الجامعة السورية الخاصة
SYRIAN PRIVATE UNIVERSITY

مقرر تصميم نظم البرمجيات

كلية الهندسة المعلوماتية

المحاضرة الأولى

Introduction to Software Engineering and Design

د. رياض سنبل

Course Details

- **Course Code:** CIEC.6.03
- **Course Title:** Software System Design
- **Credits:** 3 ECTS (2 Lectures & 2 Practical Sessions per week)
- **Grading:**
  - 15% Test 1
  - 15% Test 2
  - 20% Practical Sessions, Assignments, etc.
  - 50% Final Exam

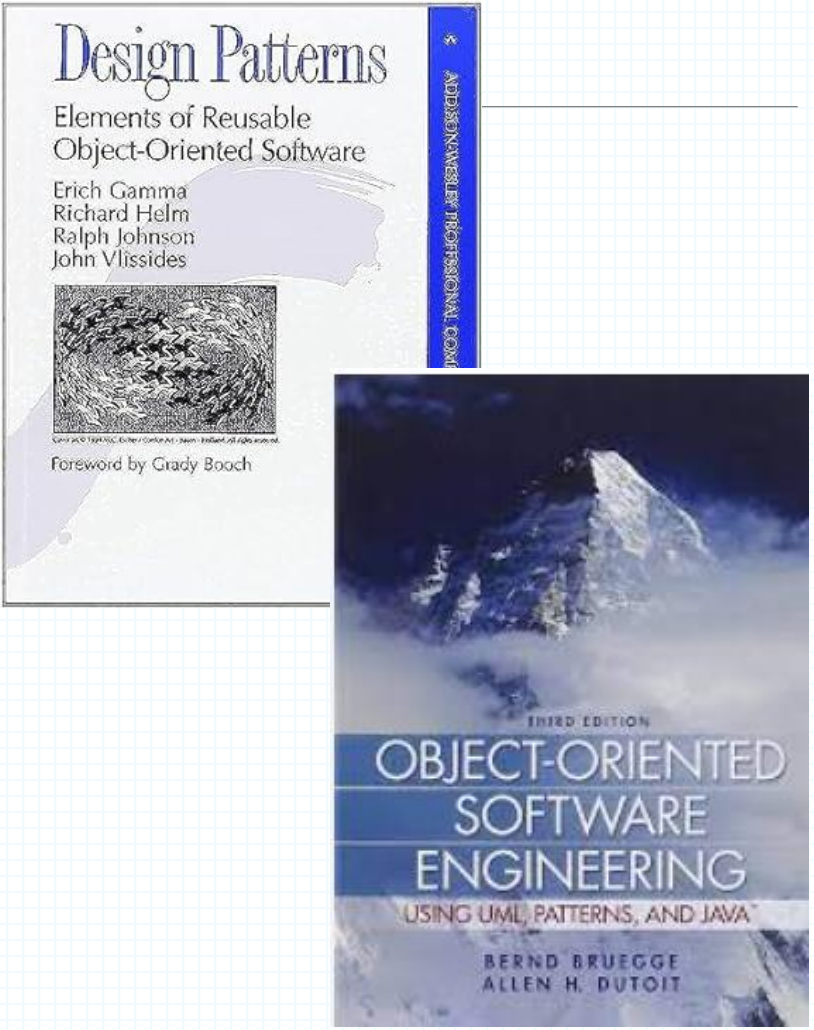
Outline of the course

- The course covers the following main topics:
  - 1. Introduction to basic concepts in software engineering.
  - 2. Key activities in software design.
  - 3. Software design principles.
  - 4. Key design patterns (creational, structural, behavioral).
  - 5. Designing software entities and using the OCL language to describe constraints and conditions.
  - 6. Overview of code generation based on the design.

About 70% of the course

Outline of the course

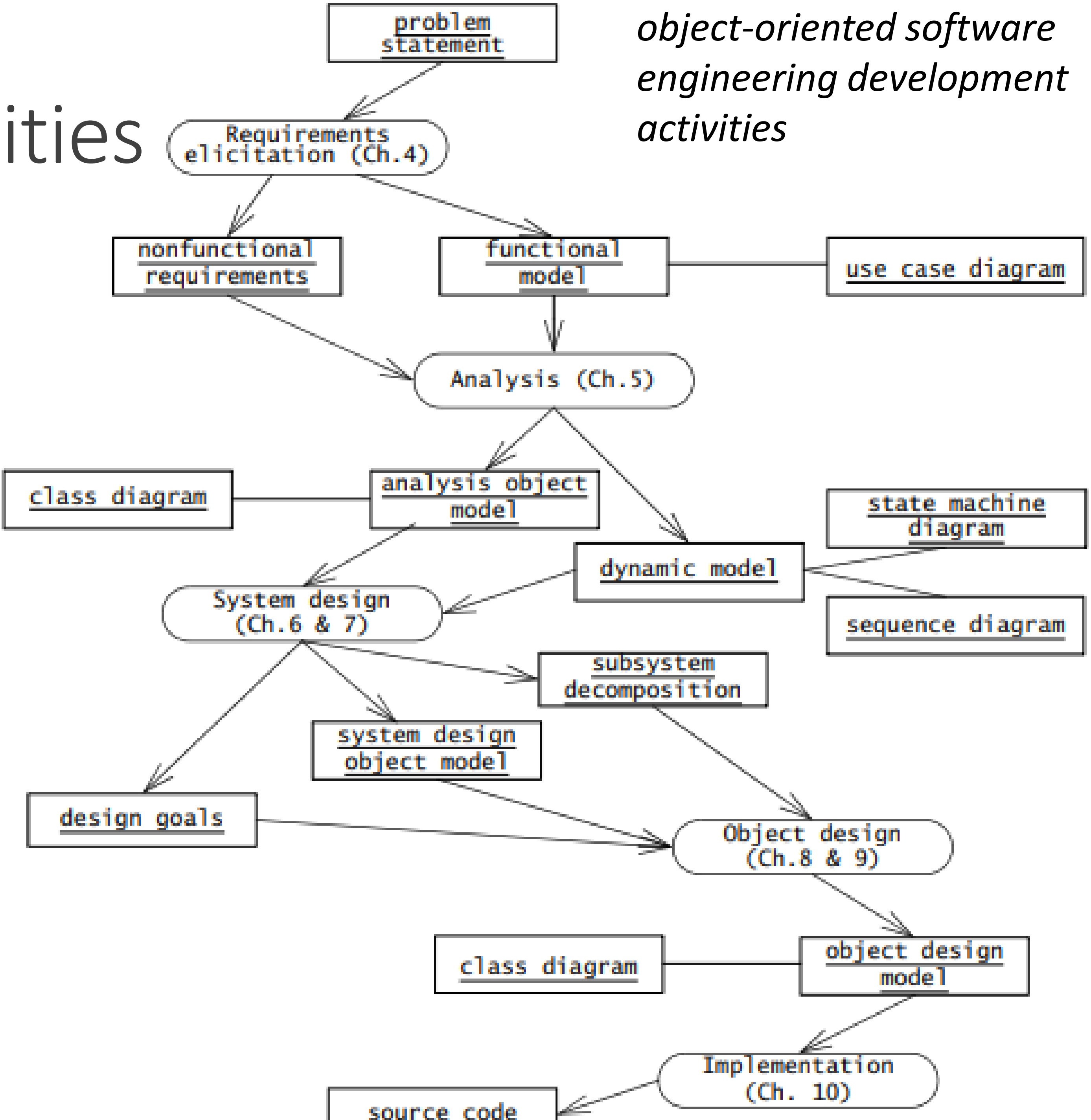
- **Textbooks:**
  - *Gamma, Erich, Richard Helm, Ralph Johnson, and John Vlissides. Design patterns: elements of reusable object-oriented software. Pearson Deutschland GmbH, 1995.*
  - *Bruegge, Bernd, and Allen H. Dutoit. "Object–oriented software engineering. using uml, patterns, and java." Learning 5, no. 6 (2009): 7.*


Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMP
THIRD EDITION
OBJECT-ORIENTED SOFTWARE ENGINEERING
USING UML, PATTERNS, AND JAVA
BERND BRUEGGE
ALLEN H. DUTOIT

The Context:
SE Development Activities

1) Requirements Elicitation.
2) Analysis.
3) System Design.
4) Object Design.
5) Implementation.
6) Testing.

