Software Engineering Essentials

Basics of Object Oriented Programming #2

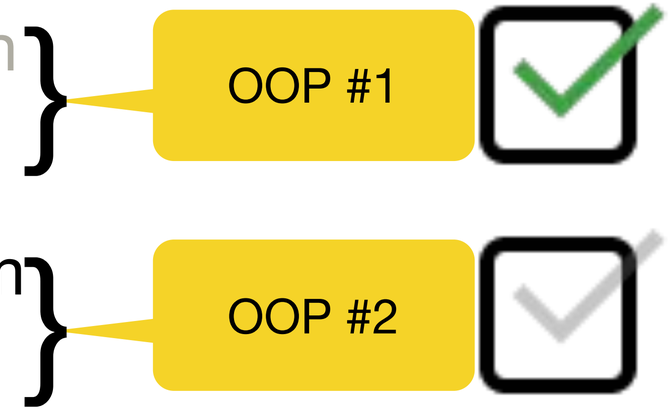
Bernd Bruegge, Stephan Krusche, Andreas Seitz, Jan Knobloch
Chair for Applied Software Engineering — Faculty of Informatics

Learning goals

1) Understand the object oriented programming principles polymorphism and abstraction
2) Apply these two principles

Object oriented programming principles

- Encapsulation
- Inheritance
- Polymorphism
- Abstraction



Polymorphism

- Extends inheritance
- Allows to modify ***functionality*** by *overriding* methods/procedures of a ***super-class***

Method signatures

A method is defined/identified by its ***method signature:***

In Java: A *method signature* is composed of:
the ***method name*** and all *input parameters* *(including their type and order)*

Examples of method signatures:

```
public void printPersonalInformation()
public void printPersonalInformation(String information)
public void printPersonalInformation(int age, String name)
public void printPersonalInformation(String name, int age)
```

Overloading

```
public void printPersonalInformation()
```

Not allowed

Method signatures

Overriding: defining **methods** with the **same signature**

- **Overriding** inside *one* class is not possible
- **Overriding** in a class hierarchy is possible: this is called ***Polymorphism***

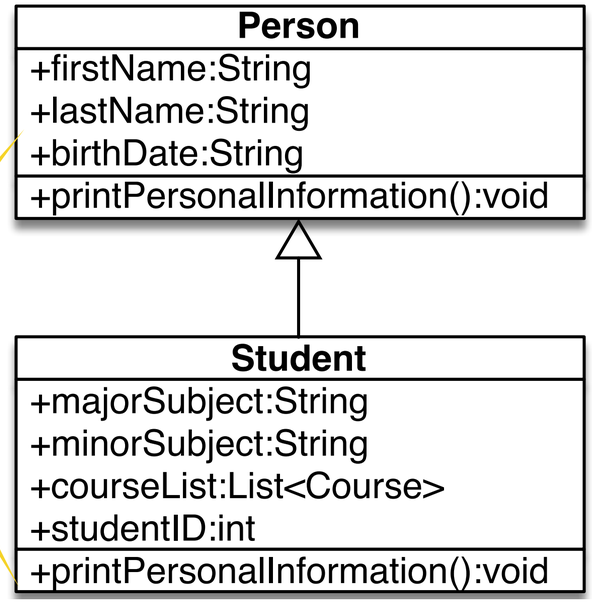
Examples of method signatures:

Person class:

```
public void printPersonalInformation()
```

Student class:

