

**Pseudocode**

//Program Header

//Program Name: Class Development

//Programmer: Your Name

//CIS247C, Week 6 Lab

//Program Description: PROVIDE A DESCRIPTITON OF THE PROGRAM

**Define class – Benefit**

private access specifier

// declare data members

healthInsurance as a string

lifeInsurance as a double

vacation as an integer

public access specifier

// declare methods

default constructor Benefit()

constructor with parameters Benefit(string, double, int)

declaration displayBenefits()

declaration getHealthInsurance() returns a string

declaration setHealthInsurance(string)

declaration getLifeInsurance() returns a double

declaration setLifeInsurance(double)

declaration getVacation() returns an integer

declaration setVacation(int)

**end define class**

**start define constructors/methods**

default constructor Benefit()

set the attributes as follows: healthInsurance = "not provided", lifeInsurance = 0.0, vacation = 14.

end constructor

constructor with parameters Benefit(string health, double life, int vacation)

should initialize all of the attributes using values passed in using its parameter list.

end constructor

procedure displayBenefits()

should display all the attributes of the Benefit object in a well-formatted string with logical labels applied to each attribute.

end procedure

procedure getHeathInsurance() returns a string

return value of healthInsurance data member.

end procedure

procedure setHealthInsurance(string hins)

assign value to healthInsurance data member using value passed in parameter list.

end procedure

procedure getLifeInsurance() returns a double

return value of lifeInsurance data member.

end procedure

procedure setLifeInsurance(double lifeIns)

assign value to lifeInsurance data member using value passed in parameter list.

end procedure

procedure getVacation() returns an integer

return value of vacation data member.

end procedure

procedure setVacation(int vaca)

assign value to vacation data member using value passed in parameter list.

end procedure

**end define constructors/methods**

**Define class – *Employee (Abstract Class)***

private access specifier

// declare data members

numEmployees as a static integer value of 0

protected access specifier

firstName as a string

lastName as a string

gender as a character

dependents as an integer

annualSalary as a double

benefit as a Benefit class composite object

public access specifier

default constructor Employee()

constructor with parameters Employee(string, string, char, int, double, string,

double, int)

constructor with parameters Employee(string, string, char, int, double, Benefits)

*declaration calculatePay() as a pure virtual function*

*declaration displayEmployee() as a pure virtual function*

declaration getFirstName()

declaration setFirstName(string)

declaration getLastName()

declaration setLastName(string)

declaration getGender()

declaration setGender(char)

declaration getDependents()

declaration setDependents(int)

declaration getAnnualSalary()

declaration setAnnualSalary(double)

declaration static getNumEmployees()

declaration setDependents(string)

declaration setAnnualSalary(string)

declaration setBenefits(Benefits)

declaration getBenefits() returns a Benefits object

**end define class**

**start define constructors/methods**

default constructor Employee() : benefits()

set the attributes as follows: firstName = "not given", lastName = "not given", gender = "U" (for unknown), dependents = 0, and annualSalary = 20,000.

Increment numEmployees by 1 in each of the constructors.

end constructor

constructor with parameters Employee(string first, string last, char gen, int dep,

double salary, string healthIns, double lifeIns,

int vacation) : benefits(healthIns, lifeIns, vacation)

should initialize all of the attributes using values passed in using its parameter list and pass on to the benefits object its attribute values.

Increment numEmployees by 1 in each of the constructors.

end constructor

constructor with parameters Employee(string first, string last, char gen, int dep,

double salary, Benefits ben)

should initialize all of the attributes using values passed in using its parameter list and pass on to the benefits object its attribute values.

