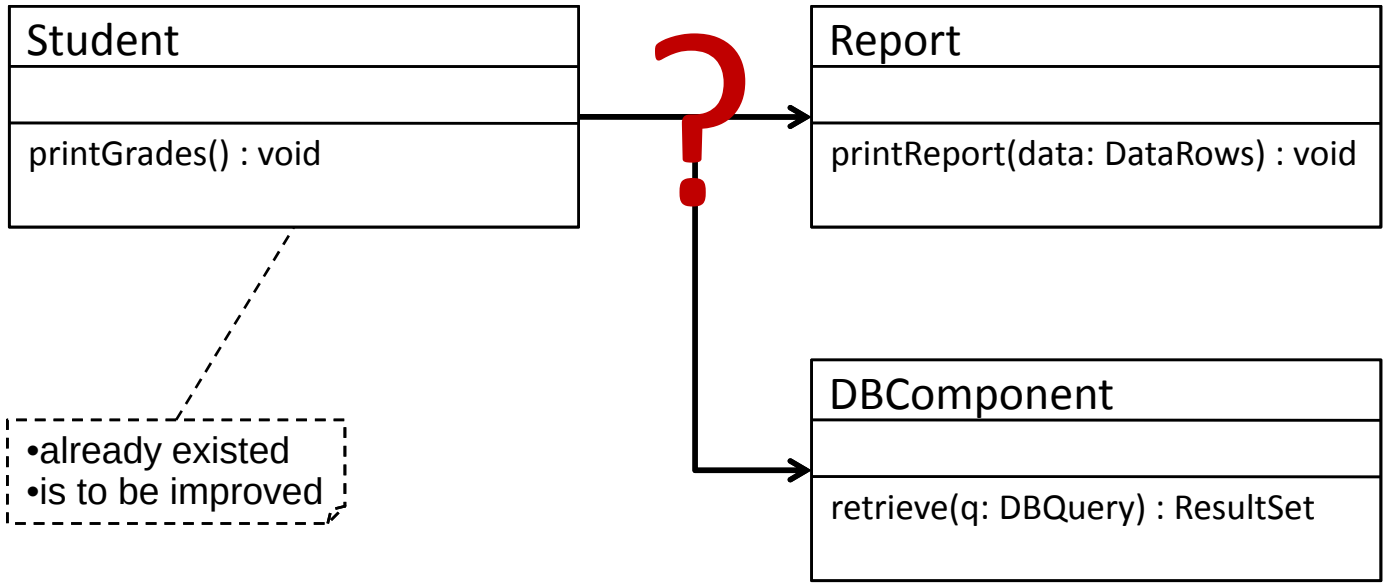
TECHNISCHE UNIVERSITÄT DRESDEN

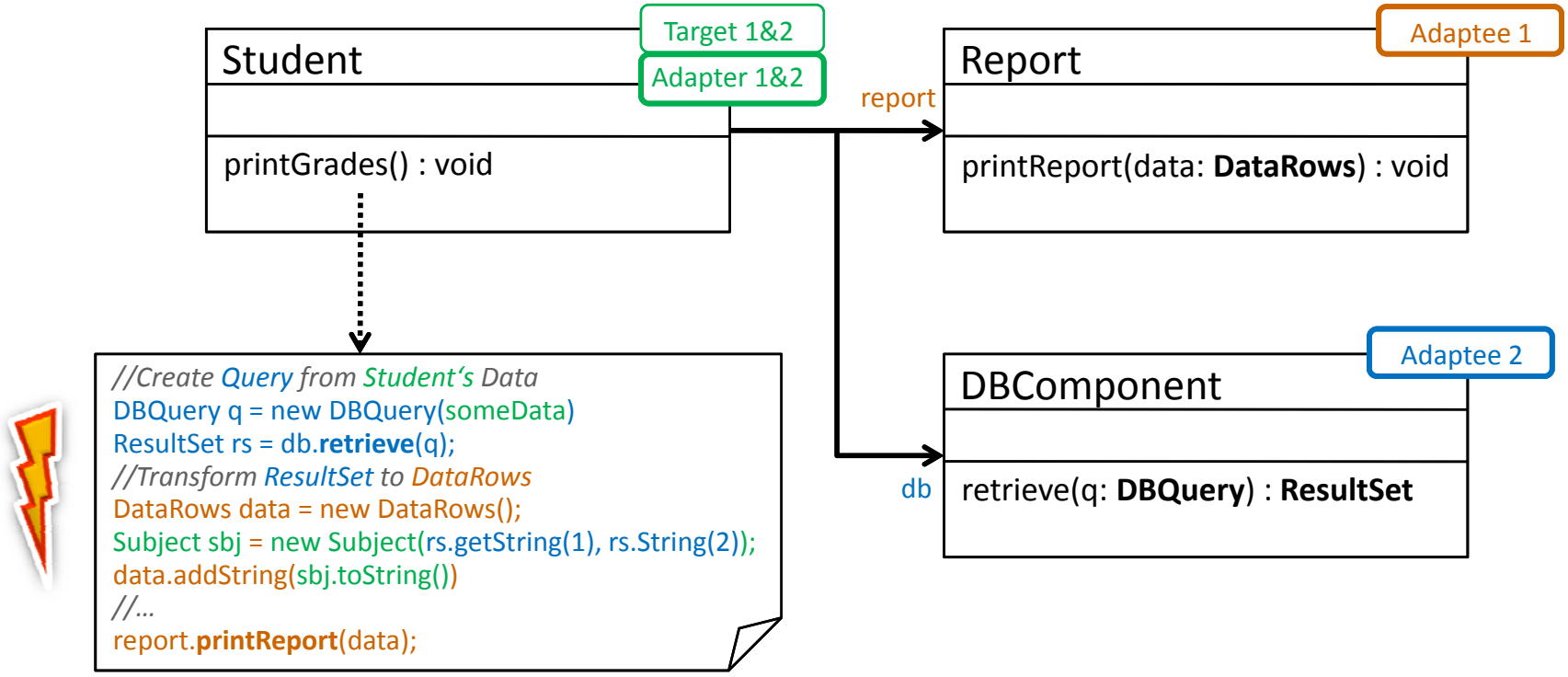
Exploring Role-Based Adaptation

Sebastian Götz, intermediate defense “Großer Beleg” 29.05.08

1. **Motivation**
2. **Key idea [5]**
3. **Realization**
4. **Discussion**
5. **Future Work**

