Das Concept `Ord`

```
template <typename T>
concept Ord =
    Equal<T> &&
    requires(T a, T b) {
        { a <= b } -> std::convertible_to<bool>;
        { a < b } -> std::convertible_to<bool>;
        { a > b } -> std::convertible_to<bool>;
        { a >= b } -> std::convertible_to<bool>;
    };
```

Any type that supports `Ord` must support `Equal`.

conceptsDefinitionOrdering.cpp

`SemiRegular` and `Regular`

`SemiRegular`

- Default constructor: `X()`
- Copy constructor: `X(const X&)`
- Copy assignment: `X& operator=(const X&)`
- Move constructor: `X(X&&)`
- Move assignment: `X& operator=(X&&)`
- Destructor: `~X()`

- Swappable: `swap(X&, Y&)`

`Regular`

- `SemiRegular`
- Equality comparable

SemiRegular and Regular

```
template<typename T>
struct isSemiRegular: std::integral_constant<bool,
                           std::is_default_constructible<T>::value &&
                           std::is_copy_constructible<T>::value &&
                           std::is_copy_assignable<T>::value &&
                           std::is_move_constructible<T>::value &&
                           std::is_move_assignable<T>::value &&
                           std::is_destructible<T>::value &&
                           std::is_swappable<T>::value
                         >{};
template<typename T>
concept SemiRegular = isSemiRegular<T>::value;

template<typename T>
concept Regular = Equal<T> && SemiRegular<T>;
```

(C++20) semiregular and regular

```
template<class T>
concept movable = is_object_v<T> && move_constructible<T> &&
assignable_from<T&, T> && swappable<T>;

template<class T>
concept copyable = copy_constructible<T> && movable<T> &&
assignable_from<T&, const T&>;

template<class T>
concept semiregular = copyable<T> && default_constructible<T>;

template<class T>
concept regular = semiregular<T> && equality_comparable<T>;
```

regularSemiRegular.cpp