



Object-Oriented Programming

CBOP3203

Object-Oriented Programming (I)

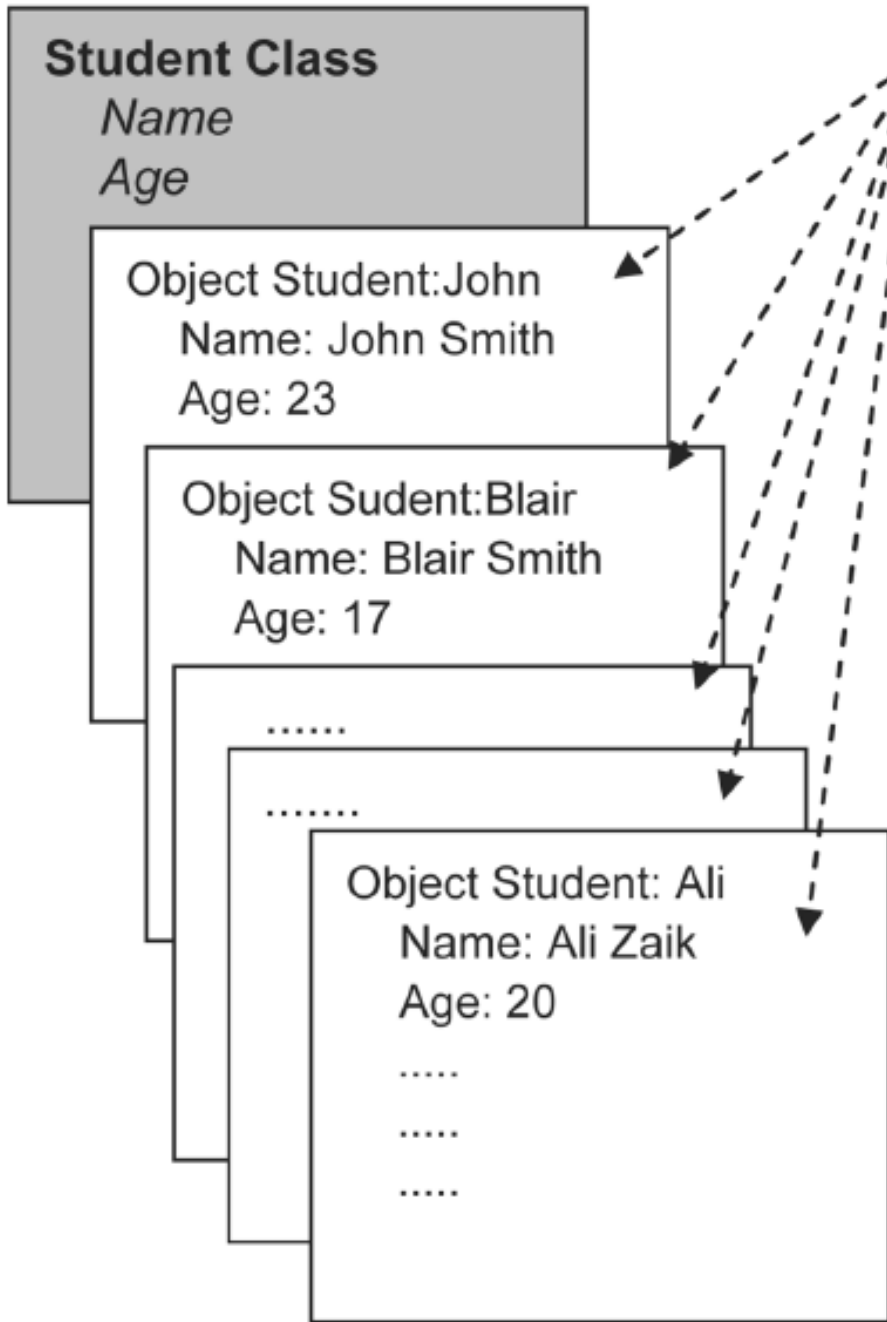
- ▶ **OBJECT-ORIENTATION** is a set of tools and methods that enable software engineers to build reliable, user friendly, maintainable, well documented, reusable software systems that fulfills the requirements of its users.
- ▶ An object-oriented programming language provides support for the following object oriented concepts:
 - Objects and Classes
 - Inheritance
 - Polymorphism and Dynamic binding

