Individual Exercises - Lecture 19

1. Read the article under additional references.
   Can be found here (Is available for free through AAU AUB here https://www.aub.aau.dk, remember to login first!):
   https://link.springer.com/article/10.1007%2FBF00289243

2. Sebesta, programming exercises 5, page 511, do the exercise in C, and Java or C#.
   5. Write an abstract data type for complex numbers, including operations for addition, subtraction, multiplication, division, extraction of each of the parts of a complex number, and construction of a complex number from two floating-point constants, variables, or expressions. Use C++, Java, C#, or Ruby.
   Example implementation using C. As C does not provide classes or objects a struct is used to represent the data while operations on the data type are implemented as functions. While objects could be simulated using function pointers taking the struct itself as the first argument, this would make the program much more complicated and, as such, much harder to read and maintain.

```c
typedef struct Complex {
  const double real;
  const double imaginary;
} Complex;
Complex new(double real, double imaginary) {
  return (Complex) { real, imaginary };
}

Complex add(Complex a, Complex b) {
  return new(a.real + b.real, a.imaginary + b.imaginary);
}

Complex sub(Complex a, Complex b) {
  return new(a.real - b.real, a.imaginary - b.imaginary);
}

Complex mul(Complex a, Complex b) {
  double real = a.real * b.real - a.imaginary * b.imaginary;
  double imaginary = a.real * b.imaginary + a.imaginary * b.real;
  return new(real, imaginary);
}

Complex div(Complex a, Complex b) {
  double x = a.real * b.real + a.imaginary * b.imaginary;
```

```
  double y = a.imaginary * b.real - a.real * b.imaginary;
  double z = b.real * b.real + b.imaginary * b.imaginary;
  return new(x / z, y / z);
}

double real(Complex a) {
  return a.real;
}

double imaginary(Complex a) {
  return a.imaginary;
}
```

Example implementation using Java. Contrary to C, Java provides support for object-oriented programming through classes and objects. Using a class both the code and data can combine into a single entity. In addition, the data can be hidden from the user of the class with the `private` access modifier while the data is only made readonly in the C version of the program. As Java does not support operator overloading (unlike C#) equations with `Complex` must be written using method calls instead of mathematical operators.

```
class Complex {
    public Complex(double real, double imaginary) {
        this.real = real;
        this.imaginary = imaginary;
    }

    public Complex add(Complex other) {
        return new Complex(this.real + other.real,
                           this.imaginary + other.imaginary);
    }

    public Complex sub(Complex other) {
        return new Complex(this.real - other.real,
                           this.imaginary - other.imaginary);
    }

    public Complex mul(Complex other) {
        double real = this.real * other.real -
            this.imaginary * other.imaginary;
        double imaginary = this.real * other.imaginary +
            this.imaginary * other.real;
        return new Complex(real, imaginary);
    }
```

```
    public Complex div(Complex other) {
        double x = this.real * other.real +
            this.imaginary * other.imaginary;
        double y = this.imaginary * other.real -
            this.real * other.imaginary;
        double z = other.real * other.real +
            other.imaginary * other.imaginary;
        return new Complex(x / z, y / z);
    }

    @Override
    public String toString() {
        return this.real + " " +  this.imaginary + "i";
    }

    public double getReal() {
        return this.real;
    }

    public double getImaginary() {
        return this.imaginary;
    }

    private final double real;
    private final double imaginary;
}
```

3. Sebesta, exercise 14 and 16, page 565.
   14. Explain diamond inheritance.
   Diamond inheritance occurs when a class (D) is a subtype of two different classes (B and C) through multiple inheritance which themself are a subtype of a class (A). A significant problem with multiple inheritance is that both B and C can define a method with the same name and the same protocol. For example, a class that implements an interface must define all of the methods declared in the interface. So, if both the parent class and the interface include methods with the same name and protocol, the subclass must reimplement that method to avoid the name conflict.

   See Also: https://en.wikipedia.org/wiki/Multiple_inheritance#The_diamond_problem

16. What is the primary reason why all Java objects have a common ancestor?

One reason why all Java objects have a common ancestor is so they can all inherit a few universally useful methods, e.g. equals(Object other), toString(), hasCode(). Another reason is that code can be written that works for all objects without using generics: https://en.wikipedia.org/wiki/Generics_in_Java.

