Introduction to the Exercises

How to survive the exercise code.

Prerequisites

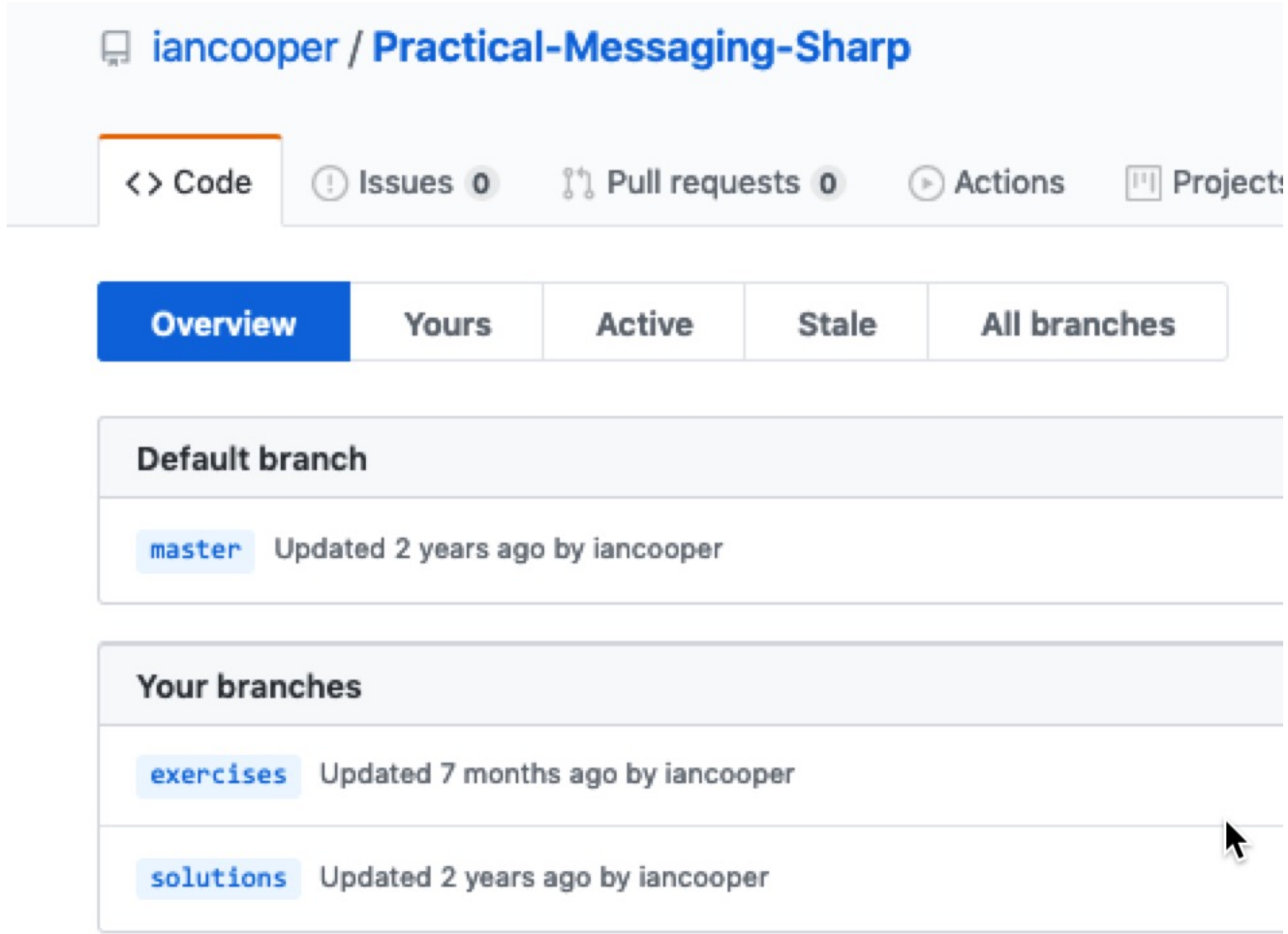
- Rabbit MQ (RMQ)
  - You need RMQ installed as it is the middleware we use for these examples
    - We use RMQ because it does not require cloud accounts and is easy to install and work with
      - Docker
        - We provide a Docker Compose file with the exercises, that if you have Docker installed can be used to run RMQ
        - Navigate to one of the directories containing the docker-compose.yml file
        - Run *docker-compose up –d*
        - Run *docker ps* to confirm that RMQ is running
        - We use a Linux container (if you are running on Windows, adjust settings)
      - Native
        - Install from https://www.rabbitmq.com/download.html

