Chapter 7 Object-Oriented Programming

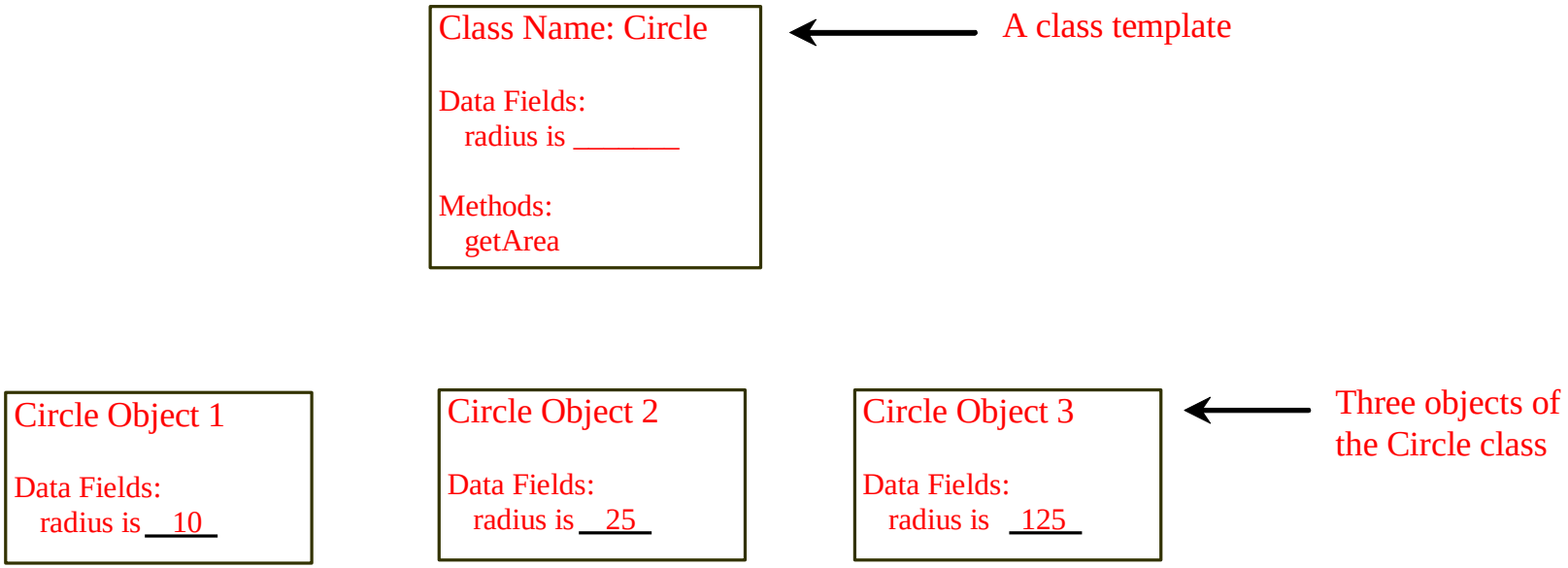
Objectives

- To describe objects and classes, and use classes to model objects
- To define classes
- To construct an object using a constructor that invokes the initializer to create and initialize data fields
- To access the members of objects using the dot operator (.)
- To hide data fields to prevent data corruption and make classes easy to maintain
- To apply class abstraction and encapsulation to software development

OO Programming Concepts

Object-oriented programming (OOP) involves programming using objects. An *object* represents an entity in the real world that can be distinctly identified. For example, a student, a desk, a circle, a button, and even a loan can all be viewed as objects.

Objects



Classes

A Python class uses variables to store data fields and defines methods (functions) to perform actions. Additionally, a class provides a special type method, known as *initializer*, which is invoked to create a new object. An initializer can perform any action, but an initializer is designed to perform initializing actions, such as creating the data fields of objects.

```
class ClassName:
    initializer
    methods
```

