Layered architecture. Files. Inheritance. TkInter

Lect. PhD. Arthur Molnar

Babes-Bolyai University

arthur@cs.ubbcluj.ro

Overview

Layered architecture

Essential principle

Any element of a layer depends (calls code) only on other elements in the same layer or on elements of the layers beneath it.

- Partition a complex program into layers.
- Develop a design within each layer that is cohesive and that depends only on the layers below

Responsibilities

Presentation Layer

- User Interface - Responsible for showing information to the user, collect information from the user, and interpreting the user's commands

Responsibilities

Middle Layer

- Controller - The first object beyond the UI layer that receives and coordinates ("controls") a system operation.
- Entities - Responsible for representing concepts of the business
- Repositories - Provide methods to add and remove objects, will encapsulate the actual insertion or removal of data in the data store. Provide methods that select objects based on some criteria and return

Responsibilities

Infrastructure Layer

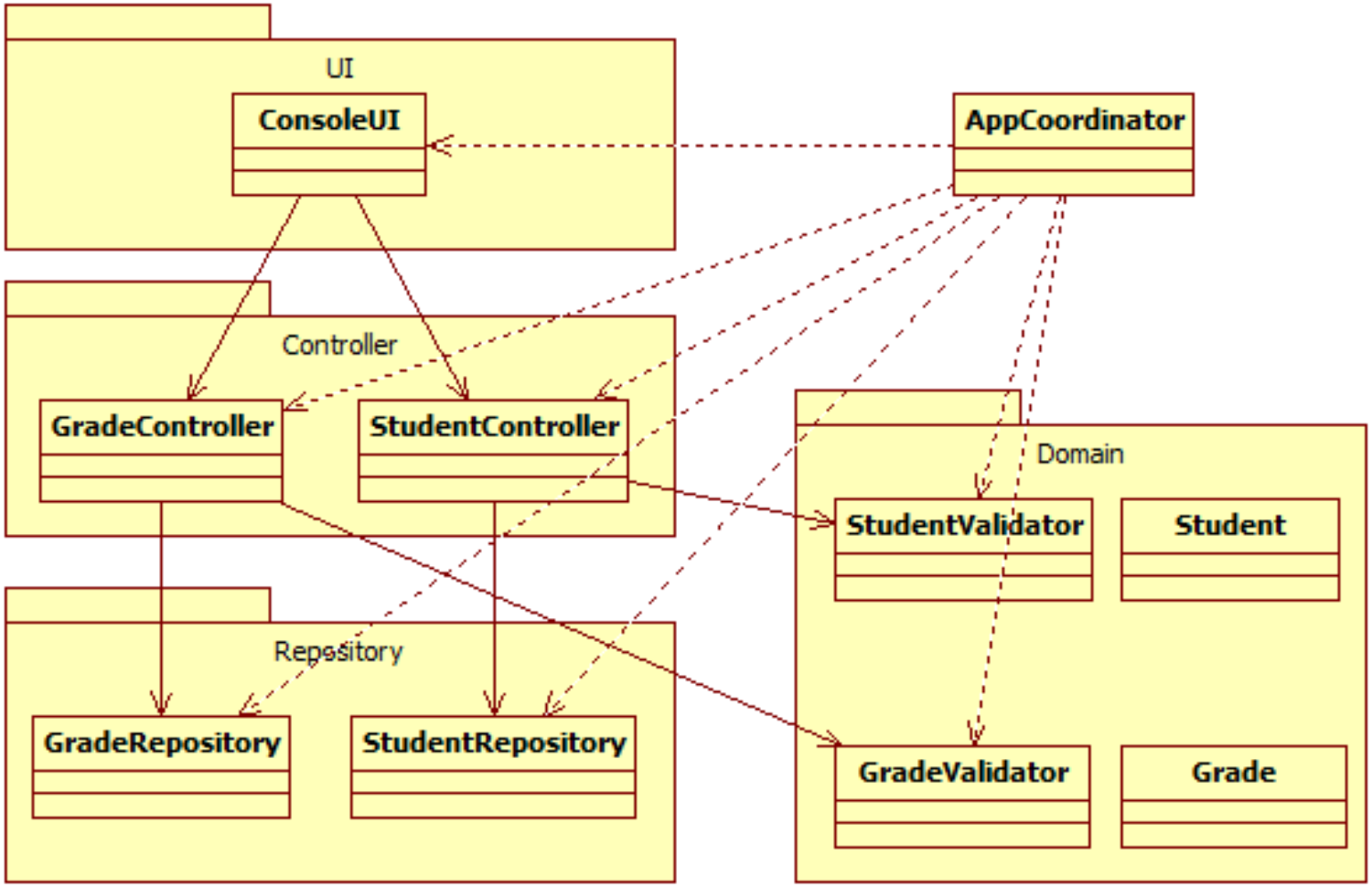
- Provides generic technical capabilities (utility, files, database, etc)

