Object Oriented Programming notes

Mikkel Helsing Andersen

September 13, 2023

Contents

Chapter 1

Programming concepts

A programming paradigm is a way of conceptualizing what it means to perform computation, and how tasks that are to be carried out on a computer should be structured and organized.

A program has two components data and algorithms.

Imperative paradigm: Also know as algorithmic paradigm, think C. No classes, references to memory, sequentiel running of program/algorithm.

Procedural paradigm: Same as Imperative except it's more modular, meaning code is easily reusable.

Declarative paradig: Computer finds the solution to a problem, think SQL. The data is "permanent".

Functional paradigm: Based on mathematical functions. Data is immutable (Underlying structure cannot be changed). Java allows for lambda functions which allows for creating functional programming.

Logic paradigm: Set the goal and specify the problem, to make the computer solve the problem, instead of writing an algorithm for finding said solution.

Object-oriented par: Objects contains the data and algorithms. Where data is the state of the object. The algorithms allows to change the state of an object. Objects are often described as classes in programming. Think Java, C# etc.

Polymorphism

Overloading Polymorphism

Multiple definition of a function with varying parameters.

```
public class MathUtil {
    public static int max(int n1, int n2) {
        /*code*/
    }
    public static double max(double n1, double n2) {
        /*code*/
    }
    public static int max(int[] num) {
        /*code*/
    }
}
```

Coercion Polymorphism

When a type is implicitly converted.

```
int num = 707;
double d1 = (double)num; // Explicit
double d2 = num; // Implicit
```

Inclusion Polymorphism

Subtype, meaning a type/class that extends or implements a type/class can also be assigned to it's parent type. Think employee extends person can also be assigned to a variable with type person.

Parametric Polymorphism

Lets classes and algorithms be generic without type specification Two conflicting imports

```
List<String> sList = new ArrayList<String>();

public <T> T example(T test) {
    return test;
}

class test<T> {
    /*code*/
}
```

can be used directly as such:

```
package pkg;
import p1.A;
import p2.A; // A compile-time error
class Test {
    A var1; // Which A to use p1.A or p2.A?
    A var2; // Which A to use p1.A or p2.A?
}
```

The alternative is stating it directly when declaring the type:

```
// Test.java
package pkg;
class Test {
    p1.A var1; // Use p1.A
    p2.A var2; // Use p2.A
}
```

Or:

```
// Test.java
package pkg;
import p1.A;
class Test {
    A var1; // Refers to p1.A
    p2.A var2; // Uses the fully qualified name p2.A
}
```

jav

Chapter 2

Classes

Chapter begins: **(P. 225[1])**

2.1 What is a Java Class

Java Classes has a set of properties and functionalities. Non-static classes can be instantiated. Classes can also work with generic types and inlusion as mentioned in polymorphism and inclusion respectivly. Include the following:

- Fields
- Methods
- Constructors
- Static initializers
- Instance initializers

Basis for creating a class:

```
// <T> is only for generic classes
[modifiers] class class-name <T> {
    // Body of the class goes here
}
// Example
public abstact class Human<T> {
    // An empty body for now
}
```

Static fields in a class are class fields, whereas none static fields are instance variables.

[1] Java Fundamentals

Imports

Import statements that uses the wild(*) modifier are on demand imports, meaning all available classes and functions can be used on demand:

```
import com.java.string.*
```

The Java compiler must resolve the simple name A to its fully qualified name during the compilation process. It searches for a type referenced in a program in the following order:

- The current compilation unit
- Single-type import declarations
- Types declared in the same package
- Import-on-demand declarations