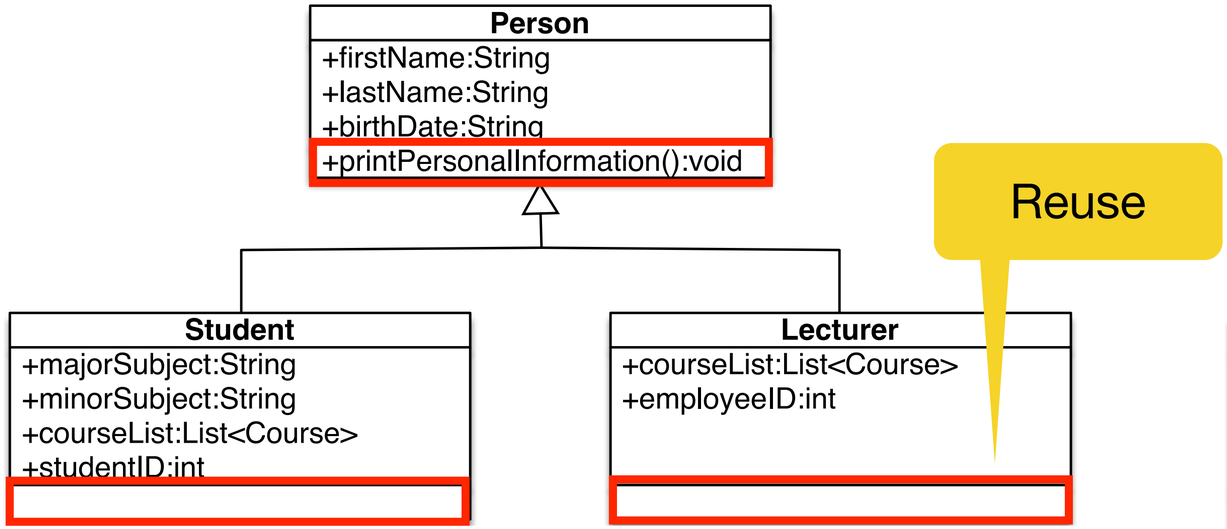
```
public void printPersonalInformation()
```



Inheritance vs. polymorphism

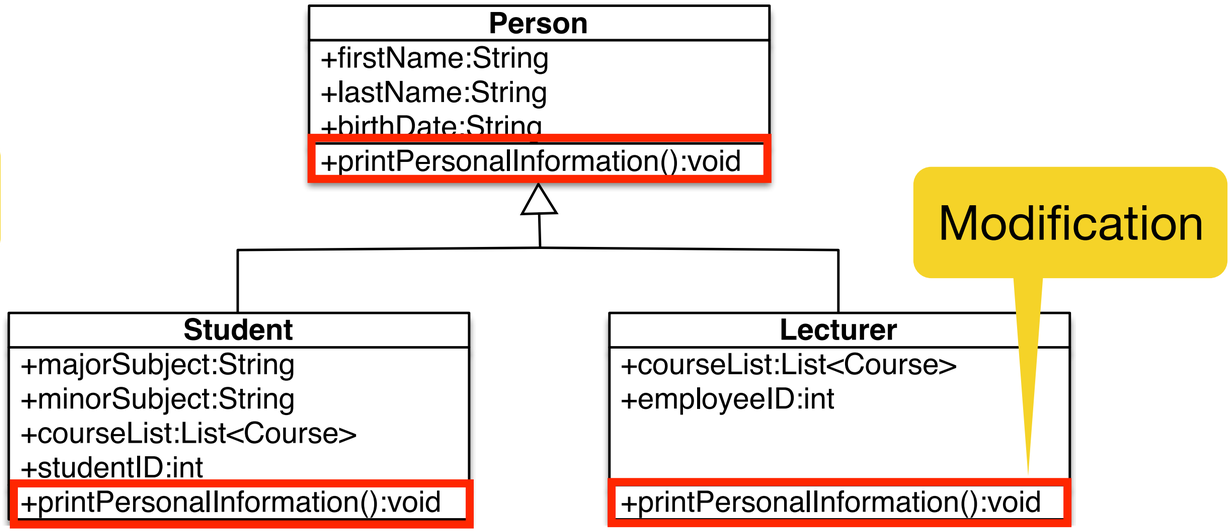
Inheritance

Student and **Lecturer** reuse the method **printPersonalInformation()** from their super-class ***Person***



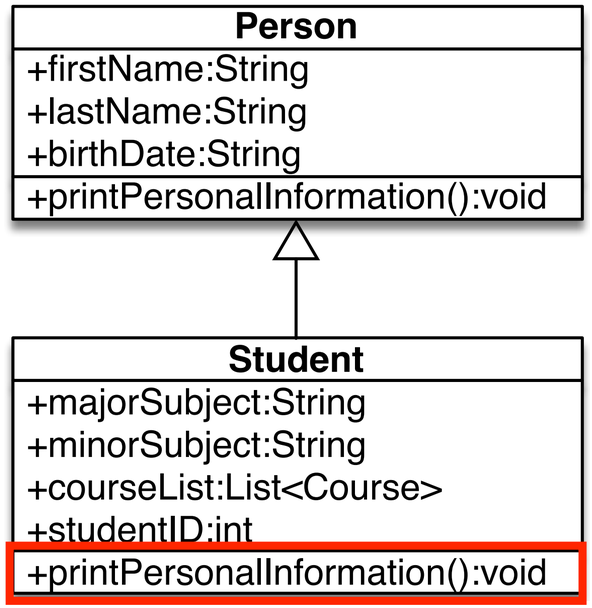
Polymorphism

Student and **Lecturer** override (modify) the method **printPersonalInformation()** and use their own implementation



