# Revisiting the Employee Class Program

*by Sophia*

WHAT'S COVERED

In this lesson, we will be extending the Employee class to create subclasses. Specifically, this lesson covers:

1. Creating the Base Class Person
2. Creating the Subclasses Employee and Contractor

## 1. Creating the Base Class Person

In a prior lesson, we had created a basic `Employee` class that looks like the following.

EXAMPLE

```
import datetime
class Employee:
  def __init__(self, fname, lname, empid, title, sal):
    self.firstname = fname
    self.lastname = lname
    self.employeeid = empid
    self.jobtitle = title
    self.salary = sal

    self.hiredate = datetime.date.today()

  #returns first name
  def get_firstname(self):
    return self.firstname

  #sets firstname if fname isn't an empty string
  def set_firstname(self,fname):
```

```
  if len(fname) > 0:
    self.firstname = fname

#returns last name
def get_lastname(self):
  return self.lastname

#sets lastname if lname isn't an empty string
def set_lastname(self,lname):
  if len(lname) > 0:
    self.lastname = lname

#returns job title
def get_jobtitle(self):
  return self.jobtitle

#sets job title if job title isn't an empty string
def set_jobtitle(self,title):
  if len(title) > 0:
    self.jobtitle = title

#return employee id
def get_employeeid(self):
  return "Employee ID: " + str(self.employeeid)

#returns salary
def get_salary(self):
  return "${:,.2f}".format(self.salary)

#sets salary if salary isn't an empty string
def set_salary(self,sal):
  if sal > 0:
    self.salary = sal

#increase salary
def increase_salary(self,percent):
  if percent > 0:
    self.set_salary(self.salary + self.salary * percent)
  else:
    print("Increase of salary must be greater than 0.")
```

TRY IT

**Directions**: If you don't already have this in the IDE, go ahead and enter it since we will be modifying this example with an updated base class and new subclasses.

Although we have `Employee` as our prior base class, we will want to consider other aspects of an employee. For example, we can have different types of employees. We could have full-time and part-time employees that get vacation hours and an annual salary as we currently have in our `Employee` class. We could also have contractors that get an hourly wage but don't accumulate vacation time or have an annual salary. Contractors could also have a contractor ID rather than an employee ID.

In order to build a correctly defined base class, we need to pull in only the key information that would be consistent across both the contractor and employee classes and make that base class. In our next example, we will define the base class as `Person` and only place in what is common.

By removing all items related to `salary` and `employee id`, we will have the following result.

EXAMPLE

```
import datetime
class Person:
  def __init__(self, fname, lname, title):
    self.firstname = fname
    self.lastname = lname
    self.jobtitle = title

    self.hiredate = datetime.date.today()

  #returns first name
  def get_firstname(self):
    return self.firstname

  #sets firstname if fname isn't an empty string
  def set_firstname(self,fname):
    if len(fname) > 0:
      self.firstname = fname

  #returns last name
  def get_lastname(self):
    return self.lastname

  #sets lastname if lname isn't an empty string
  def set_lastname(self,lname):
    if len(lname) > 0:
      self.lastname = lname
```

```
  #returns job title
  def get_jobtitle(self):
    return self.jobtitle

  #sets job title if job title isn't an empty string
  def set_jobtitle(self,title):
    if len(title) > 0:
      self.jobtitle = title
```

TRY IT

**Directions**: Go ahead and remove all aspects of `salary` and `employee id` information from this updated base class called `Person` now.

Make sure your program looks like the code above. Now, with the `Person` base class we are set up to create our unique subclasses.

---

# 2. Creating the Subclasses Employee and Contractor

#### The Employee Subclass

In our `Person` base class, we have the common information for all information still defined: `firstname`, `lastname`, `jobtitle` and the `hiredate` variable. Now that we have that content in place, we can create the `Employee` subclass to extend the `Person` base class and have the custom content that makes it unique. We'll be taking most of the elements that we had in the prior base class that we had initially set up.

