Introduction

Objectives

- A Language for Complex Applications
- Object Terminology
  - Abstraction
  - Encapsulation
  - Hierarchy
  - Polymorphism

A Language for Complex Applications

- Many software **applications** are **complex**.
  - The underlying **problem domain** is often quite **intricate** and **detailed**.
- For an application to be **practical** and **usable**, it must represent some of the complexity of the **problem domain**.

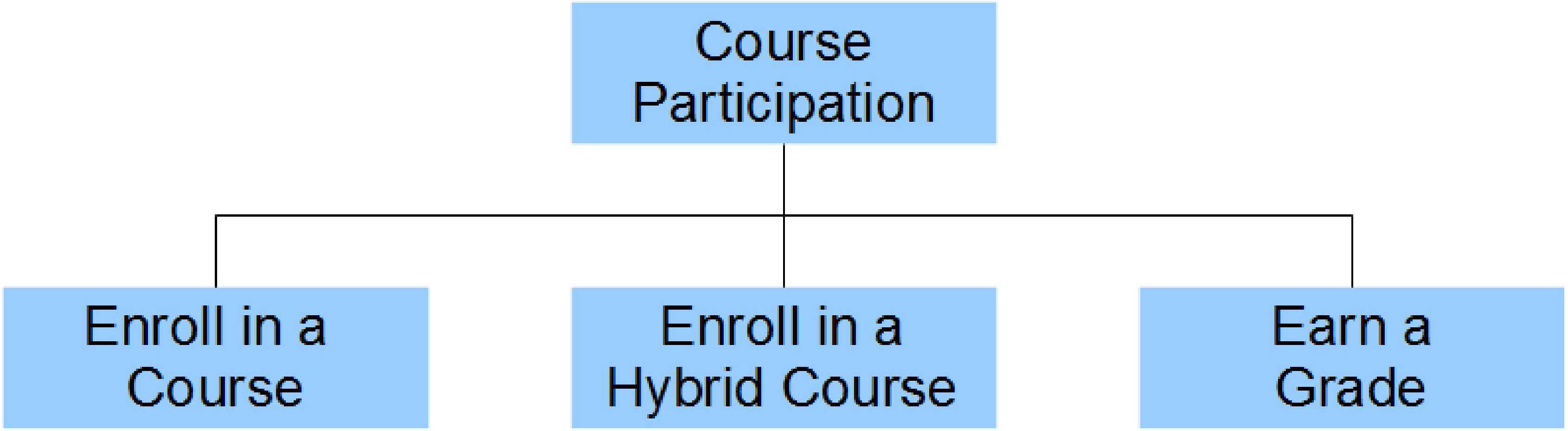
Complexity

- We create a software solution by **extracting** the **most important features** of the problem domain.
- There are **2 ways** to identify the most important features:
  - into **activities** (distinct algorithms)
  - into **things** (distinct objects)
- The two approaches are **not mutually exclusive**. We **start** with **one approach** and **use** its **results** as the **basis for the other**.
  - This decomposition is **an iterative process.**

Complexity (Example)

Consider a course enrollment system for a program in a college or university. Each participant

- enrolls in several face-to-face courses
- enrolls in several hybrid courses
- earns a grade in each course