Advantages of OO programs

- ▶ It allows reusability of codes
- ▶ Programs are easy to maintain
- ▶ Programs are more flexible and expandable

OBJECTS AND CLASSES

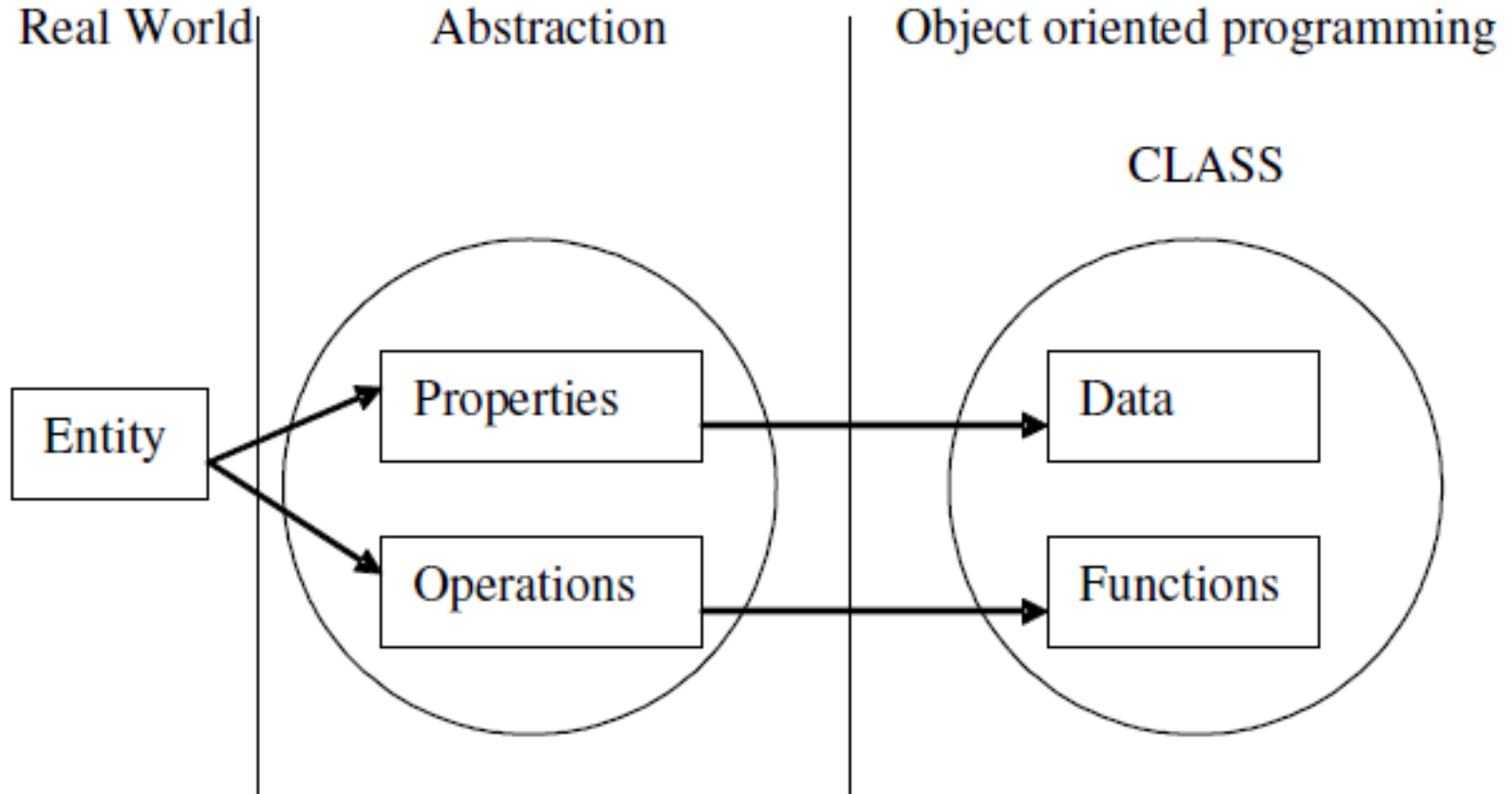
- ▶ All **objects** are unique.
- ▶ Examples of objects:
 - people, animals, plants, cars, planes, buildings, computers and so on
- ▶ What is the difference between a class and primitive data type?
- ▶ A **class** is a blueprint that defines the variables (or attributes) and the methods common to all objects of a certain kind.



