Object Oriented Concepts

Humayun Kabir
Professor, CS, Vancouver Island University, BC, Canada

Outline

- Objects and Classes
- Message Passing
- Encapsulation

Real World Objects

Real-world objects have ***attributes*** and ***behaviors.***

Examples:

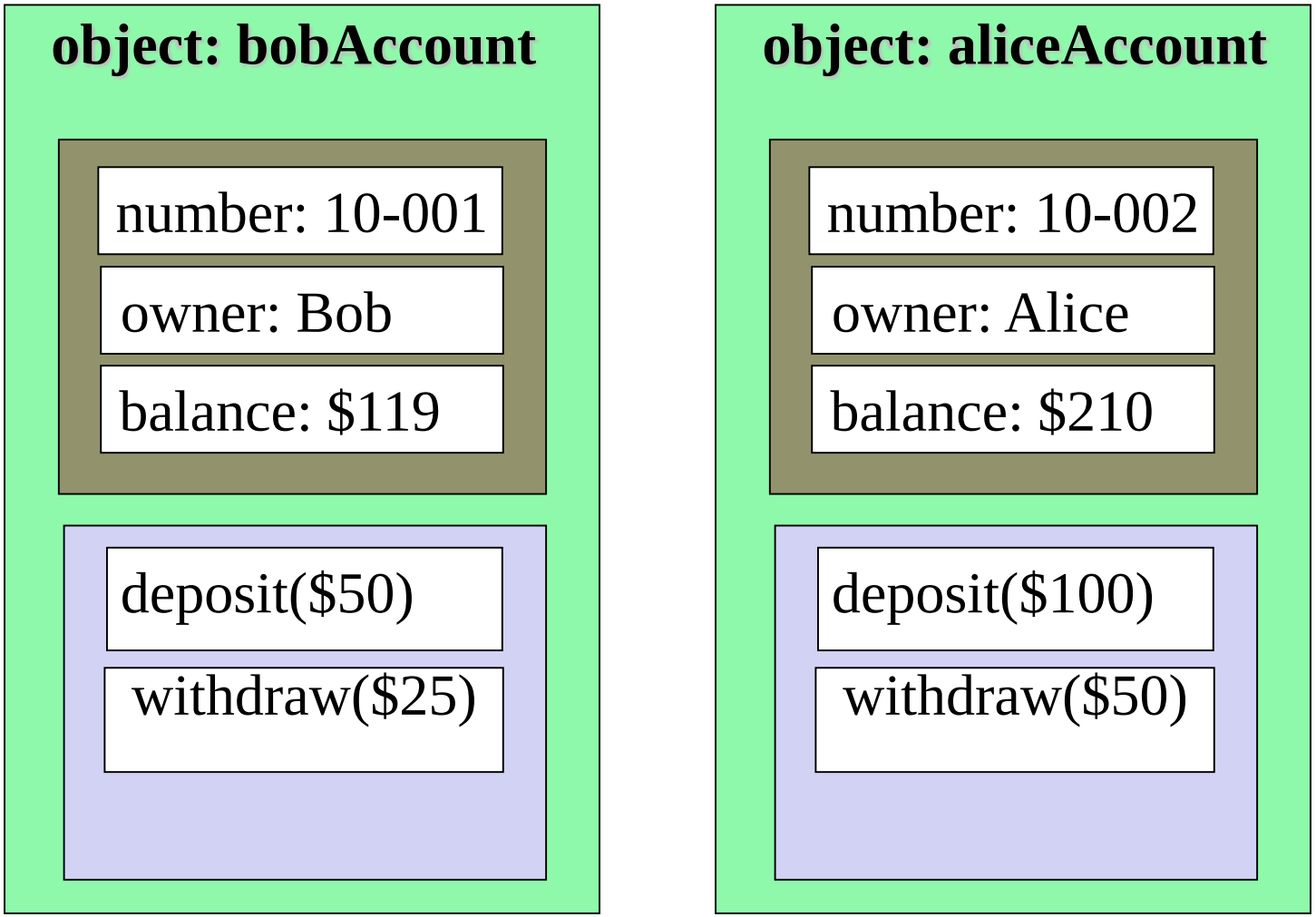
- Dog (**Physical**)
  - **Attributes**: breed, color, hungry, tired, etc.
  - **Behaviors**: eat, sleep, bark, run etc.
- Bank Account (**Conceptual**)
  - **Attributes**: account number, owner, balance
  - **Behaviors**: withdraw, deposit

Software Objects

Software objects are **conceptual objects**, a computational model of real-world objects and processes, also have **attributes (state)** and **behaviors (operations)**.