- Object-oriented software engineering is **iterative**; that is, activities can occur in parallel and more than once



object-oriented software engineering development activities

problem statement

Requirements elicitation (Ch.4)

nonfunctional requirements

functional model

use case diagram

Analysis (Ch.5)

class diagram

analysis object model

state machine diagram

dynamic model

sequence diagram

System design (Ch.6 & 7)

subsystem decomposition

system design object model

design goals

Object design (Ch.8 & 9)

class diagram

object design model

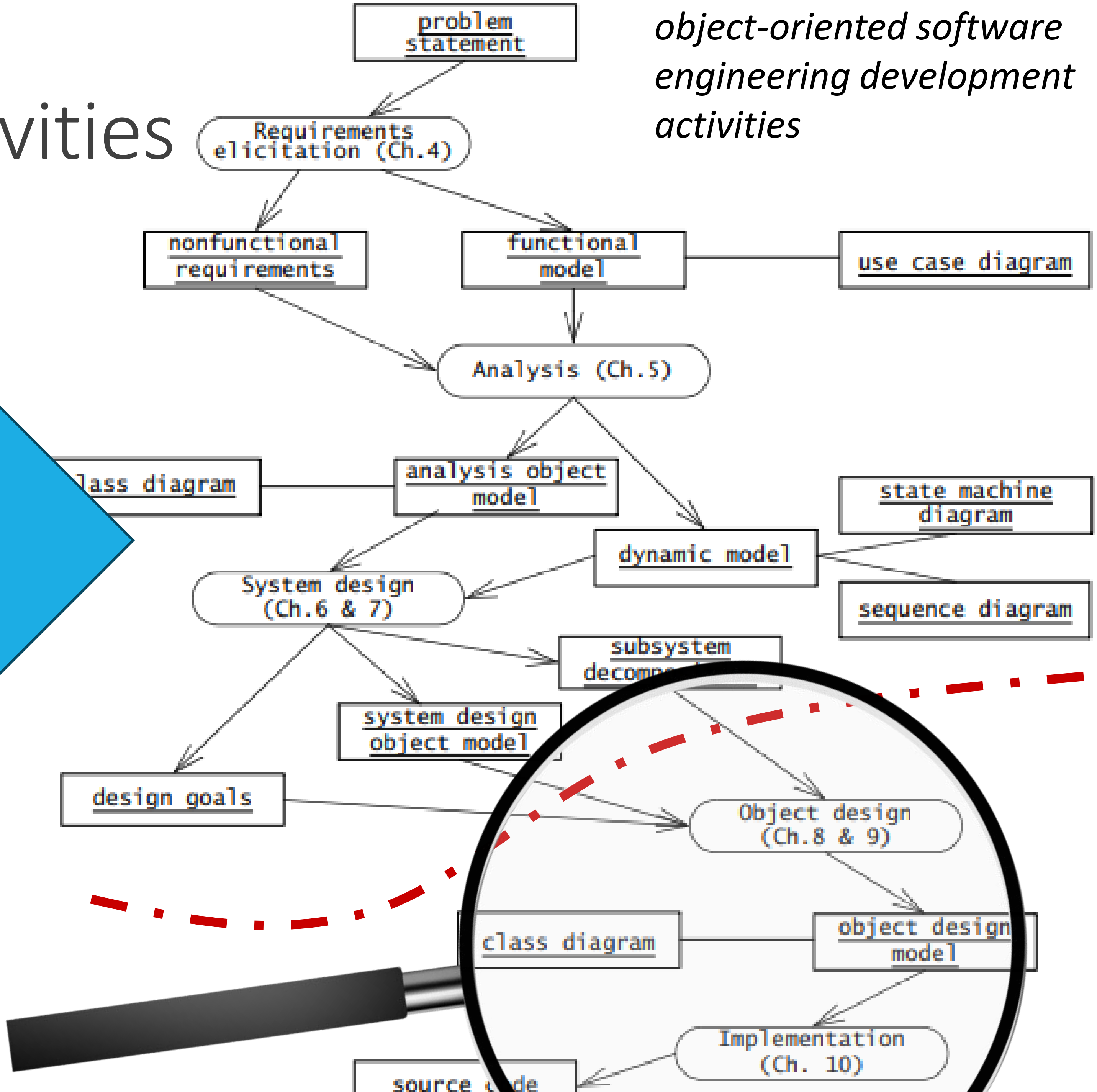
Implementation (Ch. 10)

source code

The Context: SE Development Activities

object-oriented software engineering development activities

In the last courses



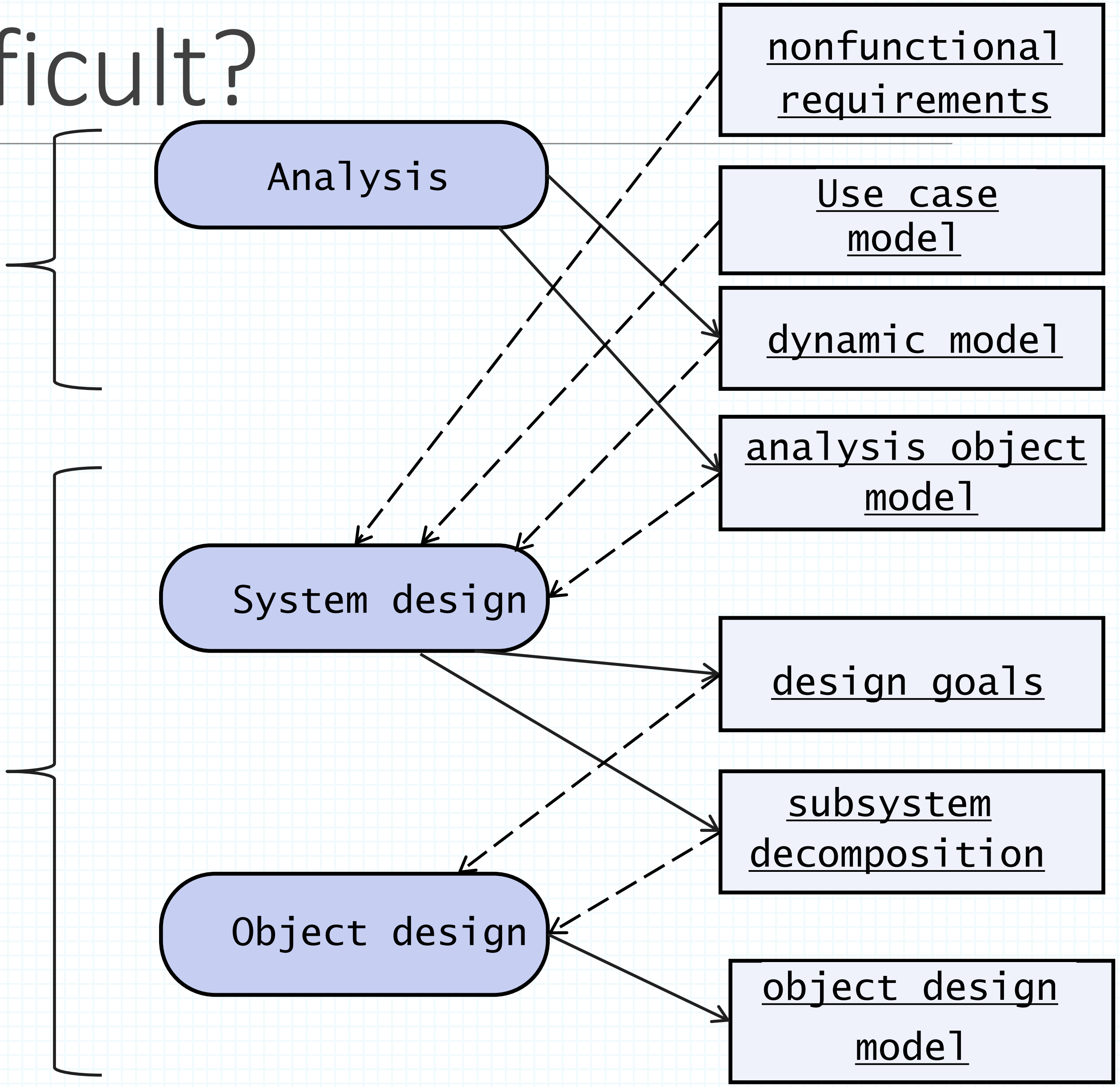
Why is Design so Difficult?