Objects of the class Student

Objects created from the class Student

Mapping real world entity to OO programming



Object Characteristics

- ▶ Object is an instance of class.
- ▶ Objects can be defined as things that has:
 - State
 - Behavior
 - Identity
- ▶ Sate – represented by attributes:
 - Example: money in an ATM machine, name, birth date
- ▶ Behavior –methods are used to define behavior:
 - Example: release money from ATM machine
- ▶ Identity – name of the object:
 - StudentA, StudentB, StudentC and etc

Defining Classes

```
class <class_name> {  
    <attribute(s)>  
    <operation(s)>  
}
```

Class Name

```
public class Vehicle {  
    private double maxLoad;
```

Attribute

```
    public void setMaxLoad( double value ) {  
        maxLoad = value;  
    }  
}
```

Operation /method
declaration

Creating an Object from a Defined Class

- ▶ `Class_Name Object_Name = new Constructor_Name();`

```
Vehicle myVehicle = new Vehicle();
```

```
Vehicle objVehicle;  
objVehicle = new Vehicle();
```

INITIALIZING CLASS OBJECTS: CONSTRUCTORS

- ▶ A constructor is a set of instructions designed to initialize an instance. Parameters can be passed to the constructor in the same way as for a method. The declaration takes the following form:

```
<modifier> <class_name>  
  ([<parameter_list>])    {  
  [<statements>]  
}
```

Declaring Constructors – Example:

```
public class Dog {  
    private int weight;  
  
    public Dog() {  
        weight = 42;  
    }  
    public int getWeight() {  
        return weight;  
    }  
    public void setWeight (int newWeight) {  
        weight = newWeight;  
    }  
}
```

Differences between Constructor and Member Method

| Constructor | Member Method |
|---|--|
| The name of the constructor method is same as the name of the class | Member method can be given any meaningful name except the name of the class |
| Constructor method cannot have any return value | Member method may or may not return value |
| Constructor method is used to initialize attributes | Member method is used to perform some calculations such calculate age, get maximum number, get random number, etc. |

Accessing Object Members

- ▶ The “dot” notation: `<object>.<member>;`
- ▶ This is used to access object members including attributes and methods
- ▶ Example:
 - `d.setWeight(42);`
 - `d.weight = 42;` //only permissible if weight is public

OVERLOADED CONSTRUCTORS

```
public class Dog {  
    private int weight;
```

```
    public Dog() {  
        weight = 42;  
    }
```

```
    public Dog(int initialWeight) {  
        weight = initialWeight;  
    }
```

```
}
```



```
1 // Fig. 3.13: Account.java
2 // Account class with a constructor to
3 // initialize instance variable balance.
4
5 public class Account
6 {
7     private double balance; // instance variable that stores the balance
8
9     // constructor
10    public Account( double initialBalance )
11    {
12        // validate that initialBalance is greater than 0.0;
13        // if it is not, balance is initialized to the default value 0.0
14        if ( initialBalance > 0.0 )
15            balance = initialBalance;
16    } // end Account constructor
17
18    // credit (add) an amount to the account
19    public void credit( double amount )
20    {
21        balance = balance + amount; // add amount to balance
22    } // end method credit
23
24    // return the account balance
25    public double getBalance()
26    {
27        return balance; // gives the value of balance to the calling method
28    } // end method getBalance
29
30 } // end class Account
```

▶ Account.java

double variable balance

Outline

AccountTest.java

```
1 // Fig. 3.14: AccountTest.java
2 // Create and manipulate an Account object.
3 import java.util.Scanner;
4
5 public class AccountTest
6 {
7     // main method begins execution of Java application
8     public static void main( String args[] )
9     {
10         Account account1 = new Account( 50.00 ); // create Account object
11         Account account2 = new Account( -7.53 ); // create Account object
12
13         // display initial balance of each object
14         System.out.printf( "account1 balance: $%.2f\n",
15             account1.getBalance() );
16         System.out.printf( "account2 balance: $%.2f\n\n",
17             account2.getBalance() );
18     }
```

