Thinking about a solution in an object-oriented way

- Describe an algorithm using **things** and **actions**
  - **Things** represent abstract data types that will become **classes** in your OO code;
  - **Actions** represent behaviours that will become **methods** inside your classes;

- Do not describe your algorithm using low-level details and types
  - e.g. to get all plasmids of a fasta file, don't say:
    - Open the text file in read mode, read the first string and call it the header, read the second string and call it the sequence, and if the substring "plasmid" or "pld" is inside the header, build a class with the two strings and output it;
  - Be more general, use **things and actions**:
    - Open a Fasta File, and for each Fasta Record, check if it is a plasmid. If yes, then output it.

Solution

```
For each plasmid in the fasta file:
  For each gene in the plasmid:
    If at least one transcript can be expressed:
      Output that the gene can be expressed;
    Else:
      Output that the gene won’t be expressed;
```

Solution

```
Open the fasta file

plasmids = empty list
For each fasta record in the fasta file:
  If fasta record is a plasmid:
     Add the plasmid to plasmids

For each plasmid in plasmids:
  For each gene in the plasmid:
    If any of the possible 6 transcripts of the gene does not have a
premature stop codon:
      Output that the gene can be expressed;
    Else:
      Output that the gene won’t be expressed;
```

Classes

- A set of data (attributes) and functions (methods);

- Helps to modularise software;

- Objects are instances of classes;

- Attributes (data) must be private to objects (encapsulation);

- Methods define what can we do with an object;

Procedural approach

- Data types are just “containers” of data;
- You simply do whatever you want with them;

```
class Point:
  def __init__(self, x, y):
     self.x = x
     self.y = y

def main():
  point = Point(0, 0)
  point.x = 2
  point.y = -3
```

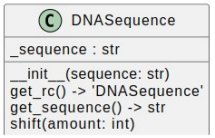
Object orientation

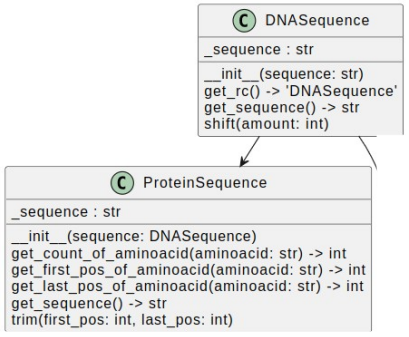
- Data types are living objects;
- They have their own data, UNACCESSIBLE to you (private attributes);
- They implement their own BEHAVIOURS (private methods);
- You don’t need to know HOW they do things, just WHAT they do;
- You TALK/ASK your objects – they can REJECT what you ask;
- You need to ABSTRACT what is important for a class in your code;
- A class Person can’t have all attributes and methods a person can do;

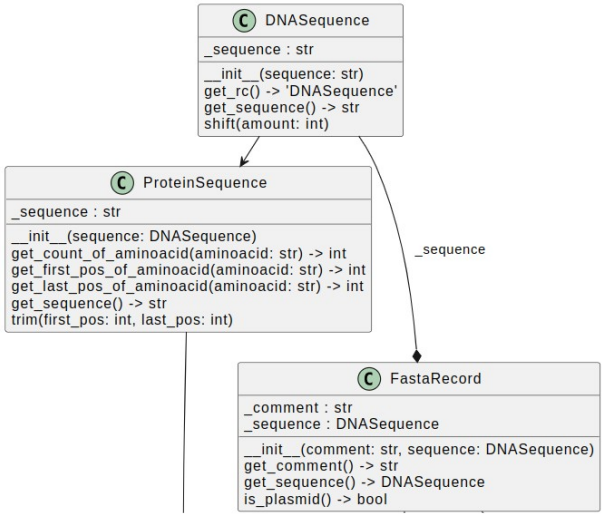
```
class Point:
  def __init__(self, x, y):
     <----------private-------->

  def move(self, new_x, new_y):
     <----------private-------->


def main():
  point = Point(0, 0)
  point.move(2, -3)
```

