AN INTRODUCTION TO **OBD**

PROGRAMMING WITH YOUR CAR

PRESENTED BY **SAM JESSO**

WHO AM I?

- Software Engineering student
- My work on **Autobit**
  - A project with the goal of **identifying** and **predicting** vehicle issues
- Not a car guy

WHAT IS **OBD**?

- On-Board Diagnostics (OBDII)
- A standard for accessing vehicle data defined by the **SAE**
- Every vehicle has a **J1962 port**
- We'll focus on software & programming

RESOURCES

- Wikipedia
  - https://en.wikipedia.org/wiki/OBD-II_PIDs
- ELM327 IC Datasheet
  - https://www.elmelectronics.com/wp-content/uploads/2016/07/ELM327DS.pdf
- Society of Automotive Engineers Standards
  - http://standards.sae.org/ground-vehicle-standards/

MODES & PIDs

- To communicate with a vehicle, write 2 bytes onto the bus
  - First byte is the **mode**
  - Second byte is the **PID** (parameter identifier)
- Read vehicle reply into a **buffer** and then parse it
- Some modes don’t require a PID

STANDARD MODES

Mode Byte	Description
`0x01`	Current vehicle data
`0x02`	Vehicle data during freeze frame
`0x03`	Diagnostic trouble codes (DTCs)
`0x04`	Clear MIL (malfunction indicator lamp) and diagnostic trouble codes
`0x05`	Test results for oxygen sensor monitoring
`0x06`	Test results for other component monitoring
`0x07`	Pending DTCs
`0x08`	Control operation of on-board system
`0x09`	Vehicle metadata
`0x0A`	Permanent DTCs