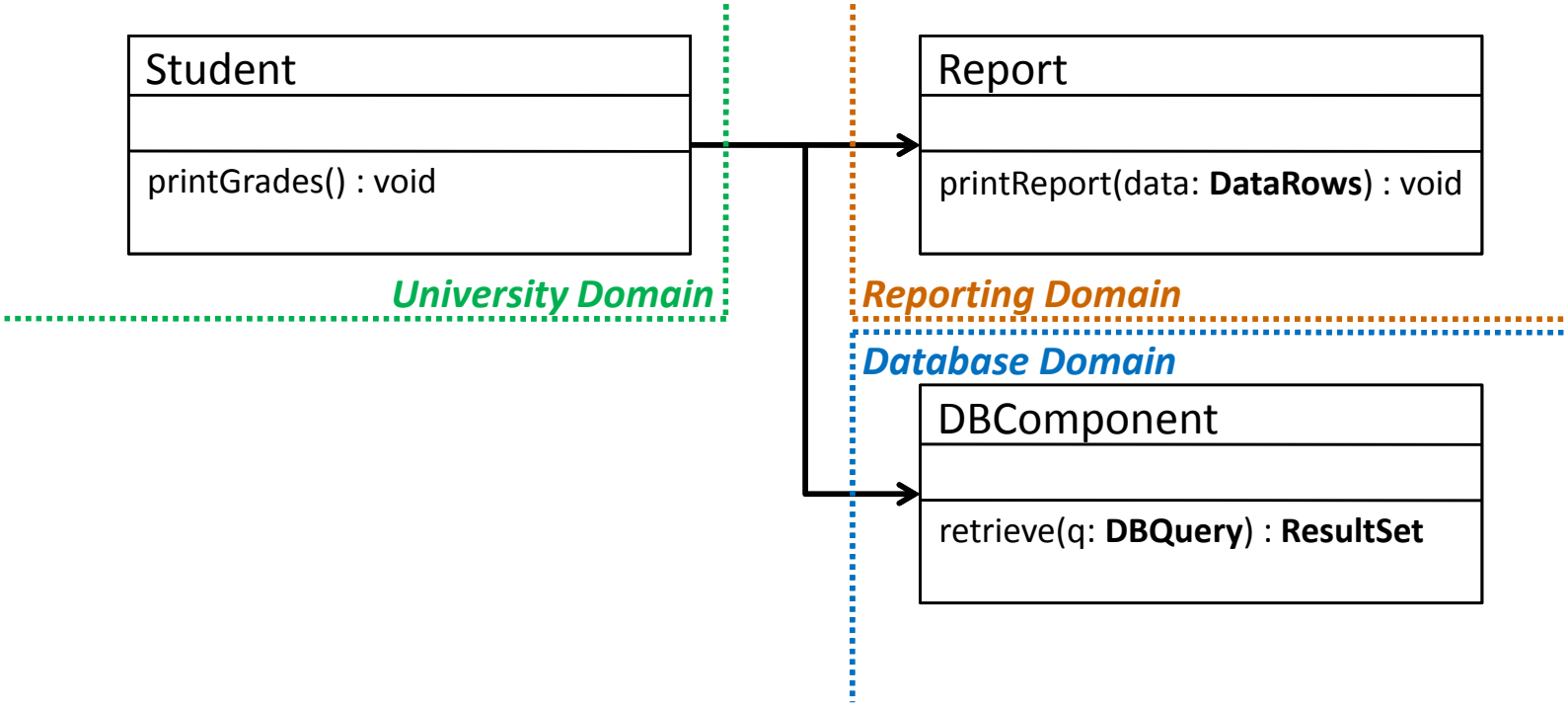

Student
printGrades() : void
?
Report
printReport(data: DataRows) : void
DBComponent
retrieve(q: DBQuery) : ResultSet
•already existed
•is to be improved

