CS-1201 Object Oriented Programming

Classes, Objects and Constructors

Arbish Akram

Department of Computer Science
Government College University

Classes

- A **class** is a user-defined data type that acts as a blueprint for creating objects.
- It encapsulates data for the object (attributes) and methods to manipulate that data (functions).
- A class typically defines:
  - **Attributes**: Variables that hold the state of an object.
  - **Methods**: Functions that define the behavior of the object.
- Classes do not occupy memory until an object is created (instantiated).
- Think of a class as a plan, template, or blueprint.

Objects

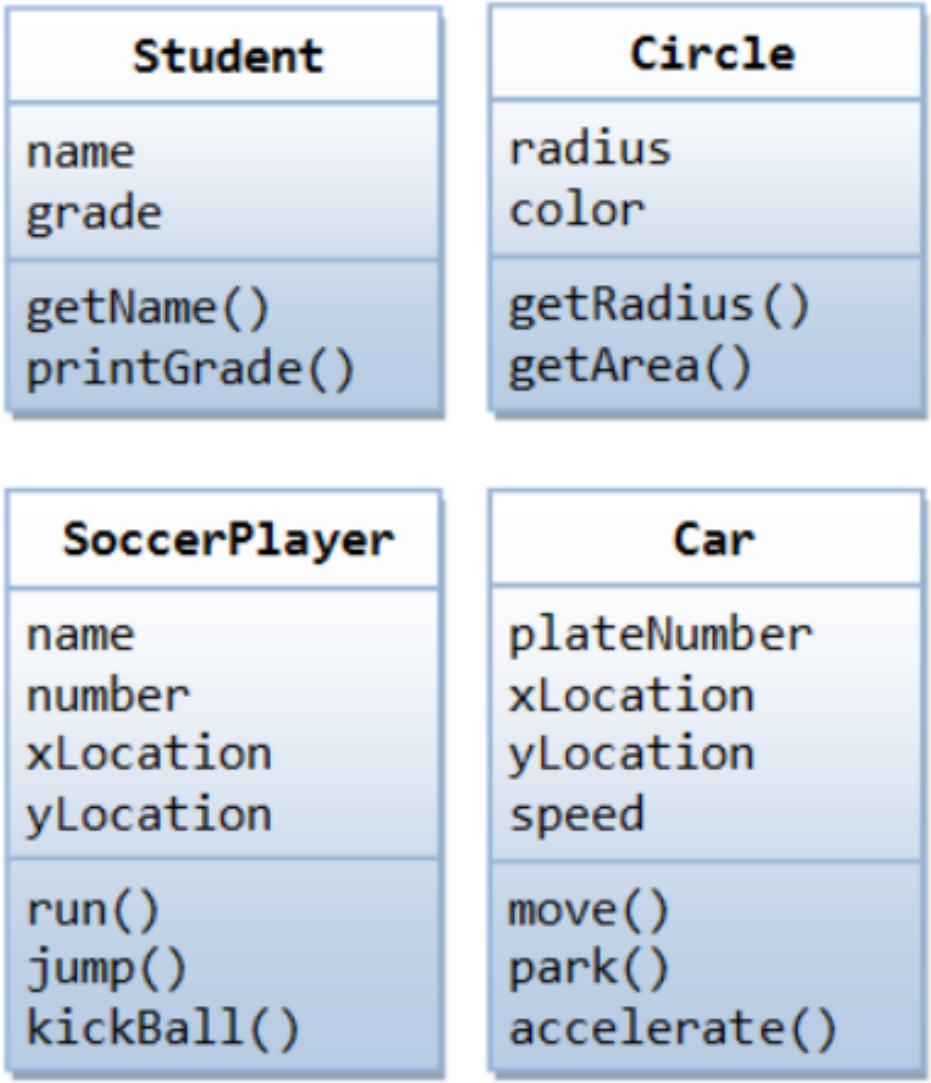
- An **object** is an instance of a class, meaning it is created based on the class definition.
- Objects hold specific data that conforms to the structure defined by their class.
- Each object can have unique values for its attributes.
- Key characteristics:
  - **State**: Defined by the values of the object's attributes.
  - **Behavior**: The actions (methods) the object can perform.
  - **Identity**: Each object is distinct, even if its state and behavior are identical to another object.
- Think of an object as a real-world entity created from a blueprint (class).