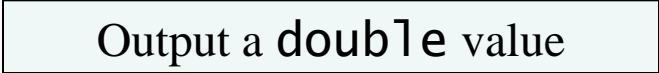


```
19 // create Scanner to obtain input from command window
20 Scanner input = new Scanner( System.in );
21 double depositAmount; // deposit amount read from user
22
23 System.out.print( "Enter deposit amount for account1: " ); // prompt
24 depositAmount = input.nextDouble(); // obtain user input
25 System.out.printf( "\nadding %.2f to account1 balance\n\n",
26     depositAmount );
27 account1.credit( depositAmount ); // add to account1 balance
28
29 // display balances
30 System.out.printf( "account1 balance: $%.2f\n",
31     account1.getBalance() );
32 System.out.printf( "account2 balance: $%.2f\n\n",
33     account2.getBalance() );
34
35 System.out.print( "Enter deposit amount for account2: " ); // prompt
36 depositAmount = input.nextDouble(); // obtain user input
37 System.out.printf( "\nadding %.2f to account2 balance\n\n",
38     depositAmount );
39 account2.credit( depositAmount ); // add to account2 balance
40
```

Input a double value

Input a double value

```
41 // display balances
42 System.out.printf( "account1 balance: $%.2f\n",
43     account1.getBalance() );
44 System.out.printf( "account2 balance: $%.2f\n",
45     account2.getBalance() );
46 } // end main
47
48 } // end class AccountTest
```



Output a double value

```
account1 balance: $50.00
account2 balance: $0.00

Enter deposit amount for account1: 25.53
adding 25.53 to account1 balance

account1 balance: $75.53
account2 balance: $0.00

Enter deposit amount for account2: 123.45
adding 123.45 to account2 balance

account1 balance: $75.53
account2 balance: $123.45
```



Exercise – 5

Q1. Modify class Account (in the example) to provide a method called debit that withdraws money from an Account. Ensure that the debit amount does not exceed the Account's balance. If it does, the balance should be left unchanged and the method should print a message indicating "Debit amount exceeded account balance." Modify class AccountTest (in the example) to test method debit.

Exercise – 5

Q2. Create a class called **Invoice** that a hardware store might use to represent an invoice for an item sold at the store. An **Invoice** should include four pieces of information as instance variables – a **part number** (type `String`), a **part description** (type `String`), a **quantity** of the item being purchased (type `int`) and a **price per item** (double). Your class should have a **constructor** that initializes the four instance variables. Provide a **set** and a **get method** for each instance variable. In addition, provide a method named **getInvoiceAmount** that calculates the invoice amount (i.e., multiplies the quantity by the price per item), then returns the amount as a double value. If the quantity is not positive, it should be set to 0. If the price per item is not positive, it should be set to 0.0. Write a test application named **InvoiceTest** that demonstrates class **Invoice**'s capabilities.

Exercise – 5

Q3. Create a class called **Employee** that includes three pieces of information as instance variables—a **first name** (type String), a **last name** (type String) and a monthly **salary** (double). Your class should have a **constructor** that initializes the three instance variables. Provide a **set** and a **get method** for each instance variable. If the monthly salary is not positive, set it to 0.0. Write a test application named **EmployeeTest** that demonstrates class **Employee**'s capabilities. Create two **Employee objects** and display each object's yearly salary. Then give each **Employee** a 10% raise and display each **Employee**'s yearly salary again.

Exercise – 5

Q4. Create a class called **Date** that includes three pieces of information as instance variables—a **month** (type int), a **day** (type int) and a **year** (type int). Your class should have a **constructor** that initializes the three instance variables and assumes that the values provided are correct. Provide a **set** and a **get method** for each instance variable. Provide a method **displayDate** that displays the month, day and year separated by forward slashes (/). Write a test application named **DateTest** that demonstrates class Date's capabilities.