➔ Adapter Design Pattern [1] applied twice
➔**intertwined code**

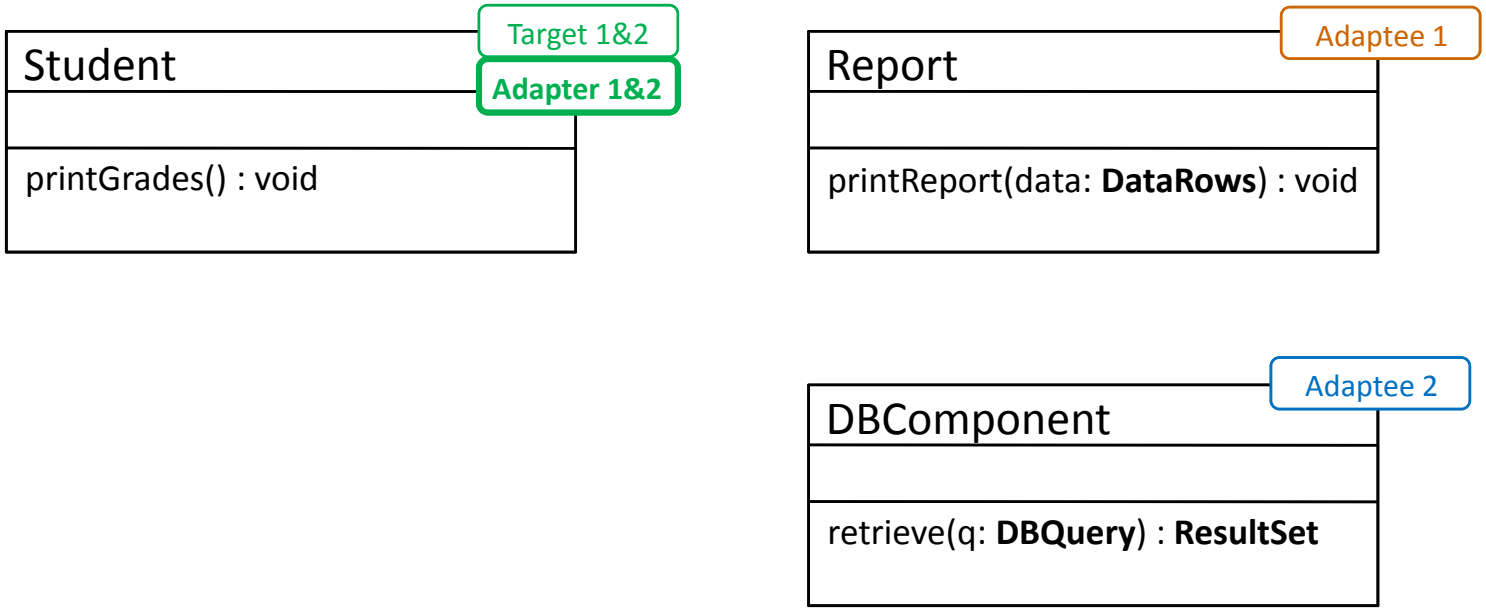


Roles annotated as in [2]

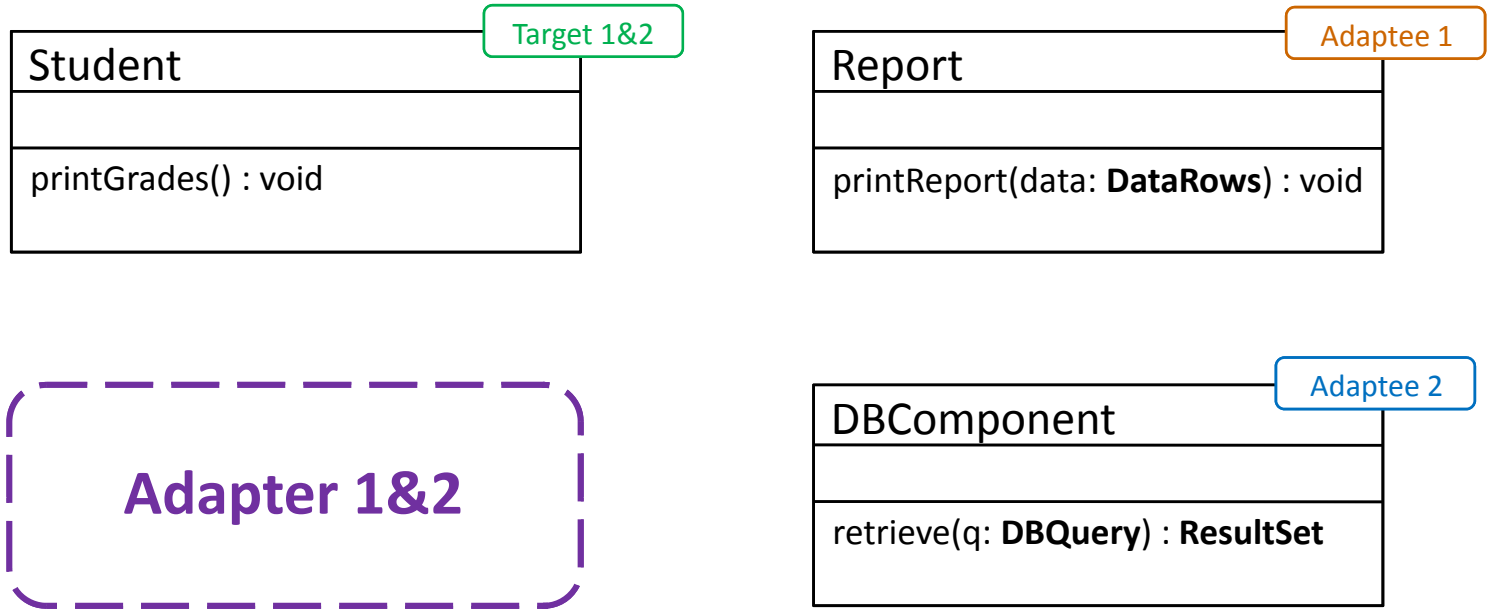
➔ **expert of 3 domains for printGrades() needed**
➔ **and experts leave**



