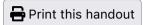
CSC110 Lecture 28: Inheritance



Exercise 1: Inheritance

- 1. Answer the following questions to review the terminology we have covered so far this lecture.
 - a. What is an abstract method?

b. What is an abstract class?

A class that define (or inherite, without overriding) at least one abstract we that.

c. Consider the following Python class. Is it abstract or concrete?

```
class MyClass:

   def do_something(self, x: int) -> int:
        return x + 5

   def do_something_else(self, y: int) -> int:
        raise NotImplementedError
```

Abstract, since it has a method (do-something-else) that is abstract.

2. Consider the Stack inheritance hierarchy introduced in lecture, where the abstract class Stack is the parent class of both Stack1 and Stack2. For each of the following code snippets in the Python

console, write the output or describe the error that would occur, and explain.

```
>>> s = Stack2()
a.
                                                       È
  >>> isinstance(s, Stack1)
   False
             since Stack 2 is not a child of Stack 1.
 >>> s = Stack1()
                                                       Ê
  >>> Stack.push(s, 'book')
  >>> Stack.pop(s)
   The Nothing Lemented Error would be raised
   Since the abstract method Stack. Push
      would be called.
c. | >>> s = Stack()
                                                       Ê
  >>> s.push('paper')
    As in (b) since Steck is an abstract
     class and Stack. Pash is not
```

- 3. We have said that inheritance serves as another form of *contract*:
 - The implementor of the subclass must implement the methods from the abstract superclass.

implemented.

o Any user of the subclass may assume that they can call the superclass methods on instances of the subclass.

What happens if we violate this contract? Once again, consider the classes Stack and Stack1, excellent this time, the method Stack1.is_empty is missing:



```
class Stack1(Stack):
    000,,,000
   # Private Instance Attributes
    # items: The elements in the stack
    items: list
    def __init__(self) -> None:
        """Initialize a new empty stack."""
        self. items = []
    def push(self, item: Any) -> None:
        """Add a new element to the top of this stack.
        self._items.append(item)
    def pop(self) -> Any:
        """Remove and return the element at the top of this stack.
        Preconditions:
            - not self.is empty()
        0.00
        return self._items.pop()
```

Try executing the following lines of code in the Python console—what happens?

```
>>> s = Stack1()
>>> s.push('pancake')
>>> s.is_empty()
```

The method Stack. is empty is in herited but it is abstract, so a Not Implemental Error is raised.

Exercise 2: Polymorphism

Consider the function weird below:

```
def weird(stacks: list[Stack]) -> None:
    for stack in stacks:
        if stack.is_empty():
            stack.push('pancake')
        else:
            stack.pop()
```

1. Suppose we execute the following code in the Python console:

```
>>> list_of_stacks = [Stack1(), Stack2(), Stack1(), Stack2()]
>>> list_of_stacks[0].push('chocolate')
>>> list_of_stacks[2].push('chocolate')
```

Now suppose we call weird(list_of_stacks). Given the list list_of_stacks, write the specific push or pop method that would be called at each loop iteration. The first is done for you.

weird loop iteration	push/pop version		
0	Stac(1)pop		
1	Stack 2. Push		
2	Stack 1. pop		
3	Stack 1. pop Stack 2. push		

2. Write a code snippet in the Python console that results in a variable list_of_stacks2 that, if passed to weird, would result in the following sequence of push/pop method calls: Stack1.push, Stack2.push, Stack1.pop, Stack2.pop.

3. Create a list list_of_stacks3 that, if passed to weird, would raise a NotImplementedError or the second loop iteration.

Additional Exercise: The object superclass and overriding methods

1. Does our Stack abstract class have a parent class? If so, what is it? If not, why not?

2. Suppose we have a variable my_stack = Stack1(). What information does the string representation str(my_stack) display?

3. In the space below, override the __str__ method for the Stack1 class, so that the string representation matches the format shown in the docstring.

Note: You should call str on each item stored in the stack.



```
class Stack1(Stack):
    items: list
   # ... other code omitted
   def __str__(self) -> str:
        """Return a string representation of this stack.
        >>> s = Stack1()
        >>> str(s)
        'Stack1: empty'
        >>> s.push(10)
        >>> s.push(20)
        >>> s.push(30)
        >>> str(s)
        'Stack1: 30 (top), 20, 10'
        Notes:
            - because this is a method, you may access the _items attribute
            - call str on each element of the stack to get string
        representations
             of the items
            - review the str.join method
            - you can reverse the items in a list by calling reversed on it
              (returns a new iterable) or the list.reverse method (mutates
        the list)
```

0.00