Circle

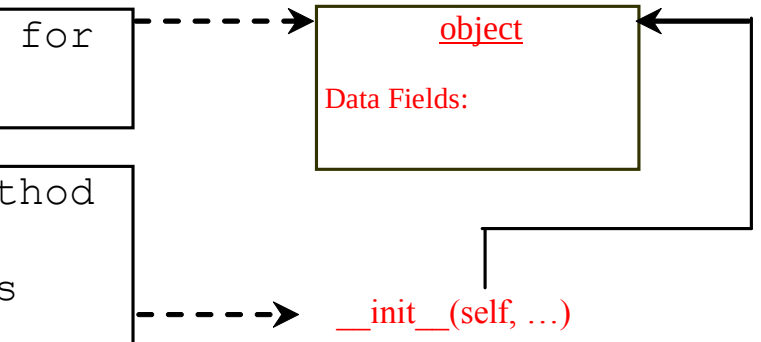
TestCircle

Constructing Objects

Once a class is defined, you can create objects from the class by using the following syntax, called a *constructor*:

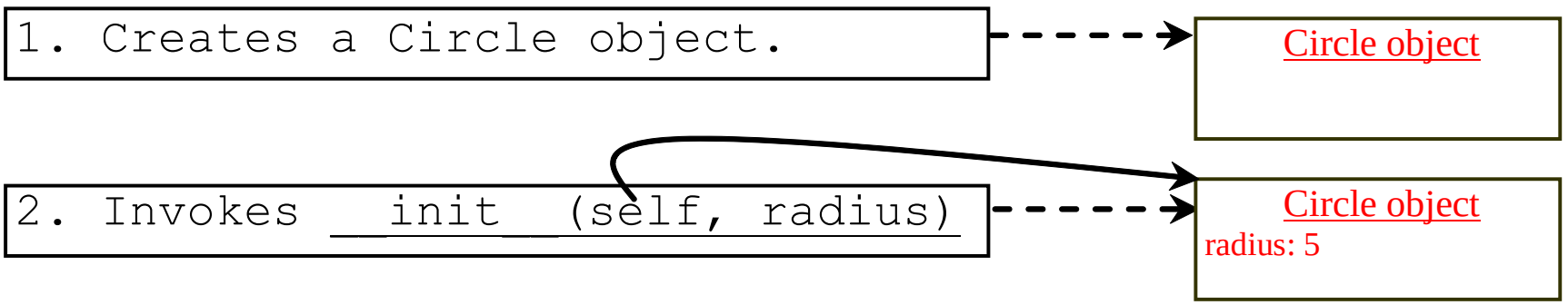
```
variable = className(arguments)
```

1. It creates an object in the memory for the class.
2. It invokes the class's `__init__` method to initialize the object. The `self` parameter in the `__init__` method is automatically set to reference the object that was just created.



Constructing Objects

The effect of constructing a Circle object using Circle(5) is shown below:



Accessing Objects

After an object is created, you can access its data fields and invoke its methods using the dot operator (.), also known as the *object member access operator*. For example, the following code accesses the radius data field and invokes the getPerimeter and getArea methods.

```
>>> from Circle import Circle
>>> c = Circle(5)
>>> c.getPerimeter()
31.415926535897932
>>> c.radius = 10
>>> c.getArea()
314.15926535897932
```

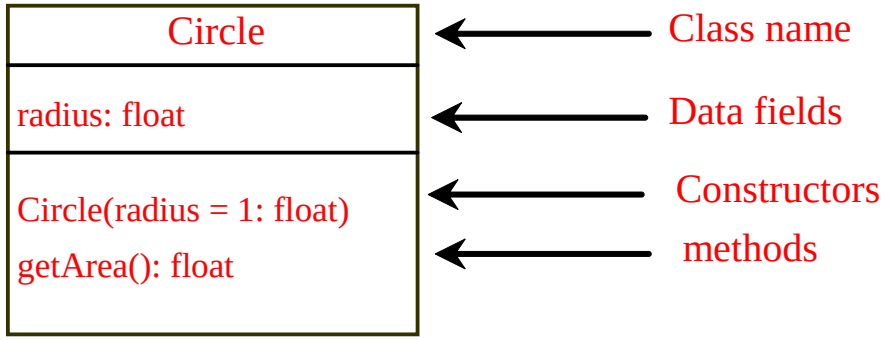
Why self?

