# Object-oriented programming

CSC148, INTRODUCTION TO COMPUTER SCIENCE
DIANE HORTON, JONATHAN CALVER, SOPHIA HUYNH,
MISHA SCHWARTZ, & JACQUELINE SMITH

#### Key terms and phrases

class initializer

instance (of a class) dot notation

instance attribute self

method

#### Accessing an instance attribute

```
def whatever(self, x):
    y = expression
    vs
self.y = expression
```

Must use self to get at instance attributes!

#### Re-assigning self

```
def mutate(self, x):
    self = NewObject(x)
```

Re-assigning self doesn't mutate anything!

### Composition of classes

HOW DO WE DESIGN CODE IN WHICH DIFFERENT CLASSES INTERACT WITH EACH OTHER?

#### Users are objects too

A Twitter user has an id, a biography, and tweets (among other things).





A bear @A\_single\_bear · Sep 10 I wish I knew how to bark. Maybe humans would yell at me less. I am a bear.

↑ 228

@A\_single\_bear

Hello, I am a bear.

#### The start of a **User** class

```
class User:
    """A Twitter user.

=== Attributes ===
    userid: the userid of this Twitter user.
    bio: the bio of this Twitter user.
    tweets: a list of the tweets that this user has made.
    """
    userid: str
    bio: str
    tweets: List[Tweet]
```

#### **User and Tweet**

#### **Composition**

a relationship between two classes where instances of one class contain references to instances of the other

"has" relationship, e.g. "user has tweets"

## Representation invariants

HOW DO WE DOCUMENT PROPERTIES THAT MUST BE TRUE FOR EVERY INSTANCE OF A GIVEN CLASS?

#### Tweets can have at most 280 characters

Every instance attribute has a *type annotation*, which restricts the kind of value this attribute can have.

But we often want to restrict attributes values even further; what do we call these restrictions, and how do we communicate them?

#### Representation invariant

A **representation invariant** is a property of the instance attributes that every instance of a class must satisfy.

#### Example

- (in words) This tweet's content is at most 280 characters.
- o (in code) len(self.content) <= 280</pre>

#### Today: two questions about RIs

- 1. Why should we care about representation invariants?
- 2. How do we enforce representation invariants?

#### Representation invariants as assumptions

A representation invariant is a property that every instance of a class must satisfy.

When given an instance of that class, we can assume that every representation invariant is satisfied.

#### Representation invariants as assumptions

```
class Tweet:
    def like(self, n: int) -> None:
        self.likes += n
```

self is an instance of Tweet, so we assume that all RIs are satisfied when this method is called.

The representation invariants of Tweet are *preconditions for every Tweet method*.

#### Representation invariants as responsibilities

A representation invariant is a property that every instance of a class must satisfy.

When a method returns, we must ensure that every representation invariant is satisfied.

#### Representation invariants as responsibilities

```
class Tweet:
  def like(self, n: int) -> None:
    """Record the fact that this tweet
    received <n> likes."""
```

The method must ensure that, at the end, self.likes >= 0.

The representation invariants of Tweet are *postconditions for every Tweet method*.

#### How to meet our responsibility?

RIs must be true at all times, except during execution of a method.

Implementers of a class are responsible for ensuring this.

How can they protect against a method call that might violate an RI?

#### Strategy 1: Preconditions

Require client code to call methods with "good" inputs. Make no promises if it doesn't.

Therefore, the method doesn't need to check that preconditions are met.

#### Strategy 2: Ignore "bad" inputs

Accept a wide range of inputs.

If an input would cause an RI to be violated, do not perform the work of the method.

Also known as failing silently.

#### Strategy 3: Fix "bad" inputs

Accept a wide range of inputs.

If an input would cause an RI to be violated, change the input to a "reasonable" or default value.

Then continue with the rest of the method.

#### Discuss the pros and cons of each

**Strategy 1**: use preconditions

**Strategy 2**: ignore bad inputs

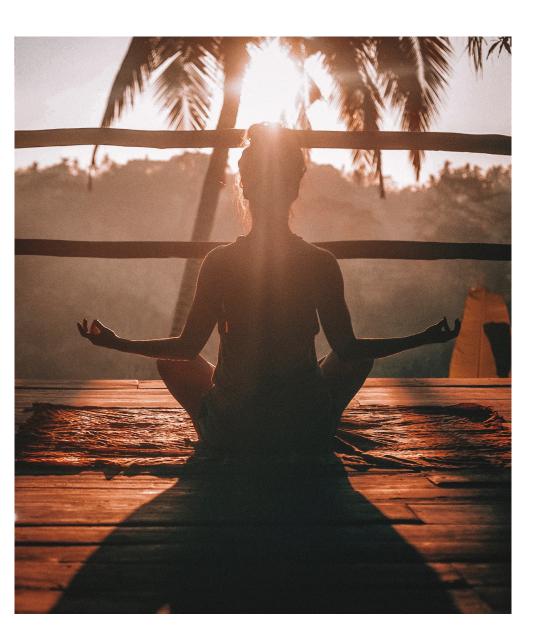
Strategy 3: fix bad inputs

#### Why we write down the RIs

The assumption part is handy, but is only legitimate if the responsibility part is fulfilled.

Writing down the Representation Invariants helps ensure:

- You remember these responsibilities
- Others on your team remember these responsibilities
- Now and a year from now when you/they are revising the code!



#### The Zen of Python

"Explicit is better than implicit."

#### Privacy

Instance attributes and methods can be marked as **private** by spelling their name with a leading underscore, e.g. \_content.

Marking an attribute/method as private signals that client code should not access it.

#### Privacy is about communication

A private attribute/method could be...

- very complicated
- subject to several representation invariants
- changed (in name, type, or meaning) at any time

#### Interface vs. Implementation



