# PA1489 Exercises

Mikael Svahnberg[*]

2024-04-19

## 1 Introduction

This document summarises the *non-graded* exercises introduced in each lecture.

## 2 Get Started with Git

### 2.1 Register an account

- Register an account at some git server:
  - `https://github.com/signup` or `https://education.github.com/pack`
  - `https://gitlab.com/users/sign_up`
  - `https://www.atlassian.com/software/bitbucket/bundle`
  - `https://codeberg.org/`
- Github is still very popular for open source projects
  - In some trouble for how they may use the code you upload ther
- Many migrated over to Gitlab when Microsoft bought Github
- Atlassian and BitBucket are very well integrated with the rest of their produxcts
  - Used to have very generous offers for students and universities (unclear status these days)
- Codeberg.org is specifically focussed on open source projects.

### 2.2 Create and Clone a Repository

- Easiest to start in the web interface
- Name the project to something creative, e.g. `gitExample`
- When you are done, there should be a link, e.g. under `<> Code` that can be used to clone the project.
  - Example: `git clone https://codeberg.org/mickesv/gitex.git`
  - This will set up `remote/origin` for you.

[*]Mikael.Svahnberg@bth.se

## 2.3 Create som git history

1. Create some files
2. Add them to the stash and commit
3. Change one of the files; stash and commit again
4. Repeat a couple of times.
5. Create a branch
6. Create some files, stash and commit.
7. Edit some of your first files and commit.
8. Check the log.
9. Check status.
10. Push to the server
11. Check the status.

## 2.4 Fork a colleagues repository

1. Find the account of a colleague (on the same server)
2. Pick a repository and fork it (for example the example account that you just created)
3. Clone it to your computer and create some more git history
4. When you have pushed everything to your fork, create a `pull request` in your colleagues repository (via the web interface)

## 2.5 Handle a Pull Request

When your colleague have created a pull request to your repo, handle it.

- Inspect every commit to see what has been changed
- Can it be merged automatically? This should be indicated somewhere.
- Create a merge commit.

Create some more commits in your respective forks.

- Create a new pull request.
- This time, *deny* the pull request.

### 2.6 More participants in the same project

- Divide into groups of around 5 people
- Pick a colleagues repository
- Enter `Settings/Collaborators` and add all of you to the same project
- Clone the repo

Now you are only allowed to work in a specific file `charlie-foxtrot.txt`

- You may add new text
- You may edit the existing text
- You may insert text; between two lines, and in the middle of a line
- You may remove text

Commit regularly (max 2-3 changes per commit) Push after every commit.

- You may need to do a `fetch/merge` in order to be allowed to do a `push`

**Handle the merge conflicts**
Discuss in small groups: What can you do to get fewer conflicts?

## 3 Testing and Debugging

### 3.1 Introduction to SorterTool

- `https://codeberg.org/mickesv/SorterTool.git`
- *SorterTool* implements and tests some sorting algorithms
- There are many ways to sort lists, with different timing characteristica
  - Seach for “Sorting out Sorting” for an old movie (30 min) from 1980 that generations of students have been forced to watch.
  - “Big-O” notation describes the complexity of algorithms.
- Examples:
  - Insertion Sort: `O(n²)` Move elements out of the way and insert the next element in the right place.
  - Selection Sort: `O(n²)` Find the smallest value and insert it at the top; continue with element 2.
  - Merge Sort: `O(n*log n)` Make sure every pair is ordered, combine pair `n` with `n+1`; repeat.
  - QuickSort: `O(n*log n)`
    1. Pick an element in the middle.
    2. Make sure that all elements to the left are smaller, and all to the right are bigger
    3. Repeat for left and right.
  - Bubble Sort: `O(n²)` Compare every element with all other elements and swap places unless they are already ordered.

## 3.2 Get Started with the Exercise

1. Clone SorterTool to your computer: `https://codeberg.org/mickesv/SorterTool.git`
2. Open the project in your IDE. Study the following files:
   - `src/Main.java` to quickly be able to start he program.
   - `src/Sorter.java` implements the different sorting algorithms.
   - `Tests/SorterTest.java` tests `Sorter`.
3. Run all tests. What happens?
4. Specifically, run the test for `bubbleSort()`.

## 3.3 Get more Information

**Read the message**
No, seriously. **Read the message!**

- What might `AssertionFailedError` mean?
- In which file, on which line does it occur?
- What's the contents of that line?
  - Is this sufficient to understand what went wrong?
  - How can you find more information?

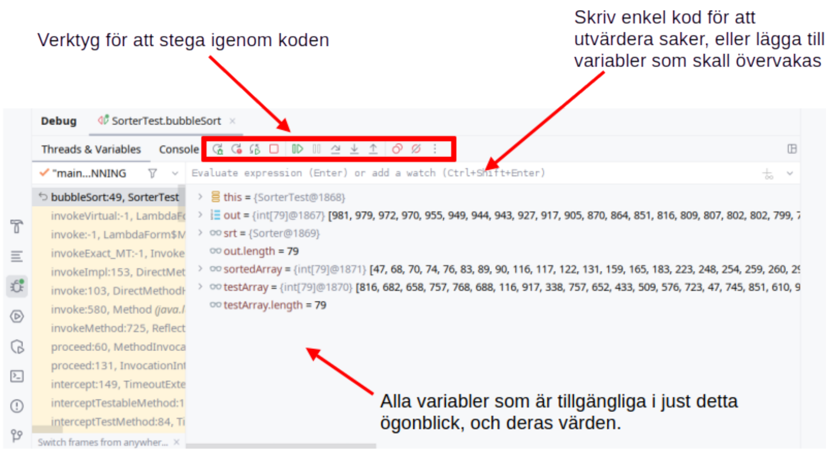
## 3.4 Use the Debugger

1. In the file `SorterTest.java`, click on the line number `49`; a small stop-sign should appear
2. Now run the test for `bubblesort()` in "debug"-mode.

Now, what can you see

- In the code window?
- In the debug window?

