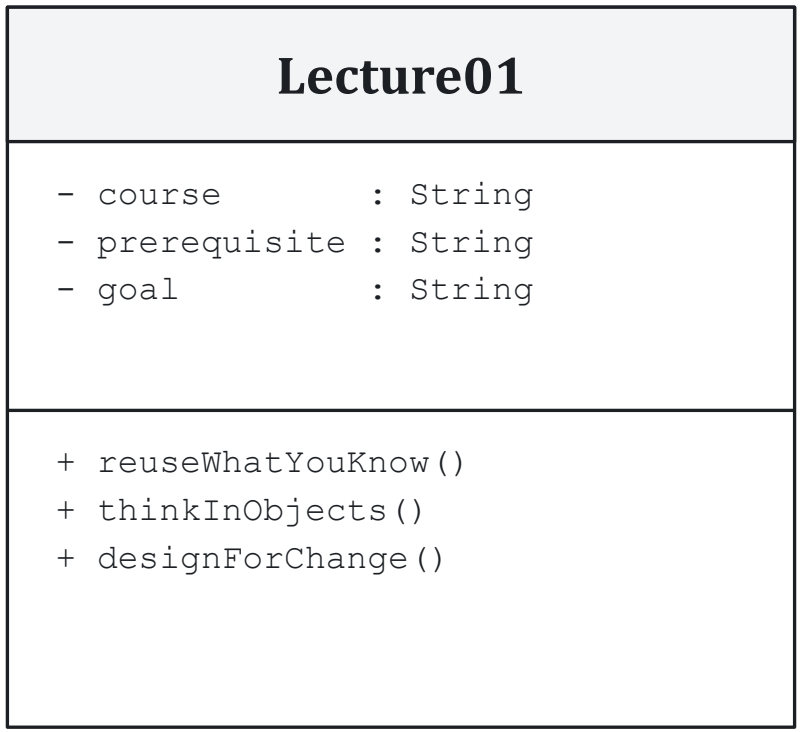
CSE 1115

Object-Oriented Programming

Lecture 01 — From C to Java: a new way to organize programs



Sheikh Tanvir Ahmed
Lecturer, Department of Computer Science and Engineering
United International University

THE BIG IDEA

You already know how to program

This course is about learning a different way to organize, design, and reason about programs.

01

What stays familiar

Variables, expressions, conditions, loops, arrays, and most basic control flow. Your C course is a foundation, not something to forget.

02

What changes

How we organize data and behavior, represent real entities, assign responsibilities, and structure larger programs.

03

Why it matters

Complex software becomes easier to understand, extend, maintain, and reuse — though OOP brings its own complexity and trade-offs.

TODAY'S ROUTE

Four steps from C to OOP

1

Familiarity

We revisit what you already know and see how much of it carries over to Java.

2

The question

If the syntax is so similar, why do we need a new course at all?

3

The motivation

How should we organize software as it grows larger and more complex?

4

The mental model

Classes, objects, and the four principles that the rest of the course builds on.

Recap of C: values and control flow

Variables

Typed, named storage for values.

```c
int age = 20;
double cgpa = 3.75;
char grade = 'A';

age = age + 1;
printf("%d %.2f %c\n",
       age, cgpa, grade);
```

Conditions

Choose a path based on a test.

```c
int mark = 72;

if (mark >= 80) {
    printf("A\n");
} else if (mark >= 60) {
    printf("B\n");
} else {
    printf("Try again\n");
}
```

Loops

Repeat work while a condition holds.

```c
int sum = 0;
for (int i = 1; i <= 5; i++) {
    sum += i;
}

int n = 3;
while (n > 0) {
    printf("%d ", n);
    n--;
}
```

Recap of C: data and reusable code

Arrays

A fixed-size sequence of same-typed values.

```c
int marks[5] = {70, 85, 62,
                90, 78};
int total = 0;

for (int i = 0; i < 5; i++) {
    total += marks[i];
}
printf("Avg: %.1f\n",
       total / 5.0);
```

Functions

Name a computation and reuse it.

```c
int max(int a, int b) {
    return (a > b) ? a : b;
}

int main(void) {
    int m = max(12, 7);
    printf("%d\n", m);
    return 0;
}
```

Strings

Characters in an array, ending in '\0'.

```c
#include <string.h>

char name[20] = "UIU";
printf("%s\n", name);
printf("%zu\n", strlen(name));

strcat(name, " CSE");
printf("%s\n", name);
```

Same logic, nearly the same syntax

C

```c
int x = 10;

if (x > 5) {
    printf("Greater\n");
}

for (int i = 0; i < 3; i++) {
    printf("%d ", i);
}
```

JAVA

```java
int x = 10;

if (x > 5) {
    System.out.println("Greater");
}

for (int i = 0; i < 3; i++) {
    System.out.print(i + " ");
}
```

Observe The declaration, the condition, and the loop's initialization, test, and update are identical. Only the output statement changed.