Focuses on the application domain

Focuses on the solution domain

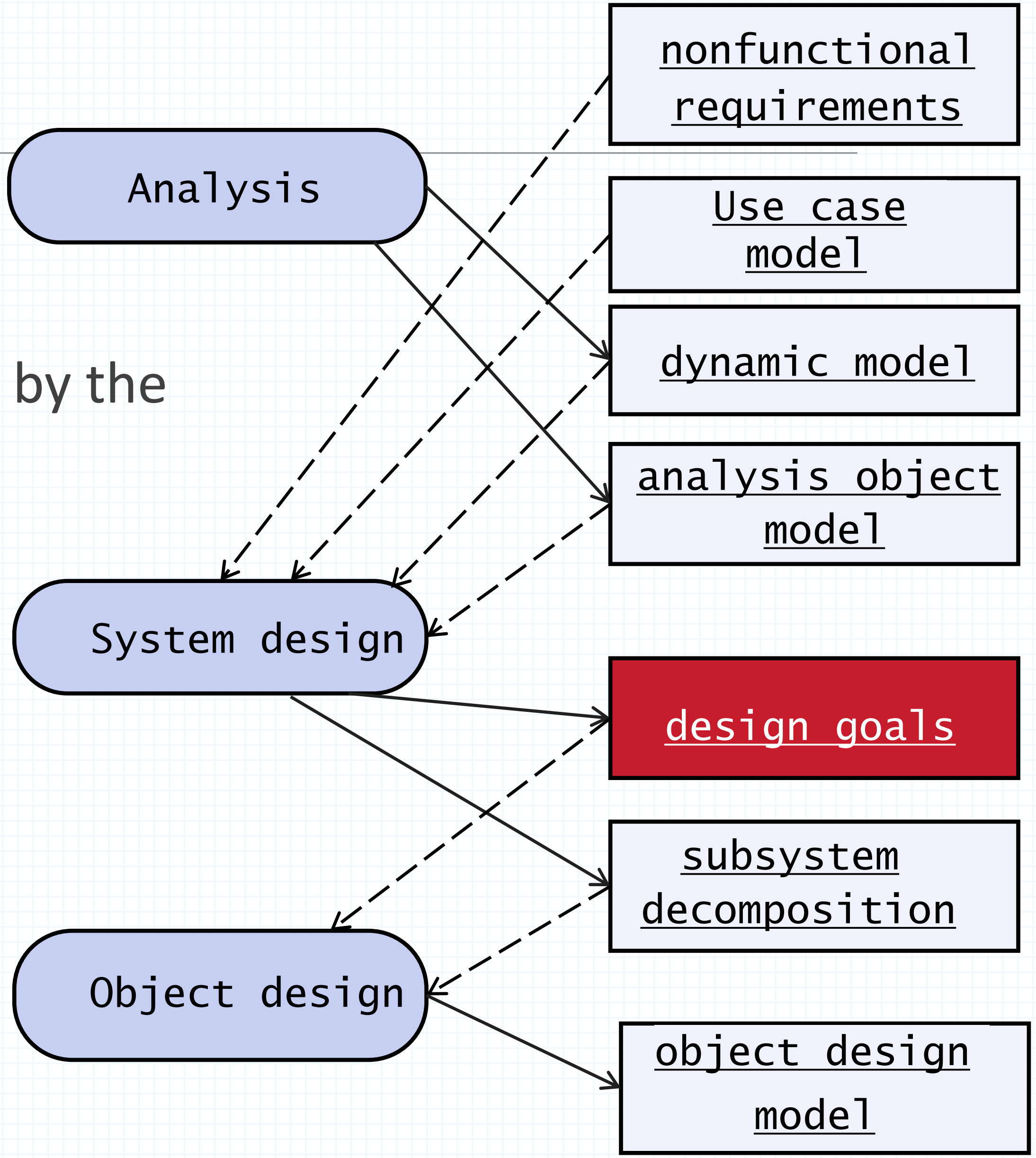
The solution domain is changing very rapidly

- *Halftime knowledge in SE ≈ 3-5 years*
- *Cost of hardware rapidly sinking*
- *Design knowledge is a moving target*



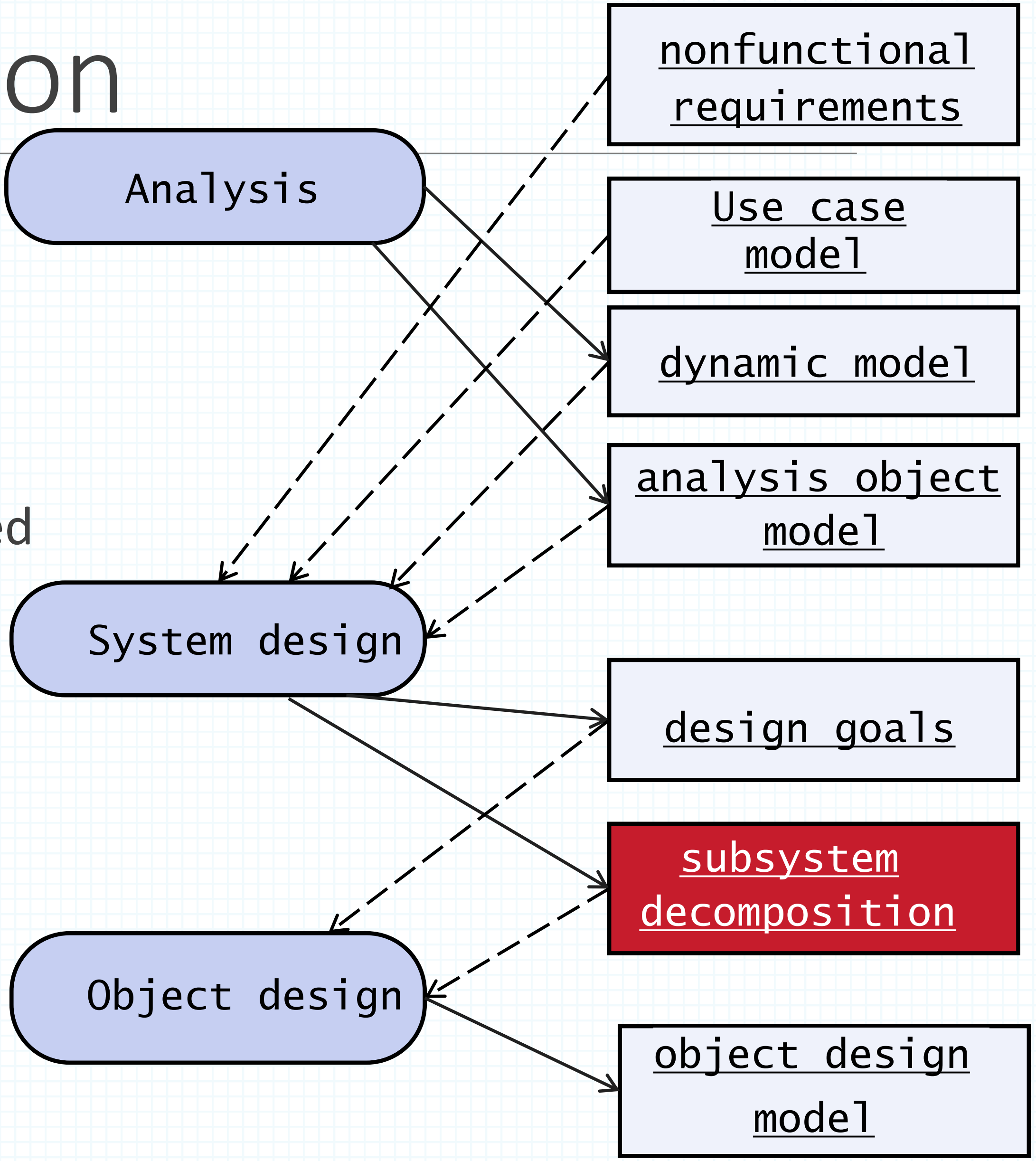
Design Goals!