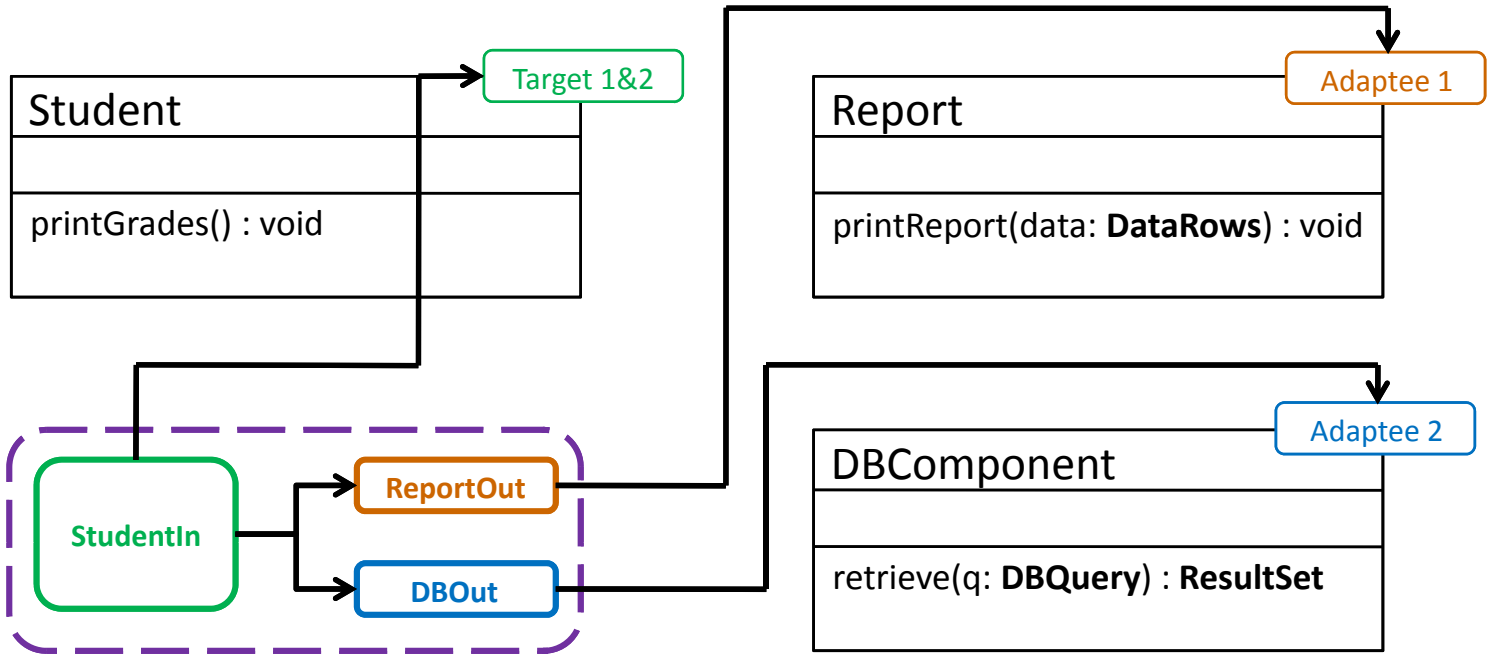
➔ implement adapter roles as firstorder programming constructs



