CS202 Java Object Oriented Programming

Introduction to Classes and Objects

Chengyu Sun
California State University, Los Angeles

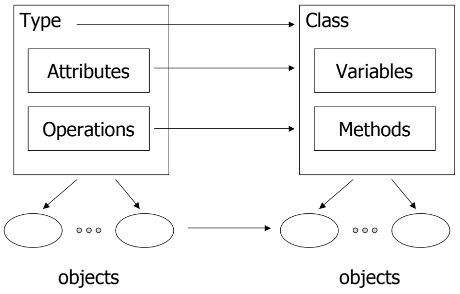
Overview

- Class
  - Variables and variable scope
  - Methods
    - Constructors and garbage collection
    - Keyword `this`
- Object
  - Reference
    - Assignment, equality, and array of objects
    - Pass by reference and pass by value
- Keyword `static`

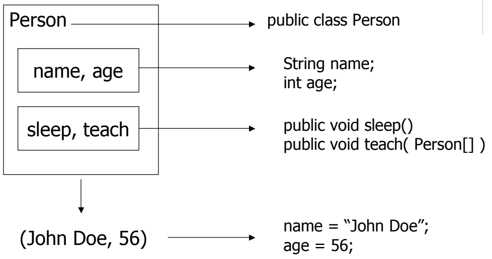
Philosophy of Object Oriented Programming Languages

- The world consists of objects
- Each object is associated with some attributes and operations
  - Attributes
    - Name, age, height, weight, eye color etc.
  - Operations
    - Walk, talk, sleep, take etc.
    - Sit on a chair, drive a car, read a book …
- The same type of objects share the same attributes and operations

From Concept to Code



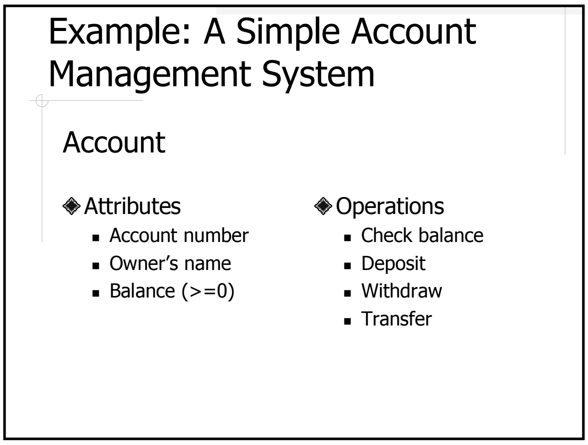
From Concept to Code Example



```
public class Person

String name;
int age;

public void sleep()
public void teach( Person[] )

name = "John Doe";
age = 56;
```

Benefits of OO Programming

- Encapsulation
- Inheritance
- Polymorphism

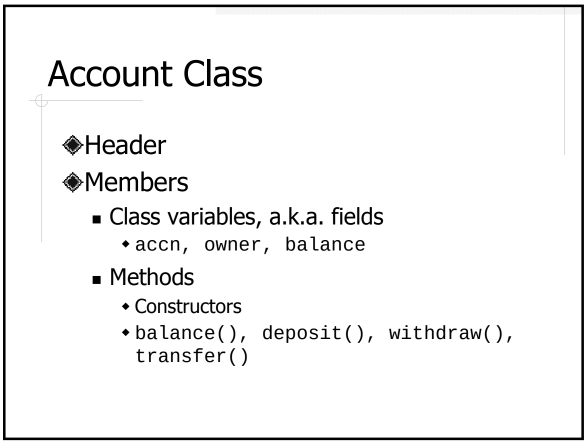


Example: A Simple Account Management System

Account

◈ Attributes
- Account number
- Owner's name
- Balance (>=0)

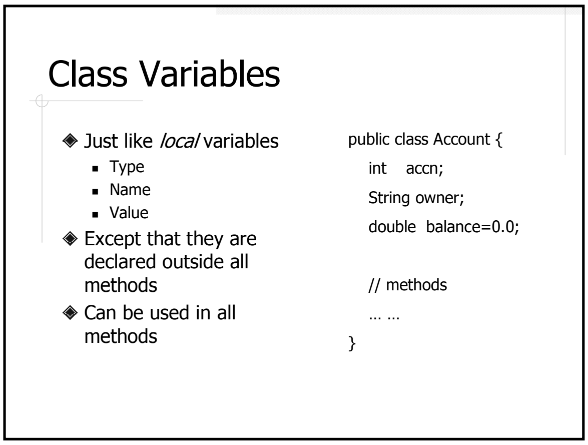
◈ Operations
- Check balance
- Deposit
- Withdraw
- Transfer