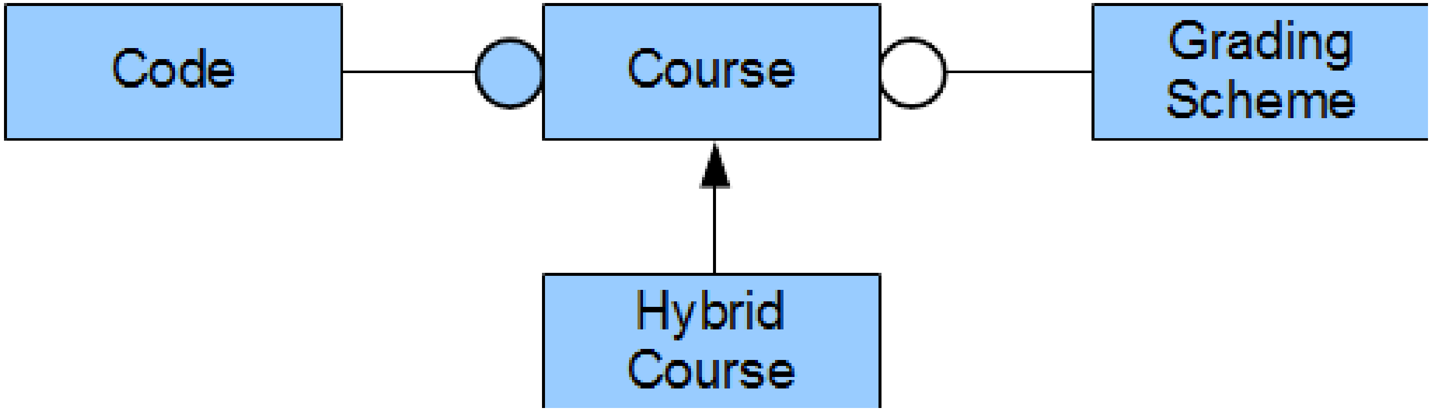


The following structure diagram identifies the **activities**.

Complexity (Example)

If we switch our attention to the objects involved, we find a **Course** and a **Hybrid Course**.

Course *has a* Code and *uses a* Grading Scheme and that a Hybrid Course is *a kind of* Course



The emphasis in this diagram is on the **objects** rather than the functional activities performed on them. The functional activities become part of the description of the objects themselves.

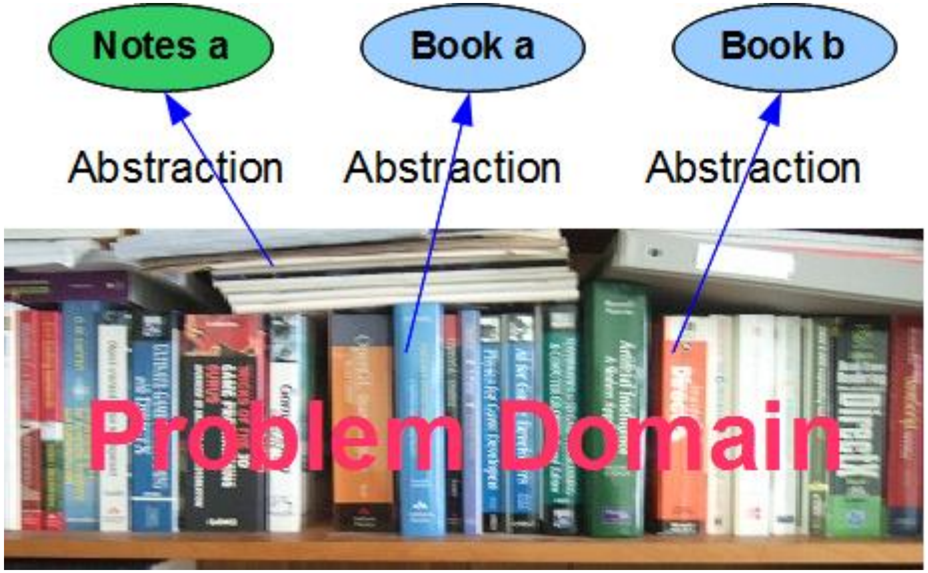
Object Terminology

There are **four fundamental** concepts

- Abstraction
- Encapsulation
- Hierarchy
- Polymorphism

Abstraction

- Abstraction **reduces the complexity** of a problem domain.
- Each object is **an abstraction of one important aspect** of the problem domain.
- The objects that make up the solution **<u>ignore</u> the <u>non-essential features</u>** of the problem.



Abstraction

- Each object has a **crisp boundary** that **distinguishes** the **object** from all other objects.
- Each object **has integrity**: it can only behave in ways that are appropriate to itself.
  - **Ex**
    - An ear cannot see, an eye cannot listen and a mouth cannot smell.
    - A horse cannot bark and a dog cannot croak.