Class

- A class can be visualized as a three-compartment box.
  1. Class name (or identifier): identifies the class.
  2. Data Members or Variables (or attributes, states, fields): contains the static attributes of the class.
  3. Member Functions (or methods, behaviors, operations): contains the dynamic operations of the class.

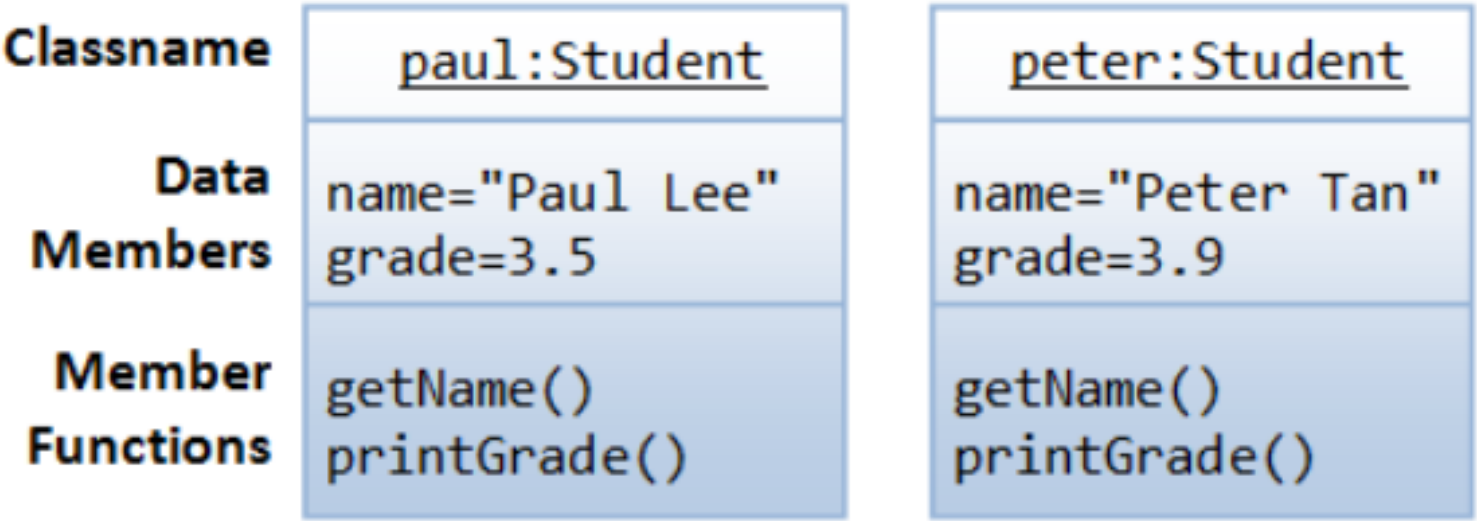
Class

Classname
(Identifier)
Data Member
(Static attributes)
Member Functions
(Dynamic Operations)

