Recap

- Git: Open source software to manage versions
- Commit: One specific version that knows its predecessor
- Branch: Multiple different commits can have the same predecessor, allowing parallel development
- Merging
  - Re-integrate parallel development
  - Mostly automatic, but sometimes not

Section 1

Last Exercise

Last Exercise

Lessons and Remarks

- Your GitHub username is not (automatically) your author name
- Commit date is fixed when you make the commit – not when you upload to GitHub
  - Shows that many actual code additions were done this week (not last week, as was the idea)

Last Exercise

Lessons and Remarks

- Your GitHub username is not (automatically) your author name
- Commit date is fixed when you make the commit – not when you upload to GitHub
  - Shows that many actual code additions were done this week (not last week, as was the idea)
- Never do a force push if you're sharing the repository with others
  - Never follow stackoverflow recommendations *without understanding their consequences*

Last Exercise

Lessons and Remarks

- Your GitHub username is not (automatically) your author name
- Commit date is fixed when you make the commit – not when you upload to GitHub
  - Shows that many actual code additions were done this week (not last week, as was the idea)
- Never do a force push if you're sharing the repository with others
  - Never follow stackoverflow recommendations *without understanding their consequences*
- Some commits produce badly broken code (e.g, undeclared variable names)
  - Never push things to the server that *do not compile*

Last Exercise

Lessons and Remarks

- Your GitHub username is not (automatically) your author name
- Commit date is fixed when you make the commit – not when you upload to GitHub
  - Shows that many actual code additions were done this week (not last week, as was the idea)
- Never do a force push if you're sharing the repository with others
  - Never follow stackoverflow recommendations *without understanding their consequences*
- Some commits produce badly broken code (e.g, undeclared variable names)
  - Never push things to the server that *do not compile*
- Pull requests: Coordination mechanism
  - To the maintainer: "Hey, I've written some code, please pull it into the main project"

Section 2

Remotes

Decentralized

- “Git is decentralized”: What does this mean exactly?

Decentralized

- “Git is decentralized”: What does this mean exactly?
- No central server required
- A local git repository stores the entire history, all branches and tags
- Every clone of the repository has the entire history
    - Offline working galore!

Remotes

- Each repository can be associated with multiple ‘remotes’
    - Usually, one remote is called ‘origin’
- `clone` makes a local clone *and* sets one remote to point to the source

Remotes

- Each repository can be associated with multiple ‘remotes’
    - Usually, one remote is called ‘origin’
- `clone` makes a local clone *and* sets one remote to point to the source
- Merging works across remote repositories
    - E.g., you can merge something from a remote branch into your local branch

Downloading stuff

- A branch can be set to ‘track’ a remote branch
  - Typically, you want the branches to have the same name
- `git fetch` downloads all tracked branches to your local repository, but keeps your working copy as it is
- `git pull` fetches the changes from the server *and* merges them into your working copy
  - Merge conflicts can occur!
- `git push` pushes your local changes to the tracking branch on the server
  - If the remote branch moved on, you’ll be forced to pull and merge first

How to ask for technical Support

- You may need to write to various people to get technical support
- Take a moment to think before clicking “send”

How to ask for technical Support

- You may need to write to various people to get technical support
- Take a moment to think before clicking “send”

Ensure that

- All relevant information is given (as far as you know)
- You use proper terminology (as far as you can)
- You make it easy for the other person
    - E.g., by including information the other person might first need to look up
- The context is still conceivable
    - I.e., click on reply instead of writing a new mail, keep the old mail text in there
- References in your text are clear
    - For instance: “this exercise” is not a clear reference
- You’re concise – long e-mails tend to be put on the read-later-pile (which never happens)

Session 4: Iterable and Iterators

Fortgeschrittene Programmierung (Java 2)

Nils Reiter
nils.reiter@uni-koeln.de

26. April 2023

INSTITUT FÜR
DIGITAL HUMANITIES
UNIVERSITÄT ZU KÖLN

Section 3

Introduction and Motivation

Iterating

- Programs with only single variables are not very powerful
- Power comes from possibility to group things of the same type
  - E.g., arrays: `int[] myArray = new int[1,2,3,4,5,6,7,8,9];`

Iterating

- Programs with only single variables are not very powerful
- Power comes from possibility to group things of the same type
  - E.g., arrays: `int[] myArray = new int[1,2,3,4,5,6,7,8,9];`
- Arrays allow treating many things the same way, because they have a common name
  - E.g.: `myArray[5] = myArray[5] * 2;`

Iterating

- Programs with only single variables are not very powerful
- Power comes from possibility to group things of the same type
  - E.g., arrays: `int[] myArray = new int[1,2,3,4,5,6,7,8,9];`
- Arrays allow treating many things the same way, because they have a common name
  - E.g.: `myArray[5] = myArray[5] * 2;`
- For this, we need a method to *iterate* over the elements of the array
  - E.g.: `for (int i = 0; i < myArray.length; i++) { }`