Account Class

◈ Header

◈ Members
- Class variables, a.k.a. fields
  - `accn, owner, balance`
- Methods
  - Constructors
  - `balance(), deposit(), withdraw(), transfer()`



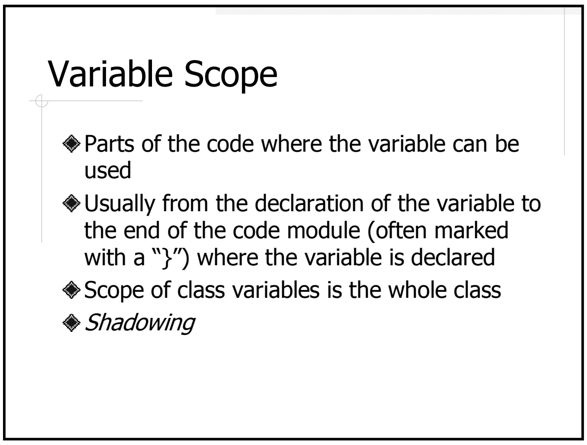
Class Variables

◈ Just like *local* variables
- Type
- Name
- Value

◈ Except that they are declared outside all methods

◈ Can be used in all methods

```
public class Account {
   int    accn;
   String owner;
   double  balance=0.0;


   // methods
   … …
}
```



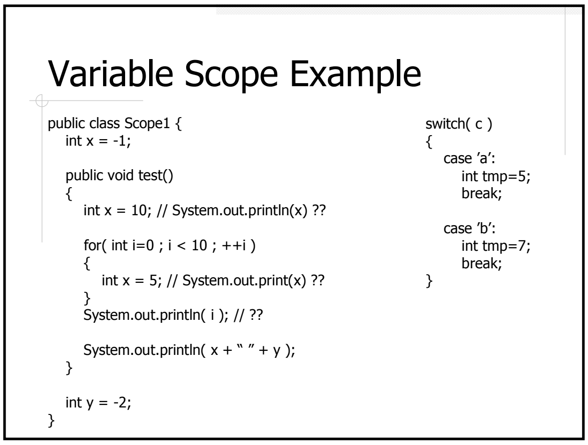
Variable Scope

◈ Parts of the code where the variable can be used

◈ Usually from the declaration of the variable to the end of the code module (often marked with a "}") where the variable is declared

◈ Scope of class variables is the whole class

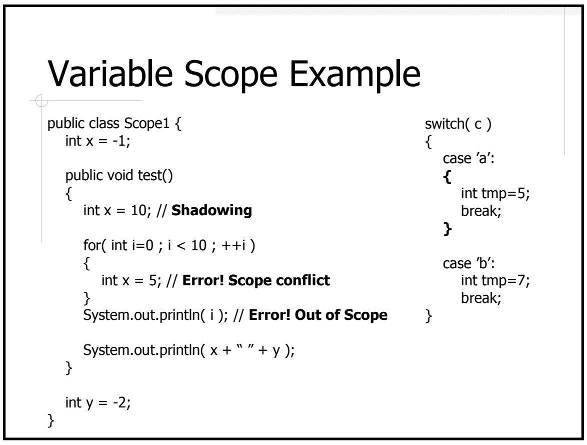
◈ *Shadowing*



Variable Scope Example

```
public class Scope1 {
   int x = -1;

   public void test()
   {
      int x = 10; // System.out.println(x) ??

      for( int i=0 ; i < 10 ; ++i )
      {
         int x = 5; // System.out.print(x) ??
      }
      System.out.println( i ); // ??

      System.out.println( x + " " + y );
   }

   int y = -2;
}
```

```
switch( c )
{
   case 'a':
      int tmp=5;
      break;

   case 'b':
      int tmp=7;
      break;
}
```



Variable Scope Example

```
public class Scope1 {
   int x = -1;

   public void test()
   {
      int x = 10; // Shadowing

      for( int i=0 ; i < 10 ; ++i )
      {
         int x = 5; // Error! Scope conflict
      }
      System.out.println( i ); // Error! Out of Scope

      System.out.println( x + " " + y );
   }

   int y = -2;
}
```

```
switch( c )
{
   case 'a':
   {
      int tmp=5;
      break;
   }

   case 'b':
      int tmp=7;
      break;
}
```

Constructors of Account

```
/** Constructor. creates an account with zero balance */
public Account( int accn, String owner )
{
    this.accn = accn;
    this.owner = owner;
}

/** Constructor. creates an account */
public Account( int accn, String owner, double balance )
{
    this( accn, owner );
    this.balance = balance > 0 ? balance : 0;
}
```