➔ **implement adapter roles as firstorder programming constructs**

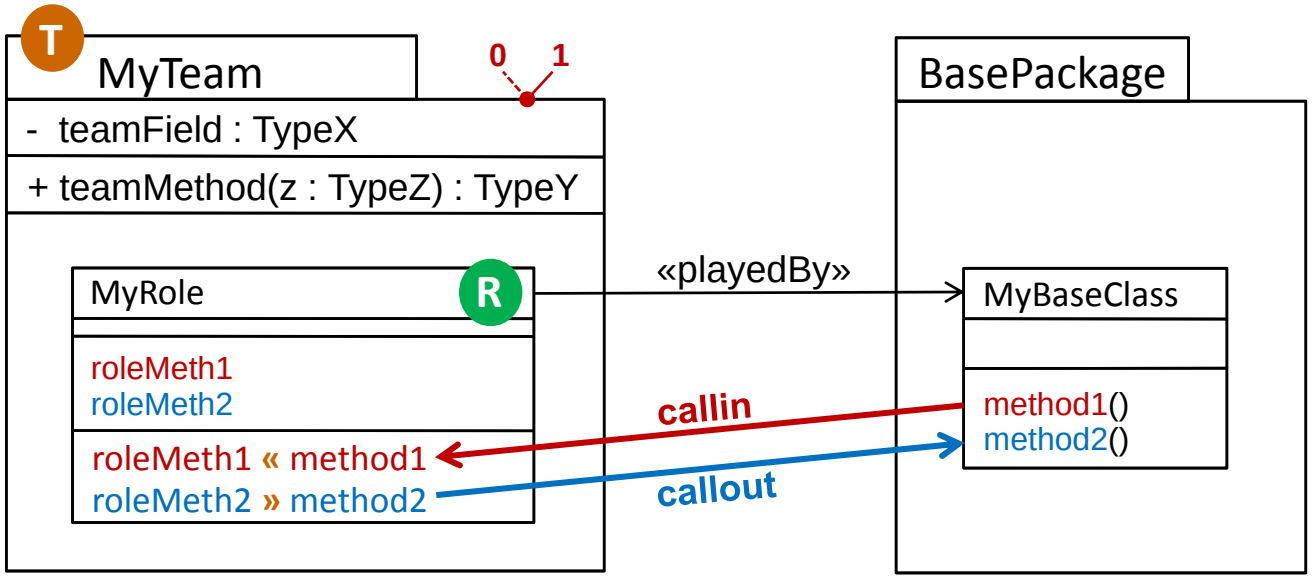


➔ split adapter role

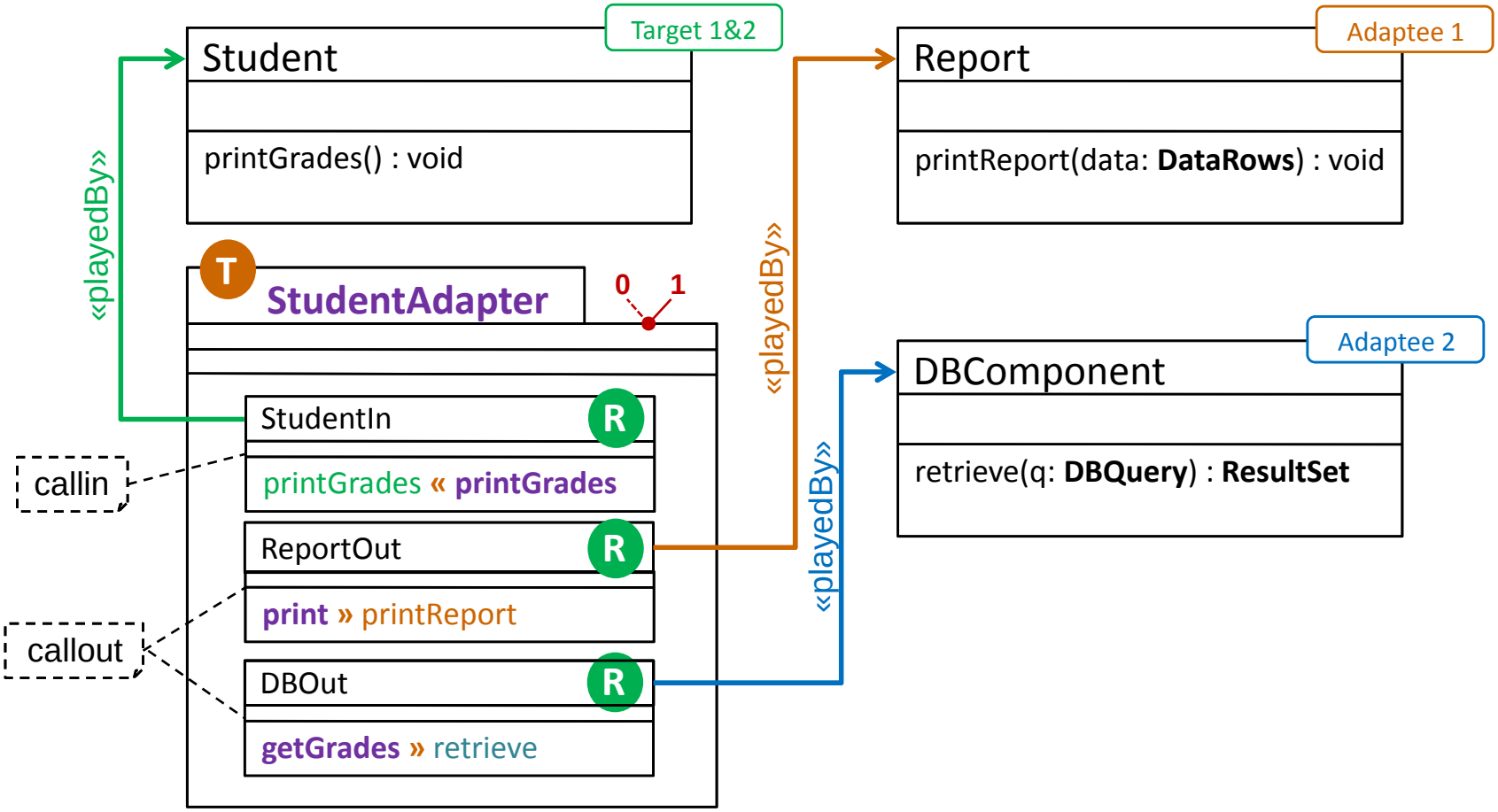


➔ separation of concerns on the level of implementation

➔ use ObjectTeams [3]
➔ **roles** are played by **actors** in a **context** [4, p. 66]



➔ introduce intermediate types/methods to decouple



```
public team class StudentAdapter {
    private ReportOut report;
    private DBOut db;
    private Map<String,Double> grades;

    public StudentAdapter(Report report, DBComponent db) {
        this.report = new ReportOut(report);
        this.db = new DBOut(db);
    }

    public class StudentIn playedBy Student {
        printGrades <- replace printGrades;
        public callin printGrades() {
            grades = db.getGrades();
            report.print(grades);
        }
    }

    //...
```

