Seminar 05

Destructor, copy constructor and operator=

1. Copy constructor.

- How our object will be copied when it's passed by value.
- Usually implemented via a copy method that can be used in the operator=.
- In code:

```
ClassName::ClassName(const ClassName& other)
{
    copyData(other);
}
```

2. Operator=.

- Overloading the '=' operator. *More on overloading operators in the next lesson.*
- In code:

```
ClassName& ClassName::operator=(const ClassName& other)
{
    if (this != &other) // Important check!
    {
        clearData();
        copyData(other);
    }
    return *this;
}
```

- ```
  ClassName c1("Sample text");
  ClassName c2;
  c2 = c1;  // operator= is called
  ```

- ```
  ClassName c1("Sample text");
  ClassName c2 = c1;  // Copy constructor is called, because
                      // ClassName c2 = c1;  ⇔  ClassName c2(c1);
  ```

3. Destructor.

- Called once the object is deleted via delete for dynamically allocated objects **or** once the program exits the code block where the object was instantiated.
- Mainly used for deleting dynamically allocated data members.
- Usually implemented via a clear method that is also used in the operator=.
- In code:

```
ClassName::~ClassName()
{
    clearData();
```

```
}
```

4. Big four (the rule of three).

- A class is in a **canonical form** if it implements **the big four**:
    - default constructor, copy constructor, operator= and destructor.

- As a rule of thumb, we implement the big four when the class has at least one data member that **cannot** be copied easily, such as dynamically allocated memory. So that the object can create a **deep copy** of itself rather than **shallow**.

5. Exceptions basics.

- Exceptions should be used sparingly and only when you can't handle it another way!
- `throw [expression];` `// Throws exceptions to be handled`
- `try {...} catch (...) {...}` `// Catches exceptions to handle`
- `std::exception` `// Standard generic exception type`