Functions in C, methods in Java

C

```c
int square(int x) {
    return x * x;
}

int main(void) {
    printf("%d\n", square(5));
    return 0;
}
```

JAVA

```java
public class MathDemo {
    static int square(int x) {
        return x * x;
    }

    public static void main(String[] args) {
        System.out.println(square(5));
    }
}
```

Observe The body of `square` is unchanged, but in Java it must live inside a class. For now, `static` lets us call it without creating an object.

Similar, but not identical

Concept	In C	In Java
Variables & expressions	int, double, char; familiar operators	Largely the same for common operations
if-else & loops	if, switch, for, while, do-while	The same control-flow structures
Arrays	int a[5]; fixed size, no bounds checking	int[] a = new int[5]; knows its length, bounds checked at run time
Functions	Standalone functions	Methods, declared inside classes
Strings	Character arrays ending in '\0', handled via <string.h>	String objects with their own methods: length(), equals()

PAUSE AND DISCUSS



If these constructs are already familiar, what exactly are we going to learn in this course?

One problem, two designs

A university system must keep student records and be able to:

1. Store a student's ID, name, and CGPA
2. Display a student's details
3. Update the CGPA when results arrive
4. Determine the student's academic status

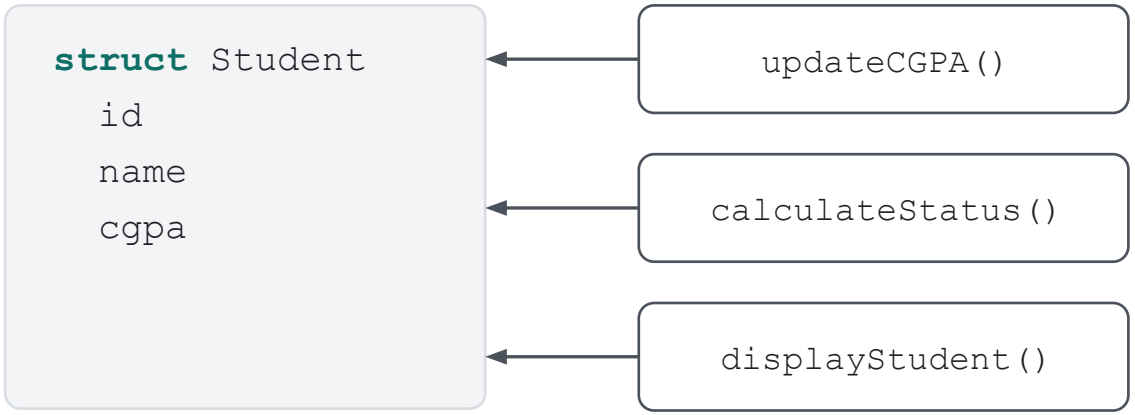
STUDENT RECORD

Student ID	**011292001**
Name	**Arif Hasan**
CGPA	**3.80**
Status	**Good standing**

Both C and Java can solve this. *The real question is how we organize the solution.*

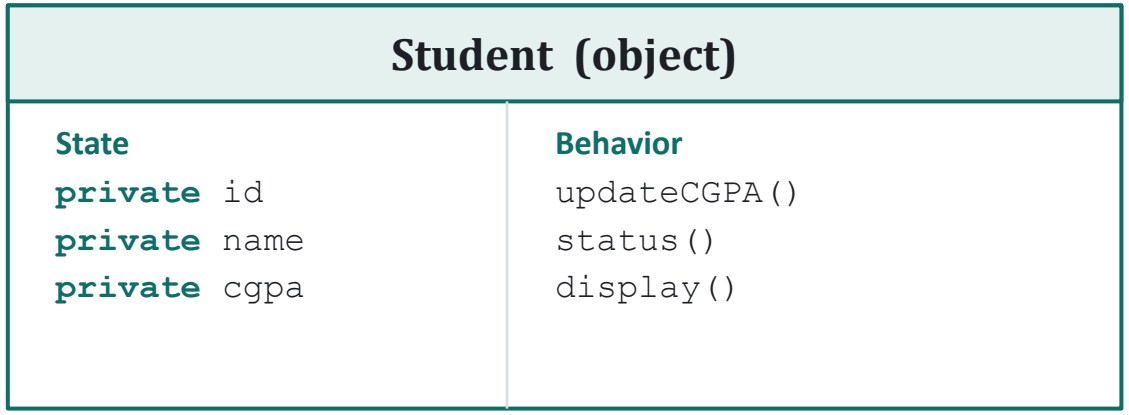
Where do the data and the behavior live?