- Object's attributes are represented as object's **variables**.
- Object's Behaviors are represented as object's **functions** or **methods**.
- Only the functions of an object should operate on its variables.

Software Objects: Example

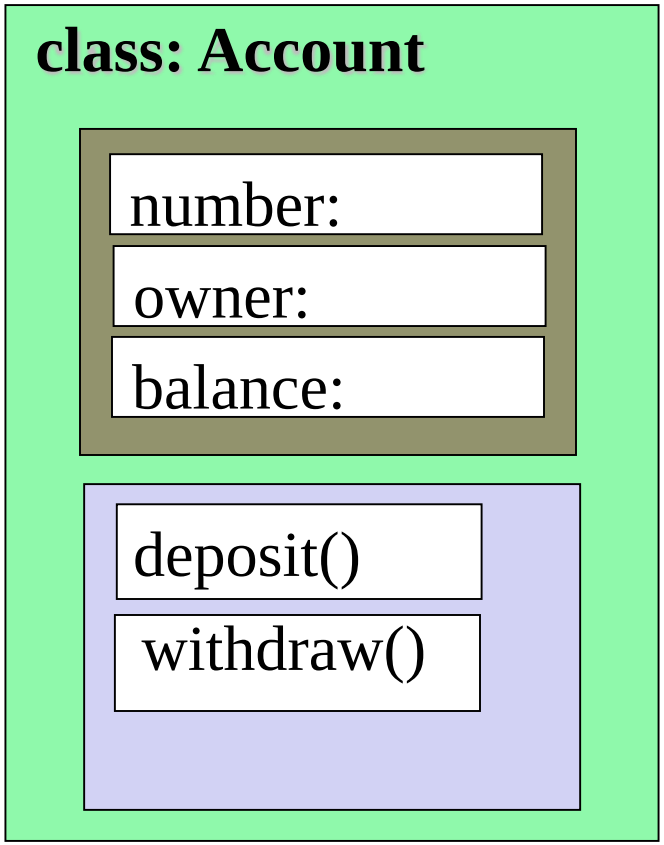


Class

- The definitions of the **attributes** and **behaviours** of **similar objects** are organized into a ***class (type)***, e.g., *Account* as a generic definition for *Bob*, *Alice* and *Jack's Accounts*)
- A class can be thought of as a model used to create a set of objects.
- A class is a static definition; a piece of code written in a programming language.
- One or more objects of a class are ***instantiated*** at runtime.
- The objects from a class are called ***instances*** of the class.

Class: Example

- The "account" class describes the attributes and behaviors of the bank accounts of many people.
- The “account” class is defined by three **state variables** (account **number**, **owner**, and **balance**) and two **functions** (**deposit** and **withdraw**).