## 3.5 The debug Window


Verktyg för att stega igenom koden
Skriv enkel kod för att utvärdera saker, eller lägga till variabler som skall övervakas
Debug
SorterTest.bubbleSort
Threads & Variables
Console
"main...NNING
Evaluate expression (Enter) or add a watch (Ctrl+Shift+Enter)
Switch frames from anywher...
Alla variabler som är tillgängliga i just detta ögonblick, och deras värden.

```
bubbleSort:49, SorterTest
invokeVirtual:-1, LambdaF
invoke:-1, LambdaForm$M
invokeExact_MT:-1, Invoke
invokeImpl:153, DirectMet
invoke:103, DirectMethodH
invoke:580, Method (java.l
invokeMethod:725, Reflect
proceed:60, MethodInvoca
proceed:131, InvocationInt
intercept:149, TimeoutExte
interceptTestableMethod:1
interceptTestMethod:84, Ti
```

```
> this = {SorterTest@1868}
> out = {int[79]@1867} [981, 979, 972, 970, 955, 949, 944, 943, 927, 917, 905, 870, 864, 851, 816, 809, 807, 802, 802, 799, 7
> srt = {Sorter@1869}
  out.length = 79
> sortedArray = {int[79]@1871} [47, 68, 70, 74, 76, 83, 89, 90, 116, 117, 122, 131, 159, 165, 183, 223, 248, 254, 259, 260, 2
> testArray = {int[79]@1870} [816, 682, 658, 757, 768, 688, 116, 917, 338, 757, 652, 433, 509, 576, 723, 47, 745, 851, 610, 9
  testArray.length = 79
```

- This shows why the test failed, but not what in the code that caused the error.
- *Clue:* Look at the array `out` and all the values there. Can you find any pattern?

## 3.6 Step through the code

1. Move the break point from line `49` to line `47`.
2. Run the test for `bubbleSort()` again in Debug-mode (Restart the test when asked).

Important tools to step through the code

**Continue/Resume** keep runing until the next breakpoint

**Step over** Run the next instruction and stop when you get back

**Step in** Follow into the next instruction (often into a method)

**Step out** Finish the method you are in now, and stop when you get back.

Use **Step in** to enter the call to `srt.bubbleSort()` .

- Note that the variable window changes. WHat do you see there now. Why?

Step a couple of rounds through the inner for-loop.

- The lines with `for` and `if` are being run in every iteration.
- When does the code enter into the `if` statement?

    - At what values on `out[outer]` and `out[inner]` ?

  Is this correct? (It's not. What should happen?)
  Correct the error and continue debugging.

- As you may notice, the change does not seem to work.
- You have to *restart* the test in order for the change to take effect.

### 3.7 Rerun all tests

- Once you have fixed the error, all tests should be green.
- Why did both `sort()` and `bubbleSort()` turn green? You have only fixed `bubbleSort()` . . .

### 3.8 Continue playing on your own

- Can you use the debugger to understand how the sorting algorithms work?
- Write your own sorting algorithm and test it.
- Right now there is a singe test for each method. Can you have more tests for each method?
    - Which other tests might be relevant?

# 4 Documentation

## 4.1 Introduction to JavaPonies

- *Desktop Ponies* is an ancient mono-application (similar to Visual Basic) that enables My Little Ponies to run around on screen.
- *Java Ponies* is "my" version of this program.
    - `https://codeberg.org/mickesv/JavaPonies.git`
    - Warning:
        * It is far from complete.
        * It is not fully documented.
        * It is slow and (I'm sure) buggy.
    - However:
        * PONIES!

## 4.2 Get Started with the Exercise

1. Clone the project: `https://codeberg.org/mickesv/JavaPonies.git`
2. Open the project in your IDE and study the program to understand what it does:
   - `src/JavaPonies.java` start the program.
   - `src/model/Pony.java` Implements a class which is instantiated once for each pony.
   - `src/model/PonyBehaviour.java` represents a behaviour that a pony may have.
   - `src/view/PonyWindow.java` Handles display and update of a pony which is active on screen.

## 4.3 Create the Documentation

- From a terminal: `javadoc src/*.java src/model/*.java src/view/*.java -d doc`
- From IntelliJ: `Tools/Generate JavaDoc/`, make sure the documentation ends up in the `doc` directory. (The result opens in your web browser)

Inspec the documentation:

- Compare to what you see in the java-files
- What is included? What isn't included?
- When you created the documentation you got a lot of warnings. Why?

## 4.4 Update JavaDoc

The file `src/model/PonyBehaviour.java` lacks any JavaDoc-comments.

1. Write these comments so that you no longer get any javadoc warnings from `PonyBehaviour.java`.
2. Did this increase the usability of the documentation? Why/why not?
3. Did this increase the readability of the code? Why/why not?

## 4.5 Create an Issue

Please note:

- If you have an account at codeberg.org, you may create an issue directly towards the JavaPonies project.
- If you do not have, or do not want an an account, write your issue in a text file.

To do:

1. Find something to address in the project. It may be
   - a bug, i.e. something which does not work as expected.
   - an enhancement, i.e. a new feature.
2. Write your issue. Make sure it contains:
   - A short but descriptive title
   - A describing text
   - Steps to trigger the bug, or steps to where the enhancement might be applicable
   - Expected result
   - Actual result
   - Other information (if relevant)

### 4.6 Improve a method

1. Pick a method or attribute that you think is unclear and conduct a *Refactoring* to clarify
   - IntelliJ has a whole menu for Refactoring; explore it to see what's there and how it works.

Think about

- What support does your IDE have for refactoring?
- Is this support helpful?
- How do you know what will be a good refactoring?

