

## Challenge 2: Handling a Bank Account

In this challenge, you will define methods for handling a bank account using concepts of inheritance.

### WE'LL COVER THE FOLLOWING ^

- Problem Statement
  - Task 1
  - Task 2
  - Task 3
  - Task 4
- Coding Exercise

## Problem Statement #

In this challenge, we will be extending the previous challenge and implementing methods in the *parent class* and its corresponding *child class*.

The initializers for both classes have been defined for you.

### Task 1 #

In the `Account` class, implement `getBalance()` *method* that **returns** `balance`.

### Task 2 #

In the `Account` class, implement `deposit(amount)` *method* that adds `amount` to the `balance`. It **does not** return anything.

### Sample Input

```
balance = 2000
deposit(500)
getbalance()
```

## Sample Output

```
2500
```

## Task 3 #

In the `Account` class, implement `withdrawal(amount)` *method* that subtracts the `amount` from the `balance`. It **does not** return anything.

## Sample Input

```
balance = 2000  
withdrawal(500)  
getbalance()
```

## Sample Output

```
1500
```

## Task 4 #

In the `SavingsAccount` class, implement a `interestAmount()` *method* that **returns** the interest amount of the **current** balance. Below is the formula for calculating the interest amount:

$$\text{interest amount} = \frac{\text{interest rate} \times \text{balance}}{100}$$

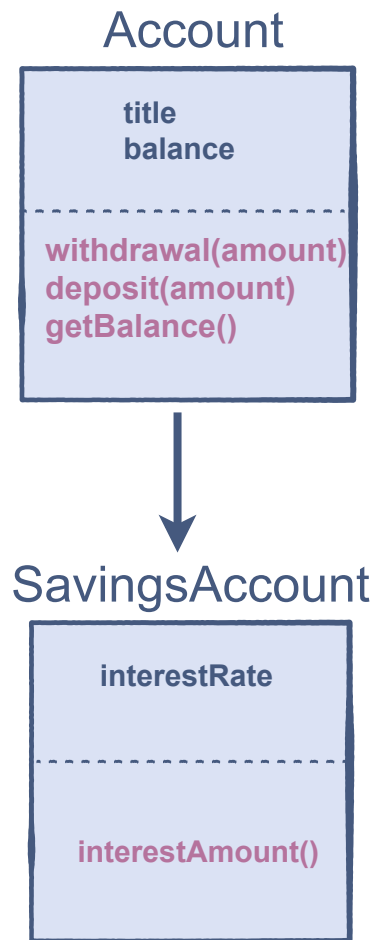
## Sample Input

```
balance = 2000  
interestRate = 5  
interestAmount()
```

## Sample Output

```
100
```

The following figure shows what the result should logically look like:



Based and Derived Classes Structure

## Coding Exercise #

First, take a close look and design a step-by-step algorithm before jumping to the implementation. This problem is designed for your practice, so initially try to solve it on your own. If you get stuck, you can always refer to the solution provided in the solution review.

**Good luck!**

**Note:** A new `SavingsClass` object is initialized at the end of the code and test results will base on it.

```
class Account:
    def __init__(self, title=None, balance=0):
        self.title = title
        self.balance = balance

    def withdrawal(self, amount):
        # write code here
        pass
```



```
pass

def deposit(self, amount):

    # write code here
    pass

def getBalance(self):
    # write code here
    pass


class SavingsAccount(Account):
    def __init__(self, title=None, balance=0, interestRate=0):
        super().__init__(title, balance)
        self.interestRate = interestRate

    def interestAmount(self):
        # write code here
        pass


# code to test - do not edit this
demo1 = SavingsAccount("Mark", 2000, 5) # initializing a SavingsAccount object
```



---

The solution will be explained in the next lesson.