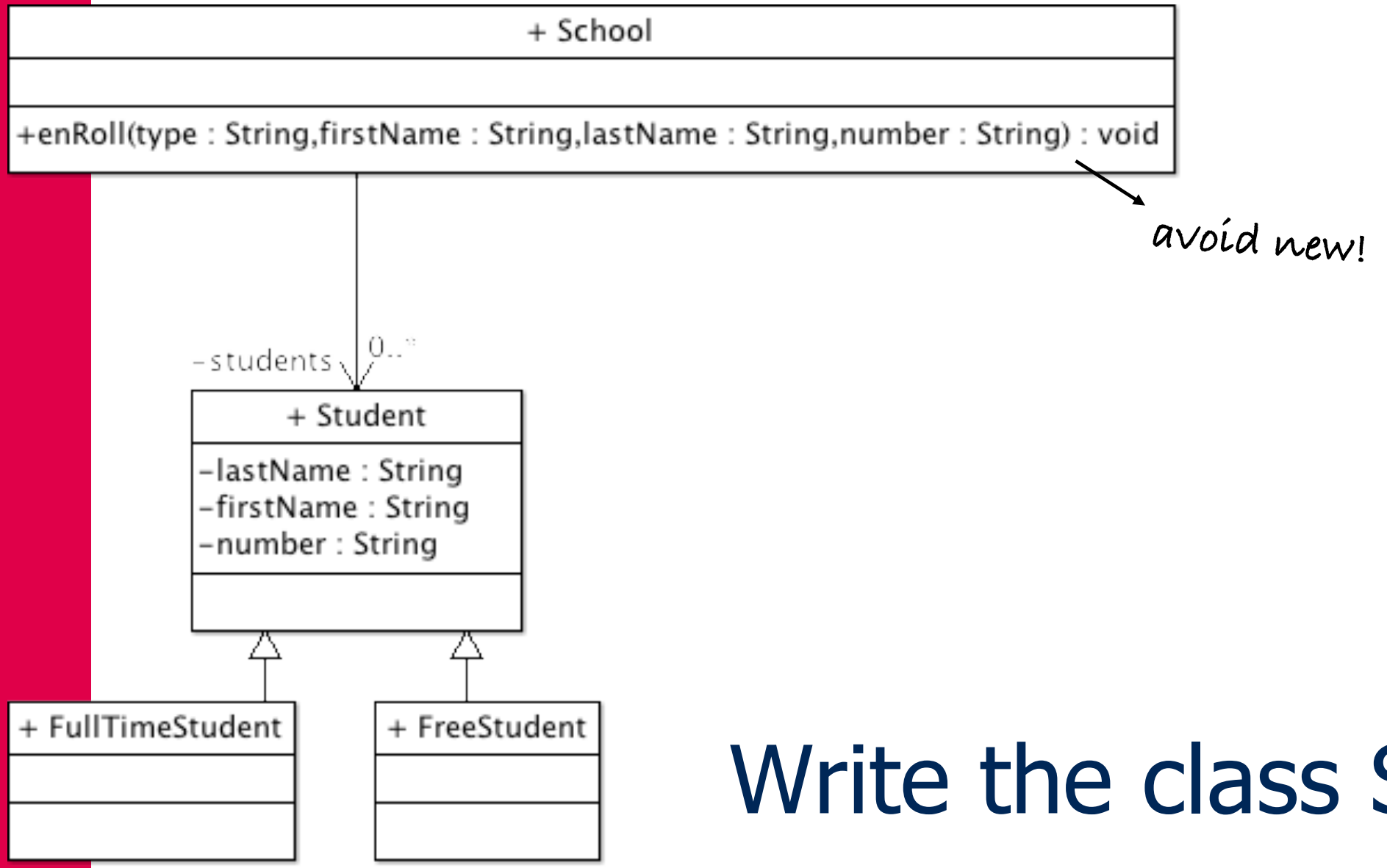
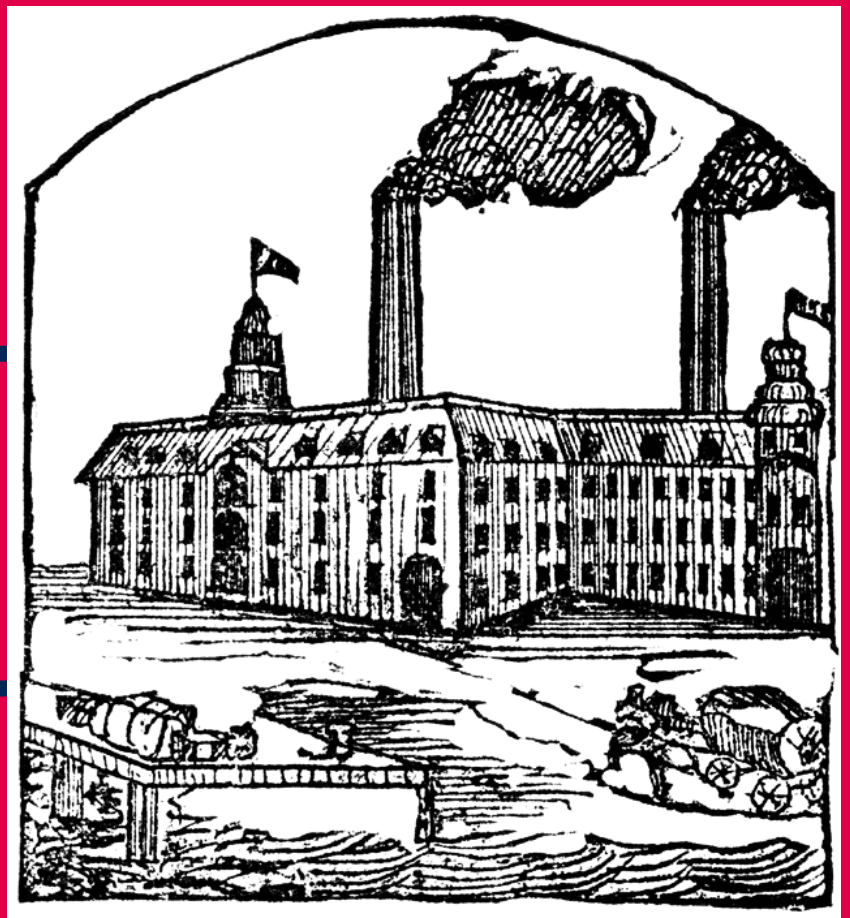