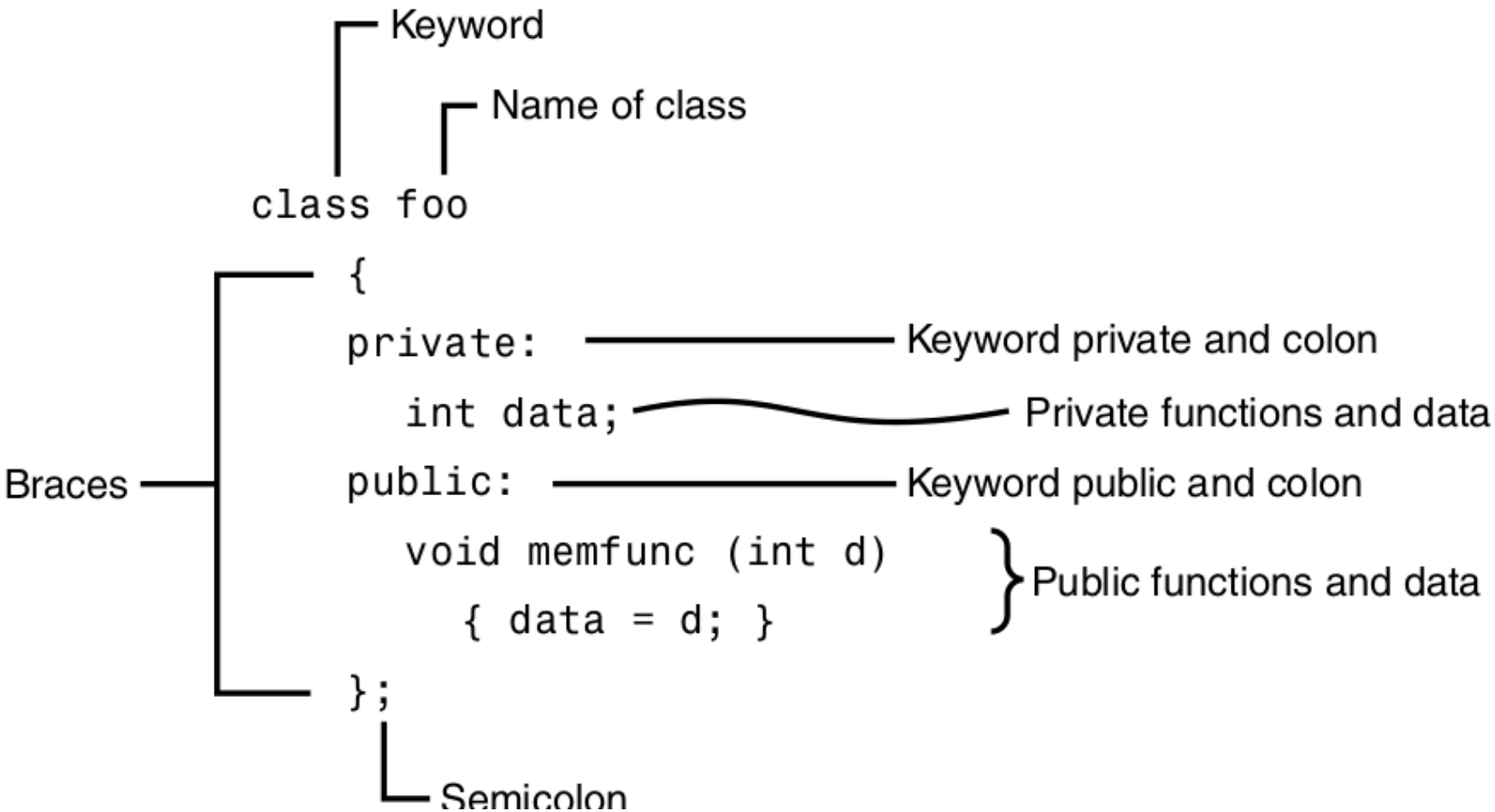


Examples of classes

Class


Classname
paul:Student
Data Members
name="Paul Lee"
grade=3.5
Member Functions
getName()
printGrade()
peter:Student
name="Peter Tan"
grade=3.9
getName()
printGrade()

Class


Keyword
Name of class
class foo
{
private:
Keyword private and colon
int data;
Private functions and data
Braces
public:
Keyword public and colon
void memfunc (int d)
{ data = d; }
Public functions and data
};
Semicolon

Class

- A `class` is a user-defined data type in C++ that serves as a blueprint for creating objects.
- It encapsulates data members (attributes) and member functions (methods) into a single unit.
- Help in organizing code and promoting reusability.

```
1  class Car {
2
3  public:
4      string brand;
5      string model;
6      int year;
7
8      void start() {
9          cout << "Car started!" << endl;
10     }
11 };
```

Object

- An object is an instance of a class.
- Objects are created using the class as a blueprint, with their own distinct values for attributes.
- Objects can perform actions using the methods defined in their class.

```
1  int main() {
2      Car myCar;  // Create an object of the Car class
3      myCar.brand = "Toyota";
4      myCar.model = "Corolla";
5      myCar.year = 2020;
6
7      myCar.start();  // Call the start method on the object
8      return 0;
9  }
```

Access Modifiers

- Access Modifiers determine the accessibility of members (attributes and methods) of a class.
- By default, all members of a class are private.
- Three types of access modifiers:
  - **public:** Members declared as ‘public‘ are accessible from outside the class.
  - **private:** Members declared as ‘private‘ are only accessible within the class itself.
  - **protected:** Members declared as ‘protected‘ are accessible within the class and by derived classes.