Abstraction

- An **application** may **contain many objects**.
- Objects that have **similar features** and respond in a similar manner may **share a common structure.**

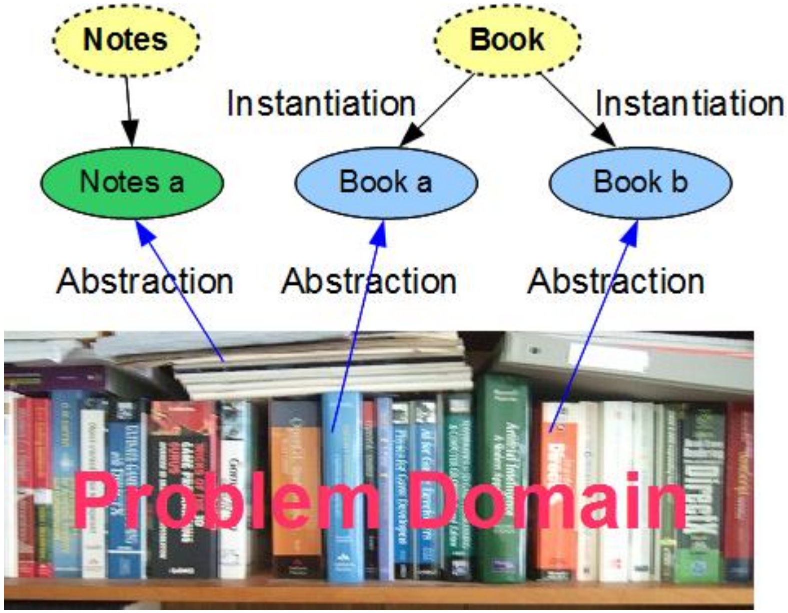
A **description of this common structure** is called a **class**. A class **describes** the **structure** of the **data** held by an object **and** the **behavior** of the object.

Abstraction – Classes & Objects

•An object may **have values** that **distinguish** it from another object in a class.

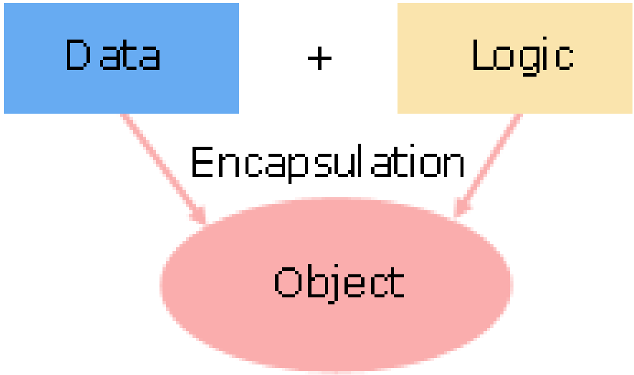
•The **values stored** in each object **may vary** from object to object, **but** the **set of variables and their data types are common**.

•Each **object is** an **instance** of the **class**. The terms object and instance are interchangeable.



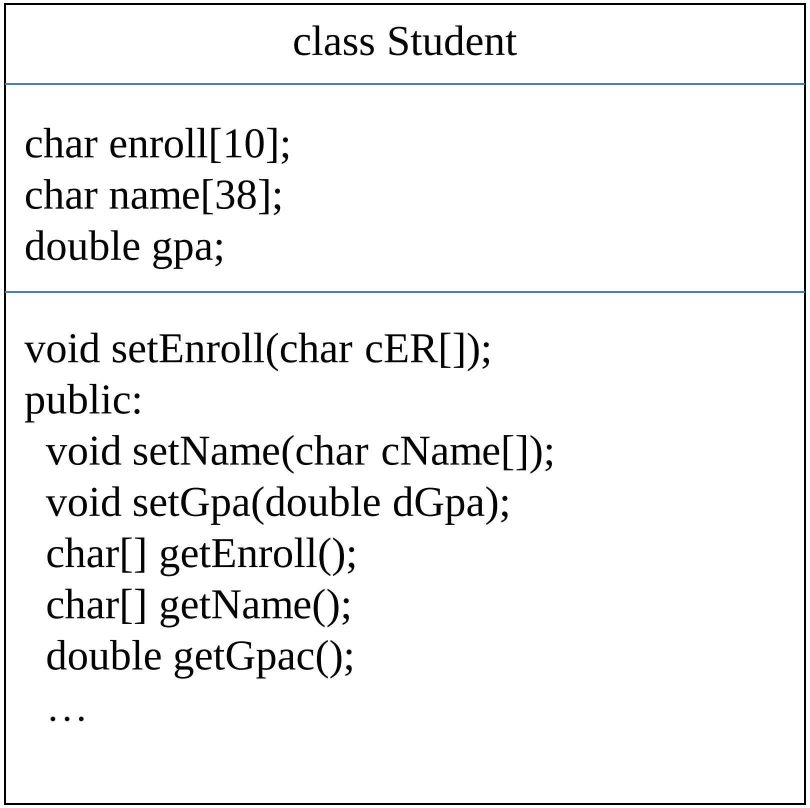
Encapsulation

- Encapsulation **separates** the **implementation details** of an object from its **external appearance**.
- Encapsulation **focuses** on the **interior** of an object, combining the data that describes the object's state and the algorithms that **define** its **behavior**.