Group Exercises - Lecture 19

1. Discuss the outcome of the individual exercises
   Did you all agree on the answers?

2. What are the language design requirements for a language that supports abstract data types?
   Sebesta lists three design issues for abstract data types:
   (i) The first design issue is whether abstract data types can be parameterized.
   (ii) The second design issue is what access controls are provided and how such controls are specified.
   (iii) Finally, the language designer must decide whether the specification of the type is physically separate from its implementation. For more information see Sebesta Section 11.3.

3. Sebesta, review questions, 21, 25, 33, 35, 36 page 509.
   21. Where are Java classes allocated?
   Instantiations are allocated from the heap
   25. Describe the fundamental differences between C# structs and its classes.
   Structs are value types, classes are references.
   Structs cannot be default constructed, inherit another struct, or implement an interface.
   33. What problems can occur using C to define abstract data types?
   C does not have complete support for abstract data types, but both abstract data types and multiple-type encapsulations can be simulated by wrapping them in a library file, i.e. a header file and then creating a .c file with the implementation of said functions.
   A problem with this solution could for example be, a user could simply cut and paste the definitions from the header file into the client program, rather than using #include. This would work, because #include simply copies the contents of its operand file into the file in which the #include appears. However, this introduces two problems. First, the documentation of the dependence of the client program on the library (and its header file) is lost. Second, suppose a user copies the header file into his or her program. Then the author of the library changes both the header and the implementation files. Following this, the user uses the new implementation file with the old header. For example, a variable x could have been defined to be **int** type in the old header file, which the client code still uses, although the implementation code has been recompiled with the new header file, which defines x to be **float**. So, the implementation code was compiled with x as an **int** but the client code was compiled with x as a **float**. The linker does not detect this error. Read more in section 11.6.2 in Sebestas book.

35. What is a Java package, and what is its purpose?
Packages provide naming encapsulation. Access modifiers (private, protected etc.) also allows restriction of access across packages.
36. Describe a .NET assembly.
An assembly is a file (.dll or .exe) that defines a module that can be separately developed. (See Sebesta Section 11.6.4)

4. Sebesta, problem set, exercise 2, page 510.
2. Suppose someone designed a stack abstract data type in which the function top returned an access path (or pointer) rather than returning a copy of the top element. This is not a true data abstraction. Why? Give an example that illustrates the problem.
The problems created by making an abstract data type a pointer are:
(a) There are many inherent problems in dealing with pointers (see Chapter 6),
(b) comparisons between objects do not work as expected, because they are only pointer comparisons (rather than pointed-to value comparisons),
and (c) the implementation of the abstract type cannot control allocation and deallocation of objects of the type (the user can create a pointer to an object with a variable declaration and use it without creating an object).

5. Sebesta, review questions, 1 - 58, page 562-564
1. Describe the three characteristic features of object-oriented languages.
Abstract data types
Inheritance
Dynamic binding
2. What is the difference between a class variable and an instance variable?
A class variable (static in C#) belongs to the class and is accessible from all instances of the class. This is opposed to the instance variable that belongs to the individual instance (object) of the class and is accessible only through that instance.
3. What is multiple inheritance?
The concept of a class inheriting from more than one class.
4. What is a polymorphic variable?
A variable that can hold values of different types.
5. What is an overriding method?
In terms of inheritance, it is sometimes useful to replace a method inherited from a superclass, that is where overriding comes in.
For example, in C# a method marked override explicitly replaces a method inherited from a superclass with the new method of the same signature. In C#, however, this is only possible if the method from the super class is marked as virtual.
6. Describe a situation where dynamic binding is a great advantage over static binding.
If you need to administrate objects of different classes, that all inherit from a common super class (base class), and call an inherited method on all of them.

7. What is a virtual method?

A dynamically bound method that can be overriding in a derived class. The keyword virtual is for example used in C# and C++.

8. What is an abstract method? What is an abstract class?

An abstract method has no implementation and must be defined in inherited derived classes.

An abstract class cannot be instantiated, and is instead meant as a base blueprint for derived classes to implement.

9. Describe briefly the seven design issues used in this chapter for object-oriented languages.

1. The exclusivity of objects: Should any other types be allowed?
2. Are subclasses subtypes?: Should two objects extending the same superclass be interchangeable?
3. Single and multiple inheritance: Can an object inherit from than one class?
4. Allocation and deallocation of objects: When and how? (Heap or stack?
5. Dynamic or static binding: Which to use?
6. Nested classes: Should we allow for classes only visible to specific classes?
7. Initialization of objects: How are objects initialized to values?