Today's example

Demo

Today's example will use the source code in **ex25_studentManagement.zip**

UML class diagram for Student Management app



Entities

- **Entity** - an object that is not defined by its attributes, but rather by a thread of continuity and its identity.
- If an object represents something with continuity and identity, it is something that is tracked through different states (or even across different implementations) it is an entity

Entities

- Entities are defined by identity and continuity
- Attributes of the entity may change but the identity remain the same
- Mistaken identity can lead to data corruption.
- Define what it means to be the same thing

Demo

Examine the source code of the **Student** class in the problem domain (**ex25_studentManagement.zip**).

Value Objects

- Many objects have no conceptual identity. These objects describe some characteristic of a thing
- An object that represents a descriptive aspect of the domain with no conceptual identity is called a VALUE OBJECT.
- When you care only about the attributes of an element of the model, classify it as a VALUE OBJECT

Value Objects

- The **Address** class in the Student Management app represents an example of Value Object
- When you specify the *county*, *city* and *street*, you already know everything about the object.
- **Other examples** of value objects: *Money*, *Location*, *Date*.
- **Generally** - any objects that represent the same thing when their attributes have the same value

Demo

Examine the source code of the **Address** class in the problem domain (**ex25_studentManagement.zip**).

Entities vs. Value Objects

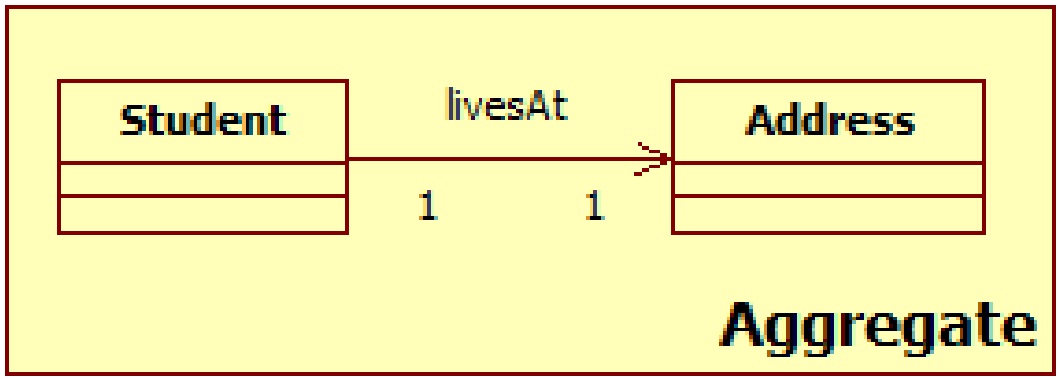
Discussion

- **Student** is an entity
- **Address** is a value object

Why isn't Student a value object?

Entities, Value objects and Repositories

- Cluster the **entities** and **value objects** into **aggregates** and define boundaries around them.
- Choose one **entity** to be the root of each aggregate, and control access to the objects inside the boundary using the root.
- Allow external objects to hold references to the root only.
- **e.g.** - only *StudentRepository*, **NOT** *AddressRepository*.



Data Transfer Objects

- **Data Transfer Objects (DTO)** are object used to carry data between processes.
- In the case where communication between processes is expensive (e.g. over the Internet), it makes sense to bundle up the data and send it in one go.
- DTO's have no behaviour, they only contain data, so should not require testing

NB!

Since our programs do not employ processes, we are not using DTO's exactly as intended. However, in real life you will find application layers on different machines/architectures.

Data Transfer Objects

Demo

Example **ex25_studentManagement.zip** uses the *StudentGrade* class as data transfer object when retrieving the Top5 students for a given discipline.

Files

- The information on your 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] There's 10 types of people, some of whom directly read hexa

Files

Possible problems when working with files

- Incorrect path/file given results in error.
- File does not exist or the user running the program does not have access to it.
- File already open

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 the existing 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.