## 5 Implementation in Java

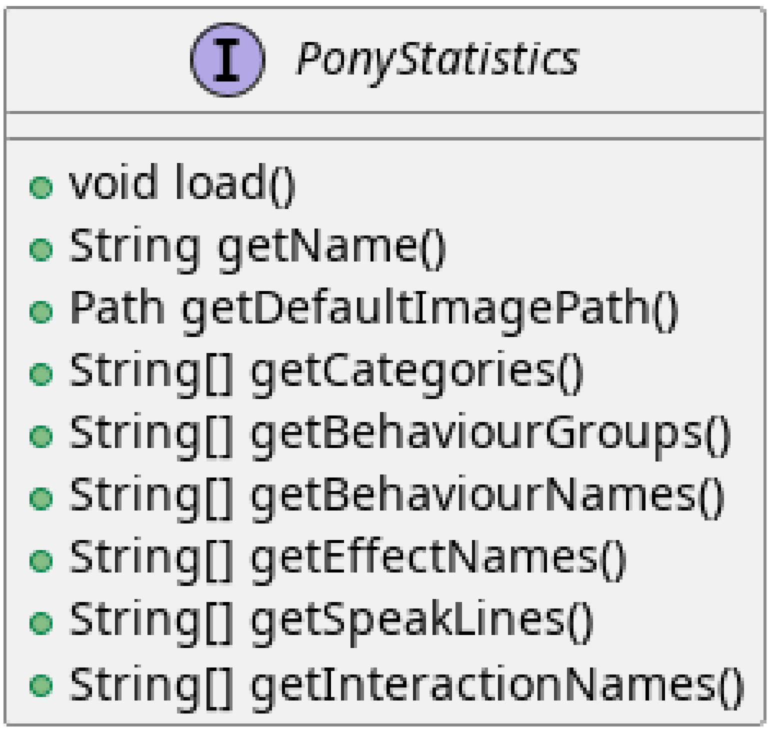
### 5.1 More Ponies

- We continue with JavaPonies
- The MLP fans have requested an extended user interface where you can find out more data about each pony.
- We will also get started with the implementation of *Interactions*, i.e. that a pony changes behaviour because they are close to some other pony.

### 5.2 MLP Data

1. Write an interface `src/model/PonyStatistics.java` as described below.
2. Make sure `model.Pony` implements this interface. Please note that
   - some methods already exist but may need to be extended
   - some methods may be called several times; especially `load()` will need to consider this.

- some classes may need to be created, e.g. to contain an Interaction.
- Some of the get-methods may need to iterate over an `ArrayList<>) of e.g. =Behaviours` to get their names and store in a String array.
- We do not have any way to call these methods yet. *Please do write unit tests instead.*



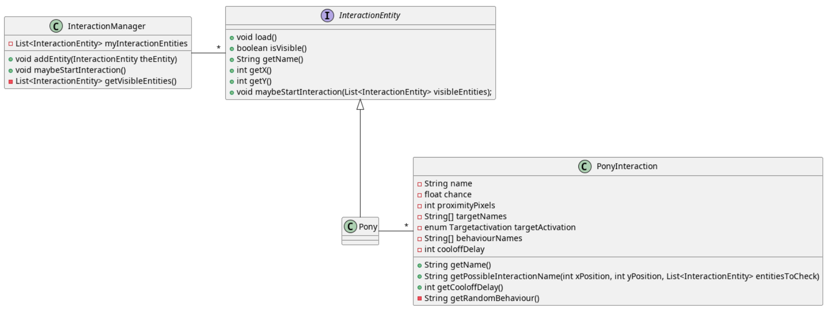
I
PonyStatistics
void load()
String getName()
Path getDefaultImagePath()
String[] getCategories()
String[] getBehaviourGroups()
String[] getBehaviourNames()
String[] getEffectNames()
String[] getSpeakLines()
String[] getInteractionNames()

## 5.3 Interactions

- For a pony to interact with another pony, they need to know that they are close to each other.
- A single pony cannot know this. What other alternatives do we have?
  - `view.PonyWindow` knows where a single pony is (by asking it), but not the other.
  - `view.MainWindow`, once it has created a `PonyCard` for each `Pony`, does not even know what Ponies exist.
  - `model.PonyContainer` might know, but this will give it two areas of responsibility: Maintaining the collection of ponies *and* handle interactions.

$\sum$

1. We need to create a new class `model.InteractionManager`, containing a collection of `InteractionEntity`
    - The method `maybeStartInteraction()` needs to be called regularly.
2. We need to create an interface `model.InteractionEntity` that the class `Pony` implements.
    - Especially important is the method `maybeStartInteration()`
3. We need a class `model.PonyInteraction` that represent a specific possible interaction.
4. The class `JavaPonies` need to "start" an `InteractionManager` object.
5. The class `view.PonyWindow` needs to collaborate with `model.Pony` so that `model.Pony` knows whether it is visible on screen or not.


InteractionManager
List<InteractionEntity> myInteractionEntities
void addEntity(InteractionEntity theEntity)
void maybeStartInteraction()
List<InteractionEntity> getVisibleEntities()
*
InteractionEntity
void load()
boolean isVisible()
String getName()
int getX()
int getY()
void maybeStartInteraction(List<InteractionEntity> visibleEntities);
Pony
*
PonyInteraction
String name
float chance
int proximityPixels
String[] targetNames
enum Targetactivation targetActivation
String[] behaviourNames
int cooloffDelay
String getName()
String getPossibleInteractionName(int xPosition, int yPosition, List<InteractionEntity> entitiesToCheck)
int getCooloffDelay()
String getRandomBehaviour()

# 6 Graphical User Interfaces in Java

## 6.1 More Pony Statistics

- In the JavaPonies project, there is a branch `PonyStatistics` where the interface from earlier exercises is impemented.
- Check out JavaPonies in a new place (in case you wish to save your implementation) and switch branch:
    - `git clone https://codeberg.org/mickesv/JavaPonies.git`
    - `cd JavaPonies && git checkout PonyStatistics`
- Have a look at what has been changed: `git diff origin/main`

## 6.2 A New Main

1. Write a new class `JavaPonyStatistics extends JavaPonies` (inheriting from JavaPonies means you can save a lot of the startup from JavaPonies)
2. Write a new `main()` - function in `JavaPonyStatistics` :

```
public static void main(String[] args) {
  JavaPonyStatistics ps = new JavaPonyStatistics();
  ps.printStatistics();
}
```

3. Implement the method `JavaPonyStatistics.printStatistics()` so that it:
   - iterates over all Ponies (get them with `myPonies.findAll()` ) and
   - Prints Categories, Behaviour Groups, Behaviours, Effects, Interactions, and Speaking Lines:

   Printing Statistics for Apple Bloom
   Categories:
   Behaviour Groups:
   Behaviours: stand, walk, $\text{follow}_{\text{aj}}$, $\text{spin}_{\text{merightround}}$, workout, aww, CMC, dance
   Effects:
   Interactions:
   Speaking lines:
   - CUTIE MARK CRUSADER DESKTOP PONIES!!!
   - Did I get my cutie mark? Did I? Did I!?
   - Scoot-Scootalooo!
   - Aww!
   - Aren't you gonna stay for brunch?
   - But I want it now!
   - I am a big pony!
   - I'm not a baby, I can take care of myself!
   - Likely story.
   - Not the cupcakes!
   - Some pony needs to put this thing out of its misery.
   - You're not using power tools, are you?
   - Scootaloo! Scoot-Scootaloo!
   - Trust me.
   - What a thing to say!

