Introduction to Algorithms and Data Structures

Lecture 5 – Inheritance

Inheritance and Derived Classes

- ***Inheritance*** is the process by which a new class is created from another class.
- The new class is called the ***derived class***.
- The class from which it is created is called the ***base class***.
- A derived class inherits all the instance variables and all the methods belonging that the base class has and can also have its own methods and variables in addition.

Derived Classes

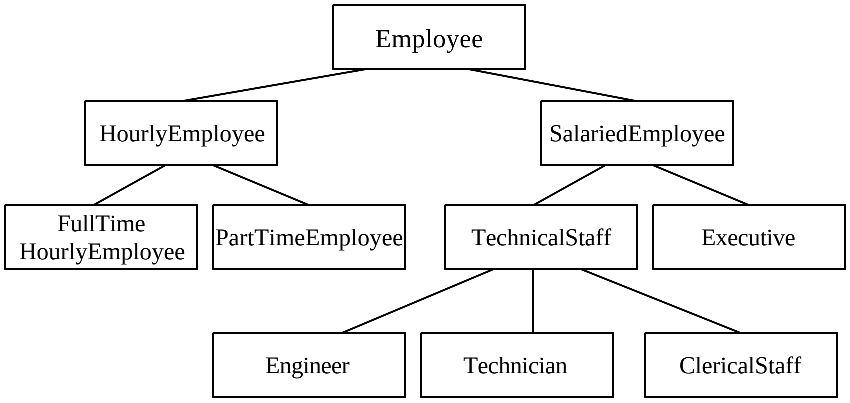
- Derived class are expected to have a great deal in common with the base class from which they are derived.
- We expect them to have all the properties and methods of the base class as well as a few additional ones of their own.
- They may even have their own way of executing methods that also belong to the base class; this is called ***<u>overriding the method</u>***.

Syntax For a Derived Class

- The general syntax for a derived class (also known as a ***subclass***) is:

```
public class DerivedClassName
                 extends BaseClassName {
    Declarations of Added Static Variables
    Declarations of Added Instance Variables
    Definitions of added and Overridden Methods
}
```

Class Hierarchy



Employee.java

```
import java.util.Scanner;

public class Employee {
  private String name;
  private Date hireDate; // The previously
   defined Date class

  // Employee() - A default constructor
  public Employee() {
    name = "no name";
    // A placeholder
    hireDate = new Date("January", 1, 1000);
  }
```

```
// Employee() - A conversion constructor
// Precondition: Neither theName nor theDate
//               is null
public Employee(String theName, Date theDate)
{
  if (theName == null || theDate == null) {
    System.out.println
         ("Fatal error creating employee");
    System.exit(0);
  }
  name = theName;
  hireDate = new Date(theDate);
}

// Employee() - A copy constructor
public Employee(Employee originalObject) {
  name = originalObject.name;
  hireDate = new Date(originalObject.hireDate);
}
```

```
// The accessors
public String getName() {
  return name;
}

public Date getHireDate() {
  return new Date(hireDate);
}

// setName() - A mutator for name
// Precondition newName is not null
public void setName(String newName) {
  if (newName == null) {
    System.out.println
        ("Fatal error setting employee name.");
    System.exit(0);
  }
  else
    name = newName;
}
```

```
// setName() - A mutator for hireDate
// Precondition newHireDate is not null
public void setHireDate(Date newHireDate) {
  if (newHireDate == null) {
    System.out.println("Fatal error setting
employee name.");
    System.exit(0);
  }
  else
    hireDate = newHireDate;
}
```

```
  public String toString() {
    return (name + " " + hireDate.toString());
  }

  public boolean equals(Employee otherEmployee) {
    return (name.equals(otherEmployee.name) &&
  hireDate.equals(otherEmployee.hireDate));
  }
}
```

Inherited Members

- A derived class inherits all the properties and methods belonging to the base class.
- The only exceptions are methods defined in the base class that are overridden in the derived class.