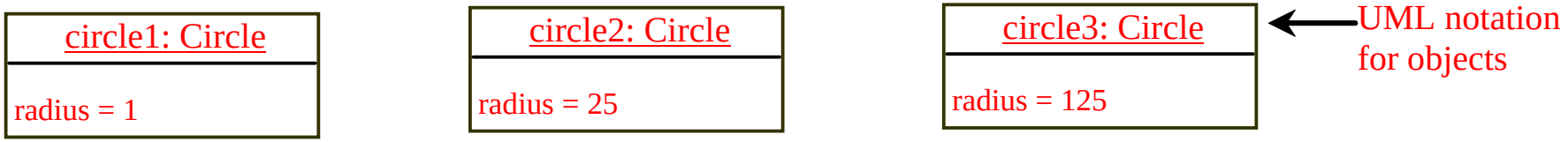
Note that the first parameter is special. It is used in the implementation of the method, but not used when the method is called. So, what is this parameter self for? Why does Python need it?

self is a parameter that represents an object. Using self, you can access instance variables in an object. Instance variables are for storing data fields. Each object is an instance of a class. Instance variables are tied to specific objects. Each object has its own instance variables. You can use the syntax self.x to access the instance variable x for the object self in a method.

UML Class Diagram

UML Class Diagram





Example: Defining Classes and Creating Objects

TV	
channel: int	The current channel (1 to 120) of this TV.
volumeLevel: int	The current volume level (1 to 7) of this TV.
on: bool	Indicates whether this TV is on/off.
TV()	Constructs a default TV object.
turnOn(): None	Turns on this TV.
turnOff(): None	Turns off this TV.
getChannel(): int	Returns the channel for this TV.
setChannel(channel: int): None	Sets a new channel for this TV.
getVolume(): int	Gets the volume level for this TV.
setVolume(volumeLevel: int): None	Sets a new volume level for this TV.
channelUp(): None	Increases the channel number by 1.
channelDown(): None	Decreases the channel number by 1.
volumeUp(): None	Increases the volume level by 1.
volumeDown(): None	Decreases the volume level by 1.

TV

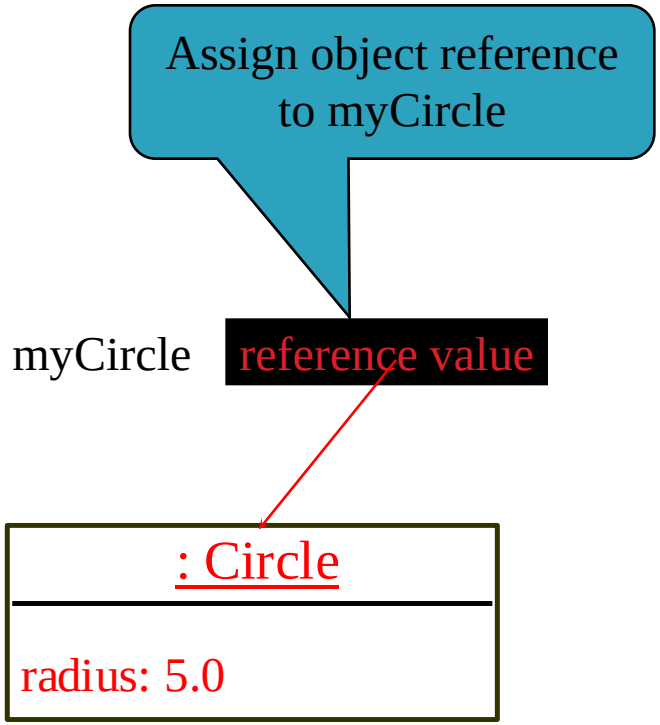
TestTV

Trace Code

```
myCircle = Circle(5.0)

yourCircle = Circle()

yourCircle.radius = 100
```

