

Software Engineering Design & Construction

Dr. Michael Eichberg
Fachgebiet Softwaretechnik
Technische Universität Darmstadt

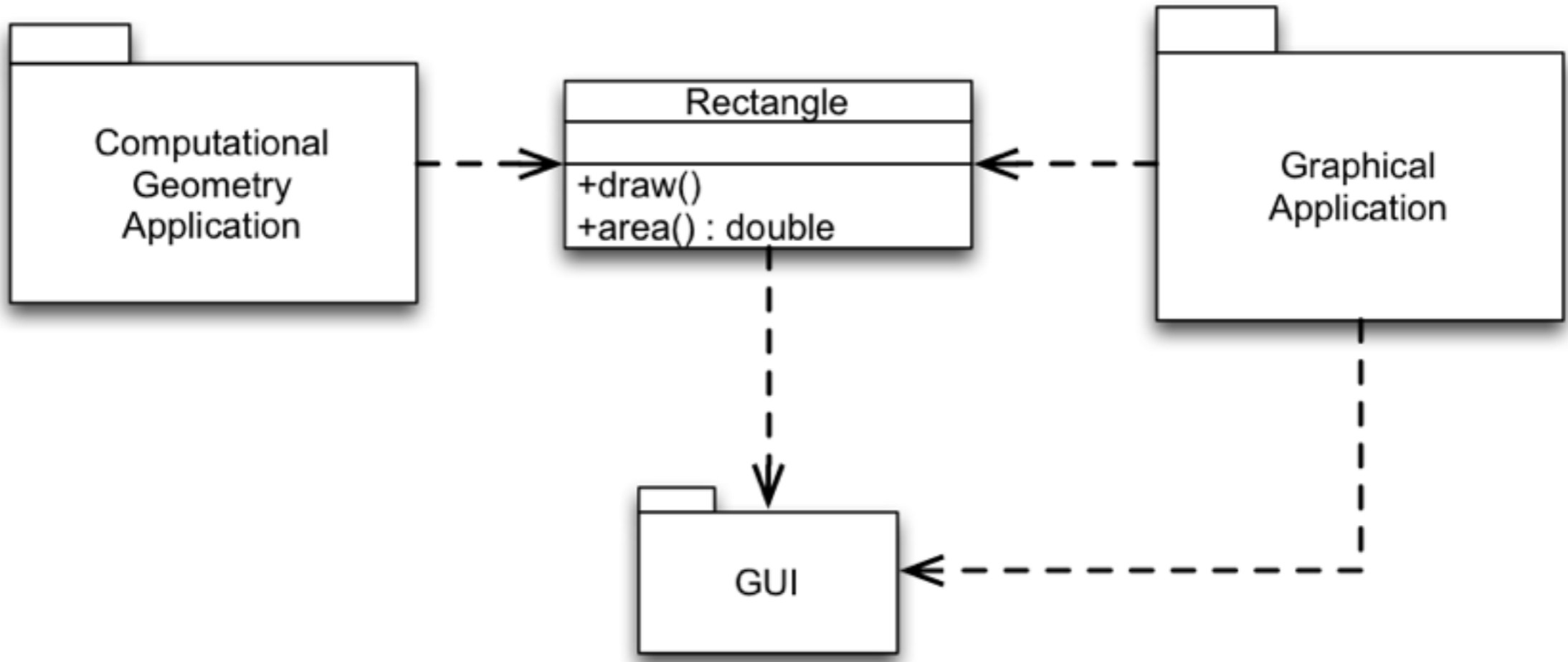
Single Responsibility Principle

*Single **R**esponsibility **P**rinciple*

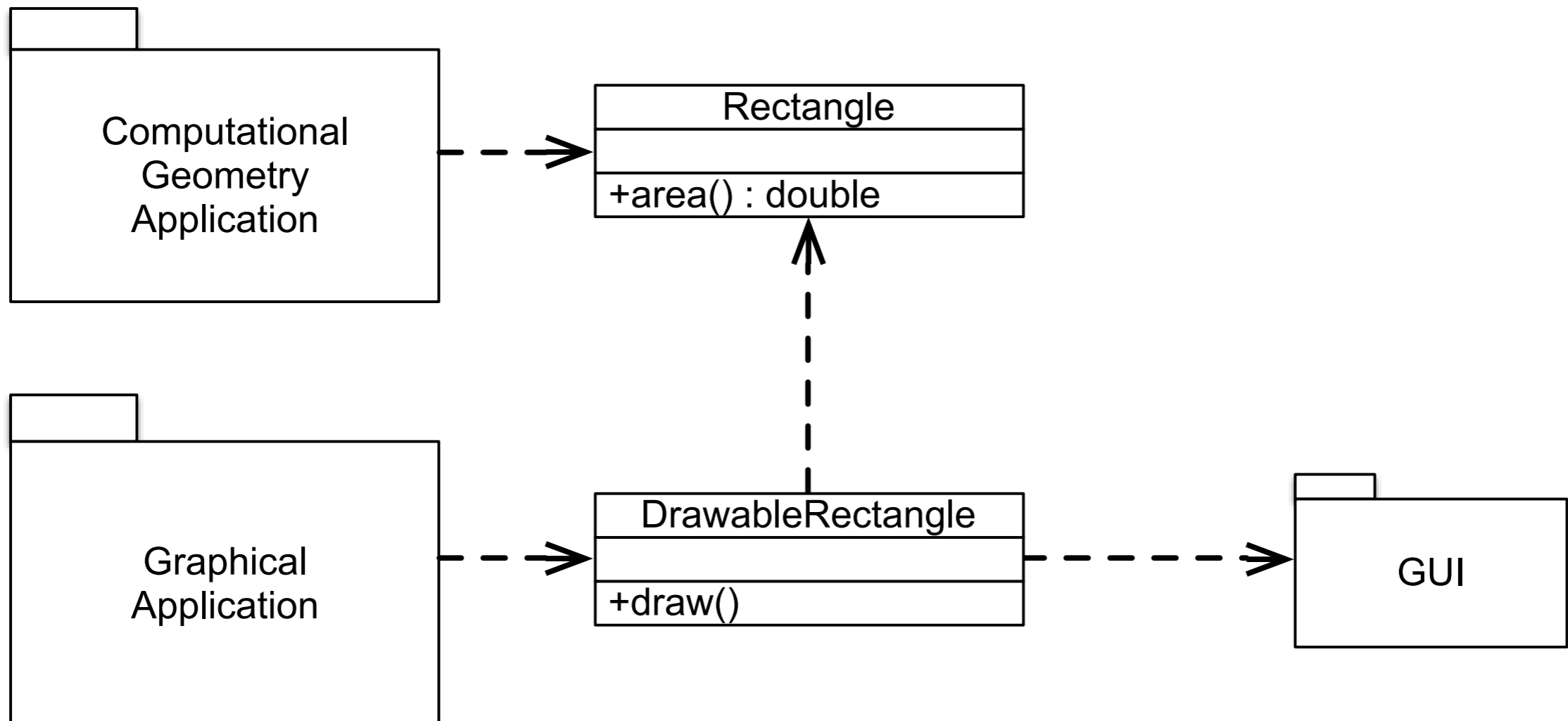
A class should have only one reason to change.

–Agile Software Development; Robert C. Martin; Prentice Hall, 2003

What do you think of the following design?



A Single-Responsibility Compliant Design



Responsibility

- In general, a class is assigned the responsibility to know or do something (one thing).
- Examples:
 - Class **PersonData** is responsible for knowing the data of a person.
 - Class **CarFactory** is responsible for creating **Car** objects.
- A responsibility is an axis of change.
- A class with only one responsibility has only one reason to change!

Cohesion

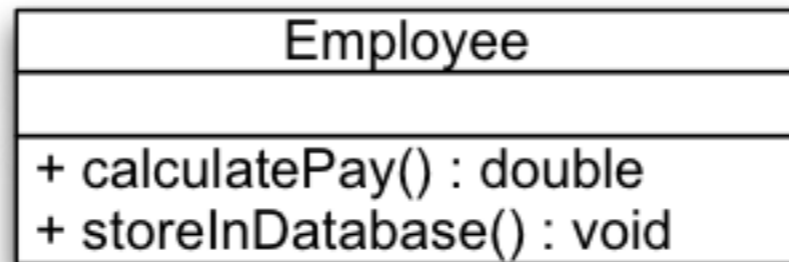
(conceptual view)

- Cohesion measures the degree of togetherness among the elements of a class.
- In a class with high cohesion every element is part of the implementation of exactly one concept. The elements of the class work together to achieve one common functionality.
- A class with high cohesion often implements only one responsibility

SRP and Cohesion

- Applying the single-responsibility principle maximizes the cohesion of classes.
- Classes with high cohesion ...
 - can be reused easily,
 - are easily understood,
 - protect clients from changes, that should not affect them.

Should we split the responsibilities of this class?