APPROACH A · PROCEDURAL (C)



Data and the functions that operate on it are declared separately. Each function receives the data it needs as a parameter.

APPROACH B · OBJECT-ORIENTED (JAVA)



State and the behavior that acts on it are packaged together. Access control decides who may change the state.

The same record, written both ways

APPROACH A · C

```c
struct Student {
    char id[12];
    char name[50];
    double cgpa;
};

void updateCGPA(struct Student *s, double value) {
    if (value >= 0.0 && value <= 4.0)
        s->cgpa = value;
}

const char *calculateStatus(struct Student s) {
    return s.cgpa >= 2.0 ? "Good standing"
                         : "Probation";
}
```

APPROACH B · JAVA

```java
public class Student {
    private String id;
    private String name;
    private double cgpa;

    public Student(String id, String name,  double cgpa) {
        this.id = id;
        this.name = name;
        this.cgpa = cgpa;
    }

    public void updateCGPA(double value) {
        if (value >= 0.0 && value <= 4.0) cgpa = value;
    }

    public String status() {
        return cgpa >= 2.0 ? "Good standing"
                           : "Probation";
    }
}
```

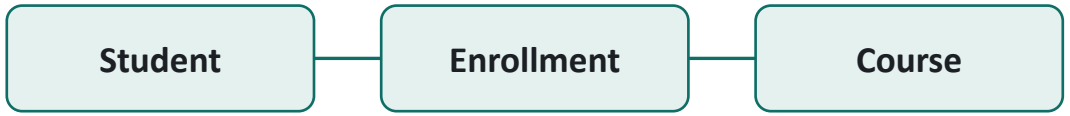
Observe Both validate the CGPA. In C, nothing stops `s.cgpa = 9.7;` elsewhere. In Java, `private` makes that line a compile-time error.

The difference is organization, not capability

A procedural design centers on functions and the sequence of operations.



An object-oriented design centers on entities that hold their own state and behavior.



A fair qualification

Procedural programs can also use structures and carefully designed modules. Object-oriented programs still use functions, loops, and conditionals.

OOP is one way to achieve modularity. It is not the only way, and it is not automatically superior for every problem.

THINK ABOUT IT

10 → **100,000**

students in the first version

students, plus registration, grades, payments, permissions

What difficulties might arise if the whole system is organized only as a growing collection of data and functions?

- Who is allowed to change a CGPA?
- What breaks if a field changes?
- Where does a new feature belong?

What OOP offers, and what it costs



Managing complexity

As a program grows, related responsibilities are grouped into coherent components.



Encapsulation

An object protects its internal state and exposes only controlled operations.



