge.enterBridgeFromOutside(car);
        bridge.crossBridge(car);
        bridge.enterCity(car);
        fromLeft = !fromLeft;
```



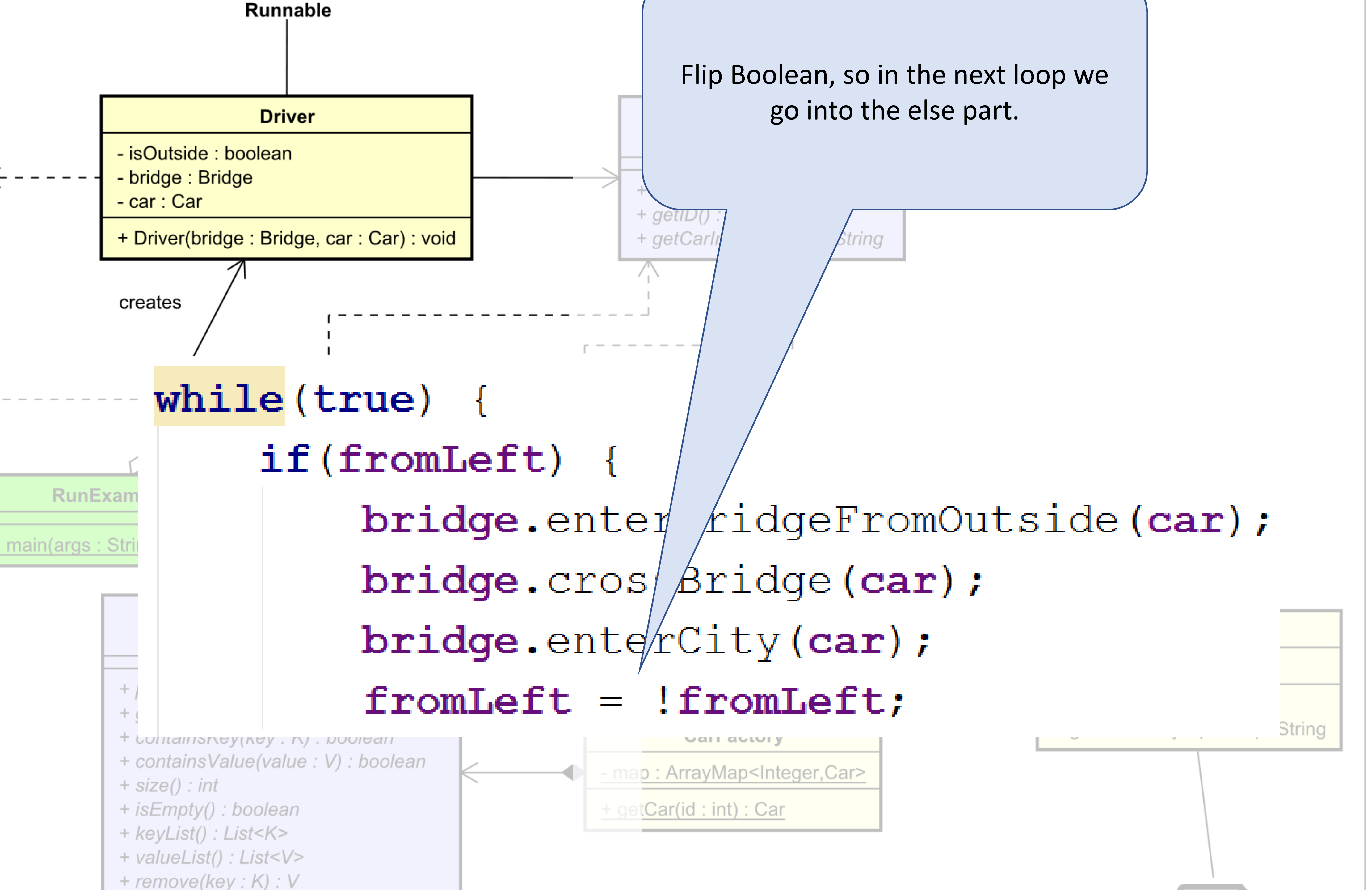
```
@Override
public void crossBridge(Car car) {
    car.drive();
    try {
        Thread.sleep(100);
```

Say we're done using the Bridge. In enterCity method, you must call car.getCarInformation()

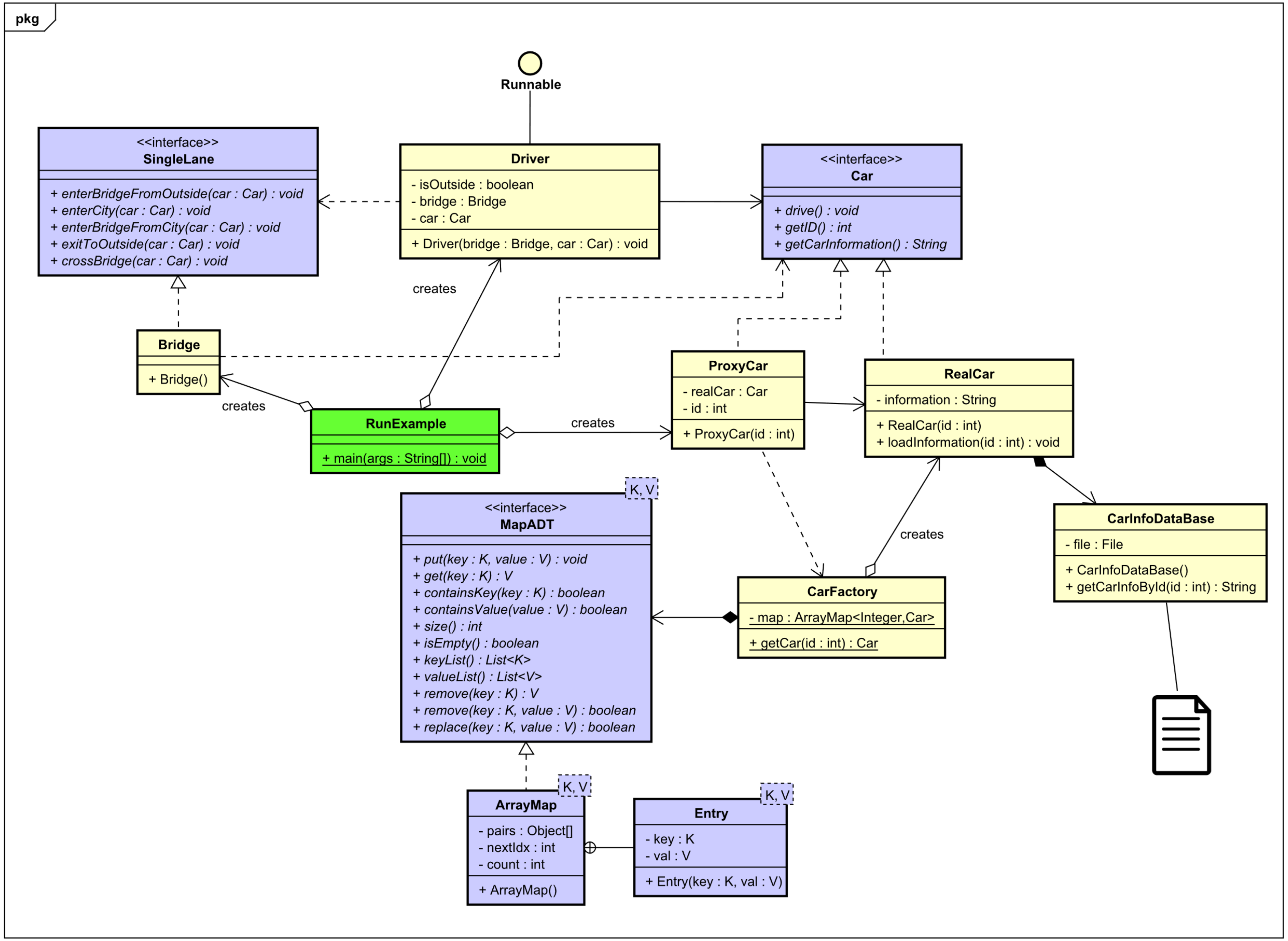


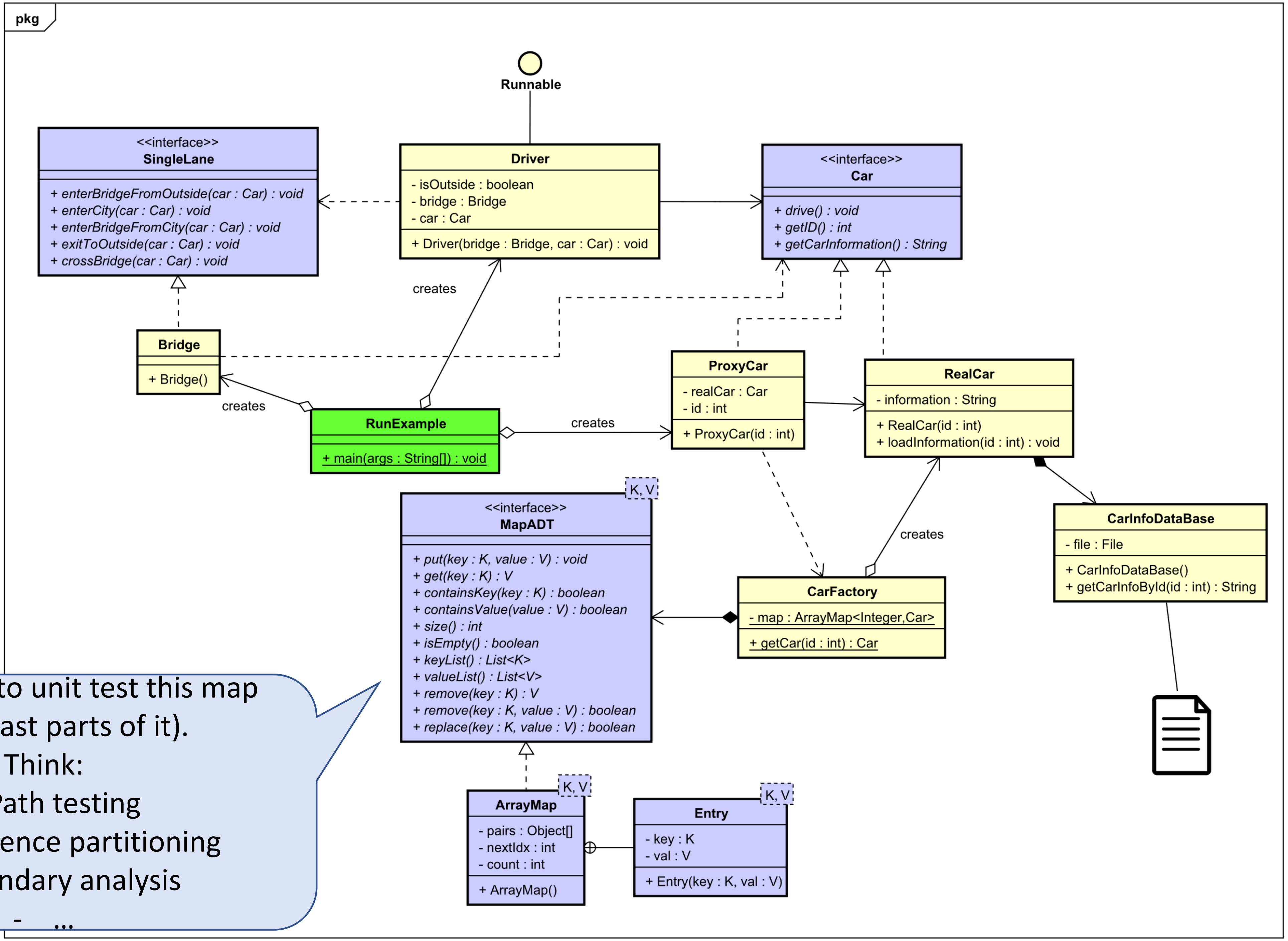
```
while(true) {
    if(fromLeft) {
        bridge.enterBridgeFromOutside(car);
        bridge.crossBridge(car);
        bridge.enterCity(car);