Text files in Python

Demo

A simple example to get you started with reading and writing text files in Python. (**ex26_textFiles.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 is in (**ex27_pickleFiles.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

- One of the motivations for using inheritance is the reuse of code that already exist in another class (implementation inheritance).
- Before the object-oriented paradigm was in use, one had to write similar functionality over and over again.
- With inheritance, behaviour of a superclass 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

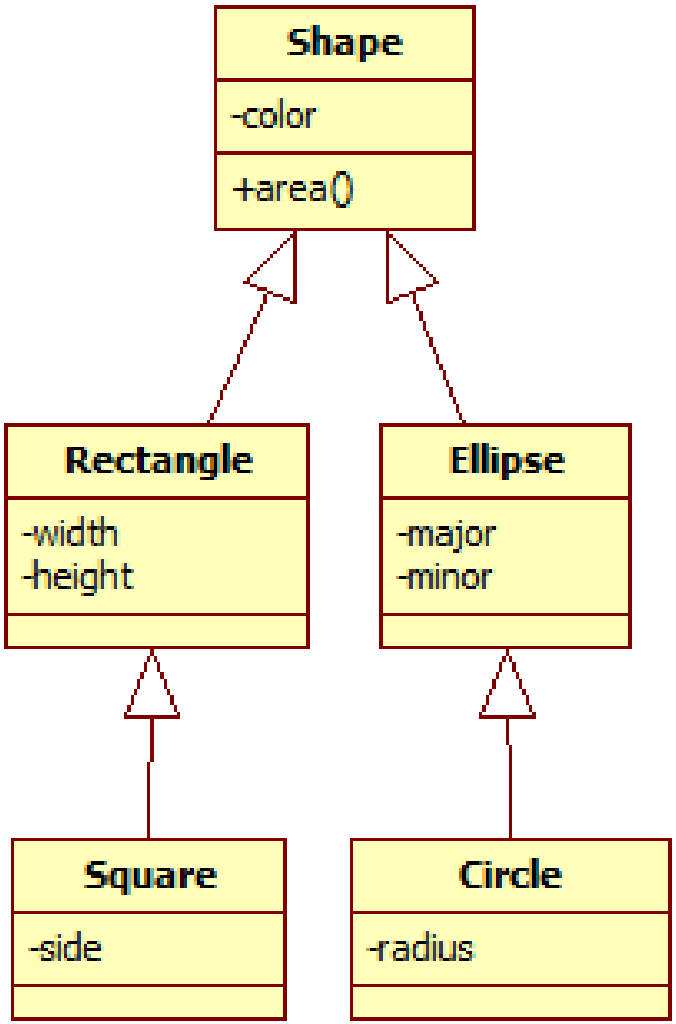
Examine the source code in **ex28_inheritance.py**

Inheritance - example

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.
- The generalization relationship is also known as the inheritance or "**is a**" relationship.

Inheritance - example

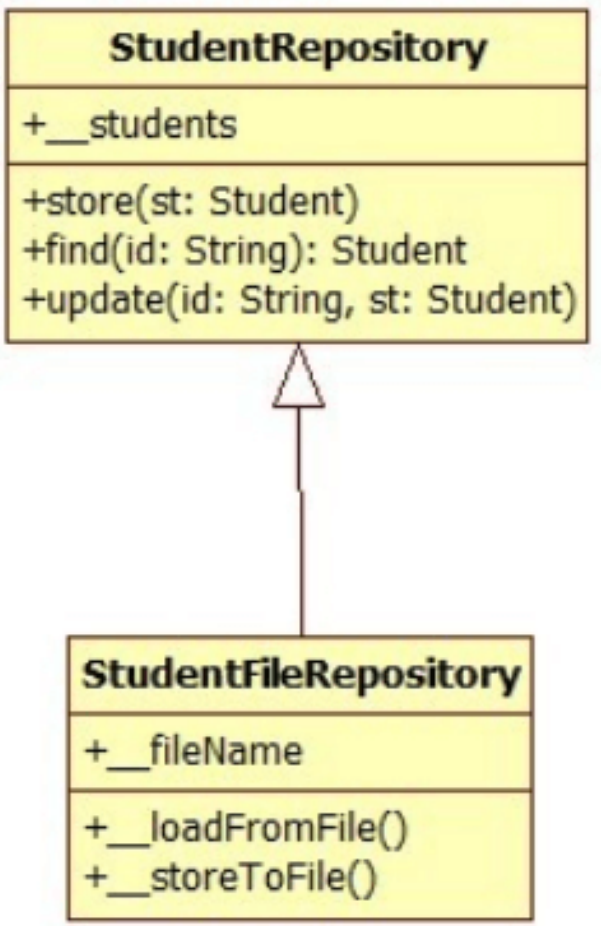


File Repositories

- 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

Demo

Examine the source code of the **StudentCSVFileRepository** and **GradeCSVFileRepository** classes in the problem domain (**ex25_studentManagement.zip**).