EXAMPLE

```
class Employee(Person):
  def __init__(self,fname,lname,title,sal,empid):
    super().__init__(fname,lname,title)
    self.self.employeeid = empid
    self.salary = sal

  #return employee id
  def get_employeeid(self):
    return "Employee ID: " + str(self.employeeid)

  #returns salary
  def get_salary(self):
    return "${:,.2f}".format(self.salary)
```

```
#sets salary if salary isn't an empty string
def set_salary(self,sal):
  if sal > 0:
    self.salary = sal

#increase salary
def increase_salary(self,percent):
  if percent > 0:
    self.set_salary(self.salary + self.salary * percent)
  else:
    print("Increase of salary must be greater than 0.")
```

Most of the methods in this new `Employee` subclass were the same as the original `Employee` base class that we initially started from, namely the addition of employee ID and salary information. We did add some new items though. We did add the `__init__` method with the `self` parameter first, followed by the `fname`, `lname`, and `title` that is declared in the `Person` base class. We also added the parameters of `sal` (salary), and `empid` (employee ID). Next, we added the `super()` function call to the base class so in this case, we're calling the `__init__` method of the `Person` base class from the `Employee` subclass passing in the `fname`, `lname` and `title` values. This will call the `__init__` method in the `Person` base class and set those `variables`. Below the `__init__` method, the variables `employeeid` and `salary` are also set.

TRY IT

**Directions**: Now add the `Employee` subclass to your program below your `Person` base class. Make sure to watch your indentations; this new subclass should be at the same level as the base class. Otherwise, this subclass will be "inside" the base class if it is indented at the class declaration line.

**methods: vacationdaysperyear and vacationdays**

We did say that employees should receive vacation days. Let's say by default for this organization, all employees have 14 days (2 weeks) of vacation. Now it is helpful to have two different variables set for vacations —one for the yearly total (14) and one for the actual days remaining for the employee. We will add the variable called `vacationdaysperyear` and set it to 14. Next, we will create the variable called `vacationdays` which is set to `vacationdaysperyear` as default. We will update our `__init__` method to include those attributes.

EXAMPLE

```
def __init__(self,fname,lname,title,sal,empid):
  super().__init__(fname,lname,title)
  self.employeeid = empid
  self.salary = sal
  self.vacationdaysperyear = 14
  self.vacationdays = self.vacationdaysperyear
```

TRY IT

**Directions**: Go ahead and add these additional attributes to the `__init__` method of the `Employee` subclass.

**method: increase_vacation_days_per_year**

There will be a few methods that will be specific to the vacations. One method will be to increase the vacation days per year. Typically this could be increased by negotiation or based on how long the employee has been at the company.

EXAMPLE

```
#increase vacation days per year
def increase_vacation_days_per_year(self,days):
  if days > 0:
    self.vacationdaysperyear = self.vacationdaysperyear + days
```

We defined a method called `increase_vacation_days_per_year()` and set parameters of `self` and `days`. That way, we can pass in an integer to change the default vacations. Next, we have a conditional statement that looks to see if the number passed is larger than 0. If it is, we add that value to the `vacationdays` attribute of the `__init__` method.

TRY IT

**Directions**: Go ahead and add this `increase_vacation_days_per_year()` method to the `Employee` subclass.

**method: increase_vacation_days**

The next method we will add will be one that will increase the actual vacation days if they were granted. We will again check if days is greater than 0, and if so, we can add to the existing attribute `vacationdays`.

EXAMPLE

```
#increase vacation days
def increase_vacation_days(self,days):
  if days > 0:
    self.vacationdays = self.vacationdays + days
```

TRY IT

**Directions**: Go ahead and add this `increase_vacation_days()` method to the `Employee` subclass.

**method: increase_vacation_days_yearly**

The other possibility is if the employee has been there for one year, we can increase `vacationdays` by `vacationdaysperyear`.

EXAMPLE

```
#increase vacation days by year
```

```
  def increase_vacation_days_yearly(self):
    self.vacationdays = self.vacationdays + self.vacationdaysperyear
```

Here we defined a method called `increase_vacation_days_yearly()` and since this is just adding `vacationdays` with `vacationdaysperyear`, we do not need to add any extra parameters. This method will just do the addition when called.

TRY IT

**Directions**: Go ahead and add this `increase_vacation_days_yearly()` method to the `Employee` subclass.

**method: take_vacation_days**

We will need to have a method that we will use when an employee wants to take some days off. This method will accept the number of requested days off. As long as the value is greater than 0 and the employee still has days left that are greater than the days available, it will be permitted. Otherwise, if the days requested are less than or equal to 0, meaning the employee entered 0 or does not have enough vacation days left per the request, we will inform the employee.