HourlyEmployee.java

```
// Class invariant: All employees have a name,
//                  hire date, non-negative wage
//                  rate and a non-negative
//                  number of ours worked.
//                  A name string of "No name“
//                  indicates no real name
//                  specicfied yet.
public class HourlyEmployee extends Employee {
  private double wageRate;
  private double hours;  // for the month
```

```
// HourlyEmployee() - A default constructor
public HourlyEmployee() {
  // Call the base class's constructor
  super();
  wageRate = 0;
  hours = 0;
}

// HourlyEmployee() - A conversion constructor
// Precondition: Neither theName nor theDate is
//               null; theWageRate and theHours
//               are non-negative.
public HourlyEmployee(String theName,
           Date theDate, double theWageRate,
                             double theHours) {
  // Use the base class’s conversion
  // constructor
  super(theName, theDate);
```

```
  if ((theWageRate >= 0) && (theHours >= 0)) {
    wageRate = theWageRate;
    hours = theHours;
  }
  else {
    System.out.println
      ("Fatal error: creating an " +
       " illegal hourly employee.");
    System.exit(0);
  }
}

// HourlyEmployee() - Copy constructor
public HourlyEmployee(HourlyEmployee
originalObject)  {
  super(originalObject); // the base class's
copy constructor
  wageRate = originalObject.wageRate;
  hours = originalObject.hours;
}
```

```
// The accessor of the derived class
public double getRate() {
  return wageRate;
}

public double getHours() {
  return hours;
}

// getPay() - this accessor will be different
 in the
//           other derived classes
public double getPay() {
  return wageRate * hours;
}
```

```
// setHours() - A mutator for hours
// Precondition: hoursWorked is non-negative
public void setHours(double hoursWorked)  {
  if (hoursWorked >= 0 )
    hours = hoursWorked;
  else {
    System.out.println
      ("Fatal error: Negative hours worked");
    System.exit(0);
  }
}
```

```
// setRate() - A mutator for wageRate
// Precondition: newWageRate is non-negative
public void setRate(double newWageRate)  {
  if (newWageRate >= 0 )
    wageRate = newWageRate;
  else {
    System.out.println
          ("Fatal error: Negative wage rate");
    System.exit(0);
  }
}
```

```
  // toString() - overwrites the base class's
  //     toString method
  public String toString() {
    return (getName() + " "
         + getHireDate().toString()
         + "\n" + wageRate + " per hour for "
         + hours + " hours.");
  }

  //equals() - Overrides the base class's equals
  //           method
  public boolean equals(HourlyEmployee other)  {
    return(getName().equals(other.getName())
         && getHireDate().equals(
                       other.getHireDate())
             && wageRate == other.wageRate
             && hours == other.hours);
  }
}
```

InheritanceDemo.java

```
public class InheritanceDemo  {
  public static void main(String[] args)  {
    HourlyEmployee joe
      = new HourlyEmployee("Joe Worker",
                 new Date("January", 1, 2004),
                   50.50, 160);
    System.out.println("joe's longer name is "
                                + joe.getName());
    System.out.println
           ("Changing joe's name to Josephine.");
    joe.setName("Josephine");
    System.out.println
              ("joe's record is as follows:");
    System.out.println(joe);
  }
}
```

One Base Class – Many Derived Classes

- We can write more than one subclass based on the same base class. The most important thing is to make sure that it is reasonable for this class is have the same properties and methods as the base class (plus the additional ones).