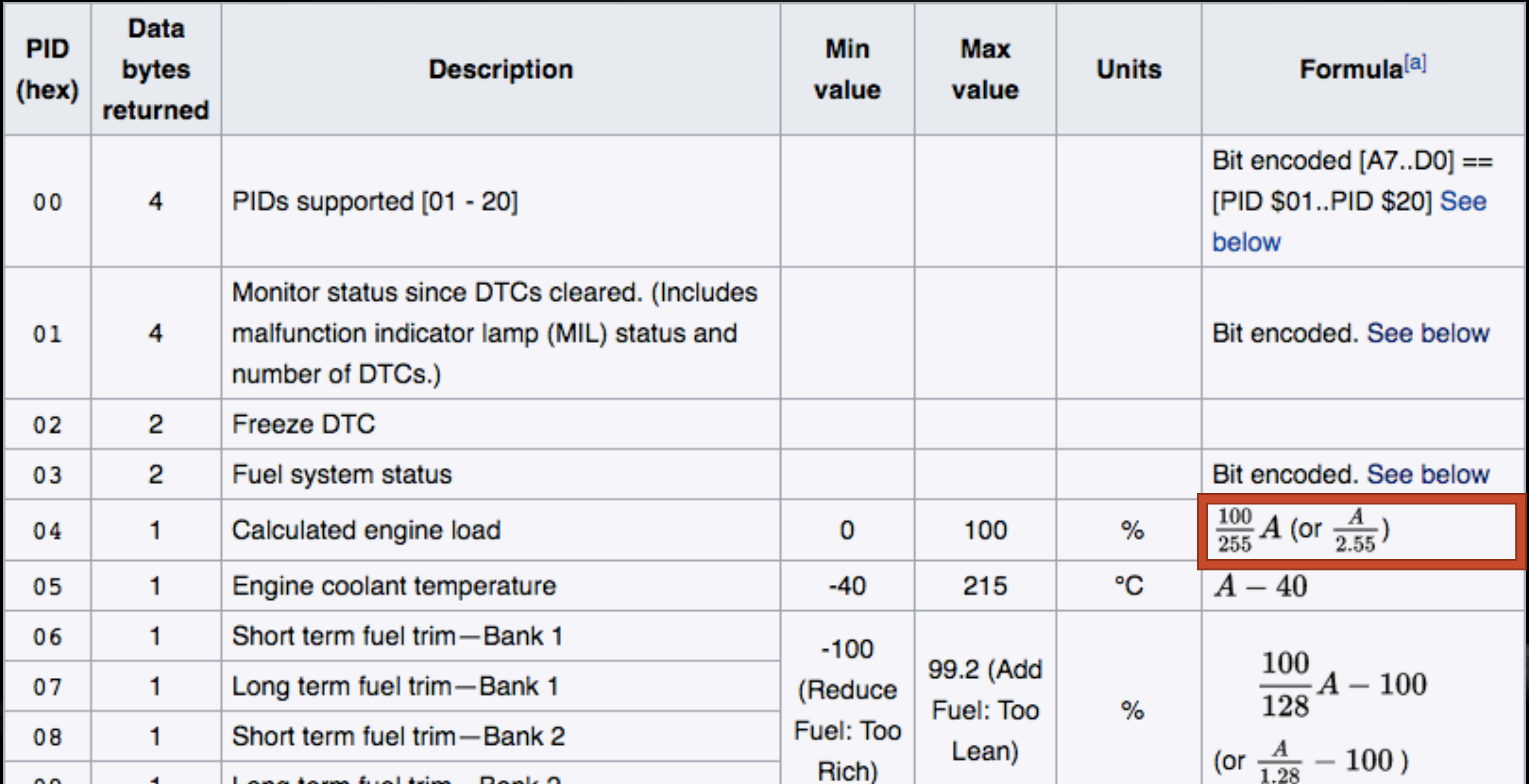
EXAMPLE MODE 1 PIDs

PID Byte	Full Request	Description
`0x00`	`0x0100`	PIDs in range 0x01 – 0x20 supported
`0x0C`	`0x010C`	Engine RPM
`0x0D`	`0x010D`	Vehicle Speed
`0x04`	`0x0104`	Engine Load
`0x0F`	`0x010F`	Intake air temperature
`0x10`	`0x0110`	Mass air flow rate
`0x1F`	`0x011F`	Engine run time (in seconds)
`0x20`	`0x0120`	PIDs in range 0x21 - 0x40 supported

MODE 1 RESPONSE FORMAT

Header	Response Payload (32 bits/4 bytes)			
Width depends on configuration	Byte A	Byte B	Byte C	Byte D

MODE 1 DOCUMENTATION



PID (hex)	Data bytes returned	Description	Min value	Max value	Units	Formula[a]
00	4	PIDs supported [01 - 20]				Bit encoded [A7..D0] == [PID $01..PID $20] See below
01	4	Monitor status since DTCs cleared. (Includes malfunction indicator lamp (MIL) status and number of DTCs.)				Bit encoded. See below
02	2	Freeze DTC				
03	2	Fuel system status				Bit encoded. See below
04	1	Calculated engine load	0	100	%	$\frac{100}{255}A$ (or $\frac{A}{2.55}$)
05	1	Engine coolant temperature	-40	215	°C	$A-40$
06	1	Short term fuel trim—Bank 1	-100 (Reduce Fuel: Too Rich)	99.2 (Add Fuel: Too Lean)	%	$\frac{100}{128}A-100$ (or $\frac{A}{1.28}-100$)
07	1	Long term fuel trim—Bank 1				
08	1	Short term fuel trim—Bank 2				
09	1	Long term fuel trim—Bank 2				

RPM PARSING EXAMPLE

- Documentation example: engine RPM (`0x010C`)

Request:	`010C`
Response:	`41 0C 1A 4E 00 00`
Formula:	$\frac{256A+B}{4}$
Value:	$\frac{256(1A_{16})+4E_{16}}{4} = \frac{256(26)+78}{4} = 1683.5$