File Repositories

- The public part of *StudentRepository* and *GradeCSVFileRepository* are identical
- The public part of *GradeRepository* and *GradeCSVFileRepository* are identical

NB!

The application must work with either repository implementation. Remember, modules are **independent** and **interchangeble**

File Repositories

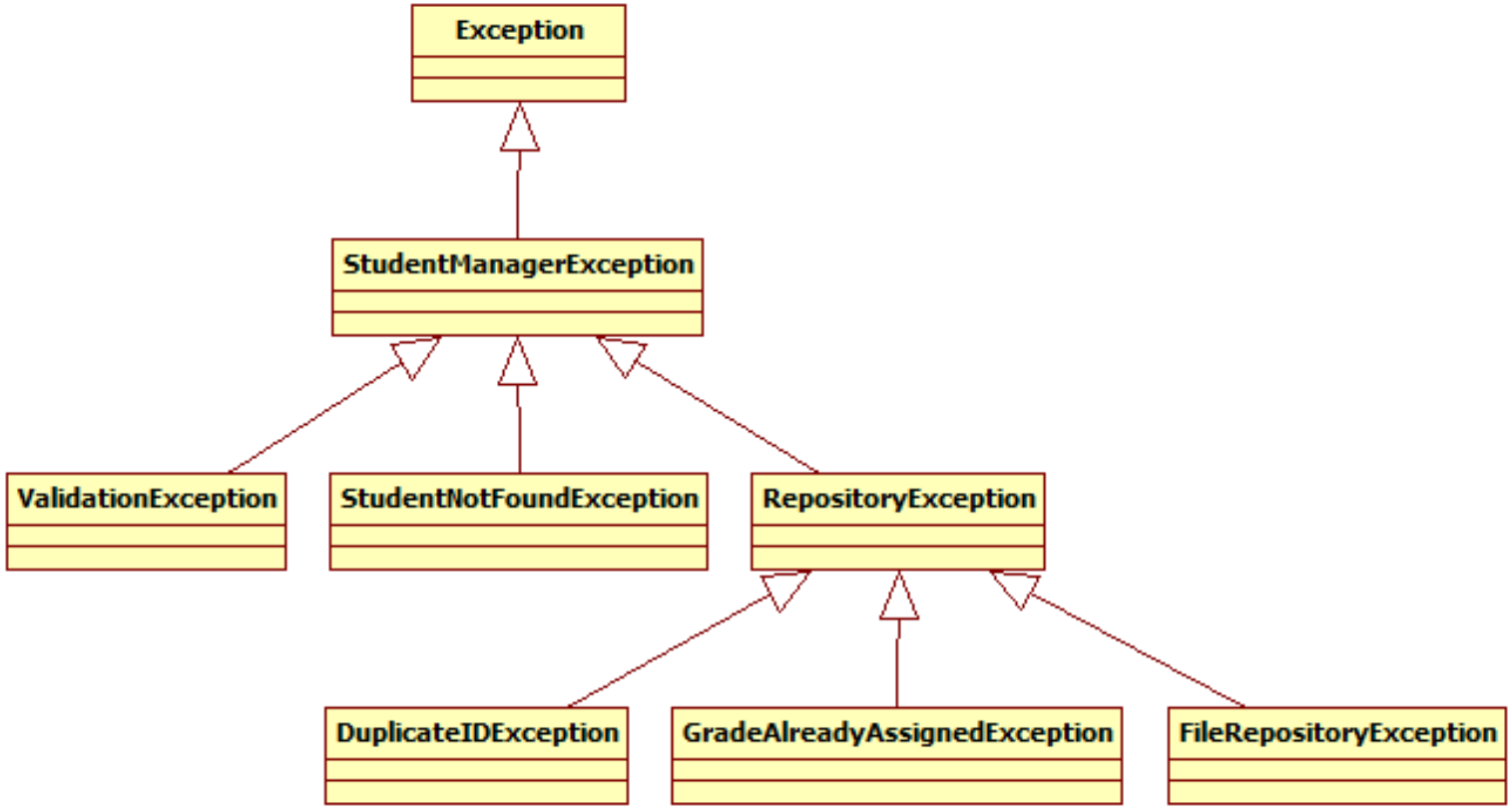
Demo

In the source code of **ex25_studentManagement.zip**, try to use the different repository implementations.