SalariedEmployee.java

```
// Class invariant: All employees have a name,
//                  hire date and non-negative
//                  salary.
//                  A name string of "No name“
//                  indicates no real name
//                  specified yet.
//                  A hire date of January 1,
//                  1000 indicated no real hire
//                  date specified yet.
public class SalariedEmployee extends Employee {
  private double salary; // annual
```

```
  // SalariedEmployee() - A default constructor
  public SalariedEmployee() {
    // Call the base class's constructor
    super();
    salary = 0;
  }
```

```
// SalariedEmployee() - A conversion
//                      constructor
// Precondition: Neither theName nor theDate is
//               null; theSalary is  non-
//               negative.
public SalariedEmployee(String theName,
           Date theDate,   double theSalary) {
  // using the base class's conversion
  // constructor
  super(theName, theDate);
  if (theSalary >= 0) {
    salary = theSalary;
  }
  else {
    System.out.println("Fatal error: Negative
salary.");
    System.exit(0);
  }
}
```

```
// SalariedEmployee() - Copy constructor
public SalariedEmployee
      (SalariedEmployee originalObject)  {
  super(originalObject);
  salary = originalObject.salary;
}

// getSalary() - An accessor
public double getSalary() {
  return salary;
}

// getPay() - this accessor will be different
//            in the other derived classes and
//            is also from different from
//            HourlyEmployee
public double getPay() {
  return salary / 12;
}
```

```
// setSalary() - A mutator for salary
// Precondition: salary is non-negative
public void setSalary(double newSalary)  {
  if (salary >= 0 )
    salary = newSalary;
  else {
    System.out.println("Fatal error: Negative
hours worked");
    System.exit(0);
  }
}
```

```
  // toString() - overwrites the base class's
  //              toString method
  public String toString() {
    return (getName() + " "
              + getHireDate().toString()
              + "\n" + salary + " per year");
  }

  //equals() - Overrides the base class's equals
  //           method
  public boolean equals(SalariedEmployee other)  {
    return(getName().equals(other.getName())
             &&
  getHireDate().equals(other.getHireDate())
             && salary == other.salary);
  }
}
```

IsADemo.java

```
public class IsADemo  {
  public static void main(String[] args) {
    SalariedEmployee joe
              = new SalariedEmployee("Josephine",
           new Date("January", 1, 2004), 100000);
    HourlyEmployee sam
              = new HourlyEmployee("Samantha",
        new Date("February", 1, 2003), 5.50, 40);

    System.out.println("joe\'s longer name is "
                       + joe.getName());
    System.out.println
                 ("showEmployee(joe) invoked:");
    showEmployee(joe);
    System.out.println
              ("showEmployee(sam) invoked:");
    showEmployee(sam);
  }
```

```
  public static void showEmployee
                   (Employee employeeObject) {
    System.out.println(employeeObject.getName());
    System.out.println
             (employeeObject.getHireDate());
  }
}
```

Limited Access

- We already know that **`private`** variables and methods can only be used by objects of the same class and that **`public`** variables and methods can be used by anyone.
- We need some way of limited access to objects of the same class AND objects of a subclass. That is what **`protected`** data is for.

`protected` Data

- **`protected`** data and methods can be accessed by objects belonging to the class or to the derived class.
- This allows for some protection of the data, but less than if it were **`private`**.

Access Modifers

```
public class A {
  public int v1;
  protected int v2;
  private int v3;
}
```

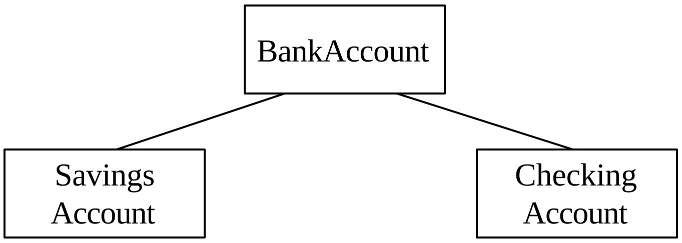
```
public class B {
  can access v1;
  can’t access v2;
  can’t access v3;
}
```

```
public class c extends  A {
  can access v1;
  can access v2;
  can’t access v3;
}
```

`Object` Class

- Every class in Java is considered a subclass of the class **`Object`**.
- Every class inherits its properties and methods, which should include toString and equals

