Abstraction and Abstract Data Types

- **Abstraction:**
  - Whatever is visible to the user?
  - Examples: Variable names & real numbers.
    - How real numbers are implemented?
    - How arrays are implemented?
      - The abstraction is rectangular arrays
      - The implementation is one-dimensional array.

- Given the following algorithm:

  \[
  S = \text{set of persons.} \\
  X \in S; \\
  \text{smallest-age} = X; \\
  S = S - \{X\}; \\
  \text{While } S \text{ is not empty do} \\
  \quad Y \in S; \\
  \quad S = S - \{Y\}; \\
  \quad \text{If } \text{age}(Y) < \text{age}(X) \\
  \quad \quad \text{Then smallest} = Y; \\
  \text{Endwhile;}
  \]

  - What do you need to implement such a problem?
For a given implementation,

- we need to make the following assumptions:

1. The data we are dealing with are of some **data types**.
2. A construct to store the data types: **Data Structures**
3. The **set of operations** on the data types defined in(1).

- (1) & (3) ===> Abstract Data Types (**ADT**).
- (2) ===> A way to implement the ADT.

**Data Types:** (1)
- Definition: The data type of a variable determines the set of values the variable can take.

- Simple data types:
  - Integer, real, character, enumeration types, etc.

- Structured data types:
  - arrays, records, files, and sets.

**Data Structures:** (2)
- Definition:
  - A data structure (D. S.) is a construct that you can define within a programming language to store a collection of data types.

**Examples:**
- All structured data types are D.S.
- Trees, Linked lists, etc.
• **Abstract Data Types (ADT):** (3)
  - **Definition:**
    - An abstract data type (ADT) is characterized by the following properties:
      - It exports a type, called domain.
      - It exports a set of operations. This set is called **interface**.
      - Operations of the interface are the one and only access mechanism to the type's data structure.
      - Axioms and preconditions define the application domain of the type.
    
    - The first property allows the creation of more than one instance of an ADT object.
    - The second property defines the only possible operations on the ADT object.
    - The third property prevents using other operations different from the ones defined in the interface.
    - Finally, the application domain is defined by the semantical meaning of provided operations. Axioms and preconditions include statements such as:
      - The denominator of a fraction is different from zero.
      - An empty list is a list.'
- **OOP and ADT:**
  - A class is an actual representation of an ADT.
  - It provides implementation details for the data structure used and operations.
  - OOP supports the implementation of an ADT using information hiding.
  - Information hiding
    - is used to hide the implementation details
    - Prevent users from directly accessing data members (\texttt{private} access modifier).
• Example:

  o ADT: Fraction
    ▪ Data type:
      - numerator: integer;
      - denominator: integer;
    ▪ Data structure: data types
    ▪ Operations:
      - Add, multiply, equal, reduce, divide, etc.
    ▪ Axioms and preconditions:
      - The denominator should be different from zero.

  o Java Implementation:

```java
//Fraction ADT
public class Fraction {

  private int numerator;
  private int denominator;

  public Fraction(){
    numerator = 0;
    denominator = 1;
  }

  public Fraction(int num, int deno){
    numerator = num;
    denominator = deno;
    if (denominator == 0){
      throw new IllegalArgumentException("Denominator cannot be zero");
    }
  }

  public void reduceF(){
    int n = numerator;
    int d = denominator;

    while (d != 0) {
      int t = d;
      d = n % d;
      n = t;
    }
    numerator /= n;
    denominator /= n;
  }

};
```
public int getNumerator() {
    return this.numerator;
}

public int getDenominator() {
    return this.denominator;
}

public Fraction add(Fraction f1, Fraction f2){
    Fraction f = new Fraction();
    f.numerator = f1.numerator * f2.denominator + f1.denominator * f2.numerator;
    f.denominator = f1.denominator * f2.denominator;
    f.reduceF();
    return (f);
}

public Fraction sub(Fraction f1, Fraction f2){
    //Implement this function. 
}

public Fraction multiply(Fraction f1, Fraction f2){
    //Implement this function. 
}

public Fraction divide (Fraction f1, Fraction f2){
    //Implement this function. 
}

public boolean equals(Fraction f2){
    return getNumerator()*f2.getDenominator() == f2.getNumerator()*getDenominator();
}

public void printF(Fraction f){
    System.out.println (f.numerator + "/" + f.denominator);
}

o Programming Assignment:
  - Design and implement the missing operations in the Fraction ADT:
    - Subtraction:
      \[
      \frac{a}{b} + \frac{c}{d} = \frac{ad + bc}{bd}
      \]
    - Multiplication:
      \[
      \frac{a}{b} \times \frac{c}{d} = \frac{ac}{bd}
      \]
    - Division
      \[
      \frac{a}{b} \div \frac{c}{d} = \frac{ad}{bc}
      \]
  - Test your implementation.