LECTURE 6

OBJECT-ORIENTED PROGRAMMING

SUBCLASSES AND INHERITANCE II

MCS 275 Spring 2023
David Dumas

LECTURE 6: SUBCLASSES AND INHERITANCE II

Reminders and announcements:

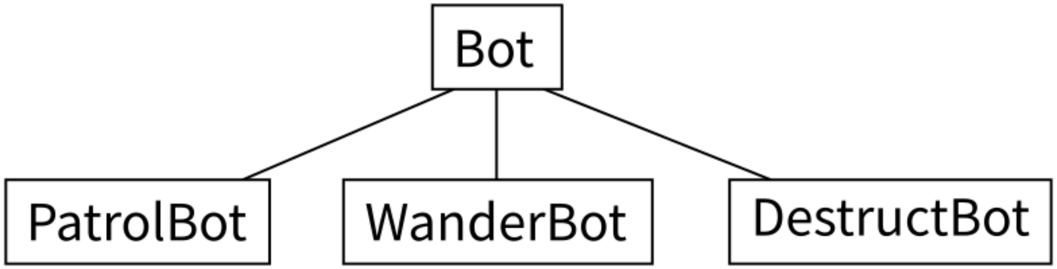
- Homework 3 is due Tuesday at Noon.
- Project 1 is posted. Please read it over to prepare for discussion in Monday's lecture.

PLAN

Finish our robot simulation class hierarchy

Discuss more OOP theory & practice

PLANNED BOT HIERARCHY



Bot

PatrolBot

WanderBot

DestructBot

- ✅ `WanderBot` walks about randomly.
- ✅`DestructBot` sits for a while and deactivates.
- 🟨`PatrolBot` walks back and forth.

SIMULATION

Instead of manual `.update()` experiments, there are two simulation programs:

- `botsimulation.py` - Active bots shown as `*`
- `botsimulation_fancy.py` - Bots have their own symbols, inactive ones are shown as ☠.

CLASS ATTRIBUTES

Attributes declared in the class definition, outside of any method, are **class attributes**.

Class attributes are shared by every instance of the class. Often used for constants.

Contrast with the **instance attributes** we have used thus far (e.g. `self.x = 1` in constructor) which exist separately for each instance.

PATROLBOT

Takes `direction` (vector) and `n` (int). Walks `n` steps of size `direction`, then `n` steps of size `-direction`. Repeats indefinitely.

This robot has internal **state**:

- Whether walking out or coming back
- How many steps it has taken in the current direction

FINITE STATE MACHINE

Keep track of which state we're in. Handle input differently depending on the state. Fixed set of possible states.

```
if state == "work":
    handle_at_work(sms_content)
elif state == "home":
    handle_at_home(sms_content)
```

Handlers may change state depending on the input.

```
def handle_at_home(sms_content):
    if announces_critical_outage(sms_content):
        send_reply("on my way")
        state = "work"
    else:
        # deal with it tomorrow
        return
```

FOUR PILLARS OF OOP

- **Encapsulation** - Objects manage their own private, internal state.
- **Abstraction** - Method calls express intent (independent of implementation).
- **Inheritance** - Distinct classes can share behavior.
- **Polymorphism** - Code using a class will also work on its subclasses.

EXTENDING THE SIMULATION

Beyond adding more robot types, how might me improve or extend the simulation?

EXTENDING THE SIMULATION

Might create a class `Arena` that manages the list of bots and the space in which they move. Would have a single `.update()` method that updates all bots.

`Arena` object would be made first, then passed to each robots constructor. Robots would call `Arena` methods to interrogate surroundings (e.g. avoid collision, seek other bots, ...)

REFERENCES

- I discussed inheritance in MCS 260 Fall 2021 Lecture 27
- See Lutz, Chapter 31 for more discussion of inheritance.
- Lutz, Chapters 26-32 discuss object-oriented programming.

REVISION HISTORY

- 2022-01-24 Last year's lecture on this topic finalized
- 2023-01-27 Updated version for spring 2023