UML. Files. Inheritance

Assoc. Prof. Arthur Molnar

Babeș-Bolyai University

2024

Overview

UML Diagrams

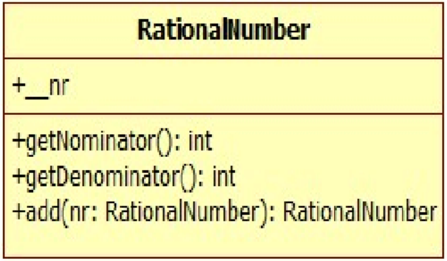
UML (Unified Modeling Language)

Standardized general-purpose modeling language in object-oriented software engineering.

- Includes a set of graphic notation techniques to create visual models of object-oriented software.
- It is language and platform agnostic (this is the whole point ☺)

Class Diagrams

UML Class diagrams - describe the structure of a system by showing the system's classes, their attributes, and the relationships between them.



Class Diagrams

```
class Rational:
    def __init__(self,a,b):
        '''
        Initialize a rational number
        a,b integers
        '''
        self.__nr = [a,b]
    def getDenominator(self):
        '''
        Denominator getter
        '''
        return self.__nr[1]
    def getNominator(self):
        '''
        Nominator getter
        '''
        return self.__nr[0]
```

Class Diagrams

In a class diagram the classes are represented using boxes which contain three parts:

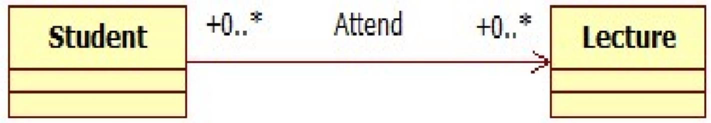
- Upper part holds the name of the class
- Middle part contains the class fields
- Bottom part contains the class methods

Relationships

- A relationship is a general term covering the specific types of logical connections found on class diagrams.
- A *Link* is the basic relationship among classes. It is represented as a line connecting two or more classes.

Associations

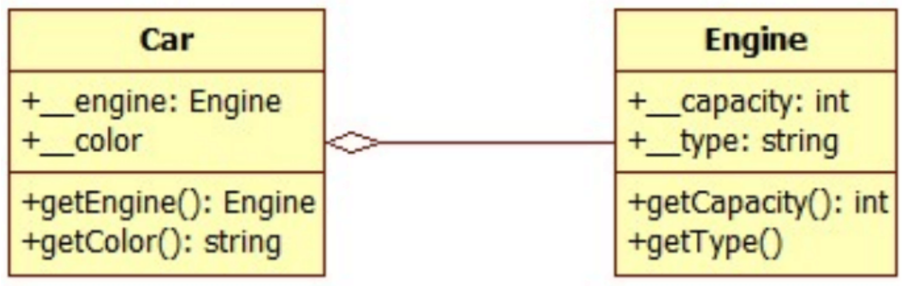
Binary associations (with two ends) are normally represented as a line, with each end connected to a class box.



An association can be named, and the ends of an association can be annotated with role names, ownership indicators, multiplicity, visibility, and other properties. Associations can be bi-directional as well as uni-directional.

Aggregation

Aggregation - an association that represents a part-whole or part-of relationship.



Aggregation

Aggregation - an association that represents a part-whole or part-of relationship.

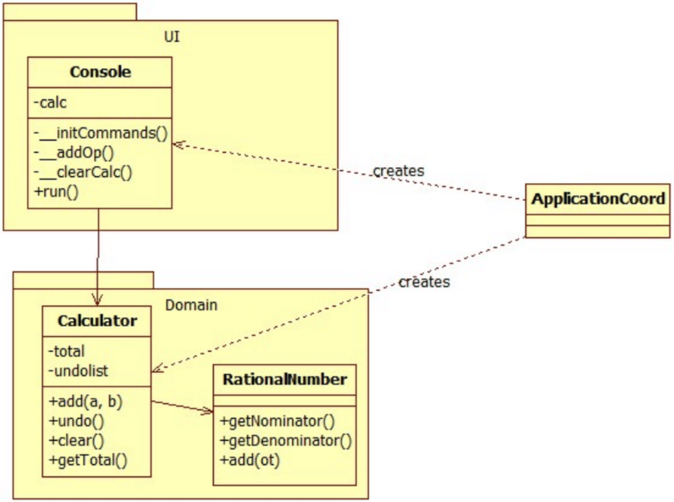
```python
class Car:
    def __init__(self, eng, col):
        '''
        Initialize a car
        eng - engine, col - string, i.e 'white'
        '''
        self.__eng = eng
        self.__color = col
class Engine:
    def __init__(self, cap, type):
        '''
        Initialize the engine
        cap - positive integer, type - string
        '''
        self.__capacity = cap
        self.__type = type
```