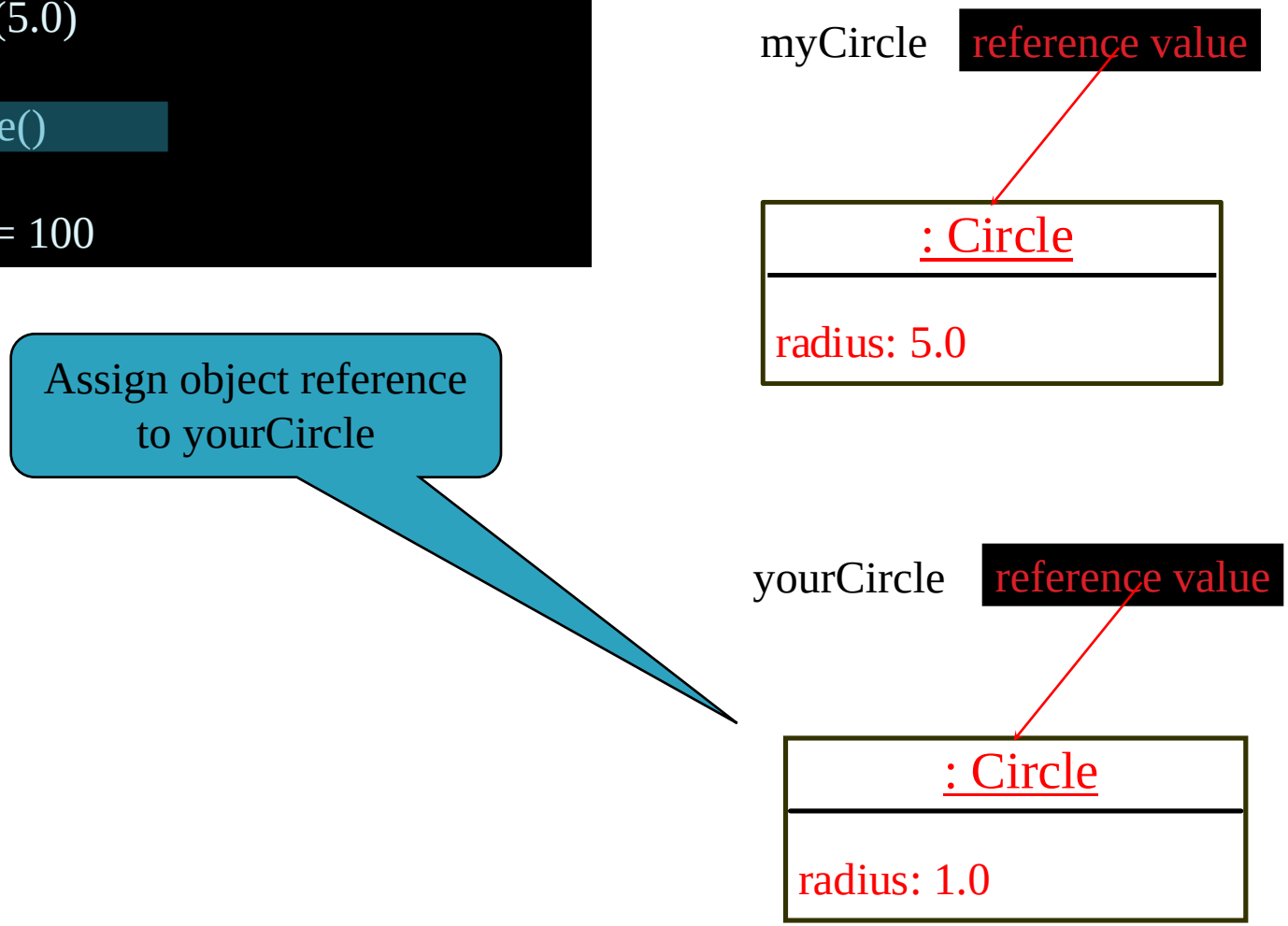


Trace Code

```
myCircle = Circle(5.0)

yourCircle = Circle()

yourCircle.radius = 100
```

