Data Abstraction

The Client's View

- Suppose we want to make the Rational class from last lecture to memorize a simplified fraction; this can be implemented in various ways
- In order to maintain the same behavior from the client's point of view no matter the implementation, we use private methods, that are accessible only inside the class
- This ability to choose different implementations of the data without affecting clients is called data abstraction

Self-Reference

- On the inside of a class, the name ***this*** represents the object on which the current method is executed
- Note that the simple name ***m***, which refers to another member of the class is an abbreviation of ***this.m***

Preconditions

- Let's say our Rational class requires that the denominator is positive; we can enforce this by calling the ***require*** function:

  ```
  class Rational(x: Int, y: Int):
    require(y > 0, "denominator must be positive")
  ```
- ***require*** is a predefined function
- It takes a condition and an optional message string
- If the condition passed to require is false, an ***IllegalArgumentException*** is thrown with the given message
- Throwing an exception will usually terminate the program, but there are ways to handle it later

Assertions

- To check the code of a function, we can use ***assert***
- ***assert*** takes a condition and an optional message string as parameters
- A falling ***assert*** will throw an ***AssertionError*** exception

Constructors

- In Scala, a class implicitly introduces a constructor; this one is called the *primary constructor* of the class
- The primary constructor:
  - Takes the parameters of the class
  - Executes all statements in the class body (such as require)
- Scala also allows the declaration of *auxiliary constructors*; these are methods named ***this***

End Markers

- With longer lists of definitions and deep nesting, it’s sometimes hard to see where a class or other constructors ends
- End markers are a tool to make this explicit
- An end marker is followed by the name that’s defined in the definition that ends at this point
- It must align with the opening keyword
- End markers are also allowed for other constructs (methods, functions)
- If the end marker terminates a control expression (such as if) the beginning keyword is repeated (ex: end if)

Exercise: *Suppose that the **Rational** class is modified so that rational numbers are kept unsimplified internally, but the simplification is applied when numbers are converted to strings. Do clients observe the same behavior when interacting with the **Rational** class?*

– Yes, for small sizes of numerators and denominators and small numbers of operations. For bigger values it is possible that they will exceed Int if left unsimplified and so the results will be in some cases miscalculated.