## 1.1 Library

- Declare a `Library` class to support the following functions.

  ```
  Output binom(final Derivative deriv, final MarketData mkt, int n)

  int impvol(final Derivative deriv, final MarketData mkt, int n,
             int max_iter, double tol, Output out)
  ```

- The `Library` class is a `final` class.

  1. The function `binom` implements the binomial model to compute the fair value and fugit of a derivative.
  2. The function `impvol` executes a loop of iterations to calculate the implied volatility of a derivative.

- Additional details, including the various input and output classes mentioned above, will be given below.

### 1.2 Class `MarketData`

- A `MarketData` object contains market data values to be used for valuation of a derivative.

```
final class MarketData
{
    public double Price;            // market price of security
    public double S;                // stock price
    public double r;                // interest rate
    public double sigma;            // volatility
    public double t0;               // current time
...
}
```

- The values of the `MarketData` class members are set in a calling application (for example the main() program), and the `MarketData` object is passed as an input to the above library functions.
- It is your responsibility to write any class methods, etc. that you consider appropriate.

### 1.3 Class `Output`

- An `Output` object contains the results of calculations by the library functions.

  ```
  final class Output
  {
      public double FV;       // fair value
      public double fugit;    // fugit
      public double impvol;   // implied volatility
      public int num_iter;    // number of iterations to compute implied volatility
  ...
  }
  ```

- The values of the `Output` class members are set in the library functions and returned to the calling application.

  1. Note that not all output fields may be populated by a given fumction.
  2. For example the function `binom` populates the values of `FV` and `fugit`, but not `impvol` and `num_iter`.

- It is your responsibility to write any class methods, etc. that you consider appropriate.

## 1.4 Class `Node`

- To calculate the fair value and fugit of a derivative, the binomial model should allocate `Node` objects.

  ```
  final class Node
  {
      ...
  }
  ```

- It is your responsibility to decide what data members and methods the `Node` class should contain.
- **It is your responsibility to decide how the binomial model allocates the `Node` objects, e.g. array or ArrayList or HashSet, etc.**
- In other words, *you formulate the software design.*

### 1.5 Class `Derivative`

- The class `Derivative` is an **abstract base class** which supports virtual functions for use in the valuation of derivatives.

  ```
  abstract class Derivative
  {
      public double T;                            // data member
      public void terminalCondition(Node n)       // virtual function
      public void valuationTest(Node n)           // virtual function
  }
  ```

- All the derivatives we shall treat in this course have an expiration time `T`.
- Hence the `Derivative` class contains a data member `double T`.
- The virtual functions must be overridden by non-abstract derived classes.
  1. The virtual function `terminalCondition` sets the payoff value and the fugit value on the expiration date.
  2. The virtual function `valuationTest` is called when traversing the tree in the binomial model, to make decisions about early exercise and to set the fair value and fugit to appropriate values.
  3. The `Node` object must therefore contain suitable data members and methods for the above functions to perform their tasks.

## 1.6 Class `VanillaOption`

- The class `VanillaOption` is a non-abstract class which extends `Derivative`, to support the valuation of ordinary (vanilla) options.
- It must support put and call options, also American and European exercise policies.
- Therefore the `VanillaOption` must contain suitable indicative data members (primary key).
- The `VanillaOption` class must also override the virtual functions.

```
class VanillaOption extends Derivative
{
    ...                                       // data members
    public void terminalCondition(Node n)     // override
    public void valuationTest(Node n)         // override
...
}
```

## 1.7 Class `BermudanOption`

- The class `BermudanOption` is a non-abstract class which is an American-style vanilla option, but it allows early exercise only in the time window $w_{\mathrm{begin}} \leq t \leq w_{\mathrm{end}}$.

```
class BermudanOption extends ...
{
    public double window_begin;
    public double window_end;
...
}
```

- A real Bermudan option can have many time windows, but we shall support only one time window.
- The `BermudanOption` class must contain suitable indicative data members and override the virtual functions in `Derivative`.

## 1.8 Classes

- The overall set of classes is therefore as follows.

```
class Library
class MarketData
class Output
class Node
abstract class Derivative & non-abstract classes
```

- It must be possible for me to declare a non-abstract class (new financial derivative) and override the virtual functions in `Derivative`.
- The functions in your library must be able to calculate the fair value, fugit and implied volatility of an object of that class I declare.
- *Polymorphism:* it must be possible for me to declare such a class and your program code must support it **without modifying any of your program code.**