Increment numEmployees by 1 in each of the constructors.

end constructor

~~procedure calculatePay()~~

~~should return the value of annual salary divided by 52 (return annualSalary / 52;).~~

~~end procedure~~

~~procedure displayEmployee()~~

~~should display all the attributes of the Employee object in a well-formatted string with logical labels applied to each attribute. Don't forget to call calculatePay from within the displayEmployee method in order to display the Employee's weekly pay as well!~~

~~should call the benefit object’s displayBenefit method.~~

~~end procedure~~

procedure getFirstName()

return value of firstName data member.

end procedure

procedure setFirstName(string first)

assign value to firstName data member using value passed in parameter list.

end procedure

procedure getLastName()

return value of lastName data member.

end procedure

procedure setLastName(string last)

assign value to lastName data member using value passed in parameter list.

end procedure

procedure getGender()

return value of gender data member.

end procedure

procedure setGender(char gen)

assign value to gender data member using value passed in parameter list.

end procedure

procedure getDependents()

return value of dependents data member.

end procedure

procedure setDependents(int dep)

assign value to dependents data member using value passed in parameter list.

end procedure

procedure setDependents(string dep)

Create an overloaded setDependents method and this time make the parameter a string.

end procedure

procedure getAnnualSalary()

return value of annualSalary data member.

end procedure

procedure setAnnualSalary(double salary)

assign value to annualSalary data member using value passed in parameter list.

end procedure

procedure setAnnualSalary(string salary)

Create an overloaded setAnnualSalary method and this time make the parameter a string.

end procedure

procedure getNumEmployees()

Make the getNumEmployees a static method. (This way, you can call it with the class name instead of an object name.)

end procedure

procedure getBenefits() returns a Benefits object

return Benefits composite object

end procedure

procedure setBenefits(Benefits ben)

assign parameter object to Benefits composite object.

end procedure

**end define constructors/methods**

**Define class – Salaried and inherit Employee as public**

private access specifier

// declare data members

MIN\_MANAGEMENT\_LEVEL as a **constant** integer value of 0

MAX\_MANAGEMENT\_LEVEL as a **constant** integer value of 3

BONUS\_PERCENT as a **constant** double value of 10

managementLevel as an integer

public access specifier

default constructor Salaried()

constructor with parameters Salaried(string, string, char, int, double, Benefit, int)

constructor with parameters Salaried(double, int)

declaration calculatePay() returns a double

declaration displayEmployee() returns no value

declaration getManagementLevel() returns an integer

declaration setManagementLevel(int)

**end define class**

**start define constructors/methods**

default constructor Salaried()

set the attributes as follows: managementLevel = 0.

end constructor

constructor with parameters Salaried(string fname, string lname, char gen, int dep,

double sal, Benefits ben, int manLevel) : Employee(fname, lname, gen, dep, sal, ben)

For each of the constructors listed in the Salaried class, ensure to invoke the appropriate base class constructor and pass the correct arguments to the base class constructor. This will initialize the protected attributes and update the numEmployees counter.

end constructor

constructor with parameters Salaried(double sal, int manLevel)

should initialize all of the attributes using values passed in using its parameter list.

end constructor

procedure calculatePay()

Override the calculatePay method to add a 10 percent bonus for each of the management levels (i.e., bonus percentage = managementLevel \* .10).

end procedure

procedure displayEmployee()

Override the displayEmployee() method to add the management level to the employee information.

end procedure

procedure getManagementLevel()

return value of managementLevel data member.

end procedure

procedure setManagementLevel(int manLevel)

assign value to managementLevel data member using value passed in parameter list.

The valid management levels are 0, 1, 2, and 3.

end procedure

**end define constructors/methods**

**Define class – Hourly and inherit Employee as public**

private access specifier

// declare data members

MIN\_WAGE as a **constant** double value of 10

MAX\_WAGE as a **constant** double value of 75

MIN\_HOURS as a **constant** double value of 0

MAX\_HOURS as a **constant** double value of 50

wage as a double

hours as a double

category as a string

public access specifier

default constructor Hourly()

constructor with parameters Hourly(double, double, string)

constructor with parameters Hourly(string, string, char, int, double, double, Benefit, string)

declaration calculatePay() returns a double

declaration setAnnualSalary() returns no value

declaration displayEmployee() returns no value

declaration getWage() returns a double

declaration setWage(double)

declaration getHours() returns a double

declaration setHours(double)

declaration getCategory() returns a string

declaration setCategory(string)