Dependency, Package

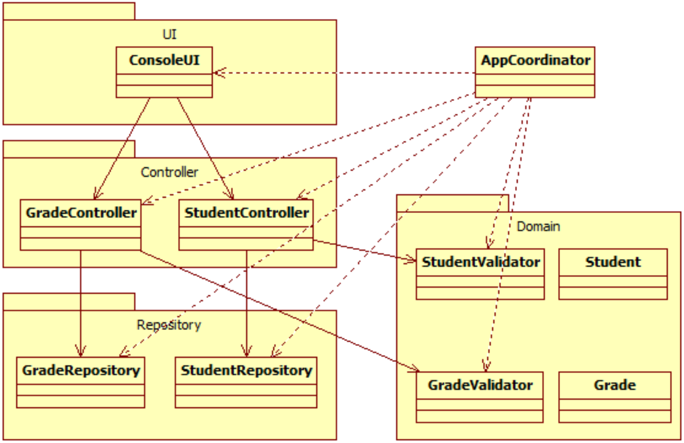
Dependency - a relationship in which one element, the client, uses or depends on another element, the supplier

- Create instances
- Have a method parameter
- Use an object in a method

Dependency, Package



UML class diagram for a Student Management app



Files

- The information on the computer is persisted using files.
- Files contain data, organized using certain rules (the file format).
- Files are organized in a hierarchical data structure over the file system, where directories (in most cases files, themselves) contain directories and files
- Operations for working with files: open (for read/write), close, read, write, seek.
- Files can be **text files** (directly human-readable) or **binary files**[1].

[1]Insert 10 types of people joke here ☺

Files

Possible problems when working with files

- Incorrect path/file given results in file not found error.
- File does not exist or the user running the program does not have access rights to it.
- File is already open by a different program (e.g. when you try to delete a file but the operating system disallows it)

Text files in Python

Common operations[2]

- Built in function: **open(filename,mode)** returns a file object.
- *filename* - string representing the path to the file (absolute or relative path)
- *mode*:
  - **"r"** - open for read
  - **"w"** - open for write (overwrites existing file content)
  - **"a"** - open for append
  - **"b"** - binary file (e.g. "**rb**" is read-mode, binary file)

[2] https://docs.python.org/3/tutorial/inputoutput.html#reading-and-writing-files

Text files in Python

Methods:

- *write(str)* - write the string to the file
- *readline()* - read a line from the file, return as a string
- *read()* - read the entire file, return as a string
- *close()* - close the file, free up any system resources

Text files in Python

Exception:

- **IOError** - raised exception if there is an input/output error.
- **FileNotFoundError** - a type of **IOError** (inheritance), when the file was not found.

Text files in Python

Demo

A simple example to get you started with reading and writing text files in Python. lecture.examples.ex37_text_files.py

Object serialization with Pickle

Pickle is a Python module for saving/loading objects from a binary file[3]

- *load(f)* - load the data from the file
- *dump(object,file)* - write the *object* to the given file in pickle's own format
- In order to use Pickle, you must f.open() using "**rb**" and "**wb**" (read binary and write binary, respectively)

[3] `https://docs.python.org/3/library/pickle.html#module-pickle`

Object serialization with Pickle

Demo

A simple example to get you started with Pickle.
lecture.examples.ex38_pickle_files.py

Inheritance

- Classes can inherit attributes and behavior (i.e., previously coded algorithms associated with a class) from pre-existing classes called **base classes** (or superclasses, or parent classes)
- The new classes are known as **derived classes** or **subclasses** or child classes. The relationships of classes through inheritance gives rise to a hierarchy.

NB!

Inheritance defines an **is a** relationhip between the derived and base classes.

Inheritance for code reuse

- We can reuse code that already exist in another class.
- We can replace one implementation with another, more specialized one.
- With inheritance, base class behaviour can be inherited by subclasses. It not only possible to call the overridden behaviour (method) of the ancestor (superclass) before adding other functionalities, one can override the behaviour of the ancestor completely.

Inheritance in Python

- Syntax: **DerivedClassName**(BaseClassName)[4]:
- DerivedClass will inherit:
  - Fields
  - Methods
- If a requested attribute (field,method) is not found in the class, the search proceeds to look in the base class
- Derived classes may override methods of their base classes.
- An overriding method in a derived class may in fact want to extend rather than simply replace the base class method of the same name.
- There is a simple way to call the base class method directly: call *BaseClassName.methodname(self,arguments)*

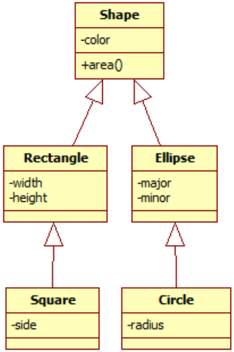
[4] https://docs.python.org/3/tutorial/classes.html#inheritance

Demo

Inheritance in Python

Examine the source code in lecture.examples.ex39_inheritance.py

Demo - UML class diagram



Inheritance

NB!