Patterns

- There are videos which introduce key patterns in messaging
  - There are many more patterns catalogued by Hohpe here
- Many of these patterns also have associated exercises
  - They show you how one type of middleware, RabbitMQ, can be used to implement these patterns

Exercises

- The exercises are in GitHub
  - They have supporting, short videos in Dropbox.
  - The videos explain key messaging patterns
  - The exercises let you write code to try some of these patterns yourself
- Solutions and Exercises are different branches
  - The master branch is blank, the solutions and questions are branches
  - You have the solutions as a clue if you need it
  - One way to work is to locally use the exercise branch, have solutions open in your browser
  - That way, if you get stuck, you can look for a hint and switch back


iancooper / Practical-Messaging-Sharp
Code
Issues 0
Pull requests 0
Actions
Projects
Overview
Yours
Active
Stale
All branches
Default branch
master Updated 2 years ago by iancooper
Your branches
exercises Updated 7 months ago by iancooper
solutions Updated 2 years ago by iancooper

Steps

Watch a messaging patterns video

- If there is an associated exercise
  - Do the exercise

Repeat

I am here to help, if you get stuck or have questions.

What Do You Do?

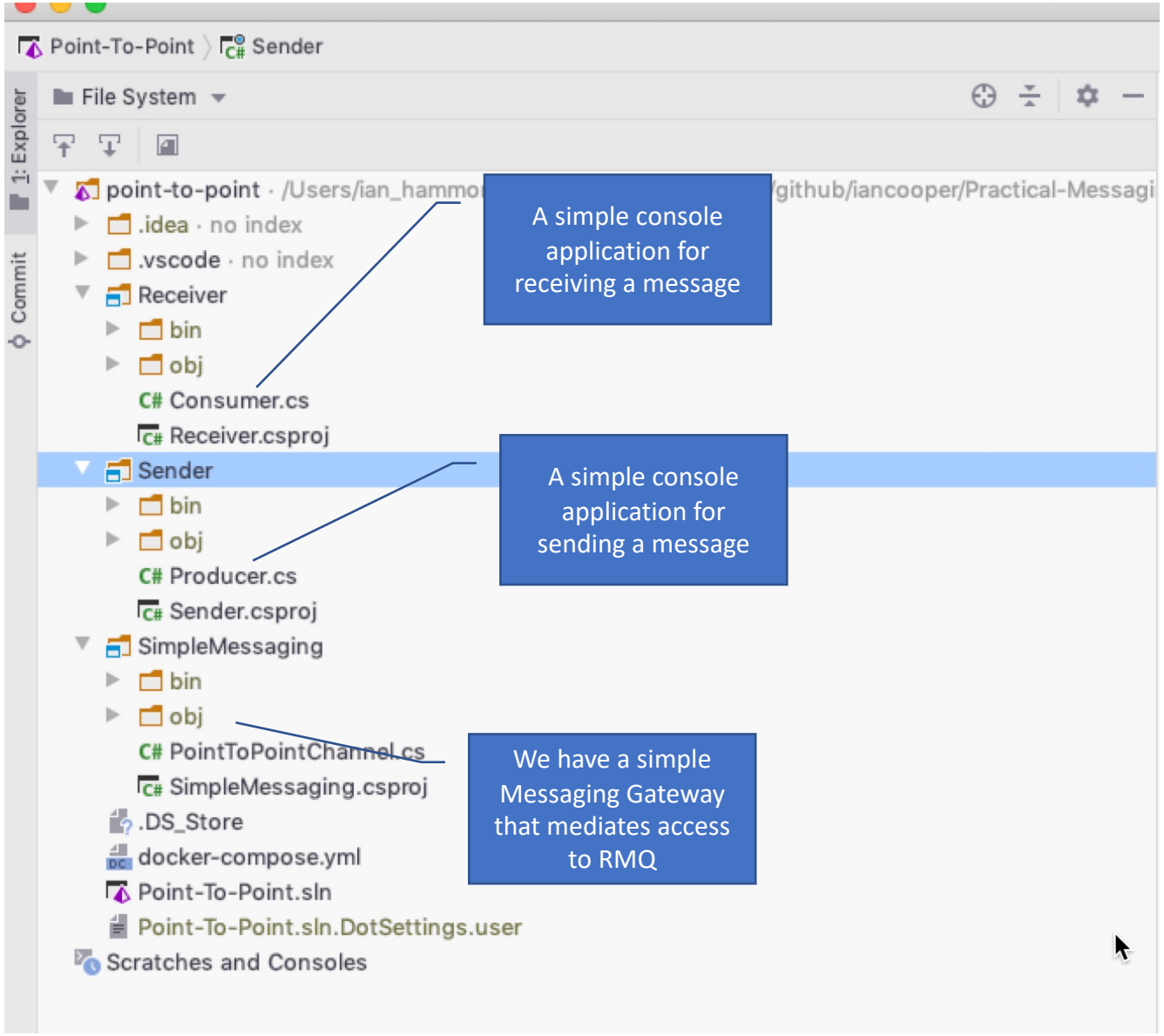
- In each exercise there is code that is missing, replaced by comments
- You need to provide the appropriate code, as indicated by the comments
- Once you have the code building, you run the samples to see how it works
- You may wish to have the RMQ management console open, to observe what is happening on the server