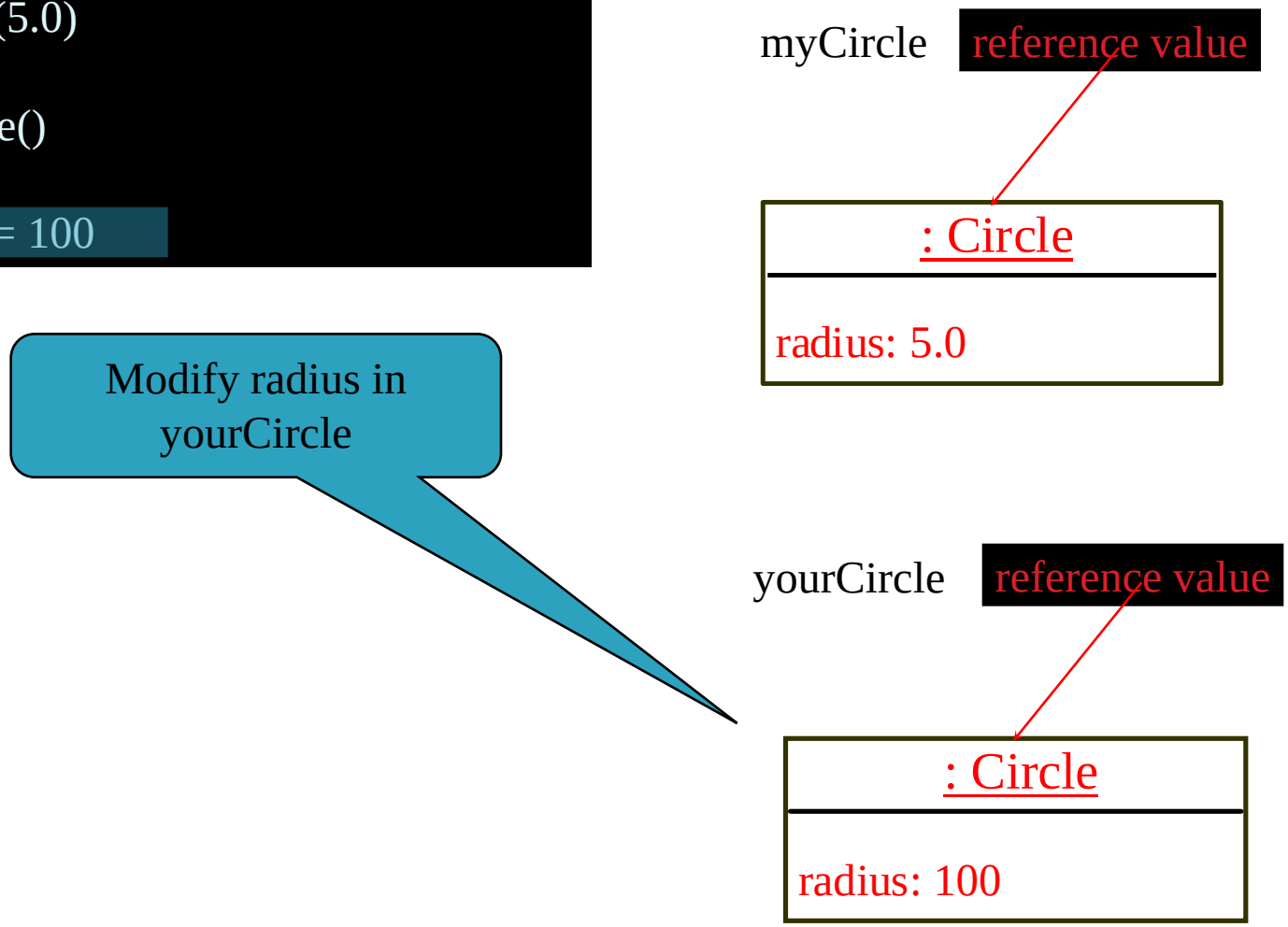


Trace Code

```
myCircle = Circle(5.0)

yourCircle = Circle()

yourCircle.radius = 100
```



Data Field Encapsulation

To protect data.
To make class easy to maintain.

To prevent direct modifications of data fields, don't let the client directly access data fields. This is known as *data field encapsulation*. This can be done by defining private data fields. In Python, the private data fields are defined with two leading underscores. You can also define a private method named with two leading underscores.