DEMO 1

INTERACTING WITH A VEHICLE USING A SERIAL TERMINAL

AUTOBIT OBD LIBRARY OSS

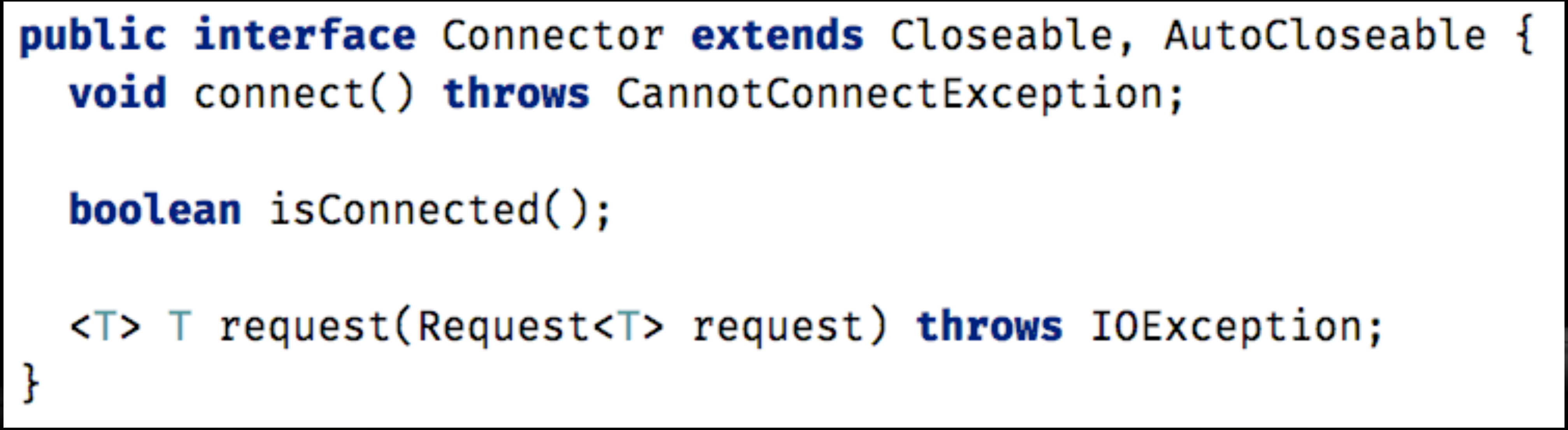
- The Autobit OBD library is **Open Source Software** (OSS)
- Released under the MIT license
- Java library (compatible with Java >= 1.7)
- Currently on version 1.4.0, version 2 is a W.I.P
- Use with any common build system: tested with Gradle and Maven

AUTOBIT OBD LIBRARY FEATURES

- Serial communication class tested with ELM327 IC
- Many built-in parsers available (vehicle speed, engine RPM…)
- Easily add your own parsers

AUTOBIT OBD CONCEPTS

- A `Connector` manages a vehicle connection and uses `Request` objects to send and receive data.
  - **Unfamiliar with Java Generics?** T is the type of the response.



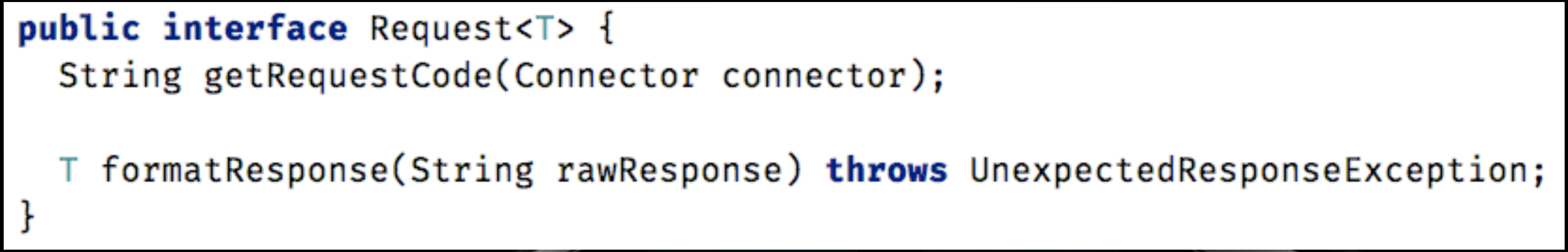
```java
public interface Connector extends Closeable, AutoCloseable {
  void connect() throws CannotConnectException;

  boolean isConnected();

  <T> T request(Request<T> request) throws IOException;
}
```

AUTOBIT OBD CONCEPTS

- A **`Request`** defines the PID to send to the car (**`getRequestCode`** method), and how to interpret (parse) the response (**`formatResponse`** method).

