

Learning Notes for Unit 4 – Part A

In the last unit, you were shown how to create custom classes for you to use in programs. Now let's assume you need to expand what that custom class is able to do. You have three options:

1. You can modify your existing class and recompile. If you take this approach, you should not modify the way methods are called or what is returned. You can add new methods and attributes and you can change the way a procedure is implemented within an existing method. If you (or others) are using this class in programs, changes to existing methods can break those existing programs.
2. You can make a new class with all the attributes and data from the original class and then make your modifications. You avoid the breakage you can encounter if you use option 1, but now you have a lot of duplicate code to maintain.
3. You can make a new class that has access to all the attributes and data in the original class and make the desired modifications **WITHOUT** affecting the original class. This avoids the problem of code duplication that arises with option 2. This process is called inheritance.

Inheritance:

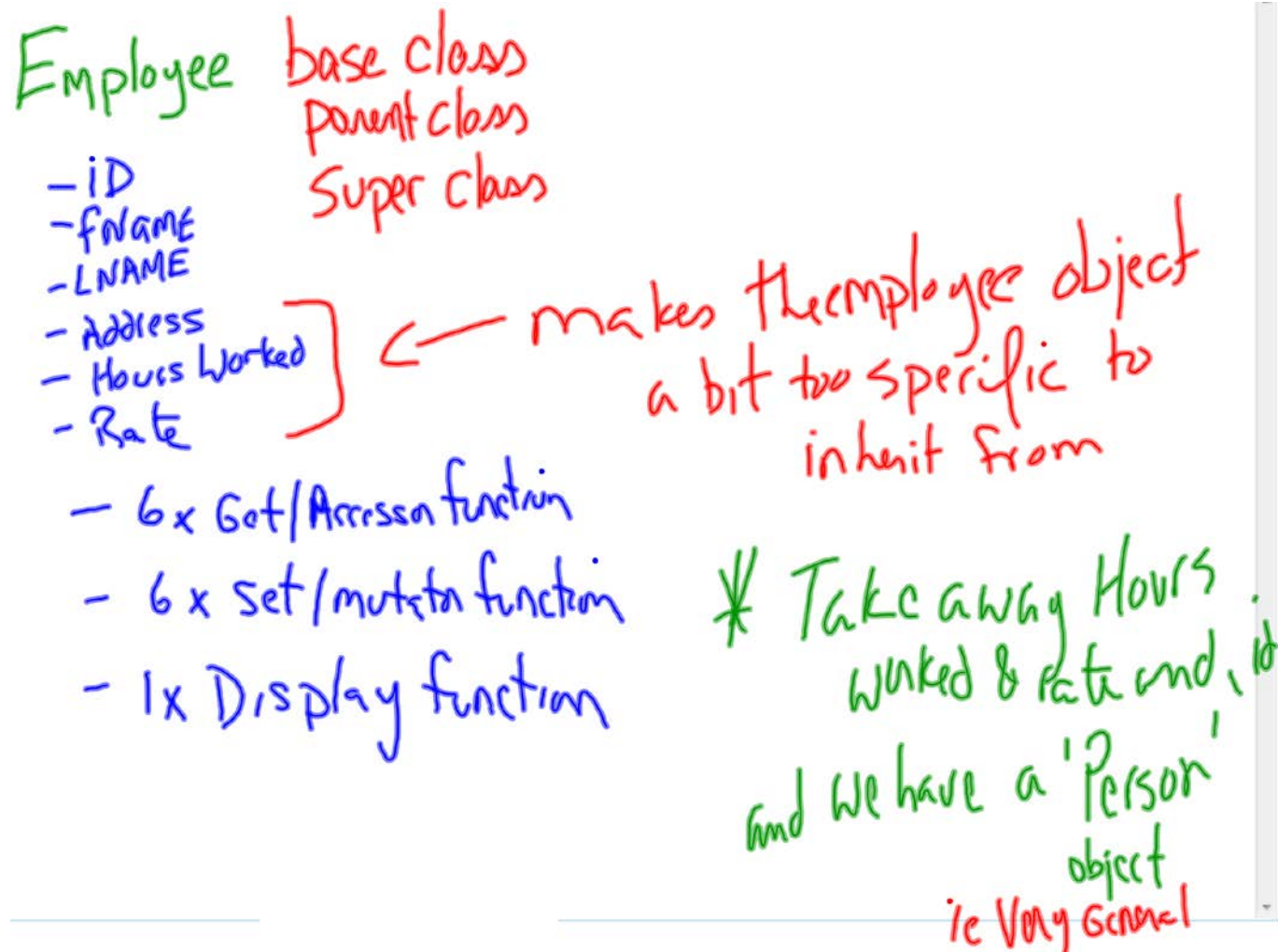
Let's consider the class example from the last unit (Employee).