**end define class**

**start define constructors/methods**

default constructor Hourly()

set the attributes as follows: wage = 0.0, hours = 0.0, category = “X”

end constructor

constructor with parameters Hourly(double wage, double hours, string category)

should initialize all of the attributes using values passed in using its parameter list.

end constructor

constructor with parameters Hourly(string fname, string lname, char gen, int dep,

double wage, double hours, Benefits ben,

string manLevel) : Employee(fname, lname, gen, dep, ben)

For each of the constructors listed in the Hourly class, ensure to invoke the appropriate base class constructor and pass the correct arguments to the base class constructor. This will initialize the protected attributes and update the numEmployees counter.

end constructor

procedure calculatePay()

Override the calculatePay method by multiplying the wages by the number of hours.

end procedure

procedure setAnnualSalary()

Overload the Employee setAnnualSalary method and set the annual salary by multiplying the weekly pay by 50. Remember to call calculatePay() to get the weekly pay.

end procedure

procedure displayEmployee()

Override the displayEmployee() method to add the category to the hourly employee information.

end procedure

procedure getWage()

return value of wage data member.

end procedure

procedure setWage(int wage)

assign value to wage data member using value passed in parameter list.

end procedure

procedure getHours()

return value of wage data member.

end procedure

procedure setHours(int hours)

assign value to hours data member using value passed in parameter list.

The provided hours must be more than 0 hours and less than 50 hours, and the limits should be implemented as constants.

end procedure

procedure getCategory()

return value of category data member.

end procedure

procedure setCategory(int category)

assign value to category data member using value passed in parameter list.

The valid category types are "temporary", "part time", and "full time".

end procedure

**end define constructors/methods**

**Start main**

1. Create two employee pointers with:

Employee \*employeeList1 = new Salaried(10000, 3);

Employee \*employeeList2 = new Hourly(50, 40, “full time”);

The first employee pointer refers to a salaried employee and the second employee pointer refers to an hourly employee.

1. Prompt the user to enter information for these two pointers and display the calculated result. Consider using your getInput method from Week 1 to obtain data from the user for this step.

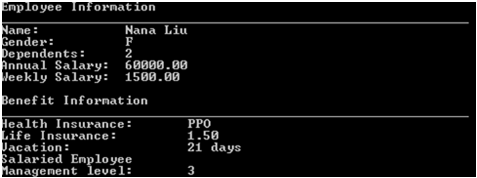
Note: when accessing members, there are two ways to use pointers:

(\*employeeList1).setFirstName(GetInput("First Name "));

employeeList1->setFirstName(GetInput("First Name "));

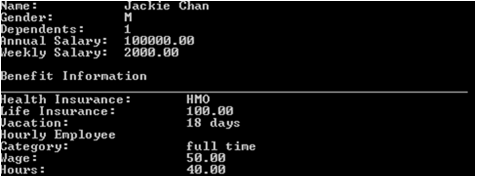
1. For the Benefits information, try creating a local Benefits object, populate it with the user information calling the setter functions, and pass the object to a setBenefits() method.
2. For salaried employee, the following information needs to be displayed:

**Partial Sample Output:**



1. For hourly employee, the following information needs to be displayed:

**Partial Sample Output:**



**end main program**

**procedure DisplayApplicationInformation**

display “Welcome the Basic User Interface Program”

display “CIS247, Week 1 Lab”

display “Name: YOUR NAME”

display “This program accepts user input as a string, then makes the appropriate data conversion”

**end procedure**

**procedure DisplayDivider(string outputTitle)**

display “\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* “ + outputTitle + “\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*”

**end procedure**

**function GetInput(string inputType) as string**

strInput as string

display “Enter the “ + inputType

get strInput

return strInput

**end function**

**procedure TerminateApplication**

display “Thank you for using the Basic User Interface program”

exit application

**end procedure**

Below is a complete sample program output for your reference.

