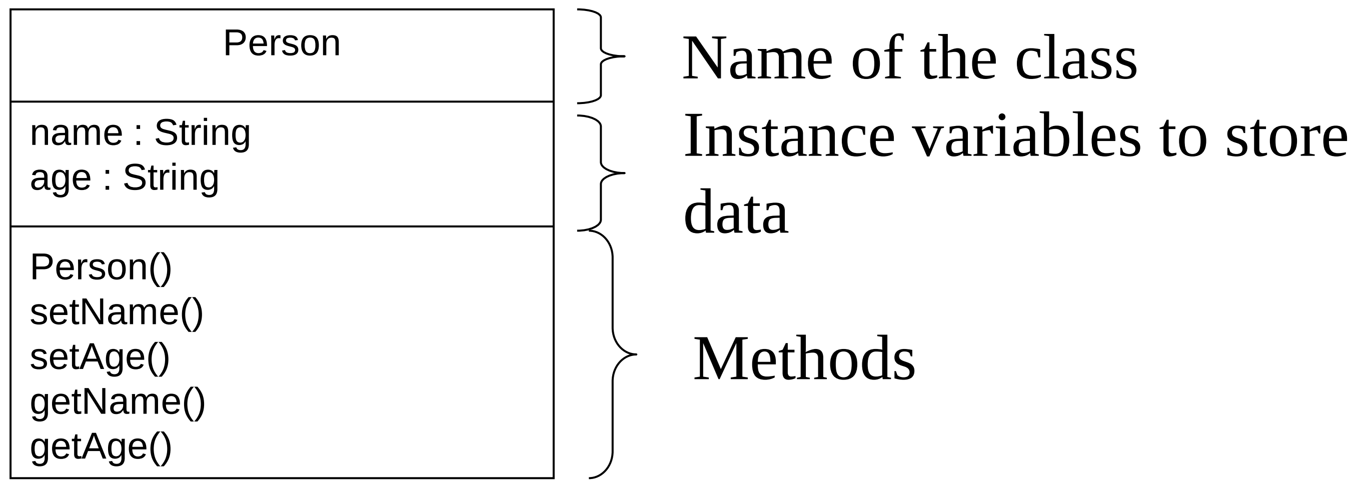
OOP Encapsulation

Example – a Person class

- We can have a Person class provider.
- Data: Person can store data about its:
  - Name
  - Age
- Methods: things that a person can do:
  - a Person can be created
  - set its data
  - give information about its data

Person Class Diagram

- A class diagram summarizes the class provider:



Person Objects

- The Person class provides the general definition of Person objects.
- However, we may want to create actual objects to represent actual Person.
- Let's say there are <u>three Person</u>: *adiPerson, budiPerson and cindyPerson*
  - Because these are all person, they will all have the same *kind* of data and be able to perform the same methods, as defined by the Person class above.
  - However, the actual data *values* will be different.

Writing a Class Provider

- Before we can actually create objects, we have to write the class provider.
- The class provider just provides the basic definition for the class, and usually does not have <u>main method.</u>

Writing a Class Provider

- A class provider generally follows the following structure:

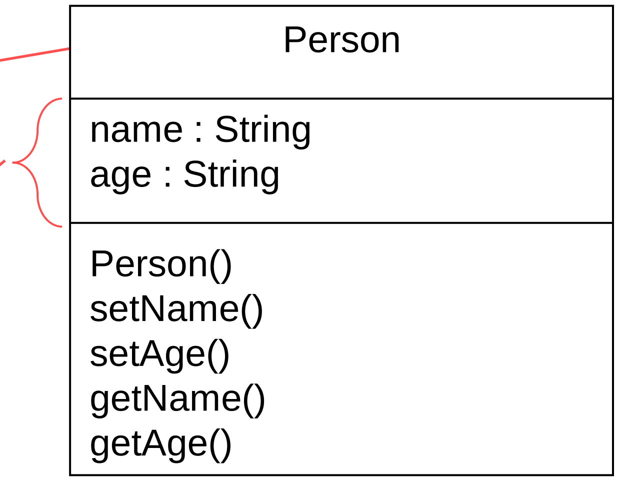
```
public class NameOfClass
{
   // private instance variables

   // Constructors, to create objects and initialize instance variables

   // Setters, which are used to set the values of instance variables

   // Getters, which are used to obtain the values
   // stored in the instance variables

   // a toString method to return the details of the
   // object as a String

   // other special methods
}
```