EXAMPLE

```
#take some vacation days
def take_vacation_days(self,days):
  if days > 0 and self.vacationdays - days >= 0:
    self.vacationdays = self.vacationdays - days
  elif days <= 0:
    print("Vacation days taken must be greater than 0.")
  elif self.vacationdays - days < 0:
    print(f"Employee does not have enough vacation days to take off {days} days.")
```

Here we defined a method called `take_vacation_days()` with the `self` parameter and days that will accept the requested days off argument. In the conditional statement, if `days` entered is greater than 0 and `vacationdays` minus days (days requested) is greater than or equal to 0 (meaning vacation days left over will still be above 0 days or exactly 0 days—they exhausted their vacation days completely), the request is permitted. Current vacation days (`vacationdays`) is then subtracted by the requested vacation days (days). If the days entered is less than or equal to 0 (meaning the days request was 0 or a negative number), output will be provided that the "Vacation days taken must be greater than 0". Finally, the last condition is if the days requested minus the current `vacationdays` is less than 0 (meaning the requested time off is greater than what the employee has left in their vacation bucket), a formatted output of "Employee does not have enough vacation days to take off {days}" with {days} being the days requested, is output to the screen.

TRY IT

**Directions**: Go ahead and add this multi-conditional `take_vacation_days()` method to the `Employee` subclass.

**method: get_vacation_days**

We'll also have a simple method to return the number of vacation days.

EXAMPLE

```
  #return vacation days
  def get_vacation_days(self):
    return "Vacation Days: " + str(self.vacationdays)
```

This `get_vacation_days()` method will return what is left of the `vacationdays`.

TRY IT

**Directions**: Add the final returning `get_vacation_days()` method to the `Employee` subclass.

**Directions**: Before we add some instance calls to test this subclass, make sure your program looks like the following.

EXAMPLE

```
import datetime
class Person:
  def __init__(self, fname, lname, title):
    self.firstname = fname
    self.lastname = lname
    self.jobtitle = title

    self.hiredate = datetime.date.today()

  #returns first name
  def get_firstname(self):
    return self.firstname

  #sets firstname if fname isn't an empty string
  def set_firstname(self,fname):
    if len(fname) > 0:
      self.firstname = fname

  #returns last name
  def get_lastname(self):
    return self.lastname

  #sets lastname if lname isn't an empty string
  def set_lastname(self,lname):
    if len(lname) > 0:
      self.lastname = lname
```

```
  #returns job title
  def get_jobtitle(self):
    return self.jobtitle

  #sets job title if job title isn't an empty string
  def set_jobtitle(self,title):
    if len(title) > 0:
      self.jobtitle = title

class Employee(Person):
  def __init__(self,fname,lname,title,sal,empid):
    super().__init__(fname,lname,title)
    self.employeeid = empid
    self.salary = sal
    self.vacationdaysperyear = 14
    self.vacationdays = self.vacationdaysperyear

  #return employee id
  def get_employeeid(self):
    return "Employee ID: " + str(self.employeeid)

  #returns salary
  def get_salary(self):
    return "${:,.2f}".format(self.salary)

  #sets salary if salary isn't an empty string
  def set_salary(self,sal):
    if sal > 0:
      self.salary = sal

  #increase salary
  def increase_salary(self,percent):
    if percent > 0:
      self.set_salary(self.salary + self.salary * percent)
    else:
      print("Increase of salary must be greater than 0.")

  #return vacation days
  def get_vacation_days(self):
    return "Vacation Days: " + str(self.vacationdays)
```

```
  #increase vacation days per year
  def increase_vacation_days_per_year(self,days):
    if days > 0:
      self.vacationdaysperyear = self.vacationdaysperyear + days

  #increase vacation days
  def increase_vacation_days(self,days):
    if days > 0:
      self.vacationdays = self.vacationdays + days

  #increase vacation days by year
  def increase_vacation_days_yearly(self):
    self.vacationdays = self.vacationdays + self.vacationdaysperyear

  #take some vacation days
  def take_vacation_days(self,days):
    if days > 0 and self.vacationdays - days >= 0:
      self.vacationdays = self.vacationdays - days
    elif days <= 0:
      print("Vacation days taken must be greater than 0.")
    elif self.vacationdays - days < 0:
      print(f"Employee does not have enough vacation days to take off {days} days.")
```

Let's test out the code, especially around the vacation days methods in the `Employee` subclass to ensure all is working as expected. First we will create an instance of the subclass called `emp1` (employee 1). We will pass some employee arguments for first name, last name, title, salary, and employee ID. Then we will create some `print()` functions so we can see output to the screen.