Bank Accounts



BankAccount.java

```
// Class invariant: name and acctNum cannot be
//                  null
public class BankAccount  {
  public String name;  // Account holder's name
                       // Anyone can access it
  protected String acctNum;  // Account number
                             // only subclasses
                             // and this class
                             // can access it
  private double balance;  // The account balance
                           // only this class can
                          //  access it.
  // BankAccount() - Default constructor
  public BankAccount() {
    name = new String("no name");
    acctNum = new String("no number");
    balance = 0.0;
  }


  // BankAccount() - A conversion constructor
  public BankAccount(String newName,
          String newAcctNum, double newBalance) {
    name = new String(newName);
    acctNum = new String(newAcctNum);
    balance = newBalance;
  }
```

```
// BankAccount() - A copy constructor
public BankAccount(BankAccount originalObject){
  name = new String(originalObject.name);
  acctNum = new String(originalObject.acctNum);
  balance = originalObject.balance;
}

// deposit() - deposits an amount in the
//             account and prints information
//             about the transaction
public void deposit(double depositAmt) {
  balance += depositAmt;
  System.out.print(name + ", account number "
                 + acctNum + ",\n deposited ");
  System.out.printf
      ("$%4.2f, making the balance $%4.2f\n",
                      depositAmt, balance);
}
```

```
// withdraw()- withdraws an amount from the
//            account and prints information
//            about the transaction
public void withdraw(double withAmt) {
  if (balance >= withAmt) {
    balance -= withAmt;
    System.out.print(name + ", account number “
                  + acctNum + ",\n withdrew ");
    System.out.printf
        ("$%4.2f, making the balance $%4.2f\n",
                      withAmt, balance);
  }
  else
    System.out.println("Sorry.. you don't have"
           + " that much in your account.");
}
```

```
// getAcctNum() - an accessor for account
//                number
public String getAcctNum() {
  return new String(acctNum);
}


// getBalance() - an accessor for balance
public double getBalance() {
  return balance;
}


// setBalance() - a mutator for balance
protected void setBalance(double newBalance) {
  balance = newBalance;
}
```

```
  // setAcctNum() - a mutator for account number
  protected void setAcctNum(String newAcctNum) {
    acctNum = newAcctNum;
  }

  // toString() - returns a string with main
  //              information
  public String toString() {
    return(name + " Account #" + acctNum
             + " Balance of $"
             + String.format("%4.2f", balance));
  }
}
```

CheckingAccount.java

```
public class CheckingAccount
                      extends BankAccount {
  // How many checks have been cashed this month
  private int checkCount;

 // The per check charge
  private final double checkFee = 0.20;

  // The total number of free checks per month
  private final int freeChecks = 10;

  // CheckingAccount() - A default constructor
  public CheckingAccount() {
    super();
    checkCount = 0;
  }
```

```
// CheckingAccount() - A conversion constructor
public CheckingAccount(String newName,
          String newAcctNum, double newBalance,
                          int newCheckCount)  {
  super(newName, newAcctNum, newBalance);
  checkCount = newCheckCount;
}


// CheckingAccount() - A copy constructor
public CheckingAccount
      (CheckingAccount originalObject) {
  // use the base class's conversion
  // constructor
  super(originalObject.name,
        originalObject.acctNum,
        originalObject.getBalance());
}
```

```
// withdraw() - withdraws an amount from the
//              account and prints information
//              about the transaction
public void withdraw(double withAmt) {
  // If the money's there, withdraw it
  if (super.getBalance() >= withAmt) {
    super.setBalance
              (super.getBalance() - withAmt);
    System.out.print(name + ", account number "
+ acctNum + ",\n withdrew ");
    System.out.printf
       ("$%4.2f, making the balance $%4.2f\n",
                  withAmt, super.getBalance());
  }
  else
    // Otherwise just print a polite message
    System.out.println("Sorry.. you don't have"
           + " that much in your account.");
    // That's one more check - is it still free?
    checkCount++;
    if (checkCount > freeChecks) {
      super.setBalance
           (super.getBalance() - checkFee);
      System.out.printf
        ("There was a check fee of $%4.2f "
     + "deducted,\n making the balance $%4.2f\n",
                  checkFee, super.getBalance());
    }
    else
      System.out.println("There are still "
        + (freeChecks - checkCount)
                      + " free checks left.");
  }
}
```

TestBankAccount.java

```
public class TestBankAccount {
  public static void main(String[] args) {
    BankAccount ba = new BankAccount
       ("John F. X. Smith", "jfxs2121", 400.0);
    CheckingAccount ca = new CheckingAccount
       ("Robert M Siegfried", "rms13", 50.0, 1);

    ba.deposit(300);
    ba.withdraw(560);
    ca.deposit(500);
    ca.withdraw(200);

    System.out.println
    ("The account holder\'s name is " + ba.name);
    System.out.println(ba);
    System.out.println(ca);
  }
}
```