The Person Class

source code: Person.java

- Start by naming the class and declaring the instance variables:

```
/** The Person class
 */
public class Person
{
      // private instance variables
      private String name;
      private String age;

//.. continued
```



Constructors

- Constructors are methods to create objects and initialize the instance variables.
- The constructors are added to the class provider:

```
// Constructor without arguments, to set the instance
// variables to default values. There is no return type
// and a constructor method uses the name of the class.
public Person()
{
       name = "Adi";
       age = "20";
}
```

Constructors

- Generally you should write a constructor without arguments and one with arguments:

```
// A Constructor with arguments to set the
// instance variables to the values passed in as arguments
public Person(String inName, String inAge)
{
      name = inName;
      age = inAge;
}
```

Setters

```
// Setters are used to set the values
// of the instance variables. Do not confuse the
// argument (e.g inName) with the instance variable (name)
public void setName (String inName) {
        name = inName;
}

public void setAge (String inAge){
        age = inAge;
}
```

Getters

```
// Getters are used to obtain the values
// stored in the instance variables. The return type
// will be the type of the instance variable
public String getName(){
       return name;
}

public String getAge() {
       return age;
}
```

toString Method

- The toString() method is used to return the information stored in an object as a String.

```
// a toString method to return the details of the
// object as a String
public String toString()
{
      return “my name is " + name + “. I am " + age
             + " years old.";
}
```

Compiling the Class Provider

- You can compile this class provider but you will not be able to execute it as there is no main() method.
- We will write a class with a main() method to test this class provider in the next topic.

Creating Objects

- Let's try to create `Person` objects based on the `Person` class that we have written.
- We will create two persons,
  - Budi 25 Years
  - Cindy 23 years

Creating Objects

- In order to create an object, we will use the Constructor and the `new` operator.
  - The `new` operator will allocate space in memory for the object.
- We will also need a name to refer to the object.
  - Let's call our two persons *budiPerson* and *cindyPerson* respectively.

Using Constructors

- Examine the constructors in the `Person` class provider:
  - The no-arguments constructor receives no arguments and sets the instance variables to default values.

```
/** The Person class
 */
public class Person
{
        // private instance variables
        private String name;
        private String age;

       // Constructor without arguments
        public Person()
        {
                name = "unknown";
                age = "unknown";
        }
```

Using Constructors

- Thus, if we use the no-arguments constructor in our statement as follows:

```
Person budiPerson = new Person();
```

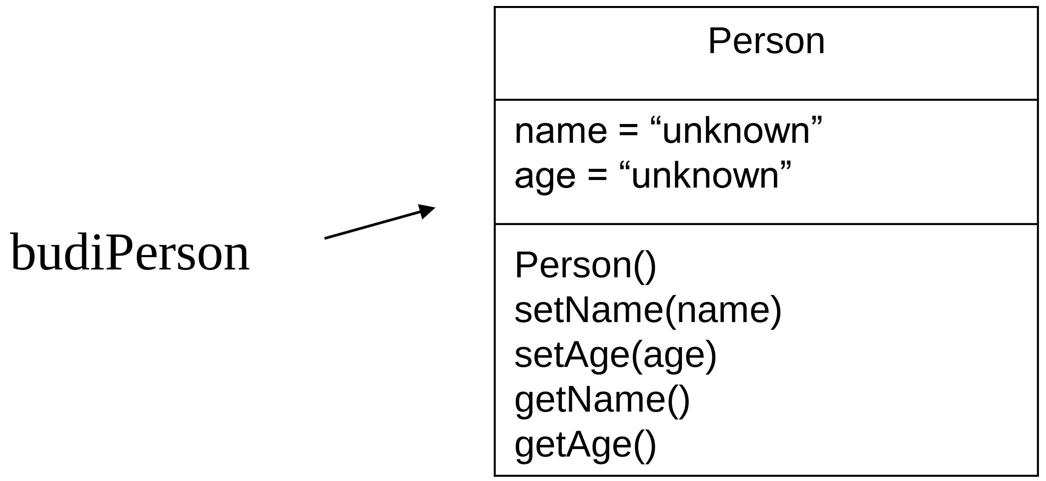
type of object

name of object

allocate memory

Constructor call

The following object will be created:



Constructor with Arguments

- The constructor with arguments will set the instance variables to the arguments passed in:

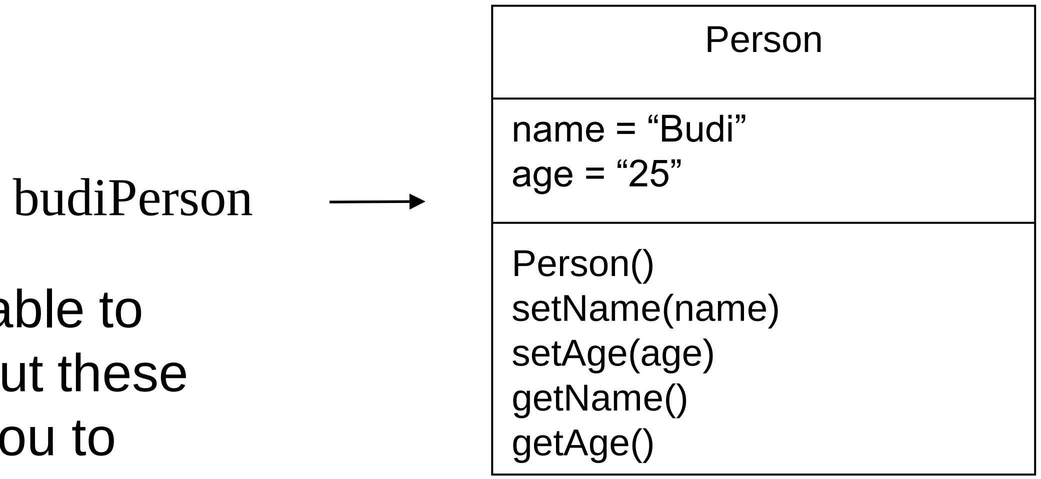
```
// A Constructor with arguments to set the instance
variables to the values passed in as arguments

public Person(String inName, String inAge)
{

      name = inName;
      age = inAge;
}
```

Constructor with Arguments

- Thus, if we use the constructor in our statement as follows:

```
Person budiPerson = new Person("Budi", "25");
```

The following object will be created instead:



* In our programs, we are unable to actually view these objects, but these diagrams are meant to help you to imagine that there are actually two persons with different data values.

Programming Example

Source code: TestPerson.java

```
/** A class to test creating Person objects.
 * Make sure that the Person class is in the
 * same directory as this class.
 */
public class TestPerson
{
     // the program will run from the main method
     public static void main(String[] args)
     {
        //using the no-args constructor
        Person budiPerson = new Person();

        // using the constructor with arguments
        Person cindyPerson = new Person("Cindy", "23");

        // display the details of the first person
        System.out.println("Hello, " + budiPerson);

        // display the details of the second person
        System.out.println("Hello, " + cindyPerson);
     }
}
```

Using Class Methods

- We can use the other methods in the class to change the data stored in the instance variables.
- Recall that we cannot directly access the instance variables as they have been declared **private**.
- Let's try using the setters to change details for *budiPerson*, as we don't want it to remain "unknown"...

Exercise

- Add the following statements to the end of the program TestPerson.java and then display the details using the toString() method again:

```
budiPerson.setName(“Budi");
budiPerson.setAge(“25");
```

Using Class Methods

- The output from the previous Exercise would be as follows:

```
Hello, My name is unknown. I am unknown years old.
Hello, My name is Cindy. I am 23 years old.
Hello, My name is Budi. I am 25 years old.
```

- Because we are changing the details for the *budiPerson* object, we must make sure that we refer to the correct object when we are using the method.

Using Class Methods

- In order to use any of the methods in a class provider, you must know:
  - the name of the method
  - what arguments to use
  - the return type
  - what the method does

  This is the same as for static methods.
- However, when using class methods, you will need to determine for *which object* you are using the method.
  - e.g., you will need to decide whether you are using a method for *budiPerson* or for *cindyPerson*.

Person Class Methods

```
// no-args constructor
public Person()

// constructor with arguments
public Person(String inName, String inAge)

// setters
public void setName(String inName)
public void setAge(String inAge)

// getters
public String getName()
public String getAge()

// toString
public String toString()
```

Exercise

Car Class Methods

```
// no-args constructor
public Car()

// constructor with arguments
public Car(String inMake, String inModel, String inRegNo, int
KMtravelled)

// setters
public void setMake(String inMake)
public void setModel(String inModel)
public void setRegNo(String inRegNo)
public void setKMtravelled(int KMtravelled)

// getters
public String getMake()
public String getModel()
public String getRegNo()
public int getKMtravelled()

// toString
public String toString()

// a method to move the car several KM
public void move(int KM)
```

Exercise

- Write a program that will perform the following tasks:
  - create two different cars with different data
  - display the details of each car using the toString() method
  - move the first car 100 km
  - move the second car 50 km
  - using the setter, change the registration number of the second car
  - using the getters, display the details of both cars
  - using the method getKMtravelled(), determine which car has traveled further and display the details with the toString() method.