Polymorphism

- We use the keyword ***extends*** to establish an inheritance hierarchy
- We override an existing method of the super-class inside of a sub-class




```
public class Student extends Person {
  public String majorSubject;
  public String minorSubject;
  public List<Course> courseList;
  public int studentID;

  @Override
  public void printPersonalInformation() {

    System.out.println(firstName+" "+lastName+" "+birthDate+" "+studentID);
  }
}
```

optional compiler information

Static vs dynamic types (example)

```
public static void main(String[] args) {

    Person s = new Student("Bob", "Davis", "12/20/1990", "Computer Science", "Commerce");
    Person l = new Lecturer("John", "Lewis", "1/23/1960");

    s.printPersonalInformation();   //this calls printPersonalInformation() from Student
    l.printPersonalInformation();  //this calls printPersonalInformation() from Lecturer
}
```

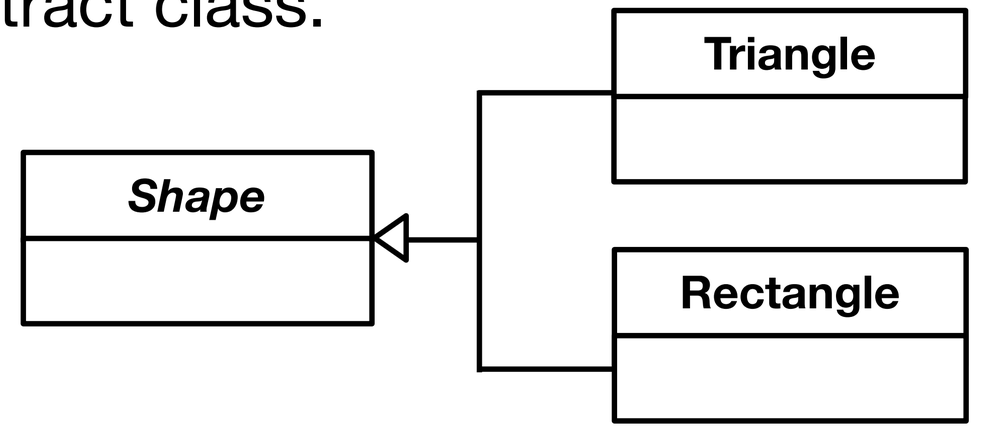
Even if the static type of both variables is ***Person***, the Java compiler determines the dynamic type (***Student*** or ***Lecturer***) and calls the individual overridden methods of each sub-class at run time

Abstraction

Allows to define an ***abstract*** structure which holds common states, behaviors or attributes

Abstract classes define what needs to be shared among sub-classes, yet it is not possible to create an instance of an abstract class.

Real world example: Geometric shapes



Abstract classes in Java can be even more restrictive forcing sub-classes to implement/override a specified method inside the abstract super-class