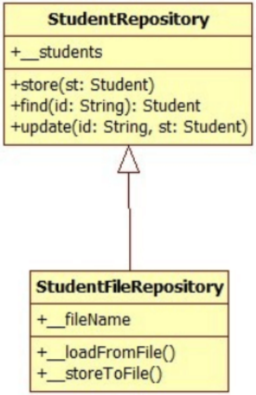
- The generalization relationship ("**is a**") indicates that one of the two related classes (the subclass) is considered to be a specialized form of the other.
- Any instance of the subtype is also an instance of the superclass.

Case Study I - File Repositories

- We would like to load/save problem entities to persistent storage.
- We already have a repository implementation, we're only missing the persistent storage functionality.
- We use **inheritance** to create a more specialized repository implementation, one that saves to/loads from files.

File Repositories

This is the UML class diagram for the repository implementation for the **Student** entity.



File Repositories

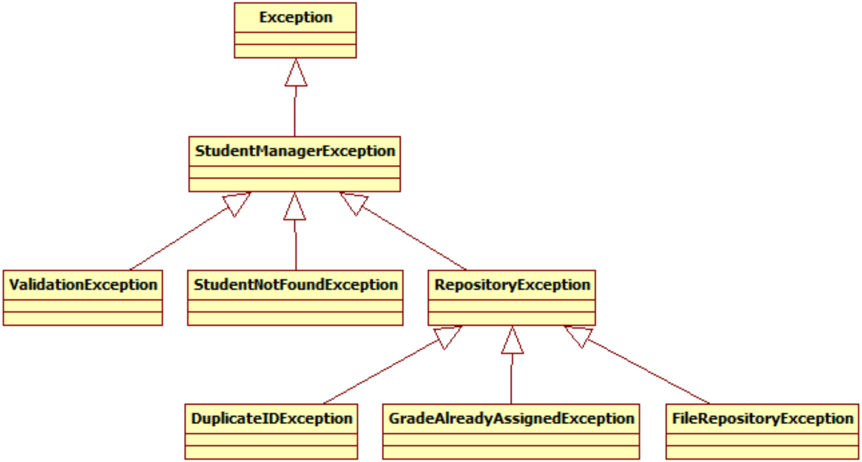
NB!

The application must work with either a *memory*, a *text file* or a *binary-file* backed repository implementation. Remember, modules are **independent** and **interchangeble**

Case Study II - Exception hierarchies

- We use exceptions to handle errors and special situations in the application
- Our exception classes are derived from **Exception**, a class that ships with the Python installation
- To handle different situations, most applications implement their own exception hierarchy

Example from a student management application



Exception hierarchies - example

What happens when we initialize the *Repository*?

- In memory implementation does not raise exceptions
- File-based implementation might raise *IOError* (input file not found, open another program, etc.)
- Database-backed implementation might raise *SQLConnectorException* (database server not started or cannot be reached)
- NoSQL database implementation might raise *CouchbaseException* (database server not started or cannot be reached)

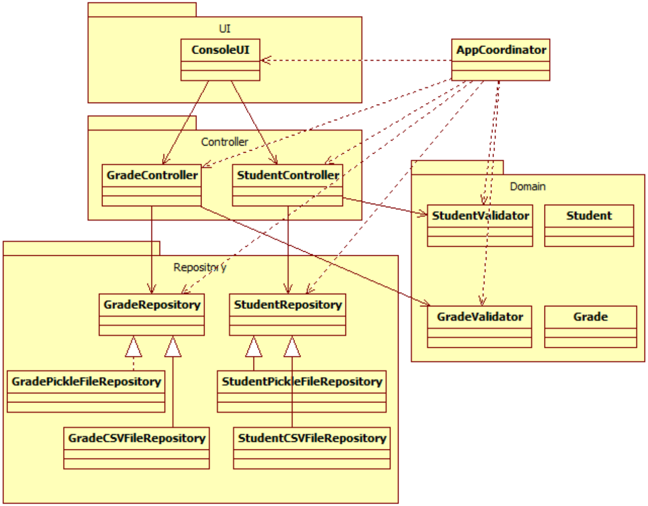
Exception hierarchies - example (cont.)

- Higher layers (*Controller*, *UI*) have to be independent from lower-layer implementations
- We cannot make the *Controller* or *UI* handle each possible exception type that a *Repository* might raise.

Solution

Define a *RepositoryException*. The repository code catches exception types that could be raised (e.g. *IOError*, *SQLConnectorException*) and re-raises them in the form of a *RepositoryException*

UML class diagram for student management app


UI
ConsoleUI
AppCoordinator
Controller
GradeController
StudentController
Domain
StudentValidator
Student
GradeValidator
Grade
Repository
GradeRepository
StudentRepository
GradePickleFileRepository
StudentPickleFileRepository
GradeCSVFileRepository
StudentCSVFileRepository