CircleWithPrivateDataRadius

Data Field Encapsulation

CircleWithPrivateDataRadius

```
>>> from CircleWithPrivateRadius import Circle
>>> c = Circle(5)
>>> c.__radius
AttributeError: 'Circle' object has no attribute
'__radius'
>>> c.getRadius()
 5
```

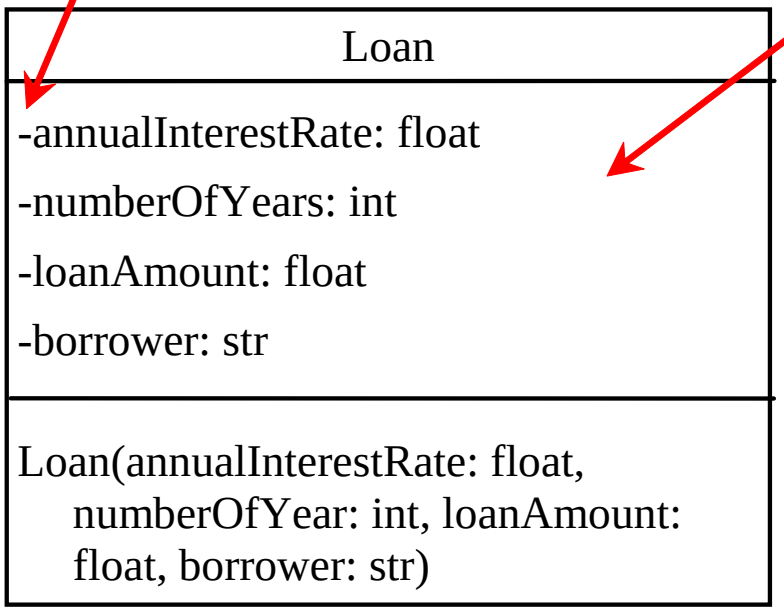
Design Guide

If a class is designed for other programs to use, to prevent data from being tampered with and to make the class easy to maintain, define data fields private. If a class is only used internally by your own program, there is no need to encapsulate the data fields.

Designing the Loan Class

The – sign denotes a private data field.

The get methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.



The annual interest rate of the loan (default: 2.5).

The number of years for the loan (default: 1)

The loan amount (default: 1000).

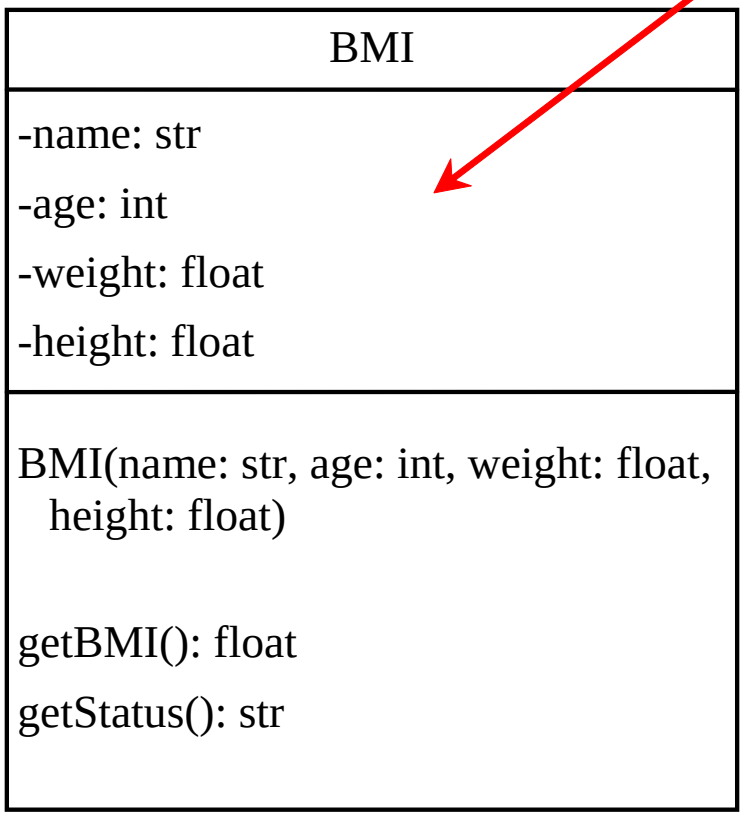
The borrower of this loan.

Constructs a Loan object with the specified annual interest rate, number of years, loan amount, and borrower.

Loan

TestLoanClass

The BMI Class



The get methods for these data fields are provided in the class, but omitted in the UML diagram for brevity.

The name of the person.

The age of the person.

The weight of the person in pounds.

The height of the person in inches.

Creates a BMI object with the specified name, weight, height, and a default age 20.

Returns the BMI

Returns the BMI status (e.g., normal, overweight, etc.)

BMI

UseBMIClass