Iterating

- Programs with only single variables are not very powerful
- Power comes from possibility to group things of the same type
  - E.g., arrays: `int[] myArray = new int[1,2,3,4,5,6,7,8,9];`
- Arrays allow treating many things the same way, because they have a common name
  - E.g.: `myArray[5] = myArray[5] * 2;`
- For this, we need a method to *iterate* over the elements of the array
  - E.g.: `for (int i = 0; i < myArray.length; i++) { }`
- Iterating is such a central activity that Java offers different ways to do it
- `for (...) {...}`, `while (...) {...}`, `do {...} while (...)`

Schleifen

Loops: `for` and `while`

- How to decide which loop to use?

Loops: `for` and `while`

- How to decide which loop to use?
- No technical difference, it's about *code clarity*
  - I.e., for future code readers, potentially yourself

Loops: `for` and `while`

- How to decide which loop to use?
- No technical difference, it's about *code clarity*
  - I.e., for future code readers, potentially yourself

Example

```
for (int i = 0; i < myArray.length; i++) { ... }
int i = 0; while (i < myArray.length) { i++; ... }
```

Loops: `for` and `while`

- How to decide which loop to use?
- No technical difference, it's about *code clarity*
    - I.e., for future code readers, potentially yourself

Example

```
for (int i = 0; i < myArray.length; i++) { ... }
int i = 0; while (i < myArray.length) { i++; ... }
```

- What are the important elements of any loop?

Loops: `for` and `while`

- How to decide which loop to use?
- No technical difference, it's about *code clarity*
  - I.e., for future code readers, potentially yourself

Example

```
for (int i = 0; i < myArray.length; i++) { ... }
int i = 0; while (i < myArray.length) { i++; ... }
```

- What are the important elements of any loop?
  - Initial state (`int i = 0`)
  - Condition to terminate (`i < myArray.length`)
  - Change in each step (`i++`)

Problems

Example (File Search)

- 1000s of files
- Search term is a single word
- We're interested in the first file with the word

Problems

Example (File Search)

- 1000s of files
- Search term is a single word
- We're interested in the first file with the word
- Solution so far
  - Create an array with all contents of the files
  - Iterate over the array
  - Return the one we want, disregard all others

Problems

Example (File Search)

- 1000s of files
- Search term is a single word
- We're interested in the first file with the word
- Solution so far
  - Create an array with all contents of the files
  - Iterate over the array
  - Return the one we want, disregard all others
- Wasteful: Most file contents will probably never be read
- Better: After inspecting each file, see if you need to load another

Section 4

Iterator

Iterator

- An interface in the Java library: `java.util.Iterator`
- A iterator iterates once over a collection of objects

java.util.Iterator

Iterator

- An interface in the Java library: `java.util.Iterator`

java.util.Iterator

- A iterator iterates once over a collection of objects
- Four methods (only two non-optional):

  `boolean hasNext()`: Returns `true` if there are more elements in the sequence

  `E next()`: Returns the next element in the collection

  `void remove()`: Removes the last element returned (optional)

  `void forEachRemaining(Consumer<? super E> a)`: Applies action to elements not yet returned

Iterator

- An iterator object represents a specific iteration over a specific collection
- Iterators can (mostly) not be used twice
- Iterators are most naturally used in combination with while-loops:

```java
Iterator iter = ...
while(iter.hasNext()) {
    Object myObject = iter.next();
}
```

Iterator

- An iterator object represents a specific iteration over a specific collection
- Iterators can (mostly) not be used twice
- Iterators are most naturally used in combination with while-loops:

```java
Iterator iter = ...
while(iter.hasNext()) {
    Object myObject = iter.next();
}
```

Benefits

- We only inspect/load as many elements as needed
- Object-oriented iteration: The iterator object represents the iteration itself
- Iterators make iterating easier (and object oriented) – they do not add something what would be impossible otherwise

demo

Section 5

Iterable

Iterable

- An interface in the Java library: `java.lang.Iterable`
- Provides a single (non-default) method: `Iterator<T> iterator()`
  - I.e.: the method returns an Iterator

Iterable

- An interface in the Java library: `java.lang.Iterable`
- Provides a single (non-default) method: `Iterator<T> iterator()`
  - I.e.: the method returns an Iterator
- An object that implements `Iterable`
  - is iterable, i.e., can be iterated on
  - can be used in a for-loop like this:

```java
for (Object o : myIterable) {
    o.doSomething();
}
```

demo

Generics

Topic for next week, but:

- Some classes are written with angle brackets: `Iterator<Student>` / `Iterable<Student>`
- Angle brackets contain the type that we iterate over
- This allows us to re-use the same code to iterate over different tyes!

Next Week: No Class!

Exercise

`https://github.com/idh-cologne-java-2-summer-2023/exercise-04`