        fromLeft = !fromLeft;
```

Flip Boolean, so in the next loop we go into the else part.



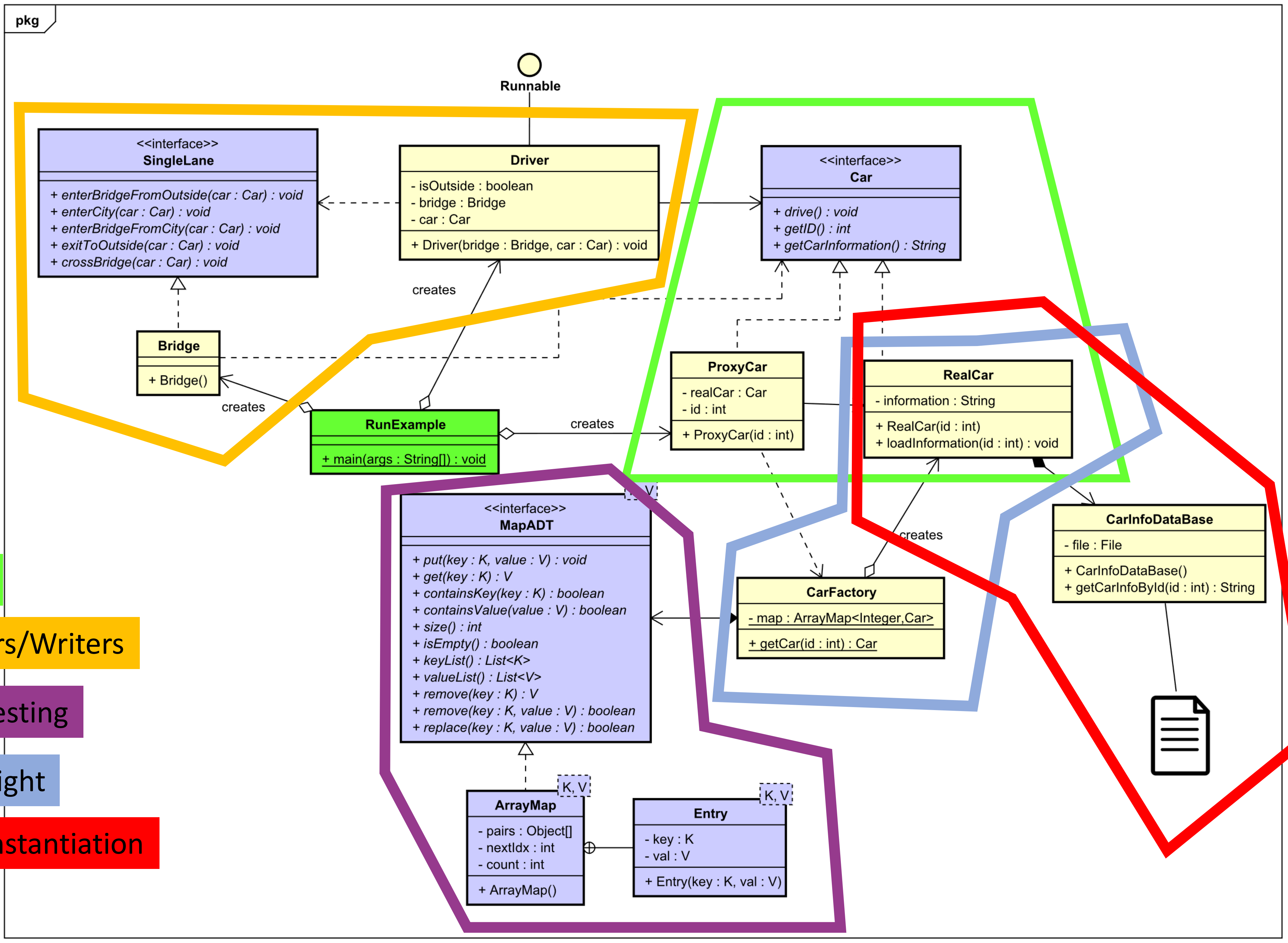
```
while(true) {
    if(fromLeft) {
        bridge.enterBridgeFromOutside(car);
        bridge.crossBridge(car);
        bridge.enterCity(car);
        fromLeft = !fromLeft;
```


pkg
Runnable
<<interface>>
SingleLane
+ enterBridgeFromOutside(car : Car) : void
+ enterCity(car : Car) : void
+ enterBridgeFromCity(car : Car) : void
+ exitToOutside(car : Car) : void
+ crossBridge(car : Car) : void
Driver
- isOutside : boolean
- bridge : Bridge
- car : Car
