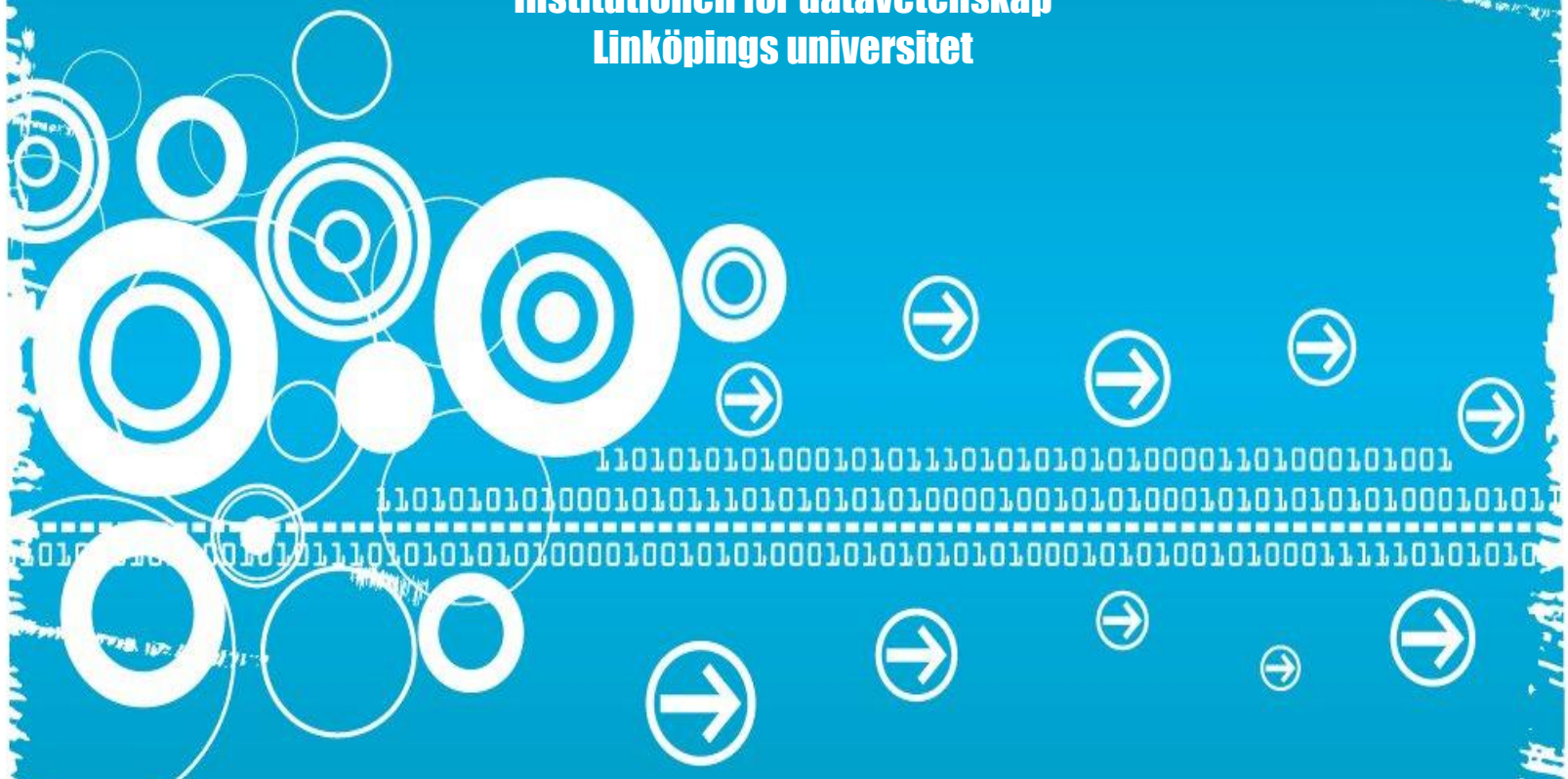
TDDC77 - Objektorienterad programmering

Laboration 4

Institutionen för datavetenskap
Linköpings universitet



4.1 MultiplicationTable*

I den här deluppgiften ska du deklarera en klass som heter `MultiplicationTable`. Klassen har en konstruktor som tar emot antalet rader och kolumner som argument. Det ska också finnas en default konstruktor som används när man vill använda default värdena på antalet kolumner och rader, 10. Det ska finnas en metod som heter `print()` som skriver ut följande tabell:

```
  * |  0  1  2  3  4  5  6  7  8
----+---------------------------
  0 |  0  0  0  0  0  0  0  0  0
  1 |  0  1  2  3  4  5  6  7  8
  2 |  0  2  4  6  8 10 12 14 16
  3 |  0  3  6  9 12 15 18 21 24
  4 |  0  4  8 12 16 20 24 28 32
  5 |  0  5 10 15 20 25 30 35 40
  6 |  0  6 12 18 24 30 36 42 48
  7 |  0  7 14 21 28 35 42 49 56
  8 |  0  8 16 24 32 40 48 56 64
  9 |  0  9 18 27 36 45 54 63 72
```

Alla kolumner i tabellen ska vara av samma bredd. Bredden ska vara större än bredden på det elementet som tar största platsen i tabellen, oavsett räknesätt (nästa delmoment).

Note: Se gärna följande tutorial om utskrift formatering:
https://dzone.com/articles/java-string-format-examples

Tabellen lagras som en 2D array varje gång klassen instansieras. Arrayen deklareras som ett fält med rätt access-modifier. Metoden `print` ska bara läsa igenom arrayen och skriva ut innehållet.

Antalet kolumner och rader ska lagras i två separata fält i klassen. Välj rätt access-modifier med inkapsling i åtanke. Varje fält ska ha egen setters och getters. Konstruktorerna får inte komma åt fälten direkt och ska använda setters-metoderna som finns.

Hela labben ska implementeras enligt inkapsling-regler.

4.2 Arv*

Anta att vi vill lägga till tabeller för fler räknesätt t.ex. addition. Man skulle då kunna tänka sig att vi har en abstract superklass `ArithmeticTable` som deklarerar en abstrakt metod `double evaluate(int, int)` som utför en aritmetisk heltalsoperation på sina två argument och returnerar resultatet.