Write the class School

UC Leuven Limburg

MOVING MINDS

Simple factory



```java
package domain.school;

import java.util.HashSet;
import java.util.Set;

import domain.DomainException;

public class School {
	private Set<Student> school = new HashSet<Student>();

	public void enroll(String type, String lastName, String firstName,
					String number) throws DomainException{
		StudentFactory factory = new StudentFactory();
		Student student = factory.createStudent(type, lastName, firstName, number);
		addStudent(student);
	}

	public void addStudent(Student student) throws DomainException{
		if(student == null){
			throw new DomainException("Invalid student");
		}

		school.add(student);
	}

}
```

Simple Factory

```
package domain.school;

import domain.DomainException;

public class StudentFactory {
        public Student createStudent(String type, String lastName, String
                                        firstName, String number) {

                Student student = null;
                if(type.equals("Free")){
                        student = new HalfTimeStudent(lastName, firstName, number);
                } else if(type.equals("Full")){
                        student = new FullTimeStudent(lastName, firstName, number);
                } else {
                        throw new DomainException("Invalid type");
                }
                return student;
        }
}
```

Why?

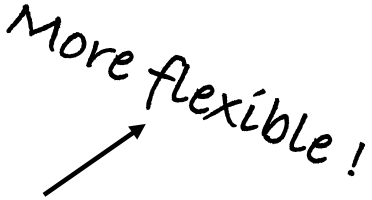
- Creating objects:
  - without exposing instantiation logic to the customer
  - through a common interface

How?

- Factory-class with method *createProduct(type: Type): Product*
  - Creates a concrete instance of product
  - … based on type-parameter
  - … and returns it

- Client
  - does not need to use new and
  - does not need to know the dynamic type

Static or not?