Abstraction

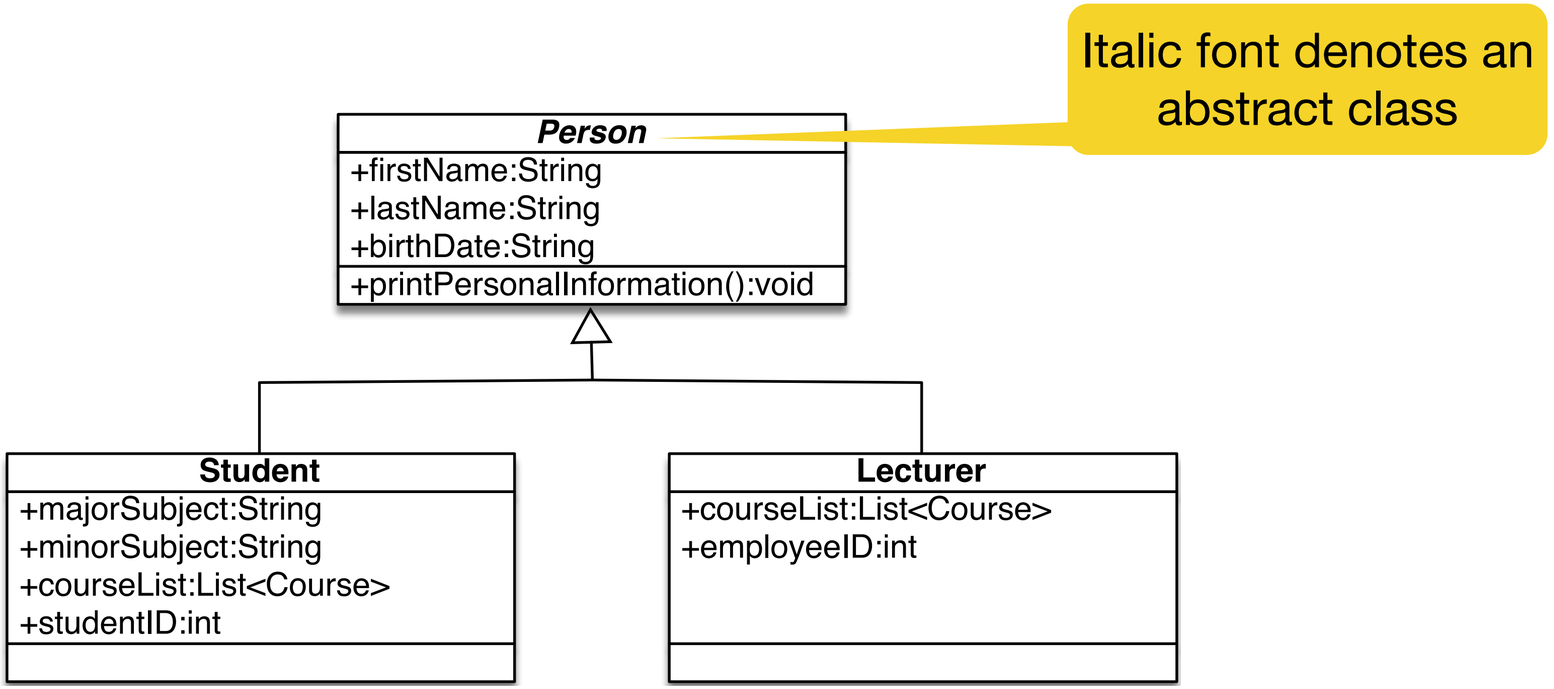
- Use the keyword ***abstract*** to create an abstract class definition
- Use the keyword ***abstract*** on methods to force sub-classes to override methods of the abstract super-class (i.e. actually implement them)

```
public abstract class Person {
	public String firstName;
	public String lastName;
	public String birthDate;

	public Person(String firstName, String lastName, String birthDate) {
		this.firstName = firstName;
		this.lastName = lastName;
		this.birthDate = birthDate;
	}

	public abstract void printPersonalInformation();
}
```

defining a contract for potential sub-classes

Abstraction (example)

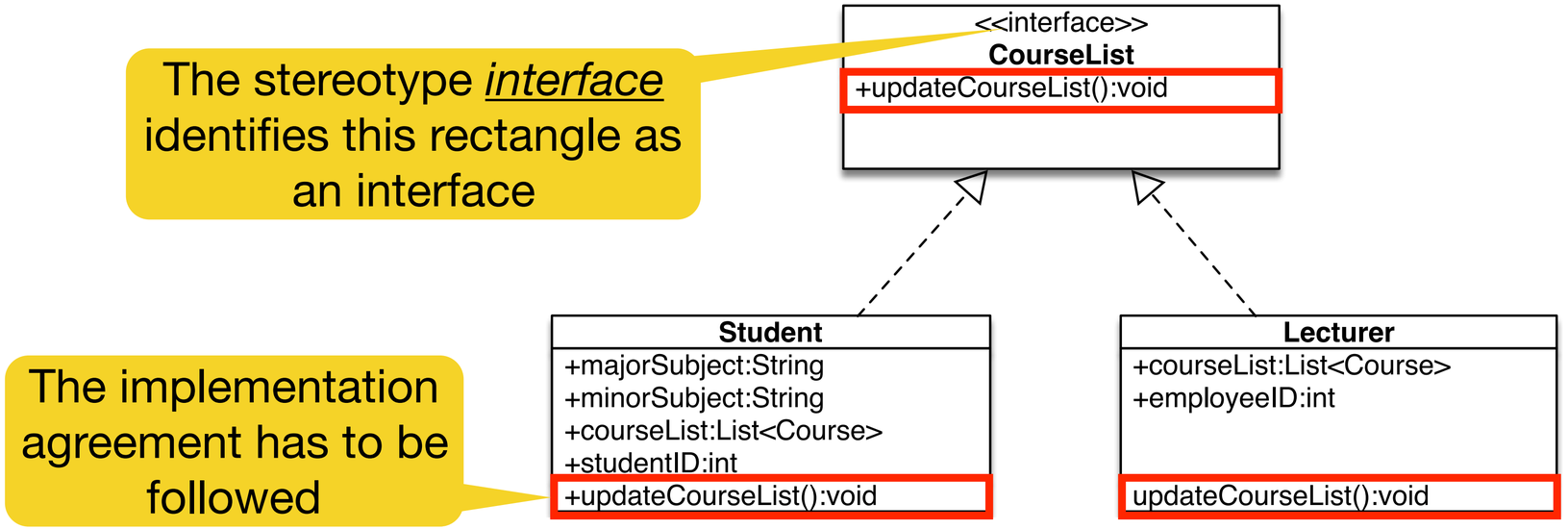
- Person is the abstract super-class
- Student and Lecturer are sub-classes, can be instantiated and can invoke methods on the abstract super-class Person