## 6.3 Just One Pony

Note the signature for main: `public static void main(String [] args)`

**public** so that it is reachable from outside the class

**static** so that you to not need to first create an object

**void** nothing is returned

**main** so that the java runtime knows which method to look for

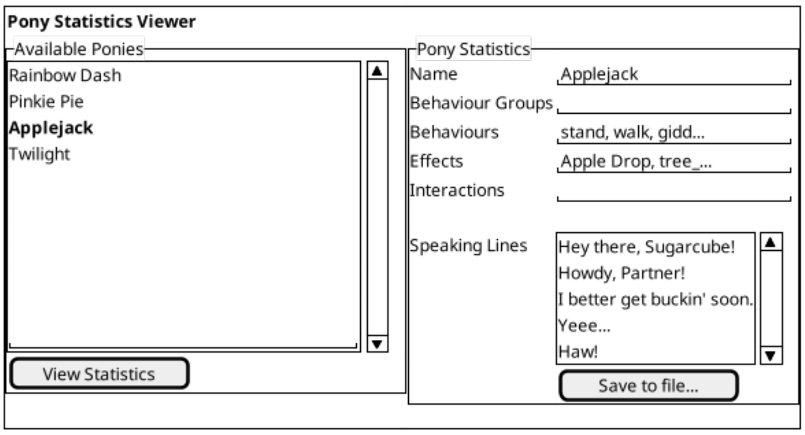
**String [] args** An array containing all the command line parameters given when starting the program.

Time to add some interactivity:

1. If `(0 == args.length)`, list statistics for all ponies (as before)
2. Otherwise, find all ponies whose name contains `arg[0]`
   - You will want to make sure that everything is written in either capitals or commons: `String::toLowerCase()`
   - It is enough to know whether the ponys name *contains* the string. Use `String::contains()`.
   - `pony.getName().toLowerCase().contains(arg[0].toLowerCase())`

### 6.4 A Graphical User Interface

1. Write a new class `view/PonyStatisticsViewer` that creates a JFrame as below.
2. Write a new function to fill the list with the names of all available ponies
3. Write code so that one ce a pony is selected and you press the button "view statistics", the statistics is shown to the rigjt.
4. Wait with the "Save to file…" - button.



### 6.5 Save to File…

Now it is time to implement "Save to file…":

1. Pressing the button should open a `javax.swing.JFileChooser`
2. Statistics about the selected pony should be written to the specified file.
3. Check that the file has the rihgt contents by opening it (or vieweing it in your terminal).

1. MWE for Save to File

```
import javax.swing.*;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import javax.swing.filechooser.FileSystemView;

public class FiCH {

public static void main(String [] args) {
  JFrame f = new JFrame("FiCH");
  f.setSize(500, 500);
  f.setVisible(true);
  JLabel l = new JLabel("no file selected");

  JButton button1 = new JButton("save");
  button1.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent evt) {
        JFileChooser j = new JFileChooser(FileSystemView.getFileSystemView().getHomeDi
        int result = j.showSaveDialog(f);
        if (result == JFileChooser.APPROVE_OPTION) {
          l.setText(j.getSelectedFile().getAbsolutePath());
        } else {
          l.setText("the user cancelled the operation");
        }
      } });

  JPanel p = new JPanel();
  p.add(button1);
  p.add(l);
  f.add(p);
}
}
```

## 7 Virtual Machines and Containers

### 7.1 Get Started with some Tutorials

1. Docker `https://docs.docker.com/get-started/`
2. Docker with node.js `https://docs.docker.com/language/nodejs/`

### 7.2 About the Project: QuoteFinder

- Download the project: `https://github.com/mickesv/ProvisioningDeployment.git`
  - Is actually part of a course on *Applied Cloud Computing and Big Data*
  - Calm down, we will not use everything in this course.
- *QuoteFinder* looks for quotes in texts.

- Three versions:
    - Version 1, looks for a whole string
    - Version 2 && 3 looks fot ther words close to each other.

Version 1 Simpler and can be run almost stand-alone.

    - Version 2 && 3 consists of a couple of collaborating microservices.
- Written in the interpreted language JavaScript / node.js `https://nodejs.org/`
- Express web app `http://expressjs.com/`
- Also uses socket.io `https://socket.io/`
- Connects to a MongoDB-database `https://www.mongodb.com/`
- Exposes three web pages: `/` , `/add` , and `/list` .

```
+----------------------+                    +--------------------+---------------+
| Web Client           |<------------------>| QuoteFinder        | Jade/Pug      |
+----------------------+   Socket.io        |                    | Page Rendering|
                           and              +--------------------+---------------+
                           HTTP             | simpleTextManager  |
                                            |                    |
                                            +---------+----------+
                                                      |
                                            +---------+----------+
                                            | MongoDB Database   |
                                            +--------------------+
```

## 7.3 Check the code: `Containers/Version1/QFStandalone/src`

- Read `index.js`
    - How is a “route” created?
    - What happens when you e.g. request the page `/add` ? What function is called?
- **You do not need to understand**
    - `res.render()` uses *Jade/Pug* to create a webpage: `https://pugjs.org/`
        - This is a common procedure, it is “easier” to create a dynamic web page from some templating system.
    - *Promises* is a way to link thinks that happens asynchronously so that they still happen in a given order.
        - A clue is when you find a call to `.then()` in the code.
- Read `simpleTextManager.js`
    - What does the class do? What methods and areas of responsibilities does the class have?
    - Look closer at the method `addText()`
        - What does it do?
        - Why do you think it saves the texts in this way?