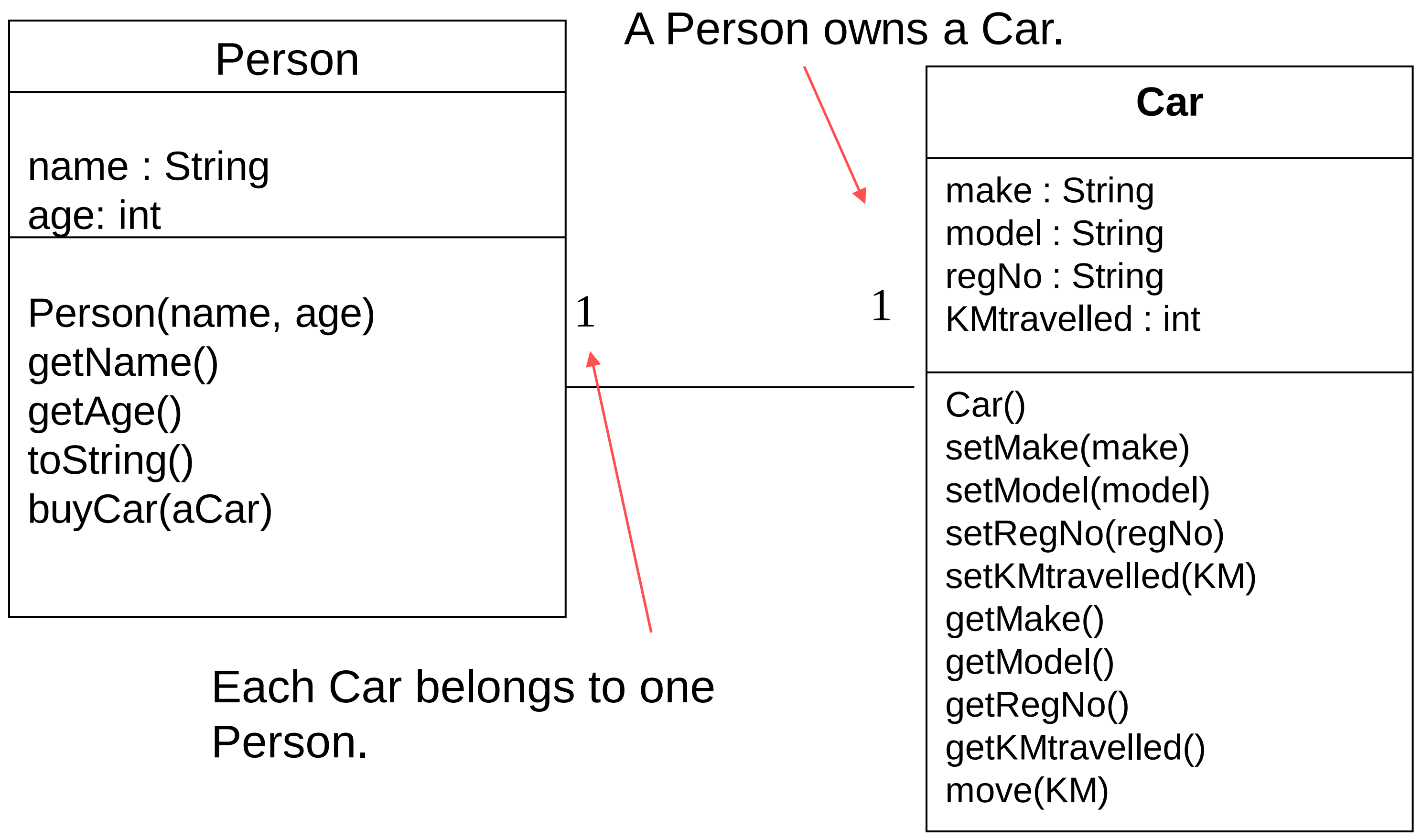
Bonus

Relationships Between Objects

- In a program, we usually create many objects of many different types. These objects are thus related in some way.
- We have already done this with the String class: a Person has a name, and the name is a String. Both Person and String are class types.