Encapsulation

- A **well-encapsulated** object has all of its implementation **details hidden within** the object.
- If an object is well-encapsulated, a developer can **change** the object's **internal structure without introducing** any **changes** to the **software** that uses the object.



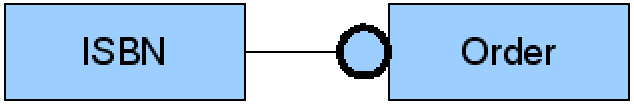
class Student

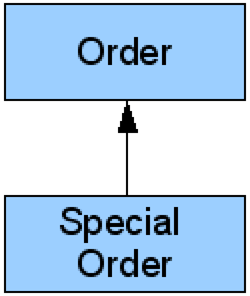
```
char enroll[10];
char name[38];
double gpa;
```

```
void setEnroll(char cER[]);
public:
  void setName(char cName[]);
  void setGpa(double dGpa);
  char[] getEnroll();
  char[] getName();
  double getGpac();
  …
```

Hierarchy

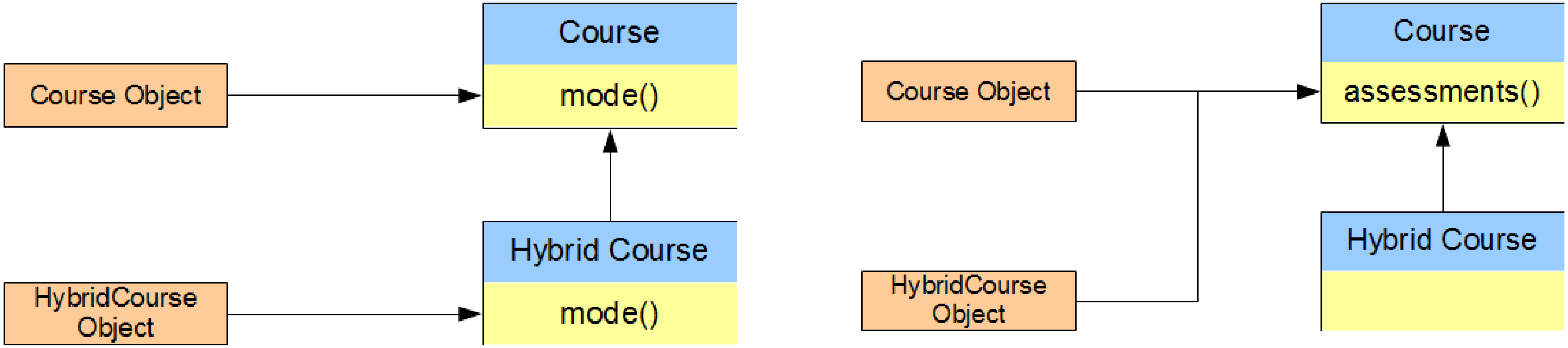
- Some of the objects in an application may be hierarchically related to one another. The hierarchy may be one of:
  - aggregation, or
  - shared structure and behavior
- **Aggregation** describes a "**has a**" relationship between objects. The parent object "has a" child object. The two objects **need not share a common structure.**
- **Shared structure and behavior** entails an "**is a kind of**" relationship. This appears as a **hierarchy** of classes. One class "is a kind of" another class





Polymorphism

Polymorphism relates the implementation for an object based on its type



- The HybridCourse object involves a different mode of delivery than the Course object, but the same assessments. Both objects belong to the same hierarchy: both are Course objects.
- A mode() query on a Course type reports a different result than a mode() query on a Hybrid Course type.

Summary

- Objects are abstractions of the most important chunks of information from a problem domain. They distinguish the different feature sets in the problem domain.
- A class describes the structure common to a set of similar objects. Each object in the set is a single instance of its class.
- Encapsulation hides the implementation details within a class - the internal data and internal logic are invisible to client applications that use objects of that class.
- We can upgrade the structure of a well-encapsulated class without altering any client code.
- The cornerstones of object-oriented programming are abstraction, encapsulation, inheritance and polymorphism.