+ Driver(bridge : Bridge, car : Car) : void
<<interface>>
Car
+ drive() : void
+ getID() : int
+ getCarInformation() : String
creates
Bridge
+ Bridge()
creates
RunExample
+ main(args : String[]) : void
creates
ProxyCar
- realCar : Car
- id : int
+ ProxyCar(id : int)
RealCar
- information : String
+ RealCar(id : int)
+ loadInformation(id : int) : void
K, V
<<interface>>
MapADT
+ put(key : K, value : V) : void
+ get(key : K) : V
+ containsKey(key : K) : boolean
+ containsValue(value : V) : boolean
+ size() : int
+ isEmpty() : boolean
+ keyList() : List<K>
+ valueList() : List<V>
+ remove(key : K) : V
+ remove(key : K, value : V) : boolean
+ replace(key : K, value : V) : boolean
creates
CarInfoDataBase
- file : File
+ CarInfoDataBase()
+ getCarInfoById(id : int) : String
CarFactory
- map : ArrayMap<Integer,Car>
+ getCar(id : int) : Car
K, V
ArrayMap
- pairs : Object[]
- nextIdx : int
- count : int
+ ArrayMap()
K, V
Entry
- key : K
- val : V
+ Entry(key : K, val : V)



You’re going to unit test this map
(or at least parts of it).
Think:

- Path testing
- Equivalence partitioning
- Boundary analysis
- …



🟩 Proxy

🟨 Readers/Writers

🟪 Unit testing

🟦 Flyweight

🟥 Lazy instantiation