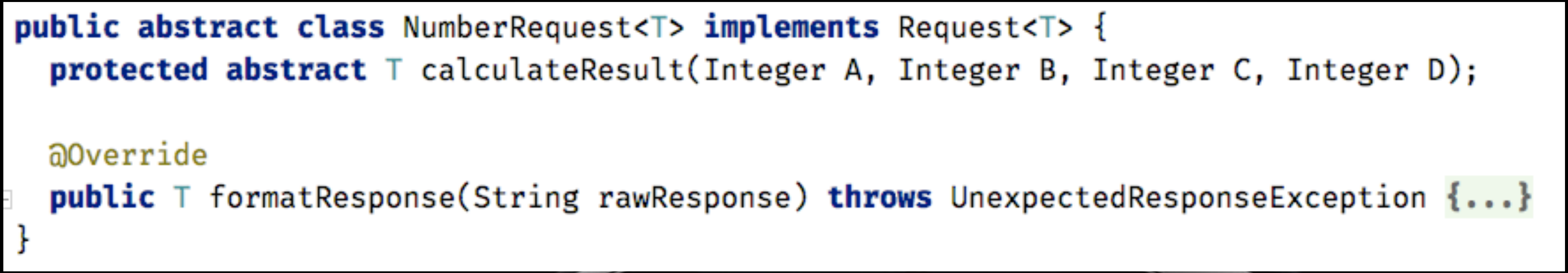


```
public interface Request<T> {
  String getRequestCode(Connector connector);

  T formatResponse(String rawResponse) throws UnexpectedResponseException;
}
```

AUTOBIT OBD CONCEPTS

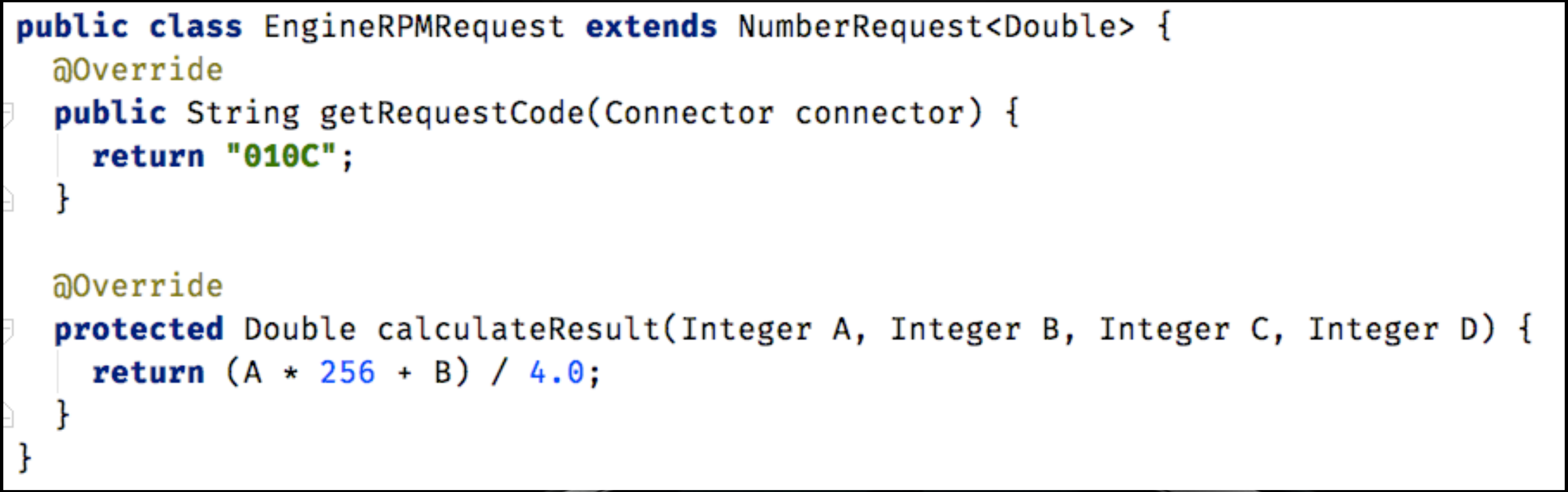
- The `NumberRequest` class is a helper class meant to make parsing responses much easier.



```java
public abstract class NumberRequest<T> implements Request<T> {
  protected abstract T calculateResult(Integer A, Integer B, Integer C, Integer D);

  @Override
  public T formatResponse(String rawResponse) throws UnexpectedResponseException {...}
}
```