### EXAMPLE

```
emp1 = Employee('Jack','Krichen','Manager',50000,1)

print(emp1.get_firstname())
print(emp1.get_lastname())
print(emp1.get_employeeid())
print(emp1.get_jobtitle())
print(emp1.get_salary())

print(emp1.get_vacation_days())
emp1.take_vacation_days(10)
print(emp1.get_vacation_days())
emp1.take_vacation_days(10)
emp1.take_vacation_days(-1)
```

```
emp1.increase_vacation_days_yearly()
print(emp1.get_vacation_days())
```

TRY IT

**Directions**: Add the instance calls and `print()` functions to your program. Give the employee a first name, last name, title, salary, and employee ID. To keep consistent with this example, test with the vacation days indicated. Once entered, run the program.

```
Jack
Krichen
Employee ID: 1
Manager
$50,000.00
Vacation Days: 14
Vacation Days 4
Employee does not have enough vacation days to take off 10 days.
Vacation days taken must be greater than 0.
Vacation Days: 18
```

In the output, we should see the employee's first name, last name, employee ID, title, and salary for the first five `print()` functions.

Next, notice that the first output of the current vacation days is 14. This is correct since the variable `vacation_days` was initially set to the variable `vacationdaysperyear`, which has 14 as the default value. Then, we pass 10 vacation days as a request (argument) to the `take_vacation_days()` method and print out the value of `vacation_days` once subtracted from the default—so 4 days left is also correct. We then try to take another 10 days of vacation; however, we get an error message since there's not enough vacation days left (we only had 4 days left the first time). Next, we try to take a negative number of vacation days which also returns an error that the argument (request for days off) needs to be greater than 0. Lastly, we increase the vacation days based on the yearly increase and accurately see 18 days, as there were 4 days left and the yearly increase was 14 days.

TRY IT

**Directions**: Now that we have a working `Employee` subclass, try changing a few arguments to see if you can change what is output to the screen.

### The Contractor Subclass

Our next step will be to create the `Contractor` subclass. The framework of this class will be the same structure as the `Employee` subclass with some small differences. In particular, we'll have a `contractorid` instead of the `employeeid`. There will also be an hourly wage instead of a salary, and no vacation.

EXAMPLE

```
class Contractor(Person):
```

```
  def __init__(self,fname,lname,title,wage,contractorid):
    super().__init__(fname,lname,title)
    self.contractorid = contractorid
    self.hourlywage = wage

  #return contractor id
  def get_contractorid(self):
    return "Contractor ID: " + str(self.contractorid)

  #set hourly wage
  def set_hourlywage(self, wage):
    if wage > 0:
      self.hourlywage = wage

  #returns wage
  def get_hourlywage(self):
    return "${:,.2f}".format(self.hourlywage)

  #sets job wage if wage greater than 0
  def set_get_hourlywage(self,get_hourlywage):
    if get_hourlywage > 0:
      self.wage = get_hourlywage
```

Most of this should be quite familiar, as the coding structure is the same in this subclass, with some slight differences with the names of the variables from the `Employee` subclass. We swapped `wage` and `contractorid` parameters in the `__init__` method instead of `sal` and `empid`. We also switched the attributes below that method to `contractorid` and `hourlywage` accordingly.

The `get_contractorid()` method was copied from the `get_employeeid()` method. The `set_hourlywage()` method is a copy of `set_salary()` method and `get_hourlywage()` is a copy of `get_salary()` method. Lastly, `set_get_hourlywage` checks and sets the job wage.