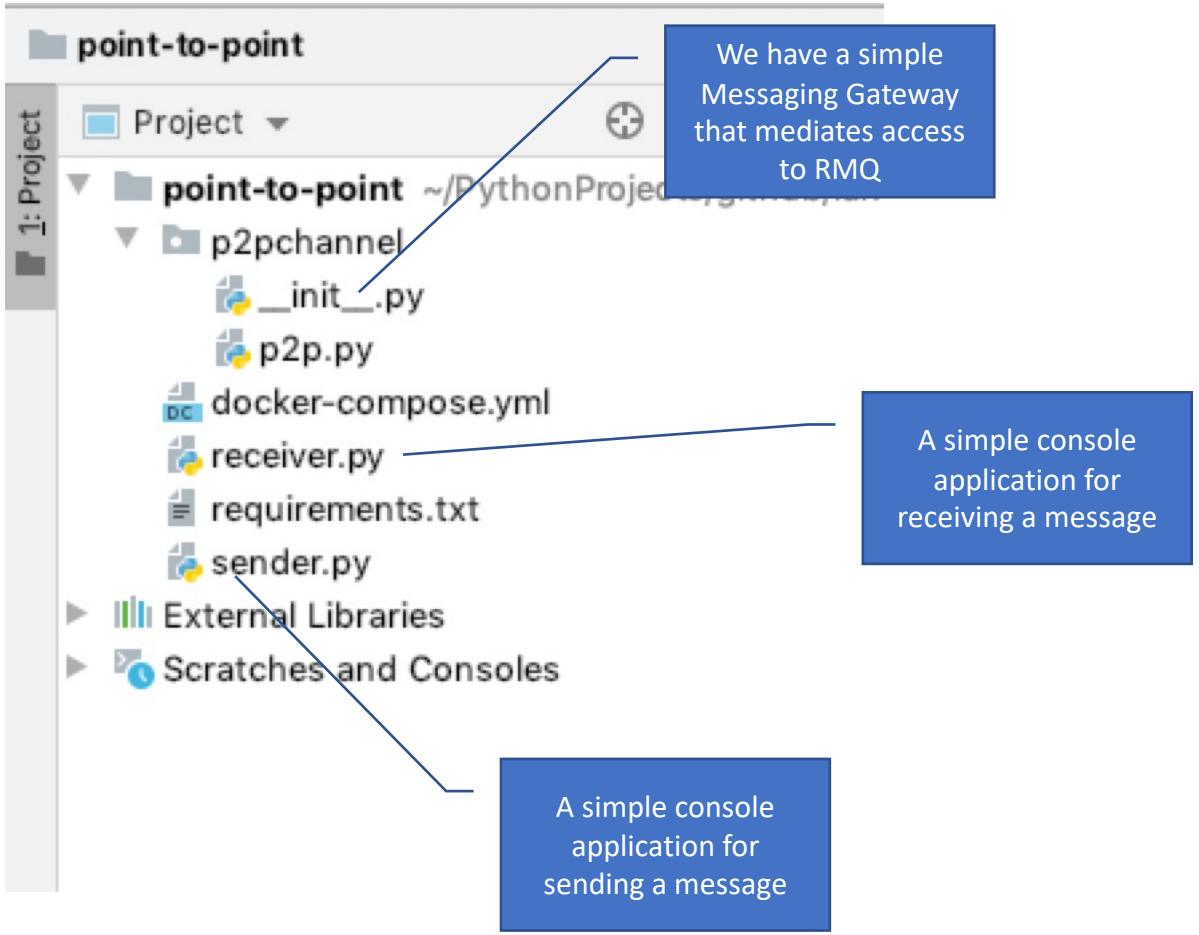
Making the Most of This

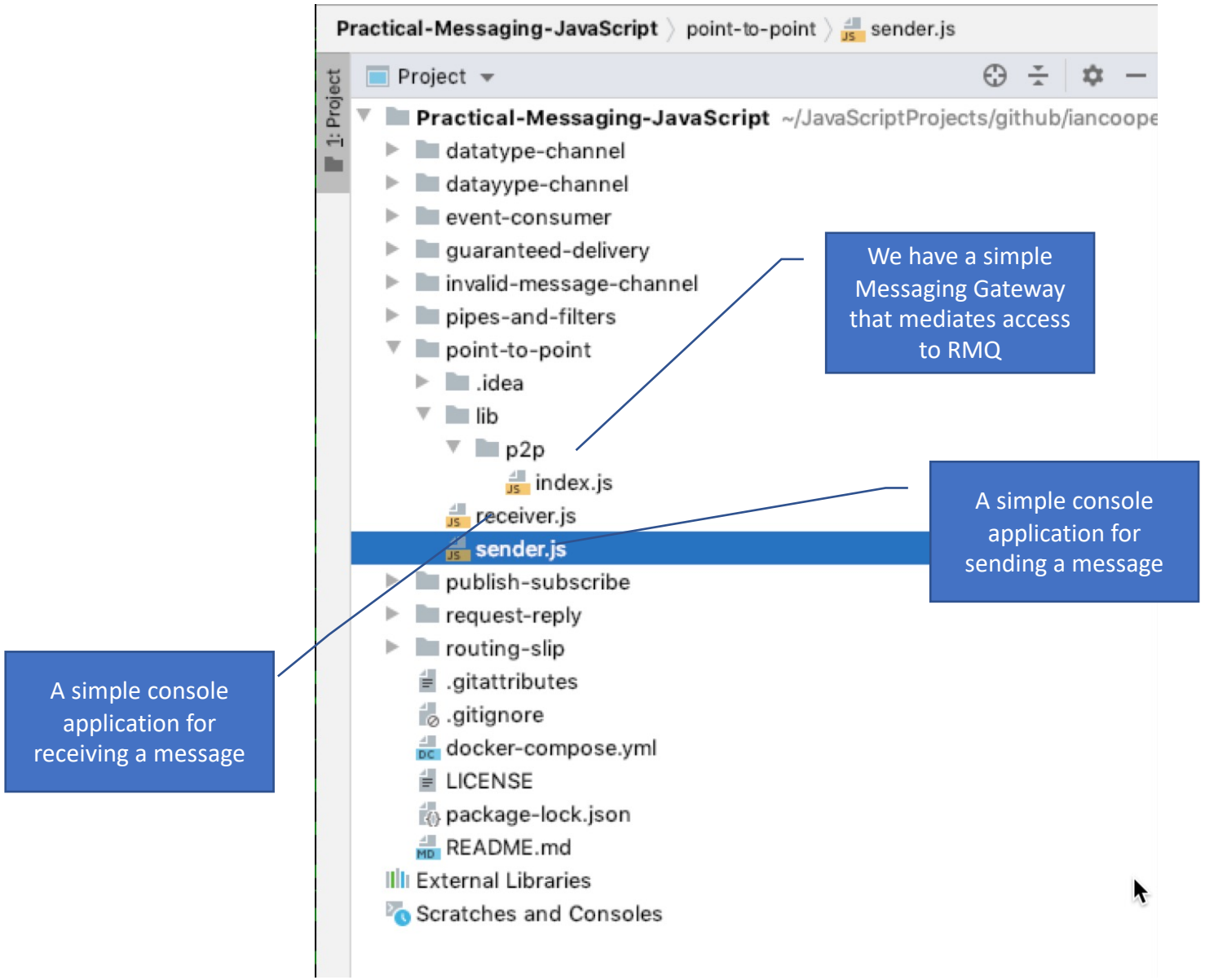
- You could just type the code from Solutions into Exercises
  - You won’t learn as much though
  - It’s better to try and understand what you are being asked to do, consult the RMQ client documentation etc.
  - That way you will leave here with knowledge of how to write this code yourself, and find answers to your problems.