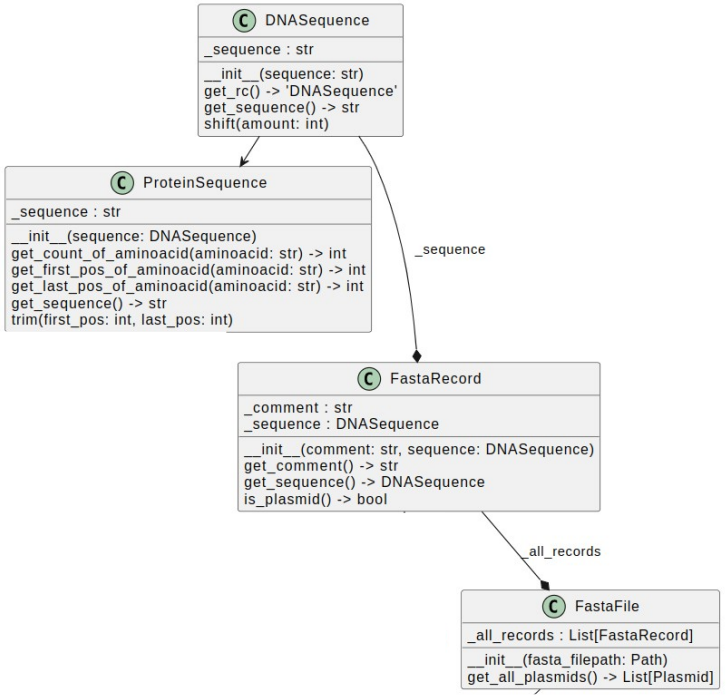
UML DIAGRAMS

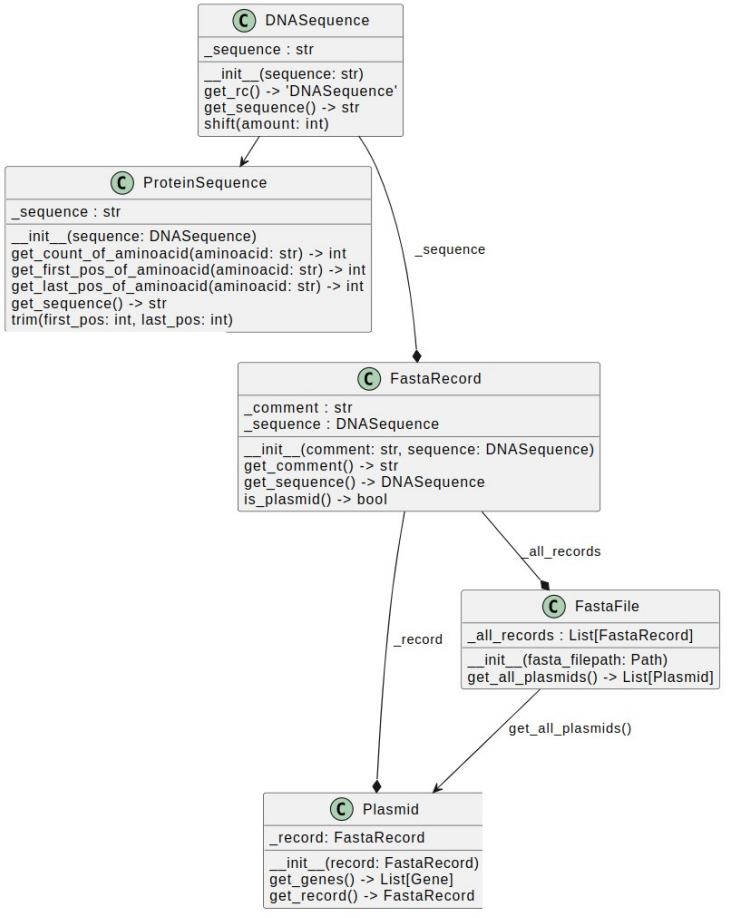
Represent set of classes and their relationship