- The design goals are derived from the nonfunctional requirements.
- Design goals guide the decisions to be made by the developers when trade-offs are needed.
- Example of Design Goals:
  - Increased productivity
  - High-performance
  - Maintainability
  - Reusability
  - Portability
  - Fault tolerance
  - Cost-effectiveness
  - etc

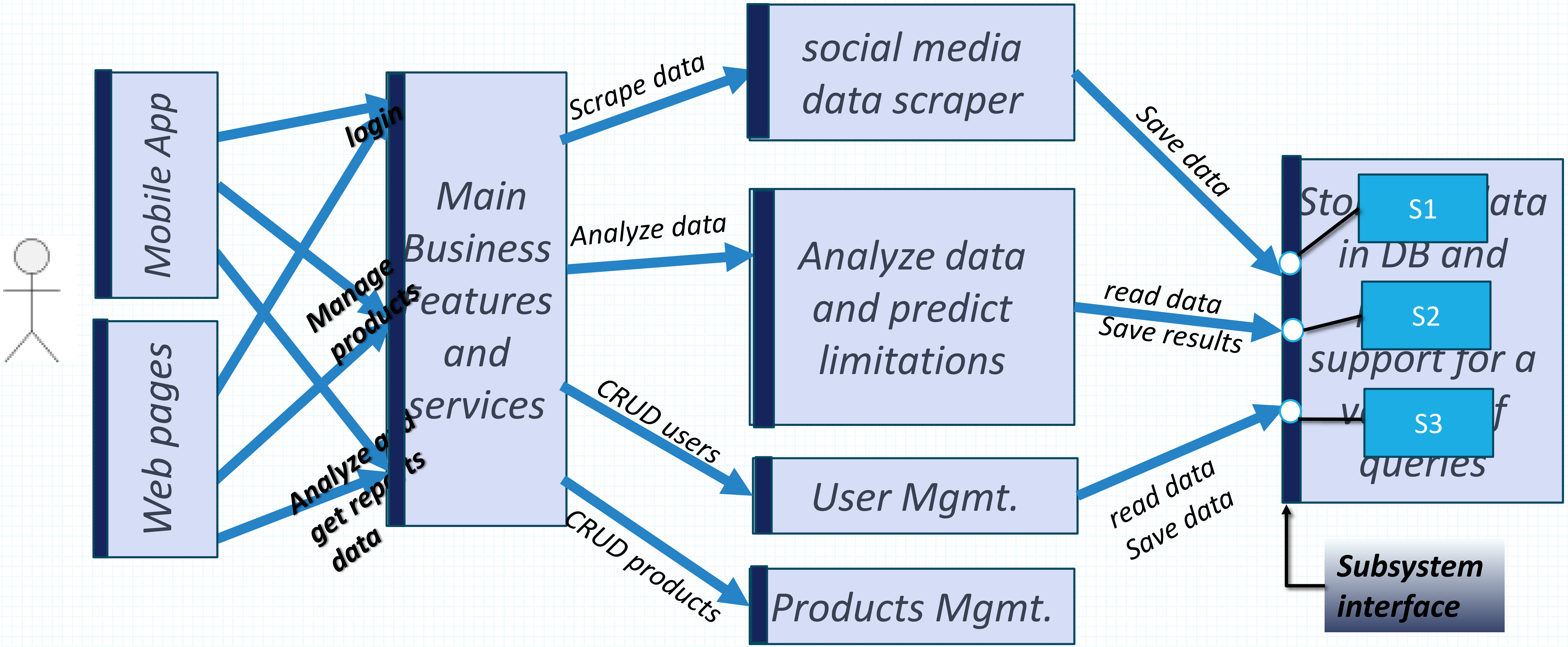


Subsystem Decomposition

- The subsystem decomposition constitutes the main part of system design.
- We divide the system into manageable pieces to deal with complexity.
- Each subsystem is assigned to a team and realized independently.
- There are many generic system decompositions best practices called “architectural styles”.
  - Client/Server
  - Repository
  - Model/View/Controller
  - Three-tier, Four-tier Architecture
  - Service-Oriented Architecture (SOA)
  - etc



Subsystem Decomposition



Mobile App

Web pages

login

Manage products

Analyze and get reports data

Main Business features and services

Scrape data

Analyze data

CRUD users

CRUD products

social media data scraper

Analyze data and predict limitations

User Mgmt.

Products Mgmt.

Save data

read data

Save results

read data

Save data

Sto ata in DB and support for a v f queries

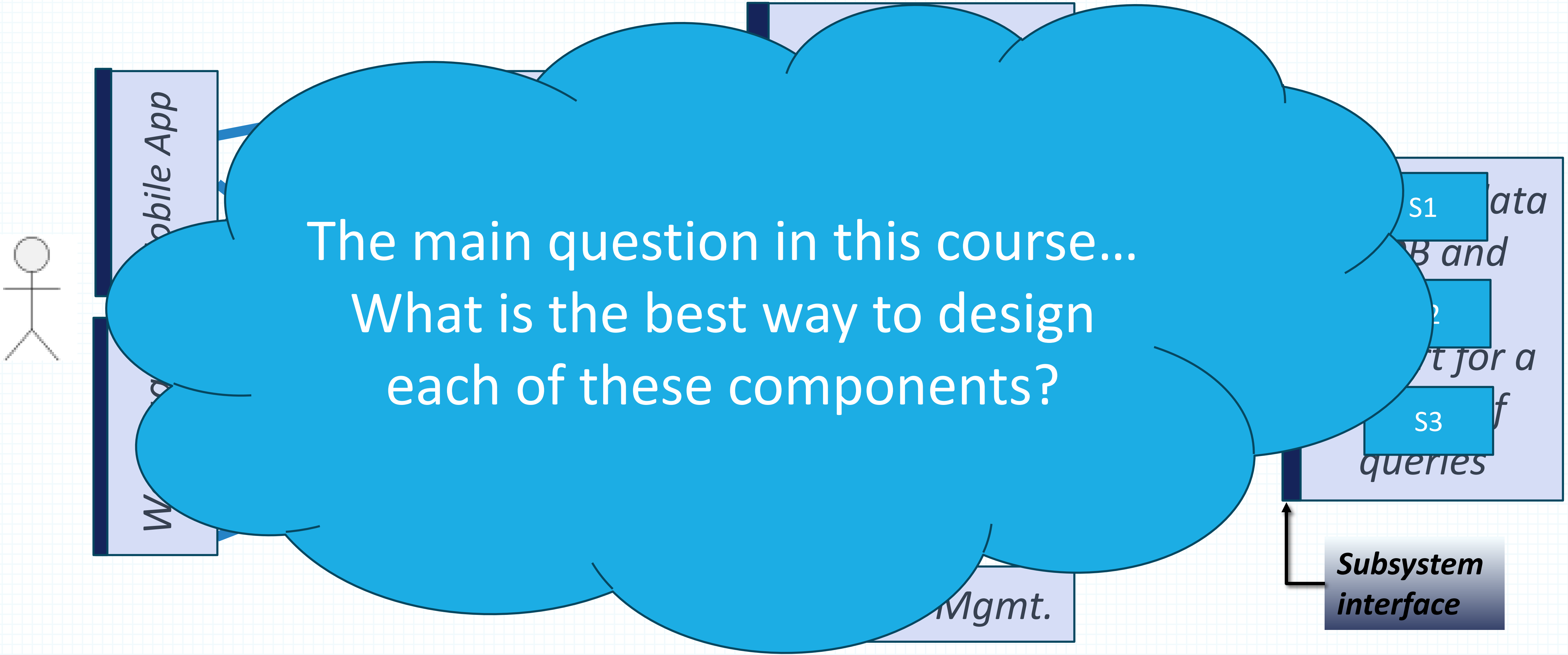
S1

S2

S3

Subsystem interface

Subsystem Decomposition



The main question in this course…
What is the best way to design each of these components?