Constructors

- A special type of methods
  - Name is the same as the class name
  - No return type (not even `void`)
- Purpose
  - Allocate the memory
  - Initialize fields
- There could be more than one constructors
  - Default constructor `Classname()`
  - A constructor can call another constructor as the *first* statement of the constructor

Overloading

- Methods have the same name but different signatures

```
System.out.println( char )
System.out.println( boolean )
System.out.println( int )
System.out.println( String )

… …
```

Keyword `this`

- A reference to an object itself
  - De-shadowing
- A reference to a constructor

```
int x = -1;

void foo()
{
    int x = 10;

    System.out.println( x );
    System.out.println( this.x );
}
```

Garbage Collection

- There are no *destructors* in Java
- Freeing memory allocated to objects is done automatically – garbage collection
- Advantage
  - Simplifies programming
  - Safer and more robust programs
    - No dangling pointers
    - Greatly reduced memory leaks
- Disadvantages
  - Less efficient

Other Methods of Account

- double balance()
- double deposit(double amount)
- double withdraw(double amount)
- double transfer(double amount, Account a)

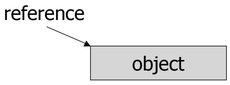
Usage of Classes

- Declaration
- Allocation and initialization
- Calling class methods
- Classes versus Objects

```
Account a; // declaration

// allocation and initialization
a = new Account( 100000, "Chengyu", 10 );

// 3 in 1
Account b = new Account( 100001, "Sun", 20 );

a.deposit( 20 );
b.withdraw( 30 );

a.transfer( 10, b );
```

Object Reference

- Object name is also called the *reference* of the object
  - Similar to *pointer* in C/C++



Object Assignment

```
public class Foo {

      int n;

   public Foo() { n = 0 }

   public Foo( Foo f ) { n = f.n; }

   public void inc() { ++n; }

   public void print()
   {
      System.out.println(n);
   }
}
```

```
Foo a = new Foo();
Foo b = a;
Foo c = new Foo(a);

a.inc();
b.inc();
c.inc();

a.print(); // ??
b.print(); // ??
c.print(); // ??
```

Object Equality

- By reference
  - ==
- By value
  - equals()

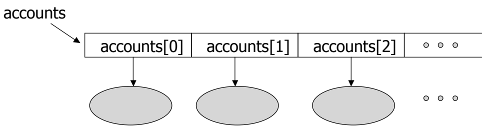
```
System.out.println( a == b); // ??
System.out.println( a == c); // ??
```

Add another method to Foo:

```
public boolean equals( Foo a )
{
    return n == a.n;
}
```

Array of Objects

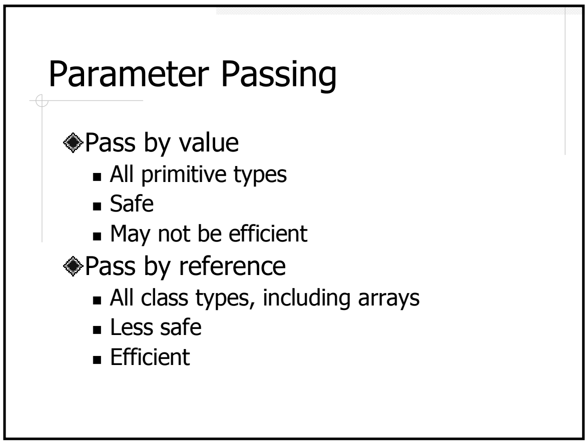
```
Account accounts[];

accounts = new Account[1000]; // allocation of references

// initialization has to be done for each element
Accounts[0] = new Account( 100000, "Chengyu", 10.0 );
Accounts[1] = new Account( 100001, "Sun", 20.3 );
… …
```



Parameter Passing Example

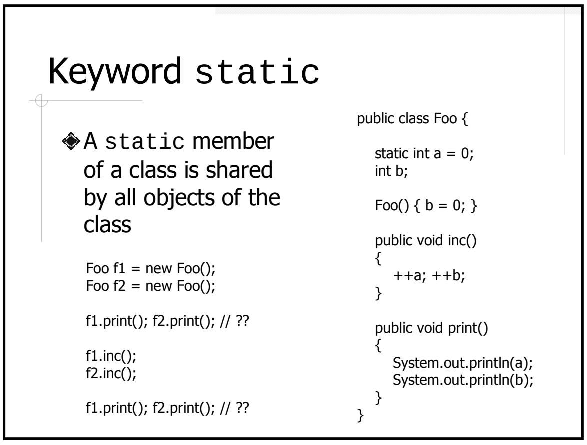
```
public class Foo {

   public int n = 0;

}
… …

void inc( int a, Foo f )
{
   ++a;
   ++f.n;
}
```

```
int a = 0;

Foo f = new Foo();

inc( a, f );

System.out.println( a );    // ??
System.out.println( f.n );  // ??
```