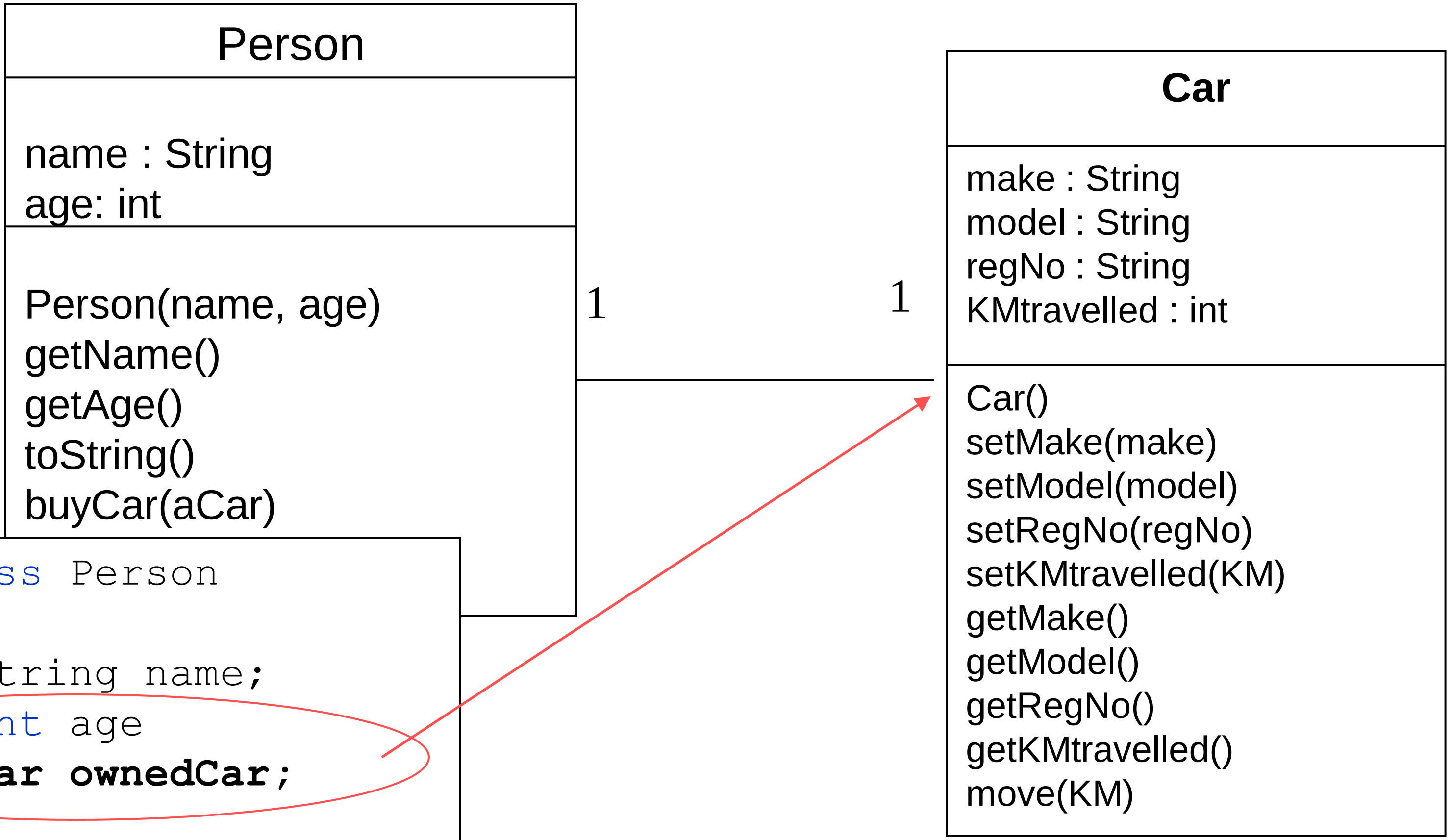
Relationships and The Class Diagram

- Let's examine the relationship between the two classes below, as shown in the class diagram:



Creating Relationships

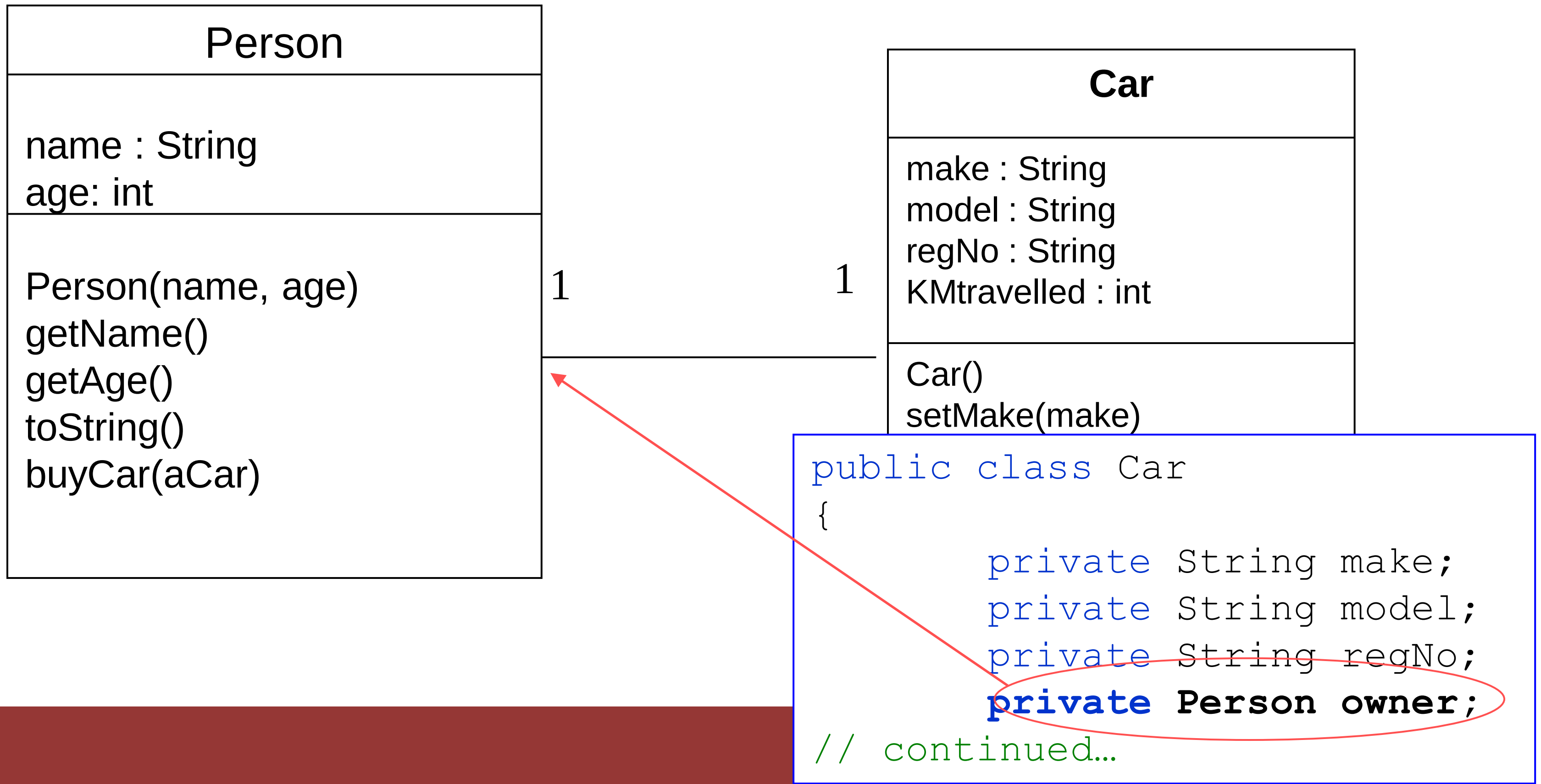
- The Class Providers for the Person and Car classes can be written as before.
- We use special instance variables to represent the relationship shown in the class diagram:



```
public class Person
{
 private String name;
 private int age
 private Car ownedCar;

// continued…
```

Creating Relationships

- The `Person` class type is used as the type for the instance variable owner in the `Car` class. This will create a link to a `Person` object.



```
public class Car
{
        private String make;
        private String model;
        private String regNo;
        private Person owner;
// continued…
```

Using Class Methods

- Modify the Car's toString() method to include the name of the owner.

```
// a toString method to return the details of the
// object as a String
public String toString()
{
        return "a " + make + " " + model +
               " with registration number "
               + regNo + " and KM travelled " + KMtravelled
               + " owned by " + owner.toString();
}
```

- The output from the previous Exercise would be as follows:

```
My new car is a BMW Z4 with registration number BBB345 and KM
travelled 0 owned by Adam 35
```