TRY IT

**Directions**: Add all the elements of the `Contractor` subclass.

EXAMPLE

```
import datetime
class Person:
  def __init__(self, fname, lname, title):
    self.firstname = fname
    self.lastname = lname
    self.jobtitle = title
```

```
    self.hiredate = datetime.date.today()

  #returns first name
  def get_firstname(self):
    return self.firstname

  #sets firstname if fname isn't an empty string
  def set_firstname(self,fname):
    if len(fname) > 0:
      self.firstname = fname

  #returns last name
  def get_lastname(self):
    return self.lastname

  #sets lastname if lname isn't an empty string
  def set_lastname(self,lname):
    if len(lname) > 0:
      self.lastname = lname

  #returns job title
  def get_jobtitle(self):
    return self.jobtitle

  #sets job title if job title isn't an empty string
  def set_jobtitle(self,title):
    if len(title) > 0:
      self.jobtitle = title

class Employee(Person):
  def __init__(self,fname,lname,title,sal,empid):
    super().__init__(fname,lname,title)
    self.employeeid = empid
    self.salary = sal
    self.vacationdaysperyear = 14
    self.vacationdays = self.vacationdaysperyear

  #return employee id
  def get_employeeid(self):
    return "Employee ID: " + str(self.employeeid)

  #returns salary
```

```
def get_salary(self):
  return "${:,.2f}".format(self.salary)

#sets salary if salary isn't an empty string
def set_salary(self,sal):
  if sal > 0:
    self.salary = sal

#increase salary
def increase_salary(self,percent):
  if percent > 0:
    self.set_salary(self.salary + self.salary * percent)
  else:
    print("Increase of salary must be greater than 0.")

#return vacation days
def get_vacation_days(self):
  return "Vacation Days: " + str(self.vacationdays)

#increase vacation days per year
def increase_vacation_days_per_year(self,days):
  if days > 0:
    self.vacationdaysperyear = self.vacationdaysperyear + days

#increase vacation days
def increase_vacation_days(self,days):
  if days > 0:
    self.vacationdays = self.vacationdays + days

#increase vacation days by year
def increase_vacation_days_yearly(self):
  self.vacationdays = self.vacationdays + self.vacationdaysperyear

#take some vacation days
def take_vacation_days(self,days):
  if days > 0 and self.vacationdays - days >= 0:
    self.vacationdays = self.vacationdays - days
  elif days <= 0:
    print("Vacation days taken must be greater than 0.")
  elif self.vacationdays - days < 0:
    print(f"Employee does not have enough vacation days to take off {days} days.")
```

```
class Contractor(Person):
  def __init__(self,fname,lname,title,wage,contractorid):
    super().__init__(fname,lname,title)
    self.contractorid = contractorid
    self.hourlywage = wage

  #return contractor id
  def get_contractorid(self):
    return "Contractor ID: " + str(self.contractorid)

  #set hourly wage
  def set_hourlywage(self, wage):
    if wage > 0:
      self.hourlywage = wage

  #returns wage
  def get_hourlywage(self):
    return "${:,.2f}".format(self.hourlywage)

  #sets job wage if wage greater than 0
  def set_get_hourlywage(self,get_hourlywage):
    if get_hourlywage > 0:
      self.wage = get_hourlywage

con = Contractor('Temp','Emp','Developer',60,2)
print(con.get_firstname())
print(con.get_lastname())
print(con.get_contractorid())
print(con.get_jobtitle())
print(con.get_hourlywage())
print(con.set_hourlywage(50))
print(con.get_hourlywage())
```

TRY IT

**Directions**: Add the instance calls and `print()` functions to your program. Give the contractor a first name, last name, title, hourly wage, and contractor ID. To keep consistent with this example, test with the hourly wage indicated. Once entered, run the program.

As we see, the output and contents are slightly different.

```
Temp
Emp
Contractor ID: 2
```

```
Developer
$60.00
None
$50.00
```

In the output, we should see the contractor's first name, last name, contractor ID, title, and hourly wage for the first five `print()` functions.

Then we changed the hourly wage to $50 an hour using the `set_hourlywage()` method and reprinted the hourly wage again using the `get_hourlywage()` method.

As you look at the code to test, think about how else you would test the code to ensure that it works correctly. What values would you try to set?

To see the final version of this program visit **Sophia's Python code page**

## SUMMARY

In this lesson, we moved the standard attributes and methods to a `Person` **base class** that we wanted to exist globally. Then, we took the `Employee` specific attributes and methods and placed those into a new `Employee` subclass. We added methods to the `Employee` **subclass** to account for salary and vacation days and tested our program for vacation requests against what an employee has in their current vacation bank. Finally, we **created the** `Contractor` **subclass** and introduced an hourly wage as opposed to a salary. In both subclasses, we added a unique ID method only associated with those subclasses.

Best of luck in your learning!