Class

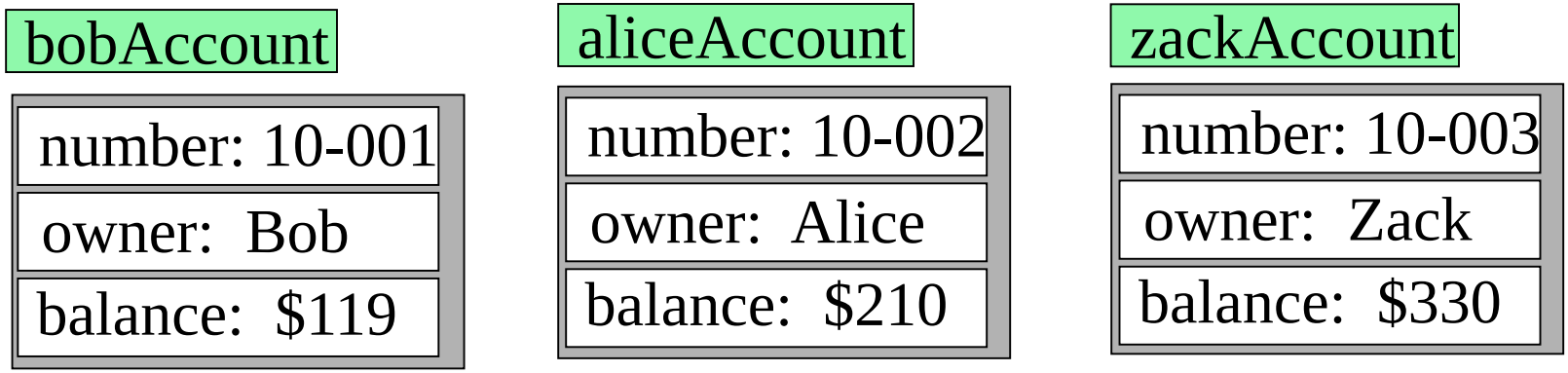
- Every instance of the same class will have the **same set of variables**.
- Each instance will however have its **own copy of the variable** set with **distinct values**. A variable of a class in two instances are essentially two distinct and independent variables except they are of the same type, one does not affect the other anyway.
- Every instance of the same class will also have the **same set of functions**.
- **Instance functions** simply point to their class counter parts instead making individual copy of the same function.

Object-Oriented Programming

- The programmer **defines** the **classes (types)** of the objects that will exist.
- The programmer **creates object instances** from the defined types as they are needed.
- The programmer specifies how these objects (various types) will **communicate** or **interact** with each other.
- Object-Oriented Programming (OOP) is a way to organize and conceptualize a program as a set of interacting objects.

Bank Example

- When the program runs there will be many instances of the account class.
- Each instance will have its own account number, owner, and balance (*object state*)



- Object's functions can only be invoked.

Classes

- Account class definition in C++.

```
class Account {
        private:
                string acctNumber;
                string acctOowner;
                double acctBalance;
        public:
                void setAcctNumber(string number) {acctNumber = number;}
                void setAcctOwner(string owner) {acctOwner = owner;}
                void setAcctBalance(double balance) {acctBalance = balance;}
                string getAcctNumber() {return acctNumber;}
                string getAcctOwner() {return acctOwner;}
                double getAcctBalance() {return acctBalance;}
                void deposit(double amount) {acctBalance += amount;}
                void withdraw(double amount) {acctBalance -= amount;}
};
```

Classes

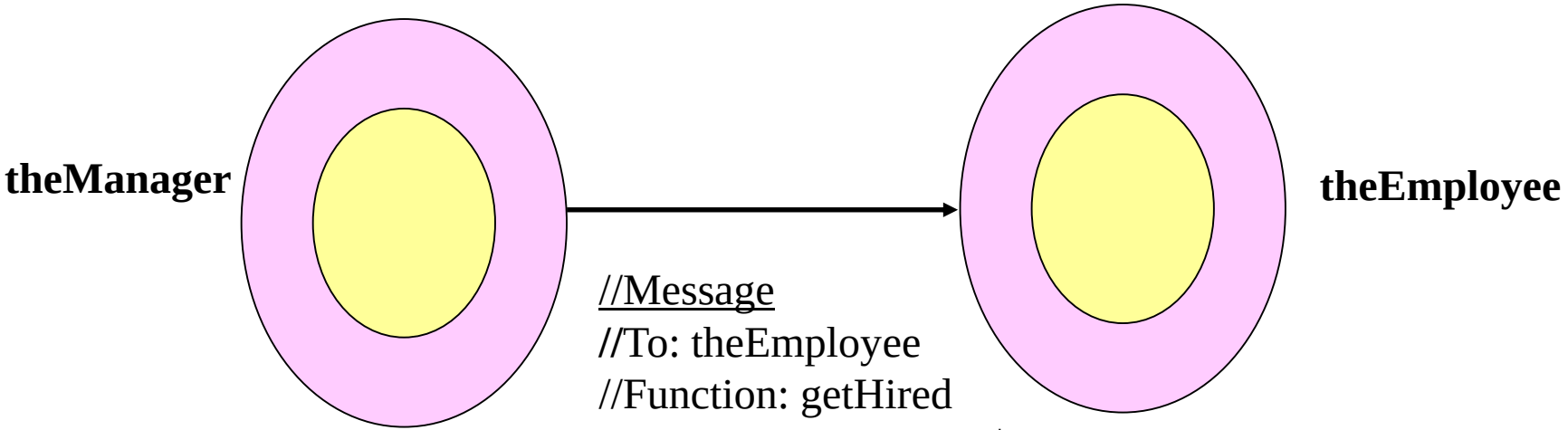
- Account class usage in C++.

```
int main ( ) {
        Account bobAccount;
        bobAccount.setAcctNumber("10-001");
        bobAccount.setAcctOwner("Bob");
        bobAccount.setAcctBalance(119.0);
        Account aliceAccount;
        aliceAccount.setAcctNumber("10-002");
        aliceAccount.setAcctOwner("Alice");
        aliceAccount.setAcctBalance(210.0);
        bobAccount.deposit(50,0);
        aliceAccount.withdraw(60.0);
        cout << "Account: " << bobAccount.getAcctNumber() <<
                              " Owner: " << bobAccount.getAcctOwner() <<
                              " Balance:  $" << bobAccount.getAcctBalance() <<endl;
        cout << "Account: " << aliceAccount.getAcctNumber() <<
                              " Owner: " << aliceAccount.getAcctOwner() <<
                              " Balance:  $" << aliceAccount.getAcctBalance() << endl;
        return 0;
}
```