Employee

- ID
- FNAME
- LNAME
- Address
- Hours Worked
- Rate
- 6 x Get/Access function
- 6 x Set/mutate function

base class
parent class
Super class

Using inheritance terminology, we will refer to this as the base class (also known as the parent class or the super class).



Unfortunately, the employee class is a bit too specific with its attributes and does not lend itself very well to be inherited from.

If we remove the attributes and functions for HoursWorked, RateOfPay, and ID, we have a much more generalized data storage object. We are essentially describing a "Person" object.

Person Class

- fname
 - lname
 - Address
 - Gender
 - 4 x Accessor fⁿs
 - 4 x mutators
 - Display
 - constructor
 - + anything else
-

Now let's build the Person object source file using the 7 steps for creating a class outlined in the previous unit.

```
public class Person {  
- }  
}
```

step 1


```
public class Person {  
    //list of attributes  
    private String FirstName;  
    private String LastName;  
    private String Address;  
    private char gender;  
}
```

step 2

```
public class Person {  
    //list of attributes  
    private String FirstName;  
    private String LastName;  
    private String Address;  
    private char gender;  
  
    public String getFirstName () { return FirstName; }  
    public void setFirstName (String temp) { FirstName = temp; }  
  
    public String getLastName () { return LastName; }  
    public void setLastName (String temp) { LastName = temp; }  
  
    public String getAddress () { return Address; }  
    public void setAddress (String temp) { Address = temp; }  
  
    public char getGender () { return gender; }  
    public void setGender (char temp) { gender = temp; }  
}
```

step 3

```
public class Person {  
  
    //list of attributes  
    private String FirstName;  
    private String LastName;  
    private String Address;  
    private char gender;  
  
    public String getFirstName () { return FirstName; }  
    public void setFirstName (String temp) { FirstName = temp; }  
  
    public String getLastName () { return LastName; }  
    public void setLastName (String temp) { LastName = temp; }  
  
    public String getAddress () { return Address; }  
    public void setAddress (String temp) { Address = temp; }  
  
    public char getGender () { return gender; }  
    public void setGender (char temp) { gender = temp; }  
  
    //Default Constructor  
    public Person () {  
        FirstName = "";  
        LastName = "";  
        Address = "";  
        gender = ' ';  
    }  
}
```



```
public String getLastName () { return LastName; }
public void setLastName (String temp) { LastName = temp; }

public String getAddress () { return Address; }
public void setAddress (String temp) { Address = temp; }

public char getGender () { return gender; }
public void setGender (char temp) { gender = temp; }

//Default Constructor
public Person () {
    FirstName = "";          LastName = "";
    Address = "";            gender = ' ';
}

//Secondary Constructor, pass first name and last name, set address/gender to blank
public Person (String temp1, String temp2) {
    FirstName = temp1;        LastName = temp2;
    Address = "";             gender = ' ';
}

//Secondary Constructor, pass all values at allocation time
public Person (String temp1, String temp2, String temp3, char temp4) {
    FirstName = temp1;        LastName = temp2;
    Address = temp3;          gender = temp4;
}
```

Step 5

```

public String getLastName () { return LastName; }
public void setLastName (String temp) { LastName = temp; }

public String getAddress () { return Address; }
public void setAddress (String temp) { Address = temp; }

public char getGender () { return gender; }
public void setGender (char temp) { gender = temp; }

//Default Constructor
public Person () {
    FirstName = "";          LastName = "";
    Address = "";            gender=' ';
}

//Secondary Constructor, pass first name and last name, set address/gender to blank
public Person (String temp1, String temp2) {
    FirstName = temp1;        LastName = temp2;
    Address = "";             gender=' ';
}

//Secondary Constructor, pass all values at allocation time
public Person (String temp1, String temp2, String temp3, char temp4) {
    FirstName = temp1;        LastName = temp2;
    Address = temp3;          gender=temp4;
}

```

any other functions - Step 6

Step 7 would be to create a program to test the Person class.