When to apply the Single-Responsibility Principle?

- We should split a class that has two responsibilities if:
 - Both responsibilities will change separately.
 - The responsibilities are used separately by other classes.
 - Responsibilities pertain to optional features of the system.
- We should not split responsibilities if:
 - Both responsibilities will only change together, e.g. if they together implement one common protocol.
 - Both responsibilities are only used together by other classes.
 - Responsibilities pertain to mandatory features.

This principle also applies at higher-abstraction levels! E.g. at the component-level.

java.util.Date

Do we have a SRP Violation?

- The class **Date** represents a specific instant in time, with millisecond precision.

int	getSeconds()
long	getTime()
int	getTimezoneOffset()
int	getYear()
int	hashCode()
static long	parse(String s)
void	setDate(int date)
void	setHours(int hours)
void	setMinutes(int minutes)
...	...

java.util.Date

Do we have a SRP Violation?

- The class **Date** represents a specific instant in time, with millisecond precision.

-

...

```
public static long parse(String s)
```

Attempts to interpret the string s as a representation of a date and time. If the attempt is successful, the time indicated is returned represented as the distance, measured in milliseconds, of that time from the epoch (00:00:00 GMT on January 1, 1970). If the attempt fails, an IllegalArgumentException is thrown.

...

Do perform the strategic application of principles!

Only apply a principle,
if there is a symptom!