Object Oriented Programming using Java

- Composition

Composition

- The concept of composition is not new; that's what we do to describe complex objects in the real world:
  - Every living creature and most manufactured products are made up of parts. Often, each part is a subsystem that is itself made up of its own set of subparts. Together, the whole system forms a composition hierarchy.
- Note the human body composition hierarchy on the next slide.

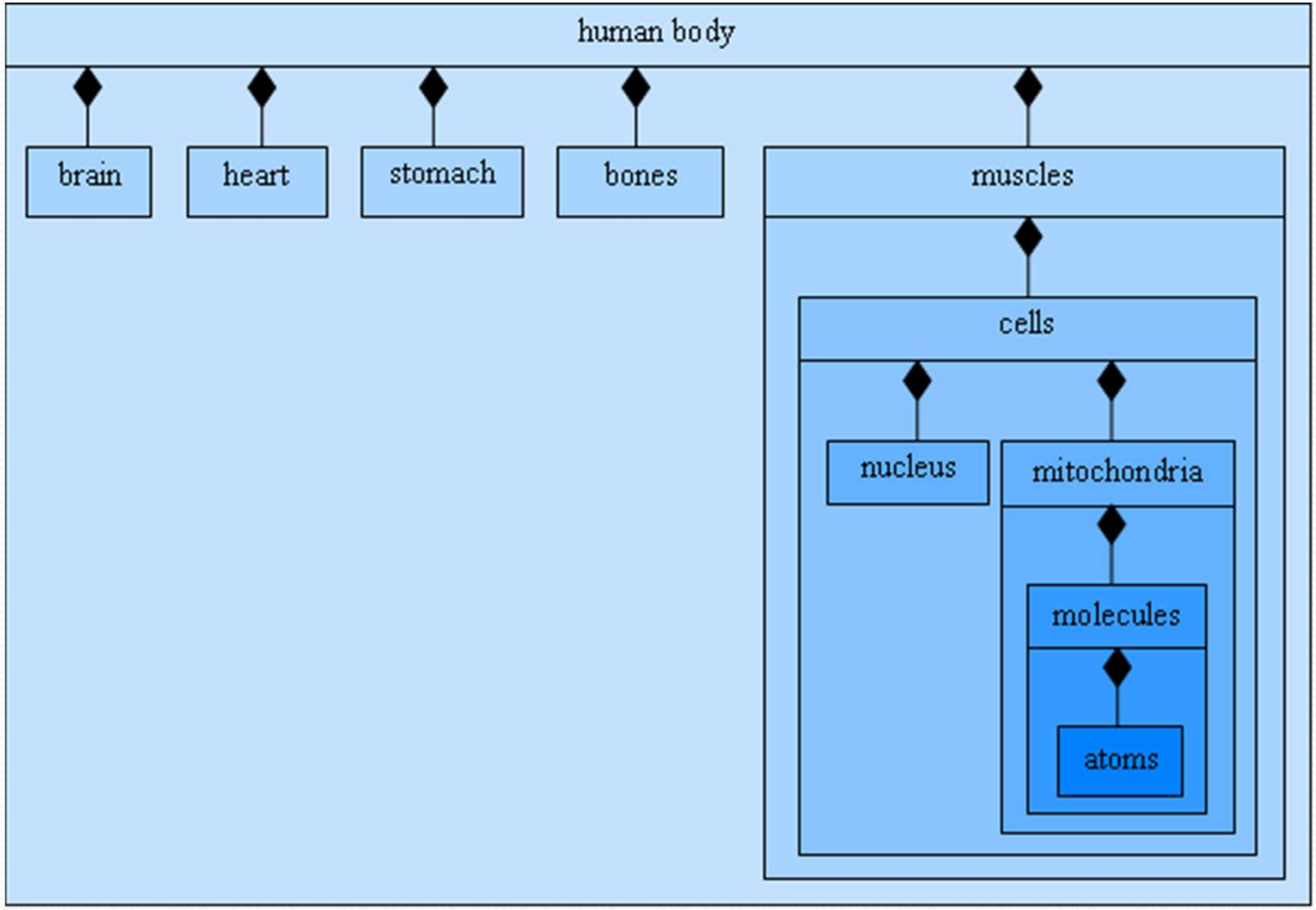
- Remember that with a composition relationship, a component part is limited to just one owner at a time. For example, a heart can be in only one body at a time.

Composition

- Prior to this chapter, all of our objects have been relatively simple, so we've been able to describe each object with just a single class.
- But for an object that's more complex, you should consider breaking up the object into its constituent parts and defining one class as the whole and other classes as parts of the whole. When the whole class is the exclusive owner of the parts classes, then that class organization is called a *composition*.

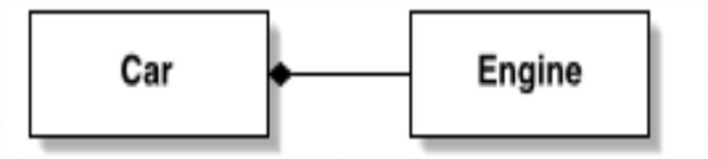
Composition

- A partial composition hierarchy for the human body:



Composition

- Composition
  - A class can have references to objects of other classes as members
  - Sometimes referred to as a *has-a* relationship
  - e.g., A car has an (has-a) engine



- One form of software reuse is composition, in which a class has as members references to objects of other classes.