A Step 8 can be added to the process. We will define a Display function to easily present the currently stored attribute values.

```
//Secondary Constructor, pass first name and last name, set address/gender to blank
public Person (String temp1, String temp2) {
    FirstName = temp1;    LastName = temp2;
    Address = "";        gender=' ';
}

//Secondary Constructor, pass all values at allocation time
public Person (String temp1, String temp2, String temp3, char temp4) {
    FirstName = temp1;    LastName = temp2;
    Address = temp3;      gender=temp4;
}

public void Display() {
    System.out.println("FirstName: " + FirstName + ", LastName:" + LastName +
        "Address:" + Address + ", Gender: " + gender);
}
```

Display
function

step
8

In the above example, we have created a lot of code to describe a Person object. With a few more attributes and their required accessor and mutator function added, we could have an Employee object or a Student object or a Client object.

Inheritance is the process of taking a utility class (i.e. relatively generic) and adding new attributes and methods within a new more specific software class

- Allows for code reuse
- Can inherit from a single parent/superclass

Creating an inherited class (aka child class, sub class, derived class):

1. In the super class (e.g. Person class), change all the **private** access modifiers to **protected**. The protected keyword provides the same level of protection as private, but allows the inherited class to directly access those attributes in the parent class. Recompile the parent class.
2. Make sure the parent (super) class has a default constructor.
3. Follow the steps in the checklist to create the inherited class. The first line must contain **extends** and then the class that you are inheriting from.

e.g. `public class Employee extends Person`

Now add the new attributes, accessors, mutators, constructors, and any other required methods. You would also redefine the Display method to now display all attributes (the inherited attributes plus the newly defined attributes)

```
public class MarconiStudent extends Person {  
}
```

↖ with no extra code

Marconi Student has

All functionality as
Person

```
public class Person {
```

```
    //list of attributes
```

```
    private String FirstName;
```

```
    private String LastName;
```

```
    private String Address;
```

```
    private char gender;
```

```
    public String getFirstName () { return FirstName; }
```

```
    public void setFirstName (String temp) { FirstName = temp; }
```

```
    public String getLastName () { return LastName; }
```

```
    public void setLastName (String temp) { LastName = temp; }
```

} must change private to
protected if used
in inheritance

```
public class TestMarconiStudent {  
    public static void main(String[] args) {  
        MarconiStudent paul = new MarconiStudent();  
        paul.setFirstName("Paul");  
        paul.Display();  
    }  
}
```

set first name
Display
are actually in
Person

* you have access to these 2
functions even though
Marconi student has no code

n] C:\Program Files\Java\jre7\bin\javaw.exe (Feb 23, 2012 2:01:43 AM)
ider:

except extends

When you create the default constructor for your derived class, you can set initial values for all attributes (including ones from the parent class):

```
public class MarconiStudent extends Person {  
  
    //declare any new attributes  
    String studentID;  
    double average;  
  
    //Get and Set for each new attribute  
    public String getStudentID () { return studentID; }  
    public void setStudentID (String temp) { studentID = temp; }  
  
    public double getAverage () { return average; }  
    public void setAverage (double temp) { average = temp; }  
  
    //set default constructor  
    public MarconiStudent() {  
        FirstName = "";           LastName = "";  
        Address = "";             gender = ' '  
        studentID = "";           average = 0;  
    }  
}
```

} one option for default constructor however first 4 attributes are already set in super constructor (we have duplication)

As shown above, this is effective, but there is duplication since the same code for four of the attributes exist in the parent class. As an alternative, you can call the constructor from the parent class to set the four attributes there and include the code to set the new attributes.

```
//set default constructor  
public MarconiStudent() {  
    super();  
    studentID="";  
    average=0;  
}
```