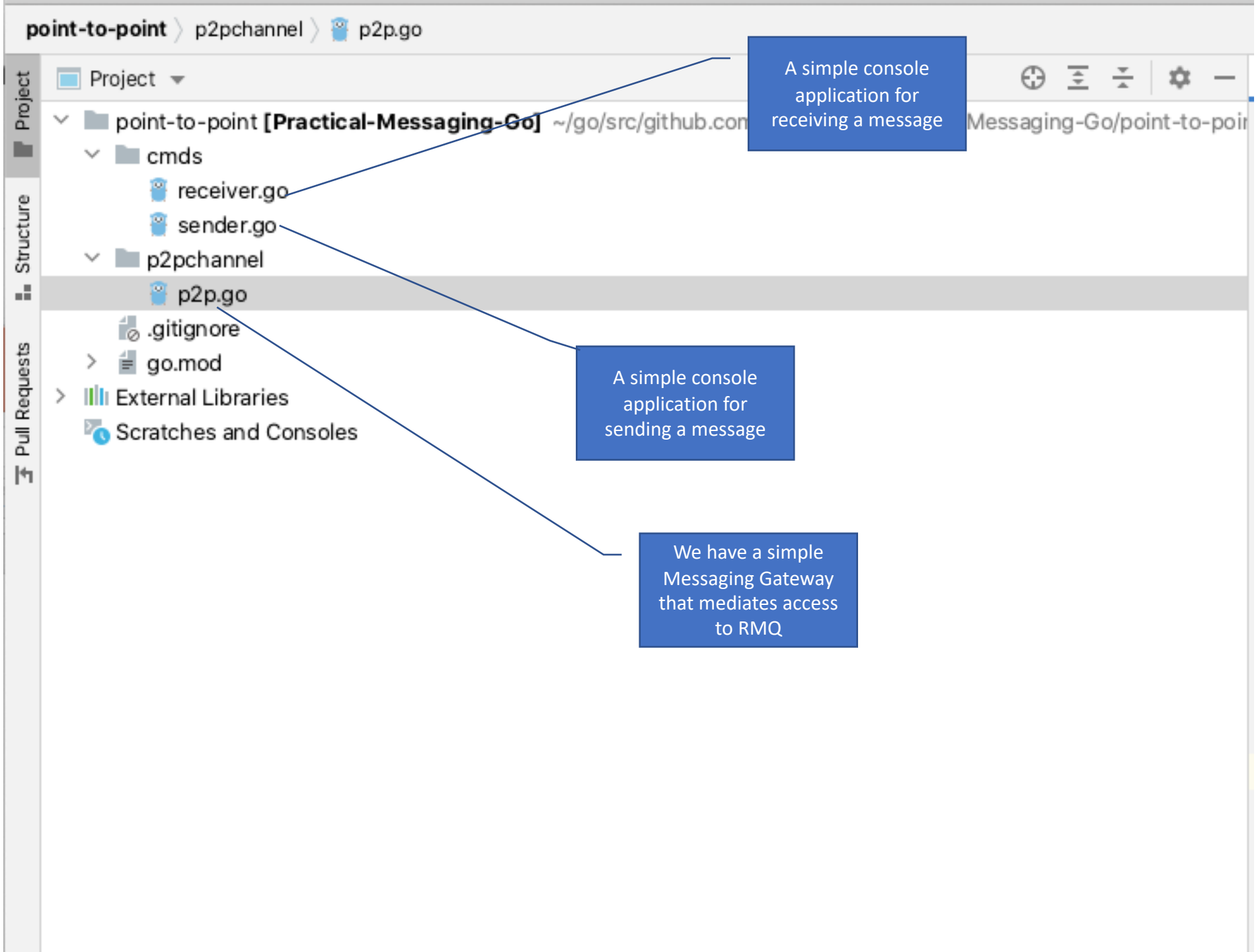
Code Structure

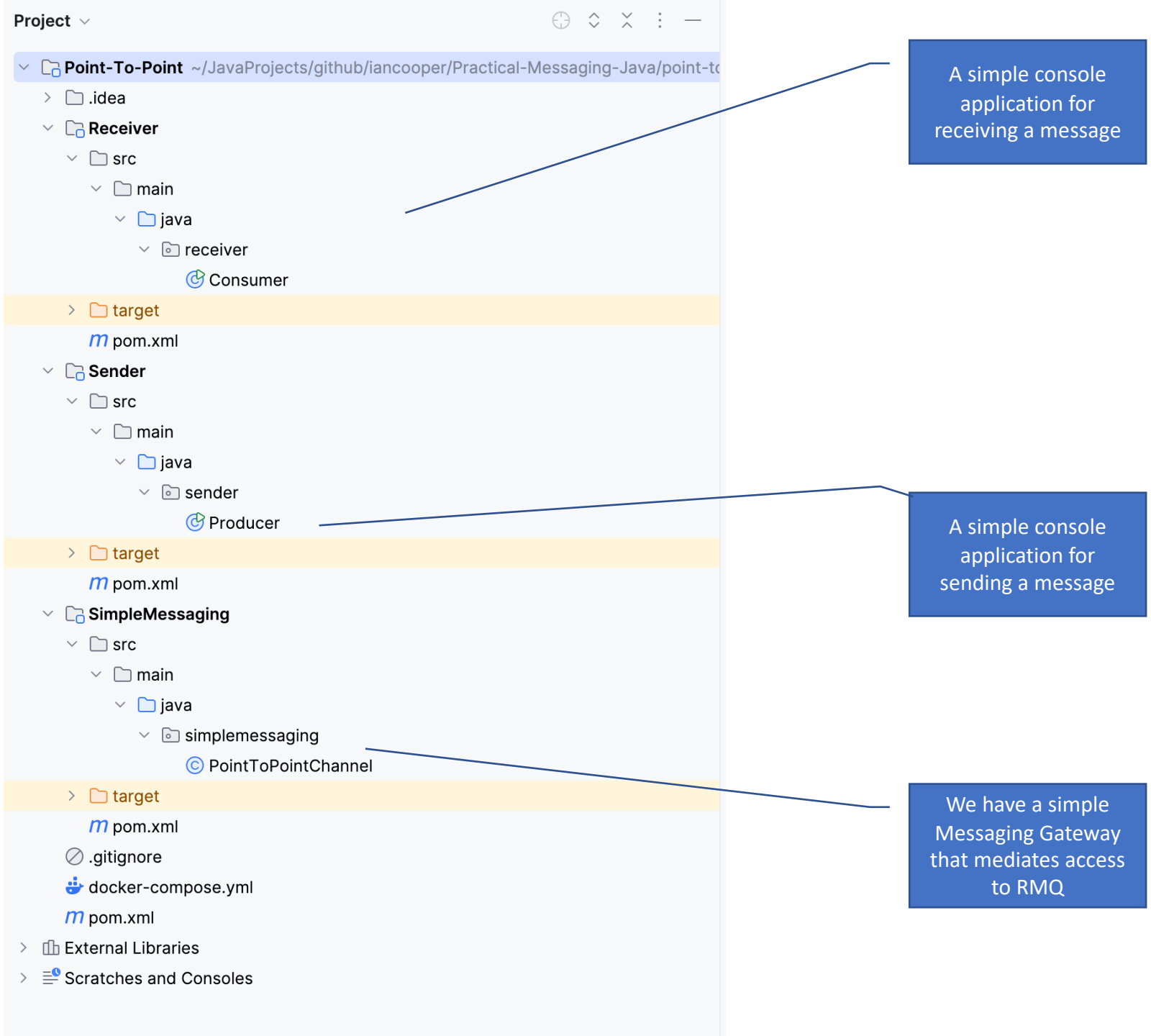
- The Exercises come in five languages:
  - C#, Go, Java, JavaScript and Python
- The Exercises are not production code
  - They are intended to provide just enough code for the exercise
  - They omit most of the error handling production code would need
  - They trade maintainability for focus on what an exercise teaches
- They should help you understand production code
  - Working through these exercises and the workshop material will enable to understand production-grade OSS projects
  - You can see approaches to structure and error handling better there


