**Procedural code**

**Initial procedural version**

So here's the deal: you'll be modelling a car and a driver. Both are going to be ridiculously simple to cover just one small but important aspect of how to develop your object-oriented code.

Here's an initial version of the **Car** class:

**package ask;**

**class Car {**

**private int speed;**

**int getSpeed() {**

**return speed;**

**}**

**void setSpeed(int speed) {**

**this.speed = speed;**

**}**

**}**

There's just one private attribute along with its public accessor and mutator (getter and setter) methods; these are sometimes added semi-automatically by development tools.

We'll add a **Driver** class:

**package ask;**

**class Driver {**

**public static final int MAX\_SPEED = 130;**

**public static final int INITIAL\_SPEED = 117;**

**final Car car;  // example is simpler with default package-private access**

**Driver() {**

**car = new Car();**

**car.setSpeed(Driver.INITIAL\_SPEED);**

**}**

**int accelerate(int increment) {**

**int speed = car.getSpeed();**

**if (speed + increment < Driver.MAX\_SPEED) {**

**speed += increment;**

**car.setSpeed(speed);**

**}**

**return speed;**

**}**

**}**

We've tried to act responsibly by setting a maximum speed and only allowing the car to accelerate up to this limit.

Let's take it out for a spin:

**package ask;**

**public class Main {**

**public static void main(String... args) {**

**int increment = 10;**

**Driver driver = new Driver();**

**System.out.println("Initial speed: " + driver.car.getSpeed());**

**System.out.println("Accelerating by: " + increment);**

**driver.accelerate(increment);**

**System.out.println("New speed: " + driver.car.getSpeed());**

**System.out.println("Accelerating by: " + increment);**

**driver.accelerate(increment);**

**System.out.println("New speed: " + driver.car.getSpeed());**

**}**

**}**

`

Running the code gives the output:

Initial speed: 117

Accelerating by: 10

New speed: 127

Accelerating by: 10

New speed: 127

It seems to work perfectly, or at least it doesn't accelerate over the limit.

### Rogue driver

Well, we've got a teenager in the family who wants to borrow the car:

**package ask;**

**public class TeenageDriver extends Driver {**

**@Override**

**public int accelerate(int increment) {**

**int speed = car.getSpeed() + increment;**

**car.setSpeed(speed);**

**return speed;**

**}**

**}**

We'll see shortly how **extends Driver** works.

The result is as expected:

Initial speed: 117

Accelerating by: 10

New speed: 127

Accelerating by: 10

New speed: 137

Accelerating by: 10

New speed: 147

and probably even higher!

(Disclaimer: this example is not intended to slur teenage drivers; many are probably better drivers than their parents. ;^) )

**Object-oriented version**

The problem is that the code so far is not object-oriented. Yes it's written in an object-oriented language, but it's *procedural* code. An object-oriented class is meant to contain data + methods that act on the data; it is supposed to take responsibility for what it does with its data. The **Car** class doesn't do this. It does contain data, the **speed** attribute, but it just supplies this data to whoever asks for it. In this case the **Driver** class asks for the **speed** with the getter method, decides what to do with it, and then resets the **Car**'s **speed** attribute with the setter method.

This is not object-oriented programming! In object-oriented programming the **Driver** would tell the **Car** what to do, not ask the **Car** for data and decide what **Car** should do with it. **Car** must take responsibility for handling its own data.

Let's try again (in a different package):

**package tell;**

**class Car {**

**public static final int MAX\_SPEED = 130;**

**private int speed;**

**Car(int initialSpeed) {**

**speed = initialSpeed;**

**}**

**int accelerate(int increment) {**

**if (speed + increment < Car.MAX\_SPEED) {**

**speed += increment;**

**}**

**return speed;**

**}**

**}**

Now **Car** takes responsibility for its **speed** attribute. If someone calls the **accelerate** method, the car object will increase speed up to a limit (and yes, there should be something to better handle an over-the-limit acceleration, but we want to keep this as simple as possible).