Introductory Example

Suggest a design for this component

“This component is a software toolkit crafted to manage item sets regardless of their row affiliations, facilitating tasks such as sorting with various algorithms and computing statistical measures like mean and median”

See more detailed description

مثال تمهيدي

- نهدف في هذه المسألة إلى بناء حزمة برمجيّة تتيح التعامل مع مجموعات من الأغراض (بغض النظر عن الصفوف المنتمية لها) سواء من جهة ترتيبها وفق خوارزميات الفرز المختلفة، أو من جهة حساب المعلومات الاحصائية كالوسطي والوسط والقيمة العظمى.
- يجب تحقيق المطلوب من أجل أية مجموعة من العناصر شريطة توفّر **آليّة قياس لقيمة كل من هذه العناصر** دون الحاجة إلى معرفة الصفوف التي تنتمي إليها، حيث **تأخذ خوارزميّة الترتيب مثلاً مصفوفة من الأغراض "القابلة للقياس"**، وتعمل على ترتيبها من خلال **مقارنة "قياسات" أغراضها**.
- تتضمن الحزمة مجموعة متنوعة من خوارزميات الترتيب التي تتشارك البنية العامة ذاتها، وتتيح إمكان استدعاء توابعها الأساسيّة بالأسلوب ذاته.

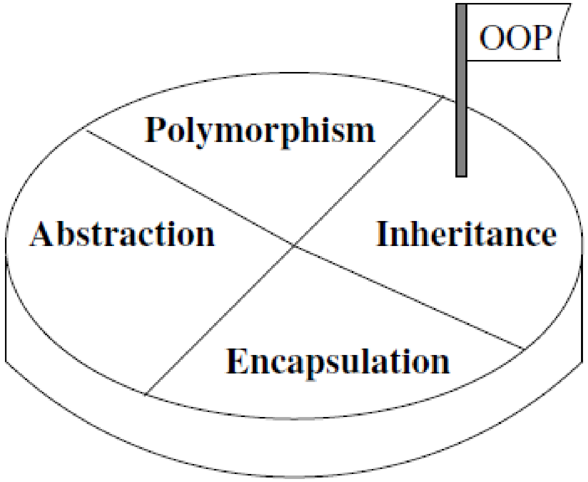
Group Discussion

OOP - Quick Overview

WHY WE NEED OOP IN THIS COURSE?

Object Oriented Programming

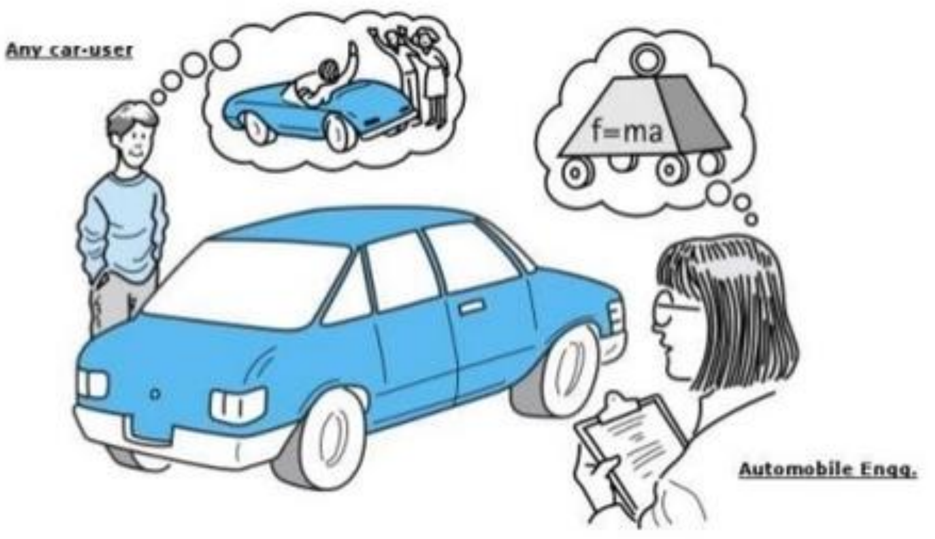
- A **programming paradigm** based on **objects** having **data and methods** defined in the **class** to which it belongs.
- Four of the major characteristics of the Object Oriented Paradigm are:
  - **Abstraction:** Simplify Reality.
  - **Encapsulation:** Hiding Data and Complexity.
  - **Inheritance:** A class can derive its methods and properties for another class
  - **Polymorphism:** A class can implement an inherited method in its own way.



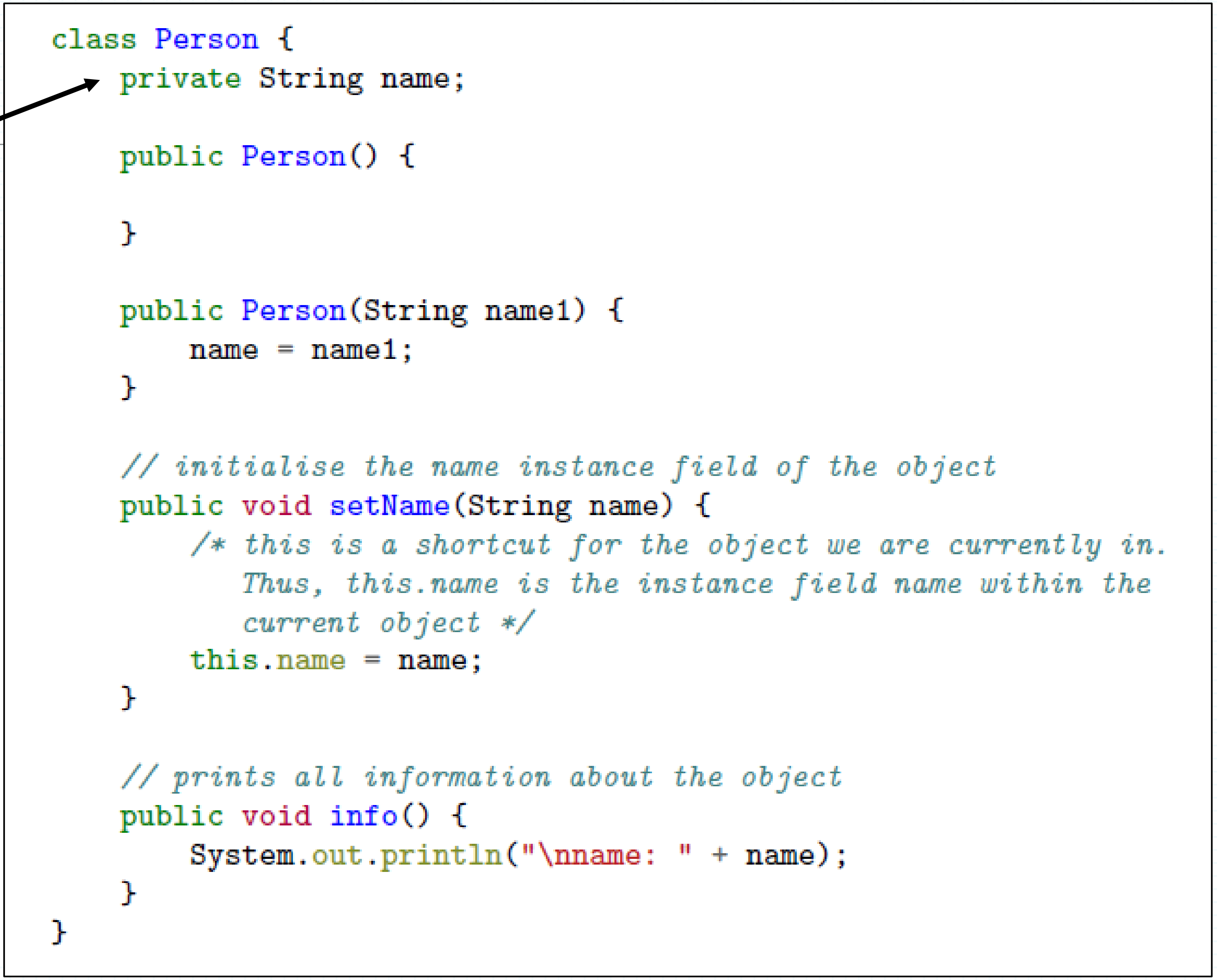
- The examples of the object-oriented paradigm are Java, C++, Python, C#, etc.