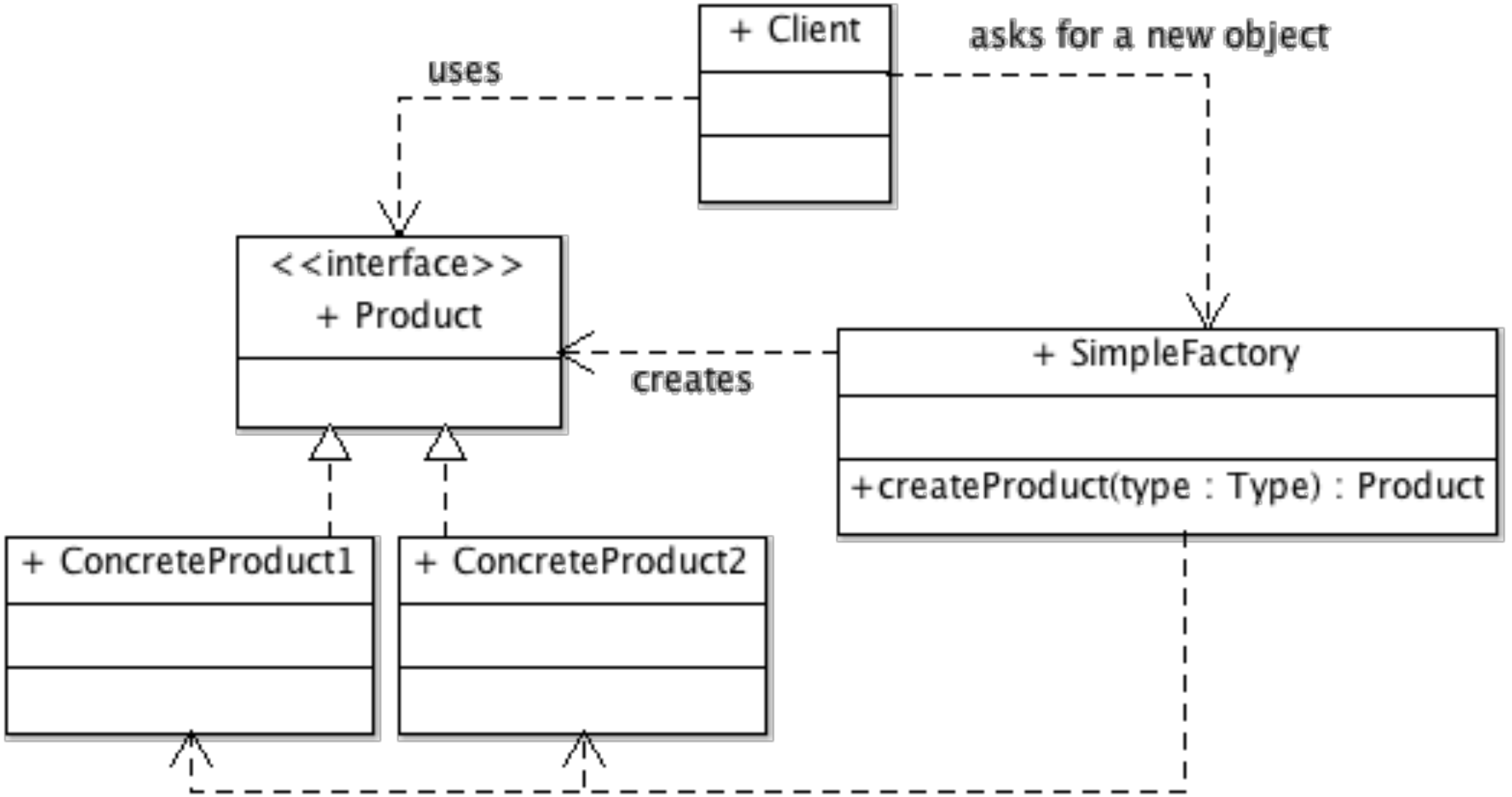


- `public Product createProduct…`
  - Inheritance still possible

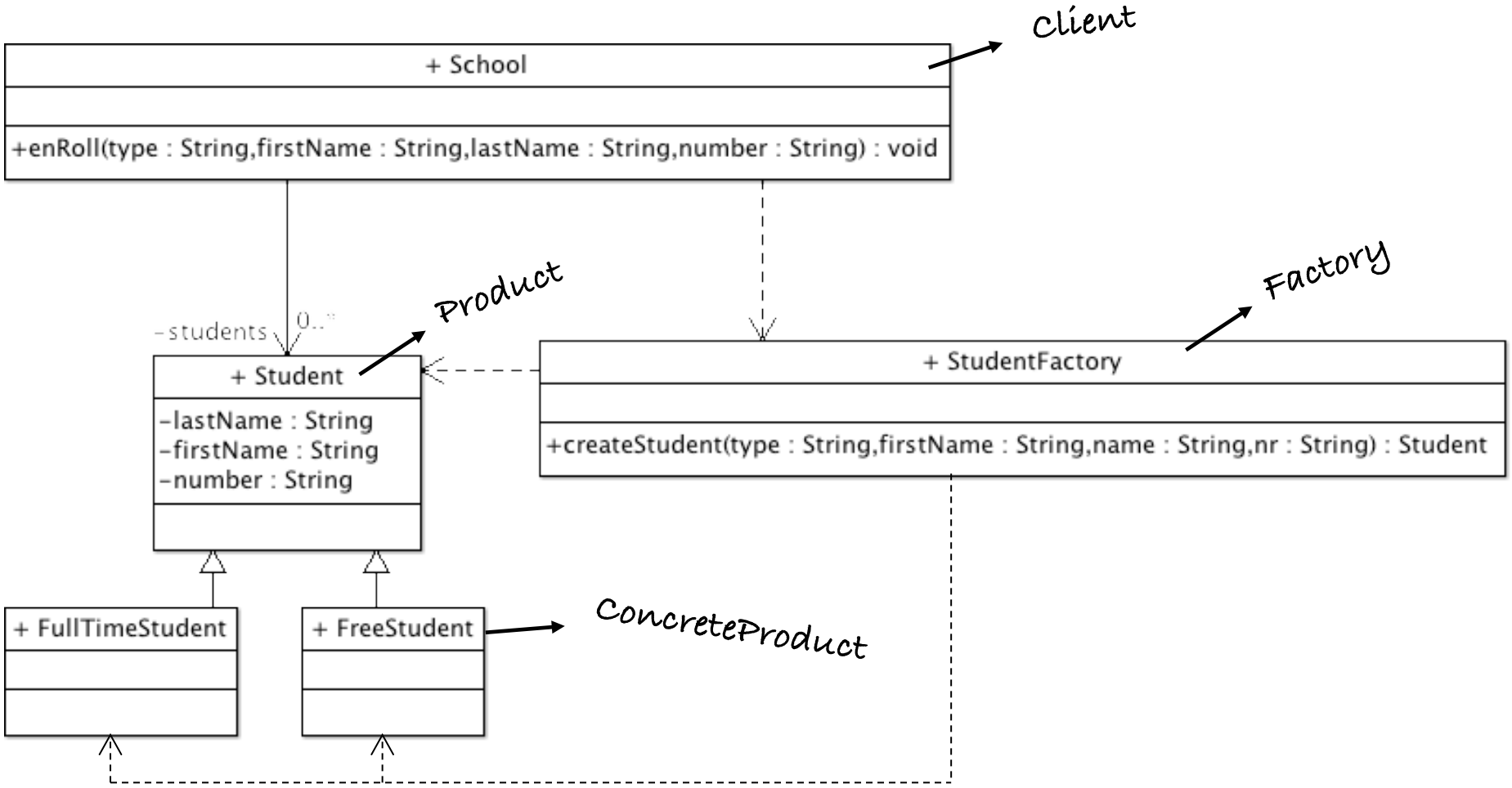
OR

- `public static Product createProduct…`
  - No instance of Factory needed
  - Inheritance **not** possible

Class diagram in general



Class diagram School



Simple Factory vs. design principes

SOLID

- SRP:
  - OK → single responsibility = create object
- OCP:
  - Client OK, Factory NOK
  - Not a major problem: no important logic and limited to one class
  - Supports OCP for the rest of the application
- ISP:
  - Not applicable
- DIP:
  - Client and ConcreteProduct both depend on Product interface

?