### 7.4 Build an Image

- Go to the directory where the `Dockerfile` is, i.e. `Containers/Version1/QFStandalone/`
- Look at the `Dockerfile`, do you undertand how it is constructed and what will happen?
- Build an image: `docker build -t qfstandalone .`
  - What happens?
  - Note how it constructs layer by layer.
- Check that the image was actually created: `docker image ls`
  - What other images do you have? Why do you think they are there?

### 7.5 Start the application: podman/docker

1. The application uses `MongoDB` so let's fetch it: `docker pull mongo`
2. We also need a network to enanle qfstandalone to communicate with he database:
   - `docker network create qfstandalone-net`
3. Start the database: `docker run -d --network qfstandalone-net --network-alias textstore --name textstore mongo`
4. Start the application: `docker run -it --network qfstandalone-net -e TEXTSTORE_HOST=textstore -w /app -v ./src:/app/src --name qfstandalone -p 8080:3000 qfstandalone`

**Explanation: Start the Database**

```
docker run                  # Start a Container
-d                          # In detached mode (in the background)
--network qfstandalone-net  # Connect to the virtual network we just created
--network-alias textstore   # Make this container accessible
                            # on the network using this name
--name textstore            # Use this name when we access
                            # the container with docker
mongo                       # Use this image as base for the container
```

**Explanation: Start the Application**

```
docker run                    # Start a container
-it                           # In interactive mode, and attach
                              # a terminal so we can also type into it
--network qfstandalone-net    # Same virtual network
-e TEXTSTORE_HOST=textstore   # Set the environment variable to the
                              # network alias of our MongoDB database
-w /app                       # Set the working directory inside the container
-v ./src:/app/src             # Attach the host directory ./src
                              # to the guest under /app/src
```

```
--name qfstandalone           # Container name
-p 8080:3000                  # Connect host port 8080 to
                              # port 3000 in the container
qfstandalone                  # Use this image (the tag we previously set)
```

## 7.6 Test

1. Add a book. Go to `http://localhost:8080/add`
   - Use e.g. a book from the Gutenberg Project: `https://www.gutenberg.org/`
   - If you don't type anything, you will add a text version of Leo Tolstoy's *War and Peace*
2. Go to `http://localhost:8080/` and search for something, e.g. 'prince'.

To do:

- Keep an eye on your terminal. What is printed? What happens?
- Since we started witht he flags `-it` we can control the application in the terminal.
  - Try typing `rs` and press `<enter>`, what happens?
  - This is because the program is run through `nodemon`: `https://nodemon.io/`
- We also started the program with a *bind mount*: `-v ./src:/app/src`
  - Open the file `src/index.js` and find the method `StartPage()`
  - replace the return line with `return listTextsPage(req, res);` and save.
  - What happens in the terminal?
  - Reload the start page in the web browser. You should now also see a list of all available texts.

## 7.7 Stop and Clean

- Stop the running application by pressing `Ctrl-C` in the terminal
  - This stops the running container `qfstandalone`
  - The database container `textstore` continues to run in the background.
  - The network is still available.
  - Check what is left: `docker ps -a`
- Time to clean up:

```
docker rm -f textstore qfstandalone
docker network rm qfstandalone-net
docker network prune -f
```

### 7.8 Start the application: podman/docker compose

- We have already introduced a `docker compose` file to start the application
- Open and study the file `docker-compose-v1.yml`
- Start the application with `docker compose -f docker-compose-v1.yml up`
- Test as before with `http://localhost:8080/` and `http://localhost:8080/add`

To do:

- Note how the terminal printouts differ
- What happens now when you write `rs` in the terminal=
- What happens when you abort with `Ctrl-C` ? Check with `docker ps -a`

### 7.9 Extra: Communicate with the application

1. Update the docker compose file as below
2. In a separate terminal, connect to the running container: `docker compose -f docker-compose-v1.yml attach app`

```
version: "3.8"
services:
  app:
    image: qfstandalone
    stdin_open: true # docker run -i
    tty: true        # docker run -t
    ports:
      - 8080:3000
    volumes:
      - ./Containers/Version1/QFStandalone/src:/app/src
    environment:
      TEXTSTORE_HOST: textstore
  textstore:
    image: mongo
    command: --quiet --syslog
    expose:
      - "27017"
```

### 7.10 Summary

1. Build an `image`
2. Start a `Container`
    - Start a single container
    - Start a collection of containers with a single command

3. Edit files locally and see them change inside a running container

Advantages

- Can run any program and programming language inside a container
- Repeatable deployment

Disadvantages:

- Can run any program and programming language you want inside a container; including malware
- It should be, but is not, entirely transparent to take the next step out onto "the cloud"
- The database is not quite persistent yet...