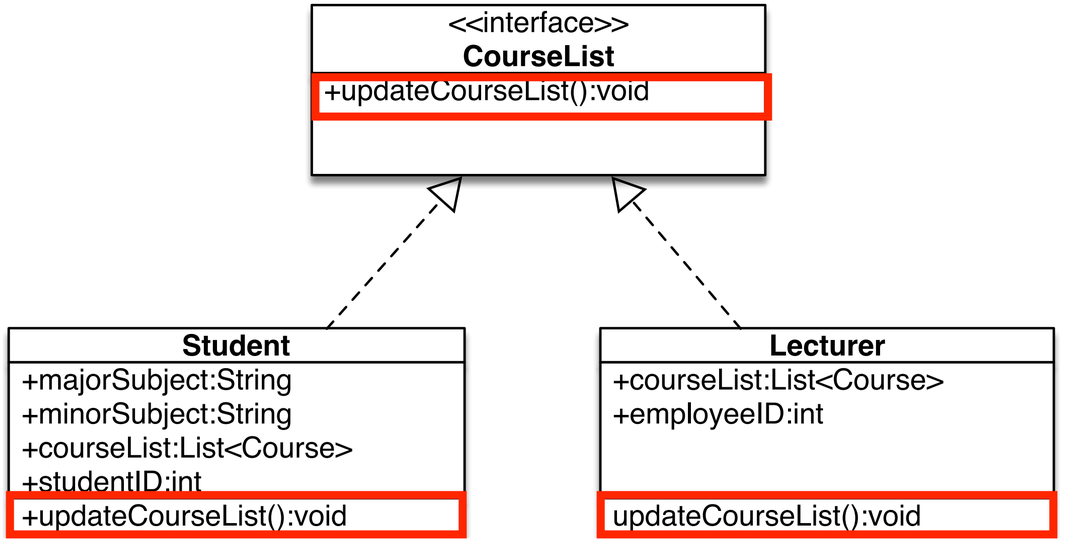
Abstraction with interfaces

Java also allows to specify i**nterfaces** to use the concept of abstraction:

- *Interfaces* specify a ***contract*** on ***functionality*** (methods) to be implemented by all classes that confirm to the interface
- Interfaces cannot specify a structure using attributes (however you can define constants)



Abstraction with interfaces




```java
public interface CourseList {

      public void updateCourseList();
}
```

```java
public class Student implements CourseList {
      public String majorSubject;
      public String minorSubject;
      public List<Course> courseList;
      public int studentID;

     @Override
      public void updateCourseList() {
      }
}
```

```java
public class Lecturer implements CourseList {
      public List<Course> courseList;
      public int employeeID;

     @Override
      public void updateCourseList() {
      }
}
```

Summary of object oriented programming basics

- Encapsulation
- Inheritance

OOP #1

- Polymorphism
- Abstraction

OOP #2

Software Engineering Essentials

Basics of Object Oriented Programming #2

Bernd Bruegge, Stephan Krusche, Andreas Seitz, Jan Knobloch
Chair for Applied Software Engineering — Faculty of Informatics