The **Driver** class is simplified:

**package tell;**

**class Driver {**

**public static final int INITIAL\_SPEED = 117;**

**final Car car;**

**Driver() {**

**car = new Car(Driver.INITIAL\_SPEED);**

**}**

**int accelerate(int increment) {**

**return car.accelerate(increment);**

**}**

**}**

Rather than changing the **Car**'s speed directly, **Driver** now just tells **Car** to **accelerate** and leaves it up to **Car** to do so. In fact, **Driver** knows nothing about how **Car** accelerates, all it knows is that **Car** can **accelerate.** It knows about **Car**'s behaviour, but not how that behaviour is implemented. This is another important feature of object-oriented programming: hiding the *how* while revealing the *what*.

Taking the new version **Car** for a drive shows again that the maximum speed attained upon acceleration is within limits, and there is nothing an over-enthusiastic driver can do about it (although they still do have a way to wreck the car - find how!).

Note that there will probably have to be a **Car#getSpeed** method. This is not a problem because:

- getting the value of **speed** does not allow code to change it

- the code that gets the value, eg, **Main#main**, can't use the value to decide what **Car** should do; it just prints the value.

**Slightly better version**

Ok, we now have more object-oriented code, but it still has many shortcomings. In particular:

- In **Car,** we have the constant

**public static final int MAX\_SPEED = 130;**

This means that *every* car has the same 130 limit. This is probably too rigid.

- The **Car** object is instantiated in the **Driver** class and has too many hard-wired values. This can make the code too rigid.

We can overcome the first issue by adding another (cascading) constructor:

**package tell.better;**

**class Car {**

**public static final int MAX\_SPEED = 130;**

**private int speed;**

**private final int maxSpeed;**

**Car(int initialSpeed) {**

**this(initialSpeed, Car.MAX\_SPEED);**

**}**

**Car(int initialSpeed, int maxSpeed) {**

**speed = initialSpeed;**

**this.maxSpeed = maxSpeed;**

**}**

**int getSpeed() {**

**return speed;**

**}**

**int accelerate(int increment) {**

**if (speed + increment < maxSpeed) {**

**speed += increment;**

**}**

**return speed;**

**}**

**}**

Now a **Car** can be instantiated either with the default maximum speed or with a custom maximum speed.

For the second issue, **Driver** can be modified to accept a **Car** object via its constructor (also, see the lesson on dependency injection):

**package tell.better;**

**class Driver {**

**final Car car;**

**Driver(Car car) {**

**this.car = car;**

**}**

**int accelerate(int increment) {**

**return car.accelerate(increment);**

**}**

**}**

This could permit testing the **Driver** with a specialized car:

**package tell.better;**

**class SlowCar extends Car {**

**public static final int MAX\_SPEED = 90;**

**SlowCar(int speed) {**

**super(speed, SlowCar.MAX\_SPEED);**

**}**

**}**

This technique becomes important for isolating code to test with unit tests. For now we can illustrate how this would work:

**package tell.better;**

**public class Main {**

**public static void main(String... args) {**

**int increment = 10;**

**Driver driver = new Driver(new Car(111));**

**System.out.println("Initial speed: " + driver.car.getSpeed());**

**System.out.println("Accelerating by: " + increment);**

**driver.accelerate(increment);**

**System.out.println("New speed: " + driver.car.getSpeed());**

**System.out.println("Accelerating by: " + increment);**

**driver.accelerate(increment);**

**System.out.println("New speed: " + driver.car.getSpeed());**

**// slowing things down  
 System.out.println("\nSlow car...");**

**driver = new Driver(new SlowCar(75));**

**System.out.println("Initial speed: " + driver.car.getSpeed());**

**System.out.println("Accelerating by: " + increment);**

**driver.accelerate(increment);**

**System.out.println("New speed: " + driver.car.getSpeed());**

**System.out.println("Accelerating by: " + increment);**

**driver.accelerate(increment);**

**System.out.println("New speed: " + driver.car.getSpeed());**

**}**

**}**

which gives:

Initial speed: 111

Accelerating by: 10

New speed: 121

Accelerating by: 10

New speed: 121

Slow car...

Initial speed: 75

Accelerating by: 10

New speed: 85

Accelerating by: 10

New speed: 85