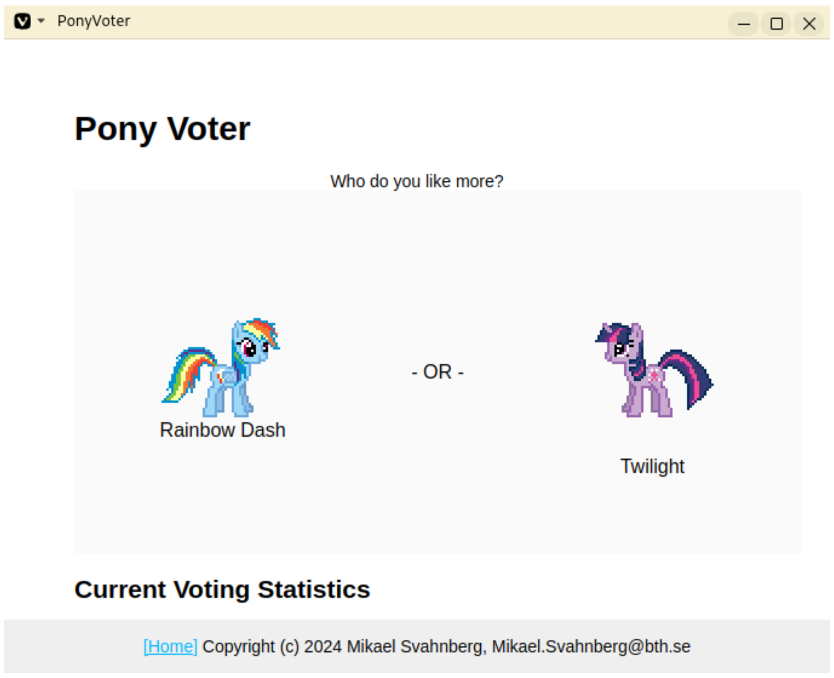
## 8 Development with Microservices

### 8.1 Introduction: PonyVoter

- We are staying in Equestria, but this time with a simple voting application
- PonyVoter presents two options and you vote by clicking on one of them
- The votes are registered in a database so that you can keep track of which pony is the most popular.

Get Started:

1. Download the project: `https://codeberg.org/mickesv/PonyVoter.git`
2. Study the files, try to form your own opinion about what you have downloaded.


PonyVoter
Pony Voter
Who do you like more?
Rainbow Dash
- OR -
Twilight
Current Voting Statistics
[Home] Copyright (c) 2024 Mikael Svahnberg, Mikael.Svahnberg@bth.se

## 8.2 Technical Overview

- PonyVoter consists of three containers and a database

  **PonyVoter** the "front page" of the application that serves web pages to the users

  **VoteCounter** registers votes and stores them to the database

  **StatsPresenter** calculates how many votes each pony has, and summarises this

  **MongoDB** the database where the votes are stored.

- PonyVoter is hopelessly over-designed and at the same time under-implemented:
  - `VoteCounter` and `StatsPresenter` are extremely simple and probably did not need to be separate containers.
  - Many things are hard coded.
  - To keep things simple, no rendering engine (e.g. `Pug`) is used to generate the HTML code.
  - To keep the project small, there are only six ponies to choose between.
  - Completeness? Only the bare necessities are implemented.
  - Scalability? What happens when the total number of votes increases?
  - Bugs! Of course there are bugs.
  - Security?

Think about:

1. What containers should be made available to the user?
2. How do you ensure that these become available?
3. How can you start all the containers with a single command?

## 8.3 Start and Test

1. The file `ponyvoter.yaml` is used by `docker compose` to build and start the application.
   - How is it constructed?
   - What is specified for each container?
   - Are the `volumes` blocks necessary? What do they do?
   - Can you see how to access each container?
2. Start the application: `docker compose -f ponyvoter.yaml up`
3. Visit `http://localhost:8080` and test the application
   - Keep an eye on the terminal while running. What is printed?
4. Abort by pressing `Ctrl-C` in the terminal.
   - What happens?

- Check with `docker images` what images you have
- Check with `docker ps -a` what container are running or no longer running

5. Start again (same command)
   - What happens?
   - Note that the statistics are not reset despite all containers being restarted.
     - Why not?
     - How can you find out more information about this?

## 8.4 Find the Datbase

1. Check with volumes that docker has created: `docker volume ls`
   - There should be two with long non-names, e.g. `aa5972d833f74bc8085bafdc32aa279e45c8d29cf63`
   - Could these contain the database?
   - Can we find more information? `docker volume inspect aa5972d833f74bc8085bafdc32aa279e4`
2. Try finding out more through the back way. `docker ps -a` shows that the database's name is `ponyvoter-mongodb-1`
   - What information can you get from `docker inspect ponyvoter-mongodb-1` ?
   - Look for "Mounts" in the printput, or filter a bit first: `docker inspect -f '{{.Mounts}}' ponyvoter-mongodb-1`

Our suspicion was true! MongoDB uses two volumes:

- `/data/configdb` and `/data/db` .

To do:

1. Read up on *Volumes* in the documentation to docker compose.
2. Mofify `ponyvoter.yaml` so that `mongodb` uses two *named* volumes; `db-data` and `db-config`.
3. Clean away the two old volumes with `docker volume prune` .

## 8.5 Scale the Application

- Since all containers adhere to REST principles, it is easy to scale
- In `ponyvoter.yaml` you specify the number of `replicas` a certain service should have when deploying,
  - It is slightly more complicated than this; please read up in the official documentation first.

To do:

1. Update `ponyvoter.yaml` so that it deploys 3 replicas of `votecounter`
2. Restart the application.
3. Vote for a couple of ponies and keep an eye on the terminal. What happens?
    - Is there any pattern to how your three replica are being used?
4. Think about
    - Can you change to five replicas *without* restarting your application? How? Try!
    - Tip 1: Is it enough to change the yaml file?
    - Tip 2: `up` has the flag `--detach`
    - Tip 3: You may not even need to change the yaml-file...
        - check what you can do with `docker compose --help`

## 8.6 Offer and Use REST

Let's have a look inside the application.

`Containers/StatsPresenter`

- Has a single code file: `src/index.js`
- Consists of four major parts:
    1. Create an express web server
    2. Connect to the database
    3. Configure and start all the REST endpoints
    4. Functions for each endpoint.

`Containers/VoteCounter`

- Essentially the same as StatsPresenter.

`Contaners/PonyVoter`

- Some more functions, but essentially the same structure.

To do:

- Which REST endpoints are offered by each container?
- Are they GET, POST, PUT, or DELETE? What should they be?
- WHat types of answers are returned from each endpoint?
- How can you test this?

## 8.7 Test the API

- Only `PonyVoter` is available from the host computer…
- How can we test the other containers?

To do:

1. Study `Containers/APITester` so you know what it does.
   - Also study `test.yaml`
2. Start the `PonyVoter` application
3. Run `docker compose -f test.yaml up` and see what happens.
   - Note that you have three different types of answers with different `content-type`
   - How can you use this when you build a REST api?

## 8.8 Think about/find out

- Can you check the health of a container?
  - How would you write such a *healthcheck* in your docker compose file?
  - Do you always need to have a separate endpoint in your REST-API for this?
    - When do you definitely need a separate endpoint?
    - Are there other solutions?
- Some deployment platforms have the concept of `Init containers`
  - What are init containers used for?
  - How can you achieve this with docker compose?
- What are docker compose *Secrets* ?
  - When should you use them?
  - How?
- What do you need to do in order to make your docker compose file *production ready*?