10. What is a nesting class?

A class defined within another class.

Useful if a class is only used in one place

11. What is the message protocol of an object?

The message protocol or message interface is all the methods defined on an object. A method call is equivalent to sending a message to the object.

12. From where are Smalltalk objects allocated?

All Smalltalk objects are allocated from the heap.

13. Explain how Smalltalk messages are bound to methods. When does this take place?

Smalltalk messages are dynamically bound to methods at runtime by traversing the inheritance hierarchy of the object receiving the message and executing the correct method when found. If the top of the hierarchy (Object) has been reached and no method matching the message is found, an error occurs. For more information see Sebesta Section 12.4.1.3 Dynamic Binding.

14. What type checking is done in Smalltalk? When does it take place?

Smalltalk only supports dynamic type checking. Type checking is performed at runtime when a message is sent to an object. For more information see Sebesta Section 12.4.1.3 Dynamic Binding.

15. What kind of inheritance, single or multiple, does Smalltalk support?

Smalltalk only supports single inheritance. All objects must derive from an existing object. For more information see Sebesta Section 12.4.1.2 Inheritance.

16. What are the two most important effects that Smalltalk has had on computing?

(i) The Smalltalk user-interface demonstrated how to integrate use of windows, mouse-pointing devices, and pop-up and pull-down menus.

(ii) The advancement of object-oriented programming. For more information see

Sebesta Section 12.4.1.4 Evaluation of Smalltalk.

17. In essence, all Smalltalk variables are of a single type. What is that type?

Everything is an object as stated in Sebesta Section 12.4.1.1 General Characteristics

18. From where can C++ objects be allocated?

Objects can be allocated statically, stack dynamic, and heap dynamic.

19. How are C++ heap-allocated objects deallocated?

Explicitly, with delete (or with smart pointers, such as shared_ptr, unique_ptr)

20. Are all C++ subclasses subtypes? If so, explain. If not, why not?

In the case of public derivation, yes.

In the case of private derivation, no.

21. Under what circumstances is a C++ method call statically bound to a method?

If it is not a pointer to the base class, or in other terms, if the class is determinable at compile time, not runtime. In general, method calls that are not defined as virtual are statically bound. See Sebesta Section: 12.4.2.3 Dynamic Binding.

22. What drawback is there to allowing designers to specify which methods can be statically bound?

Dynamic binding is slower than statically bound methods, so allowing users to define statically bound methods can increase the program's performance if a method does not need to be dynamically bound.

23. What are the differences between private and public derivations in C++?

In a private derivation public and protected members of the derived class are replaced by private copies.

24. What is a friend function in C++?

A function that has been granted access to private or protected fields of a class.

25. What is a pure virtual function in C++?

Roughly equivalent to an abstract function in C#

Has no implementation, written like this: virtual void foo() = 0;

Requires implementation in the deriving classes.

26. How are parameters sent to a superclass's constructor in C++?

Parameters are sent to the superclass constructor as part of the subclass constructor definition. Defining the inheritance requires the superclass's constructor parameters to be provided as well.

27. What is the single most important practical difference between Smalltalk and C++?

C++ uses static type checking which allows the compiler to detect errors at compile time which cannot be done in Smalltalk. C++ is generally also more efficient than Smalltalk

28. If an Objective-C method returns nothing, what return type is indicated in its header?

Void

29. Does Objective-C support multiple inheritance?

No

30. Can an Objective-C class not specify a parent class in its header?

Yes it can. As part of its interface declaration.