Point-To-Point
Sender
File System
1: Explorer
Commit
point-to-point · /Users/ian_hammo /github/iancooper/Practical-Messagi
.idea · no index
.vscode · no index
Receiver
bin
obj
Consumer.cs
Receiver.csproj
Sender
bin
obj
Producer.cs
Sender.csproj
SimpleMessaging
bin
obj
PointToPointChannel.cs
SimpleMessaging.csproj
.DS_Store
docker-compose.yml
Point-To-Point.sln
Point-To-Point.sln.DotSettings.user
Scratches and Consoles
A simple console application for receiving a message
A simple console application for sending a message
We have a simple Messaging Gateway that mediates access to RMQ


point-to-point
Project
1: Project
point-to-point ~/PythonProjec
p2pchannel
__init__.py
p2p.py
docker-compose.yml
receiver.py
requirements.txt
sender.py
External Libraries
Scratches and Consoles
We have a simple Messaging Gateway that mediates access to RMQ
A simple console application for receiving a message
A simple console application for sending a message


Practical-Messaging-JavaScript 〉 point-to-point 〉 sender.js
Project
1: Project
Practical-Messaging-JavaScript ~/JavaScriptProjects/github/iancoope
datatype-channel
datayype-channel
event-consumer
guaranteed-delivery
invalid-message-channel
pipes-and-filters
point-to-point
.idea
lib
p2p
index.js
receiver.js
sender.js
publish-subscribe
request-reply
routing-slip
.gitattributes
.gitignore
docker-compose.yml
LICENSE
package-lock.json
README.md
External Libraries
Scratches and Consoles
We have a simple Messaging Gateway that mediates access to RMQ
A simple console application for sending a message
A simple console application for receiving a message


point-to-point
p2pchannel
p2p.go
Project
Structure
Pull Requests
point-to-point [Practical-Messaging-Go]
cmds
receiver.go
sender.go
p2pchannel
p2p.go
.gitignore
go.mod
External Libraries
Scratches and Consoles
A simple console application for receiving a message
A simple console application for sending a message
We have a simple Messaging Gateway that mediates access to RMQ


Project
Point-To-Point ~/JavaProjects/github/iancooper/Practical-Messaging-Java/point-to
.idea
Receiver
src
main
java
receiver
Consumer
target
pom.xml
Sender
src
main
java
sender
Producer
target
pom.xml
SimpleMessaging
src
main
java
simplemessaging
PointToPointChannel
target
pom.xml
.gitignore
docker-compose.yml
pom.xml
External Libraries
Scratches and Consoles
A simple console application for receiving a message
A simple console application for sending a message
We have a simple Messaging Gateway that mediates access to RMQ