Example A (1/3)

Abstraction?
(Simplify Reality**)**

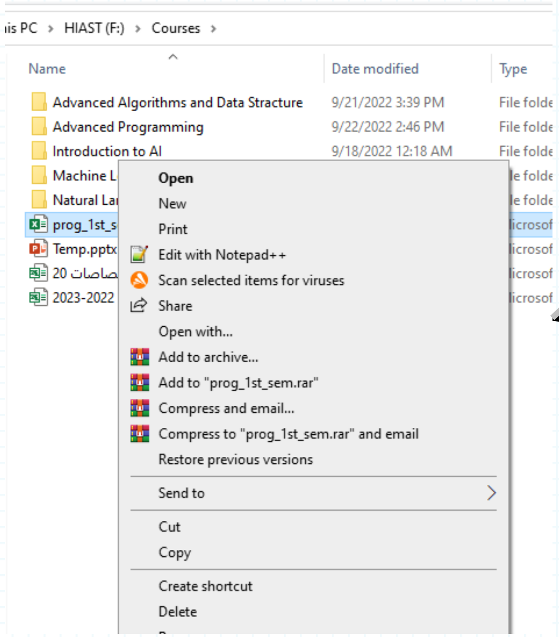


An abstraction includes the essential details relative to the perspective of the viewer



```
class Person {
    private String name;

    public Person() {

    }

    public Person(String name1) {
        name = name1;
    }

    // initialise the name instance field of the object
    public void setName(String name) {
        /* this is a shortcut for the object we are currently in.
           Thus, this.name is the instance field name within the
           current object */
        this.name = name;
    }

    // prints all information about the object
    public void info() {
        System.out.println("\nname: " + name);
    }
}
```

Example (1/3)

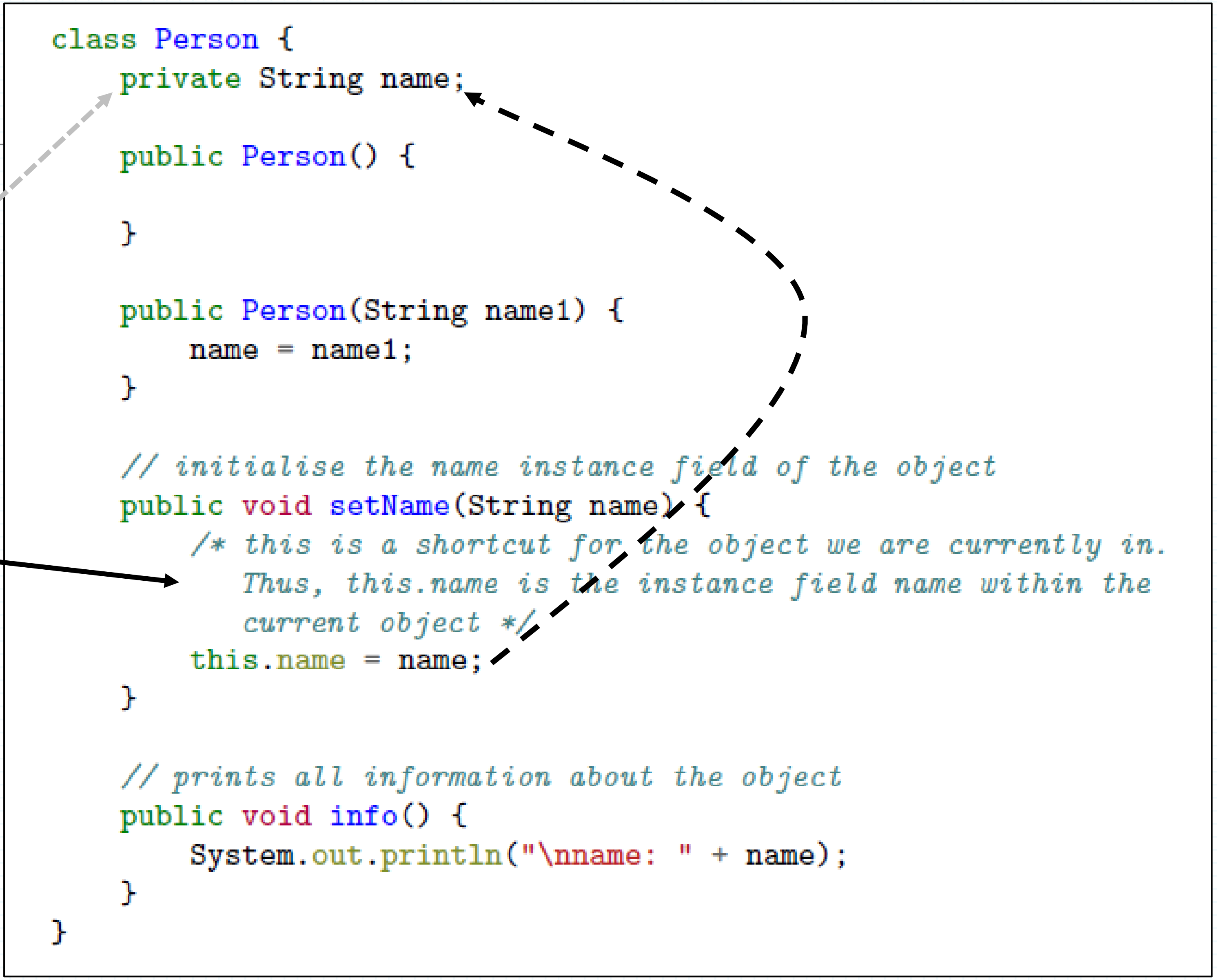


is PC › HIAST (F:) › Courses ›

Name	Date modified	Type
Advanced Algorithms and Data Structure	9/21/2022 3:39 PM	File folde
Advanced Programming	9/22/2022 2:46 PM	File folde
Introduction to AI	9/18/2022 12:18 AM	File folde
Machine L		le folde
Natural La		le folde
prog_1st_s		licrosof
Temp.pptx		licrosof
20 صاصات		licrosof
2023-2022		licrosof

Open
New
Print
Edit with Notepad++
Scan selected items for viruses
Share
Open with...
Add to archive...
Add to "prog_1st_sem.rar"
Compress and email...
Compress to "prog_1st_sem.rar" and email
Restore previous versions
Send to
Cut
Copy
Create shortcut
Delete

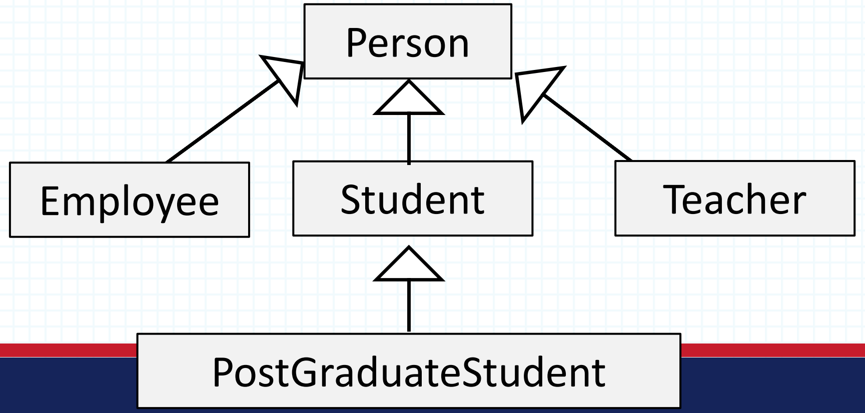
Encapsulation?
(Hiding Data and Complexity**)**