Parameter Passing

- Pass by value
  - All primitive types
  - Safe
  - May not be efficient
- Pass by reference
  - All class types, including arrays
  - Less safe
  - Efficient

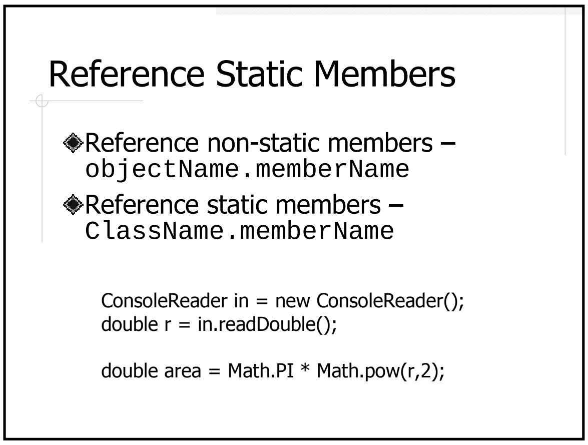


Keyword `static`

- A `static` member of a class is shared by all objects of the class

```
Foo f1 = new Foo();
Foo f2 = new Foo();

f1.print(); f2.print(); // ??

f1.inc();
f2.inc();

f1.print(); f2.print(); // ??
```

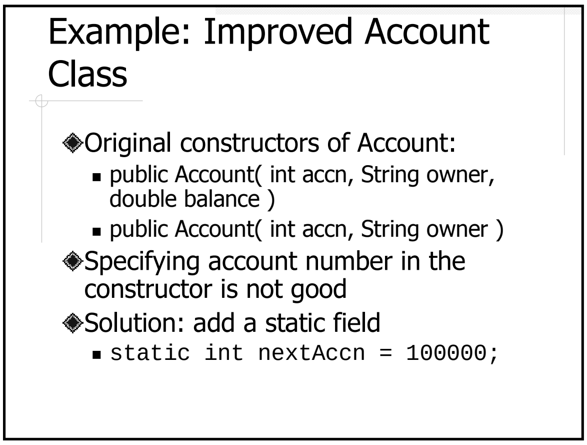
```
public class Foo {

    static int a = 0;
    int b;

    Foo() { b = 0; }

    public void inc()
    {
        ++a; ++b;
    }

    public void print()
    {
        System.out.println(a);
        System.out.println(b);
    }
}
```



Reference Static Members

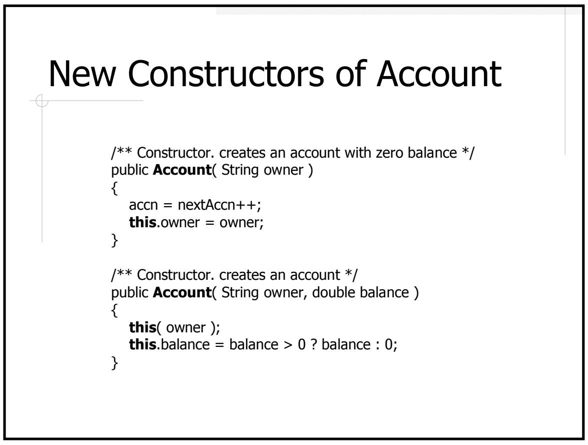
- Reference non-static members – `objectName.memberName`
- Reference static members – `ClassName.memberName`

```
ConsoleReader in = new ConsoleReader();
double r = in.readDouble();

double area = Math.PI * Math.pow(r,2);
```



Example: Improved Account Class

- Original constructors of Account:
  - public Account(int accn, String owner, double balance)
  - public Account(int accn, String owner)
- Specifying account number in the constructor is not good
- Solution: add a static field
  - `static int nextAccn = 100000;`



New Constructors of Account

```
/** Constructor. creates an account with zero balance */
public Account( String owner )
{
    accn = nextAccn++;
    this.owner = owner;
}

/** Constructor. creates an account */
public Account( String owner, double balance )
{
    this( owner );
    this.balance = balance > 0 ? balance : 0;
}
```