Object's Interactions: Messages

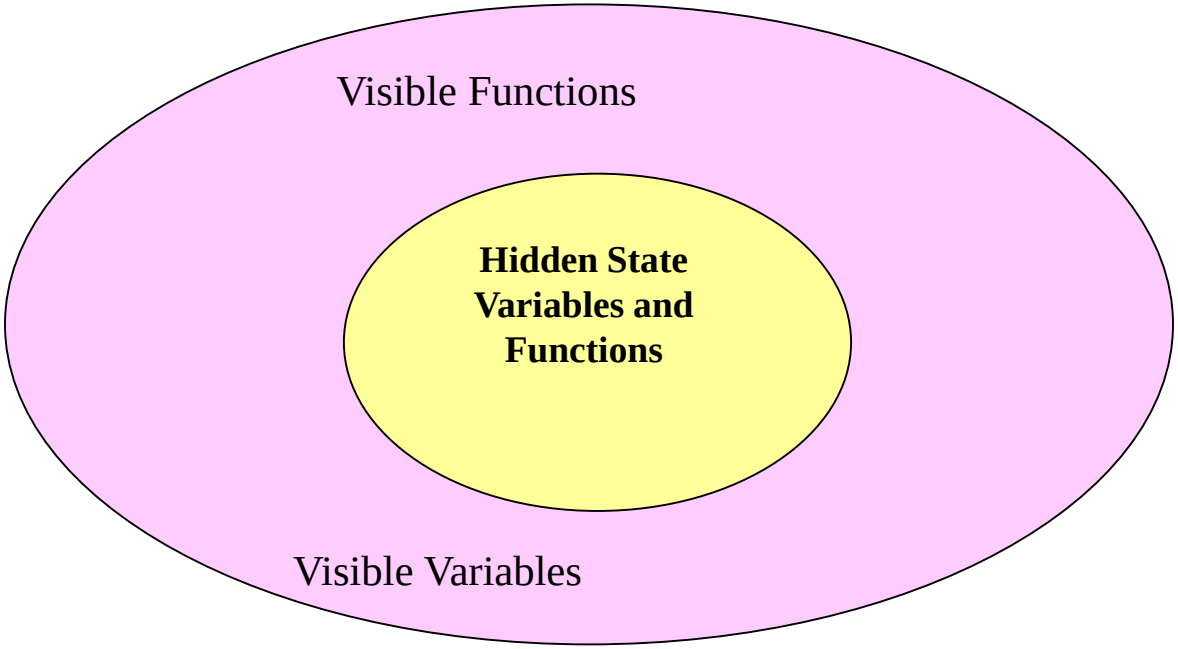
- Conceptually, one object communicates with another object by **message passing**.