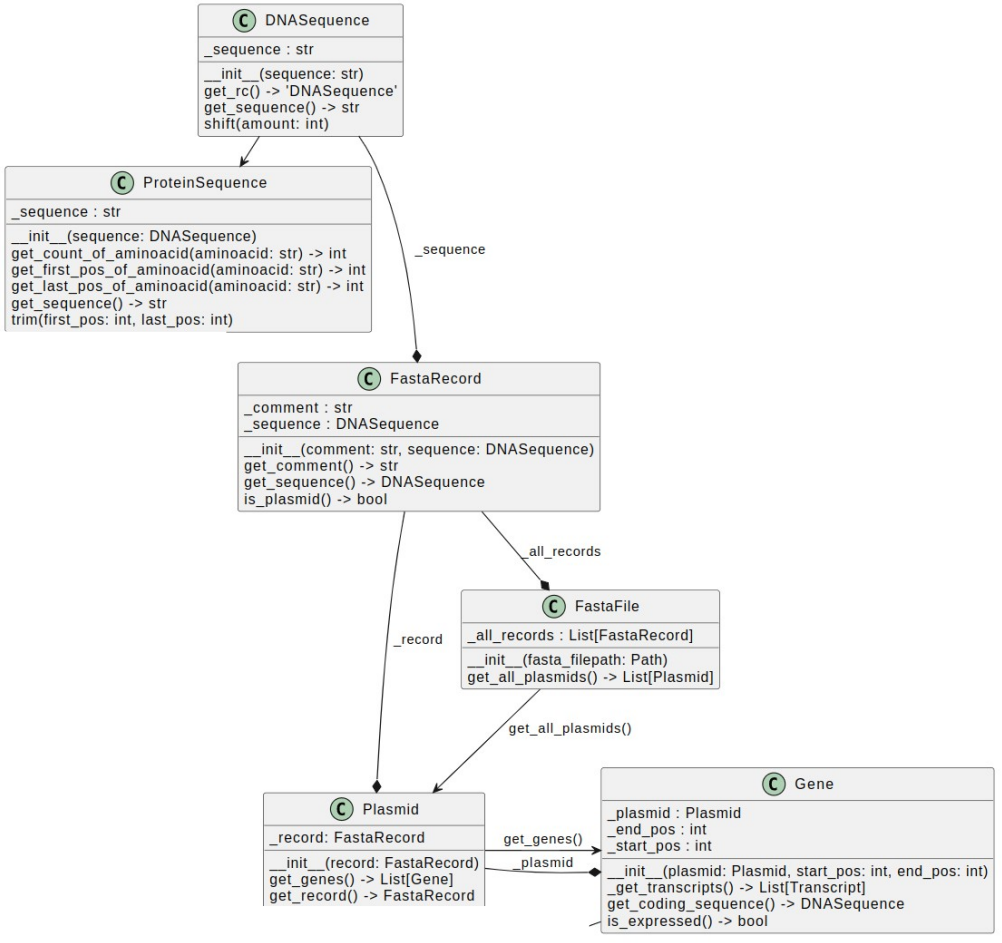

DNASequence
_sequence : str
__init__(sequence: str)
get_rc() -> 'DNASequence'
get_sequence() -> str
shift(amount: int)

