STUDY GUIDE

OBJECT ORIENTED MODELING AND DESIGN

Design Principles

- SRP: Single Responsibility Principle
  - "A class should have only one reason to change." (Martin)
  - En klass med flera ansvarsomraden ger bracklighet.
  - En klass utan ansvar ar onodig.
- OCP: Open-Closed Principle
  - "Classes should be open for extension and closed for modification." (Meyer)
  - Det skall vara mojligt att lagga till ny funktionalitet utan att modifiera existerande kod.
- DIP: Dependency Inversion Principle
- DRY: Don't Repeat Yourself Principle
  - "Every piece of knowledge must have a single, unambiguous, authoritative representation within a system" (Hunt and Thomas)
- ISP: Inferface Segregation Principle
- LSP:

Design Patterns/Mnster

- Visitor
- Command
- Composite
- Template
- Strategy

1. Lecture 1

- Info
  - Points for Övning on Tenta!
- DelA: OMD objekt-modeling-design
  - Classes
  - Object
  - Abstraction
  - Encapsulation (Inkapsling)
- Graphs
  - Intro
    - a set of nodes and a set of bows:
      - { A , B , C }
      - { <A,B> , <B,C> , <C,A> }

        - ∗ weighted graphs
      - Representation
        - ∗ Matrix
        - ∗ Adjacency-list “Närhetslista”
      - Before Lab 1
        - ∗ Must implement our own data structures for Graphs
        - ∗ Make sure you understand the ADT Graph
        - ∗ construct
        - ∗ add Node
        - ∗ add Bow
        - ∗ Iterate over all nodes
        - ∗ Iterate over all bows
        - ∗ SOMETHING
  - This week
    - first 6 chapters
    - lab 1 redovisas 20/1
    - lab 2 and övning redovisas 25/1
  - Example:

```
//test
"string"
public class DiGraph{
   public Vertex firstVertex(){...}
   public void insertVertex(Vertex v){...}
   public void insertEdge(Vertex v , Vertex x , Edge e){...}

   public void iterateVertex(){
      Vertex v = firstVertex;
      while(v != null){
         v = v.nextVertex();
      }
   }

}

public class Vertex{
   public Vertex nextVertex(){...}
   public Edge firstEdge(){...}
   ...
}

public class Edge{
   public Edge nextEdge(){...}
   public Vertex endpoint(){...}
   ...
```

```
}
```

2. Lecture 2

- Agenda
  - Grafer (depth-first-traversal)
  - Object oriented modeling
- Depth-first
  - Grafer (depth-first-traversal)
  - Object oriented modeling

3. Lecture 3

3.1. principles.

- SRP
- OCP
- DIP
- DRY
- Theres discussion of a "Lokalitetsprincpen"

3.2. patterns.

- Visitor
- Command
- Composite

3.3. Pittfalls/smells.

- Rigidity "Stelhet" - The design is difficult to modify
- Fragility "Brcklighet"- The design doesn't handle modification
- Immobility "Orrlighet - Design prevents reuse and recycling
- Viscosity "Seghet"? maybe "Trghet" - It's more difficult to implement slick changes than it is to hack

3.4. Measurement of design quality.

- Coupling "Koppling" - The grade (level?) of dependency on other modulels in the system
- Cohesion "Sammanhang" - The grade (level?) of affenity/togetherness in the same moduel

3.5. Lokalitetsprincpen (Locality Principle?)

- Competition?
- Delegerar arbetet till den klass som vet allt om Race:
- Delegerar arbetet till den klass som vet allt om Result:

```
public class Competition {
   private ArrayList<Race> races;
   public void computeTotal() {
      for(Race race: races ) {
      race.computeTotal();
```

```
      }
   }
   ...
}
```

3.6. SRP - Enkelt ansvar.

- Single Responsibility Principle
- "A class should have only one reason to change."
- "A class with many responcibilities leads to fragility "brcklighet"
- A class without responsibility is uneccessary.
- Delegate the responcibilities.

```
public class Result {
   private Name name;
   private IdNumber idNumber;
   private Time start, end;

   public String fullName() {
      return name.toString();
   }

   public Time total() {
   return end.difference(start);
   }
}
```

3.7. OCP - Open to extention/Closed to modification.

- Det skall vara mojligt att lagga till ny funktionalitet utan att modifiera existerande kod.

```
// --------- GOOD ---------
public interface Expr{
   public int value();
}

public class Abs implements Expr {
   private Expr expr;
   public int value(){
      return Math.abs(expr.value()); }
   }


// --------- BAD ---------
public class Circle {
   int radius;
```

```
  int x, y;
  public Circle(int radius, int x, int y) {
     this.radius = radius; this.x = x;
     this.y = y;
  }
}
```

4. Lecture 4

- Template method
- Strategy
- State