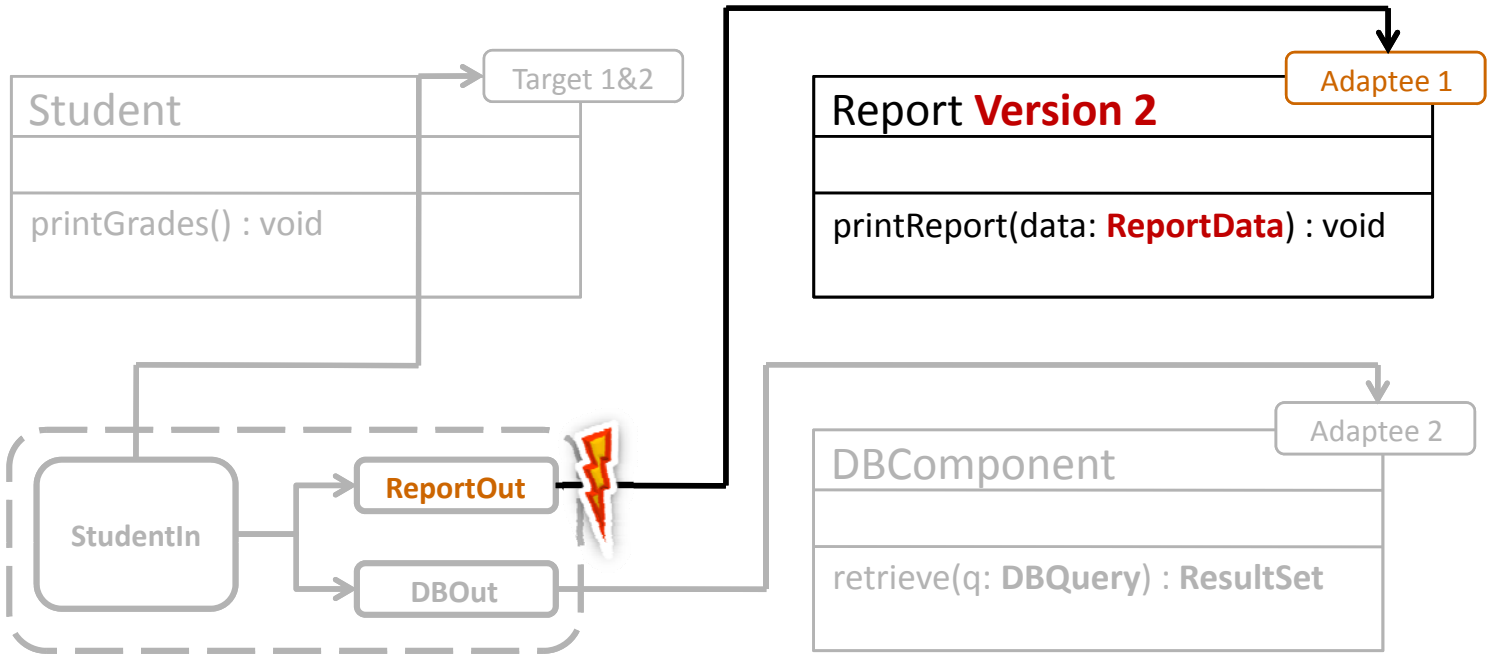
Intermediate Type
Subject + Grade

```
public class DBOut playedBy DBComponent {
    Map<String,Double> getGrades()
        -> ResultSet retrieve(DBQuery q)
        with { new DBQuery(„SELECT * FROM Grades“) -> q,
               result <- transformToMap(result) }

    Map<Subject,Grade> transformToMap(ResultSet rs) {
        Map<String,Double> ret = new HashMap();
        while(rs.next() != null) {
            ret.put(rs.getString(1), rs.getDouble(2));
        }
        return ret;
    }
}
```

```
public class ReportOut playedBy Report {
        void print(Map<String,Double> map)
            -> void printReport(DataRows rows)
            with { transformToDataRows(map) -> rows }

        private DataRows transformToDataRows(Map<String,Double> map) {
            DataRows ret = new DataRows();
            for(String subject : map.keySet()) {
                ret.addString(subject+ ' - ' +map.get(subject));
            }
            return ret;
        }
    }
```

- **separate roles for separate concerns**
  - Hence, no more tangling code
  - Easier extensible (just add a new role)
  - intermediate types decouple domains
  - **lower maintenance costs**
- **roles enable unanticipated adaptation**

- **evolution of integrated components → role composition**



- **Upgrade ReportOut invasively ?**
- **Generate additional ReportOut (from change information)**
- **and compose with old ReportOut !**

[1] Gamma, E., Helm, R., Johnson, R., Vlissides, J.: **Design Patterns: Elements of Reusable Object-Oriented Software.** Addison-Wesley, Reading, Massachusetts (1995)

[2] Riehle, D.: **Framework Design – A Role Modeling Approach.** PhD-Thesis, Swiss federal institute of technology, Zurich. 2000.

[3] Herrmann, S., Hundt, C., Mosconi, M.: **ObjectTeams/Java Language Definition - version 1.0.** Technical Report 2007/03, Technical University Berlin (2007)

[4] Steimann, F.: **Formale Modellierung mit Rollen.** Habilitationsschrift, Universität Hannover. 2000

[5] Götz, S., Savga, I.: **Exploring Role-Based Adaptation.** To Appear in Proceedings of RAM-SE'08

Thank you very much for your attention.

Any Questions ?