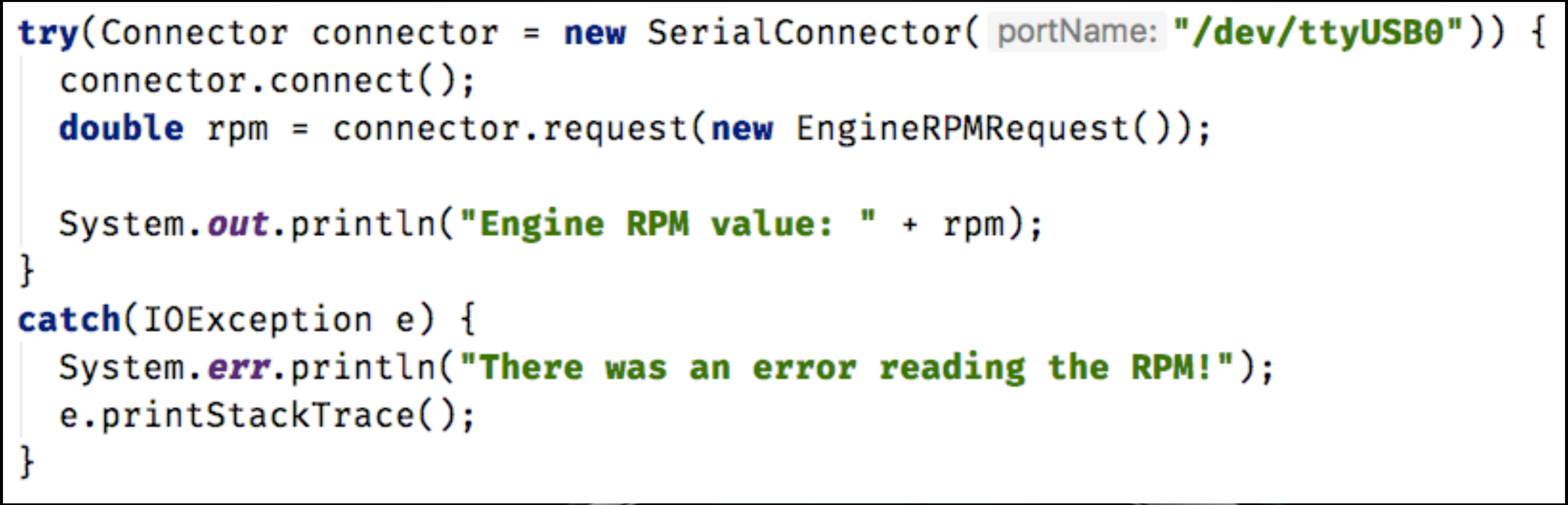
AUTOBIT OBD CONCEPTS



```java
public class EngineRPMRequest extends NumberRequest<Double> {
  @Override
  public String getRequestCode(Connector connector) {
    return "010C";
  }

  @Override
  protected Double calculateResult(Integer A, Integer B, Integer C, Integer D) {
    return (A * 256 + B) / 4.0;
  }
}
```