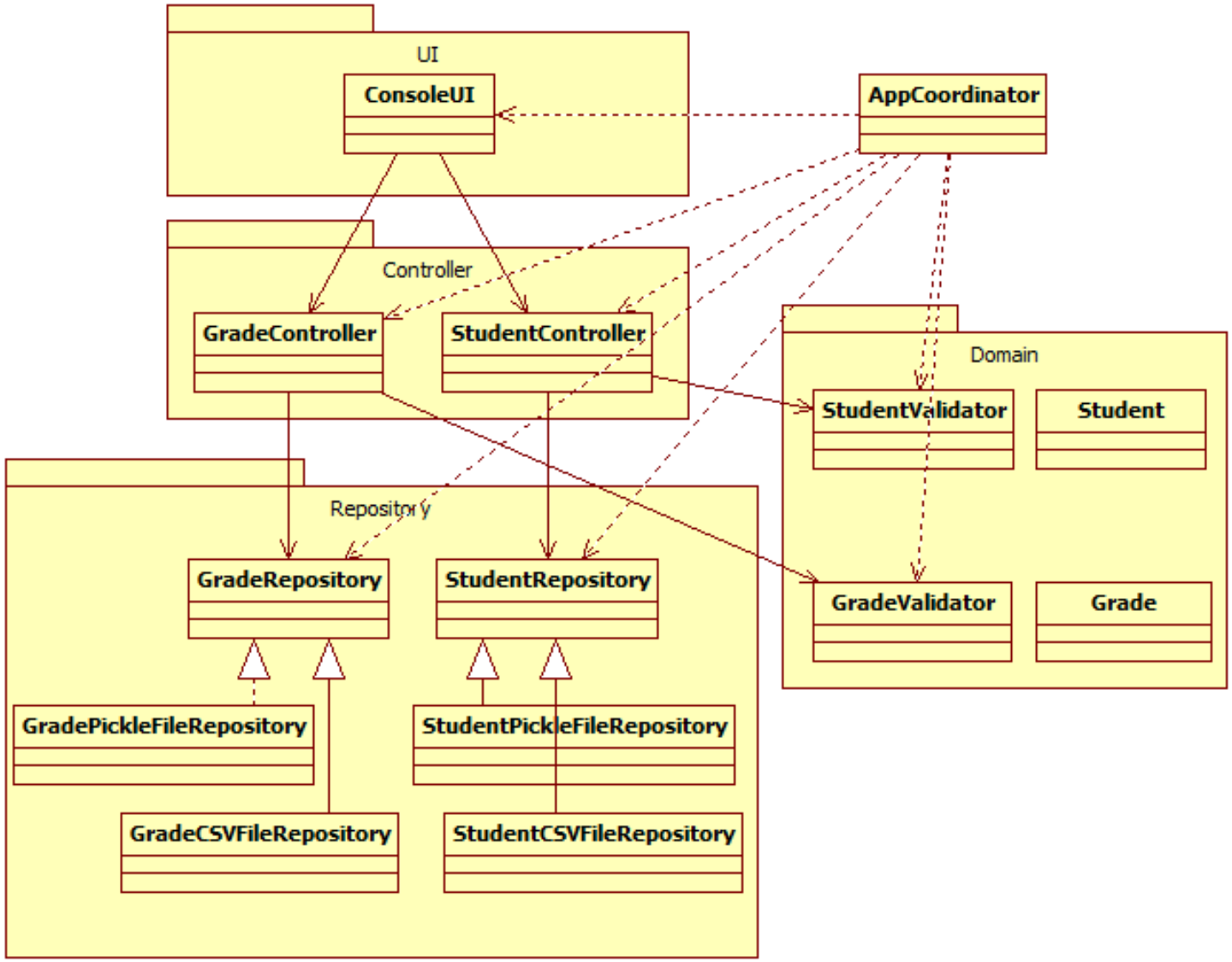
Exception hierarchies

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

Exceptions



Project Structure



About GUIs

- GUI applications are built using toolsets such as TkInter, AWT, Swing, SWT, WPF, JavaFX and many, many more
- What these libraries provide
  - Graphical components such as buttons, lists, tables, and so on (also called **widgets**).
  - Management of events (e.g. what happens when you click a button)

About GUIs

- To build your first GUI, you must essentially take three steps
  1. Build the window and fill it with widgets
  2. Tell the GUI library which events you want to handle and how (known. Basically when an event is encountered (e.g. a button is clicked) a function is called.
  3. Start the main event loop.

About GUIs

A few things to take in consideration

- The GUI code must be contained within the program's presentation layer
- Your program must work both with a GUI as well as using a console UI
- Switching betweem them must be (very) easy

Demo

Without further ado

Let's examine the code in source file **ex29_gui.zip**