Composition ("Has-a")

- A composition relation exists between two classes if one classes' field(s) consist of another class object

```
Class ABC
{
        private   XYZ   x1;
}
```

```java
// Fig. 8.7: Date.java
// Date class declaration.

public class Date
{
   private int month; // 1-12
   private int day;   // 1-31 based on month
   private int year;  // any year

   // constructor: call checkMonth to confirm proper value for month;
   // call checkDay to confirm proper value for day
   public Date( int theMonth, int theDay, int theYear )
   {
      month = checkMonth( theMonth ); // validate month
      year = theYear; // could validate year
      day = checkDay( theDay ); // validate day

      System.out.printf(
         "Date object constructor for date %s\n", this );
   } // end Date constructor

```

- Date.java
- (1 of 3)

```
22    // utility method to confirm proper month value
23    private int checkMonth( int testMonth )
24    {
25       if ( testMonth > 0 && testMonth <= 12 ) // valida
26          return testMonth;
27       else // month is invalid
28       {
29          System.out.printf(
30             "Invalid month (%d) set to 1.", testMonth );
31          return 1; // maintain object in consistent state
32       } // end else
33    } // end method checkMonth
34
35    // utility method to confirm proper day value based on month and year
36    private int checkDay( int testDay )
37    {
38       int daysPerMonth[] =
39          { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
40
```

Validates month value

Validates day value

- Date.java
- (2 of 3)

```
41       // check if day in range for month
42       if ( testDay > 0 && testDay <= daysPerMonth[ month ] )
43          return testDay;
44
45       // check for leap year
46       if ( month == 2 && testDay == 29 && ( year % 400 == 0 ||
47            ( year % 4 == 0 && year % 100 != 0 ) ) )
48          return testDay;
49
50       System.out.printf( "Invalid day (%d) set to 1.", testDay );
51       return 1;  // maintain object in consistent state
52    } // end method checkDay
53
54    // return a String of the form month/day/year
55    public String toString()
56    {
57       return String.format( "%d/%d/%d", month, day, year );
58    } // end method toString
59 } // end class Date
```

Check if the day is February 29 on a leap year

- Date.java

f 3)

```java
// Fig. 8.8: Employee.java
// Employee class with references to other objects.

public class Employee
{
   private String firstName;
   private String lastName;
   private Date birthDate;
   private Date hireDate;

   // constructor to initialize name, birth date and hire date
   public Employee( String first, String last, Date dateOfBirth,
      Date dateOfHire )
   {
      firstName = first;
      lastName = last;
      birthDate = dateOfBirth;
      hireDate = dateOfHire;
   } // end Employee constructor

   // convert Employee to String format
   public String toString()
   {
      return String.format( "%s, %s  Hired: %s  Birthday: %s",
         lastName, firstName, hireDate, birthDate );
   } // end method toString
} // end class Employee
```

Employee contains references to two **Date** objects

Implicit calls to **hireDate** and **birthDate**'s **toString** methods

- Employee.java

- EmployeeTest.java

```java
// Fig. 8.9: EmployeeTest.java
// Composition demonstration.

public class EmployeeTest
{
   public static void main( String args[] )
   {
      Date birth = new Date( 7, 24, 1949 );
      Date hire = new Date( 3, 12, 1988 );
      Employee employee = new Employee( "Bob", "Blue", birth, hire );

      System.out.println( employee );
   } // end main
} // end class EmployeeTest
```

Create an **Employee** object

Display the **Employee** object

```
Date object constructor for date 7/24/1949
Date object constructor for date 3/12/1988
Blue, Bob  Hired: 3/12/1988  Birthday: 7/24/1949
```

Exercise

- Create a class **Address** with following Data members:
  - home;
  - street;
  - town;
  - city;
  - state;

- Create constructors and set get methods for all data members.
- Now create a class **Person** which contain “has-a” relation with Address class. It has data members : firstName and LastName.
- Create constructors, set get methods and display function in Person Class

Exercise

- Create an Address class, which contains street#, house#, city and code.
- Create another class Person that contains an address of type Address, name and phone. Give constructors and appropriate get and set functions and a display function that prints all details of Person.
- Test class person in main.
- Create another class Book that contains an author of type Person. Other data members are bookName and publisher. Give constructors and a display function that prints all details of Book. Test class Book in main.

- Write a command to change the address of the author of this book
- Book b = new Book()
- Address a = new Address(?,?)
- B.getAuthor().setAddress(a)

Exercise

- Create a class Job with following attributes:
  - Data Members: Designation, Salary, Id
  - Create constructors and setters and getters for all
- Modify the Employee class (discussed in class) and add a Job Object as data member. Create a new method in Employee class that returns the Salary of the Employee.