Example

```
1  class Car {
2      private:
3          string engineNumber;
4
5      public:
6          string brand;
7          string model;
8          int year;
9
10         void setEngineNumber(string number) {
11             engineNumber = number;
12         }
13
14         string getEngineNumber() {
15             return engineNumber;
16         }
17 };
```

- 'engineNumber' is private, accessible only through public methods.
- 'brand', 'model', and 'year' are public, accessible directly by any object.

Example

```
1  class Base{
2  private:
3      string privateData = "Private Data";  // Accessible only within the Base class
4  public:
5      string publicData = "Public Data";  // Accessible from outside the class
6      // Function to demonstrate access to private data within the class
7      void showPrivateData() {
8          cout << "Base class accessing private data: " << privateData << endl;
9      }
10     // Function to demonstrate access to public data within the class
11     void showPublicData() {
12         cout << "Base class accessing public data: " << publicData << endl;
13     }
14 };
15 int main() {
16     Base Obj;
17     // Accessing data from Base object
18     Obj.showPrivateData();  // Accessing private data through a public function
19     Obj.showPublicData();  // Accessing public data through a public function
20     // Direct access to data members
21     // cout << baseObj.privateData << endl;  // Error: privateData is not accessib
22     cout << "Accessing public data directly: " << Obj.publicData << endl;  // pub
23     return 0;
24 }
```

Constructors

- A constructor is a special member function of a class.
- It is automatically called when an object of the class is created.
- Constructors are used to initialize objects.
- The name of the constructor is the same as the class name.
- Constructors do not have a return type.
- A class can have more than one constructor. However, all constructors of a class have the same name.

Types of Constructors

- Default Constructor:
  - A constructor with no parameters.
  - Automatically provided by the compiler if no constructors are defined.
- Parameterized Constructor:
  - A constructor that takes arguments to initialize an object with specific values.
- Copy Constructor:
  - A constructor that creates a new object as a copy of an existing object.

Example

```
1  class Car {
2  public:
3      string brand;
4      string model;
5      int year;
6      Car() { // Default Constructor
7          brand = "Unknown";
8          model = "Unknown";
9          year = 0;
10     }
11     Car(string b, string m, int y) { // Parameterized Constructor
12         brand = b;
13         model = m;
14         year = y;
15     }
16 };
17 int main() {
18     Car car1;  // Calls Default Constructor
19     Car car2("Toyota", "Corolla", 2020);  // Calls Parameterized Constructor
20     cout<<"Car1: "<<car1.brand<<", "<<car1.model<<", "<<car1.year<<endl;
21     cout<<"Car2: "<<car2.brand<<", "<<car2.model<<", "<<car2.year<<endl;
22     return 0;
23 }
```

Example

```
1  class Counter
2  {
3      private:
4          unsigned int count;
5      public:
6          //constructor
7          Counter() : count(0) {}
8          void inc_count(){
9              count++;
10         }
11         int get_count(){
12             return count;
13         }
14 };
15 int main()
16 {
17     Counter c1;
18     //define and initialize
19     cout << "\nc1=" << c1.get_count();
20     c1.inc_count();
21     cout << "\nc1=" << c1.get_count()<<endl;
22     return 0;
23 }
```

Constructor: Initialization list

- Constructors often initialize data members.
  - Instead of:

```
Counter() { count = 0; }
```

  - Preferred Approach:

```
Counter() : count(0) { }
```

- Why Use Initialization List?
  - Initialization lists initialize members before the constructor body executes.

```
class someClass {
public:
    someClass() : m1(7), m2(33), m3(4) { }
private:
    int m1, m2, m3;
};
```

  - Members are separated by commas in the initialization list.