AUTOBIT OBD EXAMPLE USAGE

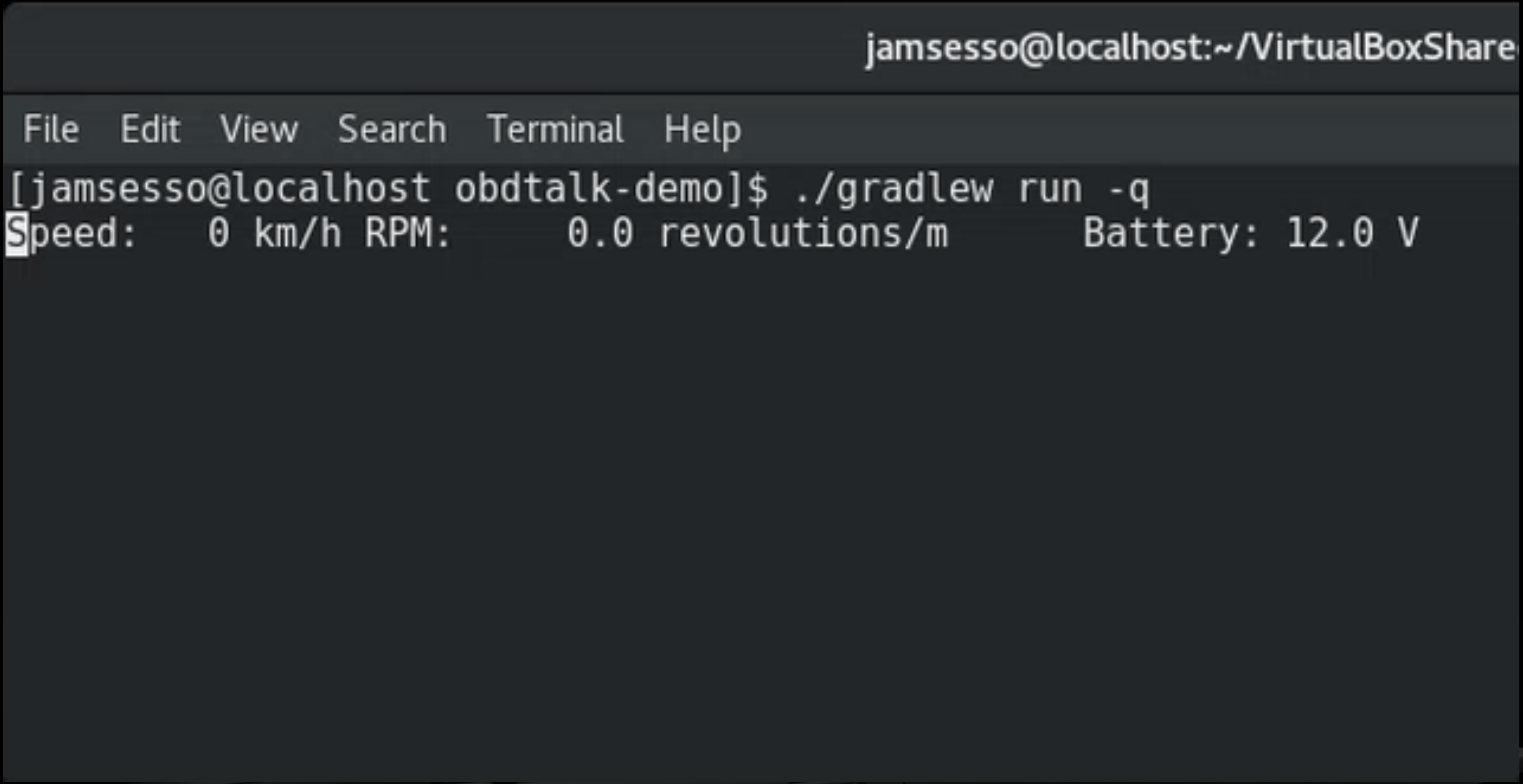


```
try(Connector connector = new SerialConnector( portName: "/dev/ttyUSB0")) {
  connector.connect();
  double rpm = connector.request(new EngineRPMRequest());

  System.out.println("Engine RPM value: " + rpm);
}
catch(IOException e) {
  System.err.println("There was an error reading the RPM!");
  e.printStackTrace();
}
```

DEMO 2

BUILDING A TERMINAL BASED TACHOMETER


jamsesso@localhost:~/VirtualBoxShare
File Edit View Search Terminal Help

```
[jamsesso@localhost obdtalk-demo]$ ./gradlew run -q
Speed:   0 km/h RPM:     0.0 revolutions/m     Battery: 12.0 V
```

HOW TO FOLLOW ALONG

- Clone the OBD starter kit repository: https://bitbucket.org/jamsesso/obd-starter-kit
  - `git clone https://bitbucket.org/jamsesso/obd-starter-kit.git`
- Prerequisite software
  - Java JDK
  - Git

THANK YOU!

GET IN TOUCH: **SAM@AUTOBIT.CA**