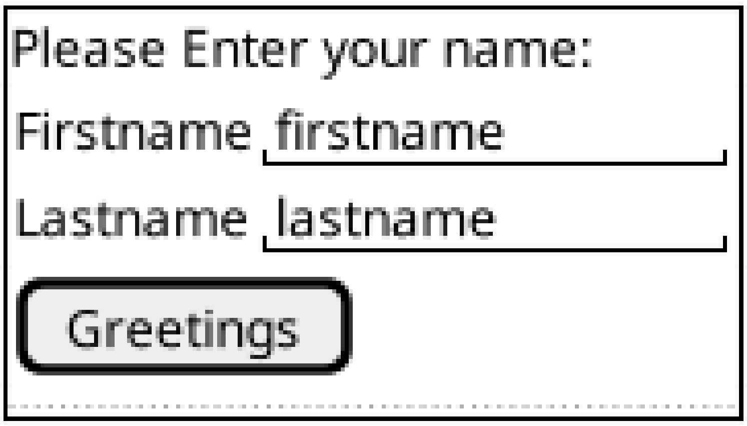
## 8.9 Summary

- You have now worked with a *microservice* applicsation
- Every conmponent (container) has its own *REST API*
- You have used several different programming languages (`JavaScript/Node.js` and `bash` )
- You have scaled parts of your application up and down.

# 9 Get Started with JavaScript

## 9.1 Introduction to the Exercise

- This time we will start from scratch with a completely new project.
- You may wish to create the project on your git server first and clone it from there.
  - If not, then at least start in a new directory with `git init`
- In this project you will create a webpage where a user may enter their first and lastname, and get a greeting in return.
- As a part of the project, *at least* create the following
  1. a `Dockerfile` that
     - installs `nodemon`
     - installs dependencies from `package.json`, and
     - starts the application with `ENTRYPOINT ["npm", "run", "dev"]`.
  2. A `package.json` that at least:
     - declares `Express ^4.19.1` as a dependency
     - has a `dev` script that starts the application using nodemon.
  3. A file `src/index.js` that starts an express application with two routes:
     - `GET /` to serve a page as below
     - `GET /greet? that adds a line =Hello, Firstname Lastname"` to the served page.
  4. A file and a class `src/person.js` to
     - represent a person with `firstname()`, `lastname()`, and `fullname()`.
     - stores the names with a capital first letter (e.g. "john" is stored as "John")
     - has a method `greet()` that returns `this.fullname()`
     - Don't forget: `module.exports = Person`
  5. (if it pleases) a `makefile` with two rules:
     - `build` (`docker build . -t namegreeter`)
     - `run` (`docker run -it -p8080:3000 -w /app -v ./src:/app/src namegreeter`)



## 9.2 Save Greetings

- Add a page `GET /list` that shows all persons that have been greeted earlier

To Do:

1. Add an array `previousGreetings` in index.js to store your `Person` objects in
2. Add a route to `GET /List` in `index.js` to list all persons from your `previousGreetings`

## 9.3 Count Greetings

- For every person you greet, check in `previousGreetings` if you have greeted them before (Assume that if "Firstname Lastname" is the same, they are the same person)
- Add a counter in the Person class which is increased every time you greet that person.

## 9.4 Family Relations

- If only the lastname is the same but not the firstname, it is a relative.
- Add the method `addRelative(aPerson)` in your Person class
  - Double check first so that the person is not already listed as a relative.
- Modify `GET /list` so that all relatives are also listed for each person.
- Modify `GET/list` so that the names are printed in alphabetical order based on the last name.

### 9.5 Summary

- You have now
  - Written a simple web application in a container
  - Written a class in JavaScript
  - Added methods to the class
  - Saved and retrieved objects to and from collections

## 10 Application development in JavaScript

### 10.1 Introduction: Craic – a simple chat prohrga,

- In this exercise we are working with a simple chat program: *Craic*
  - *Craic* is an irish word for gossip
  - The application consists of typing short messages to each other.
  - `https://codeberg.org/mickesv/craic.git`

### 10.2 Challenges

**Understand an existing codebase**
- You get the pleasure of understanding the system and system architecture on your own
- How do you run the program?
- What are the main components?
- What are the responsibilities of each module?

**User Interfaces from inside a container**
- We *could* have written yet another web client, but wanted something different
- Text-based UI, so called *TUI*

### 10.3 Get Started

1. Download the project: `https://codeberg.org/mickesv/craic.git`
2. Undestand the project
   - How do you run the program?
   - What are the main components?
   - What are the responsibilities of each module?
3. Test run
   - Write some messages
   - Put your mouse upside-down and try to only use the keyboard.
     - (tip, you need to press `<escape>` to exit a text field).
4. Can you connect to a colleagues server? How?

## 10.4 Update the Client

1. Add a textfield with the name of the server to use
2. Make sure the specified server is actually used
3. Testrun together with a colleague

## 10.5 Update the Server

1. Add more banned words and names (note that some are given as *Regular Expressions* )
2. Add a module that enables `#tagging` of keywords and `@mentioning` other users
   - Are these only stored temporarily in the server or should they be added to the database?
   - How do you search for a certain `#tag`? Add this to the REST API of the Server
3. Add support for getting pages of messages
   - You need to add `page=xxx` to the query
   - You need to add `page=xxx` and `nextPage: yyy` in the reply.
   - Extra points if you do not use page numbers but instead calculate a key. (Being able to figure out page numbers is a common security vulnerability).

Think about

- What happens when you use a banned word but spell it differently, e.g. "belGIUm"?
- How can you test your API changes?
- Is it a new major version of the product when you add new REST endpoints?
- Is it a new major version when you add support for pagination?
- Can you simplify the server so that it has a configurable list of filters to apply rather than them being hard coded?
  - How would you implement this?