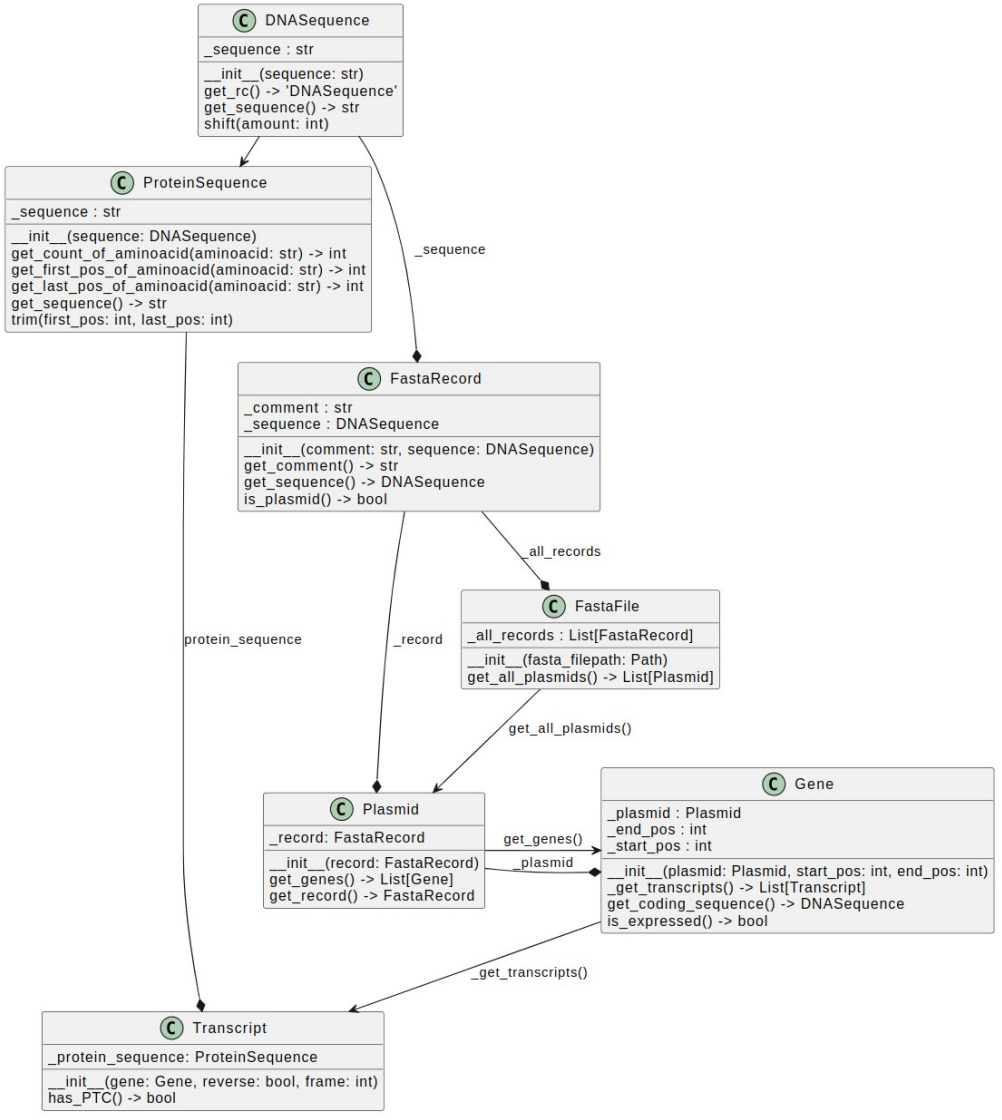

DNASequence
_sequence : str
__init__(sequence: str)
get_rc() -> 'DNASequence'
get_sequence() -> str
shift(amount: int)
ProteinSequence
_sequence : str
__init__(sequence: DNASequence)
get_count_of_aminoacid(aminoacid: str) -> int
get_first_pos_of_aminoacid(aminoacid: str) -> int
get_last_pos_of_aminoacid(aminoacid: str) -> int
get_sequence() -> str
trim(first_pos: int, last_pos: int)


DNASequence
_sequence : str
__init__(sequence: str)
get_rc() -> 'DNASequence'
get_sequence() -> str
shift(amount: int)
ProteinSequence
_sequence : str
__init__(sequence: DNASequence)
get_count_of_aminoacid(aminoacid: str) -> int
get_first_pos_of_aminoacid(aminoacid: str) -> int
get_last_pos_of_aminoacid(aminoacid: str) -> int
get_sequence() -> str
trim(first_pos: int, last_pos: int)
_sequence
FastaRecord
_comment : str
_sequence : DNASequence
__init__(comment: str, sequence: DNASequence)
get_comment() -> str
get_sequence() -> DNASequence
is_plasmid() -> bool


DNASequence
_sequence : str
__init__(sequence: str)
get_rc() -> 'DNASequence'
get_sequence() -> str
shift(amount: int)
ProteinSequence
_sequence : str
__init__(sequence: DNASequence)
get_count_of_aminoacid(aminoacid: str) -> int
get_first_pos_of_aminoacid(aminoacid: str) -> int
get_last_pos_of_aminoacid(aminoacid: str) -> int
get_sequence() -> str
trim(first_pos: int, last_pos: int)
_sequence
FastaRecord
_comment : str
_sequence : DNASequence
__init__(comment: str, sequence: DNASequence)
get_comment() -> str
get_sequence() -> DNASequence
is_plasmid() -> bool
_all_records
FastaFile
_all_records : List[FastaRecord]
__init__(fasta_filepath: Path)
get_all_plasmids() -> List[Plasmid]


