Handout 10

OOP Basics: Classes and Objects

In Object Oriented approach – application is viewed as a collection of interacting domain objects.

Example:
- course registration software involves students, courses, course sections
- library catalog involves borrowers, books, other borrowed items
- payroll application involves employees, departments, etc

Each domain object in an application

- Is described with a set of data values (a.k.a attributes) that characterize it
e.g. each employee has a name, an address, a SSN, salary, etc...
- Performs certain functions
e.g. employee object can print employee’s name and address, report employee’s SSN, calculate amount earned, update hours worked, etc

Domain Objects are classified according to their commonalities: objects of the same class share the same data and same behavior (functions).

A Class is a template for Objects. A Class defines the common (data) attributes and behaviors (functions) of all Objects that belong to that Class.

Each Object is an instance of its Class.

Example: Application that allows to
- enter and update number of hours an employee worked,
- compute the pay due to an employee based on employee’s hourly pay rate and amount of hours worked
- print a check.

Class Employee:
- Data attributes: name, hourly rate, number of unpaid hours
- Functions: create/initialize employee data, update the hours worked, compute the amount due, reset the number of unpaid hours, print employee data

JAVA uses terms
- Instance variables – for attributes
- Instance methods – for function members
In JAVA – a program is a collection of Class definitions.

Data attributes that characterize each individual object are called \textit{instance variables}.

Functions that objects perform are called \textit{instance methods}, i.e.

\textbf{CLASS Employee}

\begin{verbatim}
String name
double payRate
double unpaidHours

IncrementHoursBy(double hrs)
AmtDue()
PrintEmployeeInfo()
\end{verbatim}

Each Object is an \textbf{instance} of its Class.

Has \textbf{specific values} for all attributes

The instance methods would return the values \textbf{based on this object's data}, e.g.

employee2.AmtDue() would return 24
employee3.AmtDue() would return 74.50
JAVA class definition: Employee.java

public class Employee
{
    // Instance variables of class Employee
    private String name;       // name of employee
    private double payRate;    // hourly pay rate
    private double unpaidHours; // number of hours for which the pay is due

    // Instance methods
    // Accessor methods for each of the fields: return value of each instance variable
    public String getName()   { return name;   // returns the value of instance variable name }
    public double getPayRate() { return payRate; // returns the value of instance variable payRate }
    public double getUnpaidHours() { return unpaidHours; // returns the value of instance variable unpaidHours }

    // mutator methods - set values of instance variables
    public void setName(String aname)   { name = aname; }
    public void setPayRate(double rate) { if (rate <= 0 ) System.out.println ("ERROR: zero or negative pay rate"); else payRate = rate; }

    // resets the value of instance variable unpaidHours to 0
    public void resetUnpaidHours() { unpaidHours = 0; }

    // Other instance methods
    //
    // increments the value of instance variable unpaidHours by hrs
    // returns the updated value of unpaidHours
    public double IncrementHoursBy(double hrs) { unpaidHours = unpaidHours + hrs; return unpaidHours; }
// computes and returns the amount due to employee

public double AmtDue()
{
    double amount;
    amount = unpaidHours * payRate;
    return amount;
}

// prints data for this employee

public void PrintEmployeeInfo()
{
    double amt = AmtDue(); // same as double amt = this.AmtDue()
    System.out.print (name + " has worked " + unpaidHours);
    System.out.println (" hours");
    System.out.print ("At the pay rate of " + payRate);
    System.out.println (" the amount due is " + amt);
}

Notes on Class definition Employee.java

Good Object Oriented style hides the details of class implementation such as
  Instance variables
  Auxiliary (helper) instance methods
by disabling access to these items from outside of class definition.
This information hiding improves robustness of software and is accomplished by declaring
instance variables and auxiliary instance methods as private.

To provide information about the Object’s private data to other Objects – accessor methods are
supplied. Accessor (get) method for each attribute returns the value of the attribute (instance
variable).

To enable changing the values of an Object’s private attributes – mutator (set) methods are
provided. Mutator methods can also perform a check on the validity of the new value, as done in
method setPayRate(double rate).

All instance methods within the class definition can access any (private or public) instance
variables and methods.

JAVA conventions
  • Each Java class definition should be a separate file
  • Use the same name for the class and the file, except add ".java" to the file name
  • Accepted JAVA programming convention: start the class (and file) name a capital
    letter.
  • For now put all the classes you define for an application in the same directory and the
    same JBuilder project
Using the class Employee.java in a program EmployeeObjectsDemo.java (must be compiled together with Employee.java)

```java
public class EmployeeObjectsDemo {
    public static void main (String args[]) {
        Employee employee1 = new Employee(); // new object is created
        employee1.setName("Jane Smith");
        employee1.setPayRate(9.50);
        employee1.resetUnpaidHours();
        System.out.println("How long did " + employee1.getName() + " work today? ");
        double hrs = SavitchIn.readLineInt();
        employee1.incrementHoursBy(hrs);

        Employee employee2 = new Employee(); // new object is created
        employee2.setName("George Clark");
        employee2.setPayRate(12);
        employee2.resetUnpaidHours();
        System.out.println("How long did " + employee2.getName() + " work today? ");
        hrs = SavitchIn.readLineInt();
        employee2.incrementHoursBy(hrs);

        Employee empEarnedMore;
        if (employee1.AmtDue() == employee2.AmtDue())
            System.out.println("Both employees earned same amount");
        else // find out who earned more today.
            if (employee1.AmtDue() < employee2.AmtDue())
                empEarnedMore = employee2;
            else
                empEarnedMore = employee1;
            System.out.println(empEarnedMore.getName + "earned more.");
        employee1.PrintEmployeeInfo();
        employee2.PrintEmployeeInfo();
    }
}
```
Difference between the instance methods and static methods:

**Instance methods** implement **object specific** behaviors, in other words, they return value or perform an action that is based on the value of data attributes, i.e. instance variables of a **concrete object**.

For example consider `charAt()` instance method of `String` class. Given

```java
String one = "Hello";
String two = "World";
```

`one.charAt(0)` would return character ‘H’ – the first char in object **one**

`two.charAt(0)` would return character ‘W’ — the first char in object **two**

**Static methods**, on the other hand, do not require an object of a class – they work **independently** of any object of their class, thus they cannot use any instance variables, because instance variables store instance-(in other words Object-) specific values.