## 10.6 Write Tests

1. Plan and write Mocha/Chai tests for the Server
2. Plan and write Mocha/Chai tests for teh Client

### 10.7 Create a Web Client

1. Create a new container that runs a web client similar to the TUI-based client.
2. Run the application with both the TUI-client and the web client running at the same time.
3. Can you reuse your tests from the TUI-client?

### 10.8 Think about the REST API

- How do the clients currently know whether there are any new messages?
  - What does this mean for the server?
- How can you do this differently / easier for the server?
  - Try!

### 10.9 Summary

- With containers and modules each part of the program becomes stand alone and easy to adapt
- REST APIs are not quite as easy as method calls but almost.
- Scalability
  - A separate container for `#tags` and `@mentions` ?
  - Multiple server containers with a load balancer?
- Different types of clients.

## 11 Get Started with Databases

### 11.1 Docker Compose file

- There is no need for a git repo this time, we can start with a simple docker-compose file as below.
  - What does this file do?
  - The default user is `postgres` but you must specify the password.
- Start with `docker compose` as usual.

```
version: ’3.9’
services:
  db:
    image: postgres
    restart: always
    shm_size: 128mb
    environment:
      POSTGRES_PASSWORD: hunter2
```

```
adminer:
  image: adminer
  restart: always
  ports:
    - 8080:8080
```

## 11.2 Overview of Adminer

- Open a web browser to `http://localhost:8080`
- At the login you need to specify

  **System** PostgreSQL

  **Server** db (since that's the name of the database in the docker compose file)

  **Username** postgres

  **Password** hunter2 (or whatever you changed it to in your docker compose file)

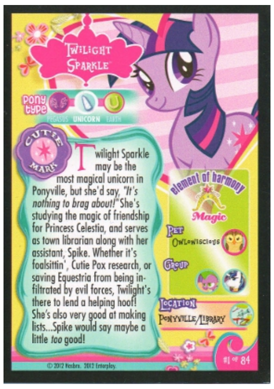
- Adminer provides a web interface to manage databases, tables, and values.
- You can also write SQL statements directly
  - This is particularly useful when you want to do more complicated or repeated operations.

**Tip** Become Good Friends™ with your IDE, and I'm sure you can connect to the database therefrom.

- In that case, you need to ensure that the database is available on port 5432 also from outside the docker compose cluster.
- Don't forget to "close it in" again once you are done with development so that no-one from the outside is able to hack your database.

## 11.3 Create a Database

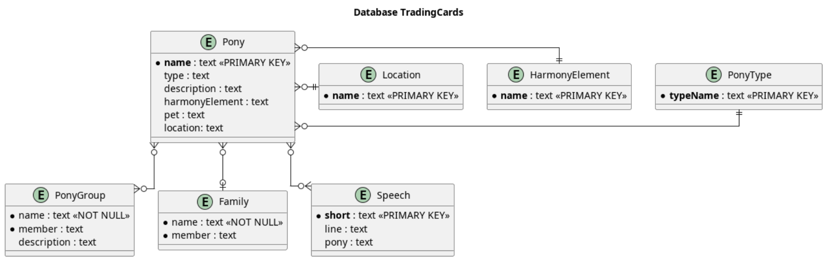
- We continue with the Pony-extravaganza
- This time we want to create a database to generate "Trading Cards"
- For now, we'll ignore images, cutie marks, etc. and stick to plain text.
- The database design is simplified; this is where you need a whole course on databases just to understand how and why.
- **TODO** Create a database `TradingCards`.



Twilight Sparkle may be the most magical unicorn in Ponyville, but she'd say, *"It's nothing to brag about!"* She's studying the magic of friendship for Princess Celestia, and serves as town librarian along with her assistant, Spike. Whether it's foalsittin', Cutie Pox research, or saving Equestria from being infiltrated by evil forces, Twilight's there to lend a helping hoof! She's also very good at making lists...Spike would say maybe a little *too* good!

## 11.4 Create Tables and Columns

1. Create the following tables and columns:



## 11.5 Add some Data

1. Download the file `https://codeberg.org/mickesv/gists/raw/branch/main/TradingCards_insert.sql`
   - Read it so you understand what it does.
2. Find the page `"SQL command"` in the adminer interface.
3. Paste the contents of the file and press execute.
   - If the database was created correctly, all the `INSERTS` should worl.

- Otherwise, *read the error message*, fix and try again.

4. Add some more ponies, e.g. from this page:
   - `https://mlp.fandom.com/wiki/My_Little_Pony_Friendship_is_Magic_Wiki`

## 11.6 Simple Searches

1. Use the adminer interface to do some simple searchers
   - To the left in the interface there are links `select` and `table name` for each table.
   - Use "select" for the right table and fill in the fields for "Select" and "Search" so that you may find:

| Show the following fields | from the table | matching the condition |
|---|---|---|
| name, type | pony | name is exactly (=) "Rarity" |
| name, type | pony | name contains (~) "Twilight" |
| member | family | name is exactly "Apple" |
| pony, line | speech | pony contains "Rainbow" |

## 11.7 Combined Searches

1. How would you express the following queries?
   - Find all Pony.name and Pony.type for ponies that are part of a Family.
   - What are all the speak.line that all Ponies currently in Ponyville may say?
   - What are all the speak.line that all Ponies currently *not* in Ponyville may say, and what are the ponies' names?
   - Find the names of all Ponies mentioned in a PonyGroup that do not yet exist in the table Pony.
2. Try to ask these questions in adminer.
   - You may need to use "SQL Command" to succeed.

## 11.8 Search and manage results from a computer program

1. Create a container 'PonyTradingCard' that searches the database and list all ponies according to the tempate below.
   - Choose whichever programming language you wish. In node.js you need the package `"pg"` for PostgreSQL.
2. Update your docker compose file so that this container is also run.

```
--------------------
Pony: Fluttershy
Type: Pegasus
Element of Harmony: Kindness
Pet: Angel
Description: Very shy and scared of dragons.
Location: Everfree Forest

Family Members:
- Mr. Shy
- Mrs. Shy
- Zephyr Breeze

Groups:
- Gen 4
- Main Character

Speech:
- "Oh, my."
- "I don’t wanna talk about it."
- "I’d like to be a tree."
```