31. What is the root class in Objective-C?
The NSObject
32. In Objective-C, how can a method indicate that it cannot be overridden in descendant classes?
There is no way to prevent a subclass from overwriting an inherited method.
33. What is the purpose of an Objective-C category?
A category is a collection of methods that can be added to a class. No new instance variables can be added to this secondary interface. It can be used in some places to support multi inheritance without naming collisions on classes.
34. What is the purpose of an Objective-C protocol?
Similarly to an interface in Java/C#, it can provide some expected functionality of a class that implements the protocol.
35. What is the primary use of the id type in Objective-C?
It is a generic type that can refer to any object. Anything can be cast to an id type and an id type can be cast to anything.
36. How is the type system of Java different from that of C++?
Only primitive scalar types are not objects (bool, char, int, float etc.). Everything else is an object in Java. This is not the case for C++.
37. From where can Java objects be allocated?
All Java objects are dynamically allocated from the heap.
38. What is boxing?
Boxing is used in Java for having an object type of the primitive types (int, float, char etc.). The boxed versions har objects with names like Integer and Double. This was done, since the Java generics can only be used with objects. The compiler, however, autoboxes / unboxes.
The following code will box the i into an object of the type Integer and add it to the list.

```
List<Integer> li = new ArrayList<>();
for (int i = 1; i < 50; i += 2)
    li.add(i);
```

Due to boxing and type erasure the compiler changes the code above to approximately:

```
List<Object> li = new ArrayList<>();
for (int i = 1; i < 50; i += 2)
    li.add(Integer.valueOf(i));
```

39. How are Java objects deallocated?
In Java deallocation is done in the background by the Java garbage collector.
40. Are all Java subclasses subtypes?
Yes. The only class (which is not a subclass) that is not a subtype is the `Object` class.
41. How are superclass constructors called in Java?
`super(args)` from inside the subclass constructor.

42. Under what circumstances is a Java method call statically bound to a method?
Static binding is used if it is defined as `final`, `static`, or `private`.
43. In what way do overriding methods in C# syntactically differ from their counterparts in C++?
C# has a different abstract implementation.
In C# it is possible to mark a method `sealed` if they should be marked as explicitly not overridable.
44. How can the parent version of an inherited method that is overridden in a subclass be called in that subclass in C#?
Using the `base` keyword.
```
base.[MethodName]([Parameters]);
```
45. How does Ruby implement primitive types, such as those for integer and floating-point data?
Every value is an object. (unlike for example Java where an `int` is a primitive type but an `Integer` is an object)
46. How are getter methods defined in a Ruby class?
Multiple getter methods can be defined in one line using the following syntax:
```
Attr_reader :variable_1, :variable_2
```
Where `Attr_reader` is a built-in method that takes the variable names as arguments.
47. What access controls does Ruby support for instance variables?
Variables are effectively private, and constants public.
48. What access controls does Ruby support for methods?
Public, protected, and private.
49. Are all Ruby subclasses subtypes?
No.
50. Does Ruby support multiple inheritance?
No.
51. What does reflection allow a program to do?
To access its metadata directly, to use it, or to intercede in execution. In java you can for example read the name of a class and its fields into strings.
52. In the context of reflection, what is metadata?
The types, names and structures of a program.
53. What is introspection?
The process of a program examining its own metadata.
54. What is intercession?
The process of interceding in the execution of a program.
55. What class in Java stores information about classes in a program?
java.lang.Class
56. For what is the Java name extension .class used?
As a way to obtain the Class object of that class.
57. What does the Java getMethods method do?
Gets a method by name.
58. For what is the C# namespace System.Reflection.Emit used?
Tools for the emission of MSIL and other metadata.

6. Sebesta, problem set, 2, 6, 19 page 564-565.
   2. Explain the principle of abstraction.
   Abstraction is the process of creating a representation of an object or system that omits unnecessary details or attributes.

   6. Compare the multiple inheritance of C++ with that provided by interfaces in Java.
   Earlier versions of Java did not allow for interfaces to provide an implementation for a method, only a definition, so the problems caused by multiple inheritance in C++ could not occur. However, with the addition of default methods to Java interfaces, two different interfaces can provide different implementations of a method with the same signature. If this occurs the programmer must manually resolve this conflict by implementing the method on the class. More information about default methods: https://dzone.com/articles/interface-default-methods-java

   19. What are the differences between a C++ abstract class and a Java interface?
   Both instance variables, method definitions, and method implementations can be part of a C++ abstract class, however, a Java interface can only contain method definitions and method implementations but not instance variables.

7. Discuss how Tennent's principle may be used in your own language design
   Tennent's correspondence principle is a powerful programming language design guideline. It says that the meaning of an expression or statement should not change when an extra level of block structure is added around it.

   You can read more about Tennet's Principle here:
   https://fanf.livejournal.com/118421.html

   If your language contains block structures or other wrapping statements, e.g. closures, consider if this affects the meaning of an expression or statement.