

# Java-koncept och JDBC

Separera data från logik

# Data från databas

Säg att vi har en enkel databas över

- students (id, name)
- courses (id, course\_id)
- registrations(student\_id, course\_id)

```
CREATE TABLE students(id integer primary key asc,name varchar(30));
CREATE TABLE courses
        (id integer primary key asc,course_code varchar(12));
CREATE TABLE registrations        (student_id int(8), course_id int(8));
```

# Innehåll i databasen

Students:

id|name

1|Anders Andersson

2|Beata Bengtsson

3|Charlie Christensen

4|Dick Dale

5|Edward Eriksson

Courses:

id|course\_code

1|TIG015

2|TIG058

Registrations:

student\_id|course\_id

1|1

1|2

2|2

3|1

4|1

4|2

5|2

(Dick Dale är reggad  
på bägge kurserna)

# Hur vi använder databasen från Java

Uppgift: skriv ett program som tar som argument ett student-id och skriver ut alla kurskoder för de kurser studenten är registrerad på.

```
"SELECT courses.course_code FROM students, courses " +  
    "JOIN registrations ON " +  
    "registrations.student_id = students.id and " +  
    "registrations.course_id = courses.id and " +  
    "students.id = "+studentID;
```

# Första idén - vi löser allt direkt!

```
import java.sql.*;
//import en himla massa andra paket för allt möjligt...
public class OneSizeFitsAll{
    void initDB(){...}
    void printCourses(String id){
        // en massa kod med SQL och allt möjligt
    }
    public static void main(String[] args){
        new OneSizeFitsAll().printCourses(args[0]);
    }
}
```

# Uppgiften ändras

Nu så ska ni göra samma sak men istället för databas så ska ni läsa kurserna från en fil...

Ni får skriva om hela klassen (det ska inte importeras sql-grejer, printCourses() måste skrivas om från grunden osv.

# Hur kan vi skapa en bättre lösning?

Genom att separera databasen från den kod som hämtar och skriver ut kurserna får vi en lösning som är lättare att:

- Läsa
- Skriva och underhålla
- Byta ut delar i

# Förslag på lösning - 1 - databasen

Skapa en databasklass som initierar databasen, låter er ställa SQL-frågor och allt annat som har med databasen att göra.

```
public class DBUtils{  
    // T ex:  
    public static ResultSet executeQuery(String query){  
        // kod...  
    }  
}
```



# Förslag på lösning 2 - Data för sig

Skapa ett interface som specificerar metoder för den data som ska behandlas:

```
package student.data;
import java.util.ArrayList;

public interface StudentData{
    public ArrayList<String>getCourses(String studentID);
    public String getStudentName(String studentID);
    public boolean studentIDExists(String studentID);
    public String getCourseCode(String courseID);
}
```

OBS! Detta interface vet inget om ***hur*** data är lagrat!

# Förslag på lösning 3 - specialisering

Skapa en implementation av interfacet för en databaslösning:

```
public class StudentDataDB implements StudentData{  
    // Implementera alla metoder genom att använda  
    // DBUtils och databasen, t ex:  
    public ArrayList<String>getCourses(String studentID){  
        // kod som via DBUtils hämtar kurser från DB  
    }  
}
```

OBS! Denna klass, behöver bara importera ResultSet för att lösa uppgiften!

# Förslag på lösning 4 - Programmet

```
public class StudentExample{
    public static void main(String[] args){
        if(args.length!=1){ System.exit(-1); }
        String studentID = args[0];
        StudentData data = new StudentDataDB();
        if(! data.studentIDExists(studentID) ){
            System.out.println("No student found with ID " + studentID);
            System.exit(0);
        }

        System.out.println("Courses for " +
            data.getStudentName(studentID)+" (id: " + studentID+":");
        for(String course : data.getCourses(studentID)){
            System.out.println(course);
        }
    }
}
```

# Lösningen - example run

```
$ javac student/StudentExample.java && java student.StudentExample 4  
Courses for Dick Dale (id: 4):  
TIG015  
TIG058
```

# Förslag på lösning 4 - Kodan

```
public class StudentExample{
    public static void main(String[] args){
        if(args.length!=1){ System.exit(-1); }
        String studentID = args[0];
        StudentData data = new StudentDataDB(); // db-version!
        if(! data.studentIDExists(studentID) ){
            System.out.println("No student found with ID " + studentID);
            System.exit(0);
        }

        System.out.println("Courses for " +
            data.getStudentName(studentID) +" (id: " + studentID+":");
        for(String course : data.getCourses(studentID) ){
            System.out.println(course);
        }
    }
}
```

Mer behövs inte!

# Uppgiften ändras

Chefen bestämmer att filer ska användas i stället för DB.

Vad behöver vi göra?

- Skapa en ny klass som implementerar StudentData men som använder filer i stället.

Hur mycket behöver vi ändra i programmet?

# Ändringar i själva programmet

```
public class StudentExample{
    public static void main(String[] args){
        if(args.length!=1){ System.exit(-1); }
        String studentID = args[0];
        StudentData data = new StudentDataFiles(); // fil-version!
        if(! data.studentIDExists(studentID) ){
            System.out.println("No student found with ID " + studentID);
            System.exit(0);
        }

        System.out.println("Courses for " +
            data.getStudentName(studentID)+" (id: " + studentID+"):");
        for(String course : data.getCourses(studentID)){
            System.out.println(course);
        }
    }
} // Allt annat är samma som i förra programmet!
```