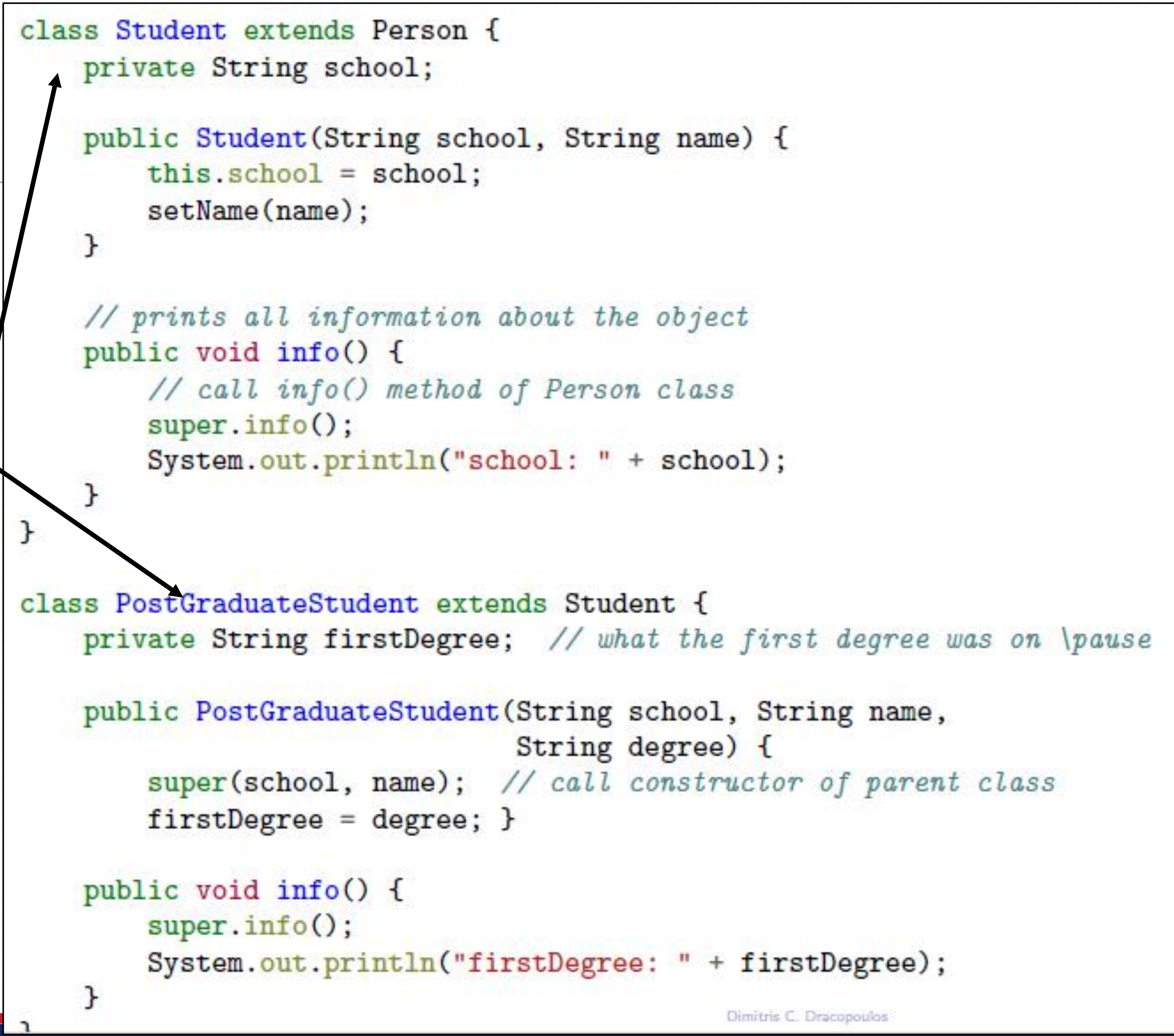


```
class Person {
    private String name;

    public Person() {

    }

    public Person(String name1) {
        name = name1;
    }

    // initialise the name instance field of the object
    public void setName(String name) {
        /* this is a shortcut for the object we are currently in.
           Thus, this.name is the instance field name within the
           current object */
        this.name = name;
    }

    // prints all information about the object
    public void info() {
        System.out.println("\nname: " + name);
    }
}
```

Example (2/3)

- **Inheritance** organizes classes in hierarchies.
- Hierarchies of class can be created for the following reasons:
    - Reusability of code (Reusability of the functionality of existing classes).
    - Code is easier to maintain.
    - Polymorphic behaviour (objects are manipulated via reference variables of the base class)



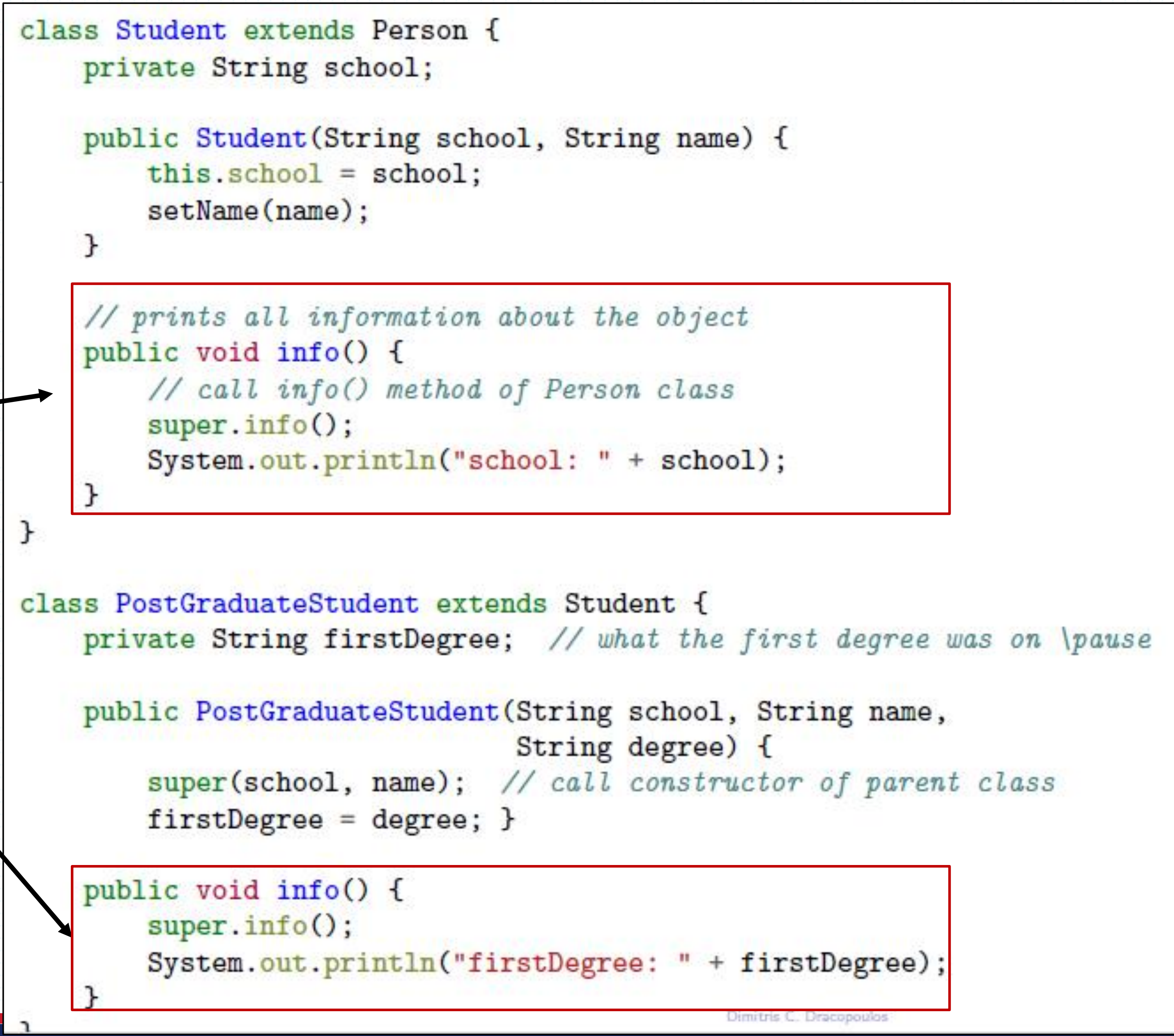


```
class Student extends Person {
    private String school;

    public Student(String school, String name) {
        this.school = school;
        setName(name);
    }

    // prints all information about the object
    public void info() {
        // call info() method of Person class
        super.info();
        System.out.println("school: " + school);
    }
}

class PostGraduateStudent extends Student {
    private String firstDegree;  // what the first degree was on \pause

    public PostGraduateStudent(String school, String name,
                               String degree) {
        super(school, name);  // call constructor of parent class
        firstDegree = degree; }

    public void info() {
        super.info();
        System.out.println("firstDegree: " + firstDegree);
    }
```

