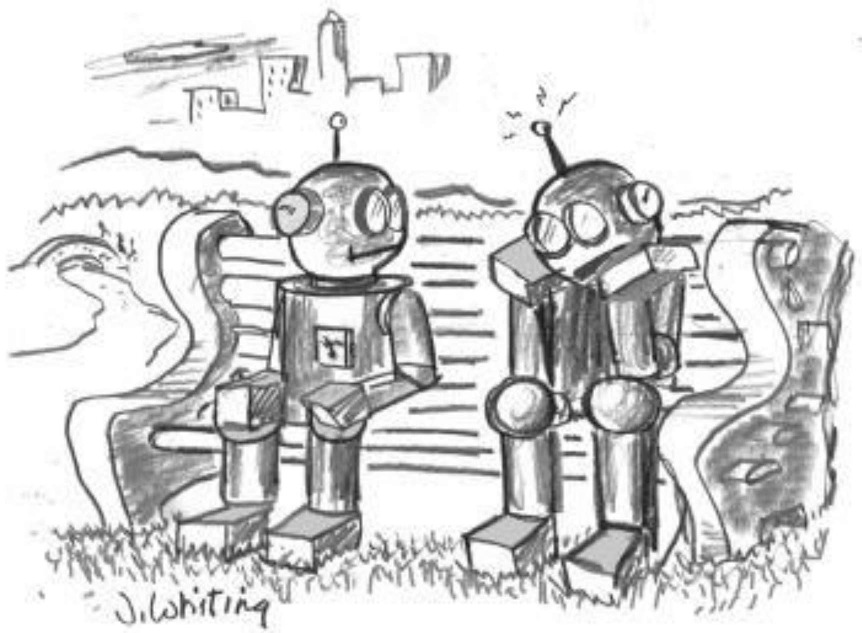
CSc 110, Autumn 2016

Lecture 33: Inheritance

Adapted from slides by Marty Stepp and Stuart Reges



“All I did is what he told me to do and I’m the one who’s a moron!”

Calling overridden methods

- Subclasses can call overridden methods with `super`

  `super(`**ClassName**`, self).`**method**`(`**parameters**`)`

  - Example:

```
class LegalSecretary(Secretary):
    def get_salary(self):
        base_salary = super(LegalSecretary, self).get_salary()
        return base_salary + 5000.0
    ...
```

Inheritance and constructors

- Imagine that we want to give employees more vacation days the longer they've been with the company.
  - For each year worked, we'll award 2 additional vacation days.
  - When an Employee object is constructed, we'll pass in the number of years the person has been with the company.
  - This will require us to modify our `Employee` class and add some new state and behavior.
  - Exercise: Make necessary modifications to the `Employee` class.

Modified Employee class

```
class Employee:
    def __init__(self, initial_years):
        self.__years = initial_years

    def get_hours(self):
        return 40

    def get_salary(self):
        return 50000.0

    def get_vacation_days(self):
        return 10 + 2 * self.__years

    def get_vacation_form(self):
        return "yellow"
```

Problem with constructors

- Now that we've added the constructor to the `Employee` class, our subclasses do not compile. The error:

```
TypeError: __init__() missing 1 required positional
  argument: 'initial_years'
```

  - The short explanation: Once we write a constructor (that requires parameters) in the superclass, we must now write constructors for our employee subclasses as well.

Modified `Marketer` class

```
# A class to represent marketers.
class Marketer(Employee):
    def __init__(years):
        super(Marketer, self).__init__(years)

    def advertise():
        print("Act now while supplies last!")

    def get_salary():
        return super(Marketer, self).get_salary() + 10000.0
```

- Exercise: Modify the `Secretary` subclass.
  - Secretaries' years of employment are not tracked.
  - They do not earn extra vacation for years worked.

Modified `Secretary` class

```
# A class to represent secretaries.
class Secretary(Employee):
    def __init__(self):
        super(Secretary, self).__init__(0)

    def take_dictation(self, text):
        print("Taking dictation of text: " + text)
```

- Since `Secretary` doesn't require any parameters to its constructor, `LegalSecretary` runs fine without a constructor.

Inheritance and fields

- Try to give lawyers $5000 for each year at the company:

```
class Lawyer(Employee):
    ...
    def get_salary(self):
        return super(Lawyer, self).get_salary() + 5000 * self.__years
    ...
```

- Does not work; the error is the following:

```
AttributeError: 'Lawyer' object has no attribute '_Employee__years'
  ^
```

- Private fields cannot be directly accessed from subclasses.
  - One reason: So that subclassing can't break encapsulation.
  - How can we get around this limitation?

Improved `Employee` code

Add an accessor for any field needed by the subclass.

```
class Employee:
    self.__years

    def __init__(self, initial_years):
        self.__years = initial_years

    def get_years(self):
        return self.__years
    ...

class Lawyer(Employee):
    def __init__(self, years):
        super(Lawyer, self).__init__(years)

    def get_salary(self):
        return super(Lawyer, self).get_salary() + 5000 * get_years()
    ...
```

Revisiting `Secretary`

- The `Secretary` class currently has a poor solution.
  - We set all Secretaries to 0 years because they do not get a vacation bonus for their service.
  - If we call `get_years` on a `Secretary` object, we'll always get 0.
  - This isn't a good solution; what if we wanted to give some other reward to *all* employees based on years of service?

- Redesign our `Employee` class to allow for a better solution.

Improved `Employee` code

- Let's separate the standard 10 vacation days from those that are awarded based on seniority.

```
class Employee:
    def __init__(self, initial_years):
        self.__years = initial_years

    def get_vacation_days(self):
        return 10 + self.get_seniority_bonus()

    # vacation days given for each year in the company
    def get_seniority_bonus(self):
        return 2 * self.__years
    ...
```

- How does this help us improve the `Secretary`?

Improved Secretary code

- Secretary can selectively override get_seniority_bonus; when get_vacation_days runs, it will use the new version.
  - Choosing a method at runtime is called *dynamic binding*.

```
class Secretary(Employee):
    def __init__(self, years):
        super(Secretary, self).__init__(years)

    # Secretaries don't get a bonus for their years of service.
    def get_seniority_bonus(self):
        return 0

    def take_dictation(self, text):
        print("Taking dictation of text: " + text)
```

Critter exercise: `Anteater`

- Write a critter class `Anteater`:

Method	**Behavior**
`__init__`	
`eat`	Eats 3 pieces of food and then stops
`fight`	randomly chooses between pouncing and roaring
`get_color`	pink if hungry and red if full
`get_move`	walks up two and then down two
`__str__`	`"a"` if hungry `"A"` otherwise