- Message components include:
  - The name of the object to receive the message.
  - The name of the service (function execution) to perform.
  - Any parameters needed for the function.

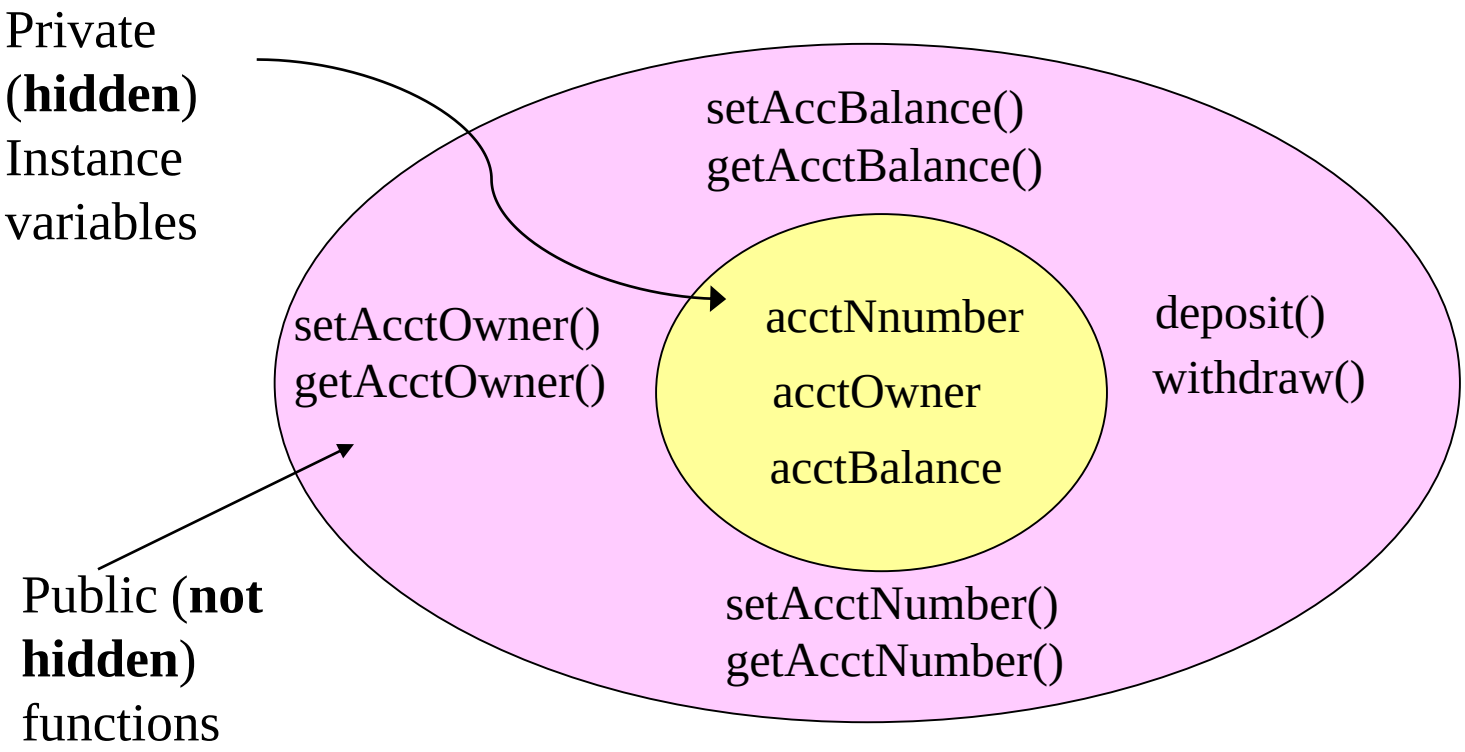


Encapsulation

- When classes are defined, programmers can specify that **certain functions** and **variables remain hidden** inside the class.
- These variables and functions are accessible from within the class, but not accessible from the outside.
- The mechanism to place all the variables and the functions of an object into a single class definition and selectively hide them from external access is known as **encapsulation**.



Encapsulation



State variables make up the nucleus of the object. Functions surround and hide (encapsulate) the state variables from the rest of the program.