Dimitris C. Dracopoulos

Example A (2/3)

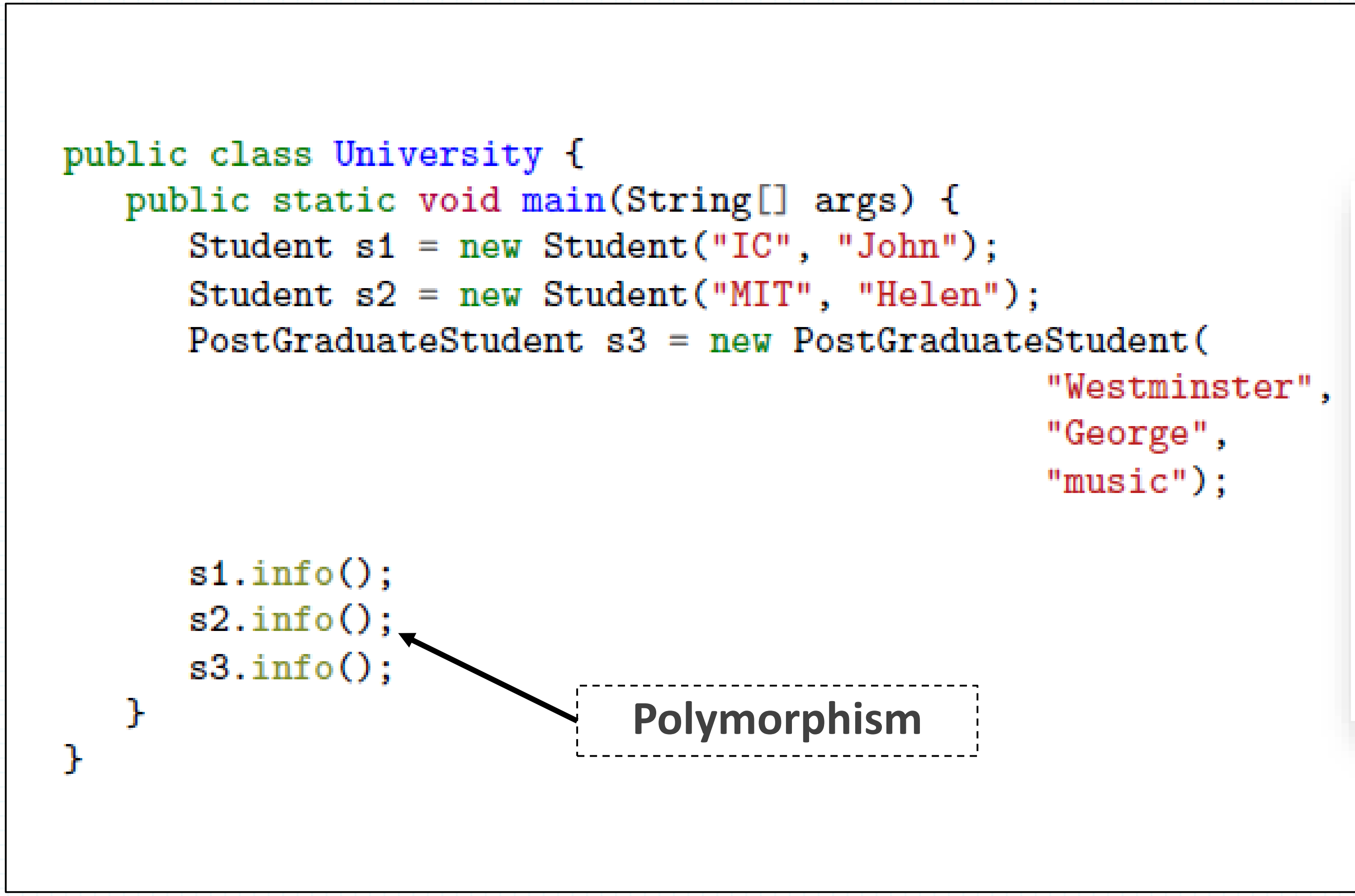
Polymorphism?



```java
class Student extends Person {
    private String school;

    public Student(String school, String name) {
        this.school = school;
        setName(name);
    }

    // prints all information about the object
    public void info() {
        // call info() method of Person class
        super.info();
        System.out.println("school: " + school);
    }
}

class PostGraduateStudent extends Student {
    private String firstDegree;  // what the first degree was on \pause

    public PostGraduateStudent(String school, String name,
                               String degree) {
        super(school, name);  // call constructor of parent class
        firstDegree = degree; }

    public void info() {
        super.info();
        System.out.println("firstDegree: " + firstDegree);
    }
```

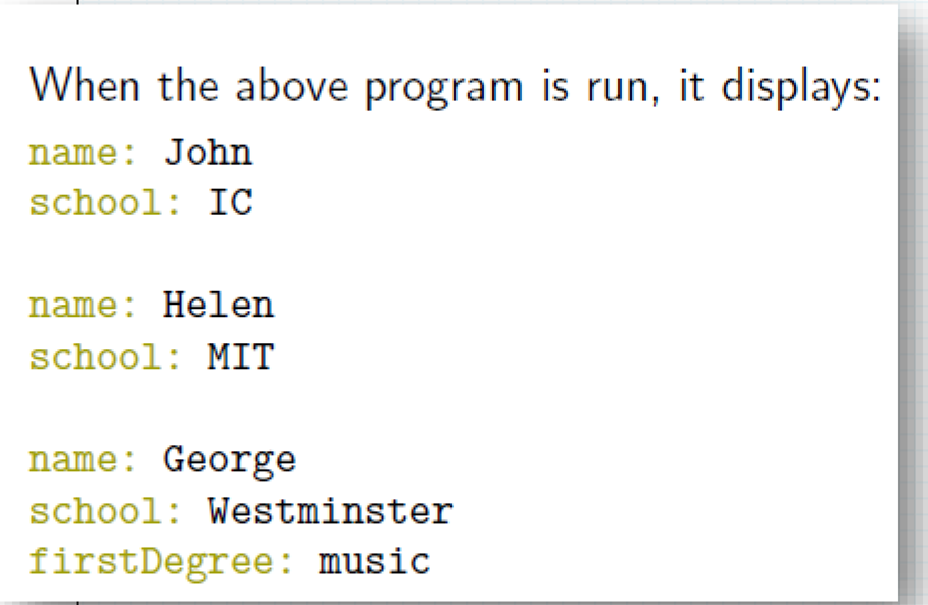
Dimitris C. Dracopoulos

Example A (3/3)



```java
public class University {
   public static void main(String[] args) {
      Student s1 = new Student("IC", "John");
      Student s2 = new Student("MIT", "Helen");
      PostGraduateStudent s3 = new PostGraduateStudent(
                                          "Westminster",
                                          "George",
                                          "music");


      s1.info();
      s2.info();
      s3.info();
   }
}
```

Polymorphism



When the above program is run, it displays:

```
name: John
school: IC

name: Helen
school: MIT

name: George
school: Westminster
firstDegree: music
```

Interfaces

```
public interface Doable
{
   public void doThis();
   public int doThat();
   public void doThis2 (float value, char ch);
   public boolean doTheOther (int num);
}
```

```
public class CanDo implements Doable
{
   public void doThis ()
   {
      // whatever
   }

   public void doThat ()
   {
      // whatever
   }
   // etc.
}
```

implements is a reserved word

Each method listed in **Doable** is given a definition

Generics - Syntax

A class can have multiple parameters, e.g:

```
public class Stuff<A,B,C> { … }
```

Subclassing parameterized classes allowed, e.g:

```
/* Extending a particular type */
class IntBox extends Box<Integer> { … }
```

Or

```
/* Extending a parameterized type */
class SpecialBox<E> extends Box<E> { … }
```