Inför den nya superklassen och visa hur den ska användas genom att deklarera subtyperna `MultiplicationTable`, `AdditionTable`, `SubtractionTable` och `DivisionTable`. Hur kan vi lösa problemet med utskrift av rätt operatortecken och rätt utskriftsbredd på ett smidigt sätt?

4.3 Komposition*

Istället för arv kan vi lösa föregående deluppgiften med komposition. Istället har `ArithmeticTable` en metod/konstruktor som tar emot ett objekt som är ansvarigt för att beräkna cellvärdena. Inför först följande interface:

```
public interface Operation {
   public char symbol();
   public int width(int rows, int cols);
   public double evaluate(int a, int b);
}
```

Nu kan vi byta konstruktoren till `ArithmeticTable` till:

```
public ArithmeticTable(Operation op, int r, int c);
```

`ArithmeticTable` är inte abstract längre.

Istället anropar metoden `evaluate` som ligger i `ArithmeticTable` den metoden `evaluate` på den inkapslade operationen. Instansiera ett antal objekt av typen `Operation`: `addition`, `substraction`, `multiplication`, `division`.

Arv kan användas för att skapa ovannämnda Operationerna eller man kan instansiera interface:et `Operation` fyra gånger genom att använda *anonymous classes*.

Exempelkod:

```
//AdditionOperation implementerar från interfacet Operation.
AdditionOperation addition = new AdditionOperation(/* kan finnas
argument om det behövs*/);
ArithmeticTable tab = new ArithmeticTable(addition, 9, 8);
tab.evaluate(7, 9); //addition
```

4.4 Enumeration*

Skapa en enumeration som heter `EnumOperation` som implementerar `Operation` och definierar värdena: `ADDITION`, `SUBTRACTION`, `MULTIPLICATION`, `DIVISION`. Utgå från följande mall:

```
public enum EnumOperation implements Operation {
      // Er kod ska in här
      //
      //

      private final char symbol;

      private EnumOperation(char symbol) {
          this.symbol = symbol;
      }
      public char symbol() {
          return symbol;
      }
}
```

Varje värde ska implementera/override:a `evaluate()` på eget sätt.