*this calls the
default constructor
in parent class*

The same thing happens if you have a constructor to set all attributes.

```
public class MarconiStudent extends Person {  
    //declare any new attributes  
    String studentID;  
    double average;  
  
    //Get and Set for each new attribute  
    public String getStudentID () { return studentID; }  
    public void setStudentID (String temp) { studentID = temp; }  
  
    public double getAverage () { return average; }  
    public void setAverage (double temp) { average = temp; }  
  
    //set default constructor  
    public MarconiStudent() {  
        super();  
        studentID="";          average=0;  
    }  
  
    //set secondary constructor where we set all  
    public MarconiStudent (String fn1, String ln2, String add3, char gen4, String sid5, double avg6)  
    {  
        FirstName = fn1;      LastName = ln2;  
        Address = add3;        gender=gen4;  
        studentID=sid5;        average=avg6;  
    }  
}
```

duplicate

```
//set secondary constructor where we set all  
public MarconiStudent (String fn1, String ln2, String add3, char gen4, String sid5, double avg6)  
{  
    super(fn1,ln2,add3,gen4);  
    studentID=sid5;          average=avg6;  
}
```

call to constructor in Person
with 4 parameters

Overriding Methods:

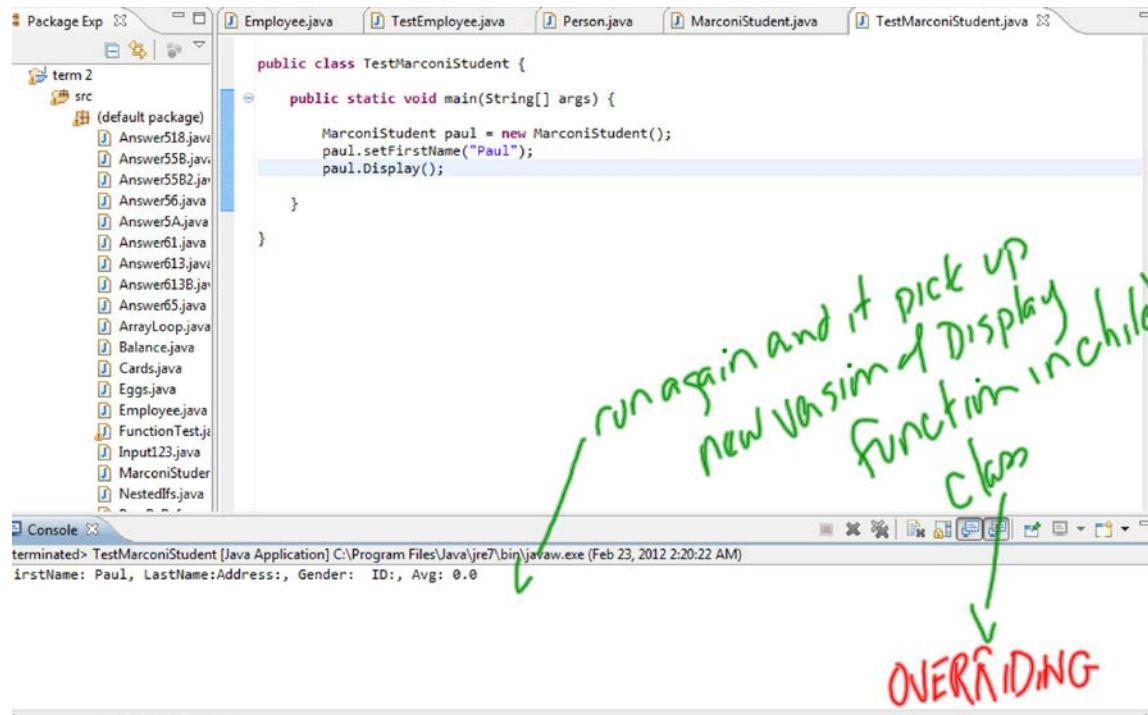
You may have to redefine methods that will behave differently in the derived class. For example, the Display() method in the parent class will display the four original attributes only. This is of limited use to us in the inherited class as we would expect the Display() method to display all six attributes that the class can access.

You would define a new Display() method for the derived class:

```
public void Display() {  
    System.out.println("FirstName: " + FirstName + ", LastName: " + LastName +  
        "Address: " + Address + ", Gender: " + gender +  
        "ID: " + studentID + ", Avg: " + average);  
}
```

} new display
function

The new class actually has access to two Display() methods. The newly defined one will override the method in the parent class.



Learning Notes for Unit 4 – Part B

Packaging your classes:

Once you have created a number of classes, you may wish to bundle them together to simplify how you reuse them in future projects. Please review the instructions on how to place classes into a package (pages 352-357) of your book.

Once you have successfully packaged your classes, they can be imported into a future project just like we do when we need to use the `JOptionPane` class to interact with users.