Assignment 8.1 Quiz

1. Can Smalltalk Inheritance be used to model Java’s inheritance? What about attribute access?

2. In a Trait-based programming language, what purpose serves a concrete class, abstract class and a trait on its own?

3. Let C be a class, composed from the Mixins M and N. Suppose, M and N both implement a method f(). Is it true that
   - The conflicting methods f() from M and N lead to a compiler error and have to be resolved manually

4. Now assume, that M and N are Traits instead of Mixins. Is it true that
   - The conflicting methods f() from M and N lead to a compiler error and have to be resolved manually
   - There is no compiler error, but one implementation of f() from M or N overwrites the other

5. Why is exclusion an important composition operator for Traits?
7. Given the traits $T_1$, $T_2$ and class $C$:

$$T_1 = \{ a \rightarrow 1 \}$$

$$T_2 = \{ a \rightarrow 2 \}$$

$$C = (\emptyset, \text{nil} \triangleright (T_1 + T_2))$$

What is the value of $(C \triangleright T_1)(a)$?

8. (Attention: Several answers might be true for this question!)

$$c_1 \sqcup c_2 = c_1 \sqcup c_2$$ is true for

- $c_1 = \{ a = 0 \}$, $c_2 = \{ a = 0 \}$
- $c_1 = \text{mixin}(c_3)(c_2)$, $c_2 = \{ a = 0 \}$, $c_3 = \{ a = 0 \}$
- $c_1 = \text{mixin}(c_2)(c_3)$, $c_2 = \{ a = 0 \}$, $c_3 = \{ a = 0 \}$
- $c_1 = \text{mixin}(c_3)(c_4)$, $c_2 = \{ a = 0 \}$, $c_3 = \{ a = 0 \}$
- $c_1 = \text{mixin}(c_2)(c_4)$, $c_2 = \{ a = 0 \}$, $c_3 = \{ a = 0 \}$
- $c_1 = \text{mixin}(c_3)(c_4)$, $c_2 = \{ a = 0 \}$, $c_3 = \{ a = 0 \}$

Assignment 8.2 Having fun with Mixins

Reconsider the example from the lecture about synchronized file- and socket-streams. The following classes are given:

- `FileStream = \{ read = 0x1, write = 0x2 \}`
- `SocketStream = \{ read = 0x3, write = 0x4 \}`
- `SyncRW = \{ read = 0x5, write = 0x6 \}`

Your task is to come up with a new class `SynchronizedFileStream` which mixes the class `SyncRW` into the class `FileStream`.

Assignment 8.3 Mixins Ruby

Implement the `Stream Wrapper` scenario from the lecture based on Ruby Mixins.
Reconsider the example from the lecture about synchronized file- and socket-streams. The following classes are given:

\[
\begin{align*}
\text{FileStream} &= \{ \text{read} = 0 \times 1, \text{write} = 0 \times 2 \} \\
\text{SocketStream} &= \{ \text{read} = 0 \times 3, \text{write} = 0 \times 4 \} \\
\text{SyncRW} &= \{ \text{read} = 0 \times 5, \text{write} = 0 \times 6 \}
\end{align*}
\]

Your task is to come up with a new class \textit{SyncedFileStream} which mixes the class \textit{SyncRW} into the class \textit{FileStream}.

**Assignment 8.3 Mixins Ruby**

Implement the \textit{Stream Wrapper} scenario from the lecture based on Ruby Mixins.

---

**Assignment 8.4 Implementation differences: Traits vs. Mixins**

A next mainstream implementation of traits comes with the Java version 8.

- Implement a solution for the \textit{Stream Wrapper} problem. You may use the following code:

```java
interface Stream {
    int read();
}

interface FileStream extends Stream {
    default int read() { /* ... */ }
}

interface NetworkStream extends Stream {
    default int read() { /* ... */ }
}

interface Sync {
    default void acquireLock() { /* ... */ }
    default void releaseLock() { /* ... */ }
}
```

- Compare your solution to the one based on Mixins from the last exercise sheet.

---

**Assignment 8.4 Implementation differences: Traits vs. Mixins**

A next mainstream implementation of traits comes with the Java version 8.

- Implement a solution for the \textit{Stream Wrapper} problem. You may use the following code:

```java
interface Stream {
    int read();
}

interface FileStream extends Stream {
    default int read() { /* ... */ }
}

interface NetworkStream extends Stream {
    default int read() { /* ... */ }
}

interface Sync {
    default void acquireLock() { /* ... */ }
    default void releaseLock() { /* ... */ }
}
```

- Compare your solution to the one based on Mixins from the last exercise sheet.

**What are the differences?** Which one is more flexible w.r.t. software engineering?
Assignment 8.4 Implementation differences: Traits vs. Mixins

A next mainstream implementation of traits comes with the Java version 8.

- Implement a solution for the Stream:Hipster problem. You may use the following code:

```java
interface Stream {
    int read();
}

interface FileInputStream extends Stream {
    default int read() { /* ... */ }
}

interface CounterInputStream extends FileInputStream {
    default void acquireLock() { /* ... */ }
    default void releaseLock() { /* ... */ }
}
```

- Compare your solution to the one based on Mixins from the last exercise sheet.

What are the differences? Which one is more flexible with software engineering?

1. How many errors are in this code? Elaborate extensively on the causes!

2. Fix the code, while conserving the given inheritance relation, so that the whole program compiles!
Consider the following Scala code:

```scala
trait Foo {
  def f = println("Foo.f")
}

trait Bar {
  def f = println("Bar.f")
}

trait Baz {
  def f = println("Baz.f")
}
```

1. Is the call to the function f the same in `FooBarBaz` and `FooBazBar`?

```scala
class FooBarBaz extends Foo with Bar with Baz {
  override def f = super f
}
class FooBarBazBaz extends Foo with Bar with Baz {
  override def f = super f
}
```

2. Would the example still compile if we change the trait Baz as in the following?

```scala
trait Baz {
  def f = super f
}
```