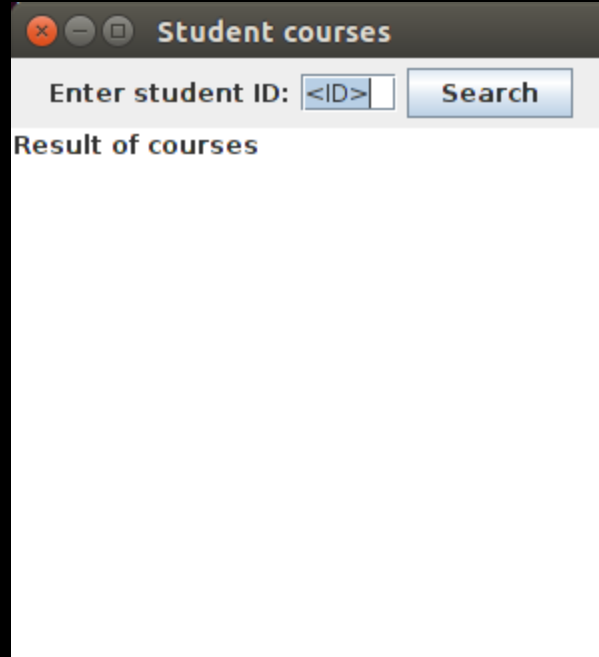
# Fler fördelar med separerad lösning

Säg att vi ska ha ett GUI (fönstergränssnitt) i stället för ett kommandoradsgränssnitt.

Hur skulle ett sådant program kunna se ut?



# GUI-version



# GUI-version

main:

```
public static void main(String[] args){  
    StudentSwing app = new StudentSwing();  
    app.initApplication();  
}
```

# GUI-version - fönsterklassen

## StudentSwing:

```
public class StudentSwing{  
    private StudentData data = new StudentDataDB();  
    private JFrame frame;  
    private JLabel label;  
    private JTextField input;  
    private JButton search;  
    private JList result;  
    private DefaultListModel<String> resultModel;  
    // methods...  
}
```

# GUI-version - fönsterklassen

```
private void initApplication(){
    frame = new JFrame("Student courses");
    label = new JLabel("Enter student ID:");
    input = new JTextField(4); // four chars wide
    input.setText("<ID>");      // hint to the user what to type
    input.selectAll(); // select the text
    search = new JButton("Search");
    search.setToolTipText("Find the courses for this student");
    resultModel = new DefaultListModel<String>();
    result = new JList<String>(resultModel);
    resultModel.addElement("Result of courses");
    frame.setSize(300,300);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    addListeners();
    placeComponents();
    frame.setVisible(true);
}
```

# GUI-version - fönsterklassen

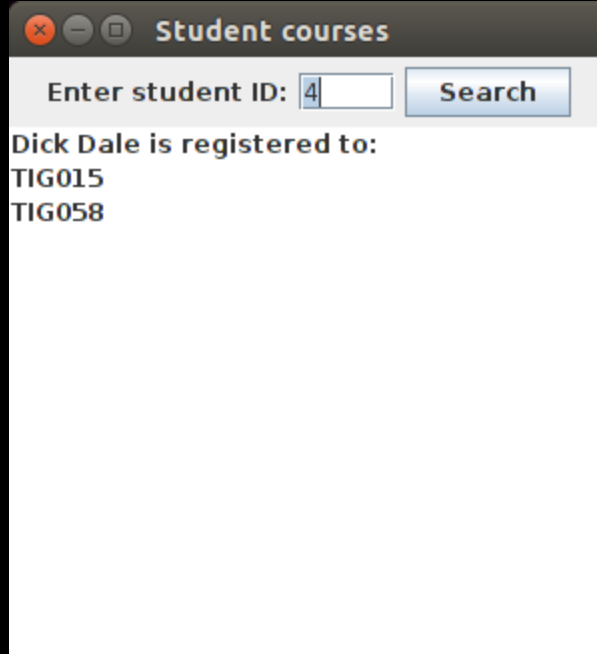
```
public void addListeners(){
    SearchListener searchListener = new SearchListener();
    search.addActionListener( searchListener );
    input.addActionListener( searchListener );
}
```

```
private void placeComponents(){
    frame.setLayout(new BorderLayout());
    JPanel north=new JPanel();
    north.add(label);
    north.add(input);
    north.add(search);
    frame.add(north, BorderLayout.NORTH);
    frame.add(result, BorderLayout.CENTER);
}
```

# GUI-version - fönsterklassen

```
private class SearchListener implements ActionListener{
    public void actionPerformed(ActionEvent e){
        String id=input.getText();
        resultModel.removeAllElements(); //clear list
        if(data.studentIDExists(id) ){
            resultModel.addElement
                (data.getStudentName(id) +" is registered to:");
            for(String course : data.getCourses(id) ){
                resultModel.addElement(course);
            }
            input.requestFocus(); //Focus input field
            input.selectAll();    //Select text
        }else{
            resultModel.addElement
                ("No such ID for a student exists.\nTry another.");
        }
    }
}
```

# GUI-version - visa resultat



A screenshot of a graphical user interface (GUI) window titled "Student courses". The window has a standard macOS-style title bar with red, yellow, and green window control buttons. Below the title bar, there is a text input field labeled "Enter student ID:" containing the number "4", and a blue "Search" button to its right. Below the search bar, the text "Dick Dale is registered to:" is displayed, followed by two lines of course identifiers: "TIG015" and "TIG058".

Student courses

Enter student ID: 4 Search

Dick Dale is registered to:  
TIG015  
TIG058

# Nästa avsnitt - MVC

Den uppmärksamme studenten såg att vår GUI-klass var en klass med “high coupling”, det vill säga samma klass innehöll

- Koppling till databasklassen (domain/modell)
- Grafisk layout
- Logik för hämtning av data



# MVC

Även en GUI-applikation kan separera ansvar mellan olika “lager” eller komponenter.

Mer om detta i MVC-föreläsningen