Extensibility and reuse

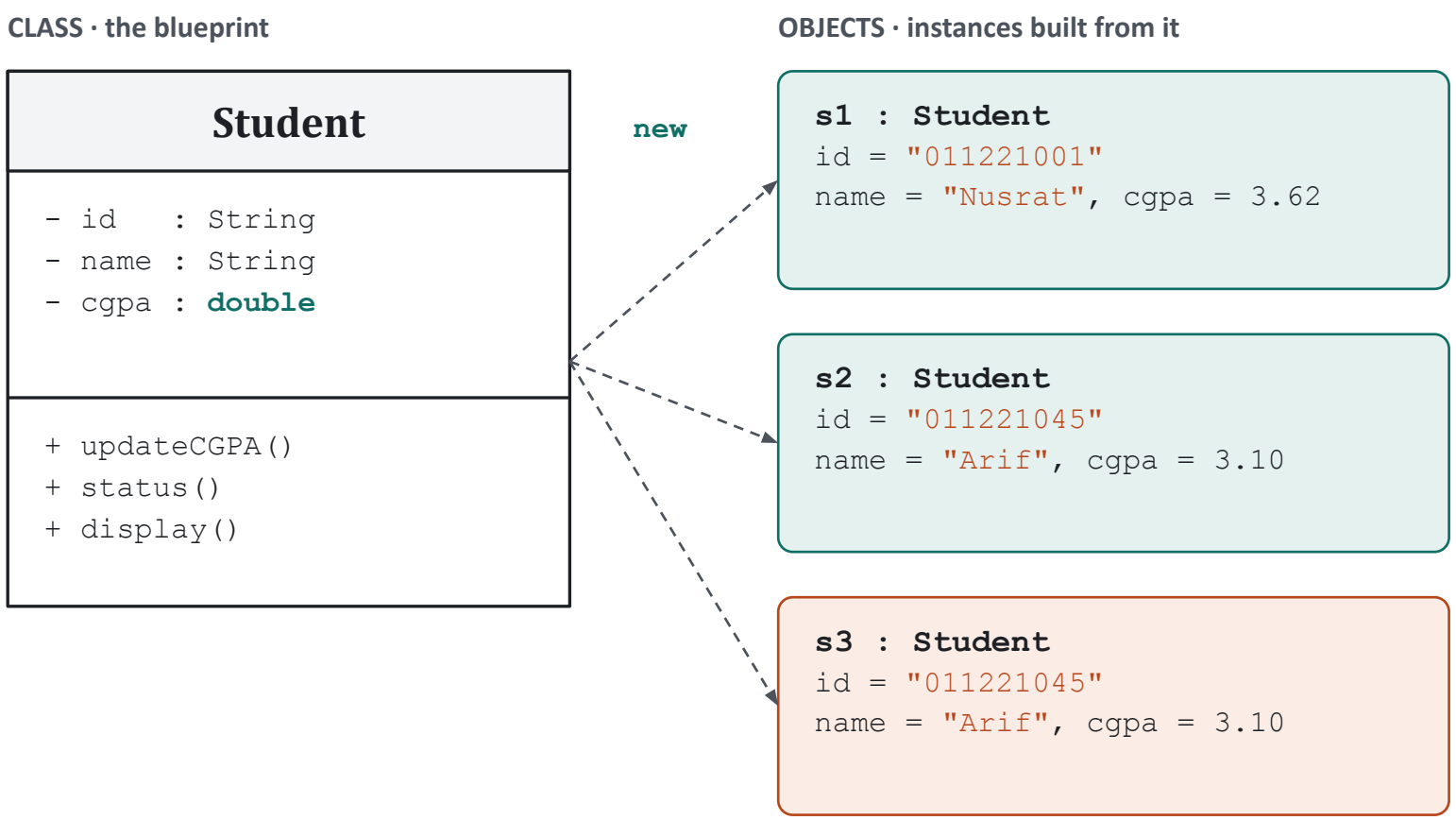
Classes, interfaces, composition, and inheritance support particular forms of reuse and variation.

The trade-offs

- More abstractions to learn and read
- More design decisions to make
- Risk of over-engineering simple problems
- Effort to understand how objects relate

The useful question is not "Is OOP better?" but ***"When does this approach help?"***

Classes and objects



State

The data an object holds right now: its field values.

Behavior

What an object can do: the methods that act on its state.

Identity

Each object is distinct. s2 and s3 hold identical values, yet they are two separate objects.

Four principles: a map of the course



Encapsulation

Group related state and behavior, and control access to the internal state.

How do we stop an object's data from being changed arbitrarily?



Abstraction

Expose the essential operations while hiding unnecessary implementation details.

How can someone use a component without knowing how it works inside?



Inheritance

Define a new class from an existing one, then extend or specialize its behavior.

When should one type be modeled as a specialized form of another?



Polymorphism

Work with different implementations through a common interface or superclass.

How can the same operation behave differently for different types?

Each principle gets its own weeks. Inheritance is one reuse tool among several; later we will compare it with composition.

Same paradigm, different design choices

Feature	Java	C++
Multiple inheritance of classes	Not allowed	Allowed
Multilevel inheritance	Supported	Supported
Runtime polymorphism	Instance methods dispatch dynamically by default	Only for functions declared virtual
Abstract methods	Abstract classes and interfaces	Pure virtual functions (= 0)
Diamond problem	Avoided for classes; interfaces have their own rules	Possible; virtual inheritance shares the base
Memory management	Garbage collected	Scope-based lifetime and RAII; no built-in garbage collector

A common misconception

Java is often called a “pure” object-oriented language. It is not, strictly.

It has primitive types such as int and boolean, and static members that belong to no object. C++ supports OOP alongside procedural and other styles.

TAKEAWAY

What to remember from today

01

Familiarity

We already know the fundamentals of programming.

02

The question

If the syntax is similar, why do we need OOP?

03

The motivation

How should we organize increasingly complex software?

04

The mental model

Classes, objects, and the four principles.

You are not starting programming from zero.
You are learning to think about program structure differently.