DNASequence
_sequence : str
__init__(sequence: str)
get_rc() -> 'DNASequence'
get_sequence() -> str
shift(amount: int)
ProteinSequence
_sequence : str
__init__(sequence: DNASequence)
get_count_of_aminoacid(aminoacid: str) -> int
get_first_pos_of_aminoacid(aminoacid: str) -> int
get_last_pos_of_aminoacid(aminoacid: str) -> int
get_sequence() -> str
trim(first_pos: int, last_pos: int)
_sequence
FastaRecord
_comment : str
_sequence : DNASequence
__init__(comment: str, sequence: DNASequence)
get_comment() -> str
get_sequence() -> DNASequence
is_plasmid() -> bool
_all_records
FastaFile
_all_records : List[FastaRecord]
__init__(fasta_filepath: Path)
get_all_plasmids() -> List[Plasmid]
_record
get_all_plasmids()
Plasmid
_record: FastaRecord
__init__(record: FastaRecord)
get_genes() -> List[Gene]
get_record() -> FastaRecord


DNASequence
_sequence : str
__init__(sequence: str)
get_rc() -> 'DNASequence'
get_sequence() -> str
shift(amount: int)
ProteinSequence
_sequence : str
__init__(sequence: DNASequence)
get_count_of_aminoacid(aminoacid: str) -> int
get_first_pos_of_aminoacid(aminoacid: str) -> int
get_last_pos_of_aminoacid(aminoacid: str) -> int
get_sequence() -> str
trim(first_pos: int, last_pos: int)
_sequence
FastaRecord
_comment : str
_sequence : DNASequence
__init__(comment: str, sequence: DNASequence)
get_comment() -> str
get_sequence() -> DNASequence
is_plasmid() -> bool
_all_records
_record
FastaFile
_all_records : List[FastaRecord]
__init__(fasta_filepath: Path)
get_all_plasmids() -> List[Plasmid]
get_all_plasmids()
Plasmid
_record: FastaRecord
__init__(record: FastaRecord)
get_genes() -> List[Gene]
get_record() -> FastaRecord
get_genes()
_plasmid
Gene
_plasmid : Plasmid
_end_pos : int
_start_pos : int
__init__(plasmid: Plasmid, start_pos: int, end_pos: int)
_get_transcripts() -> List[Transcript]
get_coding_sequence() -> DNASequence
is_expressed() -> bool


DNASequence
_sequence : str
__init__(sequence: str)
get_rc() -> 'DNASequence'
get_sequence() -> str
shift(amount: int)
ProteinSequence
_sequence : str
__init__(sequence: DNASequence)
get_count_of_aminoacid(aminoacid: str) -> int
get_first_pos_of_aminoacid(aminoacid: str) -> int
get_last_pos_of_aminoacid(aminoacid: str) -> int
get_sequence() -> str
trim(first_pos: int, last_pos: int)
_sequence
FastaRecord
_comment : str
_sequence : DNASequence
__init__(comment: str, sequence: DNASequence)
get_comment() -> str
get_sequence() -> DNASequence
is_plasmid() -> bool
_all_records
FastaFile
_all_records : List[FastaRecord]
__init__(fasta_filepath: Path)
get_all_plasmids() -> List[Plasmid]
protein_sequence
_record
get_all_plasmids()
Plasmid
_record: FastaRecord
__init__(record: FastaRecord)
get_genes() -> List[Gene]
get_record() -> FastaRecord
get_genes()
_plasmid
Gene
_plasmid : Plasmid
_end_pos : int
_start_pos : int
__init__(plasmid: Plasmid, start_pos: int, end_pos: int)
_get_transcripts() -> List[Transcript]
get_coding_sequence() -> DNASequence
is_expressed() -> bool
_get_transcripts()
Transcript
_protein_sequence: ProteinSequence
__init__(gene: Gene, reverse: bool, frame: int)
has_PTC() -> bool